O'REILLY®

alist

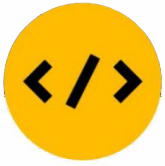
Базовая математика для искусственного интеллекта

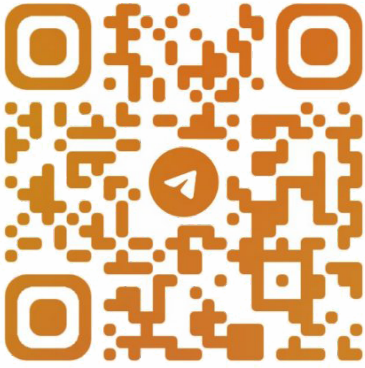
Математика нового уровня
для эффективных
и успешных систем ИИ

Хала Нельсон

Essential Math for AI

Next-Level Mathematics for Efficient and Successful AI Systems





@CODELIBRARY_IT

Hala Nelson

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY

Хала Нельсон

Базовая математика для искусственного интеллекта

Математика нового уровня
для эффективных
и успешных систем ИИ

Астана
«АЛИСТ»
2024

УДК 51
ББК 22.1
Н49

Нельсон Х.

Н49 Базовая математика для искусственного интеллекта: Пер. с англ. — Астана: АЛИСТ, 2024. — 592 с.: ил.

ISBN 978-601-09-7540-8

Книга дает прочные знания математики, лежащей в основе работы современных систем ИИ. Приведены необходимые и достаточные сведения для успешной работы в области ИИ, без углубления в сложные академические теории, с акцентом на практическом применении и современных моделях. Даны основы машинного обучения и науки о данных. Рассмотрены регрессия, нейронные сети, свертка, оптимизация, вероятность, марковские процессы, дифференциальные уравнения и многое другое исключительно в контексте искусственного интеллекта. Показано, как объединять модели машинного обучения и естественного языка, работать с графовыми и сетевыми данными, визуализировать преобразования пространства, уменьшать размерность, обрабатывать изображения, выбирать модели и для проектов, основанных на данных.

Для специалистов в области ИИ, машинного обучения и науки о данных

УДК 51
ББК 22.1

© 2024 ALIST LLP

Authorized Russian translation of the English edition of ***Essential Math for AI***, (ISBN 9781098107635) © 2023 Hala Nelson. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Авторизованный перевод с английского языка на русский издания ***Essential Math for AI***, (ISBN 9781098107635) © 2023 Hala Nelson. Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc.

ISBN 978-1-098-10763-5 (англ.)
ISBN 978-601-09-7540-8 (каз.)

© Hala Nelson, 2023
© Издание на русском языке. ТОО "АЛИСТ", 2024

Содержание

https://t.me/it_boooks/2

Отзывы о книге "Базовая математика для искусственного интеллекта"

Рынки технологий и искусственного интеллекта подобны реке, где отдельные участки движутся быстрее других. Для успешного применения искусственного интеллекта требуется умение оценить направление течения и закрепить это прочным фундаментом, чему способствует книга, причем в увлекательной, интересной, всеохватывающей форме. Хала сделала математику увлекательной для широкого круга участников будущего под эгидой искусственного интеллекта!

— Адри Пуркаястха,
руководитель группы по аналитике операционных рисков
в сфере искусственного интеллекта и цифровых рисков,
банк BNP Paribas

Книги по искусственному интеллекту обычно представляют собой либо рукописи на техническом языке, написанные специалистами для других таких же специалистов, либо не содержащие математических сведений общие вводные положения, рассчитанные на широкую аудиторию. Эта книга идет по третьему пути и знакомит с математическими основами читателей, занятых в сфере бизнеса, работающих с данными и в других подобных областях и не имеющих высшего математического образования. Автор вкрапляет в повествование изящные уравнения и остроумные замечания, предлагая читателю задуматься о серьезных последствиях искусственного интеллекта для общества.

Я рекомендую книгу "Базовая математика для искусственного интеллекта" всем, кому будет интересно строгое изложение основ искусственного интеллекта сквозь призму практического подхода.

— Джордж Маунт,
аналитик данных, преподаватель

Хала проделала огромную работу в изложении важнейших математических понятий. Книга обязательна к прочтению для каждого серьезного специалиста по машинному обучению. После знакомства с книгой вы полюбите эту область еще больше.

— Уманг Шарма,
старший специалист по изучению данных, автор

Для того чтобы понять, что такое искусственный интеллект, необходимо уяснить взаимосвязь между математикой и искусственным интеллектом. Доктор Нельсон облегчила эту задачу, предоставив основу, на которой строятся симбиотические отношения между этими двумя дисциплинами.

— Хуан Нгуен,
контр-адмирал (в отставке), инженер-кибернетик,
командование морских систем ВМС (NAVSEA)

Введение

Почему я написала эту книгу

Искусственный интеллект базируется на математических моделях. Необходимо разобраться в том, как это устроено.

Книга написана легким и доступным языком, опущено множество технических подробностей. В ней говорится об искусственном интеллекте, дается минимум математических формул и уравнений, нет доказательств теорем, нет кодирования. Ее цель — *обеспечить читателю доступ к важнейшим знаниям, чтобы они не оставались лишь в элитарном кругу посвященных, и привлечь как можно больше внимания к техническим областям*. На мой взгляд, многие люди разочаровываются в математике слишком рано, так и не раскрыв ее чары и свои склонности к ней. Такое бывает не только в период студенчества, но и во время учебы в аспирантуре, где многие меняют математику на любую другую дисциплину или, приступив к кандидатской, так и не доводят ее до защиты. И здесь правильнее говорить не об отсутствии способностей, а об отсутствии мотивации, которая позволила бы увидеть конечную цель за мучительным заучиванием нудных методов и способов, от которых в жизни, на первый взгляд, нет никакого толка. Это все равно что каждый день усердно заниматься в интеллектуальном тренажерном зале только ради самих занятий. Ведь даже в обычный спортзал не принято ходить каждый день (это необъективное утверждение, но смысл понятен). В математике формализация объектов в функции, пространства, мерные пространства и целые математические области происходит *после* мотивации, а не *до* нее. К сожалению, процесс обучения выстроен в обратной последовательности: сначала идет процедура преобразования, или формализации, и только потом, если повезет, мотивация.

Самое прекрасное в математике — ее выразительная способность связывать воедино, на первый взгляд, несопоставимые вещи. Такая большая и значимая область, как искусственный интеллект, не только опирается на математику, что естественно, но также нуждается в этой связующей способности (которую дает только математика) для лаконичного изложения своей обширной тематики. В книге базовая математика для искусственного интеллекта рассматривается так, чтобы она не шла вразрез с его практическим применением. Очень трудно изучать подробно существующие инструменты и не впасть при этом в энциклопедичность и чрезмерно сложную подачу. Поэтому моя задача — сделать так, чтобы вы *понимали эти инструменты* и *смотрели на них* только как на средство достижения цели, которое в случае необходимости можно подстроить и откорректировать. Из книги вы узнаете,

как все связано друг с другом и почему мы разрабатываем или используем среди прочих те или иные методы. В некотором смысле можно считать ее платформой, с которой можно перейти к той области, которая покажется наиболее интересной, или в которой вы захотите специализироваться.

Еще одна цель книги — демократизация математики и повышение доверия к вопросам о том, как все устроено. Такие распространенные ответы, как "это сложная математика", "это сложная технология" или "это сложные модели", уже не удовлетворяют, тем более что технологии, основанные на математических моделях, сегодня влияют на все аспекты нашей жизни. Не нужно быть экспертами во всех областях математики (таковых не существует), чтобы понять, как устроены вещи и почему они работают так, а не иначе. Есть одна особенность математических моделей, которую необходимо знать каждому — они *всегда* выдают ответ и всегда выдают число. Проверенная, подтвержденная, подкрепленная надежной теорией модель выдает решение. Так же, как и *совершенно абстрактная* модель. Обе модели вычисляют математические функции. Когда мы говорим, что наши решения основаны на математических моделях и алгоритмах, это не делает их сакральными. На чем построены эти модели? Каковы их предположения? Ограничения? На каких данных они обучались? На каких данных тестировались? Какие переменные они учитывали?

А что они упустили? Обладают ли они обратной связью, ведущей к модернизации, или базовыми истинами, с которыми можно сравнивать и совершенствоваться? Существует ли теория, подтверждающая их? Необходимо соблюдать *прозрачность* информации при разработке собственной модели и настаивать на этом принципе в случаях, когда модели решают за нас вопросы жизнеобеспечения.

Темы в книге скомпонованы нестандартно. Это сделано намеренно, чтобы не увязнуть в математических деталях, прежде чем мы перейдем к практическим вопросам. На мой взгляд, не стоит углубляться в справочный материал, пока мы не перейдем к практике, где без информации, восполняющей пробел в наших знаниях, мы не сможем двинуться вперед. Только в таких случаях не жалко уделить много времени на изучение подробностей. Но что гораздо важнее — это понять всеобщую взаимосвязанность и согласованность. Другими словами, книга дает наглядное представление о том, как все, что связано с математикой и искусственным интеллектом, прекрасно взаимодействует друг с другом.

Новичкам следует обратить внимание на то, что мы живем в эпоху огромных массивов данных. До работы с большими данными, реальными или имитационными (смоделированными), структурированными или неструктурированными, компьютеры и Интернет воспринимались как должное. Когда мы создавали модель или нам требовалось провести аналитику на небольших и ограниченных по объему наборах данных, можно было предположить, что аппаратное обеспечение нашего компьютера справится с вычислениями, или что в случае необходимости можно найти нужный объем данных либо более подробную информацию об аналогичных моделях в Интернете. Однако сама реальность и ее ограничения доступа к данным, ошибки в данных, ошибки в результатах запросов, аппаратные ограничения, хранение, передача данных между устройствами, векторизация неструктурированных данных, таких как естественный язык или изображения и фильмы, привносят в на-

шу работу существенные корректировки. И тогда мы обращаемся к параллельным и облачным вычислениям, управлению данными, базам данных, структурам данных, архитектурам данных, инженерии данных с целью определить вычислительную инфраструктуру, позволяющую запустить наши модели. Какая инфраструктура у нас имеется? Какова ее структура? Как она развивалась? Куда она движется? Что представляет собой архитектура, в том числе задействованные в ней твердые материалы? Как эти материалы работают? А что за шумиха по поводу квантовых вычислений? Программное обеспечение нельзя рассматривать отдельно от аппаратных средств, а модели — отдельно от инфраструктуры, позволяющей их моделировать. В книге повествуется только о математике, моделях искусственного интеллекта и некоторых видах данных. Здесь нет упражнений или кодирования. Другими словами, мы сосредоточены на *"программной"*, т. е. *интеллектуальной*, "бестелесной", а не аппаратной составляющей. Но мы должны продолжать учиться до тех пор, пока не сможем осознать технологию, обеспечивающую многие аспекты нашей жизни, как единый взаимосвязанный организм, состоящий из аппаратного и программного обеспечения, датчиков и измерительных приборов, хранилищ данных, соединительных кабелей, беспроводных узлов, спутников, центров связи, физических и программных средств защиты, математических моделей.

Кому предназначена книга

Прежде всего, книга предназначена:

- математикам, желающим заняться искусственным интеллектом, машинным обучением, наукой о данных;
- специалистам в области искусственного интеллекта, науки о данных, машинного обучения, желающим освежить свои знания математики и ознакомиться с математическими идеями, лежащими в основе самых современных моделей;
- студентам и аспирантам, изучающим математику, науку о данных, информатику, исследование операций, естественные науки, инженерное дело или другие области, связанные с искусственным интеллектом;
- руководителям, желающим внедрить искусственный интеллект и аналитику данных в свою деятельность и вместе с тем глубже понять, как в действительности работают модели, на основе которых им, возможно, предстоит принятие решений;
- аналитикам данных из области бизнес-аналитики, которые сегодня, как и весь мир, используют искусственный интеллект в анализе бизнеса. Ведь перед тем как доверить ему принятие важнейших решений, необходимо разобраться в его механизме;
- всем, кому небезразличны проблемы этики, которые искусственный интеллект ставит перед миром, и кто захочет посмотреть на работу моделей изнутри и сформировать собственное мнение по таким вопросам, как, например, автономное оружие, целевая реклама, управление данными и т. д.;

- педагогам, заинтересованным в организации и проведении курсов по математике и искусственному интеллекту;
- всем интересующимся искусственным интеллектом.

Кому противопоказана книга

Эта книга вряд ли вызовет интерес у того, кто склонен терпеливо работать с массой упражнений, осваивая какой-либо математический прием или метод. Она также не подойдет ни любителям теорем, ни желающим научиться кодированию и разработке. Это не учебник ни по математике, ни по кодированию. Существует огромное количество отличных учебников, в которых излагаются исчисление, линейная алгебра, теория вероятностей (и только некоторые из них связывают свой материал с искусственным интеллектом). В общем тот, кто захочет разобраться в технических аспектах, строгих формулировках и доказательствах, найдет здесь множество ссылок на соответствующую литературу. На протяжении всей книги мы делаем акцент на концепции, интуиции, общем понимании технологий и в значительной степени меньше внимания уделяем самим технологиям.

Как в книге представлена математика

Написание книги — это в конечном счете процесс принятия решений: как организовать изложение материала таким образом, чтобы наиболее полно раскрыть его суть, и как выбрать, на чем остановиться подробнее, а что оставить за скобками. Местами математика описана подробно, а местами — более сжато. Это сделано намеренно с целью не отвлекаться от основного повествования — о том, *какую математику мы применяем, зачем она нужна и как именно она используется в искусственном интеллекте.*

Сначала мы создаем контекст, в который включены всевозможные приложения искусственного интеллекта. Затем идет обсуждение соответствующего математического материала — детальное или в общих чертах. Когда подробности опускаются, мы задаем вопросы, позволяющие самостоятельно разобраться в предмете. Математика, искусственный интеллект, модели — все они рассматриваются как единое целое. Мы углубляемся в математику только тогда, когда она необходима для основы. Но даже в этом случае мы отдаем предпочтение не формальному, а интуитивному изложению. За это приходится платить тем, что иногда мы вводим новые технические термины и не даем определений, втайне надеясь, что читателю они известны. В каком-то смысле так мы практикуем философию *трансформации* искусственного интеллекта (см. статью Google Brain "Все, что вам нужно, — внимание" ("Attention Is All You Need", 2017), **https://oreil.ly/bJ0is**) для понимания естественного языка, когда модель узнает значения слов из контекста. Поэтому, когда в книге появляется новый технический термин и к нему нет определения, нужно внимательно отнестись к контексту. И тогда по ходу раздела, в котором встречается этот термин, можно будет интуитивно догадаться, о чем идет речь. Еще один вариант — погуглить. Впрочем, постараемся обойтись без жаргонизмов и аббревиатур.

В книге пересекаются математика, наука о данных, искусственный интеллект, машинное обучение и философия, поэтому она рассчитана на разнородную аудиторию, обладающую совершенно различными навыками и опытом. Как следствие, в зависимости от тематики один и тот же материал кому-то покажется тривиальным, а кому-то — чересчур сложным. Я никого не хотела обидеть. Тем не менее такой рискованный шаг предпринят намеренно в надежде, что каждый читатель сможет почерпнуть для себя из книги что-то новое и полезное. Например, математики познакомятся с приложениями искусственного интеллекта, а специалисты по изучению данных и приложениям искусственного интеллекта освежат свои знания из более глубокой математики.

Иногда в том или ином разделе нам будет встречаться материал, изобилующий технической терминологией, так что, если в каком-то месте будет совсем непонятно, пропустите этот раздел и переходите к следующему. Запомните это место — к нему можно будет вернуться чуть позднее.

В книге можно сразу переходить к конкретной теме, поскольку большинство глав не связаны между собой повествованием. На наличие такой связи в тексте всегда дается указание. Каждая глава задумывалась максимально самостоятельной, поэтому в них встречаются объяснения одних и тех же понятий. Тематика распределений вероятностей (в частности, совместного распределения вероятностей признаков набора данных) проходит сквозь все повествование, хотя сама глава, посвященная вероятности, расположена ближе к концу (*см. главу 11*). Это сделано с той целью, чтобы еще до знакомства с грамматикой привыкнуть к языку вероятностей и его использованию в моделях искусственного интеллекта, тем самым формируя верное представление о контексте.

Можно выделить два типа обучающихся: первые — это те, кто изучает конкретику и детали, после чего постепенно формирует общую картину и схему взаимосвязи, тогда как вторым нужно, в первую очередь, увидеть общую картину и взаимосвязи, и только потом по мере необходимости углубляться в детали. И те и другие в равной степени важны, ведь вся разница — лишь в особенностях мышления и природных склонностях. Лично я отношусь, скорее всего, ко второй категории, что, в принципе, подтверждает книга — она дает панорамную картину того, как математика взаимодействует с искусственным интеллектом. Возможно, к концу книги в вашей голове все смешается — не волнуйтесь, это скоро пройдет, зато останутся отличные знания по математике и искусственному интеллекту плюс здоровая порция уверенности в своих способностях.

Помню, как отец обучал меня вождению. Как-то раз он посадил меня за руль, сам сел на пассажирское сиденье и велел трогаться. Где-то через десять минут дорога уперлась в обрыв. Он попросил остановиться, вышел из машины и спустя какое-то время сказал: "Теперь можешь ехать, только постарайся не скатиться с обрыва. Ничего не бойся, я за всем слежу" (как будто это помогло бы). У меня все получилось тогда, да и вообще — дороги вдоль обрыва мне нравятся больше всего. Эту историю можно рассматривать в контексте обучения самоуправляемых автомобилей методом обучения с подкреплением с той лишь разницей, что цена падения с обрыва для меня оказалась бы минус бесконечностью. Безусловно, я не могу себе этого позволить, поскольку я не симулятор, а реальный живой человек, сидящий за рулем реального автомобиля.

Именно так мы будем заниматься математикой и искусственным интеллектом в книге. Мы обойдемся без введений, выводов, определений, теорем, упражнений и тому подобного. Мы полностью погружаемся.

Мы уже погрузились. Мы уже все знаем. Можно трогаться.

Инфографика

Инфографика связывает визуально все темы, изложенные в книге, в единую картину (рис. B1). Ее также можно найти на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**).

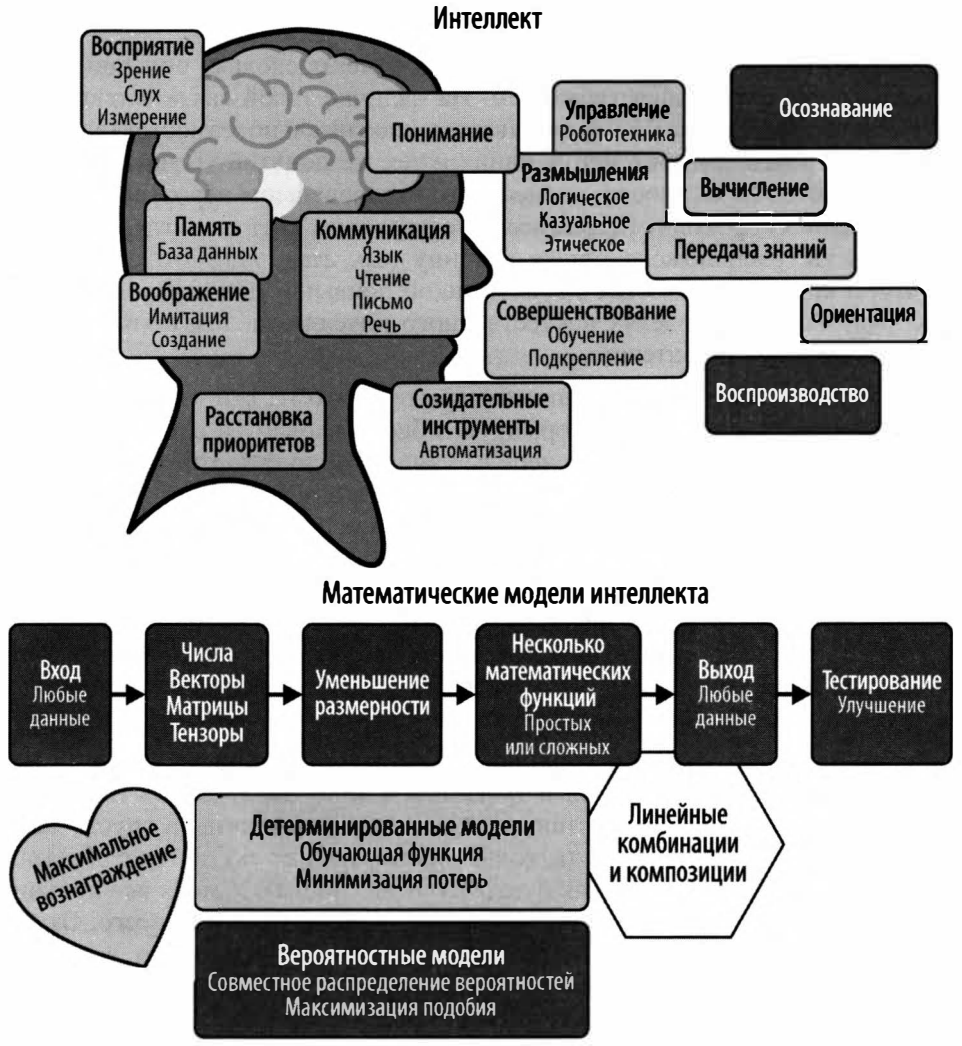


Рис. B1. Инфографика

Требования к математической подготовке для понимания материала книги

Книга является самодостаточной в том смысле, что в ней мотивировано все, что можно применять на практике. Надеюсь, что вы уже знакомы с исчислением и линейной алгеброй, в том числе с операциями с векторами и матрицами, такими как сложение, умножение и некоторые разложения матриц. Также надеюсь, что вам известно о функции и о том, как она преобразует входные данные в выходные. Бóльшая часть математических операций в искусственном интеллекте связана с построением функции, оценкой функции, оптимизацией функции или составлением набора функций. Вам понадобится знание производных (они показывают, насколько быстро все меняется) и цепного правила для производных. Причем сегодня можно даже не знать, как они вычисляются для каждой функции, поскольку современные компьютеры, оснащенные математическим аппаратом Python, Desmos и/или Wolfram|Alpha, выполняют большинство работы за нас, но все же необходимо понимать их назначение. Также не будет лишним знакомство с вероятностным и статистическим мышлением. Но в целом, даже если вы не знаете ничего из вышеперечисленного, в этом не будет трагедии. Возможно, для понимания некоторых аспектов вам придется самостоятельно разобрать несколько примеров (из других книг). Основная хитрость заключается в том, чтобы знать, *когда* искать неизвестное — *только тогда, когда в этом есть необходимость*, т. е. только тогда, когда встречается непонятный термин и вы хорошо представляете себе контекст, в котором он используется. Даже если вы приступаете к чтению, так сказать, с абсолютного нуля, то не сильно отстаете.

По ходу повествования постараемся любой ценой избегать так называемого техницизма.

Краткий обзор глав

Книга содержит в общей сложности 14 глав.

Если вас интересуют математика и технологии искусственного интеллекта в контексте этики, политики, влияния на общество и вытекающих последствий, возможностей, проблем, начинайте чтение с *глав 1* и *14*. Если это вас не интересует, приведу аргументы в пользу того, что их все же следует прочитать. Математика представлена в книге, скорее, как связующее звено, объединяющее, казалось бы, разнородные темы, а не в привычном формате в виде нагромождения сложных формул, теорем, греческих букв.

Тем, кто не знаком с дифференциальными уравнениями (обыкновенными дифференциальными уравнениями или дифференциальными уравнениями в частных производных), *глава 13* покажется обособленной, при этом весьма ценной для тех, кто занимается математическим моделированием, физическими и естественными науками, имитационным или математическим анализом и хочет узнать, как можно использовать искусственный интеллект в их области, и как, в свою очередь, диффе-

ренциальные уравнения могут быть полезны в искусственном интеллекте. На дифференциальные уравнения опираются многочисленные научные достижения, поэтому невозможно оставить их без внимания на заре развития вычислительной технологии, способной решить множество давних проблем, существующих в этой области. Эта глава не связана непосредственно с искусственным интеллектом, но окажется весьма полезной как для понимания математики в целом, так и для формирования теоретических основ искусственного интеллекта и нейрооператоров.

В остальных главах содержится материал, важный для таких областей, как искусственный интеллект, машинное обучение, наука о данных. Невозможно себе представить оптимальное расположение *главы 6*, посвященной сингулярному разложению (важнейший математический инструмент для анализа главных компонент и латентно-семантического анализа, а также отличный метод сокращения размерности). Пусть ваше природное любопытство подскажет, когда переходить к этой главе: до или после той, которая покажется наиболее подходящей. Здесь все зависит от собственного опыта и той отрасли или научной дисциплины, которую вы представляете.

Далее приведем краткий обзор глав в хронологическом порядке с первой по четырнадцатую.

Глава 1. Почему так важно изучать математику искусственного интеллекта?

Искусственный интеллект уже с нами. Он проник во многие сферы жизни, участвует в принятии важных решений, а в обозримом будущем будет использоваться во всех отраслях, во всей деятельности нашего общества. Технологии стремительно развиваются, соперничая по темпу с инвестициями в них. Что такое искусственный интеллект? На что он способен? Каковы его ограничения? Куда он движется? И самое главное — как он работает и почему нам важно знать, как он работает? В этой вводной главе мы кратко рассмотрим важные области применения искусственного интеллекта, проблемы, с которыми обычно сталкиваются компании, пытающиеся интегрировать его в свои системы, обсудим инциденты, возникающие при некачественном внедрении систем, и математику, которая обычно используется в решениях искусственного интеллекта.

Глава 2. Данные, данные, данные.

В главе подчеркивается центральное место данных в искусственном интеллекте. Также в ней даются объяснения различий между понятиями, обычно вызывающими путаницу: структурированные и неструктурированные данные, линейные и нелинейные модели, реальные и имитационные (смоделированные) данные, детерминированные функции и случайные величины, дискретные и непрерывные распределения, предшествующие вероятности, последующие вероятности и функции правдоподобия. Кроме того, без погружения в детали составлена карта вероятностей и статистик, необходимых для искусственного интеллекта, и перечислены наиболее популярные распределения вероятностей.

Глава 3. Подгонка функций под данные.

В основе многих популярных моделей машинного обучения, включая вполне успешные нейросети, благодаря которым в 2012 году искусственный интеллект

вернулся в центр внимания общественности, лежит очень простая математическая задача: подогнать заданный набор точек данных под соответствующую функцию, а затем убедиться, что эта функция будет также превосходно работать на основе новых данных. Этот очевидный факт освещается в главе на примере реального набора данных и других простых примеров. Мы рассмотрим регрессию, логистическую регрессию, машины опорных векторов и другие популярные методы машинного обучения, объединяющие в одной тематике обучающую функцию, функцию потерь, оптимизацию.

Глава 4. Оптимизация нейронных сетей.

Нейросети построены по образцу коры головного мозга, которая состоит из миллионов нейронов, расположенных в виде слоистой структуры. Когда мозг сталкивается с уже известными ему понятиями, он обучается, и связи между нейронами усиливаются, а когда получает новую информацию, которая отменяет или противоречит усвоенным ранее понятиям, эти связи ослабевают. Машины понимают только числа. С математической точки зрения, наиболее крепкие связи соответствуют бóльшим числам (весам), а более слабые — меньшим. В главе обсуждаются этапы оптимизации и обратного распространения ошибки при обучении нейросетей аналогично тому, как происходит обучение в мозге человека (не факт, что люди понимают это до конца). Здесь также рассматриваются различные методы регуляризации, показаны их преимущества, недостатки, примеры использования. Кроме того, вводится понятие интуиции, лежащей в основе теории аппроксимации, и универсальной теоремы аппроксимации для нейросетей.

Глава 5. Сверточные нейронные сети и компьютерное зрение.

Сверточные нейросети широко распространены в компьютерном зрении и обработке естественного языка. Глава начинается с операций свертки и кросс-корреляции, затем показано их применение в проектировании систем, а также при фильтрации сигналов и изображений. Затем происходит интеграция свертки с нейросетями, и из изображений извлекаются признаки более высокого порядка.

Глава 6. Сингулярное разложение: обработка изображений, обработка естественного языка и социальные сети.

Диагональные матрицы ведут себя как скалярные числа и поэтому крайне востребованы. Сингулярное разложение — это важнейший метод линейной алгебры, преобразующий плотную матрицу в диагональную. При этом раскрывается действие матрицы на само пространство: вращение и/или отражение, растяжение и/или сжатие. Этот простой процесс можно применить к любой матрице чисел. Масштабное применение наряду с возможностью значительно уменьшить размерность, сохранив при этом важную информацию, обеспечивают востребованность сингулярного разложения в таких областях, как наука о данных, искусственный интеллект, машинное обучение. Именно этот метод лежит в основе математического анализа главных компонент и латентно-семантического анализа. Глава посвящена сингулярному разложению и его наиболее актуальным, современным приложениям.

Глава 7. Искусственный интеллект для естественного языка и финансов: векторизация и временные ряды.

Математика в этой главе представлена в контексте моделей естественного языка, например определение темы, машинный перевод, модели внимания. Здесь необходимо преодолеть главный барьер — перейти от несущих смысл слов и предложений к низкоразмерным векторам чисел, которые способна обработать машина. Обсуждаются современные модели, например трансформер Google Brain (выпущен в 2017 году) и другие, уделено внимание только актуальной математике. Естественно, говорится о данных временны́х рядов и моделях (например, рекуррентных нейросетях). Дается краткое представление о финансовом искусственном интеллекте, поскольку он пересекается с естественным языком как с точки зрения моделирования, так и с точки зрения взаимовлияния этих двух областей.

Глава 8. Вероятностные генеративные модели.

С каждым днем машинные изображения, в том числе и людей, получаются все более реалистичными. Порой бывает сложно определить, изображение фотомодели перед нами — это реальный человек или образ, созданный компьютером. В этом заслуга генеративно-состязательных сетей и прочих генеративных моделей, в которых виртуальное еще теснее переплелось с реальным. Генеративно-состязательные сети задуманы для повторения простого математического процесса с использованием двух нейросетей до тех пор, пока машина сама не научится отличать реальное изображение от созданного компьютером, отсюда пошла известная концепция "очень близко к реальности". Поскольку две нейросети "конкурируют" друг с другом, мы говорим здесь о теории игр и игре с нулевой суммой. В главе также рассматриваются генеративные модели, имитирующие воображение в человеческом сознании. Область их применений распространяется от дополнения наборов данных для создания маски лица до физики высоких энергий, например, при моделировании наборов данных, аналогичных тем, которые получают на Большом адронном коллайдере в ЦЕРНе.

Глава 9. Графовые модели.

Графы и сети встречаются повсюду — это и населенные пункты с их схемами проезда, и аэропорты со своими стыковочными рейсами, это также Всемирная паутина, облачные технологии (в вычислительной технике), молекулярные сети, наша нервная система, социальные сети, даже разнообразные модели машинного обучения и искусственные нейросети. Данные, имеющие естественную графовую структуру, проще понять с помощью механизма, который использует и сохраняет эту структуру, создавая функции для работы непосредственно с графами, а не встраивает данные о графах в существующие модели машинного обучения, которые до начала анализа пытаются искусственно изменить форму данных. Именно по той же причине сверточные нейросети успешно работают с данными изображений, а рекуррентные нейросети — с последовательными данными и т. д. Математика, лежащая в основе графовых нейросетей, сочетает в себе теорию графов, вычислительную технику, нейросети. О них идет речь в главе, и приводится множество приложений.

Глава 10. Исследование операций.

Исследования операций по-другому можно назвать оптимизацией логистики. Глава знакомит с проблемами, возникающими на стыке искусственного интеллекта и исследования операций, такими как цепочки поставок, коммивояжерство, составление расписания и подбор персонала, постановка в очередь, и другими проблемами, отличительными признаками которых являются высокая размерность и сложность вкупе с необходимостью балансировать между конкурирующими интересами и ограниченными ресурсами. Математика, необходимая для решения подобных задач, опирается на оптимизацию, теорию игр, двойственность, теорию графов, динамическое программирование, алгоритмы.

Глава 11. Вероятность.

Теория вероятностей позволяет дать систематическую количественную оценку случайности и неопределенности. Она обобщает логику до ситуаций, имеющих первостепенное значение для искусственного интеллекта, когда информация и знания неопределенны. В главе обсуждаются основные виды вероятности, которые применяются в приложениях искусственного интеллекта. Это байесовские сети и каузальное (причинно-следственное) моделирование, парадоксы, большие случайные матрицы, стохастические процессы, цепи Маркова, обучение с подкреплением. Конец главы посвящен строгой теории вероятностей, которая объясняет теорию мер и знакомит заинтересованных читателей с универсальной теоремой аппроксимации для нейросетей.

Глава 12. Математическая логика.

Эта важная тема обсуждается ближе к концу книги, чтобы не прерывать естественный ход повествования. В основе искусственного интеллекта лежит создание агентов, способных накапливать знания, логически рассуждать о среде существования, делать выводы, принимать правильные решения на основе своих логических рассуждений. В главе дан краткий обзор пропозициональной логики, логики первого порядка, вероятностной логики, нечеткой логики, темпоральной логики в контексте интеллектуального агента, основанного на знаниях.

Глава 13. Искусственный интеллект и дифференциальные уравнения в частных производных.

Дифференциальные уравнения моделируют бесчисленное множество явлений в реальном мире: турбулентность воздуха, галактика, фондовый рынок, поведение материалов, рост населения. Реалистичные модели часто бывают слишком сложными для решения и требуют огромных вычислительных мощностей при использовании традиционных численных методов. Недавно искусственный интеллект помог в решении дифференциальных уравнений. Первая часть главы представляет собой краткий курс по дифференциальным уравнениям, где освещаются самые значимые темы и дается общая панорама. Во второй части мы обсуждаем новые методы, основанные на искусственном интеллекте, которые упрощают весь процесс решения дифференциальных уравнений. Они

способны разрешить давние проблемы в естественных науках, финансах и других областях.

Глава 14. Искусственный интеллект, этика, математика, право и политика.

Эта глава, на мой взгляд, должна быть первой в любой книге по искусственному интеллекту, причем ее тематика настолько обширна и глубока, что для полного изложения потребуется не одна книга. В главе лишь слегка затронуты различные этические вопросы, связанные с искусственным интеллектом, включая такие понятия, как равенство, справедливость, предвзятость, инклюзивность, прозрачность, политика, регулирование, конфиденциальность, использование оружия, безопасность. Показано возможное решение каждой проблемы (математическое или политическое и регуляторное).

Мои любимые книги по искусственному интеллекту

Существует множество прекрасных и невероятно глубоких книг по искусственному интеллекту и близким к ним тематикам. Приведенный перечень ни в коем случае не претендует на роль исчерпывающего.

В него вошли как технические книги с большим содержанием математики, так и вводная или вовсе не техническая литература. Часть из них ориентирована на работу с кодом (Python 3). Я многое узнала из этих книг.

- Brunton S. L., Kutz J. N. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control. — 2nd edition. — Cambridge University Press, 2022.
- Crawford K. Atlas of AI. — Yale University Press, 2021.
- Ford M. Architects of Intelligence. — Packt Publishing, 2018.
- Geron A. Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow. — O'Reilly, 2022.
- Goodfellow I. Yoshua Bengio, and Aaron Courville, Deep Learning. — MIT Press, 2016.
- Grus J. Data Science from Scratch. — O'Reilly, 2019.
- Hawkins J. A Thousand Brains. — Basic Books, 2021.
- Izenman A. J. Modern Multivariate Statistical Techniques. — Springer, 2013.
- Jones H. Data Science: The Ultimate Guide to Data Analytics, Data Mining, Data Warehousing, Data Visualization, Regression Analysis, Database Querying, Big Data for Business and Machine Learning for Beginners. — Bravex Publications, 2020.
- Kleppmann M. Designing Data-Intensive Applications. — O'Reilly, 2017.
- Lakshmanan V., Robinson S., Munn M. Machine Learning Design Patterns. — O'Reilly, 2020.

- Lane H., Hapke H., Howard C. Natural Language Processing in Action. — Manning, 2019.
- Lee Kai-Fu. AI Superpowers. — Houghton Mifflin Harcourt, 2018.
- 97 Things Every Data Engineer Should Know / ed. by T. Macey. — O'Reilly, 2021.
- Marr B., Ward M. Artificial Intelligence in Practice. — Wiley, 2019.
- Moroney L. AI and Machine Learning for Coders. — O'Reilly, 2021.
- Mount G. Advancing into Analytics: From Excel to Python and R. — O'Reilly, 2021.
- Norvig P., Russell S. Artificial Intelligence: A Modern Approach. — Pearson, 2021.
- Pearl J. The Book of Why. — Basic Books, 2020.
- Planche B., Andres E. Hands-On Computer Vision with Tensor-Flow2. — Packt Publishing, 2019.
- Potters M., Bouchaud J.-P. A First Course in Random Matrix Theory for Physicists, Engineers, and Data Scientists. — Cambridge University Press, 2020.
- Rosenthal J. S. A First Look at Rigorous Probability Theory. — World Scientific Publishing, 2016.
- Roshak M. Artificial Intelligence for IoT Cookbook. — Packt Publishing, 2021.
- Strang G. Linear Algebra and Learning from Data. — Wellesley Cambridge Press, 2019.
- Stone J. V. Artificial Intelligence Engines. — Sebtel Press, 2020.
- Stone J. V. Bayes' Rule, A Tutorial Introduction to Bayesian Analysis. — Sebtel Press, 2013.
- Stone J. V. Information Theory: A Tutorial Introduction. — Sebtel Press, 2015.
- Vajjala S. et al. Practical Natural Language Processing. — O'Reilly, 2020.
- Van der Hofstad R. Random Graphs and Complex Networks. — Cambridge, 2017.
- Vershynin R. High-Dimensional Probability: An Introduction with Applications in Data Science. — Cambridge University Press, 2018.

Условные обозначения и соглашения

В книге используются следующие общепринятые типографские обозначения.

Курсивом

Обозначены новые термины.

`Моноширинный шрифт`

Используется в программных выводах, а также внутри абзацев для обозначения таких элементов программы, как имена переменных и функций, базы данных, типы данных, переменные среды, операторы, ключевые слова.

Данный элемент обозначает подсказку или совет.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

Примеры применения кода

В книге приведено несколько примеров кода, доступных по адресу **https://github.com/halanelson/Essential-Math-For-AI**.

В случае технических вопросов или проблем с использованием примеров кода, пожалуйста, отправьте письмо по адресу **bookquestions@oreilly.com**.

Книга предназначена для практической работы. В большинстве случаев можно свободно пользоваться кодами, приведенными в ней в качестве примеров, в своих программах и документациях. Для воспроизведения незначительной части кода не требуется разрешения. Например, в тех случаях когда вы пишете программу, используя несколько фрагментов кода из книги, или отвечаете на вопрос и даете ссылку на книгу с приведением примера кода из нее. Но для того чтобы включить в документацию своего продукта большое количество примеров из этой или других книг издательства O'Reilly, уже потребуется соответствующее разрешение.

Мы приветствуем, но, как правило, не требуем указывать авторство. Авторство включает название произведения, имя автора, название издательства и ISBN. Например: "Essential Math for AI by Hala Nelson (O'Reilly). Copyright 2023 Hala Nelson, 978-1-098-10763-5".

Если вы считаете, что использование примеров кода не соответствует принципам добросовестного использования или приведенному выше разрешению, обращайтесь к нам по адресу **permissions@oreilly.com**.

Платформа онлайн-обучения O'Reilly

Более 40 лет компания O'Reilly Media организует технические и бизнес-тренинги, знания и опыт, чтобы помочь компаниям достичь успеха.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом через книги, статьи и платформу онлайн-обучения. Эта платформа предоставляет доступ по требованию к интерактивным учебным курсам, углубленным программам обучения, интерактивным средам кодирования, а также к обширной коллекции текстовых и видеоматериалов от O'Reilly и более 200 других издательств. Дополнительная информация представлена на сайте **https://oreilly.com**.

Как с нами связаться?

Просим направлять комментарии и вопросы относительно данной книги в издательство по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA, 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

На веб-странице книги можно ознакомиться с редакциями, примерами и другой дополнительной информацией. Она доступна по адресу **https://oreil.ly/essentialMathAI**.

Если вы хотите оставить комментарий или задать технические вопросы по книге, пишите по адресу **bookquestions@oreilly.com**.

Новости и информацию о наших книгах и курсах можно найти на сайте **https://oreilly.com**.

Ищите нас на LinkedIn: **https://linkedin.com/company/oreilly-media**.

Следите за нами в Twitter: **https://twitter.com/oreillymedia**.

Смотрите нас на YouTube: **https://www.youtube.com/oreillymedia**.

Благодарности

Моему отцу, Йосифу Зейну, обучившему меня математике и постоянно напоминавшему: *"Не думай, что земля или деньги — это лучшее, что мы дали тебе в этой жизни. Они приходят и уходят. Люди делают деньги, покупают активы и делают еще больше денег. Единственное, что мы передали тебе, — это ум, действительно прекрасный ум. Вот твой настоящий актив, так что используй его по полной"*. Я восхищаюсь твоим умом, и эта книга посвящена тебе, папа.

Моей матери, Самире Хамдан, обучившей меня английскому языку и философии, и отдавшей все силы ради нашего счастья и успеха. Благодаря тебе, мама, я написала эту книгу на английском — не родном для меня языке.

Дочери Саре, которая поддерживала во мне жизнь в самые отчаянные моменты. Она — радость моей жизни.

Мужу Киту, который дарит мне любовь и преданность, а также стабильность и опору, что позволяет мне быть собой и делать множество вещей, часть из которых неразумны, например писать пятисотстраничную книгу о математике и искусственном интеллекте. Я люблю тебя.

Сестре Раше, которая является моей второй половинкой. Этим все сказано.

Брату Хайтаму, который ради поддержки мне пошел против всех наших культурных норм и традиций.

Памяти дяди Омара Зейна, который также преподавал философию и влюбил меня в тайны человеческого разума.

Моим друзьям Шэрон и Джейми, которые разрешили мне написать большую часть этой книги у них дома и были прекрасными редакторами в любое время, когда бы я ни попросила.

Моему закадычному другу Орену, который, помимо того, что является одним из лучших друзей, каких только можно пожелать, согласился прочитать и прорецензировать книгу.

Моему другу Хуану Нгуену (**https://oreil.ly/XdWpu**), история которого должна выйти отдельной книгой, и который также нашел время, чтобы прочитать и просмотреть книгу. Спасибо, адмирал.

Моему другу и коллеге Джону Веббу, который прочитал каждую главу слово за словом и поделился своим бесценным видением чистой математики.

Моим замечательным друзьям Дебе, Панкаджу, Джейми, Тамару, Саджиде, Джамиле, Джене, Маттиасу и Карен, которые стали частью моей семьи. Мне нравится жить в одном мире с вами.

Моим наставникам Роберту Кону (Нью-Йоркский университет) и Джону Шотланду (Йельский университет), без которых не состоялся бы мой профессиональный рост. Вы научили меня многому.

В память о Питере, чье влияние было огромным и который всегда будет вдохновлять меня.

Рецензентам этой книги, которые, несмотря на свою занятость, нашли время и сделали ее намного лучше. Спасибо за ваше бесценное экспертное мнение и щедрость, с которой вы делитесь со мной своими уникальными взглядами из разных областей.

Всем официантам и официанткам во многих городах мира, которые терпели, когда я часами сидела за ноутбуком в их ресторанах и писала эту книгу. Я получила от вас столько энергии и счастья.

Моим невероятным, терпеливым, веселым и всегда поддерживающим меня редакторам Анжеле Руфино и Кристен Браун.

ГЛАВА 1

Почему так важно изучать математику искусственного интеллекта?

https://t.me/it_boooks/2

Только когда кто-то сказал: "Это разумно", я прекратила всякие поиски и обратила на это внимание.

— Хала

Искусственный интеллект (ИИ — artificial intelligence, AI) уже присутствует в нашей жизни. Он проник во многие сферы и все чаще участвует в принятии нами самых важных решений. В ближайшей перспективе его будут использовать во всех областях общественной жизни, обеспечивая большинство наших повседневных операций. Технологии развиваются стремительно, но инвестиции в них растут еще быстрее. В то же время создается впечатление, будто мы находимся в эпицентре ажиотажного спроса на искусственный интеллект. Каждый день мы слышим о его новом достижении. Искусственный интеллект побеждает лучшего игрока в го. Искусственный интеллект превзошел человеческое зрение в задачах классификации. Искусственный интеллект создает неотличимые подделки. Искусственный интеллект генерирует данные по физике высоких энергий. Искусственный интеллект решает сложные дифференциальные уравнения, моделирующие естественные явления окружающего мира. На дорогах появились самоуправляемые автомобили. В каких-то частях света парят беспилотные летательные аппараты.

Мы также слышим о неограниченном, на первый взгляд, потенциале искусственного интеллекта. Искусственный интеллект произведет революцию в здравоохранении и образовании. Искусственный интеллект устранит голод в мире. Искусственный интеллект будет бороться с изменением климата. Искусственный интеллект спасет исчезающие виды животных. Искусственный интеллект будет бороться с болезнями. Искусственный интеллект оптимизирует цепочку поставок. Искусственный интеллект разгадает тайну происхождения жизни. Искусственный интеллект составит карту наблюдаемой Вселенной. Наши города и дома становятся "умными". Кажется, что постепенно мы вступаем на территорию научной фантастики. Человеческий мозг будет подключен к компьютеру. Людей будут *совершенствовать* с помощью искусственного интеллекта. И, наконец, раздаются испуганные или порой скептические голоса: искусственный интеллект может захватить власть и уничтожить человечество.

Поскольку при переизбытке такого рода информации стираются грани между реальностью и домыслами, чаяниями, преувеличениями и чистой фантастикой, я сна-

чала приведу четкое определение искусственного интеллекта исключительно в контексте книги. Затем мы обсудим его ограничения, направления развития, закладывая базу для математики, которая применяется в современном искусственном интеллекте. Понимание математики, возможно, позволит нам еще больше углубиться в предмет, и тогда границы между реальностью и вымыслом начнут обретать более ясные очертания. Мы также обсудим главные идеи, лежащие в основе современного математического аппарата в искусственном интеллекте и позволяющие с уверенностью как использовать, так и совершенствовать и даже создавать совершенно новые системы искусственного интеллекта.

Что такое искусственный интеллект

Сегодня практически невозможно найти единое определение искусственного интеллекта. Каждый эксперт в этой области отвечает по-своему. Но даже если мы попросим об этом кого-то одного из них, то, вероятнее всего, каждый раз он ответит по-новому. Причина этого феномена заключается в том, что мы до сих пор так и не можем дать четкого определения, что такое "Я". Что такое интеллект? Что делает нас людьми, причем уникальными? Что заставляет нас осознавать свое существование? Как нейроны в нашем мозге собирают крошечные электрические импульсы и преобразуют их в образы, звуки, чувства, мысли? Эти необъятные темы на протяжении веков волновали философов, антропологов, нейробиологов. В книге мы стараемся их обойти.

Итак, рассмотрим искусственный интеллект в контексте его агента и для целей книги обозначим определяющие принципы. Так, в 2022 году агент искусственного интеллекта должен был отвечать одному или нескольким из перечисленных далее критериев.

- Агент искусственного интеллекта является либо чисто программным обеспечением, либо физическим роботизированным устройством.
- Агент искусственного интеллекта либо ориентирован на выполнение конкретной задачи, либо является гибким агентом, который исследует свое окружение и манипулирует им, накапливает знания с определенной целью или без нее.
- Агент искусственного интеллекта *обучается* с учетом накопленного опыта, т. е. с увеличением практики повышается качество выполнения задачи.
- Агент искусственного интеллекта воспринимает окружающую среду, после чего создает, обновляет и/или развивает модель этой среды.
- Агент искусственного интеллекта воспринимает, моделирует, анализирует и принимает решения, ведущие к достижению цели. Эта цель может быть предопределенной и зафиксированной, а может меняться по мере поступления новых данных.
- Агент искусственного интеллекта понимает причину и следствие, способен различать закономерности и причины.

В книге всегда отмечается, когда речь идет о математической модели искусственного интеллекта, функционирующей по аналогии с мозгом, и мы можем сравнить искусственный и человеческий интеллект без их определений.

Современный искусственный интеллект пока еще во многом уступает человеческому, за исключением решений специфических задач, среди которых, например, классификация изображений, программа AlphaGo и т. д. Безусловно, это послужило причиной того, что сегодня все больше и больше специалистов объединяют свои усилия в разработке искусственного интеллекта, так что в ближайшее время мы может стать очевидцами небывалых темпов роста и мощных прорывов в этом направлении.

Важно также отметить, что в некоторых опубликованных работах термины "искусственный интеллект", "машинное обучение", "наука о данных" используются как взаимозаменяемые. Эти три области пересекаются, но они не одно и то же. Четвертой значимой, хотя чуть менее популярной, областью является робототехника, где в процессы обучения и рассуждения необходимо интегрировать физические компоненты и двигательные навыки, объединяя машиностроение, электротехнику, биоинженерию с информационной и компьютерной инженерией. Если коротко, то можно примерно так объяснить взаимосвязь этих областей: *данные питают алгоритмы машинного обучения, которые, в свою очередь, питают множество популярных систем искусственного интеллекта и/или робототехники*. В целом математика, о которой говорится в книге, так или иначе окажется полезной во всех четырех областях.

Секрет популярности искусственного интеллекта

За последние десять лет искусственный интеллект привлек всеобщее внимание к себе благодаря удачному сочетанию ряда факторов.

Способность генерировать и оцифровывать огромные массивы данных.

Сюда относятся текстовые данные, фото- и видеоизображения, медицинские записи, данные электронной коммерции, сетевые данные, данные датчиков. Немаловажную роль играют социальные сети и Интернет вещей (Internet of Things, IoT), обеспечивающие непрерывный поток огромных объемов данных.

Развитие вычислительных мощностей.

Речь идет о параллельных и распределенных вычислениях, а также инновациях в аппаратном обеспечении, позволяющих эффективно и относительно дешево обрабатывать большие объемы сложных структурированных и неструктурированных данных.

Современные достижения нейросетей в осмыслении больших данных.

В некоторых задачах искусственный интеллект превзошел человека, например в распознавании изображений и игре го. В 2012 году сеть AlexNet (**https://oreil.ly/GubT1**) победила в конкурсе ImageNet Large Scale Visual Recognition Challenge

(**https://oreil.ly/1LchH**) и тем самым дала начало активной работе в области сверточных нейросетей (на базе графических процессоров), а в 2015 году сеть PReLU-Net (ResNet, **https://oreil.ly/0TYPA**) впервые превзошла человека в классификации изображений.

Проанализировав все указанные аспекты, можно предположить, что современный искусственный интеллект — это совсем не то, о чем пишут в научно-фантастических книгах. Искусственный интеллект ориентирован на работу с большими данными (всеми видами данных), алгоритмами машинного обучения, причем в значительной степени он нацелен не на развитие и адаптацию различных типов интеллекта и целей в условиях окружающей среды, а на идеальное выполнение одной конкретной задачи.

Возможности искусственного интеллекта

Существует гораздо больше областей и отраслей для успешного применения искусственного интеллекта, чем специалистов в этой области, способных удовлетворить непрерывно растущие потребности.

Человечество стремится к автоматизации процессов, а искусственный интеллект обещает в итоге сделать именно это, причем в огромных масштабах. В организациях, как больших, так и малых, скапливаются объемы необработанных данных, которые можно анализировать, превращая их в выводы по увеличению прибыли, разработке оптимальных стратегий, распределению ресурсов. К примеру, сегодня можно использовать неисчерпаемый потенциал искусственного интеллекта и найти ему массу применений в медицине, где ощущается острая нехватка врачей.

Мировые финансовые системы, фондовые рынки, банковская сфера зависят в значительной степени от способности делать верные прогнозы и сильно страдают, когда эти прогнозы не сбываются.

Сегодня благодаря прогрессу вычислительных возможностей научные исследования значительно продвинулись вперед, и мы вступили на порог нового рассвета, когда достижения в области искусственного интеллекта позволяют проводить вычисления в масштабах, которые еще несколько десятилетий назад считались невозможными. Эффективные системы и операции требуются везде — в энергосистемах, транспорте, цепочках поставок, охране лесов и дикой природы, в борьбе с голодом, болезнями и изменениями климата. Более того, в самих системах искусственного интеллекта наблюдается тенденция к автоматизации, когда система самопроизвольно выбирает оптимальные конвейеры, алгоритмы, параметры и легко добивается желаемых результатов при решении поставленных задач, так что необходимость в контроле со стороны человека полностью исчезает.

Конкретные задачи агента искусственного интеллекта

В книге, по мере изучения математики, рассматриваются популярные прикладные области искусственного интеллекта в контексте конкретных задач его агента. В це-

лом полезные математические идеи и методы легко переносятся в различные прикладные области. Причина такой видимой легкости и универсальности заключается в том, что мы живем в эпоху *внедрения* искусственного интеллекта, и это означает, что основные идеи для решения конкретных задач уже разработаны и после небольшой доработки могут быть *реализованы* в различных отраслях и направлениях. Тематика и/или задачи нашего агента искусственного интеллекта включают в себя следующее.

Имитированные (смоделированные) и реальные данные.

Наш агент искусственного интеллекта обрабатывает данные, делает выводы, принимает решения на основе этих данных (с использованием математики и алгоритмов).

Новая кора головного мозга (неокортекс).

Нейросети в искусственном интеллекте моделируются на основе неокортекса, или *нового мозга*. Эта часть нашего мозга отвечает за такие тончайшие функции, как восприятие, память, абстрактное мышление, язык, волевые физические действия, принятие решений, воображение, сознание. Неокортекс имеет множество слоев, шесть из которых хорошо различимы. Он гибок и обладает огромной способностью к обучению. Ниже неокортекса расположены *"старый мозг"* и *"рептильный мозг"*. Они отвечают за эмоции и простые, самые примитивные функции выживания, такие как дыхание, регуляция сердцебиения, страх, агрессия, сексуальные позывы и др. "Старый мозг" хранит записи действий и переживаний, вызывающих благоприятные или неблагоприятные ощущения, что формирует эмоциональную память, которая влияет на наше поведение и будущие поступки. В самых общих чертах, наш агент искусственного интеллекта имитирует работу неокортекса и иногда "старого мозга".

Компьютерное зрение.

Агент искусственного интеллекта ощущает и распознает окружающую среду с помощью камер, датчиков и т. д. Он "заглядывает" во все, начиная с наших повседневных фотографий и видео, заканчивая снимками магнитно-резонансной томографии и изображениями далеких галактик.

Обработка естественного языка.

Агент искусственного интеллекта взаимодействует с окружающей средой и автоматизирует утомительные, трудоемкие задачи, такие как реферирование текста, перевод на другой язык, анализ настроений, классификация и ранжирование документов, создание подписей к изображениям, общение с пользователями.

Финансовые системы.

Агент искусственного интеллекта распознает мошеннические операции в ежедневных транзакциях, оценивает кредитные риски, а также круглосуточно предоставляет обратную связь и информацию по нашим финансовым привычкам.

Сети и графы.

Агент искусственного интеллекта обрабатывает сетевые и графовые данные, включая социальные сети животных, инфраструктурные сети, сети профессио-

нального сотрудничества, экономические сети, транспортные сети, биологические сети и многие другие.

Социальные сети.

Агент искусственного интеллекта благодарен социальным сетям за большое количество данных, необходимых для его обучения. В свою очередь, он старается дать характеристику пользователям социальных сетей, выявить их модели поведения и активные сети.

Цепочка поставок.

Агент искусственного интеллекта является специалистом по оптимизации. Он помогает прогнозировать оптимальные потребности в ресурсах и стратегии их распределения на каждом уровне производственной цепочки. Он также ищет способы покончить с голодом в мировом масштабе.

Планирование и подбор персонала.

Агент искусственного интеллекта облегчает выполнение ежедневных операций.

Прогноз погоды.

Агент искусственного интеллекта решает дифференциальные уравнения, которые используются в прогнозировании погоды.

Изменение климата.

Агент искусственного интеллекта борется с изменением климата.

Образование.

Агент искусственного интеллекта обеспечивает персональное обучение.

Этика.

Агент искусственного интеллекта стремится быть справедливым, равноправным, инклюзивным, прозрачным, беспристрастным, а также защищать безопасность и конфиденциальность данных.

Ограничения искусственного интеллекта

Впечатляющие достижения в области искусственного интеллекта раскрывают широчайшие перспективы для совершенствования и даже революционизирования целых отраслей, но для этого нужно преодолеть ряд реальных ограничений. К самым актуальным из них относятся следующие.

Интеллект.

Современный ИИ даже отдаленно не напоминает *интеллект* (ум) в том смысле, в котором мы, люди, считаем себя единственно разумными. Даже превзойдя человека в решении множества задач, он все еще не способен естественным образом переключаться и адаптироваться к новым задачам. Например, система ИИ, обученная распознаванию лиц, не может без переобучения распознавать кошек или генерировать текст без изменения своей архитектуры и алгоритмов. Если

говорить о трех типах ИИ, то на сегодня лишь частично создан искусственный интеллект узкого назначения, или *слабый интеллект*, который обладает узким спектром возможностей. До сих пор не разработаны ни *искусственный общий интеллект*, сопоставимый с человеческими возможностями, ни *искусственный сверхинтеллект*, превосходящий человеческий. Более того, современные машины не могут переживать никаких человеческих эмоций, таких как любовь, близость, счастье, гордость, достоинство, забота, печаль, потеря и многие другие. Ведь имитация эмоций — это не то же самое, что их подлинное переживание и проявление. В этом смысле машина пока еще не может заменить человека.

Большие объемы размеченных данных.

Для большинства популярных приложений ИИ требуются большие объемы размеченных данных, например снимки МРТ размечаются как раковые или нераковые, видеоролики на YouTube — как рекомендованные или не рекомендованные для детей, в ценах на жилье учитываются такие показатели, как район, количество спален, средний доход семьи и другие (в данном случае стоимость недвижимости является меткой). К ограничениям можно отнести тот факт, что необходимые для обучения системы данные обычно не всегда доступны, а их получение, маркировка, обслуживание, хранение могут обходиться недешево. Значительный объем данных является конфиденциальным, неорганизованным, неструктурированным, необъективным, неполным, неразмеченным. Получение данных, их хранение, предварительная обработка, маркировка являются серьезными препятствиями, требующими больших временны́х и ресурсных затрат.

Разнообразие методов и гиперпараметры.

Иногда для решения определенной задачи ИИ можно воспользоваться несколькими методами или алгоритмами. Каждая задача, набор данных и/или алгоритм имеют параметры, которые называются *гиперпараметрами* и могут настраиваться в процессе реализации, причем их оптимальные значения не всегда бывают понятны. Большое количество разнообразных методов и гиперпараметров, способных решить конкретную задачу ИИ, может показать в итоге совершенно разные результаты, и в таком случае выбор нужного метода остается за человеком. В некоторых приложениях, например при выборе фасона платья для конкретного покупателя, эти расхождения могут быть несущественными.

Решения, принимаемые искусственным интеллектом в других областях, могут оказаться судьбоносными: пациенту говорят, что у него нет конкретного заболевания, хотя в реальности оно есть; заключенному ошибочно указывают на высокую вероятность повторного совершения преступления и, как следствие, отказывают в условно-досрочном освобождении; квалифицированному специалисту отказывают в кредите. В настоящее время ведутся исследования в поисках решения всех этих проблем, и мы еще вернемся к ним в процессе дальнейшего повествования.

Ограниченность ресурсов.

Возможности и потенциал человека ограничены возможностями мозга и биологического тела, а также ресурсами Земли и Вселенной, с которыми мы научи-

лись обращаться. Ресурсы, в свою очередь, также ограничены производительностью и возможностями человеческого мозга. Аналогичным образом системы ИИ ограничены вычислительной мощностью и аппаратными возможностями систем, поддерживающих программное обеспечение искусственного интеллекта. Недавние исследования показали, что интенсивное глубокое обучение приближается к своим вычислительным пределам, а для повышения эффективности алгоритмов и аппаратных средств, а также разработки совершенно новых методов требуются новые идеи. Прогресс в области искусственного интеллекта во многом зависит от значительного увеличения вычислительной мощности. Однако вычислительные мощности не безграничны, они обходятся слишком дорого для больших систем обработки огромных массивов данных и несут значительный углеродный след, который невозможно игнорировать, например энергия, затрачиваемая на работу и охлаждение хранилищ данных, отдельных устройств, поддержания связи с облаком и т. д. Кроме того, данные и алгоритмическое программное обеспечение не существуют в вакууме, сами по себе. Компьютеры, телефоны, планшеты, аккумуляторы, а также хранилища и системы, необходимые для хранения, передачи, обработки данных и алгоритмов, — все эти устройства сделаны из реальных физических материалов, добытых из недр Земли. Процесс образования некоторых из них занял миллионы лет, так что сегодня стоит задумываться о том, что запас для поддержания этих технологий не бесконечен.

Расходы на безопасность.

Основными проблемами ИИ остаются безопасность, конфиденциальность, враждебные атаки, особенно с появлением взаимосвязанных систем. Их решению посвящено множество исследований, на них выделяется огромное количество ресурсов. Большинство современных систем ИИ представляют собой программное обеспечение, а большинство данных являются цифровыми, порождая непрекращающуюся своего рода "гонку вооружений" в этой сфере. Это значит, что системы искусственного интеллекта должны жестко контролироваться и постоянно обновляться, что требует привлечения более дорогостоящих специалистов по ИИ и кибербезопасности, но в итоге это позволит уменьшить стоимость автоматизации в масштабах всей компании.

Более масштабные последствия.

До недавнего времени отрасли, занимающиеся исследованиями и внедрением ИИ, не придавали особого значения экономическим и социальным последствиям, а также последствиям в сфере безопасности, возникающим в связи с развитием их технологий. В большинстве случаев эти последствия отмечаются как важные и заслуживающие внимания, но выводятся за рамки проектов. По мере того как ИИ все больше проникает в нашу жизнь, усиливая свое влияние на структуру и характер общества, поведение рынков, возникновение потенциальных угроз, всем специалистам, занятым в этой области, необходимо уделять более пристальное внимание этим важнейшим вопросам. В этом смысле разработчики ИИ ограничены в ресурсах, которые выделяются на решение более общих проблем, связанных с последствиями масштабного применения новых технологий.

Что происходит, когда системы искусственного интеллекта дают сбой

Достаточно важно при изучении ИИ вывить его склонность к ошибкам и сбоям. Это позволяет предсказать и избежать пагубных последствий еще на этапе разработки искусственного интеллекта *до* его реализации в реальном мире. Ошибка реализованной системы может привести к крайне нежелательным, даже катастрофическим, результатам.

Онлайновая база данных об инцидентах в работе искусственного интеллекта "AI Incident Database" (**https://incidentdatabase.ai/**) содержит более тысячи подобных случаев, например:

- самоуправляемый автомобиль сбивает насмерть пешехода;
- на улицах Сан-Франциско (28 июня и 18 мая 2022 года) самоуправляемые автомобили на 20 минут теряют связь с сервером своей компании, и тут же все глохнут;
- торговый алгоритм провоцирует *обвал рынка*, в результате которого миллиарды долларов автоматически перетекают от одних участников к другим;
- система распознавания лиц служит причиной ареста невиновного человека;
- печально известный чат-бот Тэй компании Microsoft закрывается уже спустя 16 часов после запуска, поскольку он быстро научился оскорбительным, расистским, а также крайне резким провокационным высказываниям и опубликовал их в Твиттере.

Для смягчения негативных последствий необходимо глубокое понимание того, как работают эти системы на всех уровнях производства, а также среды и пользователей, на которые они рассчитаны. Важнейшим шагом в данном направлении является понимание математики, лежащей в основе искусственного интеллекта.

Направления развития искусственного интеллекта

Прежде чем ответить или даже подумать о пути развития искусственного интеллекта, необходимо напомнить его первоначальную цель с момента возникновения — имитация человеческого интеллекта. Это произошло в пятидесятых годах прошлого века. Сделаем небольшой экскурс в его семидесятилетнюю историю и, *может быть*, там найдем подсказку о вероятных перспективах. К тому же история развития искусственного интеллекта и его тенденций дает превосходный панорамный вид на предмет сверху в окружении контекста и вероятных перспектив. Безусловно, это позволяет сделать процесс изучения нужной математики менее утомительным. Затем, опустив технические подробности, рассмотрим этапы развития ИИ вплоть до его появления на авансцене на фоне впечатляющих успехов в глубоком обучении.

Самый ранний этап исследований в этой области можно охарактеризовать многочисленными попытками смоделировать интеллект с помощью правил и логики.

Идея заключалась в том, что машины достаточно "нагрузить" фактами и логическими правилами рассуждений об этих фактах (примеры такой логической структуры представлены в *главе 12*). При этом речи не шло об обучении. Проблема, с которой столкнулись разработчики, заключалась в огромном количестве правил и ограничений в системе человеческих знаний, не поддающихся кодированию, следовательно, данный подход представлялся невыполнимым.

На рубеже 1990–2000-х годов методы машинного обучения обретают популярность. Теперь машинное обучение не программирует правила, не делает выводы и не принимает решения на основе запрограммированных правил, а выводит правила из данных. Чем больше данных может обработать система машинного обучения, тем выше ее производительность. Основные цели — данные и возможность экономичной и эффективной обработки их больших объемов, а также *обучения на их основе*. Популярными алгоритмами машинного обучения на тот период становятся машины опорных векторов, байесовские сети, эволюционные алгоритмы, деревья решений, случайные леса, регрессия, логистическая регрессия и др. Сегодня они не менее популярны.

После 2010 года, особенно на фоне успехов сверточной нейросети AlexNet в распознавании образов в 2012 году, в мире отмечается повальное увлечение нейросетями и глубоким обучением.

В последние пять лет стало набирать популярность обучение с подкреплением после уверенной победы сети AlphaGo компании DeepMind над чемпионом мира в сложнейшей древнекитайской игре го.

Важно напомнить, что это лишь краткий экскурс в историю, ведь понятие регрессии возникло еще во времена Лежандра и Гаусса в начале 1800-х годов, а первые формулировки искусственных нейронов и нейросетей появились на рубеже 1940–1950-х годов в работах нейрофизиолога Уоррена Маккаллоха, математика Уолтера Питтса, психологов Дональда Хебба и Фрэнка Розенблатта. В 1950 году специалист по вычислительной технике, криптоаналитик, математик, биолог-теоретик Алан Тьюринг в работе "Вычислительные машины и разум" (**https://oreil.ly/bJp5a**) предложил так называемый тест Тьюринга, который первоначально назывался "Игра в имитацию". Тьюринг полагал, что машина обладает искусственным интеллектом, если ее реакции неотличимы от человеческих. Таким образом, умная машина — это машина, способная имитировать человеческие реакции. Но для человека, далекого от компьютерной области, такое определение интеллекта выглядит весьма ограниченным, и здесь возникает вопрос: насколько тест Тьюринга ограничивает цели и направление исследований в этой области?

Сегодня машины могут имитировать человеческий интеллект для решения конкретных задач, но первоначальная цель — воспроизвести человеческий интеллект — до сих пор так и не достигнута, поэтому можно с уверенностью предположить, что развитие предстоит именно в данном направлении, хотя для этого, возможно, потребуется возрождение старых концепций и идей или изобретение абсолютно новых. Растущие инвестиции в эту область, подъем исследовательской активности, повышение общественного интереса к предмету — все это постепенно

подготавливает условия для новых прорывов. Более того, последние достижения в области искусственного интеллекта уже совершают революцию в целых отраслях, где идет активное внедрение новейших технологий. Все эти достижения связаны с множеством важных математических понятий, которые мы будем обсуждать на протяжении всей книги.

Кто сегодня вносит основной вклад в развитие искусственного интеллекта

Основная конкуренция в области искусственного интеллекта ведется между США, Европой и Китаем. Среди мировых лидеров в области технологий следует отметить Google и ее материнскую компанию Alphabet, а также Amazon, Facebook[1], Microsoft, Nvidia, IBM в США, DeepMind в Великобритании и США (принадлежит Alphabet), Baidu и Tencent в Китае. Кроме того, есть крупные участники из академической среды, но поскольку их количество достаточно велико, перечислить их здесь не представляется возможным. Новичкам будет полезно узнать имена крупных игроков, их историю и вклад, а также цели, которые они преследуют сегодня. Причем не менее важным будет изучить противоречия (если таковые имеются), возникающие в их деятельности. Эти общие знания пригодятся в дальнейшем по мере освоения материала и приобретения опыта работы с ИИ.

Какая математика используется в искусственном интеллекте

Когда вы слышите слово "математика", какие темы и предметы приходят вам на ум?

Независимо от того, являетесь ли вы знатоком математики или пока еще новичком, отвечая на этот вопрос, можно подумать о любом предмете математики, и вероятнее всего, он уже применяется в искусственном интеллекте. Исчисление, линейная алгебра, оптимизация, вероятность, статистика — все эти дисциплины просто необходимы, чтобы успешно реализовать ИИ. Для этого не требуется быть экспертом во всех этих областях, нужно только хорошо разбираться в отдельных предметах из них. В зависимости от конкретной области применения могут пригодиться знания из теории случайных матриц, теории графов, теории игр, дифференциальных уравнений, исследования операций. В книге мы обсуждаем эти предметы, не претендуя на роль учебника. Их объединяет одна тематика — применение и реализация искусственного интеллекта. Возможно, упрощая многие технические определения или опуская целые теоремы и подробные детали, я рискую обидеть математиков и экспертов в области ИИ и других смежных отраслей. Но моя задача в том, чтобы

[1] Деятельность Facebook признана экстремистской на территории Российской Федерации, а сама социальная сеть заблокирована. — *Прим. ред.*

книга была доступной и легкой для чтения и вместе с тем охватывала бы большинство математических аспектов, играющих важную роль в приложениях искусственного интеллекта. Для тех, кто захочет подробнее изучить математику или конкретную область ИИ, существуют более глубокие работы по интересующей теме. В книге я постаралась изложить материал лаконично в виде краткого всестороннего обзора, прочитав который можно уверенно переходить к любой интересующей математической дисциплине или области применения ИИ. Надеюсь, у меня получилось.

Итоги и перспективы

Человеческий интеллект проявляется в восприятии, зрении, общении на естественном языке, мышлении, принятии решений, сотрудничестве, эмпатии, моделировании и манипулировании окружающей средой, передаче навыков и знаний от одного поколения к другому, обобщении врожденных и приобретенных навыков в новых и неизведанных областях. Искусственный интеллект стремится повторить все особенности человеческого интеллекта. Хотя в настоящее время он может воспроизвести одновременно только один или несколько аспектов человеческого интеллекта. Но даже с учетом этого, искусственный интеллект демонстрирует величайшие достижения, как, например, в моделировании процессов сворачивания белков и предсказания структуры белка — основной структурной части любого живого организма. Перспективы применения этой системы искусственного интеллекта, как и множества других, для понимания природы живых организмов и борьбы со всеми видами заболеваний безграничны.

Вступая в область ИИ, важно не забывать о том, какой конкретно аспект интеллекта разрабатывается или используется: восприятие, зрение, естественный язык, навигация, управление, рассуждения и т. д. Когда мы знаем, о чем конкретно идет речь, можно самим выбирать нужный предмет математики и сосредоточиться на нем. Тогда будет проще выяснить математические методы и инструменты, используемые специалистами в работе над определенным аспектом искусственного интеллекта. Книга построена аналогичным способом: сначала мы определяем тип искусственного интеллекта, смотрим его применение, а затем обсуждаем математику.

В этой главе мы рассмотрели общие вопросы, что такое искусственный интеллект и на что он способен, обозначили его ограничения и тренды, разобрались в том, как он функционирует. Мы также кратко обсудили важные области применения ИИ, выяснили проблемы, с которыми обычно сталкиваются компании, пытающиеся интегрировать его в свои системы, рассмотрели инциденты, происходящие при некачественном внедрении систем, и обозначили математические дисциплины, которые требуются для внедрения искусственного интеллекта.

Следующая глава посвящена данным и подтверждает их тесную связь с ИИ.

Когда мы говорим о данных, речь идет, в том числе, и об их распределении, что сразу же переносит нас в теорию вероятностей и статистику.

ГЛАВА 2

Данные, данные, данные

Возможно, если бы мне было известно, откуда и почему все это взялось, то я бы знала, куда все это движется и зачем.
— Хала

https://t.me/it_boooks/2

Данные являются топливом для большинства систем ИИ. В этой главе мы узнаем, что в основе *искусственного интеллекта восприятия* лежат данные и разработка методов извлечения из них полезной и применимой информации.

Искусственный интеллект восприятия базируется на статистическом обучении на основе данных, когда агент искусственного интеллекта, или машина, воспринимает данные из окружающей среды, а затем раскрывает в них закономерности, позволяющие делать выводы и/или принимать решения.

Искусственный интеллект восприятия отличается от трех других типов искусственного интеллекта.

Искусственный интеллект понимания.

Система ИИ понимает, что изображение, которое она классифицировала как стул, выполняет функцию сидения, а изображение, которое она классифицировала как онкологическое заболевание, означает, что человек болен и нуждается в дальнейшей медицинской помощи, или прочитанный учебник по линейной алгебре можно использовать для извлечения полезной информации из данных.

Искусственный интеллект управления.

Речь идет об управлении физическими компонентами агента ИИ для навигации в пространстве, открывания дверей, подачи кофе и т. д. Значительного прогресса в этом направлении достигла робототехника. Задача заключается в том, чтобы оснастить роботов "мозгами", содержащими в своей структуре искусственные интеллекты восприятия и понимания, подключенные к искусственному интеллекту управления. В идеале искусственный интеллект управления должен обучаться по аналогии с человеком через физическое взаимодействие с окружающей средой, передавая полученную информацию системам восприятия и понимания, которые, в свою очередь, передают команды системам управления агента.

Искусственный интеллект осознания.

Подразумевается, что агент ИИ обладает внутренним опытом, аналогичным человеческому. Поскольку нам пока неизвестно, как математически определить осознание, оставим это понятие за рамками книги.

В идеале настоящий человекоподобный интеллект должен сочетать в себе все четыре аспекта: восприятие, понимание, управление, осознание. В этой и последующих главах мы обсудим искусственный интеллект восприятия. Важно, что ИИ и данные переплелись настолько тесно, что сейчас принято, хотя и ошибочно, пользоваться понятиями "наука о данных" и "искусственный интеллект" как синонимами.

Данные для искусственного интеллекта

В основе многих популярных моделей машинного обучения, в том числе весьма успешных нейросетей, вернувших искусственный интеллект в центр внимания общественности благодаря успехам AlexNet в 2012 году, лежит очень простая математическая задача:

подогнать заданный набор точек данных под соответствующую функцию (отображение входа на выход), которая улавливает важные сигналы в данных и игнорирует помехи, а затем убедиться, что эта функция хорошо работает на новых данных.

И здесь можно споткнуться о подводный камень в самом неожиданном месте.

Гипотезы и признаки.

Нам неизвестны ни исходная функция, породившая данные, ни признаки, от которых она зависит. Мы просто наблюдаем данные, а затем стараемся оценить эту гипотетическую функцию. Функция пытается выяснить, насколько признаки данных важны для прогнозов, классификаций, решений, общих целей. Она также изучает, как эти признаки взаимодействуют между собой для получения наблюдаемых результатов. Одним из важнейших потенциалов ИИ в данном случае является его способность улавливать тончайшие взаимодействия между признаками данных, которые человеку обычно не доступны, поскольку мы прекрасно замечаем сильные признаки, но упускаем при этом более тонкие. Так, можно сказать, что кредитоспособность человека зависит от его ежемесячного дохода, упуская при этом, сколько он тратит ежедневно, к примеру, на поездку до работы, и другие рутинные расходы. Бывают взаимодействия признаков, которые намного проще других, среди которых, например, линейные взаимодействия. Бывают более сложные — нелинейные. С математической точки зрения, независимо от того, являются ли наши взаимодействия признаков простыми (линейными) или сложными (нелинейными), мы все равно преследуем одну и ту же цель — найти гипотетическую функцию, которая подойдет к данным и будет способной делать верные предсказания на основе новых данных. И здесь возникает дополнительная сложность: существует множество гипотетических функций, которые могут *подойти* к одному и тому же набору данных, но как понять, какую выбрать?

Производительность.

Даже вычислив гипотетическую функцию для своих данных, как узнать, насколько хорошо она будет работать на новых и незнакомых данных? Как опре-

делить показатель производительности и как отслеживать его после внедрения в реальный мир? Поскольку данные и сценарии реального мира не являются для нас истиной в последней инстанции, нам достаточно сложно определить, насколько хорошо работает система ИИ и насколько верные, адекватные прогнозы и решения она принимает. Мы не знаем, с чем можно сравнить результаты ее работы. Если бы данные и сценарии реального мира были размечены как истины, то нам просто ничего не останется делать — мы будем знать наверняка, что нам делать в любой ситуации, а на Земле воцарится мир, и мы будем жить долго и счастливо (не совсем верно, но хотелось бы, чтобы все было вот так просто).

Объем.

Практически все в области искусственного интеллекта обладает высокой размерностью! Количество экземпляров данных, наблюдаемых признаков, неизвестных весов, которые подлежат расчетам, может исчисляться миллионами, а количество необходимых вычислительных шагов — миллиардами. Центральное место здесь занимают эффективное хранение, транспортировка, исследование, предварительная обработка, структурирование и вычисления на таких объемах данных. Кроме того, нетривиальной задачей является изучение ландшафтов задействованных высокоразмерных математических функций.

Структура.

Большинство данных, генерируемых в современном мире, являются неструктурированными. Они не организованы в удобные для запросов таблицы, содержащие размеченные поля, например, такие как имя, номер телефона, пол, возраст, почтовый индекс, цены на недвижимость, уровень дохода и т. д. Неструктурированные данные окружают нас повсюду — это сообщения в социальных сетях, активность пользователей, документы Word, PDF-файлы, изображения, аудио- и видеофайлы, данные программ совместной работы, данные о дорожном движении, сейсмические или метеорологические данные, GPS, военные перемещения, электронная почта, мгновенные сообщения, данные мобильных чатов и многое другое. Некоторые из этих примеров, например данные электронной почты, можно считать слабоструктурированными, поскольку электронные письма имеют заголовки, содержащие метаданные сообщения: "От кого", "Кому", "Дата", "Время", "Тема", "Тип содержимого", "Фильтрация спама" и т. д.

Более того, большие объемы ценных данных не доступны в цифровом формате и разбросаны по многочисленным разрозненным базам данных. В качестве примера можно привести исторические военные данные, музейные архивы, больничные карты.

Сегодня наблюдается стремительный переход к глобальной цифровизации, доступной даже в малых поселениях, что стимулирует к более масштабному использованию приложений ИИ. В целом нужную информацию проще извлечь из структурированных и размеченных данных, чем из неструктурированных. Для работы с неструктурированными данными требуются инновационные методы, которые на сегодняшний день служат движущей силой в таких областях, как наука о данных, машинное обучение, искусственный интеллект.

Реальные и имитационные данные

При работе с данными крайне важно понимать разницу между реальными и имитационными (смоделированными) данными. Оба типа данных крайне важны в плане достижений и прогресса человечества.

Реальные данные.

Их собирают в процессе изучения реального мира с помощью измерительных приборов, датчиков, опросов, структурированных форм. Это могут быть медицинские анкеты, телескопы, устройства формирования изображений, веб-сайты, фондовые рынки, контролируемые эксперименты и т. д. Реальные данные часто несовершенны и искажены из-за ошибок в методах и сбоев в измерительных устройствах. С точки зрения математики, *мы не знаем ни точной функции, ни распределения вероятностей, порождающие реальные данные*, но, создавая имитационные модели на основе различных данных, мы можем выдвигать гипотезы. Затем тестируем модели и в конечном счете применяем их в прогнозировании.

Имитационные (смоделированные) данные.

Их получают с помощью *известной* функции или случайной выборки из *известного* распределения вероятностей. Когда у нас имеется известная математическая функция (или *модель*), подставляем числовые значения в нее и получаем точку данных. Можно привести здесь множество примеров: численные решения дифференциальных уравнений, имитирующие всевозможные природные явления во всевозможных масштабах, например турбулентные потоки, сворачивание белков, диффузия тепла, химические реакции, движение планет, разрушение материалов, движение транспорта и даже высококачественная анимация диснеевских фильмов, как, например, моделирование естественного движения воды в мультфильме "Моана" или движение волос Эльзы в мультфильме "Холодное сердце".

В этой главе мы увидим, чем различаются реальные и имитационные данные на двух примерах данных о росте и массе тела человека. В первом примере мы пользуемся двумя реальными наборами данных, содержащими измерения роста и веса реальных людей, скачанными на сайте открытой онлайн базы данных. Во втором примере мы смоделируем собственный набор данных о росте и весе *на основе гипотетической функции*, полагая существование линейной зависимости между весом и ростом человека. Другими словами, на графике зависимости веса от роста мы рассчитываем увидеть прямую (или плоскую) визуальную картину.

Математические модели: линейные и нелинейные

Линейные зависимости моделируют плоскостное строение мира, например одномерные прямые линии, двумерные плоские поверхности (*плоскости*), гиперплоскости высших измерений.

График линейной функции, моделирующей линейную зависимость, всегда плоский и не имеет изгибов. В случае любого плоского объекта, будь это стол, стержень, потолок или множество точек данных, собранных в окрестности прямой линии или плоской поверхности, его представительная функция будет линейной. Все неплоское считается нелинейным, поэтому функции, чьи графики имеют изгибы, являются нелинейными, а точки данных в районе изгибающихся кривых или поверхностей порождены нелинейными функциями.

Записать формулу линейной функции, представляющую собой линейную зависимость выхода функции от *признаков*, или *переменных*, довольно легко. Признаки фигурируют в формуле сами по себе без степеней и корней и не входят в другие функции типа знаменателей дробей, синуса, косинуса, экспоненциальных, логарифмических или других вычислительных функций. Их можно только умножать на *скаляры* (не векторы или матрицы, а действительные и компле́ксные числа), складывать или вычитать друг из друга. Например, функцию, линейно зависящую от трех характеристик x_1, x_2 и x_3, можно записать как:

$$f(x_1, x_2, x_3) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3,$$

где параметры $\omega_0, \omega_1, \omega_2, \omega_3$ — скалярные числа. Параметры или *весá* $\omega_1, \omega_2, \omega_3$ линейно объединяют признаки и после добавления термина смещения ω_0 дают результат $f(x_1, x_2, x_3)$. Другими словами, результат — это итог линейного взаимодействия между признаками x_1, x_2, x_3 плюс смещение.

Мы можем также легко найти формулу нелинейной функции, представляющую собой нелинейную зависимость выхода функции от признаков. В формулу функции могут входить один или несколько признаков с мощностью, отличной от единицы. Они могут либо умножаться или делиться на другие признаки, либо встраиваться в некоторые другие вычислительные функции, такие как синусы, косинусы, экспоненты, логарифмы и т. д. Приведем три примера функций, нелинейно зависящих от трех признаков x_1, x_2, x_3:

$$f(x_1, x_2, x_3) = \omega_0 + \omega_1 \sqrt{x_1} + \omega_2 \frac{x_2}{x_3};$$

$$f(x_1, x_2, x_3) = \omega_0 + \omega_1 x_1^2 + \omega_2 x_2^2 + \omega_3 x_3^2;$$

$$f(x_1, x_2, x_3) = \omega_1 e^{x_1} + \omega_2 e^{x_2} + \omega_3 \cos(x_3).$$

Очевидно, можно придумать самые разные нелинейные функции, причем возможности относительно того, что мы можем делать и какую часть мира мы способны моделировать с помощью нелинейных взаимодействий, безграничны. Фактически успех нейросетей обусловлен способностью улавливать соответствующие *нелинейные* взаимодействия между признаками данных.

На протяжении всего повествования мы будем пользоваться прежними обозначениями и терминологией, что позволит быстро освоить такие термины, как линейная комбинация, весá, признаки, линейные и нелинейные взаимодействия между признаками.

Пример реальных данных

Код на языке Python для исследования данных и рисунки из представленных двух примеров можно посмотреть на странице книги на сайте GitHub (**https://github.com/halanelson/Essential-Math-For-AI**).

Структурированные данные

Нам предстоит работать с двумя наборами данных о росте, весе и поле, представляющими пример *структурированных* наборов данных. Они организованы в виде строк и столбцов. Столбцы содержат характеристики, такие как вес, рост, пол, индекс здоровья и т. д. В строках дается оценка признаков каждого экземпляра данных, в нашем случае — каждого человека. С другой стороны, неструктурированные наборы данных можно представить в виде совокупности аудиофайлов, постов в Facebook, изображений, видеоматериалов.

Мы пользуемся двумя наборами данных для специалистов в области исследования данных, скачанными на сайте Kaggle (**https://www.kaggle.com/**). Данные содержат информацию о росте, весе, поле нескольких человек. Наша задача — выяснить зависимость между ростом и весом человека.

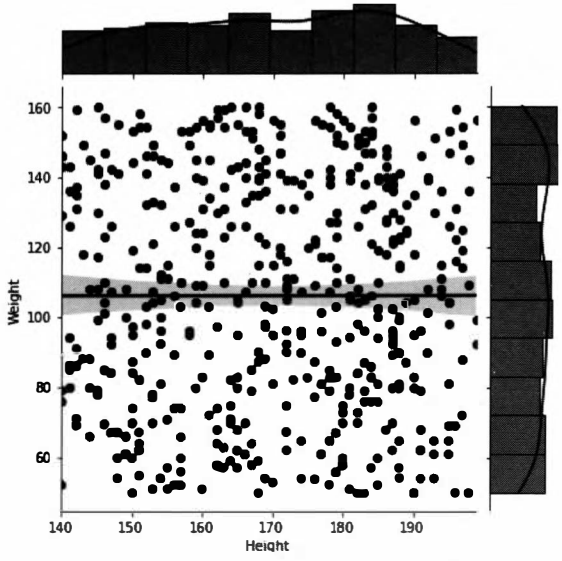


Рис. 2.1. График первого набора данных (**https://oreil.ly/aaSEl**) не демонстрирует зависимость между весом (weight) и ростом (height). Графики, расположенные сверху и справа от диаграммы рассеяния, показывают соответствующие гистограммы и эмпирические распределения данных о росте и весе

С точки зрения математики, мы хотим записать математическую формулу для веса в виде функции одного признака — роста:

$$вес = f(рост),$$

так, чтобы, получив данные о росте *незнакомого человека*, можно было *предсказать* его вес. Безусловно, на вес человека влияет не только рост, но и другие признаки, такие как пол, привычки питания, физическая активность, генетические предрасположенности и т. д. К сожалению, скачанные наборы данных содержат только данные о росте, весе и поле. Поэтому во избежание поисков более подробных наборов данных или сборов новых данных мы будем работать с тем, что есть. Тем более что наш пример всего лишь наглядно демонстрирует разницу между реальными и имитационными данными. Когда стоят более серьезные задачи, мы работаем с более сложными наборами данных с большим числом признаков.

Для первого набора данных (**https://oreil.ly/pxgwe**) строим столбец веса (weight) против столбца роста (height), как показано на рис. 2.1, и, на первый взгляд, вообще не имеем никаких зависимостей!

Аналогичным образом поступаем со вторым набором данных (**https://oreil.ly/8bE36**) и на рис. 2.2 видим явную линейную зависимость. Точки данных как бы сходятся вдоль прямой линии!

Так что же происходит? Почему первый реальный набор данных не показал никакой зависимости между ростом и весом человека, а второй показал? Чтобы ответить на эти вопросы, нужно подробнее присмотреться к данным.

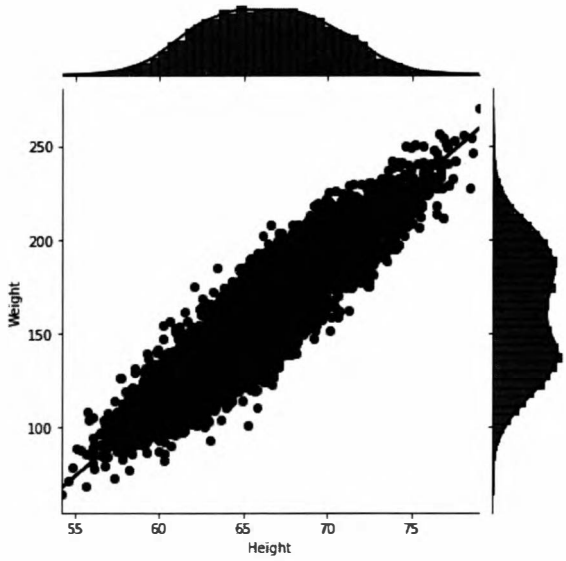


Рис. 2.2. На графике второго набора данных (**https://oreil.ly/rZNBS**) видна линейная зависимость между весом (weight) и ростом (height). Эмпирическое распределение данных по весу показано в правой части рисунка, а эмпирическое распределение данных по росту — вверху. Оба показателя имеют два пика (бимодальные), что свидетельствует о существовании смешанного распределения. По сути, оба набора данных (о росте и весе) можно смоделировать смешением двух нормальных распределений, или гауссовыми смесями, представляющими собой смешение базовых распределений мужских и женских данных. Таким образом, если построить график только для женской или только для мужской группы населения, как показано на рис. 2.6, мы увидим нормальное распределение (колоколообразное) данных о росте и весе

В этом заключается одна из сложностей в работе с реальными данными. Никогда неизвестно, какая функция породила данные и почему они выглядят именно так, а не иначе. Мы изучаем данные, получаем информацию, выявляем закономерности, если они есть, и выдвигаем гипотезу о функции. Затем проверяем свою гипотезу и в случае отличных результатов, соответствующих нашим тщательно проработанным показателям эффективности, мы реализуем ее в реальном мире. На ее основе мы строим прогнозы до тех пор, пока не появятся новые данные и наша гипотеза больше не будет считаться верной, после чего мы исследуем обновленные данные и выдвигаем новую гипотезу.

Такой процесс и цикл обратной связи продолжаются на протяжении всего времени функционирования моделей.

Прежде чем мы перейдем к имитационным данным, объясню, почему первый набор данных не дал ни малейшего представления о зависимости между ростом и весом. При более подробном изучении данных выяснилось, что в их наборе преобладают люди с индексами 4 и 5, означающими, соответственно, ожирение и крайнюю степень ожирения. Поэтому было решено разбить данные по индексам и построить график зависимости веса от роста для людей с одинаковыми индексами. На этот раз на рис. 2.3 можно четко увидеть линейную зависимость, так что загадка разрешилась. Может показаться, что мы обманываем самих себя, опираясь на значения индекса в достижении линейности. Нет — ради исследования данных все делается по-честному.

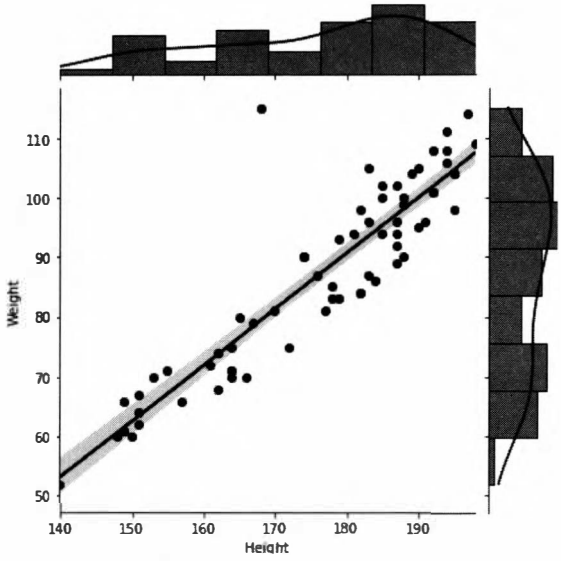


Рис. 2.3. На графике первого набора данных (**https://oreil.ly/0cPnT**) видна линейная закономерность зависимости между весом (weight) и ростом (height) людей с одинаковым значением индекса. На рисунке показана зависимость веса от роста для людей с индексом 3

Теперь можно смело выдвигать гипотезу о том, что вес линейно зависит от роста:

$$вес = \omega_0 + \omega_1 \times рост.$$

Безусловно, остается задача поиска подходящих значений для параметров ω_0 и ω_1. В *главе 3* мы узнаем, как именно это делается. Фактически основная часть работы в машинном обучении, в том числе в глубоком обучении, заключается в *обучении* параметрам ω *из данных*. В нашем простейшем примере нужно обучиться только двум функциям ω, поскольку у нас лишь один признак — рост, и мы *предположили* линейную зависимость после наблюдения линейной модели в реальных данных. В ближайших главах мы обсудим сети глубокого обучения с миллионами параметров ω для обучения и увидим, что математическая структура задачи представляет собой по факту структуру, аналогичную той, о которой говорится в *главе 3*.

Пример имитационных данных

Для примера смоделируем собственный набор данных о росте и весе, чтобы не озадачиваться поисками данных в Интернете или проведением реальных исследований, предполагающих в отдельных случаях работу в лабораторных условиях. Это является важнейшим преимуществом в ситуациях, когда данные либо отсутствуют, либо обходятся слишком дорого. Кроме того, можно тестировать различные сценарии простой заменой чисел в функции и не утруждать себя созданием новых материалов или лабораторий для экспериментов. Моделирование данных весьма ценно тем, что единственное, что для него требуется, — это математическая функция, распределение вероятностей (если мы хотим включить случайность и/или помеху) и компьютер.

Снова предположим линейную зависимость между ростом и весом, для чего будем пользоваться все той же функцией

$$вес = \omega_0 + \omega_1 \times рост.$$

Для симуляции числовых пар *"рост — вес"*, или точек данных, необходимо принять числовые значения параметров ω_0 и ω_1. Не имея представления о правильном выборе параметров ω на основе реальных данных, мы можем лишь делать обоснованные предположения, исходя из контекста задачи, и экспериментировать с различными значениями. Отметим, что для примера с ростом и весом у нас есть реальные данные, которые можно использовать для определения соответствующих значений ω, и мы займемся этим в *главе 3*. Однако в большинстве других сценариев у нас нет реальных данных, поэтому единственным выходом остается экспериментирование с разными численными значениями параметров ω.

Для дальнейших симуляций зададим значения $\omega_0 = -314{,}5$ и $\omega_1 = 7{,}07$, после чего наша функция примет вид:

$$вес = -314{,}5 + 7{,}07 \times рост.$$

Теперь можно генерировать сколько угодно числовых пар *"рост — вес"*. Например, подставляя $рост = 60$ дюймов (~152 см) в формулу функции веса, получаем

вес $= -314{,}5 + 7{,}07 \times 60 = 109{,}7$ фунтов (~50 кг). Таким образом, наша линейная модель ***предсказывает***, что человек ростом 60 дюймов весит 109,7 фунтов, а на графике "рост — вес" можно построить точку данных с координатами (60; 109,7). На рис. 2.4 показано 5000 таких точек данных, полученных отбором 5000 значений для роста от 54 дюймов (~140 см) до 79 дюймов (~2 м) и подстановкой их в функцию веса. График на рис. 2.4 представляет собой идеальную прямую линию без помех и отклонений в имитированных данных, т. к. они не включены в нашу линейную модель.

Отличительной чертой имитационных данных является то, что они делают то же самое, что и породившая их функция (функции). Если мы знаем функцию (называемую *моделью*), применявшуюся в симуляции, и если в наших вычислениях не накопилось слишком много численных ошибок и/или очень больших чисел, выходящих за пределы модели, то мы знаем данные, которые генерирует наша модель, и можем использовать их по своему усмотрению. Для сюрпризов остается не так много места. В нашем примере предложенная функция является линейной, поэтому ее уравнение представляет собой уравнение прямой, и, как видно на рис. 2.4, сгенерированные данные идеально ложатся на нее.

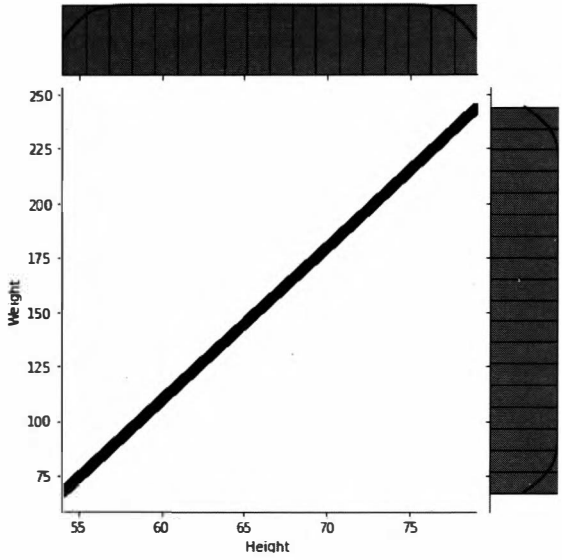


Рис. 2.4. Имитационные данные — 5000 точек (рост, вес), сгенерированных с помощью линейной функции *вес* $= -314{,}5 + 7{,}07 \times$ *рост*

А если мы захотим смоделировать более реалистичные данные о росте и весе? Тогда можно взять значения роста из более реалистичного распределения — колоколообразного нормального распределения. Опять же, мы *знаем* распределение вероятностей, из которого делаем выборку, что отличается от случая с реальными

данными. Отобрав значения роста, подставляем их в линейную модель веса и, если хотим, чтобы наши имитируемые данные были реалистичными, добавляем немного помех. Поскольку помехи имеют случайный характер, необходимо также выбрать распределение вероятностей, откуда будет производиться выборка. Мы снова выбираем колоколообразное нормальное распределение, хотя для моделирования равномерных случайных флуктуаций подошло бы и равномерное распределение. Наша более реалистичная модель "рост — вес" принимает вид:

$$вес = -314{,}5 + 7{,}07 \times рост + помеха.$$

Результат приведен на рис. 2.5.

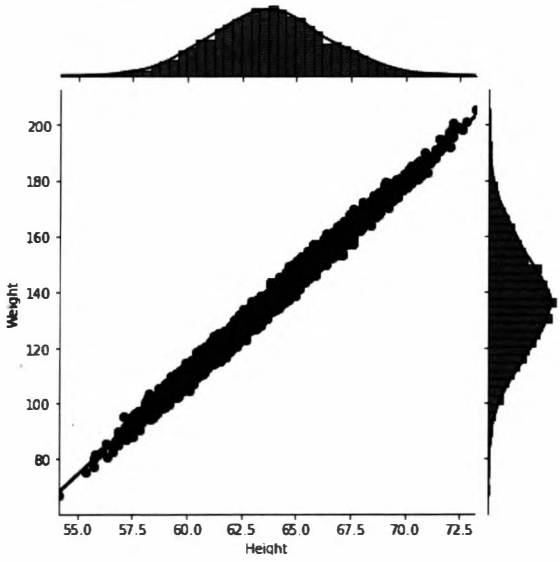


Рис. 2.5. Имитационные данные — 5000 точек (рост, вес), сгенерированных с помощью линейной функции *вес* = –314,5 + 7,07 × *рост*. Точки роста имеют нормальное распределение, и мы добавили нормально распределенную помеху. Отметим, что данные по весу и росту, соответственно, в правой и верхней частях рисунка являются нормально распределенными. Это неудивительно, поскольку в нашей имитации они спроектированы именно таким образом

Сравним имитированные данные о росте и весе (см. рис. 2.5) с реальными данными (рис. 2.6) о росте и весе 5000 женщин из нашего второго набора данных (**https://oreil.ly/rZNBS**). Выглядит вполне неплохо, учитывая, что мы не занимались сбором реальных данных, а для их генерации нам потребовалось всего пять минут работы над кодом! А если мы уделим чуть больше времени на настройку значений ω и параметров нормально распределенных помех (среднего и стандартного отклонения), то получим еще более привлекательный набор имитационных данных. Пока отложим это в сторону, т. к. очень скоро мы сосредоточимся на *обучении* соответствующих значений параметров наших гипотетических моделей.

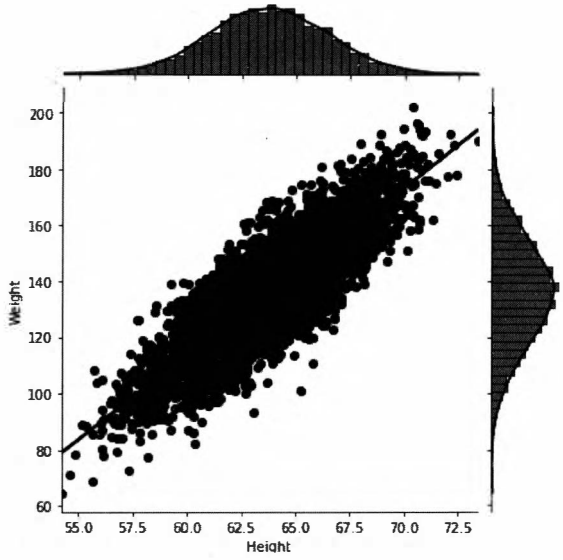


Рис. 2.6. Реальные данные — данные о весе и росте 5000 женщин из второго набора данных (**https://oreil.ly/rZNBS**). Обратите внимание, что данные о весе и росте женщин, соответственно, в правой и верхней частях рисунка являются нормально распределенными. Более подробную информацию можно найти на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**)

Математические модели: симуляции и искусственный язык

Можно в любой момент скорректировать математические модели, делая их более реалистичными. Кому как не нам, разработчикам, решать, что именно мы заложим в них. Часто бывает так, что чем точнее модель имитирует природу, тем больше математических объектов она включает. Поэтому при построении математической модели часто приходится искать компромисс между реалистичностью и простотой, а также доступностью модели для математического анализа и вычислений. Математические модели отличаются друг от друга, и некоторые из них имитируют определенные явления лучше, чем остальные. В поисках естественных форм поведения эти модели непрерывно совершенствуются и развиваются. К счастью, в последние десятилетия мы наблюдаем значительное расширение вычислительных возможностей, которое позволяет создавать и тестировать более сложные и реалистичные математические модели.

Наша природа изобилует мельчайшими подробностями, оставаясь при этом практически неохватной. Взаимодействия в ней можно наблюдать на всех уровнях — от субатомной квантовой области до межгалактических пространств. Человек стремится понять природу, выяснить ее сложнейшие компоненты, раскрыть ее многочисленные взаимодействия и взаимосвязи. И на это есть масса причин. К примеру,

это может быть простой интерес к происхождению жизни и Вселенной или создание новых технологий, совершенствование систем связи, разработка и поиск лекарств от болезней, создание оружия и систем обороны, путешествия к далеким планетам и, возможно, их заселение в будущем. Математические модели позволяют удивительным образом описать природу во всех ее деталях, используя только числа, функции, уравнения и ссылаясь на квантованную случайность через вероятность в условиях неопределенности. Компьютерное моделирование позволяет исследовать и визуализировать простые и сложные формы поведения имитируемых систем или явлений. В свою очередь, результаты компьютерного моделирования помогают усовершенствовать модель, а также получить более глубокие математические знания. Этот невероятный цикл положительной обратной связи превращает математическое и компьютерное моделирование в незаменимый инструмент, который с ростом вычислительных возможностей становится все более эффективным.

Загадка Вселенной заключается в том, что различные явления, происходящие в ней, могут быть точно имитированы с помощью абстрактного языка математики, а чудо человеческого разума заключено в способности изучать и постигать математику, а также создавать мощные устройства для решения самых разных задач. Не менее удивительно и то, что все эти устройства не выполняют, по сути, ничего другого, кроме вычислений или передачи математических данных, а точнее, набора нулей и единиц.

Ярким примером *обобщения* усвоенных знаний является тот факт, что человек способен обобщать свое знание простых чисел вплоть до построения и применения математических моделей природных явлений в самых разных масштабах, что является отличительной чертой человеческого интеллекта. В области искусственного интеллекта общей целью как общего искусственного интеллекта (человекоподобного и сверхинтеллекта), так и слабого (ориентированного на решение конкретных задач) является *обобщение* — способность агента ИИ обобщать полученные знания относительно новых и неизвестных ситуаций. В *главе 3* мы разберем этот принцип для слабого, ориентированного на конкретную задачу искусственного интеллекта, когда *агент ИИ учится на данных, а затем дает верные предсказания для новых и неизвестных данных*.

Искусственный интеллект взаимодействует с математическими моделями и симуляторами тремя способами.

Математические модели и симуляторы создают данные, на которых обучаются системы искусственного интеллекта.

Некоторые полагают, что эталоном ИИ являются самоуправляемые автомобили. Нельзя допустить, чтобы пробные образцы "умных" автомобилей падали с обрывов, сбивали пешеходов или врезались в новые рабочие зоны, пока система ИИ автомобиля не научится понимать, что этих неблагоприятных событий следует избегать. В этом случае обучение на имитационных данных становится крайне важным, т. к. с помощью симуляторов можно обучить автомобиль всевозможным виртуальным опасным ситуациям, прежде чем он появится на ре-

альных дорогах. Аналогичным образом имитационные данные можно успешно применять в обучении систем ИИ марсоходов, поиска лекарственных формул, разработки материалов, прогнозирования погоды, в области авиации, военной подготовки и т. д.

Искусственный интеллект совершенствует существующие математические модели и симуляторы.

Искусственный интеллект обладает огромным потенциалом в областях, которые традиционно считаются были сложными, ограничивающими математические модели и симуляторы, куда можно отнести поиск верных значений для используемых в моделях соответствующих параметров, распределений вероятностей, форм и размеров сеток при дискретизации уравнений (тонкие сетки передают мелкие детали и тонкое поведение на различных пространственных и временны́х масштабах), а также масштабирование вычислительных методов на более длительные временны́е периоды или на большие области сложной формы. На математическое моделирование и симуляцию в значительной степени опираются такие области, как навигация, авиация, финансы, материаловедение, гидродинамика, исследование операций, молекулярные и ядерные науки, науки об атмосфере и океане, астрофизика, физическая безопасность, кибербезопасность и многие другие. Интеграция возможностей ИИ в эти области постепенно начинает приносить первые плоды. В последующих главах мы познакомимся с примерами имитационного моделирования с использованием ИИ.

Искусственный интеллект сам по себе является математической моделью и имитацией.

Одним из главных притязаний ИИ является *вычислительное* воспроизведение человеческого интеллекта. Успешные системы машинного обучения, например, нейросети со всеми своими архитектурами и вариациями, представляют собой математические модели, нацеленные на имитацию задач, которые человек связывает с интеллектом: зрение, распознавание и обобщение образов, общение на естественном языке, логические рассуждения. С интеллектом также связаны понимание, эмоциональный опыт, эмпатия, сотрудничество, которые в значительной степени способствовали успеху и процветанию человечества, а значит, необходимо найти и способы их воспроизведения, если мы действительно хотим создать глобальный искусственный интеллект и вместе с тем еще глубже понять природу ума и работу человеческого мозга. Работа в этом направлении уже ведется. Важно помнить, что во всех подобных областях машины занимаются *вычислениями*. Машины *вычисляют* значения документов для обработки естественного языка, комбинируют и *вычисляют* пикселы цифровых изображений для компьютерного зрения, преобразуют аудиосигналы в векторы чисел, *вычисляют* новые аудиосигналы для взаимодействия человека с машиной и т. д. Отсюда легко понять, что программный ИИ представляет собой одну большую математическую модель и имитацию. Это будет еще более очевидным по ходу нашего дальнейшего повествования.

Как получить данные

Когда я впервые решила заняться искусственным интеллектом, у меня было желание использовать свои математические знания для решения реальных проблем, волновавших меня на тот момент. Я выросла в охваченной войной стране и видела, как внезапно могло возникнуть множество проблем, но затем они так или иначе разрешались, пока, наконец, полностью не исчезали — либо прямым устранением, либо благодаря тому, что человеческие сети приспосабливались к ним, устанавливая совершенно новое (неустойчивое) равновесие. К общим проблемам во время войны можно отнести и внезапные массовые нарушения различных цепочек поставок, и разрушение значительной части энергосистемы, и паралич всей дорожной сети в результате прицельной бомбардировки мостов, и возникновение террористических сетей, черных рынков, торговли людьми, инфляции, бедности. Возможности математики в решении таких задач, включая тактику и стратегию военных действий, неограниченны. Оказавшись в безопасности в США, получив степень доктора математических наук и работу в университете, я попросила ряд компаний, государственных учреждений и военных ведомств предоставить мне для работы реальные проекты с реальными данными. Причем помощь в поиске решений их проблем предлагалась абсолютно бесплатно. Но я даже не предполагала, насколько сложно получить реальные данные. На пути к ним возникает множество препятствий в виде нормативных актов, вопросов конфиденциальности, институциональных наблюдательных советов и пр. Но даже преодолев всю эту волокиту, приходится практически умолять компании поделиться реальными данными, поскольку те стараются не раскрывать свои данные посторонним, хотя в большинстве случае понимают, что используют их не лучшим образом. Как выяснилось, мой опыт не уникален. С тем же самым сталкиваются и многие другие специалисты в нашей области.

Ни в коем случае я никого не отговариваю обращаться за реальными данными для обучения систем ИИ. Главное — не удивляться и не расстраиваться, когда упретесь в колебания, нерешительность и даже сопротивление хозяев нужных вам данных. Не отступать — и все получится!

Иногда нужные данные доступны в Интернете. Наборы данных для построения простых моделей, описанных в этой главе, доступны на сайте Kaggle (**https://www.kaggle.com/**). Существуют и другие крупные публичные хранилища данных, но мы не будем на них останавливаться — достаточно ввести в строке поиска Google ключевые слова типа "лучшие хранилища данных", и мы получим множество ссылок на них. Некоторые хранилища ориентированы на компьютерное зрение, другие — на обработку естественного языка, генерацию и расшифровку звука, научные исследования и т. д.

Поиск данных в Интернете является обычной практикой, при этом важно соблюдать правила посещения сайтов. Кроме того, необходимо уметь грамотно работать с информацией в Интернете (говорят, что разница между специалистами по изучению данных и статистиками заключается в том, что первые умеют взламывать сайты). В некоторых случаях для посещения интернет-ресурса требуется письменное

разрешение. Например, при изучении поведения пользователей социальных сетей или сетей сотрудничества в таких социальных и профессиональных сетях, как Facebook, Instagram[1], YouTube, Flickr, LinkedIn и других, на предмет статистики по учетным записям пользователей, в частности, по количеству друзей или связей, лайков, комментариев, активности на сайте. В итоге мы получаем колоссальные наборы данных с сотнями тысяч записей, на основе которых можно проводить вычисления.

Для того чтобы получить интуитивное представление о том, как данные интегрируются в ИИ, какого типа данные поступают в различные системы, и при этом не перегружать себя всей доступной информацией и данными, нужно выработать привычку изучать доступные наборы данных, на которых обучались успешные системы ИИ. Их необязательно скачивать и использовать в работе.

Достаточно просмотреть набор данных, узнать их метаданные, признаки, разметки (если таковые имеются) и прочие характеристики и можно уверенно приступать к работе с ними. Например, нейросеть WaveNet (2016 г., **https://oreil.ly/TI5N8**) компании DeepMind, которую мы обсуждаем в *главе 7*, генерирует необработанный машинный аудиосигнал с реалистично звучащими человеческими голосами или приятными музыкальными композициями. Она решает такие задачи, как преобразование текста в аудио с естественным звучанием человеческого голоса и даже с голосом конкретного человека, если сеть обусловлена его голосом. Математический смысл обусловленности мы поймем в ходе обсуждения WaveNet в *главе 7*. Пока ее можно считать ограничением, искусственно наложенным на задачу так, чтобы результаты ограничивались определенным набором выходов. Так на каких же данных обучалась нейросеть WaveNet? Для генерации многоголосного аудио, не обусловленного текстом, WaveNet обучалась на наборе данных, состоящем из 44 часов аудио с участием 109 различных дикторов, например English Multispeaker Corpus from CSTR Voice Cloning Toolkit (2012 г., **https://oreil.ly/lvLPX**). Для преобразования текста в речь — на наборах данных североамериканского английского языка (North American English data set, **https://oreil.ly/rY4qS**), содержащих 24 часа речевых данных, и мандаринского китайского языка (Mandarin Chinese data set, **https://oreil.ly/FGcrO**), содержащих 34,8 часа речевых данных. Для генерации музыки — на наборе данных YouTube Piano data set (**https://oreil.ly/Jwwxm**), содержащем 60 часов сольной фортепианной музыки, полученной из видеофайлов с YouTube, и наборе данных MagnaTagATune (2009 г., **https://oreil.ly/7fnPa**), содержащем порядка 200 часов музыкальных аудиоданных, в которых каждый 29-секундный фрагмент размечен 188 тегами, описывающими жанр, инструментарий, темп, громкость, настроение и др. Размеченные данные крайне важны для систем искусственного интеллекта, т. к. они служат базовой истиной, по которой оцениваются результаты работы гипотетической функции. Мы обсудим это в следующих разделах.

[1] Деятельность Instagram признана экстремистской на территории Российской Федерации, а сама социальная сеть заблокирована. — *Прим. ред.*

А как насчет известной классификации изображений (для компьютерного зрения) AlexNet (2012)? На каких данных обучалась и тестировалась ее сверточная нейросеть? AlexNet обучалась на наборе данных ImageNet (**https://oreil.ly/Yq1pJ**), в которых содержатся миллионы изображений (взятых из Интернета), размеченные (реальными людьми в рамках краудсорсинга) на тысячи классов.

Обратите внимание, что все перечисленные примеры — это примеры неструктурированных данных.

Если данных, на которых обучалась та или иная система, нет в открытом доступе, можно найти публикации, посвященные этой системе, или ее описание, и попробовать узнать там, как были получены нужные данные. Важно, что даже сам факт поиска этой информации может многому научить.

Прежде чем мы приступим к математике, подытожим и отметим для себя следующее:

- системам искусственного интеллекта требуются цифровые данные;
- нужные данные не так просто получить;
- в мире происходит процесс цифровизации.

Словарь распределений данных, вероятностей, статистики

Когда мы вступаем в новую сферу деятельности, мы начинаем с изучения свойственной ей лексики. Точно так же, как и в изучении иностранного языка. И здесь есть разница в обучении — сидеть в душных классах и мучаться, заучивая новые слова, или же переехать в страну, где говорят на этом языке, и окунуться в живую речь. И, возможно, вы никогда не знали, что означает "бонжур" по-французски. Но, оказавшись во Франции, вы часто слышите это слово в общении людей между собой и тоже пробуете применять его. Правда, не всегда удается попасть в нужный контекст, например, вечером, когда принято говорить "бонсуар", т. е. добрый вечер, а не "бонжур" — добрый день. Но чем дольше вы будете жить во Франции, тем точнее будете использовать правильную лексику в правильном контексте.

Еще одно преимущество быстрого пополнения словарного запаса, не вникая в детали, заключается в том, что в различных областях одни и те же понятия обозначаются разными терминами вследствие массового конфликта между словарями. В итоге это ведет к путанице и, как следствие, образованию *языковых барьеров*. После знакомства с общепринятой лексикой мы поймем, что, так или иначе, уже знаем эти понятия, только теперь они по-другому называются.

Для работы с приложениями ИИ потребуется совсем немного словарных терминов из теории вероятностей и статистики. Каждому новому термину в тексте дается определение, но отметим, что целью теории вероятностей являются детерминированные утверждения о случайных или стохастических величинах и событиях, поскольку люди ненавидят неопределенность и любят, чтобы их мир был контролируемым и предсказуемым. Всякий раз, когда вы будете читать об искусственном интеллек-

те, машинном обучении или науке о данных, обратите внимание на формулировки из области теории вероятностей и статистики. Опять же, необязательно знать все эти определения — просто обратите внимание на термины, которые мы обсуждаем в следующих разделах, и порядок их появления в тексте.

Случайные величины

Все начинается со *случайных величин*. Математики обсуждают функции в режиме нон-стоп. Функции имеют определенные, или детерминированные, результаты. Оценивая функцию, мы точно знаем, какое значение она вернет. Функция x^2 со значением 3 обязательно даст $3^2 = 9$. У случайных величин, в свою очередь, не бывает детерминированных результатов. Они носят неопределенный, непредсказуемый или стохастический характер. Когда мы обращаемся со случайной переменной, мы не знаем, какое значение она вернет, пока не получим результат. Поскольку мы больше не можем стремиться к определенности, вместо этого попробуем выяснить, с какой вероятностью можно получить тот или иной результат. Например, когда мы бросаем кубик, можно с уверенностью сказать, что шанс выпадения 4 равен 1/6 при условии, что кубик не деформирован. Пока кубик не брошен, невозможно заранее указать результат. В противном случае закрылись бы все казино, а в финансовом секторе расформировались бы подразделения предсказательной аналитики и управления рисками. Как и детерминированные функции, случайная величина возвращает результат из дискретного множества (дискретная случайная величина) или из континуума (непрерывная случайная величина). Ключевое различие между случайной величиной и функцией — отличие случайности от определенного результата.

Распределение вероятностей

После случайных величин введем определения *функции плотности вероятности* для непрерывных случайных величин и *функции массы вероятности* для дискретных случайных величин. Во избежание путаницы назовем и то и другое *распределениями*. В большинстве случаев можно понять из контекста, представляет ли распределение дискретную или непрерывную случайную величину.

Пользуясь данной терминологией, мы иногда говорим, что одна случайная величина, непрерывная или дискретная, *отбирается* из распределения вероятностей, а несколько случайных величин — из *совместного распределения вероятностей*. На практике редко бывает так что мы знаем полное совместное распределение вероятностей всех случайных величин, участвующих в наших данных. Но в отдельных случаях, когда это так или мы можем *узнать это из данных*, у нас появляется мощный инструмент.

Маргинальные (частные) вероятности

Маргинальные, или *частные, распределения вероятностей* буквально заходят на границы совместного распределения вероятностей (если представить совместное

распределение вероятностей в виде таблицы, в которую внесены вероятности всех совместных состояний участвующих переменных; см., например, первую таблицу на странице Википедии (**https://oreil.ly/11WiO**)). В данной ситуации нам повезло — у нас есть доступ к полному совместному распределению вероятностей нескольких случайных величин, а нас интересует распределение вероятностей только одной или нескольких из них.

Маргинальные распределения вероятностей можно найти, применив, например, *правило суммы* вероятностей:

$$p(x) = \sum_{y \in \text{ все состояния}} p(x, y).$$

Равномерное и нормальное распределения

Самыми известными непрерывными распределениями являются *равномерное* и *нормальное распределения*, поэтому начнем с них. Нормальное распределение тесно связано с фундаментальной *центральной предельной теоремой* из теории вероятностей. Существует множество других важных распределений, в которых представлены различные случайные величины, участвующие в наших данных, но сейчас обойдемся без них и отложим, пока в них не возникнет необходимость.

Условные вероятности и теорема Байеса

Как только речь заходит о множестве случайных величин (например, данные о поле, росте, весе, индексе здоровья), что случается довольно часто, мы вводим такие понятия, как *условные вероятности*, *правило* или *теорему Байеса*, *правило произведения или цепное правило* для условных вероятностей, а также *независимых и условно независимых случайных величин* (когда известное значение одной из них не меняет вероятности другой).

Условные вероятности и совместные распределения

Условные вероятности и совместные распределения включают в себя множество случайных величин, поэтому вполне разумно, что они как-то связаны друг с другом. Если разрезать график совместного распределения вероятностей (при зафиксированном значении одной переменной), то получится условное распределение вероятностей (см. рис. 2.7).

Правило Байеса и совместное распределение вероятностей

Очень важно иметь в виду следующее: если у нас есть доступ к полному совместному распределению вероятностей всех интересующих нас множеств случайных величин, то правило Байеса нам не понадобится. Другими словами, правило Байеса помогает вычислить нужные условные вероятности, *когда у нас нет доступа* к полному совместному распределению вероятностей задействованных случайных величин.

Предварительное распределение, последующее распределение, функция правдоподобия

С точки зрения математической логики, можно дать определение условной вероятности и продолжать спокойно заниматься вычислениями. Но на практике все поразному называют различные условные вероятности в зависимости от того, что берется за основу — наблюдаемые данные или веса (параметры), которые еще предстоит определить. Введем новые термины из нашего словаря: *предшествующее распределение* (общее распределение вероятностей весов нашей модели до наблюдения каких-либо данных), *последующее распределение* (распределение вероятностей весов с учетом наблюдаемых данных), *функция правдоподобия* (функция, кодирующая вероятность наблюдения точки данных при определенном распределении весов). Все они связаны между собой правилом Байеса и совместным распределением.

Функция, а не распределение правдоподобия

Мы называем вероятность функцией, а не распределением, поскольку распределения вероятностей *должны* складываться в единицу (или интегрироваться в единицу, когда мы имеем дело с непрерывными случайными величинами), в то время как функция вероятности необязательно складывается в единицу (или интегрируется в единицу, когда речь идет о непрерывных случайных величинах).

Смеси распределений

Можно смешивать распределения вероятностей и получать *смеси распределений*. Большую известность обрели *гауссовы смеси*. В качестве примера гауссовой смеси можно привести данные о росте, в которых присутствуют показатели и мужчин, и женщин.

Суммы и произведения случайных величин

Можно складывать и умножать случайные величины, взятые из простых распределений, и получать новые случайные величины с более сложными распределениями, представляющими более сложные случайные события. И здесь возникает вполне резонный вопрос: *а что такое распределение суммы случайных величин и произведения случайных величин?*

Использование графов для представления совместных распределений вероятностей

И наконец, для эффективного разложения совместных распределений вероятностей мы пользуемся представлениями (диаграммами) направленных и неориентированных графов, что значительно удешевляет и упрощает наши вычисления.

Ожидание, среднее значение, вариация, неопределенность

Центральное место в теории вероятностей, статистике, науке о данных занимают четыре величины: *математическое ожидание* и *среднее значение*, количественно характеризующие среднее значение, а также *дисперсия случайной величины* и *стандартное отклонение*, количественно характеризующие разброс вокруг среднего значения и, следовательно, кодирующие неопределенность. Наша цель заключается в снижении неопределенности за счет контроля над дисперсией. Чем больше дисперсия, тем больше ошибка, которую мы можем допустить при использовании среднего значения в прогнозировании. Поэтому в процессе изучения определенной области часто замечаешь, что математические утверждения, неравенства, теоремы в большинстве случаев предполагают определенный контроль над ожиданием и дисперсией любых величин, несущих в себе фактор случайности.

В случаях одной случайной величины с соответствующим распределением вероятностей, мы вычисляем *ожидание* (ожидаемое среднее значение результата), *дисперсию случайной величины* (ожидаемое квадратичное расстояние от ожидаемого среднего значения), *стандартное отклонение* (ожидаемое расстояние от среднего значения). Когда у нас имеются предварительно отобранные или исследованные данные, например, в нашем случае с данными о росте и весе, мы вычисляем *выборочное среднее* (среднее значение), дисперсию случайной величины (среднее квадратичное расстояние от среднего), стандартное отклонение (среднее расстояние от среднего, что позволяет определить разброс вокруг среднего). Таким образом, если нужные данные еще не отобраны и не исследовались, мы рассуждаем о них на языке *ожиданий*, но, когда у нас есть исследованная или измеренная выборка, мы вычисляем ее *статистику*. Естественно, нам интересно, насколько наши предположения расходятся с рассчитанной статистикой наблюдаемых данных и что произойдет в предельном (но идеалистическом) случае, когда у нас, действительно, появится возможность изучить данные всего населения. На этот вопрос отвечает *закон больших чисел*, согласно которому в таком предельном случае (когда размер выборки стремится к бесконечности) наше ожидание совпадает со средним значением выборки.

Ковариация и корреляция

При наличии двух или более случайных величин мы вычисляем *ковариацию*, *корреляцию*, *ковариационную матрицу*. Это происходит в тех случаях, когда линейная алгебра с ее языком векторов, матриц, матричных разложений (собственных и сингулярных) входит в область вероятности и статистики. Дисперсия каждой случайной величины располагается на диагонали ковариационной матрицы, а ковариации каждой возможной пары — вне диагонали. Ковариационная матрица является симметричной. В процессе ее диагонализации с помощью стандартных методов линейной алгебры происходит *некоррелированность* участвующих случайных величин.

А пока сделаем паузу и убедимся, что понимаем разницу между независимостью и нулевой ковариацией. С помощью ковариации и корреляции можно найти линейную зависимость между двумя случайными величинами. Корреляция работает с *нормализованными* случайными величинами, так что линейную зависимость можно обнаружить даже в тех случаях, когда случайные величины или измерения данных имеют совершенно разные масштабы. Для нормализованной величины масштаб не имеет значения — будь то 1 000 000 или 0,001. Ковариация работает с ненормированными случайными величинами. Но в жизни не все так линейно. Независимость превыше нулевой ковариации.

Марковский процесс

Марковские процессы играют довольно важную роль в парадигме обучения с подкреплением в искусственном интеллекте. Они характеризуются всеми возможными состояниями системы; набором всевозможных действий, которые способен выполнять агент (движения влево, вправо и т. д.); матрицей, содержащей вероятности перехода между состояниями; распределением вероятностей относительно состояний, в которые перейдет агент после выполнения того или иного действия; функцией вознаграждения, которую мы хотим максимизировать. В качестве примеров из области ИИ можно привести настольные игры и "умный" термостат типа Nest. Мы обсудим их в *главе 11*.

Нормализация, масштабирование, стандартизация случайной переменной или набора данных

Это один из множества случаев, когда возникает конфликт между словарями. Такие термины, как "нормализация", "масштабирование", "стандартизация", встречаются в различных контекстах как синонимы. При этом цель остается прежней: вычесть число (сдвиг) из данных или из всех возможных выходов случайной переменной, после чего разделить на постоянное число (масштаб). Вычитание среднего значения выборки данных (или ожидания случайной величины) и деление на их стандартное отклонение дает новые *стандартизированные* или *нормализованные* значения данных (или новую стандартизированную или нормализованную случайную величину) со средним значением, равным нулю (или ожиданием равным нулю), и стандартным отклонением, равным единице. Вместе с тем вычитание минимального значения и деление на диапазон (максимальное значение минус минимальное значение) дают новые значения данных или новую случайную величину, выходы которой лежат в диапазоне от нуля до единицы. Иногда речь заходит о нормализации векторов чисел. В таких случаях имеется в виду, что, разделив каждое число в нашем векторе на его длину, мы получаем новый вектор единичной длины. Итак, независимо от того, о чем говорится — нормализации, масштабировании или стандартизации набора чисел, речь идет о попытках контролировать значения этих чисел, центрировать их вокруг нуля и/или ограничить их разброс до значения меньше или равного единице, сохраняя свойственную их изменчивость.

Общие примеры

Математики часто иллюстрируют понятия вероятности такими примерами, как подбрасывание монеты, бросание игральных костей, вытягивание шаров из урны или карты из колоды, прибытие поездов на станции, звонки клиентов на горячую линию, переход на сайт или рекламную страницу по ссылке в Интернете, болезни и симптомы, уголовные процессы и доказательства, а также время до наступления того или иного события, например сбоя машины. И это неудивительно, т. к. перечисленные примеры можно легко обобщить на многие другие реальные ситуации.

Помимо теории вероятностей, небольшое количество терминов и функций заимствовано из статистической механики (например, *функция разбиения*) и теории информации (например, отношение "сигнал/помеха", энтропия, *функция перекрестной энтропии*).

Мы обсудим их в дальнейшем.

Непрерывные и дискретные распределения (плотность *vs* масса)

Обсуждая тему непрерывных распределений, принято говорить о *приблизительном*, а не *точном* значении точки наблюдаемых или отобранных данных. Фактически вероятность изучения точного значения в этом случае сводится к нулю.

Когда числа образуют континуум, не существует дискретного разделения между одним и следующим значениями. Действительные числа имеют бесконечную точность. Например, когда мы измеряем рост мужчины и получаем 6 футов, нам неизвестно, насколько точен результат: скажем, 6,00000000785 футов или 5,9999111134255 футов. Поэтому правильнее будет задать интервал наблюдения в районе 6 футов, например $5{,}95 < рост < 6{,}05$, а затем количественно оценить вероятность того, что значение роста окажется в диапазоне между 5,95 и 6,05 футами.

В случае дискретных случайных величин такой проблемы не возникает, поскольку одно возможное значение легко отделяется от другого. Например, когда мы бросаем кубик, возможными значениями являются 1, 2, 3, 4, 5, 6. Поэтому можно уверенно утверждать, что вероятность точного выпадения 5 равна 1/6. Более того, дискретная случайная величина может показывать и нечисловые результаты. К примеру, при подбрасывании монеты возможными значениями являются орел или решка. Непрерывная случайная величина имеет только числовые результаты.

Исходя из этих соображений, мы всегда определяем функцию *плотности* вероятности непрерывной случайной величины, а не функцию *массы* вероятности, как в случае с дискретными случайными величинами. Плотность определяет, сколько вещества присутствует в заданной длине, площади, объеме пространства (в зависимости измерения, в котором мы работаем). Для того чтобы найти массу вещества в заданной области, нужно умножить плотность на длину, площадь или объем этой области. В случаях бесконечно малой области, для того чтобы определить массу в ней,

необходимо выполнить интегрирование по всей области, поскольку интеграл можно представить как сумму по бесконечному множеству бесконечно малых областей.

Эти идеи, а также их математическая формализация обсуждаются более подробно в *главе 11*.

Пока подчеркнем следующее.

- В случаях когда имеется только одна непрерывная случайная величина, например рост мужчин в определенной популяции, представляем ее распределение вероятностей с помощью одномерной функции плотности вероятности $f(x_1)$. Для того чтобы определить вероятность попадания значения роста в интервал $5{,}95 < рост < 6{,}05$, интегрируем функцию плотности вероятности $f(x_1)$ в интервале (5,95; 6,05) и записываем:

$$P(5{,}95 < рост < 6{,}05) = \int_{5{,}95}^{6{,}05} f(x_1)\,dx_1.$$

- В случае двух непрерывных случайных величин, например рост и вес мужчин в определенной популяции или значения настоящего и измеренного роста человека (куда обычно включены случайные помехи), их совместное распределение вероятностей можно представить с помощью двумерной функции плотности вероятности $f(x_1, x_2)$. Именно поэтому для поиска совместной вероятности попадания значения роста в пределы $5{,}95 < рост < 6{,}05$, а веса — в пределы $160 < вес < 175$, *интегрируем дважды* совместную функцию плотности вероятности $f(x_1, x_2)$, полагая, что знаем формулу для $f(x_1, x_2)$ в интервалах (5,95; 6,05) и (160; 175), и записываем:

$$P(5{,}95 < рост < 6{,}05,\ 160 < вес < 175) = \int_{160}^{175}\int_{5{,}95}^{6{,}05} f(x_1, x_2)\,dx_1 dx_2.$$

- В случаях когда имеется более двух непрерывных случайных величин, представляем их совместное распределение с помощью функции плотности вероятности более высокой размерности. Например, при наличии данных о росте, весе, артериальном давлении мужчин в определенной популяции мы применяем трехмерную совместную функцию распределения вероятностей $f(x_1, x_2, x_3)$. Как и в предыдущих рассуждениях, для поиска совместной вероятности попадания значения первой случайной величины в интервал $a < x_1 < b$, второй — в интервал $c < x_2 < d$, а третьей — в интервал $e < x_3 < f$, *интегрируем трижды* совместную функцию плотности вероятности в интервалах (a; b), (c; d), (e; f) и записываем:

$$P(a < x_1 < b, c < x_2 < d, e < x_3 < f) = \int_{e}^{f}\int_{c}^{d}\int_{a}^{b} f(x_1, x_2, x_3)\,dx_1 dx_2 dx_3.$$

Даже после определения функции плотности вероятности для непрерывной случайной величины не все наши опасения (что все математически сойдется) исчеза-

ют. Это происходит из-за все той же бесконечной точности действительных чисел. Если допустить, что *все наборы* обладают вероятностью, то возникают парадоксы в том смысле, что можно построить непересекающиеся множества (например, множества в виде фрактала или множества, образованные путем преобразования множества рациональных чисел), чьи вероятности в сумме дают больше единицы! Следует признать, что такие множества являются патологическими, и их нужно аккуратно создавать вручную, обладая массой свободного времени. Тем не менее они существуют и порождают парадоксы. В таких случаях на помощь приходит математическая *теория меры*, которая позволяет работать с функциями плотности вероятности, не сталкиваясь с парадоксами. Она определяет множества нулевой меры (которые вообще не занимают объем в рабочем пространстве), а затем выдает достаточное количество теорем, позволяющих проводить вычисления *практически повсеместно*, за исключением множеств нулевой меры. Для наших целей этого более чем достаточно.

Сила совместной функции плотности вероятности

Наличие доступа к совместному распределению вероятностей многих случайных величин — штука сильная, но достаточно редкая. Это объясняется тем, что совместное распределение вероятностей содержит в себе распределение вероятностей каждой отдельной случайной величины (маргинальные распределения) с учетом всех возможных сочетаний (и условных вероятностей), возникающих между этими случайными величинами. Это аналогично тому, как если бы мы взглянули на город с высоты птичьего полета, а не изнутри, где можно разглядеть лишь ближайшие улицы и перекрестки.

В случае независимых случайных величин совместное распределение представляет собой произведение каждого из их индивидуальных распределений. Если же случайные величины не являются независимыми, например рост и вес человека или измеренный рост (включающий помехи измерений) и настоящий рост человека (не включающий помехи), то получить совместное распределение гораздо сложнее и дороже в плане хранения данных. Необходимо хранить значение каждого сочетания двух или более переменных. Такой экспоненциальный рост требований к хранению данных (а также к вычислениям и пространству поиска) по мере увеличения числа зависимых случайных переменных — одно из воплощений пресловутого *проклятия размерности*.

При *"нарезке"* совместной функции плотности вероятности, скажем, $f(x_1, x_2)$, т. е. определении точного значения одной из случайных величин (или нескольких в более высоких размерностях), получается распределение, пропорциональное *распределению апостериорных вероятностей* (распределение вероятностей параметров модели с учетом измерений), т. е. то, что мы стараемся найти в большинстве случаев. Например, если нарезать $f(x_1, x_2)$ при $x_1 = a$, получится $f(a, x_2)$, что пропорционально распределению вероятностей $f(x_2 \mid x_1 = a)$ (рис. 2.7).

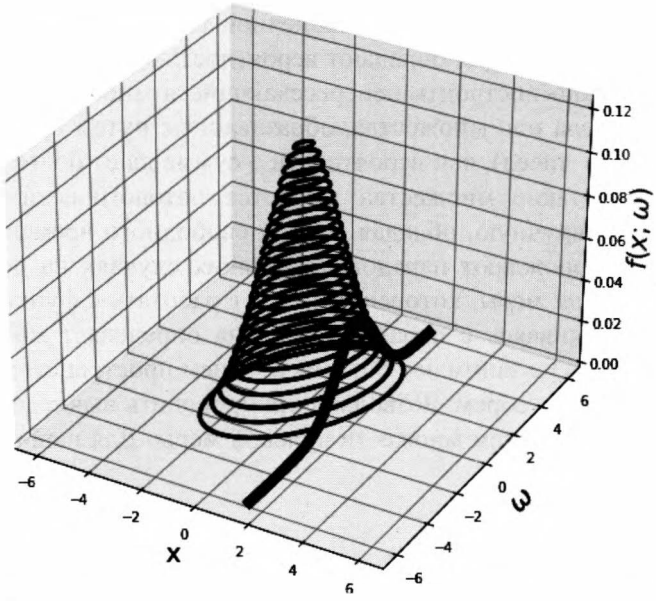


Рис. 2.7. Нарезка совместного распределения вероятностей

Еще раз отметим, что это происходит лишь в том исключительном случае, когда нам известно совместное распределение вероятностей, во всех остальных — применяем правило Байеса и получаем то же самое распределение апостериорных вероятностей (с помощью априорного распределения и функции правдоподобия).

В некоторых приложениях система ИИ обучается совместному распределению вероятностей, разделив его на произведение условных вероятностей по правилу произведения вероятностей. После обучения совместному распределению она делает выборку из него и генерирует новые интересные данные. Подобным образом функционирует нейросеть WaveNet компании DeepMind в процессе генерации необработанного звука.

В следующих разделах представлены самые распространенные в приложениях искусственного интеллекта распределения вероятностей.

Начнем с двух широко распространенных непрерывных распределений — *равномерного* и *нормального* (также известного как *гауссово распределение*). Скопировать рисунки и получить более подробную информацию можно в Jupyter Notebook (**https://github.com/halanelson/Essential-Math-For-AI**).

Распределение данных: равномерное распределение

Для интуитивного понимания равномерного распределения приведем пример *неравномерного* распределения, с которым мы уже познакомились ранее в этой главе. В наших реальных наборах данных о росте и весе мы не можем пользоваться

равномерным распределением для моделирования данных о росте. То же самое относится и к моделированию данных о весе. Причина заключается в том, что рост и вес человека распределены неравномерно. В общей массе населения вероятность встретить человека высокого роста около 7 футов (2,13 метра) не *равнозначна* вероятности повстречать человека среднего роста в районе 5 футов 6 дюймов (1,7 метра).

Равномерное распределение моделирует только равномерно распределенные данные. Когда имеется интервал (x_{min}, x_{max}), содержащий все значения данных в континууме между x_{min} и x_{max}, при этом данные равномерно распределены в нем, вероятность наблюдения точки данных вблизи любого конкретного значения одинакова для всех значений в этом интервале. То есть в интервале (0; 1) вероятность выбора точки в районе 0,2 равнозначна вероятности выбора точки в районе 0,75.

Поэтому функция плотности вероятности равномерного распределения постоянна. Запишем выражение функции плотности вероятности непрерывного равномерного распределения для одной случайной величины x в интервале (x_{min}, x_{max}):

$$f(x; x_{min}, x_{max}) = \frac{1}{x_{max} - x_{min}} \quad \text{для } x_{min} < x < x_{max}$$

и ноль в противном случае.

Построим график функции плотности вероятности равномерного распределения в интервале (x_{min}, x_{max}). Как показано на рис. 2.8, график представляет собой прямой отрезок, т. к. равномерно распределенные данные (реальные или имитированные) распределяются равномерно по всему интересующему нас интервалу. Причем все значения данных в этом интервале имеют одинаковую вероятность обнаружения.

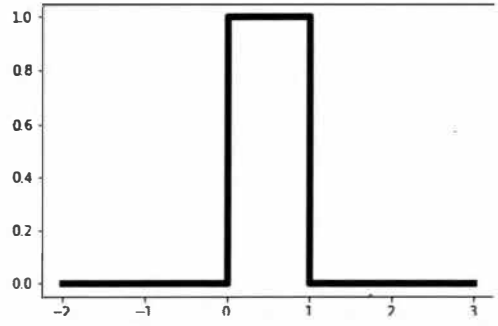


Рис. 2.8. График функции плотности вероятности равномерного распределения на отрезке [0; 1]

Равномерное распределение широко применяется в компьютерном моделировании для генерации случайных чисел *из любого другого распределения вероятностей*. В генераторах случайных чисел, используемых в Python, равномерное распределение встречается в базовых алгоритмах.

Распределение данных: колоколообразное нормальное (гауссово) распределение

Непрерывным распределением вероятностей, лучше всего подходящим для моделирования данных о росте человека (при ограничении одним полом), является *колоколообразное нормальное распределение*, называемое также *гауссовым распределением*. Выборки из нормального распределения скапливаются в районе среднего значения, где распределение достигает максимума, называемого *средним* μ, а затем симметрично убывают по мере удаления от среднего. Разброс распределения по мере удаления от среднего значения регулируется вторым параметром нормального распределения, который называется *стандартным отклонением* σ. Около 68% данных находится в пределах одного стандартного отклонения от среднего, 95% данных — в пределах двух стандартных отклонений от среднего и примерно 99,7% данных — в пределах трех стандартных отклонений от среднего (рис. 2.9).

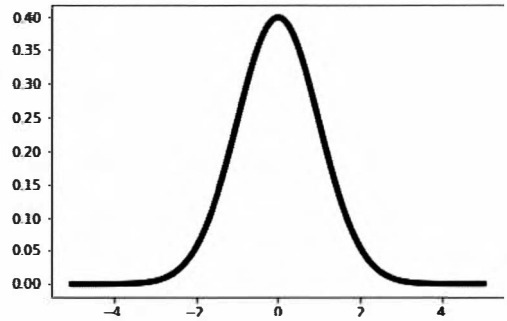


Рис. 2.9. График функции плотности вероятности колоколообразного нормального распределения при $\mu = 0$ и $\sigma = 1$

Более вероятно, что в выборку данных из нормального распределения попадут (появятся или будут наблюдаться) значения, близкие к среднему, а не крайне малые ($\rightarrow -\infty$) или крайне большие ($\rightarrow \infty$) значения. Пик в районе среднего значения и затухание на внешних границах распределения образуют всем известную колоколообразную форму. Заметим, что существуют и другие колоколообразные непрерывные распределения, из которых самым распространенным считается нормальное распределение. Его популярность обеспечена четким математическим обоснованием на базе *центральной предельной теоремы* (ЦПТ) — важнейшей теоремы из теории вероятностей.

Центральная предельная теорема утверждает, что среднее значение множества независимых случайных величин с одинаковым распределением (необязательно нормальным) имеет нормальное распределение. Этим объясняется тот факт, что нормальное распределение повсеместно встречается как в общественных отношениях, так и в природе. Его используют в моделировании веса новорожденных, распределении оценок учащихся, распределении доходов в национальных бюджетах, распределении показаний артериального давления и т. д.

Существуют специальные статистические тесты, позволяющие определить, насколько возможно смоделировать реальный набор данных с помощью нормального распределения. Это обсуждается более подробно в *главе 11*.

В условиях неопределенности и отсутствия предварительного понимания того, какое распределение лучше всего подойдет для решения конкретной задачи, предпочтение обычно отдается нормальному распределению. По сути, из всех вариантов распределений с одинаковой дисперсией нормальное распределение обладает максимальной *неопределенностью*, так что оно действительно кодирует наименьшее количество предварительных знаний в модели.

Формула функции плотности вероятности нормального распределения для одной случайной величины x (одномерной) со средним значением μ и стандартным отклонением σ имеет вид:

$$g(x;\mu,\sigma)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{(x-\mu)^2}{2\sigma^2}}.$$

Ее график при $\mu=0$ и $\sigma=1$ приведен на рис. 2.9.

Формула функции плотности вероятности нормального распределения двух случайных величин x и y (двумерное) имеет такой вид (рис. 2.10):

$$g(x,y;\mu_1,\sigma_2,p)=\frac{1}{\sqrt{(2\pi)^2\det\begin{pmatrix}\sigma_1^2 & p\sigma_1\sigma_2\\ p\sigma_1\sigma_2 & \sigma_2^2\end{pmatrix}}}\times$$

$$\times\exp\left(-\left\{\frac{1}{2}\begin{pmatrix}x-\mu_1 & y-\mu_2\end{pmatrix}\begin{pmatrix}\sigma_1^2 & p\sigma_1\sigma_2\\ p\sigma_1\sigma_2 & \sigma_2^2\end{pmatrix}^{-1}\begin{pmatrix}x-\mu_1\\ y-\mu_2\end{pmatrix}\right\}\right).$$

Можно записать приведенную двумерную формулу в более компактной форме на языке линейной алгебры:

$$g(x,y;\mu,\Sigma)=\frac{1}{\sqrt{(2\pi)^2\det(\Sigma)}}\exp\left(-\left\{\frac{1}{2}(u-\mu)^T\Sigma^{-1}(u-\mu)\right\}\right).$$

На рис. 2.11 показана выборка из 6000 точек двумерного нормального распределения. Точки, расположенные ближе к центру, будут отобраны с большей вероятностью, а удаленные от центра — с меньшей. Линии примерно повторяют контурные линии нормального распределения, причем неизвестно, из какого они распределения.

Теперь сделаем небольшую паузу и сравним формулы функции плотности вероятности для двумерного и одномерного нормального распределения.

- В ситуации одной случайной величины имеется только одно среднее значение μ и одно стандартное отклонение σ.

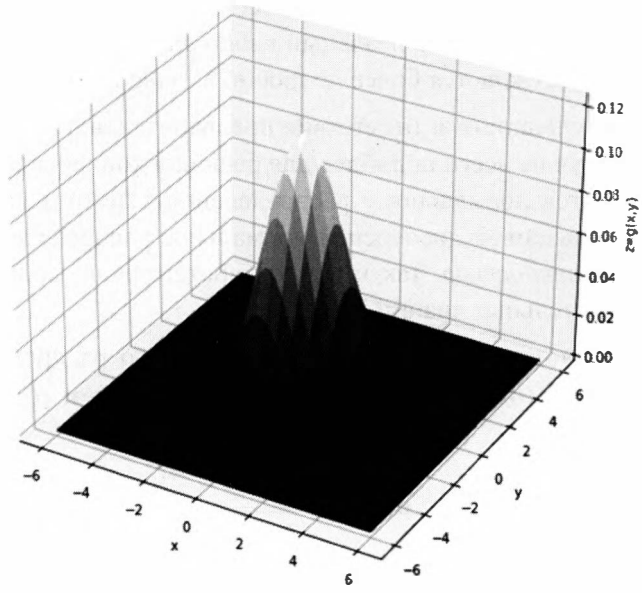


Рис. 2.10. График функции плотности вероятности колоколообразного двумерного нормального распределения

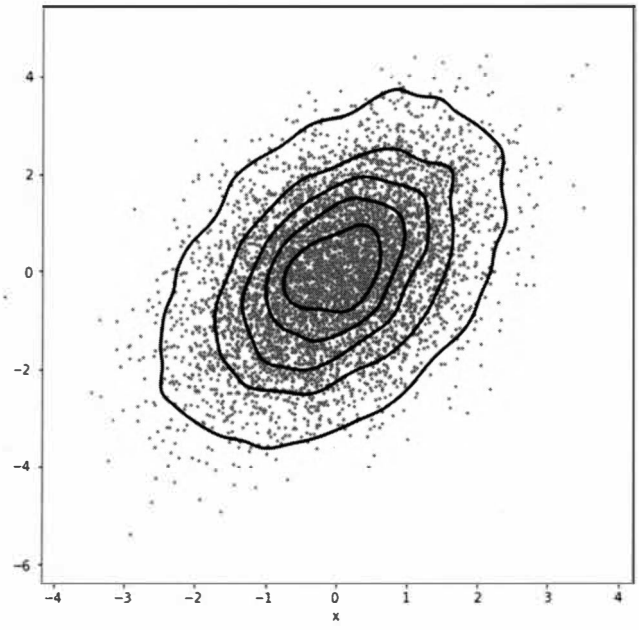


Рис. 2.11. Выборка из 6000 точек двумерного нормального распределения

- При наличии двух случайных величин мы имеем два средних значения $\begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}$ и два стандартных отклонения $\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$. Произведение σ^2 заменено ковариационной матрицей $\Sigma = \begin{pmatrix}\sigma_1^2 & \rho\sigma_1\sigma_2\\ \rho\sigma_1\sigma_2 & \sigma_2^2\end{pmatrix}$ и ее определителем, где ρ — корреляция между двумя случайными величинами, которая является ковариацией двух вариантов нормированных случайных величин.

Ту же самую формулу функции плотности вероятности двумерного нормального распределения можно обобщить на любую размерность, где имеется не две, а множество случайных величин. Например, если имеется 100 случайных величин, представляющих 100 признаков в наборе данных, то средний вектор в формуле будет иметь 100 записей, а размер ковариационной матрицы составит 100×100, причем дисперсия каждой случайной величины расположится на диагонали, а ковариация каждой из 4950 пар случайных величин — вне диагонали.

Распределение данных: другие важные и часто используемые распределения

Материал, изложенный в этой главе, будет многократно повторяться на протяжении всей книги, а *глава 11* посвящена исключительно вероятности. Понятия будут закрепляться по мере того, как они будут встречаться в различных интересных контекстах. Цель данной главы — познакомиться с лексикой вероятности и статистики и получить представление о важных понятиях, которые часто используются в приложениях ИИ. Не менее важным будет выработать вероятностную интуицию, которой мы можем пользоваться в последующих главах, не вникая в детали и не задерживаясь на них без необходимости.

Существует множество вероятностных распределений. Каждое из них симулирует различные типы реальных ситуаций. Наиболее распространенными считаются равномерное и нормальное распределения, хотя есть и другие важные распределения, которые часто встречаются в области ИИ. Напомним, что наша цель — моделирование окружающего мира для создания эффективных проектов, составления правильных прогнозов, принятия верных решений. Распределения вероятностей помогают нам делать прогнозы, когда в модели присутствует элемент случайности или когда мы не уверены в результатах.

В изучении распределений огорчает то, что большинство из них имеют странные названия, из-за чего не понятно, в каких процессах можно использовать данное распределение. Поэтому приходится либо, прикладывая максимум усилий, запоминать все эти названия, либо иметь шпаргалку по распределениям. Я предпочитаю шпаргалку. Еще одна досадная особенность заключается в том, что большинство приме-

ров в учебниках ограничены подбрасыванием монеты, бросанием кубика или вытаскиванием цветных шариков из урны. Так, у нас не остается примеров из реальной жизни и мотивации для понимания предмета, поскольку редко встретишь человека, бродящего по улицам и подбрасывающего монетку в ожидании орла или решки, разве что Харви Дента по прозвищу Двуликий в "Темном рыцаре" (весьма неплохой фильм 2008 года, где Джокер в исполнении Хита Леджера произносит несколько высказываний о случайности и вероятности, которые глубоко затронули меня, как например: "Мир жесток. И единственная мораль в жестоком мире... — это случайность. Беспристрастная. Непредвзятая. Справедливая"). В книге я постараюсь исправить ситуацию и приведу массу примеров из реальной жизни, насколько это мне позволит формат книги.

Некоторые из приведенных ниже распределений либо математически связаны друг с другом, либо естественным образом следуют одно из другого. Такая связность обсуждается в *главе 10*. Сейчас рассмотрим несколько известных распределений, укажем, какое является дискретным (предсказывает количество того, что нас интересует), а какое — непрерывным (предсказывает величину, присутствующую в континууме, например, время, которое должно пройти, прежде чем произойдет какое-то событие, причем важно помнить, что имеется в виду не количество часов, т. к. количество часов дискретно, а длина временно́го интервала), перечислим управляемые ими параметры, укажем отличительные характеристики, востребованные в приложениях искусственного интеллекта.

Биномиальное распределение.

Является дискретным. Представляет собой вероятность получения определенного количества успехов при многократном независимом повторении одного эксперимента. Его управляющими параметрами являются n — количество проводимых экспериментов и p — *заданная* вероятность успеха. В качестве примеров из реального мира здесь можно привести такие случаи, как прогнозирование количества пациентов со склонностями развития побочных эффектов от вакцины или нового лекарства в ходе клинических испытаний, количества рекламных кликов, приводящих к покупке, а также количества клиентов с просрочкой ежемесячных платежей по кредитным картам. Когда мы моделируем примеры из реального мира с помощью распределения вероятностей, требующего независимых испытаний, это означает, что мы *предполагаем* независимость, даже если реальные испытания фактически не являются независимыми. Хорошим тоном считается указывать в своих моделях на допущения.

Распределение Пуассона.

Является дискретным. Предсказывает количество редких событий, которые произойдут за определенный промежуток времени. Такие события считаются независимыми или слабо зависимыми, т. е. однократное наступление события не влияет на вероятность того, что в следующий раз оно произойдет за такой же период времени. Кроме того, они происходят с *известной* и *постоянной* средней периодичностью (скоростью) λ. Таким образом, нам известна средняя периодичность, и мы хотим предсказать, *сколько* таких событий произойдет за опре-

деленный период времени. Управляющим параметром распределения Пуассона выступает заданная частота редких событий λ. В качестве примеров из реального мира можно привести прогнозирование количества детей, которые родятся в указанное время, людей старше 98 лет в определенной популяции, количества альфа-частиц, излучаемых радиоактивной системой за определенный период времени, количества дубликатов счетов, рассылаемых налоговой службой, количества непопулярных товаров, проданных в определенный день, числа опечаток на одной странице в этой книге, количества бракованных изделий, выпущенных конкретным станком в назначенный день, количества посетителей магазина в конкретный час, числа автомобильных аварий, которые предстоит покрыть страховой компании в течение определенного периода времени, количества землетрясений, происходящих за определенный промежуток времени.

Геометрическое распределение.

Является дискретным. Предсказывает количество испытаний, необходимых для достижения успеха в ходе независимых испытаний, каждое из которых имеет *известное* значение вероятности успеха p. Очевидно, это значение является управляющим параметром. Здесь в качестве примеров из реального мира можно привести оценку количества недель, в течение которых компания может работать без сбоев в сети, количества часов, в течение которых машина не будет допускать брака в готовой продукции, или количества людей, которых требуется опросить перед дискуссией с оппонентом определенного политического законопроекта, который необходимо принять. Опять же, глядя на эти примеры, при моделировании с помощью геометрического распределения можно допускать независимость, тогда как в действительности испытания не всегда получаются независимыми.

Экспоненциальное распределение.

Является непрерывным. В случаях когда известно, что некоторое событие происходит с постоянной периодичностью λ, экспоненциальное распределение позволяет предсказать время ожидания до наступления этого события. Оно *не имеет памяти* в том смысле, что оставшееся время жизни элемента, входящего в экспоненциальное распределение, также является экспоненциальным. Управляющим параметром служит постоянная скорость λ. В качестве примеров из реального мира приведем расчет времени до очередного землетрясения, времени до возникновения просрочки по кредиту, времени поломки детали машины, времени до очередного теракта. Это достаточно востребовано в сфере надежности, когда рассчитывается надежность той или иной детали машины, что позволяет заявлять о таких вещах, как 10-летняя гарантия и т. д.

Распределение Вейбулла.

Является непрерывным. Широко применяется в технической отрасли при прогнозировании сроков службы изделий (здесь будет уместным вспомнить о 10-летних гарантиях). Когда изделие состоит из множества частей и какая-нибудь из них выходит из строя, все изделие перестает работать. Например, автомобиль не будет работать при выходе из строя аккумулятора или когда в коробке пере-

дач перегорит предохранитель. Распределение Вейбулла довольно точно выдает примерный срок службы автомобиля до полного прекращения работы, учитывая множество составляющих его частей и их *уязвимых мест* (при условии, что мы не занимаемся ремонтом автомобиля или переустановкой часов). Им управляют три параметра: форма, масштаб, расположение. Экспоненциальное распределение представляет собой его частный случай, поскольку имеет постоянную периодичность наступления событий, а распределение Вейбулла моделирует периодичность, которая, в зависимости от времени, либо увеличивается, либо уменьшается.

Лог-нормальное распределение.

Является непрерывным. Логарифмы каждого значения из этого распределении представляют собой нормально распределенные данные. Это означает, что хотя изначально данные не выглядят как нормально распределенные, можно попытаться изменить ситуацию, преобразовав их с помощью логарифмической функции. Этим распределение хорошо применимо к работе с асимметричными данными, имеющими низкое среднее значение, большую дисперсию, а также *допускающими только положительные значения*. Аналогично тому, как нормальное распределение возникает в процессе усреднения множества независимых выборок случайной величины (с помощью центральной предельной теоремы), логнормальное распределение возникает при *произведении* множества положительных значений выборки. С точки зрения математики, это связано с удивительным свойством логарифмических функций — логарифм произведения равен сумме логарифмов. Логнормальное распределение управляется тремя параметрами: формой, масштабом, расположением. В качестве примеров здесь можно привести объем газа в нефтяном запасе, отношение цены ценной бумаги на конец дня к ее цене на конец предыдущего дня.

Распределение хи-квадрат.

Является непрерывным. Представляет собой распределение суммы квадратов нормально распределенных независимых случайных величин. И здесь возникает вопрос: зачем нужно возводить в квадрат нормально распределенные случайные величины, а затем складывать их? Ответ в том, что именно так мы часто вычисляем дисперсию случайной величины или выборки данных, имея в основных задачах контроль дисперсии для снижения неопределенности. С этим распределением связаны два типа тестов на значимость: тест на хорошую подгонку, который показывает, насколько расходятся наши ожидания и наблюдения, а также тест на независимость и однородность признаков данных.

Распределение Парето.

Является непрерывным. Используется в множестве реальных приложений, скажем, при расчете времени выполнения задания, порученного суперкомпьютеру (например, вычисления в машинном обучении), уровня дохода семей определенной группы населения, количества друзей в социальной сети, размера файла в интернет-трафике. Управляется только одним параметром α и имеет *"тяжелый хвост"* (тяжелее, чем у экспоненциального распределения).

Прежде чем мы продолжим, кратко приведем еще несколько видов распределений.

Распределение Стьюдента (t-распределение).

Является непрерывным, аналогичным нормальному распределению, но используется при малом объеме выборки и неизвестной дисперсии совокупности.

Бета-распределение.

Является непрерывным. Дает случайные значения в заданном интервале.

Распределение Коши.

Является непрерывным, патологическим, поскольку ни среднее, ни дисперсия не определены. Может образоваться с помощью тангенсов произвольно выбранных углов.

Гамма-распределение.

Является непрерывным. Связано с временем ожидания наступления не одного, как в экспоненциальном распределении, а нескольких независимых событий *n*.

Отрицательное биномиальное распределение.

Является дискретным. Связано с количеством независимых испытаний, необходимых для получения определенного количества успехов.

Гипергеометрическое распределение.

Является дискретным, аналогичным биномиальному распределению, но испытания не являются независимыми.

Отрицательное гипергеометрическое распределение.

Является дискретным. Отражает количество зависимых испытаний, необходимых для получения определенного количества успехов.

Варианты значений слова "распределение"

Возможно, вы уже заметили, что слово *"распределение"* в различных контекстах используется для обозначения разных (но связанных между собой) понятий. Такое непоследовательное использование одного и того же слова вызывает путаницу и может оттолкнуть от себя часть желающих войти в эту область.

Для того чтобы облегчить понимание слова "распределение" в том или ином контексте, перечислим связанные с ним понятия.

- Когда при наличии реальных данных, например данных о росте и весе, приведенных в этой главе, строится гистограмма одного из признаков набора данных, скажем роста, получается *эмпирическое распределение* данных о росте. В большинстве случаев основная функция плотности вероятности роста во всей популяции, называемая также распределением, неизвестна, т. к. имеющиеся в нашем распоряжении реальные данные являются лишь выборкой из этой популяции. Поэтому мы стараемся оценить или смоделировать ее, используя распределения вероятностей из области теории вероятностей. Для работы с характеристиками

роста и веса, разделенных по половому признаку, лучше воспользоваться гауссовым распределением.

- В случае дискретной случайной величины под словом "распределение" понимается либо ее массовая функция вероятности, либо кумулятивная функция распределения (которая определяет вероятность того, что случайная величина меньше или равна определенному значению $f(x) = \text{Prob}(X \leq x)$[2]).
- В случае непрерывной случайной величины слово "распределение" подразумевает либо ее функцию плотности вероятности, либо кумулятивную функцию распределения, интеграл которой дает вероятность получения случайной величины, меньшей или равной определенному значению.
- В случае нескольких случайных величин (дискретных, непрерывных или смешанных) слово "распределение" относится к их совместному распределению вероятностей.

Общая цель — обеспечить соответствие между идеализированной математической функцией, например случайной переменной с соответствующим распределением, и реальными наблюдаемыми данными или явлениями, имеющими свое эмпирическое распределение.

Работая с реальными данными, можно с помощью случайной переменной смоделировать каждый признак набора данных. Таким образом, в некотором смысле математическая случайная величина с соответствующим распределением является идеализированной версией измеряемого или наблюдаемого признака.

Наконец, распределения повсеместно встречаются в приложениях ИИ. Мы еще не раз увидим их в последующих главах книги, например: распределение весов на каждом слое нейросети, распределение помех и погрешностей различных моделей машинного обучения.

A/B-тестирование

В завершение главы совершим краткий экскурс в область A/B-тестирования, которое называется также *сплит-тестированием* либо *рандомизированными одинарным тестированием*, либо *двойными слепыми тестированием*. Оно представляет большой интерес для специалистов по обработке данных на фоне того, как бесчисленное множество организаций в погоне за повышением доходов, вовлеченности, удовлетворенности клиентов полагаются на данные, полученные в результате A/B-тестирования.

Ежегодно такие компании, как Microsoft, Amazon, LinkedIn, Google и другие, проводят тысячи A/B-тестирований.

Идея A/B-тестирования довольно проста: делим население на две группы. В одной группе (тестовой) запускаем тестовую версию (новый дизайн веб-страницы, другой

[2] Здесь и далее Prob в формуле означает вероятность. — *Прим. ред.*

размер шрифта, новое лекарство, новая политическая реклама), а вторую группу назначаем контрольной. Далее сравниваем данные между двумя группами.

Если испытуемые не знают, к какой группе они принадлежат (некоторые вообще не знают, что участвуют в тестировании), но это известно организаторам эксперимента, такой тест называется *одинарным слепым*. *Двойное слепое* тестирование заключается в том, что ни экспериментаторы, ни испытуемые не знают, с какой группой они имеют дело.

Итоги и перспективы

В этой главе мы подчеркнули ключевую роль данных в искусственном интеллекте. Мы также выяснили различия между понятиями, часто вызывающими путаницу: структурированные и неструктурированные данные, линейные и нелинейные модели, реальные и имитационные данные, детерминированные функции и случайные величины, дискретные и непрерывные распределения, апостериорные вероятности и функции правдоподобия. Вдобавок, не вникая в детали, мы наметили карту вероятностей и статистик, необходимых в ИИ, и познакомились с наиболее известными распределениями вероятностей.

Если по ходу книги у вас в голове возникнет путаница из-за новых понятий вероятности, всегда можно обратиться к карте в этой главе и посмотреть, как данное понятие вписывается в общую картину теории вероятностей и, что особенно важно, как оно связано с искусственным интеллектом. Без знания о том, как та или иная математическая концепция связана с ИИ, мы всего лишь имеем инструмент, понимая, как он включается, но не имея ни малейшего представления о том, для чего он нужен.

До сих пор мы не упоминали *случайные матрицы* и *высокоразмерные вероятности*. В этих областях теория вероятностей со свойственным ей постоянным отслеживанием распределений, ожиданий, вариаций любых соответствующих случайных величин сливается с линейной алгеброй с ее повышенным вниманием к собственным значениям и разнообразным матричным разложениям. Эти области чрезвычайно важны для работы с высокоразмерными данными в приложениях ИИ. Они обсуждаются в *главе 11*, посвященной вероятности.

В следующей главе мы научимся подгонять данные под функцию, а затем применять эту функцию для прогнозирования и/или принятия решений. С точки зрения математики, мы находим веса ω, которые характеризуют силу различных взаимодействий между признаками данных. После того как участвующие типы взаимодействий (формула подгоночной функции, которая называется *обучающей функцией*) и сила этих взаимодействий (значения весов ω) охарактеризованы, можно делать прогнозы. В области искусственного интеллекта идею *характеризации подгоночной функции с подходящими значениями весов* можно успешно реализовать в приложениях, связанных с компьютерным зрением, обработкой естественного языка, прогнозной аналитикой (например, цен на жилье, времени до ремонта и т. д.), и многих других.

ГЛАВА 3

Подгонка функций под данные

https://t.me/it_boooks/2

Сегодня оно подходит. А как насчет завтра?
— Хала

В этой главе мы познакомимся с основными математическими идеями, лежащими в основе многих приложений ИИ, в том числе с математическими движками нейросетей. Наша цель — усвоить структуру части задач машинного обучения, связанных с искусственным интеллектом.

Постановка задачи.

Задача зависит от конкретного случая использования, среди которых классификация изображений, классификация документов, прогнозирование цен на жилье, выявление мошенничества или аномалий, рекомендация нового товара, прогнозирование вероятности повторного совершения преступления, вычисление внутренней структуры здания по внешним изображениям, преобразование речи в текст, генерация аудио, генерация изображений, генерация видео и т. д.

Получение нужных данных.

Речь идет об обучении моделей правильным действиям. Мы говорим, что наши модели *обучаются* на данных. Необходимо убедиться, что наши данные — чистые, полные и при необходимости, в зависимости от конкретной модели, преобразованные (нормализованные, стандартизованные, с агрегированием некоторых признаков и т. д.). Обычно этот этап занимает гораздо больше времени, чем реализация и обучение моделей машинного обучения.

Создание функции гипотезы.

Термины *"функция гипотезы"*, *"обучающая функция"*, *"функция предсказания"*, *"обучающая функция"*, *"модель"* используются как взаимозаменяемые. Наше основное предположение состоит в том, что математическая функция ввода-вывода объясняет наблюдаемые данные, и в дальнейшем ее можно применять в прогнозировании новых данных. Для модели мы задаем признаки, например ежедневные привычки человека, а она выдает прогноз, скажем, вероятность того, что этот человек выплатит кредит. В модель из данной главы мы вводим длину рыбы и получаем ее вес.

Определение численных значений весов.

Во многих моделях (в том числе и в нейросетях) мы столкнемся с тем, что обучающая функция имеет неизвестные параметры, называемые *весами*. Задача в

том, чтобы найти числовые значения этих весов по имеющимся данным. После того как будут найдены значения весов, можно использовать обученную функцию в прогнозировании, подставляя признаки новой точки данных в формулу обученной функции.

Создание функции ошибки.

Для того чтобы найти значения неизвестных весов, создадим еще одну функцию, называемую *функцией ошибки*, *функцией затрат*, *функцией цели* или *функцией потерь* (в области ИИ все имеет три названия или более). Эта функция должна измерять некое расстояние между истинным положением дел и нашими предсказаниями. Естественно, мы хотим, чтобы наши прогнозы были как можно ближе к истине, поэтому ищем значения весов, которые минимизируют нашу функцию потерь. С точки зрения математики, мы решаем задачу минимизации. Область *математической оптимизации* играет важную роль в ИИ.

Принятие решения о математических формулах.

На протяжении всего процесса мы выступаем инженерами, так что именно мы принимаем решение о математических формулах для функций обучения, функций потерь, методов оптимизации, компьютерных реализаций. Инженеры решают разные задачи, добиваясь тех или иных результатов, и это нормально. Оценкой их работы в конечном счете является эффективность реализованной модели, и вопреки распространенному мнению математические модели довольно гибки, а в случае необходимости их можно подстроить и изменить. Поэтому после реализации модели крайне важно следить за ее производительностью.

Определение способа поиска минимизаторов.

Поскольку наша цель — найти такие значения весов, которые минимизируют погрешность между прогнозами и истиной, необходимо найти эффективный математический способ поиска *минимизаторов* — особых значений весов, которые дают наименьшую ошибку. Ключевую роль здесь играет метод *градиентного спуска*. Этот мощный и в то же время простой метод предполагает вычисление *одной производной* функции ошибки. Именно поэтому половина уроков по математике посвящена вычислению производных (и градиента — одной производной в высших измерениях). Существуют и другие методы, в которых требуются вычисления двух производных. Мы познакомимся с ними и обсудим преимущества и недостатки применения методов более высокого порядка.

Использование алгоритма обратного распространения ошибки.

В случаях когда массивы данных огромны, а модель представляет собой многоуровневую нейронную сеть, требуется эффективный способ вычисления одной производной. Здесь на помощь приходит *алгоритм обратного распространения ошибки*. Этот алгоритм, а также метод градиентного спуска обсуждаются в *главе 4*.

Регуляризация функции.

Если наша обучающая функция чрезмерно подогнана под заданные данные, она будет плохо работать на новых данных. Такая функция будет восприимчива не

только к сигналу, но и к помехам в данных (например, функция на рис. 3.1 слева). Нам не нужны помехи. И в этом нам поможет *регуляризация*. Существует множество математических способов регуляризировать функцию, т. е. сделать ее более гладкой и устойчивой. В целом функция, учитывающая помехи в данных, достаточно сильно колеблется. Нам нужны более регулярные функции. Методы регуляризации мы рассмотрим в *главе 4*.

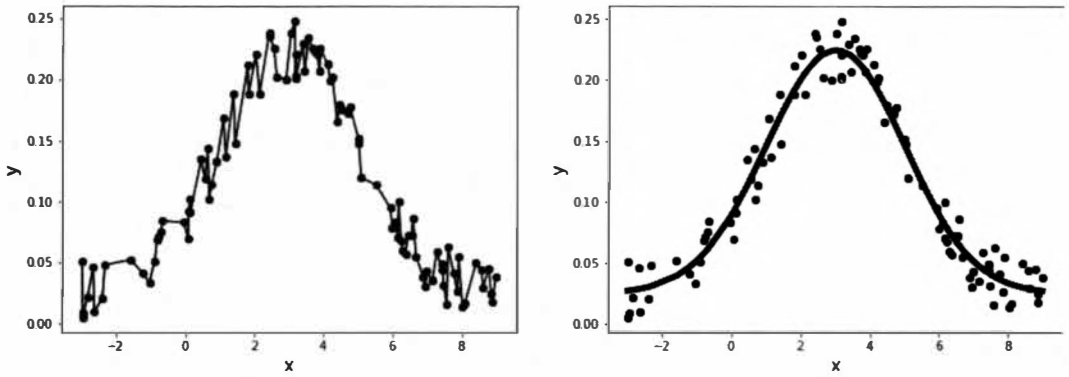


Рис. 3.1. *Слева*: функция подгонки идеально подходит к данным, но не является хорошей функцией прогнозирования, т. к. подогнана не под основной сигнал в данных, а под помехи. *Справа*: более регулярная функция, подогнанная под тот же набор данных. Она дает более верные прогнозы, чем функция слева, несмотря на то что последняя лучше подходит к точкам данных

В следующих разделах мы изучим эту структуру проблемы ИИ на примере реальных, но простых наборов данных. В последующих главах мы увидим, как эти же концепции обобщаются на гораздо более сложные задачи.

Лучшие традиционные модели машинного обучения

Все данные, о которых пойдет речь в этой главе, *размечены* "базовой истиной", а целью наших моделей является *предсказание* меток новых (незнакомых) и неразмеченных данных. В этом и заключается *супервизорное* обучение.

В следующих разделах мы подгоним обучающие функции под размеченные нами данные, используя приведенные ниже известные модели машинного обучения. Несмотря на то что в последнее время можно часто слышать о новейших разработках в области ИИ, все же в типичной бизнес-среде лучше начать с более традиционных моделей.

Линейная регрессия.

Предсказание числового значения.

Логистическая регрессия.

Классификация на два класса (бинарная классификация).

Регрессия softmax.

Классификация на несколько классов.

Машины опорных векторов.

Классификация на два класса, или регрессия (предсказание числового значения).

Деревья решений.

Классификация на любое количество классов, или регрессия (предсказание числового значения).

Случайные леса.

Классификация на любое количество классов, или регрессия (предсказание числового значения).

Ансамбли моделей.

Объединение результатов многих моделей путем усреднения значений предсказаний, голосования за наиболее популярный класс или с помощью другого механизма объединения.

Кластеризация k средних.

Классификация на любое количество классов, или регрессия.

Для сравнения производительности можно протестировать сразу несколько моделей на одних и тех же наборах данных. В реальном мире довольно сложно запустить какую-либо модель без сравнения со множеством других. В этом заключается специфика искусственного интеллекта, требующего больших вычислений, и возникающая с связи с этим необходимость в параллельных вычислениях, позволяющих обучать сразу несколько моделей (за исключением моделей, опирающихся на результаты других моделей, например, в случае *суммирования*; в них мы не можем использовать параллельные вычисления).

Прежде чем мы перейдем к моделям машинного обучения, важно отметить, что, как уже неоднократно сообщалось, на обучение моделей машинного обучения тратится не более 5% времени специалиста по исследованию данных и/или по искусственному интеллекту. Основная часть времени уходит на получение данных, их очистку, организацию, создание соответствующих конвейеров и т. д., *прежде* чем данные попадут в модели машинного обучения. Таким образом, машинное обучение — всего лишь шаг в производственном процессе, зато он легко дается, когда данные готовы к обучению модели. Мы узнаем, как работают эти модели машинного обучения — в них содержится значительная часть необходимой нам математики. Разработчики ИИ неустанно совершенствуют их и автоматически встраивают в производственные конвейеры. Поэтому в конечном счете важно изучить весь конвейер — от исходных данных (включая их хранение, аппаратное обеспечение, протоколы запросов и т. д.) до реализации и контроля. Обучение машинному обучению — это только один из фрагментов более важной и увлекательной истории.

Мы обязаны начать с *регрессии*, поскольку ее идеи играют фундаментальную роль в большинстве моделей и приложений ИИ, о которых пойдет речь далее. Только в

случаях *линейной регрессии* можно найти минимизирующие веса аналитическим методом, создав формулу для желаемых весов непосредственно в терминах обучающего набора данных и его целевых меток. Такое явное *аналитическое* решение возможно благодаря простоте модели линейной регрессии. Поскольку большинство других моделей не имеют таких явных решений, в этих случаях требуется найти минимизаторы с помощью численных методов, среди которых наиболее распространенным является градиентный спуск.

В процессе моделирования в регрессионных и множестве других перспективных моделях, включая нейросети, о которых пойдет речь в следующих главах, можно выделить такую последовательность действий:

1. Обучающая функция.
2. Функция потерь.
3. Оптимизация.

Численные и аналитические решения

Важно знать, чем отличаются численные решения от аналитических решений математических задач. Математическая задача может быть любой, например:

- найти минимизатор функции;
- найти наилучший способ добраться из пункта назначения А в пункт назначения Б при ограниченном бюджете;
- найти оптимальный способ создания хранилища данных и составления запросов к нему;
- найти решение математического уравнения (где левая часть с математическими данными равна правой части с математическими данными). Это могут быть алгебраические уравнения, обыкновенные дифференциальные уравнения, дифференциальные уравнения в частных производных, интегро-дифференциальные уравнения, системы уравнений и любые другие математические уравнения. Их решения могут быть статическими или с течением времени меняющимися. Они могут моделировать что угодно из физического, биологического, социально-экономического, природного миров.

Объясним некоторые термины.

Численный.

Связанный с числами.

Аналитический.

Связанный с анализом.

В основном численные решения гораздо проще и доступнее, чем аналитические, при условии, что мы обладаем достаточной вычислительной мощностью для их моделирования и вычисления. Единственное, что нам остается, — дискретизиро-

вать несколько непрерывных пространств и/или функций (превратить континуум в множество точек), порой довольно хитроумными способами, и оценить функции на этих дискретных величинах. Правда, проблема с численными решениями заключается в том, что они являются лишь приближенными. Численные решения нельзя считать точными без оценок того, насколько далеки они от истинных аналитических решений и как быстро они к ним приходят, что, в свою очередь, требует должной математической и аналитической подготовки. Тем не менее с их помощью можно получить невероятно полезную информацию об истинных решениях. Во многих случаях численные решения являются единственно возможными, а многие области науки и техники вообще не продвинулись бы вперед, не опираясь на численные решения сложных задач. Если бы они ждали, когда появятся аналитические решения и доказательства или, другими словами, когда математическая теория наверстает свое, их прогресс не был бы настолько очевидным.

Аналитические решения, напротив, надежны, точны и целиком опираются на математическую теорию. Они сопровождаются теоремами и доказательствами. Аналитические решения, когда доступны, обладают огромной силой. К сожалению, в действительности получить их непросто, а иногда вообще невозможно, для этого требуются глубокие знания и опыт в таких областях, как исчисление, математический анализ, алгебра, теория дифференциальных уравнений и т. д. Тем не менее аналитические методы весьма эффективны при описании важных свойств решений (даже если явные решения недоступны), использовании численных методов, сравнении приближенных численных методов с истинными (в тех редких случаях, когда аналитические решения доступны).

Одним исследователям близка чисто аналитическая и теоретическая деятельность, другим — числовая, вычислительная, однако наилучшая позиция — где-то посередине, когда достигнуто достаточное понимание аналитических и численных аспектов математических проблем.

Регрессия: предсказание числового значения

Быстрый поиск на сайте Kaggle (**https://www.kaggle.com/**) дает множество замечательных наборов данных и соответствующих блокнотов для регрессии. Выбираем наугад простой набор данных Fish Market ("Рыбный рынок", **https://oreil.ly/yaV96**)[1] и воспользуемся им в дальнейшем при обсуждении математики. Наша цель — построить модель, предсказывающую вес рыбы по пяти измерениям ее длины или признакам, размеченным в наборе данных как Length1 (Длина1), Length2 (Длина2), Length3 (Длина3), Height (Высота) и Width (Ширина) (рис. 3.2). Еще можно включить в нашу модель категориальный признак "Вид" (что дало бы более точные результаты, поскольку по виду рыбы можно легко предсказать ее вес), но для простоты не будем этого делать. Если мы решим включить признак Species (Вид), нам

[1] Ссылка на момент публикации книги оказалась нерабочей. См., например, **https://www.kaggle.com/datasets/vipullrathod/fish-market**. — *Прим. ред.*

придется преобразовать его значения в числовые методом *однократного горячего кодирования (one-hot encoding)*, смысл которого, как можно догадаться из названия, заключается в присвоении каждой рыбе кода, состоящего из единиц и нулей, в зависимости от ее принадлежности к определенному классу (виду). В признак Species (Вид) включены семь видов: окунь (Perch), лещ (Bream), плотва (Roach), щука (Pike), корюшка (Smelt), парки (Parkki) и сиг (Whitefish). Таким образом, в случае со щукой мы кодируем ее вид как (0, 0, 0, 1, 0, 0, 0), а с лещом — (0, 1, 0, 0, 0, 0, 0). Безусловно, это увеличит пространство признаков на семь дополнительных значений и добавит еще семь весов для обучения.

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

Рис. 3.2. Первые пять строк набора данных Fish Market с сайта Kaggle (**https://oreil.ly/URygY**)[2]. Столбец Weight (Вес) является целевым признаком, и задача заключается в том, чтобы построить модель, которая будет предсказывать вес новой рыбы на основе измерений ее длины

В целях экономии обозначим наши пять признаков как x_1, x_2, x_3, x_4, x_5, а затем запишем вес рыбы как функцию этих признаков, $y = f(x_1, x_2, x_3, x_4, x_5)$. Таким образом, после нахождения подходящей формулы для функции остается лишь ввести значения признаков для определенной рыбы, и наша функция выдаст ее прогнозируемый вес.

В этом разделе мы закладываем основу для всего дальнейшего повествования, поэтому в первую очередь разберем, как организован материал.

Обучающая функция.

- Параметрические и непараметрические модели.

Функция потерь.

- Предсказанное значение и истинное значение.
- Расстояние по абсолютной величине и квадратичное расстояние.
- Функции, имеющие сингулярности (точки излома).
- Для линейной регрессии функцией потерь является среднеквадратическая ошибка.
- Векторы в этой книге всегда являются векторами-столбцами.

[2] См., например, **https://www.kaggle.com/code/drfrank/fish-market-data-visualisation-machine-learning**. — *Прим. ред.*

- Обучающие, проверочные и тестовые поднаборы.
- При наличии в обучающих данных сильно коррелированных признаков.

Оптимизация.

- Выпуклые и невыпуклые ландшафты.
- Как находятся минимизаторы функций.
- Исчисление в двух словах.
- Пример одномерной оптимизации.
- Производные от выражений линейной алгебры, которыми мы постоянно используемся.
- Минимизация функции потерь при среднеквадратической ошибке.
- Внимание! Умножение больших матриц друг на друга обходится слишком дорого, поэтому лучше умножать матрицы на векторы.
- Внимание! Мы не стремимся к слишком тщательной подгонке обучающих данных.

Обучающая функция

Быстро изучив данные, например, построив график зависимости между весом и различными признаками длины, мы можем предположить линейную модель (хотя в данном случае нелинейная модель была бы лучше). То есть мы предполагаем, что вес линейно зависит от признаков длины (рис. 3.3).

Это значит, что вес рыбы y можно вычислить с помощью *линейной комбинации* пяти показателей ее длины плюс показатель смещения ω_0, что в результате дает следующую *обучающую функцию*:

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5.$$

После того как в процессе моделирования мы приняли основное решение об использовании линейной обучающей функции $f(x_1, x_2, x_3, x_4, x_5)$, остается только найти подходящие значения параметров ω_0, ω_1, ω_2, ω_3, ω_4, ω_5. Наилучшие значения параметров ω мы *узнаем* из данных. Процесс поиска подходящих значений ω с помощью данных называется *обучением* модели. Модель считается *обученной*, когда в ней определены значения ω.

В общем случае обучающие функции, как линейные, так и нелинейные, в том числе представляющие собой нейросети, имеют неизвестные параметры ω, которые необходимо узнать из заданных данных. В линейных моделях каждый параметр придает каждому признаку определенный вес в процессе прогнозирования. Так, если значение ω_2 больше значения ω_5, значит, в прогнозировании второй признак будет играть более важную роль, чем пятый, в предположении, что второй и пятый признаки имеют сопоставимые масштабы. Это одна из причин, по которой перед обучением модели стоит масштабировать или нормализовать данные.

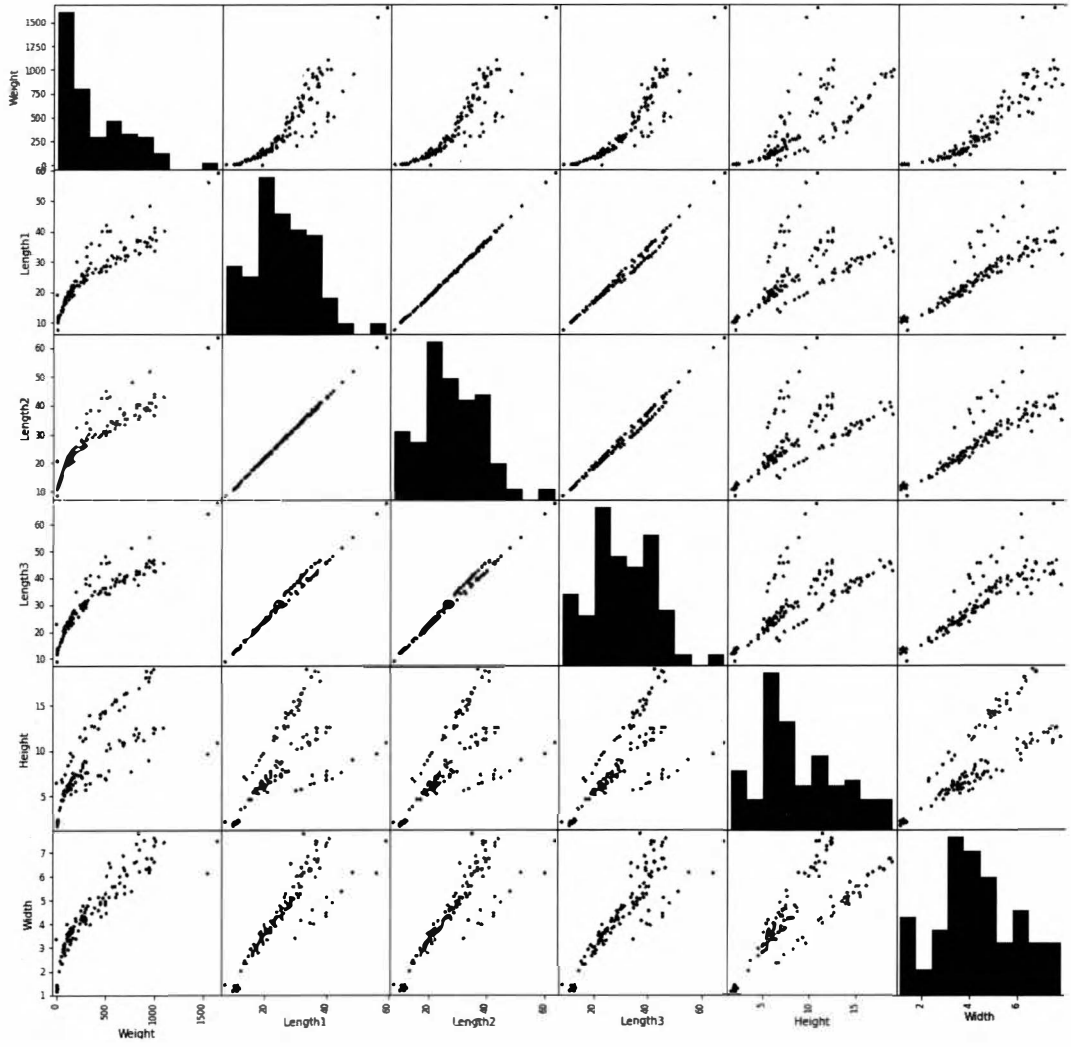


Рис. 3.3. Диаграммы рассеяния числовых характеристик набора данных Fish Market. Более подробную информацию можно найти на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**) или в публичных блокнотах, связанных с этим набором данных, на Kaggle

В случае когда значение ω_3, связанное с третьим признаком, уменьшается, т. е. становится нулевым или пренебрежимо малым, третий признак можно исключить из набора данных, т. к. он не играет никакой роли в прогнозировании. Таким образом, обучение ω на основе данных позволяет математически вычислить вклад каждого признака в прогноз (или важность комбинаций признаков, когда некоторые из них были объединены до обучения на этапе подготовки данных). Другими словами, модели учатся тому, как взаимодействуют признаки данных и насколько сильным является это взаимодействие. Идея состоит в том, что с помощью обученной функ-

ции обучения мы можем количественно оценить, как признаки объединяются для получения наблюдаемых и еще не наблюдаемых результатов.

Параметрические и непараметрические модели

Модель, в формулу которой заранее заложены параметры (мы называем их весами), например ω в нашей текущей модели линейной регрессии:

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

(а в дальнейшем и ω нейросетей), называется *параметрической моделью*. Это означает, что мы фиксируем формулу обучающей функции до начала обучения, и все, что происходит в процессе обучения, сводится к определению параметров, входящих в эту формулу. Предварительное фиксирование формулы аналогично определению *семейства*, к которому принадлежит обучающая функция, а нахождение значений параметров определяет именно тот член этого семейства, который наилучшим образом объясняет данные.

Непараметрические модели, такие как деревья решений и случайные леса, которые мы рассмотрим в этой главе, заранее не задают формулу обучающей функции и ее параметры. Поэтому при обучении непараметрической модели мы не знаем, сколько параметров в итоге будет иметь обученная модель. Модель *адаптируется* к данным и на их основе сама определяет необходимое количество параметров. И здесь важно не переусердствовать с чрезмерной подгонкой. Напомним, что нам не нужно, чтобы модели адаптировались к данным слишком сильно, т. к. в этом случае они будут плохо обобщаться на неизвестные данные. Для этого в моделях обычно применяются методы, позволяющие избежать чрезмерной подгонки.

И параметрические, и непараметрические модели обладают дополнительными параметрами — так называемыми *гиперпараметрами*, которые также требуют настройки в процессе обучения. Но их не закладывают в формулу обучающей функции (и не включают в формулу непараметрической модели). Гиперпараметры будут довольно часто встречаться нам на протяжении всего повествования.

Функция потерь

Мы убедились, что следующим логическим шагом будет поиск подходящих значений ω, входящих в функцию обучения (линейной параметрической модели), на основе имеющихся у нас данных. Для этого необходимо *оптимизировать соответствующую функцию потерь*.

Предсказанное и истинное значения

Предположим, что каждому из неизвестных ω_0, ω_1, ω_2, ω_3, ω_4, ω_5 мы присваиваем случайные числовые значения, например $\omega_0 = -3$, $\omega_1 = 4$, $\omega_2 = 0{,}2$, $\omega_3 = 0{,}03$,

$\omega_4 = 0,4$, $\omega_5 = 0,5$. Тогда формула для линейной обучающей функции $y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$ примет вид

$$y = -3 + 4x_1 + 0,2x_2 + 0,03x_3 + 0,4x_4 + 0,5x_5$$

и готова делать предсказания. Подставляем числовые значения признаков длины i-й рыбы и получаем предсказанное значение ее веса. Допустим, первая рыба в нашем наборе данных — лещ с измерениями длины $x_1^1 = 23,2$, $x_2^1 = 25,4$, $x_3^1 = 30$, $x_4^1 = 11,52$, $x_5^1 = 4,02$. Подставляем их в функцию обучения и получаем предсказание веса рыбы:

$$\begin{aligned} y^1_{\text{предсказ}} &= \omega_0 + \omega_1 x_1^1 + \omega_2 x_2^1 + \omega_3 x_3^1 + \omega_4 x_4^1 + \omega_5 x_5^1 = \\ &= -3 + 4 \cdot 23,2 + 0,2 \cdot 25,4 + 0,03 \cdot 30 + 0,4 \cdot 11,52 + 0,5 \cdot 4,02 = \\ &= 102,398 \text{ грамма.} \end{aligned}$$

В общем случае для i-й рыбы имеется:

$$y^i_{\text{предсказ}} = \omega_0 + \omega_1 x_1^i + \omega_2 x_2^i + \omega_3 x_3^i + \omega_4 x_4^i + \omega_5 x_5^i.$$

В нашем случае рыба обладает определенным *истинным* весом $y^i_{\text{истина}}$, т. е. меткой в соответствующем наборе размеченных данных. У первой рыбы из нашего набора данных истинный вес $y^1_{\text{истина}}$ = 242 грамма, а наша линейная модель со случайно выбранными значениями ω предсказала 102,398 грамма. Конечно, разница достаточно существенная, причем она вызвана тем, что мы вообще не калибровали значения ω. В любом случае можно измерить ошибку между истинным весом и весом, предсказанным моделью, а затем поискать способы улучшить ситуацию с выбором значений ω.

Расстояние по абсолютной величине и квадратичное расстояние

Одна из приятных особенностей математики заключается в том, что она обладает множеством способов измерения расстояния между объектами с помощью различных метрик. Например, мы можем условно обозначить расстояние между двумя величинами как 1, если они разные, и 0, если одинаковые, закодировав слова "разные" — 1 и "одинаковые" — 0. Конечно, при такой упрощенной метрике мы теряем массу информации, т. к. расстояние, например, между 2 и 10 будет равно расстоянию между 2 миллионами и 1 миллионом, т. е. 1.

В машинном обучении популярны несколько метрик. Сначала приведем две самые распространенные:

- расстояние по абсолютной величине: $\left|y_{\text{предсказ}} - y_{\text{истина}}\right|$, обусловленное функцией $|x|$;

- квадратичное расстояние: $\left|y_{\text{предсказ}} - y_{\text{истина}}\right|^2$, обусловленное функцией $|x|^2$ (аналогично x^2 для скалярных величин). Естественно, при этом единицы измерения тоже возводятся в квадрат.

Анализируя графики функций $|x|$ и $|x|^2$ на рис. 3.4, можно заметить существенные различия в их гладкости в точке (0; 0). Здесь у функции $|x|$ имеется угол, вследствие чего она становится недифференцируемой при $x = 0$. Такая *сингулярность* $|x|$ при $x = 0$ заставляет многих разработчиков (и математиков!) отказаться от этой функции или функций с подобными сингулярностями в своих моделях. Между тем необходимо усвоить следующее.

Математические модели достаточно гибки: при возникновении препятствий мы углубляемся в них, выясняем, что происходит, а затем обходим препятствие.

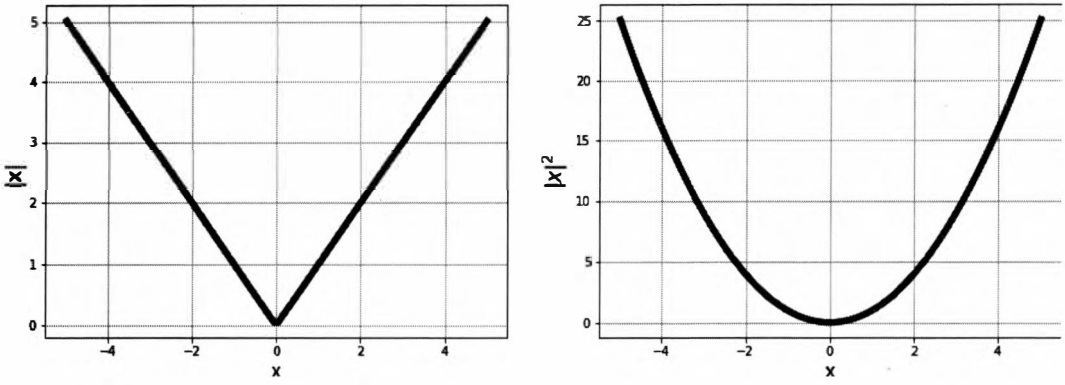


Рис. 3.4. *Слева*: при $x = 0$ график $|x|$ имеет угол, в результате чего его производная в этой точке не определена. *Справа*: при $x = 0$ график $|x|^2$ гладкий, значит, в этой точке все хорошо с производной этой функции

Помимо отличий в регулярности функций $|x|$ и $|x|^2$ (в смысле наличия или отсутствия у них производных во всех точках), следует обратить внимание на еще один аспект, прежде чем включать какую-либо функцию в формулу ошибки: *квадрат большого числа будет еще больше*. Это нехитрое наблюдение означает, что если мы решим измерять ошибку с помощью квадратичных расстояний между истинными и предсказанными значениями, то наш метод станет *более чувствительным к выбросам* в данных. Один неудачный выброс может перекосить всю функцию предсказания в свою сторону, а значит, и в сторону других, более характерных закономерностей в данных. В идеале нам следует позаботиться о выбросах и решить, оставлять или не оставлять их на этапе подготовки данных до загрузки в модель машинного обучения.

И последнее отличие $|x|$ (и других аналогичных кусочно-линейных функций) от $|x|^2$ (и других аналогичных, но дифференцируемых функций) состоит в том, что производная $|x|$ находится довольно легко:

$$\begin{cases} 1, & \text{если } x > 0, \\ -1, & \text{если } x < 0 \\ \text{(не определена,} & \text{если } x = 0). \end{cases}$$

Такое свойство позволяет *обойтись без вычислений* при использовании производной $|x|$, что чрезвычайно актуально для модели, предполагающей миллиарды вычислительных шагов. Производные других функций, отличных от линейных и кусочно-линейных, обычно требуют вычислений (т. к. в их формулах присутствуют не только константы, как в кусочно-линейном случае, но и значения x), а в условиях больших данных это может обойтись довольно дорого.

Функции с сингулярностями

В общем случае графики *дифференцируемых* функций не имеют ни перегибов, ни переломов, ни углов, ни каких-либо иных изломов. У функции с такими *сингулярностями* в подобных точках не бывает производной. Это связано с тем, что при наличии точки излома можно по-разному провести две касательные линии к ее графику — слева или справа от точки (рис. 3.5). Напомним, что производная функции в точке равна наклону касательной к графику функции в этой точке. При наличии двух *разных* наклонов определить производную в точке невозможно.

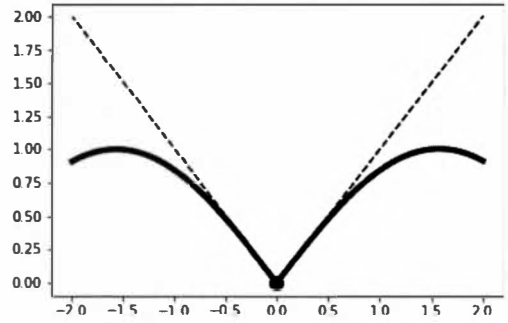


Рис. 3.5. При наличии сингулярных точек производной не бывает. В таких точках возможно более одного наклона касательной

В конечном счете возникает проблема в применении методов, основанных на вычислении производной функции, например метода градиентного спуска. Причем эта проблема двойственного характера.

Неопределенная производная.

Каким значением производной нужно пользоваться? Когда нам посчастливится столкнуться с точкой излома, метод перестанет работать, т. к. мы не сможем

найти производную. Хотя кто-то в таком случае просто присваивает значение производной в этой точке (так называемый *субградиент* или *субдифференциал*) и идет дальше. И все же, насколько вероятно, что нам действительно не повезет и достанется столь коварная точка? Возможно, нам поможет избежать этого такой инструмент, как численный метод, если только ландшафт функции не будет похож на гористый рельеф Альп (хотя на практике многие похожи).

Нестабильность.

Вторая проблема — нестабильность. В момент пересечения ландшафта функции в такой точке значение производной начинает резко меняться, в связи с чем также резко начинает менять значения и метод, в котором она применяется, что ведет к нестабильности при попытке хоть как-то сблизить их. Например, глядя на рис. 3.6, можно представить, как мы спускаемся по швейцарским Альпам (ландшафт функции потерь) и наш пункт назначения — уютный маленький городок, который мы видим внизу, в долине (участок с наименьшим значением ошибки). И вдруг нас уносит какой-то инопланетянин на другую сторону горы (инопланетянин — это математический метод поиска, который опирается на такое резкое изменение производной), откуда место назначения уже не видно. Теперь внизу виднеются лишь какая-то редкая кривая растительность и довольно узкое ущелье, способное оказаться ловушкой, если нас занесет туда метод. Можно сказать, что в данном случае сближение с первоначальным пунктом назначения у нас нестабильно, если не полностью утеряно.

Рис. 3.6. Пример оптимизации: поход по швейцарским Альпам аналогичен изучению ландшафта функции

Тем не менее функции с такими сингулярностями постоянно применяются в машинном обучении. Они встречаются как в формулах функций обучения нейросетей (выпрямленная линейная единичная функция — кто же их так назвал?), так и в функциях потерь (расстояние по абсолютной величине), а также в нескольких понятиях регуляризации (лассо-регрессия — а эту кто так назвал?).

Среднеквадратическая ошибка в качестве функции потерь для линейной регрессии

Вернемся к основной цели текущего раздела — построению функции ошибки, называемой также *функцией потерь*, способной кодировать величину ошибки, которую в процессе предсказания допустит модель, и сделать ее незначительной.

В случаях линейной регрессии применяется *функция среднеквадратической ошибки* (mean squared error, MSE). Она усредняет квадратичные ошибки удаленности прогнозов от истинных значений m точек данных (какие точки данных сюда нужно включать, мы обсудим совсем скоро).

$$\text{MSE} = \frac{1}{m}\left(\left|y^1_{\text{предсказ}} - y^1_{\text{истина}}\right|^2 + \left|y^2_{\text{предсказ}} - y^2_{\text{истина}}\right|^2 + \ldots + \left|y^m_{\text{предсказ}} - y^m_{\text{истина}}\right|^2\right).$$

Запишем это же выражение более компактно, обозначив сумму как

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left|y^i_{\text{предсказ}} - y^i_{\text{истина}}\right|^2.$$

Теперь мы начнем привыкать к еще более компактным системам представления, или нотации, векторов и матриц, принятых в линейной алгебре. Эта привычка весьма пригодится в полевых условиях и поможет не растеряться с индексами. У нас может сформироваться обманчивое представление, будто мы всё знаем и понимаем, но это довольно быстро обернется жуткими кошмарами. Другим весомым аргументом в пользу использования компактной нотации линейной алгебры является то, что как программное, так и аппаратное обеспечение моделей машинного обучения оптимизировано для матричных и *тензорных* вычислений (только вместо устройства плоской квадратной формы представим себе структуру, состоящую из слоистых матриц, наподобие трехмерной коробки). Более того, такая замечательная область, как численная линейная алгебра, проработала многие потенциальные проблемы, проложив путь к быстрым методам всевозможных матричных вычислений.

Используя нотацию линейной алгебры, можно записать среднеквадратическую ошибку как[3]

$$\text{MSE} = \frac{1}{m}\left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right)^T\left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right) = \frac{1}{m}\left\|\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right\|^2_{l^2}.$$

В последнем равенстве вводится значение *нормы* вектора l^2, которая по определению представляет собой обычный $\sqrt{\text{сумма квадратов компонентов}}$.

Основную идею можно сформулировать так: *наша функция потерь кодирует расхождения между предсказаниями и истиной по точкам данных, включенных в процесс обучения, в виде определенной нормы — математической величины, которая обозначает расстояние*. Значение *нормы* l^2 считается довольно распространенным среди множества других норм, которыми можно воспользоваться.

---

[3] Символ T в верхнем индексе означает транспонирование. — *Прим. ред.*

Система представления: символ вектора для обозначения вектора-столбца

В целях единой нотации на протяжении всего повествования *любой* вектор — это вектор-столбец. Так, если вектор $\vec{v}$ имеет четыре компоненты, символ $\vec{v}$ обозначает $\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$.

Таким образом, транспонированный вектор $\vec{v}$ будет всегда вектором-строкой. В нашем примере транспонированный вектор с четырьмя компонентами будет выглядеть как $\vec{v}^T = \begin{pmatrix} v_1 & v & v_3 & v_4 \end{pmatrix}$.

Можно также воспользоваться обозначением точечного произведения (также называемое скалярным произведением, поскольку в результате умножения двух векторов получается скалярное число). Точечное произведение двух векторов $\vec{a}\vec{b}$ — это то же самое, что $\vec{a}^T\vec{b}$. По сути, $\vec{a}^T\vec{b}$ представляет вектор-столбец в виде матрицы формы, *с длиной вектора, умноженной на 1*, а его транспонирование — в виде матрицы формы, *с 1, умноженной на длину вектора*.

Предположим, что и $\vec{a}$, и $\vec{b}$ имеют четыре компоненты; тогда:

$$\vec{a}^T\vec{b} = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4 = \sum_{i=1}^{4} a_ib_i.$$

Более того,

$$\left\|\vec{a}\right\|_{l^2}^2 = \vec{a}^T\vec{a} = a_1^2 + a_2^2 + a_3^2 + a_4^2.$$

Аналогичным образом,

$$\left\|\vec{b}\right\|_{l^2}^2 = \vec{b}^T\vec{b} = b_1^2 + b_2^2 + b_3^2 + b_4^2.$$

Таким образом, мы везде используем матричную нотацию, указывая буквами со стрелкой только на то, что имеем дело с вектором-столбцом.

Обучающие, валидационные, тестовые поднаборы

Какие точки данных мы включим в нашу функцию потерь? Мы возьмем весь набор данных, его отдельные части или даже одну точку? Для каких точек данных определяется среднеквадратическая ошибка: *обучающего*, *валидационного* или *тестового поднабора*? И что это вообще за поднаборы?

На практике набор данных можно разбить на три поднабора[4].

Обучающий поднабор (training subset).

Поднабор данных для подгонки обучающей функции. Это означает, что точки данных из этого поднабора войдут в функцию потерь (путем подстановки значений их признаков и меток в $y_{\text{предсказ}}$ и $y_{\text{истина}}$ функции потерь).

Валидационный (или проверочный) поднабор (validation subset).

Точки данных этого поднабора используются различными способами.

- В общем случае этот поднабор применяется в *настройке гиперпараметров модели машинного обучения*. Гиперпараметры — это любые параметры модели машинного обучения *кроме* ω-параметров обучающей функции, для которой мы пытаемся решить задачу. В машинном обучении много гиперпараметров, причем их значения влияют на результаты и производительность модели. В качестве примеров можно привести (пока необязательно знать, что это такое): скорость обучения в методе градиентного спуска; гиперпараметр, определяющий ширину поля в методах опорных векторов; процент исходных данных, разбиваемых на обучающие, проверочные и тестовые поднаборы; размер пакета при выполнении рандомизированного пакетного градиентного спуска; гиперпараметры затухания веса, например, используемые в гребневой регрессии, регрессии лассо и регрессии эластичной сети; гиперпараметры, используемые в импульсных методах, среди которых градиентный спуск с импульсом и Adam[5] (они содержат условия, ускоряющие сходимость метода к минимуму, и эти условия умножаются на гиперпараметры, которые необходимо настроить до тестирования и запуска); количество *эпох* в процессе оптимизации (количество проходов по всему обучающему поднабору, видимому оптимизатору); архитектура нейросети (количество слоев, ширина каждого слоя и т. д.).
- Валидационный поднабор также позволяет понять, когда остановить оптимизацию, чтобы *не переусердствовать* с обучающим поднабором.
- Он также служит в качестве тестового набора для сравнения производительности различных моделей машинного обучения на одном и том же наборе данных, например для сравнения производительности модели линейной регрессии, случайного леса или нейросети.

Тестовый поднабор (test subset).

Приняв решение о выборе оптимальной модели (или усреднения, или агрегирования результатов нескольких моделей) и ее обучения, мы пользуемся этим нетронутым поднабором данных в качестве последнего этапа тестирования модели перед ее реализацией в реальном мире. Поскольку ни одна точка данных из это-

[4] Автор использует термин "поднабор" (subset), хотя более распространен термин "набор" (set) — обучающий набор, валидационный набор, тестовый набор. — *Прим. ред.*

[5] Название происходит от *ada*ptive *m*oment (см. **https://arxiv.org/pdf/1412.6980**). — *Прим. ред.*

го поднабора модели не известна (а значит, они не включались в процесс оптимизации), это можно считать наиболее близким аналогом реальной ситуации, что позволяет оценить эффективность нашей модели до того, как мы начнем применять ее к совершенно новым реальным данным.

Подведение итогов

Прежде мы пойдем дальше, давайте кратко подытожим.

- Наша текущая модель машинного обучения называется *линейной регрессией*.
- Наша обучающая функция является линейной и имеет формулу

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5,$$

  где x — признаки; ω — неизвестные веса, или параметры.
- Подставляя в формулу обучающей функции значения признаков для конкретной точки данных, например десятой, мы получаем предсказание нашей модели для нее:

$$y^{10}_{\text{предсказ}} = \omega_0 + \omega_1 x^{10}_1 + \omega_2 x^{10}_2 + \omega_3 x^{10}_3 + \omega_4 x^{10}_4 + \omega_5 x^{10}_5.$$

  Надстрочный индекс 10 указывает на то, что эти значения соответствуют десятой точке данных.
- В качестве функции потерь мы используем функцию среднеквадратической ошибки, имеющую формулу

$$\text{MSE} = \frac{1}{m}\left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right)^T \left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right) = \frac{1}{m} \left\| \vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}} \right\|^2_{l^2}.$$

- Нужно найти такие значения ω, которые минимизируют эту функцию потерь. Поэтому следующим шагом должно быть решение задачи минимизации (оптимизации).

Можно значительно облегчить себе жизнь в процессе оптимизации, снова используя удобную нотацию линейной алгебры (векторы и матрицы). Она позволит включить в формулу функции потерь весь обучающий поднабор данных в виде матрицы и проводить вычисления сразу на нем, а не вычислять каждую точку данных отдельно. Такой небольшой маневр с нотацией избавит нас от множества ошибок и головной боли по поводу громоздких вычислений с огромным количеством компонентов, которые трудно отслеживаются в колоссальных массивах данных.

Итак, запишем сначала предсказание нашей модели, соответствующее каждой точке данных обучающего поднабора:

$$\begin{aligned}
y^1_{\text{предсказ}} &= 1\omega_0 + \omega_1 x^1_1 + \omega_2 x^1_2 + \omega_3 x^1_3 + \omega_4 x^1_4 + \omega_5 x^1_5; \\
y^2_{\text{предсказ}} &= 1\omega_0 + \omega_1 x^2_1 + \omega_2 x^2_2 + \omega_3 x^2_3 + \omega_4 x^2_4 + \omega_5 x^2_5; \\
&\vdots \\
y^m_{\text{предсказ}} &= 1\omega_0 + \omega_1 x^m_1 + \omega_2 x^m_2 + \omega_3 x^m_3 + \omega_4 x^m_4 + \omega_5 x^m_5.
\end{aligned}$$

Можно легко организовать эту систему как

$$\begin{pmatrix} y^1_{\text{предсказ}} \\ y^2_{\text{предсказ}} \\ \vdots \\ y^m_{\text{предсказ}} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \omega_0 + \begin{pmatrix} x^1_1 \\ x^2_1 \\ \vdots \\ x^m_1 \end{pmatrix} \omega_1 + \begin{pmatrix} x^1_2 \\ x^2_2 \\ \vdots \\ x^m_2 \end{pmatrix} \omega_2 + \begin{pmatrix} x^1_3 \\ x^2_3 \\ \vdots \\ x^m_3 \end{pmatrix} \omega_3 + \begin{pmatrix} x^1_4 \\ x^2_4 \\ \vdots \\ x^m_4 \end{pmatrix} \omega_4 + \begin{pmatrix} x^1_5 \\ x^2_5 \\ \vdots \\ x^m_5 \end{pmatrix} \omega_5.$$

И даже лучше:

$$\begin{pmatrix} y^1_{\text{предсказ}} \\ y^2_{\text{предсказ}} \\ \vdots \\ y^m_{\text{предсказ}} \end{pmatrix} = \begin{pmatrix} 1 & x^1_1 & x^1_2 & x^1_3 & x^1_4 & x^1_5 \\ 1 & x^2_1 & x^2_2 & x^2_3 & x^2_4 & x^2_5 \\ \vdots & & & & & \\ 1 & x^m_1 & x^m_2 & x^m_3 & x^m_4 & x^m_5 \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \omega_5 \end{pmatrix}.$$

В левой части уравнения записан вектор $\vec{y}_{\text{предсказ}}$; матрица в правой части — обучающий поднабор $\mathbf{X}$, дополненный вектором единиц; последний вектор в правой части — все аккуратно упакованные в него неизвестные веса. Назовем его $\vec{\omega}$, затем компактно запишем $\vec{y}_{\text{предсказ}}$ с учетом обучающего поднабора и $\vec{\omega}$ как

$$\vec{y}_{\text{предсказ}} = \mathbf{X}\vec{\omega}.$$

Формула функции потерь при среднеквадратической ошибке, которую мы ранее записали как

$$\text{MSE} = \frac{1}{m}\left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right)^T \left(\vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}}\right) = \frac{1}{m} \left\| \vec{y}_{\text{предсказ}} - \vec{y}_{\text{истина}} \right\|^2_{l^2},$$

примет вид

$$\text{MSE} = \frac{1}{m}\left(\mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}}\right)^T \left(\mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}}\right) = \frac{1}{m} \left\| \mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}} \right\|^2_{l^2}$$

Итак, мы готовы найти значение $\vec{\omega}$, минимизирующее четко записанную функцию потерь. Для этого нам придется заглянуть в замечательную, бескрайнюю математическую область — оптимизацию.

Когда обучающие данные содержат сильно коррелированные признаки

Рассмотрим обучающую матрицу (дополненную вектором единиц)

$$\mathbf{X} = \begin{pmatrix} 1 & x^1_1 & x^1_2 & x^1_3 & x^1_4 & x^1_5 \\ 1 & x^2_1 & x^2_2 & x^2_3 & x^2_4 & x^2_5 \\ \vdots & & & & & \\ 1 & x^m_1 & x^m_2 & x^m_3 & x^m_4 & x^m_5 \end{pmatrix},$$

которая появляется в векторе $\vec{y}_{\text{предсказ}} = \mathbf{X}\vec{\omega}$, формулу функции потерь для среднеквадратической ошибки, а также формулу, определяющую неизвестную $\vec{\omega}$ (называемую также *нормальным уравнением*)

$$\vec{\omega} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{y}_{\text{истина}}.$$

Совершенно очевидно, что в случае сильной корреляции двух или более признаков данных (столбцы x) в нашей модели может возникнуть проблема. Поскольку между признаками существует сильная линейная зависимость, можно определить (или почти определить) один из них с помощью линейной комбинации остальных. Таким образом, соответствующие столбцы признаков *не являются линейно независимыми* (и не близки к этому). И это — проблема при работе с матрицами, говорящая о том, что матрица либо не может быть инвертирована, либо она *плохо обусловлена*. Плохо обусловленные матрицы порождают большую неустойчивость в вычислениях, т. к. даже незначительные изменения в обучающих данных (которые необходимо допускать) приводят к ощутимым изменениям параметров модели, вследствие чего ее предсказания становятся ненадежными.

Мы стремимся к тому, чтобы использовать в своих вычислениях только хорошо обусловленные матрицы, для чего нам нужно перекрыть источники плохой обусловленности. При наличии сильно коррелированных признаков одним из возможных вариантов будет включение в модель только одного из них, поскольку остальные не добавляют много информации. В качестве другого решения можно применить методы снижения размерности, например анализ главных компонент, с которым мы познакомимся в *главе 11*. Набор данных Fish Market имеет сильно коррелированные признаки, которые рассматриваются в Jupyter Notebook.

В связи с этим важно отметить, что некоторые модели машинного обучения, такие как деревья решений и случайные леса (о них речь пойдет ниже), не подвержены влиянию коррелированных признаков, в то время как на другие модели, в том числе и на рассматриваемую модель линейной регрессии, а также модели логистической регрессии и машины опорных векторов, о которых мы поговорим позднее, эти признаки оказывают негативное воздействие. Что касается моделей нейросетей, то даже несмотря на то, что они могут *обучаться* корреляциям, связанным с признаками данных, в процессе обучения, работают они все-таки лучше, когда подобные избыточные признаки учтены заранее, не говоря уже о снижении в этой связи вычислительных затрат и экономии времени.

Оптимизация

Оптимизация означает поиск оптимального, наилучшего, максимального, минимального или экстремального решения.

Мы записали линейную обучающую функцию

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5$$

и оставили неизвестными шесть ее параметров: ω_0, ω_1, ω_2, ω_3, ω_4, ω_5. Задача состоит в поиске значений, при которых наша обучающая функция *наилучшим образом подходит к обучающему поднабору данных*, причем слово *"наилучшим"* задается функцией потерь. Последняя определяет, насколько далеко от истины предсказание, сделанное обучающей функцией модели. Нам не нужно, чтобы эта функция была большой, поэтому мы будем решать задачу минимизации.

Мы не будем попусту тратить время на перебор всевозможных значений ω, пока не отыщется комбинация, дающая минимальные потери. Но даже если попытаться это проделать, мы все равно не будем знать, на чем остановиться, поскольку другие, *лучшие* значения нам останутся неизвестны. Для того чтобы воспользоваться математическими свойствами функции потерь, необходимо предварительно познакомиться с ее ландшафтом. Можно провести аналогию с походом по Швейцарским Альпам с повязкой на глазах и походом без повязки, к тому же с подробной картой в придачу (на рис. 3.7 показан гористый рельеф Швейцарских Альп). Так и в нашем случае: чтобы не искать минимизаторы функции потерь с завязанными глазами, воспользуемся *оптимизацией*. Оптимизация — это превосходная область математики, предлагающая различные методы эффективного поиска и нахождения оптимизаторов функций и соответствующих им оптимальных значений.

В этой и последующих главах задача оптимизации имеет вид

$$\min_{\vec{\omega}} \text{ функция потерь}.$$

Рис. 3.7. Пример оптимизации: поход по Швейцарским Альпам аналогичен изучению ландшафта функции. Пункт назначения — дно самой низкой долины (*минимизация*) или вершина самого высокого пика (*максимизация*). Требуются две вещи: координаты точек минимизации/максимизации и высота ландшафта в этих точках

Для текущей модели линейной регрессии это

$$\min_{\vec{\omega}} \frac{1}{m}\left(\mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}}\right)^T\left(\mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}}\right) = \min_{\vec{\omega}} \frac{1}{m}\left\|\mathbf{X}\vec{\omega} - \vec{y}_{\text{истина}}\right\|_{l^2}^2.$$

Занимаясь математикой, нужно не забывать о том, что именно нам известно и что именно мы ищем. В противном случае мы рискуем попасть в ловушку круговой логики. В формуле, приведенной выше, нам известно:

m

Количество экземпляров в обучающем поднаборе.

X

Обучающее подмножество, дополненное вектором единиц.

$\vec{y}_{\text{истина}}$

Вектор меток, соответствующих обучающему поднабору.

Требуется найти:

- минимизацию $\vec{\omega}$;
- минимальное значение функции потерь в процессе минимизации $\vec{\omega}$.

Выпуклые и невыпуклые ландшафты

Наиболее простыми для решения являются *линейные* функции и уравнения. К сожалению, большинство функций (и уравнений), с которыми мы имеем дело на практике, — нелинейные. С другой стороны, и не так уж это печально, поскольку линейная жизнь — плоская, скучная, немотивированная, неинтересная. В некоторых случаях, когда мы работаем с нелинейной функцией, мы *линеаризуем* ее возле важных для нас точек. Идея заключается в том, что даже если функция целиком окажется нелинейной, ее можно аппроксимировать линейной функцией в интересующей нас области. Другими словами, в очень малой окрестности нелинейная функция может выглядеть и вести себя линейно, даже если эта окрестность окажется бесконечно малой. Здесь можно провести аналогию с тем, что когда мы смотрим на Землю, находясь на ее поверхности, она выглядит (и ведет себя с точки зрения вычисления расстояний и т. д.) плоской, а ее нелинейную форму мы можем увидеть только с высоты. Для того чтобы линеаризовать функцию в окрестности точки, мы должны ее аппроксимировать касательным пространством в этой же окрестности (в случае функции одной переменной это касательная прямая, двух переменных — касательная плоскость, трех и более переменных — касательная гиперплоскость). Для этого требуется вычислить одну производную функции по всем ее переменным, что даст нам угол наклона (определяющий наклон) аппроксимирующего плоского пространства.

Плохая новость заключается в том, что линеаризации в окрестности одной точки может оказаться недостаточно, и потребуется использовать линейные аппроксимации в нескольких точках. К счастью, это вполне осуществимо, поскольку с вычислительной точки зрения нам нужно всего лишь вычислить одну производную в нескольких точках. Это подводит нас к другим функциям (после линейных), с которыми *проще всего* работать — кусочно-линейным функциям, которые линейны, но только в кусочных конструкциях, или линейны, но только не в изоли-

рованных точках или местах. Такими функциями оперирует *линейное программирование*, где оптимизируемые функции — линейные, а границы областей, в которых происходит оптимизация, — кусочно-линейные (они являются пересечениями полупространств).

В случае когда стоит задача оптимизации, лучше всего подойдут либо линейные функции (в этом нам поможет область линейного программирования), либо выпуклые функции (в этом случае мы не будем беспокоиться о том, что застрянем в локальных минимумах, и у нас будут подходящие неравенства, которые помогут в анализе).

Нужно отметить еще один важный тип функций, встречающийся в машинном обучении, — функция, которая является максимумом двух или более выпуклых функций. Такие функции всегда выпуклые. Напомним, что линейные функции — плоские, а значит, одновременно и выпуклые, и вогнутые. В этом есть смысл, поскольку некоторые функции определяются как максимумы линейных функций, и можно сказать, что они негарантированно линейные (кусочно-линейные), но гарантированно выпуклые. То есть, даже если мы и теряем линейность, когда берем максимум линейной функции, все равно это компенсируется выпуклостью.

В качестве примера функции, определяемой как максимум двух линейных функций, можно привести функцию ReLU, которая применяется в нейросетях в качестве нелинейной функции активации: $\mathrm{ReLU}(x) = \max(0, x)$. Другим примером является функция потери петли, используемая в машинах опорных векторов: $H(x) = \max(0, 1 - tx)$, где t — либо 1, либо -1.

Следует отметить, что минимум семейства выпуклых функций не является гарантированно выпуклым и может быть двойственным. Но их максимум обязательно будет выпуклым.

Существует еще одна зависимость между линейностью и выпуклостью. Если имеется выпуклая функция (нелинейная, т. к. линейная была бы тривиальной), то максимум всех линейных функций, лежащих ниже ее, в точности равен ей. Другими словами, выпуклость заменяет линейность в том смысле, что, когда линейность недоступна, но доступна выпуклость, можно заменить нашу выпуклую функцию на максимум всех линейных функций, график которых лежит ниже графика нашей функции (рис. 3.8).

Рис. 3.8. Выпуклая функция равна максимуму всех своих касательных

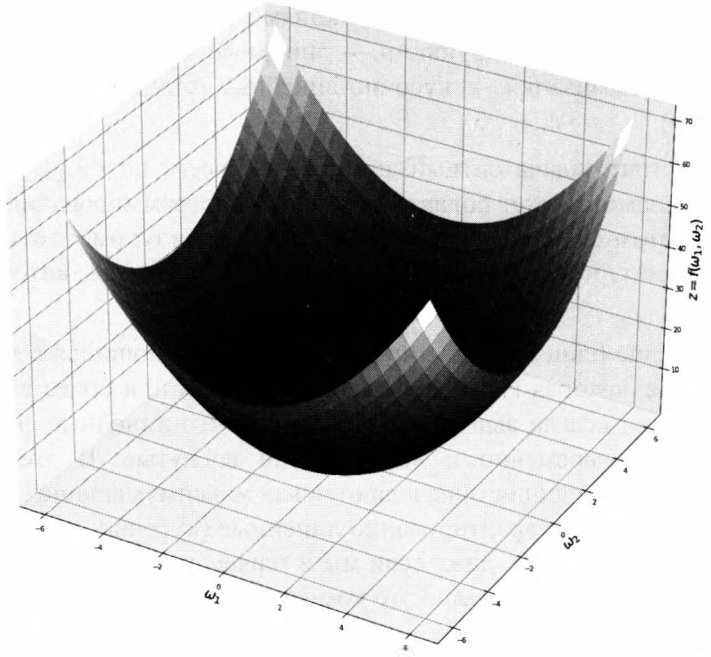


Рис. 3.9. Ландшафт выпуклой функции хорошо подходит для решения задач минимизации. Мы не боимся застрять в локальных минимумах, поскольку для выпуклой функции любой локальный минимум является также глобальным минимумом

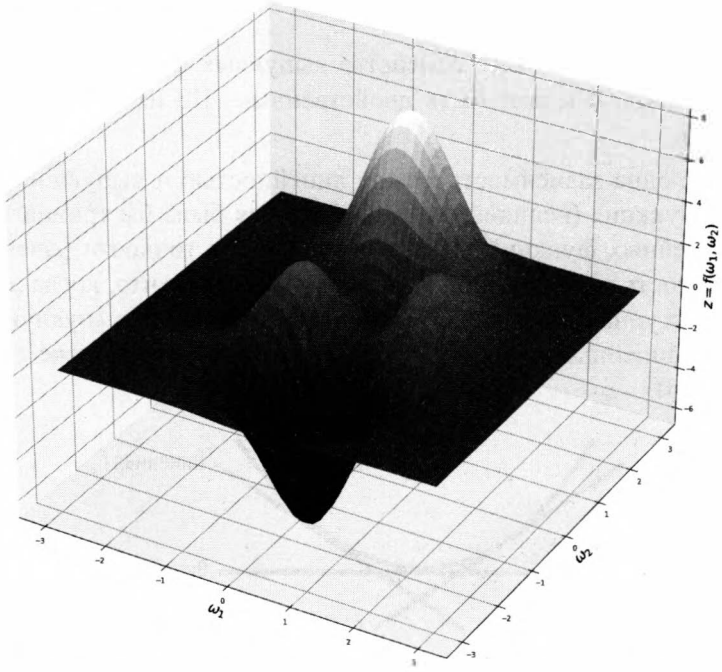


Рис. 3.10. Ландшафт невыпуклой функции имеет пики, долины и седловые точки. В таком ландшафте можно увязнуть в локальном минимуме, так и не найдя глобальный

Напомним, что в любой точке график выпуклой функции лежит выше графика ее касательной, а касательные линейны. Это позволяет напрямую использовать простоту линейных функций при наличии выпуклых. Учитывая максимум *всех* касательных, мы получаем равенство, а максимум касательных в нескольких точках — только аппроксимацию.

На рис. 3.9 и 3.10 показаны, соответственно, общие ландшафты нелинейных выпуклых и невыпуклых функций. В целом ландшафт выпуклой функции хорошо подходит для решения задач минимизации. Можно не бояться застрять в локальных минимумах, поскольку для выпуклой функции любой локальный минимум является также глобальным минимумом. Ландшафт невыпуклой функции имеет пики, долины и седловые точки, что таит в себе риск никогда не прийти к глобальному минимуму, увязнув в локальных.

Наконец, важно понимать разницу между выпуклой функцией, выпуклым множеством и задачей выпуклой оптимизации, которая оптимизирует выпуклую функцию на выпуклом множестве.

Поиск минимизаторов функций

В общем случае существует два подхода к нахождению минимумов (и/или максимумов) функций. Компромисс обычно заключается в следующем:

1. Вычислением только одной производной и медленной сходимостью к минимуму (хотя существуют методы ускорения сходимости). Такие методы называются *градиентными*. Градиент — это одна производная функции нескольких переменных. Например, наша функция потерь является функцией нескольких ω (или одного вектора $\vec{\omega}$).
2. Вычислением двух производных (гораздо более затратное в вычислительном отношении, что является большим недостатком, особенно при наличии тысяч параметров) и более быстрым сведением их к минимуму. Можно слегка сэкономить вычислительные затраты за счет аппроксимации второй производной вместо ее точного вычисления. Методы второй производной называются методами *Ньютона*. В них фигурирует *гессиан* (матрица вторых производных) или его аппроксимация.

Можно всегда ограничиться вычислением двух производных.

Но почему в поиске оптимизаторов так важны первая и вторая производные? Если кратко, то первая производная содержит информацию о том, насколько быстро функция возрастает или убывает в точке (поэтому, если проследить ее направление, можно подняться к максимуму или опуститься к минимуму), а вторая производная содержит информацию о том, как *изогнется* график функции — вверх или вниз.

По-прежнему в основе лежит ключевая идея исчисления: минимизация (и/или максимизация) происходит либо в *критических точках* (определяемых как точки, в которых одна производная функции либо равна нулю, либо отсутствует), либо в граничных точках. Поэтому мы должны искать оптимизаторы *как* в граничных точ-

ках (если наше пространство поиска имеет границу), *так и* во внутренних критических точках.

Как найти критические точки внутри поискового пространства?

Первый способ.

Выполним следующие действия:

- находим производную функции (мы делали это на уроках математики);
- задаем ее равной нулю (каждый сможет написать символы "равно" и "ноль");
- находим значения ω, при которых производная равна нулю (не лучшее действие!).

Для функций, производные которых линейны, в том числе для нашей функции потерь среднеквадратической ошибки, найти такие значения ω несложно. Так, область линейной алгебры создавалась преимущественно для решения систем линейных уравнений. Область численной линейной алгебры предназначена для решения реалистичных и больших систем линейных уравнений с преобладанием плохой обусловленности. В нашем распоряжении имеется множество инструментов (и программных пакетов) для работы с линейными системами.

В случае же нелинейных уравнений поиск решений — это совсем другая история. Он больше напоминает игру "попал — не попал", причем преимущественно с промахами. Проиллюстрируем разницу между решением линейного и нелинейного уравнения на небольшом примере.

Решение линейного уравнения.

Найти такое значение ω, при котором $0,002\omega - 5 = 0$.

Решение: переносим 5 в правую часть уравнения, делим на 0,002 и получаем $\omega = 5 / 0,002 = 2500$. Готово.

Решение нелинейного уравнения.

Найти такое значение ω, при котором $0,002\sin\omega - 5\omega^2 + e^{\omega} = 0$.

Решение: здесь все ясно — требуется численный метод! (Графическая аппроксимация решения этого нелинейного уравнения приведена на рис. 3.11.)

Существует множество численных методов, предназначенных исключительно для решения нелинейных уравнений (а также целые области, посвященные численному решению нелинейных обыкновенных и дифференциальных уравнений). С помощью этих методов можно найти приближенные решения, а затем установить границы, определяющие степень удаленности численных решений от точных аналитических решений. В таких случаях выстраивается последовательность, которая в определенных условиях приближается к аналитическому решению. Некоторые методы позволяют делать это быстрее и лучше всего подходят для решения определенных задач.

Второй способ.

Другой способ — следовать направлению градиента, спускаясь к минимуму или поднимаясь к максимуму.

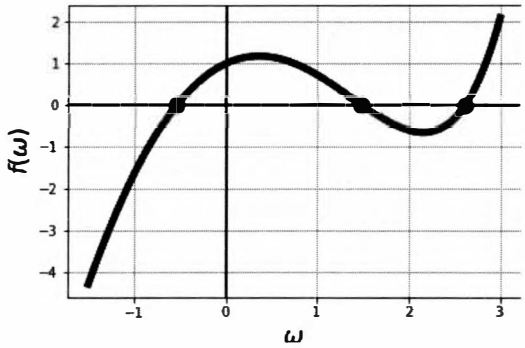


Рис. 3.11. Решение нелинейных уравнений сопряжено с определенными трудностями. На рисунке изображены график функции $f(\omega) = 0{,}002\sin\omega - 5\omega^2 + e^{\omega}$ и аппроксимация трех ее корней (точки, в которых $f(\omega) = 0$)

Для иллюстрации градиентных методов представим себе спуск с горы (или скоростной спуск на лыжах, если метод ускорен либо имеет импульс). Мы начинаем со случайной точки в пространстве поиска, которая и задает начальный уровень высоты ландшафта функции. Затем метод переносит нас в новую точку пространства поиска, которая, как мы надеемся, установит новый уровень высоты — *ниже* того, откуда мы пришли. Тогда можно сказать, что мы *спустились*. Повторим это действие, и в идеале данная последовательность точек приблизится к минимуму искомой функции, насколько ландшафт функции будет благоприятствовать этому. Безусловно, если в ландшафте функции окажется множество пиков и долин, то нам будет важно, с чего начинать — или, другими словами, инициализировать, — поскольку существует вероятность того, что мы спустимся в совершенно другую долину — совсем не туда, где по итогу хотим оказаться. Мы можем попасть не в глобальный, а в локальный минимум.

Графики выпуклых и ограниченных снизу функций напоминают по форме салатницу, так что в случаях с ними можно не волноваться, что мы застрянем в локальных минимумах и удалимся от глобальных. Здесь возникает другой повод для беспокойства — когда салатница оказывается слишком узкой, метод работает очень медленно. Мы рассмотрим это подробно в *главе 4*.

Оба способа считаются весьма практичными и популярными. Иногда не остается другого выбора, кроме как воспользоваться одним из них, в зависимости от того, насколько быстро он подойдет к решению нашей конкретной задачи, насколько *регулярна* функция, которую мы пытаемся оптимизировать (сколько у нее удобных для анализа производных), и т. д. В других случаях это просто дело вкуса. В случаях с функцией потерь среднеквадратической ошибки линейной регрессии работают оба способа, так что воспользуемся первым, но только потому, что для *всех* остальных функций потерь в книге мы будем применять методы градиентного спуска.

Следует отметить, что аналогия между методом и спуском с горы является отличной, но несколько ошибочной. Во время реального спуска с горы мы физически находимся в одном и том же трехмерном пространстве, что и горный ландшафт,

т. е. мы находимся на определенной высоте и можем спуститься на более низкую высоту, даже с завязанными глазами или в густой туман, делая по шагу вниз. Мы ощущаем высоту и двигаемся вниз по склону. Численные методы спуска, в свою очередь, не ищут минимум в той же размерности пространства, что и ландшафт функции. Поиск ведется на отметке на одно измерение *ниже* ландшафта (рис. 3.12). Это значительно усложняет спуск к минимуму, т. к. на уровне отметки можно перемещаться из одной точки в другую, при этом не зная, какой уровень высоты существует над нами, пока в определенной точке мы не вычислим саму функцию и определим ее высоту. Поэтому наш метод может случайно переместить нас из одной точки поверхности с определенной высотой над нами в другую, с *большей* высотой, а значит, еще дальше от минимума. Таким образом, на уровне отметки важно определить направление, быстро *снижающее* высоту функции, и то, насколько далеко мы можем двигаться в этом направлении на уровне отметки (размер шага), *снижая при этом высоту функции над нами*. Размер шага также называется *гиперпараметром скорости обучения*, с которым мы будем постоянно встречаться, применяя метод спуска.

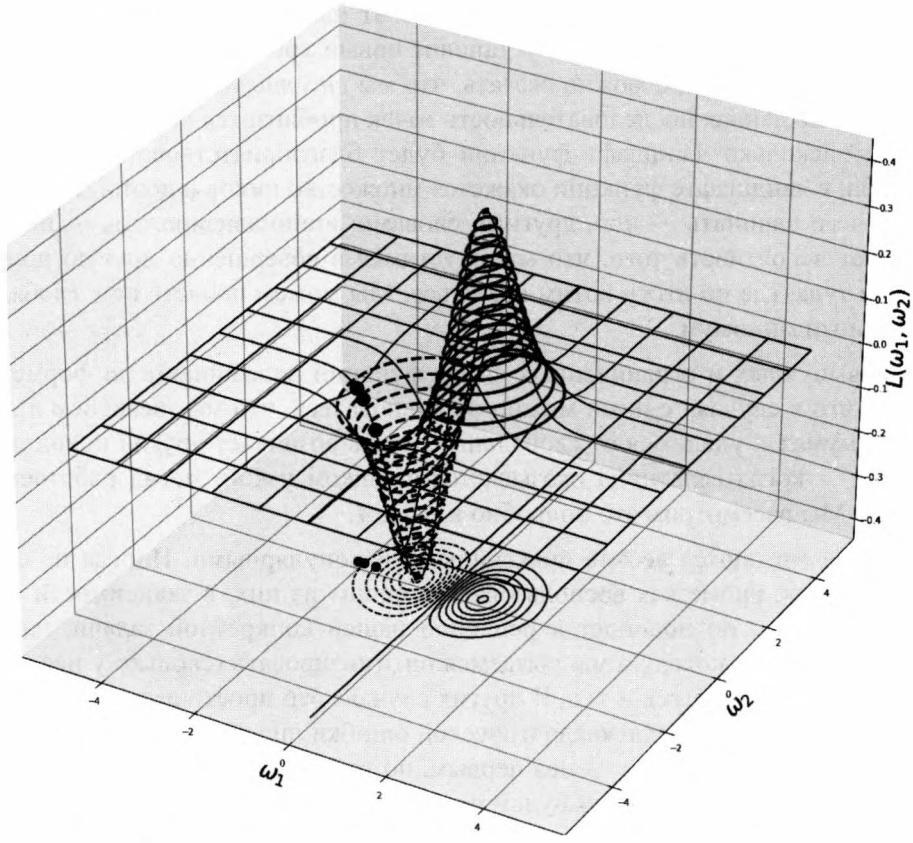


Рис. 3.12. Поиск минимума происходит не на ландшафте функции, а на уровне отметки

Вернемся к нашей основной цели — найти наилучшее значение $\vec{\omega}$ для обучающей функции. Итак, необходимо минимизировать функцию потерь среднего квадрата ошибки, используя первый способ: берем одну производную функции потерь и задаем ее равной нулю, после чего определяем значение вектора $\vec{\omega}$. Для этого нужно мастерски оперировать с выражениями линейной алгебры. Так что сначала вернемся к основам исчисления.

Об исчислении в двух словах

На первых занятиях по исчислениям мы знакомимся с функциями одной переменной ($f(\omega)$), их графиками и методами исчисления в определенных точках. Затем мы изучаем важнейшую операцию математического анализа — предел. Исходя из понятия предела, мы определяем непрерывность и прерывность функций, производную в точке $f'(\omega)$ (предел наклонов секущих через точку) и интеграл по области (предел сумм мини-областей, определяемых функцией по области). В конце курса рассматривается фундаментальная теорема исчисления, связывающая интегрирование и дифференцирование как обратные операции. Одним из ключевых свойств производной является то, что она определяет скорость возрастания или убывания функции в некоторой точке, а значит, играет важную роль в нахождении минимума и/или максимума функции внутри ее области (граничные точки обособлены).

В курсе по исчислению функций многих переменных, который преподается обычно на третьем году обучения, много концепций заимствовано из анализа функций одной переменной, в том числе значимость производной, называемой *градиентом* из-за наличия нескольких переменных, в поиске внутренних минимумов и/или максимумов. Градиент $\nabla\left(f(\vec{\omega})\right)$ от $f(\vec{\omega})$ — это производная функции по вектору переменных $\vec{\omega}$.

В глубоком обучении неизвестные веса организованы не в векторы, а в матрицы, следовательно, необходимо взять производную функции $f(\mathbf{W})$ по матрице переменных $\mathbf{W}$.

Для целей искусственного интеллекта требуется вычислить производную функции потерь, в которую встроена функция обучения. Согласно *цепному правилу для производных*, нужно также найти производную обучающей функции по значениям ω.

Продемонстрируем это на простом примере исчисления функции одной переменной и мгновенного перехода к получению производных из выражений линейной алгебры.

Пример одномерной оптимизации

Задача: найти минимизатор (минимизаторы) и минимальное значение (при наличии) функции $f(\omega) = 3 + (0{,}5\omega - 2)^2$ на отрезке $[-1; 6]$.

Одним из невероятно долгих способов решения этой задачи будет перебор бесконечно большого числа значений ω в интервале от –1 до 6 и выбор тех ω, которые дают наименьшее значение *f*. Другой способ — воспользоваться знаниями из области исчислений о том, что оптимизация (минимизация и/или максимизация) происходит либо в критических точках (где производной не существует или она равна нулю), либо в граничных точках (рис. 3.13).

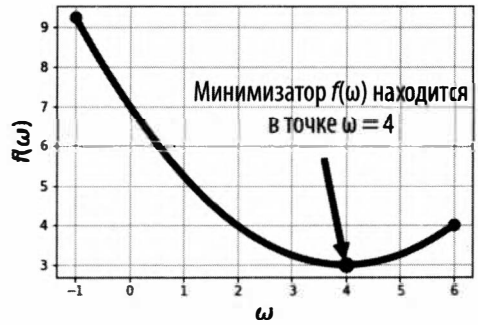


Рис. 3.13. Минимальное значение функции $f(\omega) = 3 + (0{,}5\omega - 2)^2$ на отрезке [–1; 6] равно 3 и находится в критической точке ω = 4. В этой критической точке производная функции равна нулю, а значит, если мы проведем касательную линию, она будет горизонтальной

В нашем случае у нас имеются граничные точки –1 и 6, так что в первую очередь вычисляем функцию этих точках: $f(-1) = 3 + (0{,}5 \cdot (-1) - 2)^2 = 9{,}25$ и $f(6) = 3 + (0{,}5 \cdot 6 - 2)^2 = 4$. Очевидно, граничная точка –1 не является минимизатором, т. к. $f(6) < f(-1)$, следовательно, она выбывает из борьбы, и теперь только граничная точка 6 соперничает с внутренней критической точкой (точками). Для того чтобы найти критические точки, определим производную функции внутри отрезка [–1; 6]: $f'(\omega) = 0 + 2 \cdot (0{,}5\omega - 2) \cdot 0{,}5 = 0{,}5\omega - 2$. Установим ее равной нулю и получим $0{,}5\omega - 2 = 0$, откуда $\omega = 4$. Таким образом, мы нашли только одну критическую точку $\omega = 4$ внутри отрезка [–1; 6]. Конкретно в этой точке значение функции равно $f(4) = 3 + (0{,}5 \cdot 4 - 2)^2 = 3$. Поскольку значение *f* здесь наименьшее, мы можем обозначить очевидного победителя первенства по минимизации, а именно $\omega = 4$ с минимальным значением *f*, равным 3.

Часто используемые производные выражений линейной алгебры

Производные лучше всего вычислять непосредственно в выражениях с векторами и матрицами, не разбивая их на составляющие. Приведем два популярных способа.

1. Если *a* и ω — скалярные величины, при этом *a* постоянна, то производная $f(\omega) = a\omega$ равна $f'(\omega) = a$. Если $\vec{a}$ и $\vec{\omega}$ — векторы (одинаковой длины) и записи $\vec{a}$ постоянны, то градиент $f(\vec{\omega}) = \vec{a}^T\vec{\omega}$ равен $\nabla f(\vec{\omega}) = \vec{a}$. Аналогично, градиент $f(\vec{\omega}) = \vec{\omega}^T\vec{a}$ равен $\nabla f(\vec{\omega}) = \vec{a}$.

2. Если s — скалярная и постоянная величина, а ω — скалярная величина, то производная квадратичной функции $f(\omega)=s\omega^2$ равна $f'(\omega)=2s\omega$. В аналогичном высокоразмерном случае если $\mathbf{S}$ — симметричная матрица с постоянными записями, то функция $f(\vec{\omega})=\vec{\omega}^T\mathbf{S}\vec{\omega}$ является квадратичной и ее градиент равен $\nabla f(\vec{\omega})=2\mathbf{S}\vec{\omega}$.

Минимизация функции потерь среднеквадратической ошибки

Наконец, мы готовы минимизировать функцию потерь среднеквадратической ошибки:

$$L(\vec{\omega})=\frac{1}{m}\left(\mathbf{X}\vec{\omega}-\vec{y}_{\text{истина}}\right)^T\left(\mathbf{X}\vec{\omega}-\vec{y}_{\text{истина}}\right).$$

Раскроем это выражение, затем установим его градиент равным нулю:

$$\begin{aligned}L(\vec{\omega})&=\frac{1}{m}\left(\left(\mathbf{X}\vec{\omega}\right)^T-\vec{y}^T_{\text{истина}}\right)\left(\mathbf{X}\vec{\omega}-\vec{y}_{\text{истина}}\right)=\\&=\frac{1}{m}\left(\vec{\omega}^T\mathbf{X}^T-\vec{y}^T_{\text{истина}}\right)\left(\mathbf{X}\vec{\omega}-\vec{y}_{\text{истина}}\right)=\\&=\frac{1}{m}\left(\vec{\omega}^T\mathbf{X}^T\mathbf{X}\vec{\omega}-\vec{\omega}^T\mathbf{X}^T\vec{y}_{\text{истина}}-\vec{y}^T_{\text{истина}}\mathbf{X}\vec{\omega}+\vec{y}^T_{\text{истина}}\vec{y}_{\text{истина}}\right)=\\&=\frac{1}{m}\left(\vec{\omega}^T\mathbf{S}\vec{\omega}-\vec{\omega}^T\vec{a}-\vec{a}^T\vec{\omega}+\vec{y}^T_{\text{истина}}\vec{y}_{\text{истина}}\right),\end{aligned}$$

где на последнем шаге задаем $\mathbf{X}^T\mathbf{X}=\mathbf{S}$ и $\mathbf{X}^T\vec{y}_{\text{истина}}=\vec{a}$. Затем задаем градиент последнего выражения по ω равным нулю. Вычисляя градиент, воспользуемся знаниями из области дифференцирования выражений линейной алгебры:

$$\nabla L(\vec{\omega})=\frac{1}{m}\left(2\mathbf{S}\vec{\omega}-\vec{a}-\vec{a}+0\right)=\vec{0}.$$

Теперь несложно вычислить $\vec{\omega}$:

$$\frac{1}{m}\left(2\mathbf{S}\vec{\omega}-2\vec{a}\right)=\vec{0}.$$

Таким образом,

$$2\mathbf{S}\vec{\omega}=2\vec{a},$$

что дает:

$$\vec{\omega}=\mathbf{S}^{-1}\vec{a}.$$

Теперь, вспомним, что мы задавали $\mathbf{X}^T\mathbf{X}=\mathbf{S}$ и $\vec{a}=\mathbf{X}^T\vec{y}_{\text{истина}}$, и перепишем минимизацию $\vec{\omega}$ применительно к обучающему набору $\mathbf{X}$ (дополненному единицами) и соответствующему вектору меток $\vec{y}_{\text{истина}}$:

$$\vec{\omega}=\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{y}_{\text{истина}}.$$

Для набора данных Fish Market это будет выглядеть следующим образом (см. Jupyter Notebook):

$$\vec{\omega} = \begin{pmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \\ \omega_5 \end{pmatrix} = \begin{pmatrix} -475{,}19929130109716 \\ 82{,}84970118 \\ -28{,}85952426 \\ -28{,}50769512 \\ 29{,}82981435 \\ 30{,}97250278 \end{pmatrix}.$$

Не умножайте большие матрицы друг на друга — это слишком затратно. Умножайте их на векторы

Всеми силами старайтесь избежать умножения матриц друг на друга — умножайте их на *векторы*. Например, в уравнении $\vec{\omega} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{y}_{\text{истина}}$ в первую очередь вычисляем $\mathbf{X}^T\vec{y}_{\text{истина}}$, оставив без внимания $\vec{\omega} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}$. В качестве обходного пути выступит решение линейной системы $\mathbf{X}\vec{\omega} = \vec{y}_{\text{истина}}$ с помощью псевдоинверсии $\mathbf{X}$ (см. Jupyter Notebook). Мы познакомимся с псевдоинверсией в *главе 11*, а пока лишь отметим, что с ее помощью можно инвертировать матрицы (что эквивалентно делению на них), не имеющие обратной матрицы.

Итак, мы нашли вектор весов $\vec{\omega}$, который дает наилучшее соответствие между нашими обучающими данными и обучающей функцией линейной регрессии:

$$f\left(\vec{\omega}; \vec{x}\right) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4 + \omega_5 x_5.$$

С помощью аналитического метода (вычислив градиент функции потерь и установив его равным нулю) мы получили решение, заданное нормальным уравнением. Это один из тех редких случаев, когда удается вывести аналитическое решение. Далее в поисках минимизирующего $\vec{\omega}$ мы будем применять только численные методы.

Избегайте чрезмерной подгонки обучающих данных

Мы получили $\vec{\omega} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{y}_{\text{истина}}$, что дает значения $\vec{\omega}$, при которых обучающая функция *лучше всего подходит* к обучающим данным; тем не менее в случае чрезмерной подгонки обучающая функция будет улавливать не только сигнал, но и помехи в данных. Поэтому во избежание таких случаев необходимо модифицировать решение или даже саму задачу минимизации. В этом нам помогут *регуляризация* и одна из ее форм — *ранняя остановка*. Мы коснемся их в *главе 4*.

Итак, долгий путь к регрессии позади. По сути, мы — еще новички, и поэтому нам пришлось пройти исчисление и линейную алгебру. В дальнейшем модели машинного обучения — логистическая регрессия, машины опорных векторов, деревья решений, случайные леса — будут рассматриваться намного быстрее, т. к. в различных функциях мы будем оперировать одними и теми же понятиями.

Логистическая регрессия: разделение на два класса

Логистическая регрессия применяется преимущественно в задачах классификации. Начнем с бинарной классификации (классификация на два класса, например онкология/отсутствие онкологии, безопасно/небезопасно для детей, вероятность возврата/невозврата кредита и т. д.). Затем обобщим модель на задачу классификации по нескольким категориям (например, классифицируем изображения рукописных цифр на 0, 1, 2, 3, 4, 5, 6, 7, 8 или 9). И снова мы имеем ту же самую математическую конструкцию:

1. Обучающая функция.
2. Функция потерь.
3. Оптимизация.

Обучающая функция

Аналогично линейной регрессии обучающая функция логистической регрессии вычисляет линейную комбинацию признаков и добавляет постоянный член смещения, причем не просто выдает результат, но и пропускает его через *логистическую функцию*, график которой приведен на рис. 3.14, а формула имеет следующий вид:

$$\sigma(s) = \frac{1}{1+e^{-s}}.$$

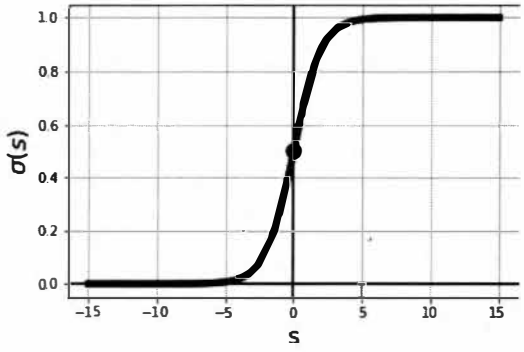


Рис. 3.14. График логистической функции $\sigma(s) = \frac{1}{1+e^{-s}}$.

Важно отметить, что эта функция вычисляется при любом s и всегда выдает число от 0 до 1, следовательно, ее результат можно интерпретировать как вероятность

Данная функция принимает значения только между 0 и 1, поэтому результат ее вычисления можно интерпретировать как вероятность принадлежности точки данных к определенному классу. Если результат меньше 0,5, точка данных относится к первому классу, а если больше 0,5 — к другому. Число 0,5 — это *пороговое значение*, при котором принимается решение о классификации точки данных.

Таким образом, в данном случае обучающая функция представляет собой линейную комбинацию признаков плюс смещение, которая составляется сначала с помощью логистической, а затем с помощью пороговой функции:

$$y = \text{Порог}\left(\sigma\left(\omega_0 + \omega_1 x_1 + \ldots + \omega_n x_n\right)\right).$$

Как и в случае линейной регрессии, ω — это неизвестные, для которых требуется оптимизировать функцию потерь. Точно так же число неизвестных равно числу признаков данных плюс один член смещения. Например, в задачах классификации изображений насчитываются тысячи признаков, т. к. признаком является каждый пиксел.

Функция потерь

Создадим *функцию потерь* и решим задачу классификации. Представим, что нам нужно наложить штрафы на неверно классифицированные точки обучающих данных. В нашем наборе размеченных данных принадлежность экземпляра к классу обозначается как $y_{\text{истина}} = 1$, а отсутствие такой принадлежности — как $y_{\text{истина}} = 0$.

Нам нужно, чтобы обучающая функция выдавала $y_{\text{истина}} = 1$ для обучающих экземпляров с принадлежностью к определенному классу (чьи $y_{\text{истина}}$ равны 1). Верные значения ω обеспечивают высокое значение t (результат линейной комбинации), входящее в логистическую функцию, что позволяет с высокой вероятностью получить положительные экземпляры и преодолеть порог 0,5 до значения $y_{\text{предсказ}} = 1$.

Следовательно, если на шаге линейной комбинации плюс смещение при $y_{\text{истина}} = 1$ получается низкое значение t, к нему нужно добавить штрафы.

Аналогично, верные значения весов обеспечивают низкое значение t, входящее в логистическую функцию для обучающих экземпляров без принадлежности к какому-либо классу (значение $y_{\text{истина}} = 0$). Поэтому, если на шаге линейной комбинации плюс смещение при $y_{\text{истина}} = 0$ получается высокое значение t, следует также наложить штрафы.

Итак, как же найти такую функцию потерь, способную "штрафовать" неверно классифицированную точку обучающих данных? Причем как ложноположительные, так и ложноотрицательные значения.

Напомним, что классификационная модель выдает по итогу либо 1, либо 0.

- Продумаем вычислительную функцию, способную вознаграждать 1 и штрафовать 0: $-\log(s)$[6] (рис. 3.15, *слева*).
- Продумаем вычислительную функцию, способную вознаграждать 1 и штрафовать 0: $-\log(1-s)$ (рис. 3.15, *справа*).

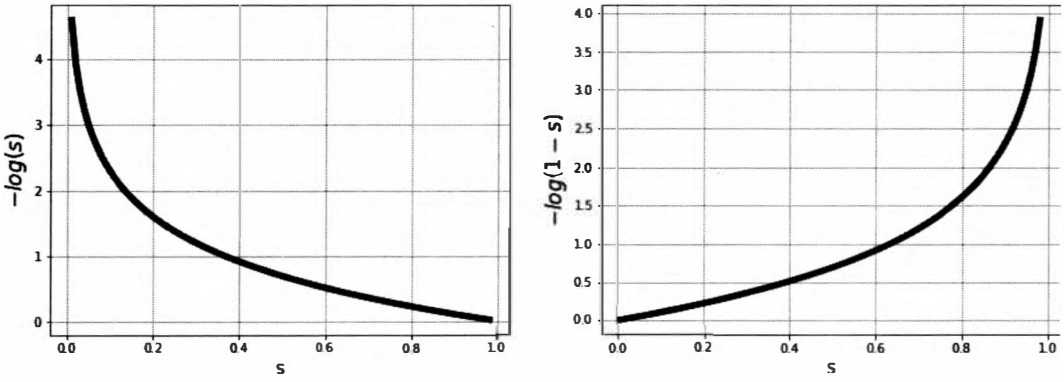


Рис. 3.15. *Слева*: график функции $f(s) = -\log(s)$. Эта функция присваивает большие значения числам в окрестности 0 и малые значения числам в окрестности 1. *Справа*: график функции $f(s) = -\log(1-s)$. Она присваивает высокие значения числам в окрестности 1 и низкие значения числам в окрестности 0

Теперь рассмотрим результат логистической функции $\sigma(s)$ для текущего выбора значений ω.

- При $\sigma(s)$ меньше 0,5 (модель дала предсказание $y_{\text{предсказ}} = 0$) получаем $y_{\text{истина}} = 1$ (ложноотрицательное значение), на модель накладывается штраф $-\log(\sigma(s))$. При $\sigma(s)$ больше 0,5, т. е. предсказание модели $y_{\text{предсказ}} = 1$ (истинноположительное значение), величина $-\log(\sigma(s))$ будет небольшой, значит, штраф будет небольшим.
- Аналогичным образом, при $\sigma(s)$ больше 0,5 получаем $y_{\text{истина}} = 0$ (ложноположительное), на модель накладывается штраф $-\log(1-\sigma(s))$. Опять же, в случае истинноотрицательного предсказания штраф будет небольшим.

[6] С математической точки зрения, правильно логарифмическую функцию записывать так: $\log_a x$ — логарифм x по основанию a; $\lg x$ — десятичный логарифм (логарифм x по основанию 10); $\ln x$ — натуральный логарифм (логарифм x по основанию e (2,7182)). В англоязычной литературе основание a, по которому берется логарифм, как правило, опускается в записи, когда значение этого основания неважно, а существенным моментом является именно применение функции логарифма. Поэтому здесь и далее запись $\log x$ тождественна записи $\log_a x$. — *Прим. ред.*

Таким образом, стоимость ошибочной классификации одного обучающего экземпляра $\left(x_1^i, x_2^i, ..., x_n^i; y_{\text{истина}}\right)$ можно записать как:

$$\text{стоимость} = \begin{Bmatrix} -\log\left(\sigma(s)\right), & \text{если } y_{\text{истина}} = 1; \\ -\log\left(1-\sigma(s)\right), & \text{если } y_{\text{истина}} = 0 \end{Bmatrix} =$$
$$= -y_{\text{истина}} \log\left(\sigma(s)\right) - \left(1 - y_{\text{истина}}\right)\log\left(1-\sigma(s)\right).$$

Наконец, функция потерь представляет собой среднюю стоимость по m обучающим экземплярам, что дает нам формулу для популярной *функции потерь перекрестной энтропии*:

$$L\left(\vec{\omega}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_{\text{истина}}^i \log\left(\sigma\left(\omega_0 + \omega_1 x_1^i + ... + \omega_n x_n^i\right)\right) + \right.$$
$$\left. + \left(1 - y_{\text{истина}}^i\right)\log\left(1 - \sigma\left(\omega_0 + \omega_1 x_1^i + ... + \omega_n x_n^i\right)\right)\right).$$

Оптимизация

В отличие от линейной регрессии, если мы решили минимизировать функцию потерь, задав $\nabla L(\omega) = 0$, то для ω нет замкнутой формулы. Есть и хорошая новость, которая заключается в том, что эта функция является выпуклой, поэтому методом градиентного спуска, о котором говорится в *главе 4* (или, по-другому, стохастический или мини-пакетный градиентный спуск), минимум гарантированно найдется (если *скорость обучения* не слишком велика и мы ждем достаточно долго).

Регрессия softmax: классификация по нескольким категориям

Идею логистической регрессии можно легко обобщить на классификацию по нескольким категориям. Известным примером такой задачи небинарной классификации служит классификация изображений 10 рукописных цифр 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9 на основе набора данных MNIST (**https://oreil.ly/HQL5F**)[7]. В этом наборе данных содержится 70 000 изображений рукописных цифр (выборки изображений приведены на рис. 3.16), разделенных на обучающий поднабор из 60 000 изображений и тестовый поднабор из 10 000 изображений. Каждое изображение помечено одной из 10 цифр, указывающей на принадлежность к определенному классу. Кроме того, в наборе содержатся результаты работы большого количества классифицирующих моделей, среди которых линейные классификаторы, *k ближайших соседей, деревья решений, машины опорных векторов* с различными *ядрами*, нейросети с различными архитектурами, а также ссылки на соответствующие статьи и годы их

[7] На момент написания этой главы этот сайт мог запрашивать учетные данные, но вы можете получить к нему доступ, используя инструкции из Hacker News (**https://oreil.ly/Csj-0**).

публикации. При этом интересно наблюдать, как по прошествии времени улучшается производительность методов по мере их развития.

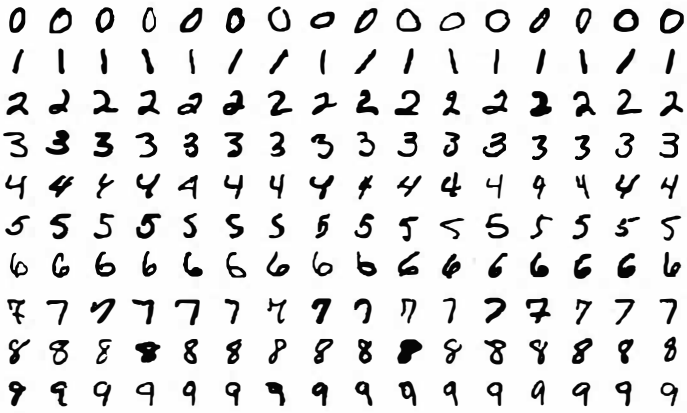


Рис. 3.16. Образцы изображений из набора данных MNIST (исходное изображение)

Не путайте классификацию на категории с многовыходными моделями

С помощью регрессии softmax за один раз получается предсказать одну категорию, так что ее нельзя использовать для классификации, например, пяти человек на одном и том же изображении. Зато с ее помощью можно проверить, кто изображен на конкретной фотографии в Facebook: я, мои сестра, брат, муж или дочка. В модель регрессии softmax можно передавать фотографию, на которой присутствует только один из нас пятерых, в противном случае классификация модели будет менее очевидной. Следовательно, категории должны быть взаимоисключающими. Поэтому, когда Facebook на одном и том же снимке автоматически отмечает пять человек, он использует не модель регрессии softmax, а многовыходную модель.

Предположим, что у нас имеются характеристики точки данных, и мы хотим использовать эту информацию, чтобы отнести эту точку данных к одной из числа k возможных категорий. Теперь несложно понять, какими будут функция обучения, функция потерь и процесс оптимизации.

Признаки данных изображений

В полутоновых изображениях признаком считается каждая интенсивность пиксела, поэтому изображения обычно содержат тысячи признаков. Полутоновые изображения часто представляются в виде двумерных матриц чисел, где элементы матрицы соответствуют интенсивностям пикселов. В цветных изображениях присутствуют три канала — красный, зеленый и синий, а каждый канал, в свою очередь, представлен в виде двумерной матрицы чисел, причем каналы наслаиваются друг на друга, образуя три

слоя двумерных матриц. Такая структура называется *тензором*. Описание процесса обработки изображений можно найти в блокноте на странице книги в GitHub (**https://github.com/halanelson/Essential-Math-For-AI**) на примере работы с полутоновыми и цветными изображениями в Python.

Обучающая функция

Первый шаг всегда один и тот же — линейно объединить признаки и добавить постоянный член смещения. В случае логистической регрессии, когда у нас было только два класса, мы подставляли результат в логистическую функцию по формуле:

$$\sigma(s) = \frac{1}{1+e^{-s}} = \frac{1}{1+\frac{1}{e^{s}}} = \frac{e^{s}}{1+e^{s}} = \frac{e^{s}}{e^{0}+e^{s}}.$$

Мы интерпретировали результат как вероятность принадлежности или непринадлежности точки данных к интересующему нас классу. Важно отметить, что мы переписали формулу для логистической функции в виде $\sigma(s) = \frac{e^{s}}{e^{0}+e^{s}}$, чтобы подчеркнуть, что она отражает две вероятности — по одной для каждого класса. Другими словами, $\sigma(s)$ дает вероятность того, что точка данных относится к интересующему нас классу, а $1-\sigma(s) = \frac{e^{0}}{e^{0}+e^{s}}$ — не относится.

Если у нас не два, а несколько классов, то для одной и той же точки данных мы несколько раз повторяем один и тот же процесс: по одному разу для каждого класса. Каждый класс имеет свое смещение и набор весов, линейно объединяющие признаки, поэтому вычисляем k различных линейных комбинаций плюс смещение для точки данных со значениями признаков x_1, x_2, ..., x_n:

$$s^1 = \omega_0^1 + \omega_1^1 x_1 + \omega_2^1 x_2 + \ldots + \omega_n^1 x_n;$$
$$s^2 = \omega_0^2 + \omega_1^2 x_1 + \omega_2^2 x_2 + \ldots + \omega_n^2 x_n;$$
$$\vdots$$
$$s^k = \omega_0^k + \omega_1^k x_1 + \omega_2^k x_2 + \ldots + \omega_n^k x_n.$$

Формируем полезные привычки

Нужно взять за правило постоянно отслеживать, сколько неизвестных ω попадает в формулу обучающей функции. Напомним, что эти ω мы находим путем минимизации функции потерь. Другой полезной привычкой будет эффективный и последовательный способ их организации в модели (в виде вектора, матрицы и т. д.). В случае с регрессией softmax, когда для каждой точки данных имеется k классов и n признаков, мы имеем в итоге $k \times n$ значений ω для линейных комбинаций, затем добавляем k смещений и получаем в общей сложности $k \times n + k$ неизвестных ω. Например, если мы

применяем регрессионную модель softmax для классификации изображений в наборе данных MNIST (**https://oreil.ly/wJfei**), содержащем рукописные цифры, причем каждое изображение имеет размер 28 × 28 пикселов, что означает 784 признака, и хотим классифицировать их на 10 категорий, то в итоге нам придется выполнить оптимизацию по 7850 значениям ω. В моделях линейной и логистической регрессии требовалось оптимизировать всего $n + 1$ неизвестных ω.

Далее мы подставляем каждый из этих k результатов в *функцию softmax*, которая обобщает логистическую функцию с двух на несколько классов, и мы также интерпретируем это как вероятность. Формула для функции softmax выглядит следующим образом:

$$\sigma\left(s^j\right) = \frac{e^{s^j}}{e^{s^1} + e^{s^2} + \ldots + e^{s^k}}.$$

Таким образом, одна и та же точка данных получит k вероятностных оценок — по одной оценке на каждый класс. Наконец, мы классифицируем точку данных как принадлежащую к тому классу, где она получила наибольшую вероятностную оценку.

Суммируя все вышесказанное, мы получаем окончательную формулу обучающей функции, которую можно теперь использовать для классификации (т. е. после нахождения оптимальных значений ω путем минимизации соответствующей функции потерь):

$$y = j \text{ так, что } \sigma\left(\omega_0^j + \omega_1^j x_1 + \ldots + \omega_n^j x_n\right)$$

является максимальным.

Важно отметить, что для этой обучающей функции достаточно ввести признаки данных (значения x), и она вернет номер одного класса j.

Логистическая функция, функция softmax и статистическая механика

Примечательно, что логистическая функция и функция softmax вычисляют вероятности точно так же, как в статистической механике с помощью *функции разбиения* вычисляют вероятность нахождения системы в определенном состоянии.

Функция потерь

Мы вывели функцию потерь перекрестной энтропии для логистической регрессии

$$L\left(\vec{\omega}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^i_{\text{истина}} \log\left(\sigma\left(\omega_0 + \omega_1 x_1^i + \ldots + \omega_n x_n^i\right)\right) + \right.$$
$$\left. + \left(1 - y^i_{\text{истина}}\right)\log\left(1 - \sigma\left(\omega_0 + \omega_1 x_1^i + \ldots + \omega_n x_n^i\right)\right)\right),$$

применяя

$$\text{стоимость} = \begin{Bmatrix} -\log(\sigma(s)), & \text{если } y_{\text{истина}} = 1; \\ -\log(1-\sigma(s)), & \text{если } y_{\text{истина}} = 0 \end{Bmatrix} =$$
$$= -y_{\text{истина}} \log(\sigma(s)) - (1 - y_{\text{истина}}) \log(1-\sigma(s)).$$

Обобщим ту же логику на несколько классов. Мы будем использовать обозначение $y_{\text{истина},\,i} = 1$, если точка данных относится к *i*-му классу, а в остальных случаях — ноль. Тогда стоимость неверной классификации точки данных примет вид:

$$\text{стоимость} = \begin{Bmatrix} -\log\left(\sigma\left(s^1\right)\right), & \text{если } y_{\text{истина},1} = 1; \\ -\log\left(\sigma\left(s^2\right)\right), & \text{если } y_{\text{истина},2} = 1; \\ -\log\left(\sigma\left(s^3\right)\right), & \text{если } y_{\text{истина},3} = 1; \\ \vdots & \\ -\log\left(\sigma\left(s^k\right)\right), & \text{если } y_{\text{истина},k} = 1 \end{Bmatrix} =$$
$$= -y_{\text{истина},1} \log\left(\sigma\left(s^1\right)\right) - \ldots - y_{\text{истина},k} \log\left(\sigma\left(s^k\right)\right).$$

После усреднения по всем *m* точкам данных в обучающем наборе получаем *обобщенную функцию потерь перекрестной энтропии* путем обобщения ее с двух на несколько классов:

$$L(\vec{\omega}) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^i_{\text{истина},1} \log\left(\sigma\left(\omega_0^1 + \omega_1^1 x_1^i + \ldots + \omega_n^1 x_n^i\right)\right) + \right.$$
$$+ y^i_{\text{истина},2} \log\left(\sigma\left(\omega_0^2 + \omega_1^2 x_1^i + \ldots + \omega_n^2 x_n^i\right)\right) + \ldots +$$
$$\left. + y^i_{\text{истина},k} \log\left(\sigma\left(\omega_0^k + \omega_1^k x_1^i + \ldots + \omega_n^k x_n^i\right)\right)\right).$$

Оптимизация

Итак, у нас есть формула для функции потерь, и можно искать ее минимизирующие значения ω. Как и в большинстве случаев с функциями потерь, которые нам предстоит разобрать, для минимумов этой функции не существует явной формулы в терминах обучающего набора и целевых меток, так что для поиска минимумов придется довольствоваться численными методами, в частности такими как градиентный спуск и стохастический градиентный или мини-пакетный градиентный спуск (*см. главу 4*). Опять же, в процессе минимизации обобщенной функции потерь перекрестной энтропии ее выпуклость работает в нашу пользу, поэтому мы гарантированно найдем нужные значения ω.

Перекрестная энтропия и теория информации

Концепция перекрестной энтропии заимствована из теории информации. Мы подробнее остановимся на ней далее в этой главе при обсуждении деревьев решений. Пока же запомним следующую величину, где p — вероятность события:

$$\log\left(\frac{1}{p}\right) = -\log(p).$$

При маленьких значениях p эта величина возрастает, следовательно, она позволяет выражать количественно бóльшую *неожиданность менее вероятных* событий.

Встраивание моделей в последний слой нейросети

Модель линейной регрессии строит свои прогнозы путем соответствующей линейной комбинации признаков данных с последующим добавлением смещения. В моделях логистической регрессии и регрессии softmax классификация осуществляется посредством линейного объединения признаков данных, добавления смещения и передачи результата в функцию оценки вероятности. В этих простых моделях возможна только линейная комбинация признаков данных, так что с точки зрения выявления потенциально важных нелинейных взаимодействий между признаками данных они считаются слабыми. В моделях нейросетей нелинейные *функции активации* включены в обучающие функции, причем на нескольких слоях, и, следовательно, эти модели лучше справляются с выявлением нелинейных и более сложных взаимосвязей. Последний слой нейросети называется выходным слоем. Непосредственно перед ним расположен слой, который выдает несколько признаков высшего порядка и вводит их в последний слой. Если нам нужно, чтобы сеть классифицировала данные по нескольким категориям, то последний слой можно сделать слоем регрессии softmax, если по двум категориям — слоем логистической регрессии, а если мы хотим, чтобы сеть предсказывала числовые значения, то сделаем последний слой слоем регрессии. Мы увидим примеры в *главе 5*.

Другие популярные методы машинного обучения и ансамбли методов

После регрессии и логистической регрессии перейдем к машинному обучению и познакомимся с идеями, лежащими в основе некоторых наиболее популярных методов классификации и регрессии. Довольно мощными и самыми популярными среди них считаются *машины опорных векторов*, *деревья решений* и *случайные леса*, способные решать задачи как классификации, так и регрессии. И здесь возникает естественный вопрос: когда использовать тот или иной метод машинного обуче-

ния, включая линейную и логистическую регрессию, а затем и нейросети? Как выбрать правильный метод и обосновать свои выводы и предсказания? В этих вопросах нам поможет разобраться математический анализ моделей машинного обучения.

Математический анализ каждого метода, включая наборы данных, для которых он лучше всего подходит, только сейчас стал привлекать серьезное внимание на фоне увеличения объема ресурсов, выделяемых на исследования в области искусственного интеллекта, машинного обучения и науки о данных, и на сегодняшний день сложилась практика опробования каждого метода на одном и том же наборе данных и использования самого результативного. Это предполагает, что в нашем распоряжении имеются необходимые вычислительные и временны́е ресурсы для опробования различных методов машинного обучения. Еще лучше, если у нас есть время и ресурсы для обучения моделей машинного обучения (здесь идеально подойдут параллельные вычисления) — в таком случае лучше воспользоваться *ансамблевыми методами*. Они объединяют результаты различных моделей машинного обучения либо путем усреднения, либо путем голосования, что — как ни парадоксально, но математически обоснованно — дает лучшие результаты по сравнению с отдельными показателями наиболее эффективных моделей, *причем даже тогда, когда последние считаются слабыми!*

В качестве примера ансамбля деревьев решений можно привести случайный лес.

При построении прогнозов на основе ансамблей появляются такие отраслевые термины, как *бэггинг* (или *бутстрэп-агрегирование*), *пэстинг*, *бустинг* (например, *адаптивный бустинг AdaBoost* и *градиентный бустинг*), *стекинг*, *случайные патчи*. Бэггинг и пэстинг обучают *одну и ту же* модель машинного обучения на различных случайных поднаборах обучающего набора. Причем бэггинг производит выборку экземпляров из обучающего набора с заменой, а пэстинг — без замены. *Случайные патчи* также делают выборку из пространства признаков, одновременно обучая модель машинного обучения на случайном поднаборе признаков.

В частности, это хорошо помогает в работе с наборами данных, содержащими огромное количество признаков, например с изображениями (где каждый пиксел является признаком). Помимо голосования и усреднения, *стекинг* обучает механизм предсказания ансамбля.

Машины опорных векторов

Машины опорных векторов (support vector machine, SVM) являются чрезвычайно популярным методом машинного обучения, способным решать задачи классификации и регрессии как с линейными (плоскими), так и с нелинейными (кривыми) границами принятия решений.

При классификации данный метод стремится отделить размеченные данные с максимально возможным запасом, в результате чего получается не тонкая линия, а оптимальный *магистральный* канал разделения. Поясним, как машины опорных векторов классифицируют размеченные экземпляры данных с учетом структуры главы: обучающая функция, функция потерь, оптимизация.

Обучающая функция

И снова мы линейно комбинируем признаки точки данных с неизвестными весами ω и добавляем смещение ω_0. Затем пропускаем ответ через функцию sign. Если линейная комбинация признаков плюс смещение является положительным числом, то возвращается 1 (или классифицируется как первый класс), если отрицательным, то возвращается –1 (или классифицируется как второй класс). Таким образом, формула для обучающей функции примет вид:

$$f(\vec{\omega}; \vec{x}) = \text{sign}\left(\vec{\omega}^T \vec{x} + \omega_0\right).$$

Функция потерь

Требуется создать функцию потерь, которая будет накладывать штраф на неправильно классифицированные точки. В случае с логистической регрессией мы использовали функцию потерь перекрестной энтропии. В работе с машинами опорных векторов мы будем опираться на *кусочно-линейную функцию потерь (потеря шарнира)*:

$$\max\left(0, 1 - y_{\text{истина}}\left(\vec{\omega}^T \vec{x} + \omega_0\right)\right).$$

Рассмотрим, как функция потери шарнира штрафует ошибки классификации. Прежде всего, напомним, что $y_{\text{истина}}$ равно либо 1, либо –1 в зависимости от того, к какому классу относится точка данных — положительному или отрицательному.

- Если в определенной точке данных $y_{\text{истина}} = 1$, но $\vec{\omega}^T \vec{x} + \omega_0 < 0$, то функция обучения классифицирует ее неверно и дает $y_{\text{предсказ}} = -1$, при этом значение функции потери шарнира будет равно $1 - 1 \cdot \left(\vec{\omega}^T \vec{x} + \omega_0\right) > 1$, на что в задачах минимизации накладывается крупный штраф.
- Если, с другой стороны, $y_{\text{истина}} = 1$, а $\vec{\omega}^T \vec{x} + \omega_0 > 0$, то функция обучения классифицирует его правильно и дает $y_{\text{предсказ}} = 1$. Однако функция потери шарнира построена таким образом, что она все равно будет штрафовать нас при $\vec{\omega}^T \vec{x} + \omega_0 < 1$, а ее значение будет равно $1 - 1 \cdot \left(\vec{\omega}^T \vec{x} + \omega_0\right)$, что уже меньше 1, но все еще больше 0.
- Только при $y_{\text{истина}} = 1$ и $\vec{\omega}^T \vec{x} + \omega_0 > 1$ (обучающая функция все равно правильно классифицирует эту точку и даст $y_{\text{предсказ}} = 1$) значение функции потери шарнира будет равно 0, т. к. оно будет максимумом между 0 и отрицательной величиной.
- Аналогичным образом работает логика при $y_{\text{истина}} = -1$. Функция потери шарнира будет накладывать огромный штраф за неверное предсказание и небольшой — за правильное, если оно не имеет слишком большого *отступа* от *делителя нуля* (отступ больше 1). Функция вернет 0 только в том случае, если предсказание верно, а точка удалена от делителя нуля на расстояние больше 1.

- Важно отметить, что для делителя нуля справедливо уравнение $\vec{\omega}^T\vec{x}+\omega_0=0$, а для краев отступа — уравнения $\vec{\omega}^T\vec{x}+\omega_0=-1$ и $\vec{\omega}^T\vec{x}+\omega_0=1$. Расстояние между краями отступа легко вычисляется как $\frac{2}{\|\omega\|_2}$. Таким образом, если нам нужно увеличить отступ, мы должны уменьшить $\|\omega\|_2$, следовательно, этот член необходимо ввести в функцию потерь и функцию потери шарнира, которая штрафует как неверно классифицированные точки, так и точки, находящиеся в пределах отступа.

Если мы усредним функцию потери шарнира по всем m точкам данных в обучающем наборе и добавим $\|\omega\|_2^2$, то получим формулу для функции потерь, которая обычно используется в машинах опорных векторов:

$$L(\vec{\omega})=\frac{1}{m}\sum_{i=1}^{m}\max\left(0,\ 1-y^i_{\text{истина}}\left(\vec{\omega}^T\vec{x}^i+\omega_0\right)\right)+\lambda\|\omega\|_2^2.$$

Оптимизация

Наша задача — найти $\vec{\omega}$, минимизирующий функцию потерь. Рассмотрим вкратце эту функцию.

- Она содержит два члена: $\frac{1}{m}\sum_{i=1}^{m}\max\left(0,\ 1-y^i_{\text{истина}}\left(\vec{\omega}^T\vec{x}^i+\omega_0\right)\right)$ и $\lambda\|\vec{\omega}\|_2^2$. Если в задаче оптимизации имеется больше одного члена, то, скорее всего, они будут конкурировать, в том смысле, что те же самые значения ω, при которых первый член будет небольшим и, следовательно, подходящим, могут сделать второй член большим и, следовательно, неподходящим. Таким образом, при поиске $\vec{\omega}$, оптимизирующего их сумму, между этими двумя терминами возникает конкурентная борьба.

- Параметр λ, присутствующий в $\lambda\|\vec{\omega}\|_2^2$, является примером гиперпараметра модели, который в процессе обучения можно настраивать на этапе валидации. Важно отметить, что, регулируя значение λ, можно задать ширину отступа: при большом значении λ оптимизатор будет отбирать $\vec{\omega}$ с крайне низким $\|\vec{\omega}\|_2^2$ с целью компенсировать это большое λ, и первому члену функции потерь будет уделено меньше внимания. Однако запомним: чем меньше $\|\omega\|_2$, тем больше отступ!

- Член $\lambda\|\vec{\omega}\|_2^2$ можно также считать *регуляризационным членом*, который мы рассмотрим в *главе 4*.

- Данная функция потерь является выпуклой и ограничена снизу значением 0, так что задача ее минимизации не будет слишком сложной — нам не нужно волноваться, что мы застрянем в локальных минимумах. Первый член имеет сингулярность, но, как уже говорилось, в точке сингулярности можно определить его субградиент, а затем применить метод спуска.

Некоторые задачи оптимизации можно переформулировать, и вместо решения исходной *первичной* задачи мы решаем *двойственную*! Как правило, одна из них решается легче, чем другая. Двойственную задачу можно считать еще одной задачей оптимизации, существующей в параллельной вселенной первичной задачи. Эти вселенные пересекаются в оптимизаторе. Следовательно, решение одной задачи автоматически приводит к решению другой. Мы знакомимся с двойственностью, когда изучаем оптимизацию. Особый интерес и огромное применение находят линейная и квадратичная оптимизации, которые называются также линейным и квадратичным программированием. На текущий момент имеется задача минимизации:

$$\min_{\vec{\omega}} \frac{1}{m}\sum_{i=1}^{m}\max\left(0,\ 1-y^i_{\text{истина}}\left(\vec{\omega}^T\vec{x}^i+\omega_0\right)\right)+\lambda\|\omega\|_2^2.$$

Она служит примером квадратичного программирования и имеет формулировку двойственной задачи, что позволяет оптимизировать ее проще, чем первичную (особенно при колоссальном количестве признаков):

$$\max_{\vec{\alpha}} \sum_{j=1}^{m}\alpha_j-\frac{1}{2}\sum_{j=1}^{m}\sum_{k=1}^{m}\alpha_j\alpha_k y^j_{\text{истина}}y^k_{\text{истина}}\left(\left(\vec{x}^j\right)^T\vec{x}^k\right)$$

в условиях ограничений $\alpha_j \geq 0$ и $\sum_{j=1}^{m}\alpha_j y^j_{\text{истина}}=0$. В процессе изучении первичных и двойственных задач запись этой формулы обычно не вызывает затруднений, поэтому в целях непрерывности повествования пропустим ее вывод.

Квадратичное программирование — довольно развитая область, к тому же существует множество программных пакетов для решения этой задачи. После того как мы найдем максимизирующее $\vec{\alpha}$, можно найти вектор $\vec{\omega}$, минимизирующий первичную задачу, используя $\vec{\omega}=\sum_{j=1}^{m}\alpha_j y^i_{\text{истина}}\vec{x}^j$. Получив $\vec{\omega}$, с помощью обученной функции можно классифицировать новые точки данных:

$$f\left(\vec{x}_{\text{новая}}\right)=\text{sign}\left(\vec{\omega}^T\vec{x}_{\text{новая}}+\omega_0\right)=\text{sign}\left(\sum_{j=1}\alpha_j y^i\left(\vec{x}^j\right)^T\vec{x}_{\text{новая}}+\omega_0\right).$$

Если мы хотим обойтись без квадратичного программирования, существует другой довольно быстрый метод, называемый *координатным спуском*, который решает двойственную задачу и отлично работает с большими наборами данных с колоссальным количеством признаков.

Ядерный трюк

Теперь можно перенести те же идеи на нелинейную классификацию. Прежде всего отметим важное замечание по поводу двойственной задачи: точки данных появляются только парами, точнее, только в виде скалярного произведения, а именно $\left(\vec{x}^j\right)^T\vec{x}^k$. Точно так же они появляются в обученной функции лишь в виде скалярного произведения.

Это простое наблюдение позволяет творить чудеса.

- Если мы найдем функцию $K\left(\vec{x}^j, \vec{x}^k\right)$, действующую на пары точек данных, и окажется, что она выводит скалярное произведение *преобразования* точек данных в некоторое пространство более высокой размерности, а именно $K\left(\vec{x}^j, \vec{x}^k\right) = \overrightarrow{\phi\left(\vec{x}^j\right)}^T \overrightarrow{\phi\left(\vec{x}^k\right)}$ (нам даже не нужно знать, что такое реальное преобразование $\vec{\phi}$ в более высокую размерность), то мы сможем решить ту же самую двойственную задачу в более высокой размерности, заменив в ее формуле скалярное произведение на $K\left(\vec{x}^j, \vec{x}^k\right)$.

- Интуиция нам подсказывает, что данные, нелинейно разделяемые в низших измерениях, почти всегда линейно разделяемы в высших измерениях. Ядерный трюк решает задачу линейной классификации в более высоких измерениях *без* преобразования каждой точки. Ядро само оценивает точечное произведение преобразованных данных, не преобразуя их.

В качестве примеров ядерных функций можно привести (заметим, что в них отсутствуют преобразования $\vec{\phi}$):

- $K\left(\vec{x}^j, \vec{x}^k\right) = \left(\left(\vec{x}^j\right)^T \vec{x}^k\right)^2$;
- полиномиальное ядро: $K\left(\vec{x}^j, \vec{x}^k\right) = \left(1 + \left(\vec{x}^j\right)^T \vec{x}^k\right)^d$;
- гауссово ядро: $K\left(\vec{x}^j, \vec{x}^k\right) = e^{-\gamma\left|x_j - x_k\right|^2}$.

Деревья решений

Исходя из основной идеи данной главы, что всё есть функция, можно утверждать, что дерево решений, по сути, является функцией, принимающей на вход булевы переменные (переменные, которые могут принимать только *истинные* (1) или *ложные* (0) значения), например: признак > 5, признак = солнечный, признак = мужчина и т. д. На выходе получается решение: одобрить кредит, классифицировать как COVID-19, вернуть 25 и т. д. Мы не выполняем сложения или умножения булевых переменных, а пользуемся логическими операторами ИЛИ (or), И (and), НЕ (not).

Но что делать, если признаки в исходном наборе данных не представлены в виде булевых переменных? Тогда их нужно преобразовать в булевы переменные до ввода в модель прогнозирования. Например, дерево решений на рис. 3.17 обучено на наборе данных Fish Market. Это дерево регрессии. Оно принимает исходные дан-

ные, хотя представляющая его функция фактически оперирует новыми переменными — исходными признаками данных, преобразованными в булевы переменные:

1. $a1 = (\text{Width} \leq 5{,}117)$
2. $a2 = (\text{Length3} \leq 59{,}55)$
3. $a3 = (\text{Length3} \leq 41{,}1)$
4. $a4 = (\text{Length3} \leq 34{,}9)$
5. $a5 = (\text{Length3} \leq 27{,}95)$
6. $a6 = (\text{Length3} \leq 21{,}25)$

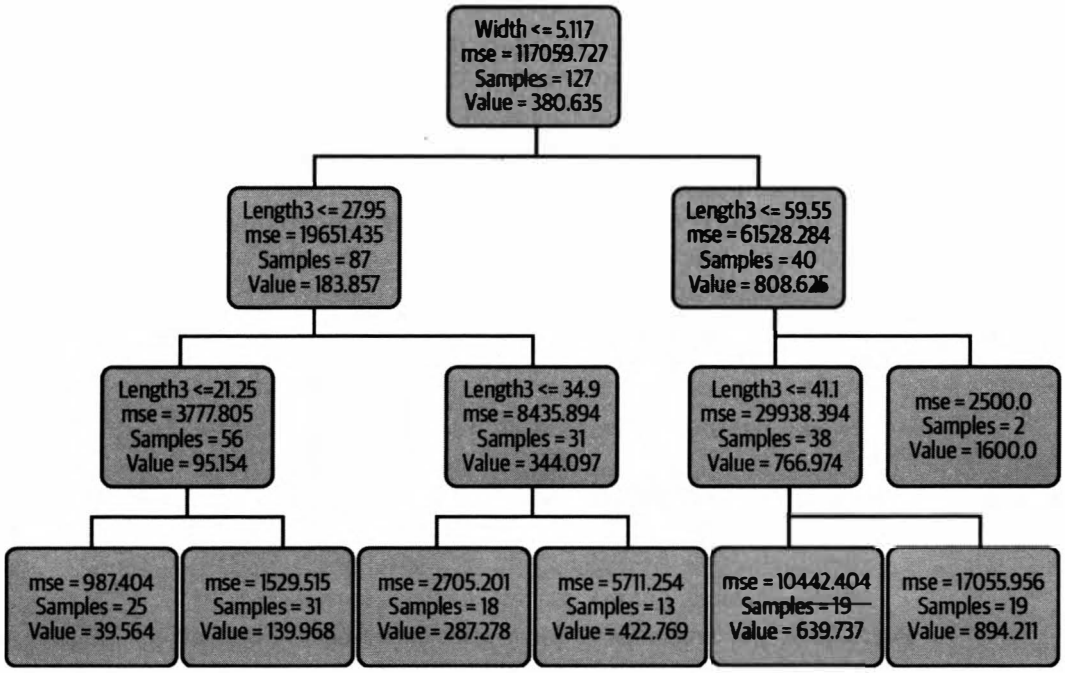


Рис. 3.17. Регрессионное дерево решений, построенное на наборе данных Fish Market. Подробности см. в блокноте Jupyter Notebook

Тогда функция, представляющая дерево решений на рис. 3.17, примет такой вид:

$$f(a1, a2, a3, a4, a5, a6) = (a1 \text{ И } a5 \text{ И } a6) \times 39{,}584 + (a1 \text{ И } a5 \text{ И НЕ } a6) \times 139{,}968 +$$
$$+ (a1 \text{ И НЕ } a5 \text{ И } a4) \times 287{,}278 + (a1 \text{ И НЕ } a5 \text{ И НЕ } a4) \times 422{,}769 +$$
$$+ (\text{НЕ } a1 \text{ И } a2 \text{ И } a3) \times 639{,}737 + (\text{НЕ } a1 \text{ И } a2 \text{ И НЕ } a3) \times 824{,}211 +$$
$$+ (\text{НЕ } a1 \text{ И НЕ } a2) \times 1600.$$

Важно отметить, что в отличие от обучающих функций, с которыми мы уже встречались в этой главе, данная функция не имеет параметров ω, которые нужно вычислить. Такая модель называется *непараметрической*. Она не фиксирует форму функции заранее, что позволяет ей гибко расти вместе с данными, или, другими

словами, адаптироваться к ним. Разумеется, такая высокая адаптивность к данным сопряжена с высоким риском чрезмерной подгонки. К счастью, существуют способы решения этой проблемы, и мы перечислим здесь некоторые из них, не вдаваясь в подробности: отсечение ветвей дерева после его выращивания, ограничение количества слоев, установка минимального количества экземпляров данных на один узел, использование не одного дерева, а совокупности деревьев — *случайным лесом*, о чем речь пойдет ниже.

Одно очень важное замечание: дерево решений *решило* выделить только два признака исходного набора данных: Width (Ширина) и Length3 (Длина3). Деревья решений построены таким образом, что более значимые признаки (которые предоставляют наибольшее количество информации, участвующей в прогнозировании) расположены ближе к корню. Таким образом, этот метод позволяет отобрать самые важные признаки, вносящие вклад в предсказания нашей конечной модели.

Неудивительно, что признаки Width и Length3 оказались самыми важными для предсказания веса рыбы. Корреляционная матрица на рис. 3.18 и диаграммы рассеяния на рис. 3.3 показывают довольно сильную корреляцию между всеми признаками длины. Это значит, что они предоставляют избыточную информацию, и включение их всех в модели прогнозирования приведет к увеличению вычислительных затрат и снижению производительности.

	Weight	Length1	Length2	Length3	Height	Width
Weight	1.000000	0.908678	0.911888	0.917883	0.747700	0.896036
Length1	0.908678	1.000000	0.999493	0.991731	0.637844	0.870414
Length2	0.911888	0.999493	1.000000	0.993869	0.653291	0.877268
Length3	0.917883	0.991731	0.993869	1.000000	0.716450	0.882716
Height	0.747700	0.637844	0.653291	0.716450	1.000000	0.802115
Width	0.896036	0.870414	0.677268	0.882716	0.802115	1.000000

Рис. 3.18. Корреляционная матрица для набора данных Fish Market. Между всеми признаками длины прослеживается довольно сильная корреляция

Отбор признаков

Мы затронули довольно важную тему отбора признаков. В реальных наборах данных содержится множество признаков, среди которых какие-то могут давать избыточную информацию, а какие-то вообще не нужны для предсказания целевой метки. Включение нерелевантных и избыточных признаков в модель машинного обучения увеличивает вычислительные затраты и снижает ее производительность. Как мы только что выяснили, одним из способов отбора важных признаков является дерево решений. Еще один способ — метод регуляризации, который называется регрессией лассо и описывается в *главе 4*. Для проверки зависимости признаков друг от друга используются статистические тесты. Так, *F-тест* проверяет линейную зависимость (что дает более высокие оценки коррелированных признаков, хотя

сама по себе корреляция обманчива), а *взаимная информация* — нелинейную зависимость. Эти тесты позволяют оценить вклад признака в определение целевой метки и, следовательно, помогают в отборе признаков, сохраняя наиболее перспективные из них. Кроме того, можно проверить зависимость признаков друг от друга, их корреляции и диаграммы рассеяния. Пороговой фильтрацией *по дисперсии* удаляются признаки с малой или нулевой дисперсией, исходя из предпосылки, что, если сам признак изменяется слабо, значит, он обладает низкой предсказательной силой.

Как обучить дерево решений на наборе данных? Какую функцию оптимизировать? Как правило, при *выращивании* деревьев решений оптимизируются две функции: энтропия и примесь Джини. Причем по итогу нет особой разницы, какую из них применять. Рассмотрим это более подробно.

Энтропия и примесь Джини

Итак, мы решили разделить узел дерева по признаку, который оценивается как *самый важный*. Энтропия и примесь Джини — это два популярных способа определения важности признака. С точки зрения математики они не эквивалентны, но оба работают и дают вполне приемлемые деревья решений. Примесь Джини обычно менее затратна для вычислений, поэтому в программных пакетах она стоит по умолчанию, но при этом всегда есть возможность изменить эту настройку и выбрать энтропию. В случаях когда имеются классы с частотой встречаемости значительно выше, чем у других, пользуясь примесью Джини, мы получим менее сбалансированные деревья. Такие классы изолируются в конечном счете в собственных ветвях. Тем не менее во многих случаях использование энтропии или примеси Джини не дает существенной разницы в итоговых деревьях решений.

Энтропийный подход подразумевает поиск такого разбиения признаков, которое дает *максимальный информационный выигрыш* (в ближайшее время мы приведем его формулу). Информационный выигрыш заимствован из теории информации и связан с понятием энтропии. Энтропия, в свою очередь, относится к термодинамике и статистической физике и количественно характеризует степень беспорядка в некоторой системе.

С помощью *примеси Джини* мы ищем такое разбиение признаков, при которых дочерние узлы имеют наименьшую среднюю примесь Джини (ее формулу мы также приведем в ближайшее время).

Для максимизации информационного выигрыша (или минимизации примеси Джини) нужно, чтобы алгоритм, выращивающий дерево решений, прошелся по каждому признаку обучающего подмножества данных и вычислил информационный выигрыш (или примесь Джини), определил, использует ли дерево данный признак в качестве узла для разбиения, а затем выбрал признак, обеспечивающий наибольший информационный выигрыш (или дочерние узлы с наименьшей средней примесью Джини). Причем, если признак имеет реальные числовые значения, то алгоритм должен решить, *какой вопрос задать в узле*, т. е. по какому значению признака производить разбиение, например, является ли $x_5 < 0{,}1$? Алгоритму при-

ходится делать это последовательно на каждом ярусе дерева, вычисляя информационный выигрыш (или примесь Джини) по признакам экземпляров данных в каждом узле, а иногда и по каждой возможности значения разбиения. Это легче понять на примерах. Но сначала запишем формулы для энтропии, информационного выигрыша и примеси Джини.

Энтропия и информационный выигрыш

Формулу энтропии проще всего понять, доверившись интуиции: если вероятность события высока, значит, оно не вызовет особого удивления. Поэтому чем больше значение p(событие), тем меньше неожиданность. Математически это можно выразить с помощью функции, которая с повышением вероятности будет уменьшаться. В таком случае нам поможет функция $\log\frac{1}{x}$, которая обладает дополнительным свойством, позволяющим складывать независимые события. Поэтому запишем:

$$\text{Неожиданность}(\text{событие}) = \log\frac{1}{p(\text{событие})} = -\log\big(p(\text{событие})\big).$$

Энтропия случайной величины (в нашем случае это конкретный признак в обучающем наборе данных) определяется как *ожидаемая неожиданность*, связанная с этой случайной величиной; следовательно, необходимо сложить неожиданности каждого возможного выхода случайной величины (неожиданности каждого значения рассматриваемого признака), умноженные на их соответствующие вероятности, в результате чего получаем:

$$\begin{aligned}\text{Энтропия}(X) = &-p(\text{выход}_1)\log\big(p(\text{выход}_1)\big) - \\ &- p(\text{выход}_2)\log\big(p(\text{выход}_2)\big) - \ldots - \\ &- p(\text{выход}_n)\log\big(p(\text{выход}_n)\big).\end{aligned}$$

Энтропия одного признака обучающих данных, принимающего множество значений, равна:

$$\begin{aligned}\text{Энтропия}(\text{Признак}) = &-p(\text{значение}_1)\log\big(p(\text{значение}_1)\big) - \\ &- p(\text{значение}_2)\log\big(p(\text{значение}_2)\big) - \ldots - \\ &- p(\text{значение}_n)\log\big(p(\text{значение}_n)\big).\end{aligned}$$

Поскольку наша задача состоит в выборе разбиения по признаку, который обеспечит большой информационный выигрыш относительно выхода (метка или целевой признак), сначала рассчитаем энтропию конечного признака.

Бинарный выход. Для простоты допустим, что мы имеем дело с задачей бинарной классификации, поэтому конечный признак имеет только два значения: положительное (входит в класс) и отрицательное (не входит в класс).

Пусть p — количество положительных экземпляров целевого признака, n — отрицательных, тогда $p + n = m$ — количество экземпляров в обучающем поднаборе

данных. Теперь вероятность выбора положительного экземпляра из этого целевого столбца будет равна $\frac{p}{m} = \frac{p}{p+n}$ и, аналогично, вероятность выбора отрицательного экземпляра будет $\frac{n}{m} = \frac{n}{p+n}$.

Таким образом, энтропия конечного признака (без информации от других признаков) составит[8]:

$$\text{Энтропия}(\text{Конечный признак}) =$$
$$= -p(\text{пол})\log(p(\text{пол})) - p(\text{отр})\log(p(\text{отр})) =$$
$$= -\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right).$$

Далее мы привлекаем информацию от еще одного признака и вычисляем разницу в энтропии итогового признака, которая, как мы ожидаем, будет уменьшаться по мере получения дополнительной информации (как правило, чем больше информации, тем меньше неожиданность).

Предположим, что для разбиения узла дерева решений мы выбрали признак A. Также предположим, что признак A принимает четыре значения и имеет k_1 экземпляра со значением$_1$, из которых p_1 помечены как положительный выход, n_1 — как отрицательный, тогда $p_1 + n_1 = k_1$. Аналогично, признак A имеет k_2 экземпляра со значением$_2$, из которых p_2 помечены как положительный, а n_2 — как отрицательный выход, так что $p_2 + n_2 = k_2$. То же самое относится к значению$_3$ и значению$_4$ признака A. Отметим, что $k_1 + k_2 + k_3 + k_4 = m$ — это общее количество экземпляров в обучающем поднаборе набора данных.

Можно рассматривать каждое значение$_k$ признака A как собственно случайную переменную, имеющую p_k положительных и n_k отрицательных выходов, следовательно, вычисляем его энтропию (ожидаемую неожиданность):

$$\text{Энтропия}(\text{значение}_1) = -\frac{p_1}{p_1+n_1}\log\left(\frac{p_1}{p_1+n_1}\right) - \frac{n_1}{p_1+n_1}\log\left(\frac{n_1}{p_1+n_1}\right);$$
$$\text{Энтропия}(\text{значение}_2) = -\frac{p_2}{p_2+n_2}\log\left(\frac{p_2}{p_2+n_2}\right) - \frac{n_2}{p_2+n_2}\log\left(\frac{n_2}{p_2+n_2}\right);$$
$$\text{Энтропия}(\text{значение}_3) = -\frac{p_3}{p_3+n_3}\log\left(\frac{p_3}{p_3+n_3}\right) - \frac{n_3}{p_3+n_3}\log\left(\frac{n_3}{p_3+n_3}\right);$$
$$\text{Энтропия}(\text{значение}_4) = -\frac{p_4}{p_4+n_4}\log\left(\frac{p_4}{p_4+n_4}\right) - \frac{n_4}{p_4+n_4}\log\left(\frac{n_4}{p_4+n_4}\right).$$

[8] В данной формуле "пол" означает "положительный", "отр" — "отрицательный". — *Прим. ред.*

Располагая такой информацией, можно вычислить *предполагаемую (ожидаемую) энтропию* после разбиения по признаку A, сложив четыре упомянутые энтропии, каждая из которых умножается на соответствующую вероятность:

$$p(\text{значение}_1)=\frac{k_1}{m},\ p(\text{значение}_2)=\frac{k_2}{m},\ p(\text{значение}_3)=\frac{k_3}{m},\ p(\text{значение}_4)=\frac{k_4}{m}.$$

Таким образом, предполагаемая энтропия после разбиения по признаку A составит:

$$\begin{aligned}&\text{Предполагаемая энтропия}(\text{признак A})=\\&=p(\text{значение}_1)\,\text{Энтропия}(\text{значение}_1)+p(\text{значение}_2)\,\text{Энтропия}(\text{значение}_2)+\\&+p(\text{значение}_3)\,\text{Энтропия}(\text{значение}_3)+p(\text{значение}_4)\,\text{Энтропия}(\text{значение}_4)=\\&=\frac{k_1}{m}\,\text{Энтропия}(\text{значение}_1)+\frac{k_2}{m}\,\text{Энтропия}(\text{значение}_2)+\\&+\frac{k_3}{m}\,\text{Энтропия}(\text{значение}_3)+\frac{k_4}{m}\,\text{Энтропия}(\text{значение}_4).\end{aligned}$$

Так каким будет информационный выигрыш от использования признака A для разделения? Он определяется разницей между энтропией конечного признака без информации от признака A и предполагаемой энтропией признака A. То есть если мы решились на разделение по признаку A, можно записать формулу для *информационного выигрыша*:

$$\begin{aligned}&\text{Информационный выигрыш}=\\&=\text{Энтропия}(\text{конечный признак})-\text{Предполагаемая энтропия}(\text{признак A})=\\&=-\frac{p}{p+n}\log\left(\frac{p}{p+n}\right)-\frac{n}{p+n}\log\left(\frac{n}{p+n}\right)-\text{Предполагаемая энтропия}(\text{признак A}).\end{aligned}$$

Теперь можно легко пройтись по каждому признаку обучающего поднабора данных и вычислить информационный выигрыш от использования этого признака для разбиения. В конечном итоге алгоритм дерева решений принимает решение о разделении по признаку с наибольшим информационным выигрышем. Алгоритм делает это рекурсивно для каждого узла и на каждом ярусе дерева, пока не исчерпает все признаки или экземпляры данных. Таким образом, мы получаем дерево решений, построенное на базе энтропии.

Многоклассовый выход. Эту логику несложно обобщить на случай многоклассового выхода, например на задачу классификации с тремя и более целевыми метками. Отличным примером с тремя целевыми метками является классический набор данных Iris (**https://oreil.ly/LZ1V9**) из репозитория UCI Machine Learning Repository (**https://oreil.ly/iOnAc**). В этом наборе содержатся четыре признака ириса: длина и ширина чашелистика, длина и ширина лепестка. Важно отметить, что каждый из этих признаков представляет собой не дискретную, а непрерывную случайную величину. Поэтому, *прежде* чем применять описанную выше логику, необходимо придумать тест для разбиения значений каждого признака. Это делается на этапе инжиниринга признаков в рамках проекта науки о данных. Инжиниринг заключа-

ется в том, чтобы преобразовать непрерывный признак в булев признак, например, является ли длина лепестка > 2,45. Мы не будем подробно рассматривать, как выбрать число 2,45, но несложно догадаться, что здесь также требуется оптимизация.

Примесь Джини

Каждое дерево решений характеризуется узлами, ветвями и листьями. Узел считается *чистым*, если он содержит только те экземпляры данных из обучающего поднабора, которые имеют одинаковую целевую метку (т. е. принадлежат к одному классу). Заметим, что чистый узел — искомый узел, т. к. мы знаем его класс. Поэтому алгоритм стремится вырастить дерево таким образом, чтобы минимизировать нечистоту узлов: узел считается *нечистым*, если в нем не все экземпляры данных принадлежат одному классу. Примесь Джини количественно оценивает эту нечистоту следующим образом.

Предположим, что в нашей задаче классификации имеются три класса, как в случае с набором данных об ирисе. Предположим также, что в некотором узле дерева решений, выращенного для этого набора данных, имеется n обучающих экземпляров, из которых n_1 относится к первому классу, n_2 — ко второму, а n_3 — к третьему (таким образом, $n_1 + n_2 + n_3 = n$). Тогда примесь Джини этого узла можно записать в виде:

$$\text{Примесь Джини} = 1 - \left(\frac{n_1}{n}\right)^2 - \left(\frac{n_2}{n}\right)^2 - \left(\frac{n_3}{n}\right)^2.$$

Таким образом, для каждого узла вычисляем долю экземпляров данных, относящихся к каждому классу, возводим в квадрат, а затем вычитаем сумму этих квадратов из 1. Важно отметить, что если все экземпляры данных узла принадлежат одному классу, то примесь Джини будет равна 0.

Теперь алгоритм выращивания дерева решений ищет в каждом признаке признак и точку разбиения, которые порождают дочерние узлы с минимальной примесью Джини в среднем. Это значит, что дочерние узлы в среднем должны быть чище родительского узла. Таким образом, алгоритм пытается минимизировать средневзвешенное значение примесей Джини двух дочерних узлов (бинарного дерева). Примесь Джини каждого дочернего элемента взвешивается по его относительному размеру, который представляет собой отношение количества его экземпляров к общему количеству экземпляров на данном ярусе дерева (равное количеству экземпляров его родителя). Таким образом, в итоге нам приходится искать комбинацию признака и точки разбиения (для каждого признака), способную решить такую задачу минимизации:

$$\min_{\substack{\text{признак,}\\ \text{значение признака разбиения}}} \frac{n_л}{n}\text{Джини}(\text{левый узел}) + \frac{n_п}{n}\text{Джини}(\text{правый узел}),$$

где $n_л$ и $n_п$ — количество экземпляров данных, оказавшихся в левом и правом дочерних узлах; n — количество экземпляров данных в родительском узле (важно, чтобы $n_л$ и $n_п$ складывались в n).

Регрессионные деревья решений

Важно отметить, что деревья решений можно применять как для регрессии, так и для классификации. Хотя регрессионное дерево решений возвращает не класс, а предсказанное значение, оно строится так же, как и дерево классификации.

Мы не будем разбивать узел, отбирая признак и его значение (например, является ли рост > 3 футов?), которые максимизируют информационный выигрыш или минимизируют примесь Джини. Вместо этого отберем признак и его значение, которые минимизируют среднеквадратическое расстояние между истинными метками и средним значением меток всех экземпляров в каждом из левых и правых дочерних узлов. То есть алгоритм выбирает признак и его значение для разбиения, затем рассматривает полученные в результате разбиения левый и правый дочерние узлы и вычисляет:

- среднее значение всех меток экземпляров обучающих данных в левом узле $y_{\text{л}}$. В случае листового узла это будет значением, предсказанным деревом решений;
- среднее значение всех меток экземпляров обучающих данных в правом узле $y_{\text{п}}$. Аналогичным образом в случае листового узла это будет значением, предсказанным деревом решений;
- сумму квадратов расстояний между значением левого узла и истинной меткой каждого экземпляра в левом узле $\sum_{\text{экземпляры левого узла}} \left| y^i_{\text{истина}} - y_{\text{л}} \right|^2$;
- сумму квадратов расстояний между значением правого узла и истинной меткой каждого экземпляра в правом узле $\sum_{\text{экземпляры правого узла}} \left| y^i_{\text{истина}} - y_{\text{п}} \right|^2$;
- средневзвешенное значение приведенных выше сумм, где каждый узел взвешивается по его размеру относительно родительского узла, как мы это делали для примеси Джини:

$$\frac{n_{\text{л}}}{n} \sum_{\substack{\text{экземпляры} \\ \text{левого узла}}} \left| y^i_{\text{истина}} - y_{\text{л}} \right|^2 + \frac{n_{\text{п}}}{n} \sum_{\substack{\text{экземпляры} \\ \text{правого узла}}} \left| y^i_{\text{истина}} - y_{\text{п}} \right|^2.$$

Такой алгоритм называется *жадным* и считается тяжелым в вычислениях, в том смысле, что *для каждого признака и каждого возможного значения разбиения признака* необходимо выполнить следующие действия: выбрать признак и разбиение признака, обеспечивающие наименьшую средневзвешенную квадратичную ошибку между левыми и правыми дочерними узлами.

В блокнотах Jupyter Notebook дополнительно к книге мы пользуемся известным алгоритмом CART (classification and regression tree — дерево классификации и регрессии), который применяется во многих программных пакетах, включая Python-библиотеку scikit-learn. Этот алгоритм создает деревья с узлами, имеющими только два дочерних элемента (бинарные деревья), где тест в каждом узле имеет лишь ответы "Да" или "Нет". Другие алгоритмы, например алгоритм ID3, могут создавать деревья с двумя дочерними элементами и более.

Недостатки деревьев решений

Деревья решений довольно просты в интерпретации и популярны по целому ряду причин: они адаптируются как к большим массивам данных, так и к различным типам данных (дискретные и непрерывные признаки, масштабирование данных не требуется), могут выполнять задачи как регрессии, так и классификации. Вместе с тем они могут быть нестабильными в том смысле, что добавление всего одного экземпляра в набор данных может изменить дерево в его корне и, следовательно, привести к появлению совершенно другого дерева решений.

Кроме того, они чувствительны к изменениям в данных, поскольку границы решений у них обычно горизонтальные и вертикальные (а не наклонные, как у машин опорных векторов). Это связано с тем, что разбиение обычно происходит при определенных значениях признаков, так что границы решений оказываются параллельными осям признаков. Один из способов решения этой проблемы — преобразование набора данных до его соответствия *главным осям* с помощью метода *сингулярного разложения*, о котором мы будем говорить в *главе 6*. Деревьям решений свойственна чрезмерная подгонка данных, поэтому необходимо отсекать их ветви. Обычно это делается с помощью статистических тестов. При построении с помощью жадных алгоритмов, когда поиск ведется по всем признакам и их значениям, деревья дорого обходятся в плане вычислений и получаются менее точными. Некоторые из этих недостатков устраняются случайными лесами, о которых мы поговорим дальше.

Случайные леса

Когда я впервые узнала о деревьях решений, у меня один за другим начали возникать вопросы, казавшиеся мне на тот момент неразрешимыми:

- С чего начинается построение дерева или, по-другому, как принимать решение о том, какой признак данных считать корневым?
- Каким именно должно быть значение признака, при котором мы принимаем решение о разбиении узла?
- Когда остановиться?
- И, в сущности, как нам вырастить дерево?

(Кстати, некоторые из них мы уже рассмотрели в предыдущем подразделе.) Поиск в Интернете также не облегчил ситуацию — мне ничего не удалось найти, кроме ссылок на то, что разобраться в деревьях решений и их построении не составит большого труда, так что по итогу у меня сложилось впечатление, будто я была единственной, кого озадачили подобные вопросы.

Мое недоумение мгновенно исчезло, когда я познакомилась со *случайными лесами*. Удивительная особенность случайных лесов заключается в возможности получить потрясающие результаты регрессии или классификации, *оставив без внимания* все мои вопросы. Рандомизация, или случайный отбор, — это построение множества деревьев решений, когда на каждый из этих вопросов можно ответить двумя слова-

ми: *случайным образом*. Довольно результативным будет дальнейшее объединение предсказаний в ансамбль, даже по сравнению с глубоко продуманным деревом решений. Ведь не зря говорят: *случайность часто порождает надежность!*

Еще одно чрезвычайно полезное свойство случайных лесов заключается в том, что они дают оценку *важности признаков* и тем самым помогают выявить признаки, оказывающие существенное влияние на предсказание, а также помогают в отборе признаков.

Кластеризация *k* средних

Общая цель аналитиков данных — разделение данных на *кластеры*, каждый из которых будет особо выделять определенные общие черты. *Кластеризация k средних* является одним из распространенных методов машинного обучения, который разбивает *n* точек данных (векторов) на *k* кластеров, причем каждая точка данных назначается в кластер с ближайшим средним значением. Прототипом кластера служит его среднее значение, или центроид. В целом кластеризация *k* средних минимизирует дисперсию (квадратичное евклидово расстояние до среднего) внутри каждого кластера.

Наибольшее распространение получил итерационный алгоритм кластеризации *k* средних.

1. Начинаем с исходного набора *k* средних. Это означает, что мы заранее задаем количество кластеров, и в связи с этим возникает вопрос: с чего начать? Как выбрать местоположение первых *k* центроидов? Этому посвящена соответствующая литература.
2. Назначаем каждую точку данных кластеру с ближайшим среднеквадратическим евклидовым расстоянием.
3. Повторно вычисляем среднее значение по каждому кластеру.

Считается, что алгоритм сходится, когда назначения точек данных каждому кластеру больше не меняются.

Показатели эффективности моделей классификации

Разработка математических моделей, которые что-то вычисляют и выдают результаты, — дело относительно простое. Но совсем другое дело — что-то предпринять, чтобы эти модели эффективно решали поставленные перед ними задачи. Более того, модели, демонстрирующие высокую эффективность по одним метрикам, оказываются неэффективными по другим. Необходимо предельно внимательно относиться к разработке метрик эффективности и принятию решения о том, на какие из них мы будем опираться в каждом конкретном случае.

Эффективность моделей (например, регрессионных), предсказывающих числовые значения, измеряется проще, чем эффективность моделей классификации, т. к. существует множество способов вычисления расстояний между числами (хорошими

и плохими предсказаниями). С другой стороны, когда ставится задача классификации (можно воспользоваться такими моделями, как логистическая регрессия, регрессия softmax, машины опорных векторов, деревья решений, случайные леса или нейронные сети), необходимо дополнительно подумать над оценкой эффективности.

Более того, довольно часто приходится идти на компромиссы. Например, поставлена задача классифицировать видеоролики YouTube как безопасные (положительные) или небезопасные (негативные) для детей. Насколько позволительно в таком случае корректировать модель, так чтобы она дала меньше ложноположительных или ложноотрицательных значений? Безусловно, классификация видеоматериала как безопасного, тогда как в реальности он является небезопасным (ложноположительным), выглядит более проблематичной, чем в обратном случае, что необходимо отразить в метрике эффективности.

Правильность (accuracy).

Доля правильных предсказаний модели[9]:

$$\text{правильность} = \frac{\text{истинноположительные} + \text{истинноотрицательные}}{\text{все ПП} + \text{все ПО}}.$$

Матрица ошибок (confusion matrix).

Подсчет всех истинноположительных, ложноположительных, истинноотрицательных и ложноотрицательных результатов:

Истинноотрицательные значения (true negative, TN)	Ложноположительные значения (false positive, FP)
Ложноотрицательные значения (false negative, FN)	Истинноположительные значения (true positive, TP)

Точность (precision).

Точность положительных предсказаний:

$$\text{точность} = \frac{\text{TP}}{\text{все предсказанные положительные значения}} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Полнота (recall).

Доля правильно классифицированных положительных экземпляров:

$$\text{полнота} = \frac{\text{TP}}{\text{все положительные метки}} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Специфичность (specificity).

Доля правильно классифицированных отрицательных экземпляров:

$$\text{специфичность} = \frac{\text{TN}}{\text{все отрицательные метки}} = \frac{\text{TN}}{\text{TN} + \text{FP}}.$$

[9] В данной формуле ПП — предсказанные положительные; ПО — предсказанные отрицательные. — *Прим. ред.*

F1-мера (F1 score).

Достигнет высокого значения только при высоких показателях точности и полноты:

$$F1 = \frac{2}{\frac{1}{\text{точность}} + \frac{1}{\text{полнота}}}.$$

Площадь под ROC-кривой (area under curve, AUC; receiver operating characteristics).

Оценка эффективности модели классификации при различных пороговых значениях. С ее помощью можно оценить, насколько хорошо конкретная переменная предсказывает определенный исход, например, насколько верным будет предсказание сдачи квалификационного экзамена в аспирантуру на первом курсе по баллу предметного теста GRE.

Отличным руководством по лучшим практикам оценки эффективности является книга Эндрю Нга "Тоска по машинному обучению" (самиздат; **https://oreil.ly/pnZF8**). В ней приведены рекомендации на основе многочисленных испытаний, успехов и неудач, так что стоит внимательно прочитать ее, прежде чем приступить к реальным приложениям искусственного интеллекта.

Итоги и перспективы

В этой главе мы рассмотрели некоторые из наиболее популярных моделей машинного обучения и обозначили особую математическую структуру, повторяющуюся на протяжении всей книги: обучающая функция, функция потерь, оптимизация. Мы обсудили линейную и логистическую регрессии, регрессию softmax, а также рассмотрели машины опорных векторов, деревья решений, ансамбли, случайные леса, кластеризацию *k* средних.

Кроме того, мы привели достойные аргументы в пользу изучения перечисленных тематик с точки зрения математики.

Исчисление.

- Минимум и максимум возникают на границе или в точках, где одна из производных равна нулю или не существует.

Линейная алгебра.

- Линейная комбинация признаков: $\omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n$.
- Запись различных математических выражений с использованием матричной и векторной нотаций.
- Скалярное произведение двух векторов $\vec{a}^T \vec{b}$.
- Норма вектора l^2.
- Не работайте с плохо обусловленными матрицами и линейно зависимыми признаками.

- Не умножайте матрицы друг на друга — это слишком дорого. Умножайте матрицы на векторы.

Оптимизация.

- В случаях с выпуклыми функциями можно не беспокоиться о застревании в локальных минимумах, т. к. локальные минимумы также являются глобальными минимумами. Здесь стоит побеспокоиться об узких долинах (*см. главу 4*).
- Для методов градиентного спуска нужна только одна производная (*см. главу 4*).
- Для методов Ньютона требуются две производные или аппроксимация двух производных (неудобно в случаях с большими данными).
- Квадратичное программирование, двойственная задача, координатный спуск (все эти методы применяются в машинах опорных векторов).

Статистика.

- Корреляционная матрица и диаграммы рассеяния.
- *F*-тест и тест взаимной информации для отбора признаков.
- Стандартизация признаков данных (вычитание среднего значения и деление на стандартное отклонение).

Другие аспекты мы не рассматривали и не будем рассматривать (пока):

- валидация моделей: настройка значений весов и гиперпараметров во избежание чрезмерной подгонки;
- проверка обученной модели на тестовом поднаборе данных, который она не использовала (или не видела) на этапах обучения и валидации (это показано в блокноте Jupyter Notebook на странице книге на сайте GitHub (**https://github.com/halanelson/Essential-Math-For-AI**);
- запуск и мониторинг доработанной модели;
- всегда думайте об усовершенствовании своих моделей и лучшем способе их интеграции в глобальный процесс производства.

В *главе 4* мы вступаем в новую прекрасную эру нейросетей.

ГЛАВА 4
Оптимизация нейронных сетей

> Каждый день я занималась оптимизацией... Мое первое озарение случилось со мной, когда я узнала, что наш мозг тоже обучается модели мира.
>
> — *Хала*

Архитектура искусственных нейросетей состоит из полностью связанных слоев. В этой главе мы рассмотрим математику полносвязной нейросети. Мы построим различные обучающие функции и поэкспериментируем с ними. Мы также узнаем, что методы оптимизации и обратного распространения ошибки, которые используются при обучении нейронных сетей, аналогичны тому, как происходит обучение мозга человека. В процессе обучения, когда мозг сталкивается с уже встречавшимися ему понятиями, он усиливает связи между нейронами, и ослабляет их, когда получает новую информацию, которая противоречит ранее усвоенным понятиям. Машины понимают только числа. С математической точки зрения, более сильные связи соответствуют бóльшим числам, а более слабые — меньшим.

В завершение мы рассмотрим некоторые методы регуляризации, объясним их преимущества и недостатки, приведем примеры их применения.

Кора мозга и искусственные нейросети

Нейросети строятся по образцу коры головного мозга, которая заключает в себе миллиарды нейронов, расположенных в виде слоистой структуры. На рис. 4.1 представлено изображение трех вертикальных сечений неокортекса головного мозга, а на рис. 4.2 — диаграмма полносвязной искусственной нейросети.

Каждый фрагмент на рис. 4.1 изображает вертикальный поперечный срез коры, причем поверхность (внешняя сторона, ближайшая к черепу) коры расположена вверху. Слева — зрительная кора взрослого человека, окрашенная по Нисслю. В центре — моторная кора взрослого человека, окрашенная по Нисслю. Справа — кора головного мозга полуторамесячного младенца, окрашенная по методу Гольджи. Окраска по Нисслю показывает клеточные тела нейронов, а по Гольджи — дендриты и аксоны случайного подмножества нейронов. На всех трех срезах коры головного мозга отчетливо видна слоистая структура нейронов.

И хотя за такие функции, как зрение, слух, логическое мышление, язык, речь и т. д., отвечают разные участки коры головного мозга, все же функцию конкретного уча-

стка фактически определяют его *связи*: с какими сенсорными и моторными областями взаимодействуют его входные и выходные слои. То есть если участок коры соединен с другой сенсорной зоной, например со зрительной, а не слуховой, то он будет выполнять зрительные, а не слуховые задачи (вычисления). В самом упрощенном виде кора головного мозга выполняет одну базовую функцию на уровне нейронов. В искусственной нейросети базовой вычислительной единицей является *перцептрон*, функционирующий одинаково по всей сети. Благодаря архитектуре нейросетей (как коры головного мозга, так и искусственных нейросетей) с ее многочисленными связями и слоями эти вычислительные структуры способны решать самые разнообразные задачи.

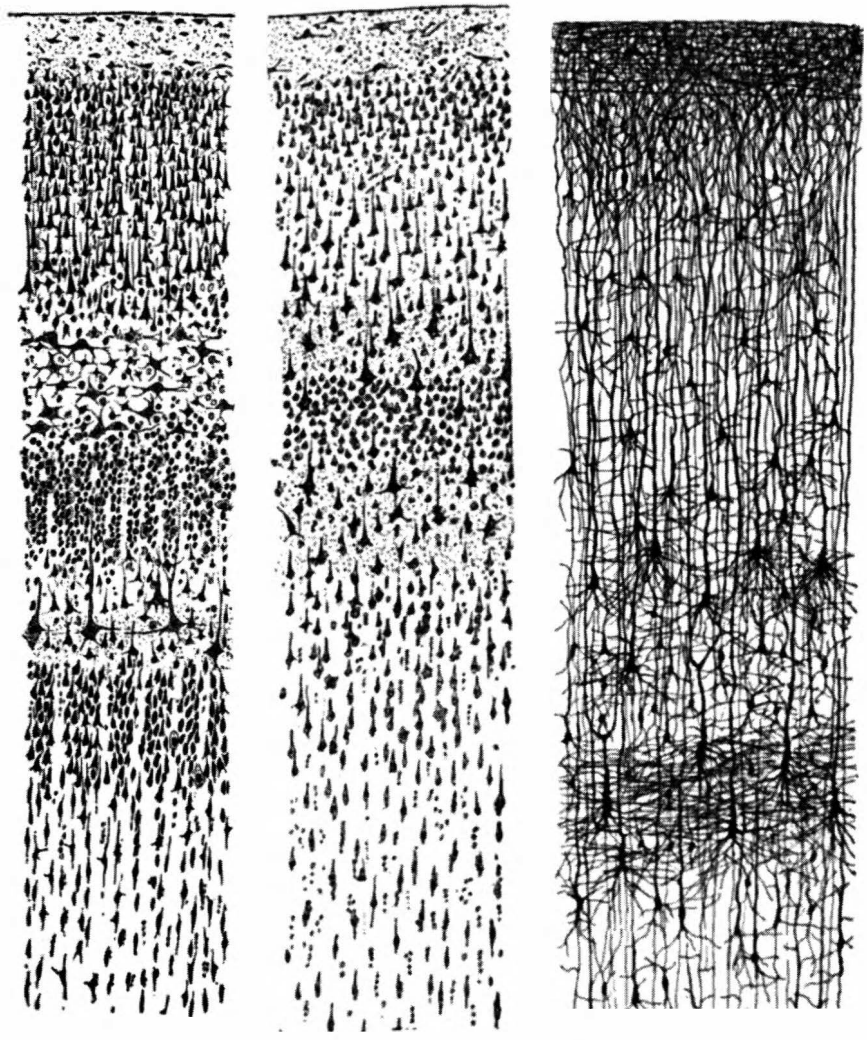


Рис. 4.1. Три рисунка слоистой структуры коры головного мозга, выполненные Сантьяго Рамон-и-Кахалем, из книги "Сравнительное исследование сенсорных зон коры головного мозга человека" (Andesite Press) (источник изображения — Википедия; **https://oreil.ly/r5xJu**)

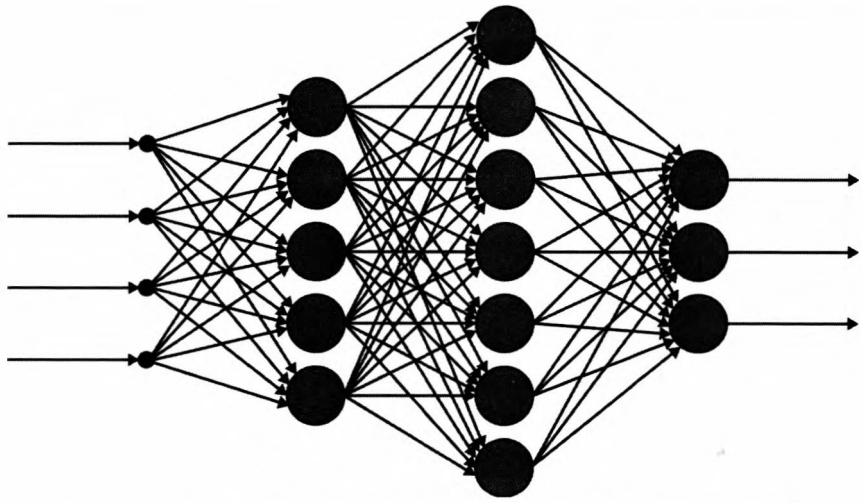


Рис. 4.2. Полносвязная, или плотная, искусственная нейросеть с четырьмя слоями

Обучающая функция: полносвязные (плотные) нейросети с прямой связью

В *полносвязной*, или *плотной*, искусственной нейросети каждый нейрон, представленный в виде узла (кружок на рис. 4.2), связан на каждом слое со всеми нейронами ближайшего слоя. Первый слой называется входным, последний — выходным, а промежуточные — *скрытыми слоями*. Сама нейросеть — полносвязная или нет (в последующих главах мы познакомимся со *сверточными сетями*, которые не являются полносвязными) — представляет собой вычислительный граф, отображающий формулу обучающей функции. Напомним, что после обучения мы используем эту функцию в прогнозировании.

Обучение в контексте нейросетей означает нахождение значений параметров, или весов, входящих в формулу обучающей функции, путем минимизации функции потерь. Подобным образом обучаются модели линейной и логистической регрессий, регрессии softmax и машин опорных векторов, которые мы рассматривали в *главе 3*.

Математическая структура остается прежней:

1. Обучающая функция.
2. Функция потерь.
3. Оптимизация.

Единственное отличие заключается в том, что в *главе 3* формулы обучающих функций для простых моделей достаточно просты. Они линейно комбинируют признаки данных, добавляют член смещения (ω_0) и передают результат не более чем в одну

нелинейную функцию (например, в логистическую функцию в логистической регрессии). Как следствие, модель выдает такие же простые результаты: линейную (плоскую) функцию в линейной регрессии и линейную границу раздела между различными классами в логистической регрессии, регрессии softmax и машинах опорных векторов. Даже применяя эти простые модели в представлении нелинейных данных, например, как в случае полиномиальной регрессии (подгонка данных под полиномиальные функции признаков) или машин опорных векторов с использованием ядерного трюка, мы все равно получаем линейные функции или границы раздела, только они будут иметь либо более высокие размерности (для полиномиальной регрессии размерностями являются признак и его мощности), либо преобразованные размерности (как при использовании ядерного трюка с машинами опорных векторов).

В моделях нейросетей, напротив, линейная комбинация признаков, добавление члена смещения, последующий прогон результата через нелинейную функцию (теперь называемую *функцией активации*) — все эти вычисления выполняются только в *одном нейроне*. Этот простой процесс неоднократно повторяется в десятках, сотнях, тысячах, а иногда и миллионах нейронов, расположенных слоями, где выход одного слоя служит входом для следующего. Как и в коре головного мозга, совокупность простых и схожих процессов во множестве нейронов и слоев порождает или позволяет представить гораздо более сложные функциональные возможности. И это, безусловно, выглядит как чудо. К счастью, искусственные нейросети нам понятны гораздо больше, чем нейронные сети мозга, прежде всего потому, что мы сами их создаем, и к тому же искусственная нейросеть — это всего лишь одна из математических функций. Никакой черный ящик не останется без расшифровки, если мы тщательно исследуем его под линзами математики. Вместе с тем математический анализ искусственных нейронных сетей — это относительно молодая область. Еще предстоит ответить на многие вопросы и многое изучить.

Нейросеть — это вычислительный граф, представляющий обучающую функцию

Даже для сети с пятью нейронами, например, как на рис. 4.3, довольно сложно написать формулу обучающей функции. Поэтому для организованного и простого представления нейронных сетей используются вычислительные графы.

Графы характеризуются двумя элементами: узлами и ребрами (поздравляю, мы прошли первый урок по теории графов). В нейронной сети ребро, соединяющее узел i на слое m с узлом j на слое n, имеет вес $\omega_{mn,ij}$. То есть четыре индекса только на одно ребро! Так мы рискуем захлебнуться в бездонном океане индексов, следовательно, нам требуется организовать веса нейросети в виде матриц.

Смоделируем обучающую функцию полносвязной нейросети с *прямой связью*. Прямая связь означает, что информация проходит через вычислительный граф, представляющий обучающую функцию сети.

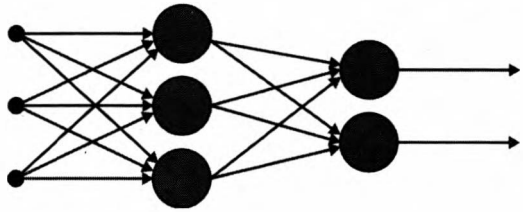


Рис. 4.3. Полносвязная (плотная) нейросеть с прямой связью, состоящая всего из пяти нейронов, расположенных в три слоя. Первый слой (три черные точки слева) — входной, второй слой — единственный скрытый слой с тремя нейронами, последний слой — выходной слой с двумя нейронами

Линейная комбинация, добавление смещения и последующая активация

Какие вычисления происходят в нейроне, когда он получает входную информацию от других нейронов? Для активации нейрона нужно с помощью различных весов найти линейную комбинацию входящей информации, добавить член смещения и применить нелинейную функцию. Рассмотрим этот процесс пошагово.

Веса

Пусть матрица $\mathbf{W}^1$ содержит веса ребер, *приходящихся на* скрытый слой 1, $\mathbf{W}^2$ — веса ребер, приходящихся на скрытый слой 2, и т. д. вплоть до выходного слоя.

Таким образом, в случае небольшой нейронной сети, представленной на рис. 4.3, имеется только $h = 1$ скрытый слой, и мы получаем две матрицы весов:

$$\mathbf{W}^1 = \begin{pmatrix} \omega_{11}^1 & \omega_{12}^1 & \omega_{13}^1 \\ \omega_{21}^1 & \omega_{22}^1 & \omega_{23}^1 \\ \omega_{31}^1 & \omega_{32}^1 & \omega_{33}^1 \end{pmatrix} \text{ и } \mathbf{W}^{h+1} = \mathbf{W}^2 = \mathbf{W}^{\text{выход}} = \begin{pmatrix} \omega_{11}^2 & \omega_{12}^2 & \omega_{13}^2 \\ \omega_{21}^2 & \omega_{22}^2 & \omega_{23}^2 \end{pmatrix},$$

где верхние индексы обозначают слой, на который указывают ребра. Заметим, что если бы на выходном слое было не два, а только один узел, то последняя матрица весов $\mathbf{W}^{h+1} = \mathbf{W}^{\text{выход}}$ была бы только вектор-строкой:

$$\mathbf{W}^{h+1} = \mathbf{W}^2 = \mathbf{W}^{\text{выход}} = \begin{pmatrix} \omega_{11}^2 & \omega_{12}^2 & \omega_{13}^2 \end{pmatrix}.$$

Теперь в одном узле этой нейросети происходят два вычисления:

1. Линейная комбинация со смещением.
2. Прогон результата через нелинейную функцию активации (операция композиции из области исчислений).

Рассмотрим подробно эти две операции, после чего построим обучающую функцию полносвязной нейросети, представленной на рис. 4.3.

Линейная комбинация со смещением

В первом узле первого скрытого слоя (единственного скрытого слоя для этой небольшой сети) линейно комбинируем входные параметры:

$$z_1^1 = \omega_{11}^1 x_1 + \omega_{12}^1 x_2 + \omega_{13}^1 x_3 + \omega_{01}^1.$$

Во втором узле первого скрытого слоя делаем то же самое, только с другими весами:

$$z_2^1 = \omega_{21}^1 x_1 + \omega_{22}^1 x_2 + \omega_{23}^1 x_3 + \omega_{02}^1.$$

В третьем узле первого скрытого слоя делаем то же самое, но уже с весами, отличными от двух предыдущих линейных комбинаций:

$$z_3^1 = \omega_{31}^1 x_1 + \omega_{32}^1 x_2 + \omega_{33}^1 x_3 + \omega_{03}^1.$$

Выразим приведенные выше уравнения с помощью векторной и матричной нотаций. В дальнейшем это нам очень поможет в решении задачи оптимизации и вообще сохранить рассудок:

$$\begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} = \begin{pmatrix} \omega_{11}^1 \\ \omega_{21}^1 \\ \omega_{31}^1 \end{pmatrix} x_1 + \begin{pmatrix} \omega_{12}^1 \\ \omega_{22}^1 \\ \omega_{32}^1 \end{pmatrix} x_2 + \begin{pmatrix} \omega_{13}^1 \\ \omega_{23}^1 \\ \omega_{33}^1 \end{pmatrix} x_3 + \begin{pmatrix} \omega_{01}^1 \\ \omega_{02}^1 \\ \omega_{03}^1 \end{pmatrix} = \begin{pmatrix} \omega_{11}^1 & \omega_{12}^1 & \omega_{13}^1 \\ \omega_{21}^1 & \omega_{22}^1 & \omega_{23}^1 \\ \omega_{31}^1 & \omega_{32}^1 & \omega_{33}^1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} \omega_{01}^1 \\ \omega_{02}^1 \\ \omega_{03}^1 \end{pmatrix}.$$

Можно компактно обобщить приведенное выше выражение в виде:

$$\vec{z}^1 = \mathbf{W}^1 \vec{x} + \vec{\omega}_0^1.$$

Пропускаем результат через нелинейную функцию активации

Линейной комбинации признаков и смещения будет недостаточно для того, чтобы уловить более сложную информацию в данных, и нейронные сети вряд ли добились бы успеха без такого важнейшего, но достаточно простого шага, как композиция с *нелинейной* функцией на каждом узле скрытых слоев.

Линейная комбинация линейной комбинации все равно остается линейной комбинацией

Если мы пропустим этап композиции с нелинейной функцией и будем передавать информацию от первого слоя к следующему, используя только линейные комбинации, то на следующем слое сеть не узнает ничего нового. Она не сможет улавливать более сложные особенности каждого слоя. С точки зрения математики это объясняется просто. Для простоты предположим, что имеются только два входных признака, на первом и втором скрытых слоях — только по два узла. Тогда выходы первого скрытого слоя без нелинейной функции активации примут вид:

$$z_1^1 = \omega_{11}^1 x_1 + \omega_{12}^1 x_2 + \omega_{01}^1;$$
$$z_2^1 = \omega_{21}^1 x_1 + \omega_{22}^1 x_2 + \omega_{02}^1.$$

На втором скрытом слое их снова будем линейно комбинировать, поэтому выход первого узла здесь будет иметь вид:

$$\begin{aligned} z_1^2 &= \omega_{11}^2 z_1^1 + \omega_{12}^2 z_2^1 + \omega_{01}^2 = \\ &= \omega_{11}^2 \left(\omega_{11}^1 x_1 + \omega_{12}^1 x_2 + \omega_{01}^1\right) + \omega_{21}^2 \left(\omega_{21}^1 x_1 + \omega_{22}^1 x_2 + \omega_{02}^1\right) + \omega_{01}^2 = \\ &= \left(\omega_{11}^2 \omega_{11}^1 + \omega_{21}^2 \omega_{21}^1\right) x_1 + \left(\omega_{11}^2 \omega_{12}^1 + \omega_{21}^2 \omega_{22}^1\right) x_2 + \left(\omega_{11}^2 \omega_{01}^1 + \omega_{21}^2 \omega_{02}^1 + \omega_{01}^2\right) = \\ &= \omega_1 x_1 + \omega_2 x_2 + \omega_3. \end{aligned}$$

По итогу получается простая линейная комбинация исходных признаков со смещением. Таким образом, добавление слоя без нелинейной активации не дает ничего нового. Другими словами, функция обучения останется линейной и будет неспособной уловить нелинейные связи в данных.

Мы сами определяем формулу для нелинейной функции активации, причем разные узлы могут иметь различные функции активации, хотя такое редко встречается на практике. Пусть f — функция активации, тогда выход первого скрытого слоя примет вид:

$$\vec{s}^1 = \vec{f}\left(\vec{z}^1\right) = \vec{f}\left(\mathbf{W}^1 \vec{x} + \omega_0^1\right).$$

Очевидно, что при большем количестве скрытых слоев их выходы будут *сцеплены* с выходами предыдущих слоев, из-за чего запись обучающей функции становится немного громоздкой:

$$\vec{s}^2 = \vec{f}\left(\vec{z}^2\right) = \vec{f}\left(\mathbf{W}^2 \vec{s}^1 + \vec{\omega}_0^2\right) = \vec{f}\left(\mathbf{W}^2 \left(\vec{f}\left(\mathbf{W}^1 \vec{x} + \vec{\omega}_0^1\right)\right) + \vec{\omega}_0^2\right);$$

$$\vec{s}^3 = \vec{f}\left(\vec{z}^3\right) = \vec{f}\left(\mathbf{W}^3 \vec{s}^2 + \vec{\omega}_0^3\right) = \vec{f}\left(\mathbf{W}^3 \left(\vec{f}\left(\mathbf{W}^2 \left(\vec{f}\left(\mathbf{W}^1 \vec{x} + \vec{\omega}_0^1\right)\right) + \vec{\omega}_0^2\right)\right) + \vec{\omega}_0^3\right).$$

Эта цепочка продолжается вплоть до выходного слоя. Ситуация на этом последнем слое зависит от поставленной задачи. Если перед сетью стоит задача регрессии (предсказание одного числового значения) или бинарной классификации (разделение на два класса), то в нем будет только один выходной узел (рис. 4.4).

- В задаче регрессии мы линейно объединяем выходы предыдущего слоя в последнем выходном узле, добавляем смещение, и всё (в этом случае мы *не* пропускаем результат через нелинейную функцию). Поскольку у нас только один узел на выходном слое, выходная матрица в таком случае — просто вектор-строка $\mathbf{W}^{\text{выход}} = \mathbf{W}^{h+1}$, и имеем одно смещение ω_0^{h+1}. Тогда предсказание сети примет вид:

$$y_{\text{предсказ}} = \mathbf{W}^{h+1} \vec{s}^h + \omega_0^{h+1},$$

где h — общее количество скрытых слоев в сети (сюда не входят входной и выходной слои).

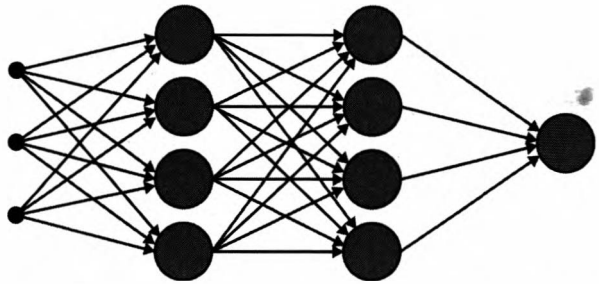


Рис. 4.4. Полносвязная (плотная) нейросеть с прямой связью, состоящая из девяти нейронов, организованных в четыре слоя. Первый слой слева — входной, второй и третий — два скрытых слоя с четырьмя нейронами на каждом, последний — выходной слой с одним нейроном (сеть выполняет либо задачу регрессии, либо задачу бинарной классификации)

- В задаче бинарной классификации мы снова имеем только один выходной узел, в котором линейно комбинируем выходы предыдущего слоя, добавляем смещение и пропускаем результат через логистическую функцию $\sigma(s)=\frac{1}{1+e^{-s}}$. В итоге получаем предсказание сети:

$$y_{\text{предсказ}}=\sigma\left(\mathbf{W}^{h+1}\vec{s}^{\,h}+\omega_0^{h+1}\right).$$

- В задаче классификации на несколько, допустим, пять классов, выходной слой будет содержать пять узлов. В каждом из этих узлов мы линейно комбинируем выходы предыдущего слоя, добавляем смещение и пропускаем результат через функцию softmax:

$$\sigma\left(z^1\right)=\frac{e^{z^1}}{e^{z^1}+e^{z^2}+e^{z^3}+e^{z^4}+e^{z^5}};$$

$$\sigma\left(z^2\right)=\frac{e^{z^2}}{e^{z^1}+e^{z^2}+e^{z^3}+e^{z^4}+e^{z^5}};$$

$$\sigma\left(z^3\right)=\frac{e^{z^3}}{e^{z^1}+e^{z^2}+e^{z^3}+e^{z^4}+e^{z^5}};$$

$$\sigma\left(z^4\right)=\frac{e^{z^4}}{e^{z^1}+e^{z^2}+e^{z^3}+e^{z^4}+e^{z^5}};$$

$$\sigma\left(z^5\right)=\frac{e^{z^5}}{e^{z^1}+e^{z^2}+e^{z^3}+e^{z^4}+e^{z^5}}.$$

Группируем их в векторную функцию $\vec{\sigma}$, которая принимает также на вход векторы $\vec{\sigma}\left(\vec{z}\right)$, тогда итоговое предсказание нейронной сети представляет собой вектор из пяти вероятностных оценок принадлежности экземпляра данных к каждому из пяти классов:

$$\vec{y}_{\text{предсказ}}=\vec{\sigma}\left(\vec{z}\right)=\vec{\sigma}\left(\mathbf{W}^{\text{выход}}\vec{s}^{\,h}+\vec{\omega}_0^{h+1}\right).$$

Принятые обозначения

При обсуждении нейронных сетей мы постараемся придерживаться единой системы обозначений, где: x — входные признаки, $\mathbf{W}$ — матрицы или векторы-столбцы, содержащие веса для линейных комбинаций, ω_0 — смещения, иногда сгруппированные в вектор, z — результаты линейных комбинаций со смещением, s — результаты передачи z в нелинейные функции активации.

Основные функции активации

Теоретически для *активации узлов* можно использовать любую нелинейную функцию (вспомним все функции исчисления, с которыми мы когда-либо сталкивались). На практике существует несколько популярных функций, которые показаны на рис. 4.5 и которые мы скоро перечислим.

В современных сетях чаще всего используется функция активации ReLU (линейный выпрямитель), а достижения AlexNet (**https://oreil.ly/c8RCQ**) в 2012 году частично стали возможными благодаря именно ей, а не гиперболическому тангенсу и логистической функции (сигмоиде), активно используемым на тот момент (и по сей день) в нейросетях.

Первые четыре из перечисленных ниже функций (рис. 4.5) вдохновлены вычислительной нейронаукой, где они пытаются смоделировать порог активации (срабатывания) одной клетки нейрона. Их графики похожи друг на друга: одни являются более гладкими вариантами других, какие-то выводят только положительные числа, третьи — более сбалансированные числа между -1 и 1 или между $-\frac{\pi}{2}$ и $\frac{\pi}{2}$.

Они *перенасыщаются* при малых или больших входах, т. е. при больших входных величинах их графики становятся плоскими. Это создает проблему в *обучении*, ведь если функции будут каждый раз выдавать одни и те же числа, то обучение вряд ли можно будет назвать успешным.

В математике этот феномен проявляется как *проблема исчезающего градиента*. Второй набор функций активации пытается устранить проблему перенасыщения и, как видно на графиках второй строки на рис. 4.5, у него это получается. Однако тут возникает другая проблема, которая называется *проблемой взрывающегося градиента*, обусловленная тем, что эти функции активации являются неограниченными и могут выдавать большие числа, причем сразу в нескольких слоях.

Для каждого нового комплекса проблем существует свой набор методов решения, например, такие как *обрезание градиента*, нормализация выходных данных после каждого слоя и т. д. Главное, что нужно усвоить, — в этом нет никакого чуда. Во многом это метод проб и ошибок: появляются новые методы, которые позволяют устранить проблемы, возникшие в результате применения других новых методов. Важно лишь знать основные принципы, разобраться в причинах и методах, ознакомиться с самыми известными наработками в этой области, не боясь при этом что-то улучшить или даже сделать все совсем по-другому.

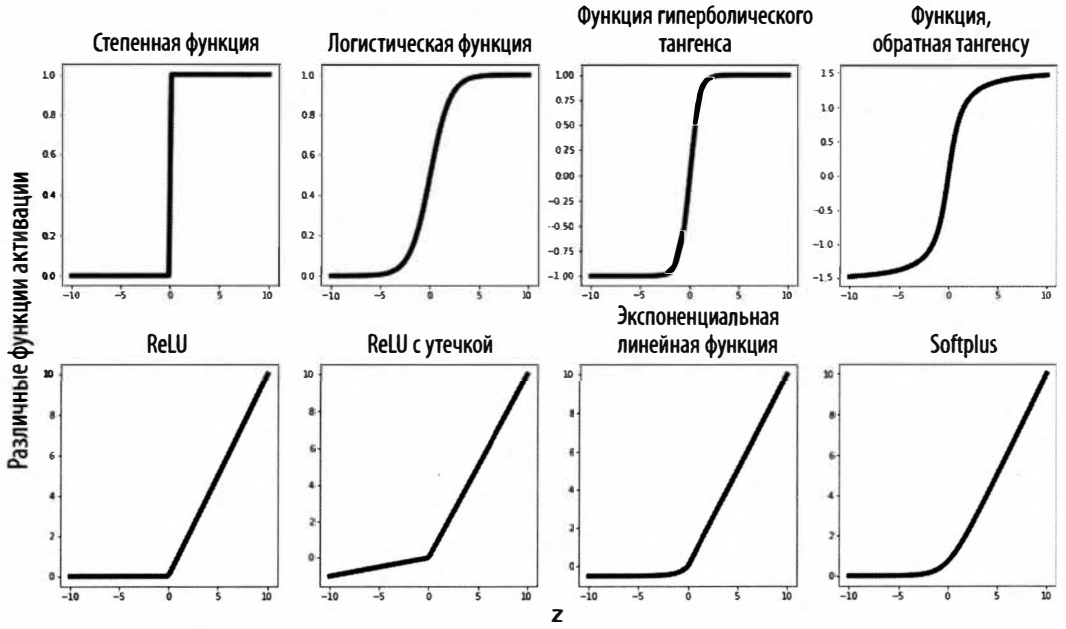


Рис. 4.5. Функции активации для нейросетей. В первом ряду представлены функции активации сигмоидального типа в форме буквы S. Они перенасыщаются (становятся плоскими и выдают одинаковые значения) при больших значениях входных данных. Второй ряд состоит из функций активации ReLU, которые не перенасыщаются. Инженер сможет проследить здесь аналогию функций активации с физической функцией транзистора

Приведем формулы наиболее распространенных функций активации и их производных.

Для оптимизации функции потерь в процессе поиска оптимальных весов нейронной сети требуется вычислить одну производную обучающей функции.

- Степенная функция:

$$f(z)=\begin{cases}0, & \text{если } z<0;\\ 1, & \text{если } z\geq 0.\end{cases}$$

Производная:

$$f'(z)=\begin{cases}0, & \text{если } z\neq 0;\\ \text{не определена}, & \text{если } z=0.\end{cases}$$

- Логистическая функция: $\sigma(z)=\dfrac{1}{1+e^{-z}}$.

Производная:

$$\sigma'(z)=\frac{e^{-z}}{\left(1+e^{-z}\right)^2}=\sigma(z)\left(1-\sigma(z)\right).$$

- Функция гиперболического тангенса:

$$\text{th}(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}=\frac{2}{1+e^{-2z}}-1.$$

Производная:

$$\text{th}'(z)=\frac{4}{\left(e^{z}+e^{-z}\right)^{2}}=1-f(z)^{2}.$$

- Функция, обратная тангенсу: $f(z)=\text{arctg}(z)$.

Производная:

$$f'(z)=\frac{1}{1+z^{2}}.$$

- Функция ReLU(z), или линейный выпрямитель:

$$f(z)=\begin{cases}0, & \text{если } z<0;\\ z, & \text{если } z\geq 0.\end{cases}$$

Производная:

$$f'(z)=\begin{cases}0, & \text{если } z<0;\\ \text{не определена}, & \text{если } z=0;\\ 1, & \text{если } z>0.\end{cases}$$

- Линейный выпрямитель с утечкой (параметрическая функция):

$$f(z)=\begin{cases}\alpha z, & \text{если } z<0;\\ z, & \text{если } z\geq 0.\end{cases}$$

Производная:

$$f'(z)=\begin{cases}\alpha, & \text{если } z<0;\\ \text{не определена}, & \text{если } z=0;\\ 1, & \text{если } z>0.\end{cases}$$

- Экспоненциальная линейная функция (ELU):

$$f(z)=\begin{cases}\alpha\left(e^{z}-1\right), & \text{если } z<0;\\ z, & \text{если } z\geq 0.\end{cases}$$

Производная:

$$f'(z)=\begin{cases}f(z)+\alpha, & \text{если } z<0;\\ 1, & \text{если } z\geq 0.\end{cases}$$

- Функция Softplus (мягкий плюс):

$$f'(z)=\ln\left(1+e^{z}\right).$$

Производная:

$$f'(z) = \frac{1}{1+e^{-z}} = \sigma(z).$$

Важно отметить, что все указанные активации являются достаточно элементарными функциями. И это неплохо, поскольку как они, так и их производные бывают часто задействованы в массивных вычислениях с тысячами параметров (весов) и экземпляров данных при обучении, тестировании и развертывании нейросетей, поэтому лучше оставить их элементарными. Другая причина заключается в том, что с теоретической точки зрения не важно, какую функцию активации мы выберем в итоге, поскольку существуют *универсальные теоремы аппроксимации функций*, о которых пойдет речь далее. И здесь осторожно — на практике, безусловно, имеет значение, какую функцию активации для узлов нашей нейросети мы выберем. Упомянутые достижения AlexNet в задачах классификации изображений отчасти объясняются использованием функции ReLU(z). В данном случае теория и практика не противоречат друг другу, хотя на первый взгляд это выглядит именно так. Об этом мы поговорим в следующем подразделе.

Универсальная аппроксимация функций

Будет просто замечательно, если нам представится случай воспользоваться теоремами аппроксимации — они с математической убедительностью и авторитетом утверждают, что нам вообще не стоит обращать внимание на неизвестную функцию (или известную, но которую сложно включить в вычисления). Ее можно аппроксимировать при помощи известных функций, которые вычисляются гораздо проще и точнее. Это означает, что при определенных условиях как в случае неизвестной или сложной функции, так и в случае известных простых (иногда элементарных) функций можно воспользоваться простыми функциями и при этом быть уверенными в правильности вычислений. Теоремы аппроксимации позволяют количественно определить, насколько истинная функция отличается от своей аппроксимации, так что мы точно знаем, насколько велика будет ошибка при замене истинной функции на подобную аппроксимацию.

Успех нейросетей (даже иногда *неглубоких*, с одним скрытым слоем) в решении различных задач в области зрения, распознавания речи, классификации, регрессии и прочих свидетельствует о том, что они обладают некоторым свойством универсальной аппроксимации. Обучающая функция, представленная нейросетью (построенной из элементарных линейных комбинаций, смещений и довольно простых функций активации), прекрасно аппроксимирует базовую неизвестную функцию, действительно представляющую или генерирующую данные.

И здесь, с математической точки зрения, при помощи одной или нескольких теорем нам нужно выяснить ряд вопросов.

Если нам действительно важна некоторая неизвестная функция (потому что мы считаем ее истинной функцией, либо лежащей в основе наших данных, или их порождающей), существует ли нейросеть, способная аппроксимировать ее с высо-

кой степенью точности (причем нам будет даже необязательно знать эту истинную функцию)?

Опыт успешного использования нейросетей, в частности применение к ним универсальной теоремы аппроксимации, позволяет ответить на этот вопрос положительно.

Если существует такая нейросеть, которая способна аппроксимировать эту неуловимую истинную функцию генерации данных, то как ее построить? Из скольких слоев она должна состоять? Сколько узлов должно быть на каждом слое? Какую функцию активации включить в эту сеть?

Другими словами, какова *архитектура* этой сети? К сожалению, сегодня нам практически неизвестно, как строить такие сети, поэтому единственный путь вперед — экспериментировать с различными архитектурами и активациями, пока еще больше математиков не подключится к этому вопросу.

Существуют ли архитектуры с несколькими нейронными сетями, которые хорошо работают? Есть ли такие из них, которые лучше других?

Эксперименты, в которых мы проводили сопоставление производительности различных архитектур на одних и тех же задачах и наборах данных, позволяют дать утвердительный ответ.

Отметим, насколько для нас важны однозначные ответы на эти вопросы. Утвердительный ответ на первый из них говорит нам: "Привет, здесь нет никакого чуда — нейросети действительно достаточно неплохо аппроксимируют *широкий класс функций*!" Такой *широкий охват* (или универсальность) очень важен, поскольку, напомним, мы не знаем порождающей функции данных, но если теорема аппроксимации охватывает широкий класс функций, то наша неуловимая неизвестная функция вполне может входить в один из них, а значит, нейросеть будет результативной. Ответы на вторую и третью группы вопросов еще больше важны в контексте практических приложений, потому что, когда мы знаем, какая архитектура лучше работает для каждого типа задач и набора данных, в таком случае пропадает необходимость во множестве экспериментов, и мы сразу выбираем нужную архитектуру.

Прежде чем мы перейдем к универсальным теоремам аппроксимации для нейросетей и рассмотрим их доказательства, приведем два примера того, как мы уже сталкивались с теоремами подобного типа еще в средней школе. Во всех примерах действует один и тот же принцип: имеется неуправляемая величина, с которой по каким-либо причинам нам трудно справиться или она неизвестна, и мы хотим аппроксимировать ее с помощью другой величины, которой нам будет легче оперировать. Если нам нужен универсальный результат, необходимо выяснить три момента:

1. К какому классу или какому пространству принадлежит неуправляемая величина или функция? Является ли это множеством действительных чисел $\mathbb{R}$? Множеством иррациональных чисел? Пространством непрерывных функций на интервале? Пространством компактно поддерживаемых функций $\mathbb{R}$? А может, это

пространство функций, измеримых по Лебегу (мы слегка коснулись *теории мер*, надеюсь, это никого не отпугнуло)? И т. д.

2. Какими более простыми величинами или функциями можно воспользоваться в аппроксимации неуправляемых сущностей и какие преимущества мы получаем от использования их вместо истинных функций? Как эти приближения соотносятся с *другими приближениями*, если уже существуют какие-то другие популярные приближения?
3. Как происходит аппроксимация, т. е. когда мы говорим, что с помощью $f_{\text{аппроксимация}}$ можно аппроксимировать $f_{\text{истина}}$, каким образом измеряется расстояние между $f_{\text{истина}}$ и $f_{\text{аппроксимация}}$? Напомним, что в математике существует множество различных способов измерения размеров объектов, в том числе расстояний. Какой из них подойдет непосредственно к нашему конкретному случаю аппроксимации? Здесь мы подходим к таким понятиям, как евклидова норма, равномерная норма, норма супремума, норма L^2 и т. д. Как соотносятся нормы (размеры) и расстояния? Это постигается на уровне интуиции: если пространство позволяет говорить о размерах объектов, то было бы логичным позволить говорить и о расстояниях.

Пример 1: аппроксимация иррациональных чисел рациональными числами

Любое иррациональное число можно аппроксимировать рациональным числом с любой желаемой точностью. Рациональные числа весьма удобны и полезны, поскольку представляют собой пары целых чисел. Мы интуитивно понимаем, что такое целые числа и дроби. С иррациональными числами — совсем противоположная ситуация. Вам когда-нибудь приходилось, скажем в 6 классе, вычислять без калькулятора $\sqrt{47} = 6{,}8556546...$ и не отвлекаться ни на что другое, пока не добьетесь однозначного ответа? Мне приходилось. И это было довольно жестко! Даже калькуляторы и компьютеры аппроксимируют иррациональные числа рациональными. Но я усердно продолжала свою безуспешную попытку, полагая, что так и могу продолжать писать цифры, пока не найду закономерность либо пока вычисления не закончатся. Конечно, ни того, ни другого не случилось, и примерно через 30 цифр я узнала, что, оказывается, бывают иррациональные числа.

Есть больше одного способа записи математического утверждения, количественно оценивающего аппроксимацию. Все они весьма полезны и равноценны.

Аппроксимирующую величину можно сделать сколь угодно близкой к истинной величине.

Это самый интуитивно понятный способ.

Если задано иррациональное число s и любая сколь угодно малая точность ε, то на расстоянии ε от s можно найти рациональное число q:

$$|s - q| < \varepsilon.$$

Это означает, что на вещественной прямой $\mathbb{R}$ рациональные и иррациональные числа существуют сколь угодно близко друг к другу. Таким образом, мы подошли к понятию плотности.

Плотность и замкнутость.

Аппроксимирующие сущности имеют *плотность* в пространстве истинных величин.

Это означает, что если мы сосредоточимся только на пространстве аппроксимирующих членов и добавим все предельные точки всех их последовательностей, то получим полное пространство истинных членов. Добавление всех предельных точек некоторого пространства S называется *замыканием* пространства, или обретением *замкнутости*, $\overline{S}$. Например, если к открытому промежутку $(a; b)$ добавить его предельные точки a и b, то получится замкнутый промежуток — отрезок $[a; b]$. Таким образом, $[a; b]$ — это замыкание (a, b). Это можно записать как $\overline{(a; b)} = [a; b]$.

Множество рациональных чисел $\mathbb{Q}$ имеет *плотность* на вещественной прямой $\mathbb{R}$. Другими словами, *замыкание* $\mathbb{Q}$ есть $\mathbb{R}$. Запишем $\overline{\mathbb{Q}} = \mathbb{R}$.

Пределы последовательностей.

Истинная величина является пределом последовательности аппроксимирующих сущностей.

Добавление предельных точек, о котором говорится в предыдущем пункте, позволяет воспользоваться терминологией последовательностей и их пределов.

Поэтому в контексте аппроксимации иррациональных чисел рациональными можно записать: для любого иррационального числа s существует такая последовательность q_n рациональных чисел, при которой $\lim_{n\to\infty} q_n = s$. Это позволяет записать в качестве примера одно из известнейших определений самого распространенного иррационального числа: $e = 2,71828182...$

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e.$$

То есть *иррациональное* число e является пределом последовательности *рациональных* чисел $\left(1+\frac{1}{1}\right)^1$, $\left(1+\frac{1}{2}\right)^2$, $\left(1+\frac{1}{3}\right)^3$, ..., что эквивалентно 2; 2,25; 2,370370...; ...

Независимо от того, как мы будем аппроксимировать иррациональное число: рациональным числом, воспользовавшись концепцией *сколь угодно близкого* значения, концепцией *плотности* и *замкнутости* или концепцией *пределов последовательностей*, — в математических утверждениях любое расстояние измеряется с помощью обычной евклидовой нормы $d(s, q) = |s - q|$, которая представляет собой нормальное расстояние между двумя числами.

Утверждения о близости должны сопровождаться конкретной нормой

Можно задаться вопросом: что будет, если изменить норму? Сохранится ли свойство аппроксимации? Можем ли мы по-прежнему аппроксимировать иррациональные числа с помощью рациональных, если измеряя расстояние между ними, мы пользуемся не обычной евклидовой нормой, а каким-то другим определением расстояния? Добро пожаловать в *математический анализ*. В общем случае ответ отрицательный. Величины могут быть близки друг к другу по одной норме и довольно далеки по другой. Поэтому когда в математике мы говорим, что величины близки друг к другу, приближаются к другим или где-то сходятся, необходимо указывать соответствующую норму, чтобы точно знать, что подразумевается под этим утверждением о близости.

Пример 2: аппроксимация непрерывных функций полиномами (многочленами)

Непрерывные функции могут представлять собой все, что угодно. Ребенок рисует на листе бумаги волнистую линию — и это будет непрерывная функция с никому не известной формулой. С другой стороны, полиномы — это особый тип непрерывных функций, которые довольно легко оценивать, дифференцировать, интегрировать, объяснять и вычислять.

Единственными операциями над полиномиальными функциями являются умножение, скалярное умножение, сложение, вычитание. Полином степени n имеет простую формулу:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n,$$

где a_i — скалярные числа. Естественно, крайне желательно иметь возможность аппроксимировать неполиномиальные непрерывные функции полиномиальными. Замечательная новость состоит в том, что это можно сделать с любой точностью ε. В математическом анализе этот классический результат называется аппроксимационной теоремой Вейерштрасса.

> Пусть f — непрерывная вещественная функция, заданная на вещественном отрезке $[a; b]$. Для любого значения точности $\varepsilon > 0$ существует такой многочлен p_n с вещественными коэффициентами, что для любого x из $[a; b]$ будет справедливо $|f(x) - p_n(x)| < \varepsilon$, или, что эквивалентно, норма супремума $\|f - p_n\| < \varepsilon$.

Отметим, что здесь действует тот же принцип, что и при использовании рациональных чисел в аппроксимации иррациональных. Теорема утверждает, что всегда можно найти *сколь угодно близкие* к непрерывной функции полиномы, что означает, что в пространстве непрерывных функций на отрезке $[a; b]$ множество полиномов имеет *плотность*, т. е. для любой непрерывной функции f можно найти последовательность полиномиальных функций, сходящихся к f (таким образом, f является пределом последовательности полиномов). Во всех этих вариантах одного

и того же факта расстояния измеряются по норме супремума. На рис. 4.6 мы подтверждаем, что непрерывная функция $\sin x$ является пределом последовательности полиномиальных функций $\left\{x, x-\frac{x^3}{3!}, x-\frac{x^3}{3!}+\frac{x^5}{5!}, \cdots\right\}$.

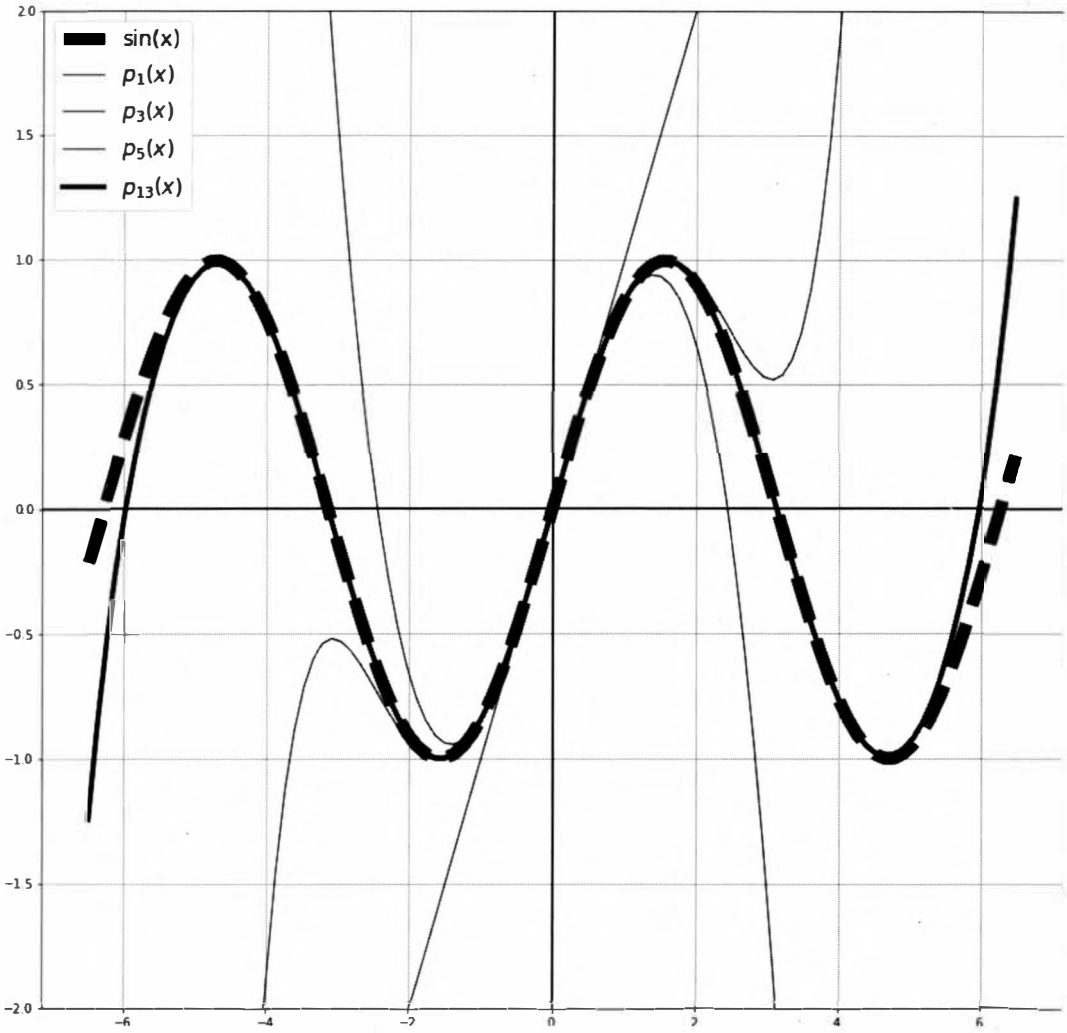


Рис. 4.6. Аппроксимация непрерывной функции $\sin x$ последовательностью полиномов

Утверждения универсальной теоремы аппроксимации в контексте нейросетей

Теперь, когда мы познакомились с принципами аппроксимации, рассмотрим современные утверждения теоремы аппроксимации применительно к нейросетям.

Напомним, что нейронная сеть — это представление обучающей функции в виде вычислительного графа. Нам нужно, чтобы обучающая функция правильно аппроксимировала неизвестную функцию, генерирующую данные. Это позволит использовать в прогнозировании обучающую функцию вместо истинной, которую мы не знаем и, возможно, никогда не узнаем. Приведенные ниже теоремы аппроксимации утверждают, что нейросети способны аппроксимировать базовые функции с любой точностью. Сравним утверждения этих теорем с двумя предыдущими примерами с иррациональными числами и непрерывными функциями — несложно заметить, что они представляют собой одинаковые математические утверждения.

Так, в работе К. Хорника, М. Стинчкомба и Х. Уайта (**https://oreil.ly/e0VWk**), опубликованной в 1989 году, находим следующее: пусть f — непрерывная функция на компактном множестве K (истинная, но неизвестная функция, лежащая в основе данных) с выходами в $\mathbb{R}^d$. Тогда:

сколь угодно близко

существует нейросеть прямой связи с одним скрытым слоем, которая равномерно аппроксимирует f в диапазоне сколь угодно $\varepsilon > 0$ на K;

плотности

множество нейросетей с предписанными нелинейными активациями и ограничениями на число нейронов и слоев в зависимости от d является плотным в равномерной топологии $C\left(K, \mathbb{R}^d\right)$.

В обоих вариантах расстояния измеряются по норме супремума для непрерывных функций.

Для доказательства нам потребуются математические понятия из *теории меры* и *функционального анализа*.

Теорию мер мы рассмотрим в *главе 11*, посвященной вероятности. А пока лишь просто перечислим, что нам потребуется для доказательства: мера Бореля и мера Радона, теорема Хана — Банаха, теорема представлений Риса.

Теория аппроксимации в глубоком обучении

Мы лишь обусловили теорию аппроксимации и изложили один из ее основных результатов для глубокого обучения. Для дополнительной информации и более серьезного знакомства можно изучить такие новейшие достижения, как способность нейросетей к обучению распределениям вероятностей (**https://oreil.ly/gZk03**), теорема Баррона, касательное нейронное ядро и др. (**https://oreil.ly/pNRT0**).

Функции потерь

Хотя в этой главе мы перешли из традиционного машинного обучения *главы 3* в эпоху глубокого обучения, структура, состоящая из функции обучения, функции

потерь и оптимизации, осталась точно такой же. Функции потерь в нейросетях не отличаются от функций, обсуждаемых в *главе 3*, поскольку задача остается прежней: выявить ошибку между "базовой истиной" и предсказанием обучающей функции. В глубоком обучении нейросеть представляет собой обучающую функцию, а в случае нейросетей с прямой связью мы увидели, что это не что иное, как последовательность линейных комбинаций, за которыми следуют умножения с нелинейными функциями активации.

Самыми востребованными в глубоком обучении функциями потерь по-прежнему остаются: в задачах регрессии — функция среднеквадратической ошибки, а в задачах классификации — функция перекрестной энтропии. За подробным объяснением этих функций можно вернуться в *главу 3*.

В работе мы иногда пересекаемся и с другими функциями потерь. Важно помнить, что если нам попадается какая-то новая функция, значит, у разработчиков модели имеются на то свои причины, чтобы предпочесть ее другим, возможно, более известным и популярным, так что нужно всегда понимать, чем обосновано применение конкретной функции в конкретной модели. В идеале хорошая функция потерь штрафует неверные предсказания, не требует больших вычислительных затрат и имеет одну легко вычисляемую производную. Эта производная нужна для корректной работы метода оптимизации. Как уже обсуждалось в *главе 3*, функции с одной верной производной имеют более гладкий ландшафт, чем функции с разрывными производными, и следовательно, по ним легче ориентироваться в процессе оптимизации при поиске минимизаторов функции потерь.

Функция перекрестной энтропии, функция логарифмического правдоподобия и KL-дивергенция

Минимизация функции потерь перекрестной энтропии аналогична максимизации логарифмической функции правдоподобия, с которой тесно связана KL-дивергенция вероятностных распределений. Напомним, что функция перекрестной энтропии, заимствованная из теории информации и статистической механики, призвана количественно оценить перекрестную энтропию между истинным (эмпирическим) распределением данных и распределением (предсказаний), полученным при помощи обучающей функции нейросети. Формула функции перекрестной энтропии имеет отрицательный знак и логарифмическую функцию. Минимизация минуса функции аналогична максимизации той же функции без знака "минус", поэтому иногда на практике встречается такое утверждение: *"максимизация логарифмической функции правдоподобия*, что для нас эквивалентно *минимизации функции потерь перекрестной энтропии"*. Близким понятием считается дивергенция Кульбака — Лейблера (Kullback — Leibler), называемая также *KL-дивергенцией*. Иногда, например, как в случаях генерации изображений или машинного звука, нам необходимо знать не детерминированную функцию, а распределение вероятностей. Тогда функция потерь должна отражать *разницу* (не будем пользоваться словом "расстояние", поскольку ее математическая формула не является метрикой

расстояния) между истинным распределением вероятностей данных и выученным распределением вероятностей. Примером такой функции потерь является KL-дивергенция, которая оценивает количество информации, потерянной при использовании выученного распределения для аппроксимации истинного распределения, или относительную энтропию истинного распределения по отношению к выученному распределению.

Оптимизация

Придерживаясь принятой в книге математической структуры, состоящей из *функции обучения*, *функции потерь* и *оптимизации*, переходим на этап оптимизации. Наша цель — осуществить эффективный поиск ландшафта функции потерь $L(\vec{\omega})$ и выявить минимизирующие ω. Отметим, что когда мы ранее записывали подробные формулы для функций обучения нейросети, мы объединяли веса ω в матрицы **W**, а смещения — в векторы $\vec{\omega}_0$. В этом разделе в целях упрощения обозначений и удержания внимания на математике мы помещаем все веса и смещения в один довольно длинный вектор $\vec{\omega}$. То есть мы записываем функцию потерь в виде $L(\vec{\omega})$, тогда как в действительности для полносвязной нейронной сети с h скрытыми слоями она имеет вид:

$$\text{Функция потерь} = L\left(\mathbf{W}^1, \vec{\omega}_0^1, \mathbf{W}^2, \vec{\omega}_0^2, ..., \mathbf{W}^{h+1}, \vec{\omega}_0^{h+1}\right).$$

Такое представление требуется только при подробном вычислении производной функции потерь *методом обратного распространения ошибки*, о чем мы расскажем далее в этой главе.

Для глубокого обучения количество ω в векторе $\vec{\omega}$ может быть достаточно большим — десятки тысяч, миллионы или даже миллиарды. Модель GPT-2 (**https://oreil.ly/u8S5L**), разработанная компанией OpenAI для естественного языка, имеет 1,5 млрд параметров и была обучена на наборе данных из 8 млн веб-страниц. Необходимо решать задачи с таким количеством неизвестных! Здесь уместно вспомнить о параллельных вычислениях, математической и алгоритмической конвейеризации.

Даже с учетом современных вычислительных мощностей применение методов оптимизации, например, таких как метод Ньютона, требующих вычисления матриц вторых производных функции потерь по такому количеству неизвестных, сегодня просто немыслимо. Это служит отличным примером того, как прекрасно работает математическая теория численного метода, но при этом как она непрактична с точки зрения вычислений и реализации. Самое печальное, что методы численной оптимизации, использующие вторую производную, обычно сходятся быстрее, чем те, что используют только первую, т. к. они пользуются дополнительными знаниями о вогнутости функции (форме ее чаши), а не только информацией о том, является ли функция возрастающей или убывающей, которую дает первая производная. Пока не появятся еще более мощные компьютеры, будем довольствоваться методами

первого порядка, использующими только одну производную функции потерь по неизвестным ω. Это *методы градиентного спуска*, и, к счастью, они отлично работают во многих реальных системах искусственного интеллекта, которые сегодня используются в нашей повседневной жизни, например в виртуальном ассистенте Alexa компании Amazon.

Математика и загадочный успех нейронных сетей

Сделаем небольшую паузу и поразмышляем об успехах нейросетей, что в контексте текущего раздела означает нашу способность находить минимизатор для функции потерь $L(\vec{\omega})$, благодаря которому обучающая функция действительно прекрасно обобщается на неизвестные новые данные. У меня нет североамериканского акцента, и Amazon Alexa прекрасно меня понимает. С математической точки зрения такой успех нейросетей до сих пор вызывает недоумение в силу ряда причин.

- Область определения ω функции потерь $L(\vec{\omega})$, в которой происходит поиск минимума, является высокоразмерной (может достигать миллиардов измерений). Существуют миллиарды и даже триллионы вариантов. Как найти нужный?
- Ландшафт функции потерь сам по себе невыпуклый, поэтому в нем имеется множество локальных минимумов и седловых точек, в которых методы оптимизации могут застревать или сходиться к неправильному локальному минимуму. Опять же, как найти правильный?
- В некоторых приложениях ИИ, например компьютерном зрении, точек ω гораздо больше, чем точек данных (изображений). Вспомним, что в изображениях каждый пиксел является характеристикой, так что уже на входном уровне у нас будет предостаточное количество ω. В таких приложениях гораздо больше неизвестных (ω), чем необходимой для их (точек данных) определения информации. С математической точки зрения, это *недоопределенная система*, а такие системы имеют бесконечное множество возможных решений! Так каким же образом метод оптимизации отбирает верные решения для нашей сети: те, что хорошо обобщаются?

Отчасти этот загадочный успех объясняется методами, ставшими неотъемлемой частью процесса обучения: регуляризация (о которой речь пойдет далее в этой главе), валидация, тестирование и т. д. Однако на сегодняшний день глубокое обучение все еще не имеет достаточно прочного теоретического фундамента. Именно поэтому в последнее время все больше и больше математиков объединяют свои усилия в поисках ответов на эти вопросы.

Для наглядной картины того, что такое математика искусственного интеллекта, приведем пару выдержек из публикаций Национального научного фонда (ННФ — National Science Foundation, NSF):

> "За последнее время ННФ открыл 11 новых исследовательских центров (**https://www.nsf.gov/cise/ai.jsp**), занимающихся развитием ИИ в различных областях, таких как взаимодействие и сотрудничество человека с ИИ, ИИ для оптимиза-

ции, ИИ и передовая киберинфраструктура, ИИ в компьютерных и сетевых системах, ИИ в динамических системах, дополненное обучение с помощью ИИ, а также инновации с помощью ИИ в сельском хозяйстве и продовольственной системе. Благодаря способности объединять многочисленные области научных исследований, в том числе компьютерные и информационные науки и инженерию, а также когнитивные науки и психологию, экономику и теорию игр, инженерное дело и теорию управления, этику, лингвистику, математику и философию, ННФ играет ключевую роль в расширении областей применения ИИ в масштабах всей страны. Бюджет ННФ позволит США в полной мере использовать потенциал ИИ в укреплении экономики, росте числа рабочих мест и принесет пользу обществу на десятилетия вперед".

Далее приведем выдержку из вебкаста ННФ "Математические и научные основы глубокого обучения" (Mathematical and Scientific Foundations of Deep Learning (SCALE MoDL), **https://oreil.ly/ttUVi**).

"Глубокое обучение достигло впечатляющих эмпирических результатов, которые легли в основу фундаментальных научных открытий и преобразований многочисленных прикладных областей искусственного интеллекта. Между тем неполное знание теоретического материала мешает получить доступ к технологиям глубокого обучения более широкому кругу участников. Поэтому изучение механизмов, лежащих в основе достижений глубокого обучения, позволит преодолеть его ограничения и расширить сферу применения. Программа SCALE MoDL финансирует новые исследовательские проекты, объединяющие специалистов из различных областей: математиков, статистиков, инженеров-электриков, информатиков. Исследовательская деятельность должна быть сосредоточена на конкретных тематиках, затрагивающих ряд наиболее сложных теоретических вопросов в рамках общей тематики „Математические и научные основы глубокого обучения“. В каждом проекте должно проводиться обучение с привлечением к исследованиям свежеиспеченных докторов наук, аспирантов и/или студентов старших курсов из этого междисциплинарного спектра. Тематикой проекта может быть любая из научных тем по теоретическим основам глубокого обучения, к числу которых относятся геометрические, топологические, байесовские или теоретико-игровые формулировки; аналитические подходы, использующие теорию оптимального транспорта, теорию оптимизации, теорию аппроксимации, теорию информации, динамические системы, дифференциальные уравнения с частичными значениями или теорию средних полей; прикладные точки зрения, изучающие эффективное обучение с малыми наборами данных, состязательное обучение, замыкание цикла „решение — действие“; а также фундаментальные работы по изучению метрик успеха, защиты конфиденциальности, причинного вывода и алгоритмической справедливости".

Градиентный спуск $\vec{\omega}^{i+1} = \vec{\omega}^{i} - \eta \nabla L\left(\vec{\omega}^{i}\right)$

Довольно распространенный метод градиентного спуска для оптимизации в глубоком обучении настолько прост, что его формулу даже можно вынести в заголовок подраздела. Вот как происходит поиск локального минимума методом градиентного спуска на ландшафте функции потерь $L(\vec{\omega})$.

Начинаем в некоторой точке $\vec{\omega}^0$.

Случайным образом выбираем начальные числовые значения для $\vec{\omega}^0 = (\omega_0, \omega_1, ..., \omega_n)$. Этот выбор помещает нас в некоторое пространство поиска и на ландшафт $L(\vec{\omega})$. Важное предупреждение: ***инициализация имеет значение***! Не следует начинать со всех нулей или со всех одинаковых чисел. Это снизит способность сети к обучению отличительным характеристикам, т. к. разные узлы будут выдавать совершенно одинаковые числа. В ближайшее время мы рассмотрим инициализацию более подробно.

Двигаемся в новую точку $\vec{\omega}^1$.

Градиентный спуск осуществляется в направлении, противоположном вектору градиента функции потерь $-\nabla L(\vec{\omega}^0)$. Это гарантированно снизит градиент, если размер шага η, называемый также *скоростью обучения*, будет не слишком большим:

$$\vec{\omega}^1 = \vec{\omega}^0 - \eta \nabla L(\vec{\omega}^0).$$

Переходим в новую точку $\vec{\omega}^2$.

Снова градиентный спуск осуществляется в направлении, противоположном вектору градиента функции потерь $-\nabla L(\vec{\omega}^1)$. Это гарантированно уменьшит градиент при незначительном η обучения:

$$\vec{\omega}^2 = \vec{\omega}^1 - \eta \nabla L(\vec{\omega}^1).$$

Продолжаем до тех пор, пока не сойдется последовательность точек $\{\vec{\omega}^0, \vec{\omega}^1, \vec{\omega}^2, ...\}$.

Важно отметить, что на практике иногда приходится останавливаться еще до того, как станет ясно, что последовательность сойдется, например, когда она становится слишком длинной из-за сглаженного ландшафта.

На рис. 4.7 показана минимизация функции потерь $L(\omega_1, \omega_2)$ методом градиентного спуска. Мы, люди, существуем в ограниченном трехмерном пространстве, поэтому нам недоступна визуализация за пределами трех измерений. И с точки зрения визуализации, это серьезное ограничение для нас, поскольку функции потерь часто работают в очень высокоразмерных пространствах. Они являются функциями многих ω, но их можно визуализировать правильно только тогда, когда они зависят не более чем от двух ω. То есть можно визуализировать функцию потерь $L(\omega_1, \omega_2)$, зависящую от двух ω, но не $L(\omega_1, \omega_2, \omega_3)$, зависящую от трех (или более) ω. Но даже с учетом такого серьезного ограничения на способность визуализировать функции потерь в высокоразмерных пространствах, на рис. 4.7 можно увидеть точное представление о том, как в целом работает метод градиентного спуска. Здесь поиск ведется в двумерной плоскости (ω_1, ω_2) (плоская площадка на рис. 4.7), и

мы отслеживаем продвижение по ландшафту функции $L(\omega_1, \omega_2)$, включенной в $\mathbb{R}^3$. Пространство поиска всегда имеет на одну размерность меньше, чем размерность пространства, в которое включен ландшафт функции потерь. Это усложняет процесс оптимизации, т. к. приходится искать минимизатор плотного ландшафта в выровненной или сжатой версии его рельефа (уровень земли на рис. 4.7).

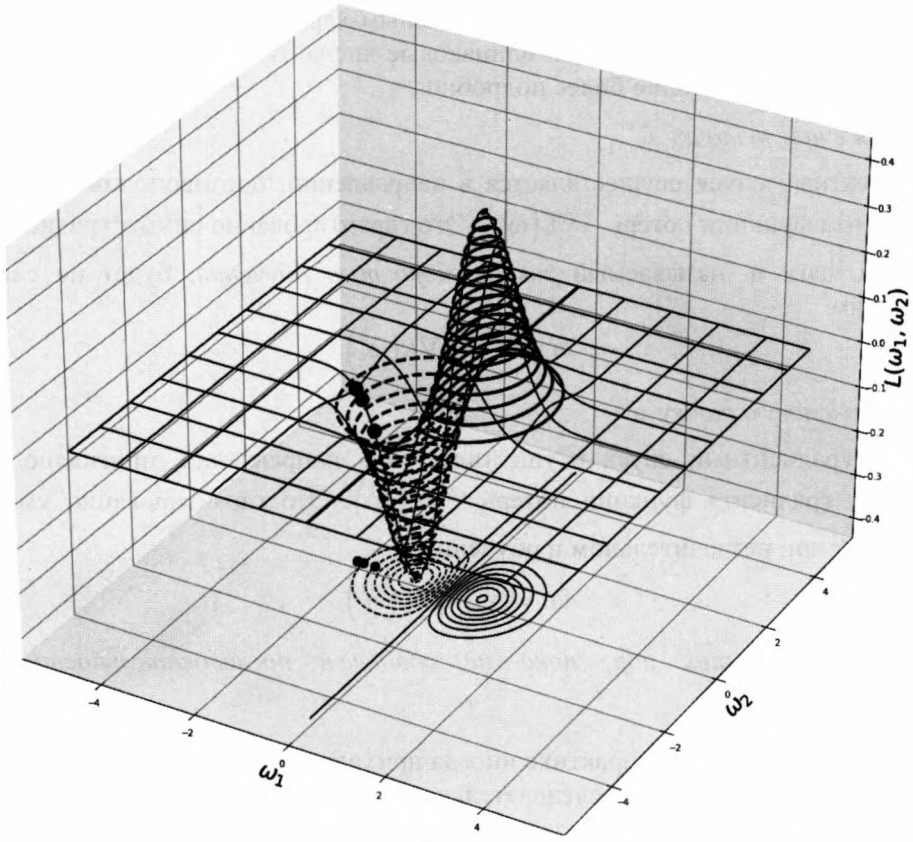


Рис. 4.7. Два шага градиентного спуска. Важно отметить, что мы не придем к минимуму, если начнем с другой стороны горы. Таким образом, при поиске минимума невыпуклой функции очень важно, откуда начинать или как инициировать ω

Роль гиперпараметра скорости обучения η

На каждой итерации метод градиентного спуска $\vec{\omega}^{i+1} = \vec{\omega}^i - \eta \nabla L\left(\vec{\omega}^i\right)$ перемещает нас в пространстве поиска из точки $\vec{\omega}^i$ в точку $\vec{\omega}^{i+1}$. Для того чтобы получить $\vec{\omega}^{i+1}$, градиентный спуск добавляет $-\eta \nabla L\left(\vec{\omega}^i\right)$ к текущей точке $\vec{\omega}^i$. Величина $-\eta \nabla L\left(\vec{\omega}^i\right)$ состоит из скалярного числа η, умноженного на отрицательную часть вектора градиента $-\eta \nabla L\left(\vec{\omega}^i\right)$, направленного из точки $\vec{\omega}^i$ в сторону значительного уменьшения функции потерь.

Таким образом, масштаб $-\eta\nabla L\left(\vec{\omega}^{i}\right)$ показывает, насколько далеко мы должны продвинуться в пространстве поиска в направлении крутого спуска для выбора следующей точки $\vec{\omega}^{i+1}$. Другими словами, вектор $-\eta\nabla L\left(\vec{\omega}^{i}\right)$ задает направление движения от текущей точки, а скалярное число η, или *скорость обучения*, определяет, насколько далеко мы можем двигаться в этом направлении. На рис. 4.8 показан один шаг градиентного спуска с разными значениями скорости обучения η. При слишком высокой скорости обучения возникает риск проскочить минимум и оказаться на другой стороне долины. Но в то же время при слишком низкой скорости обучения достижение минимума займет достаточно много времени. Таким образом, необходимо найти компромисс между высокой скоростью обучения с риском проскочить минимум и низкой скоростью обучения с возрастанием вычислительных затрат и времени сходимости.

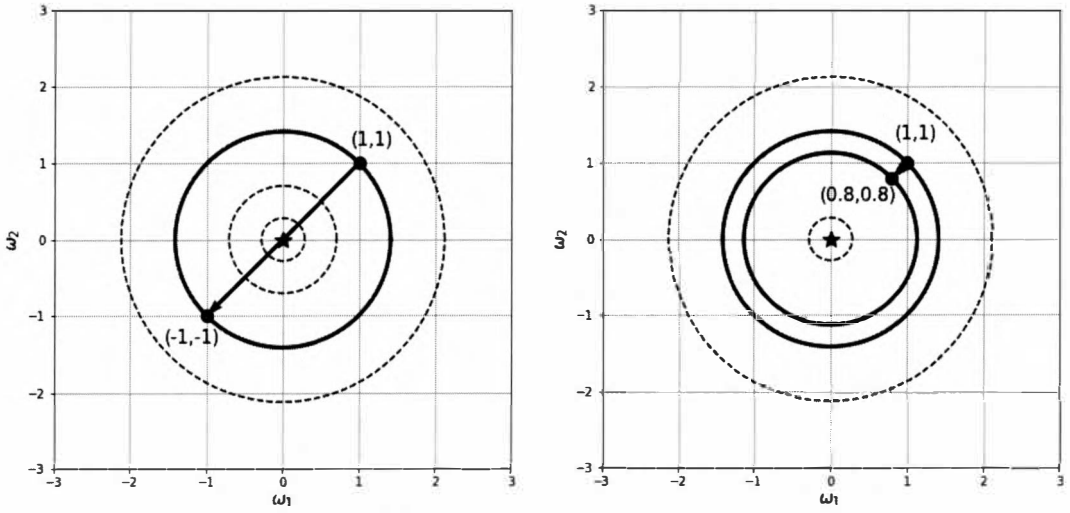


Рис. 4.8. Одношаговый градиентный спуск с двумя различными скоростями обучения. *Слева*: слишком высокая скорость обучения, поэтому градиентный спуск проскакивает минимум (точка со звездой) и попадает на другую сторону долины. *Справа*: низкая скорость обучения, но для достижения минимума (точки со звездой) потребуется некоторое время. Важно отметить, что вектор градиента в точке перпендикулярен заданному в этой точке уровню

Скорость обучения η — еще один пример гиперпараметра модели машинного обучения. Он не относится к весам, входящим в формулу обучающей функции. Этот параметр свойственен алгоритму, которым мы пользуемся для оценки весов обучающей функции.

Эффективность градиентного спуска зависит от масштаба признаков

Это одна из причин, по которой необходимо заранее стандартизировать признаки. Стандартизация признака означает вычитание среднего значения из каждого экземпляра данных и деление на стандартное отклонение. В результате мы получаем

значения данных, сведенные к одному масштабу со средним значением 0 и стандартным отклонением 1, вместо кардинально различающихся масштабов (например, признак, измеряемый в миллионах, и признак, измеряемый в 0,001). Но почему это влияет на эффективность метода градиентного спуска? Идем дальше.

Вспомним, что в обучающей функции значения входных признаков умножаются на веса, а обучающая функция, в свою очередь, входит в формулу функции потерь. Вследствие разительных отличий масштабов входных признаков форма чаши функции потерь изменяется, и процесс минимизации усложняется. На рис. 4.9 показаны наборы уровней функции $L(\omega_1, \omega_2) = \omega_1^2 + a\omega_2^2$ с разными значениями a, имитирующими разные масштабы входных признаков. Важно отметить, что с увеличением значения a множества уровней функции потерь еще больше сужаются и вытягиваются. Это означает, что форма чаши функции потерь представляет собой длинную узкую долину.

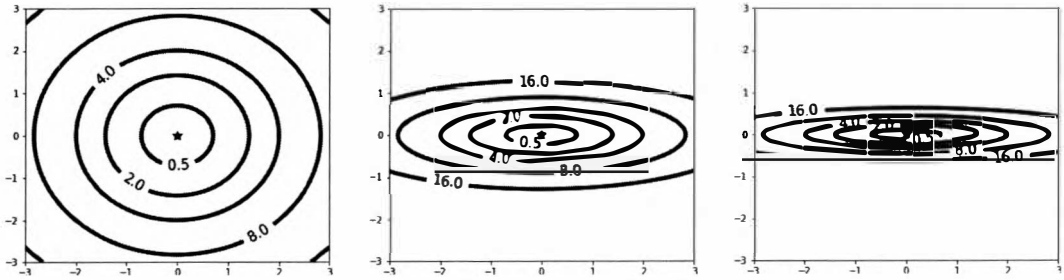


Рис. 4.9. Множества уровней функции потерь $L(\omega_1, \omega_2) = \omega_1^2 + a\omega_2^2$ значительно сужаются и вытягиваются с увеличением значения a от 1 до 20–40

Когда в такой узкой долине применяется метод градиентного спуска, его точки перескакивают с одной стороны долины на другую, совершая зигзаги в попытках найти минимум, и это значительно замедляет сходимость. Представьте, что прежде чем попасть в Ватикан, на автомобиле нам придется выписывать зигзаги по всем улицам Рима, хотя на вертолете мы можем сразу долететь прямиком туда.

Но почему происходит такое зигзагообразное движение? Одним из отличительных признаков вектора градиента функции является то, что он перпендикулярен множествам уровней этой функции. Таким образом, если долина функции потерь настолько длинная и узкая, что ее множества уровней выглядят почти как параллельные друг другу прямые, то при достаточно большом размере шага (скорости обучения) мы можем буквально перейти с одной стороны долины на другую. Можно набрать в поиске Google *"зигзагообразная траектория градиентного спуска"*, и мы увидим массу изображений, иллюстрирующих такое поведение.

Одним из способов устранить зигзагообразное движение даже в случае узкой длинной долины (при условии, что мы не масштабировали значения входных признаков заранее) будет выбор предельно низкой скорости обучения — это предотвратит перескок градиентного спуска с одной стороны долины на другую. Однако это по-своему замедлит достижение минимума, поскольку на каждой итерации будет вы-

полняться только один инкрементный шаг. В конце концов, мы попадем из Рима в Ватикан, но только черепашьими темпами.

Градиентный спуск ползет в окрестности минимумов (локальных и/или глобальных), плоских областей и седловых точек ландшафта функции потерь

Метод градиентного спуска обновляет текущую точку $\vec{\omega}^i$, добавляя вектор $-\eta\nabla L\left(\vec{\omega}^i\right)$. Таким образом, точная длина шага из точки $\vec{\omega}^i$ в направлении вектора отрицательного градиента равна η, умноженному на длину вектора градиента $\nabla L\left(\vec{\omega}^i\right)$. В минимуме, максимуме, седловой точке или любой плоской области ландшафта функции потерь вектор градиента равен нулю, следовательно, его длина также равна нулю. Это означает, что в окрестностях минимума, максимума, седловой точки или любой плоской области размер шага градиентного спуска становится совсем маленьким, и метод существенно замедляется. В окрестности минимума это не вызывает особого беспокойства, т. к. может быть использовано в качестве критерия остановки при условии, что этот минимум не является локальным, весьма далеким от глобального минимума. В плоской области или в окрестности седловой точки метод может на какое-то время застрять, а это нежелательно.

Некоторые специалисты помещают скорость обучения η на график, изменяя ее значение в процессе оптимизации. Если мы изучим эти графики повнимательнее, мы поймем, что делается это с целью избежать сползания, сэкономить вычислительное время и ускорить сходимость.

Далее в этой главе мы рассмотрим стохастический (случайный) градиентный спуск. В силу случайного характера метода точки многократно перемещаются, а не двигаются последовательно к минимуму. Это дает нам преимущество в ситуациях, когда мы застреваем, например, в седловых точках или локальных минимумах, т. к. у нас появляется возможность выбраться из локального минимума или из седловой точки в часть ландшафта с лучшим маршрутом к минимуму.

Выпуклые и невыпуклые ландшафты

В главе, посвященной оптимизации, невозможно обойтись без разговора о *выпуклости*. Фактически целые математические области посвящены исключительно *выпуклой оптимизации* (**https://oreil.ly/xq7bx**). Не менее важно сразу отметить и то, что в большинстве случаев в нейросетях речь идет о *невыпуклой оптимизации*.

В случаях когда мы пользуемся невыпуклыми функциями активации, например сигмоидальными функциями (первый ряд на рис. 4.5), ландшафты функций потерь, участвующих в итоговых нейросетях, будут тоже невыпуклыми. Вот почему мы уделяем так много времени вопросу застревания в локальных минимумах, плоских областях и седловых точках, о которых мы даже не беспокоились бы с выпуклыми ландшафтами. Контраст между выпуклыми и невышуклыми ландшафтами хорошо заметен на рис. 4.10, где показаны выпуклая функция потерь и ее множества уровней, и на рис. 4.11 и 4.12 с изображениями невыпуклых функций и их множествами уровней.

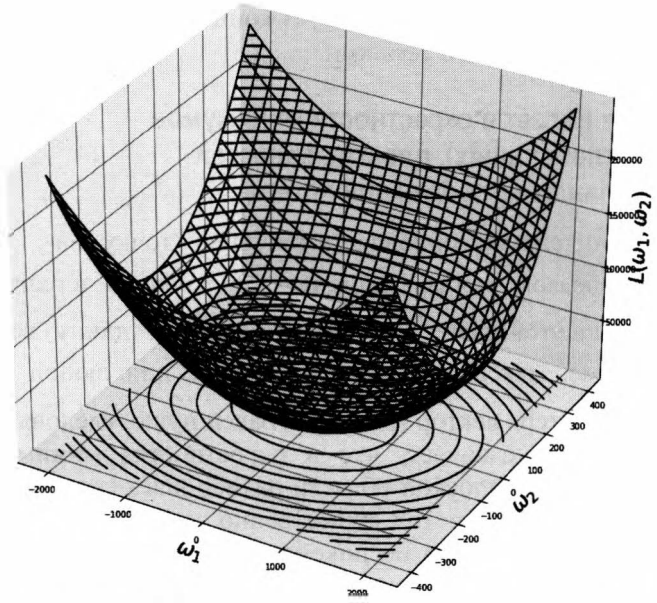


Рис. 4.10. График трехмерной выпуклой функции с ее множествами уровней. Векторы градиента существуют не в $\mathbb{R}^3$, а в том же пространстве ($\mathbb{R}^2$), что и множества уровней

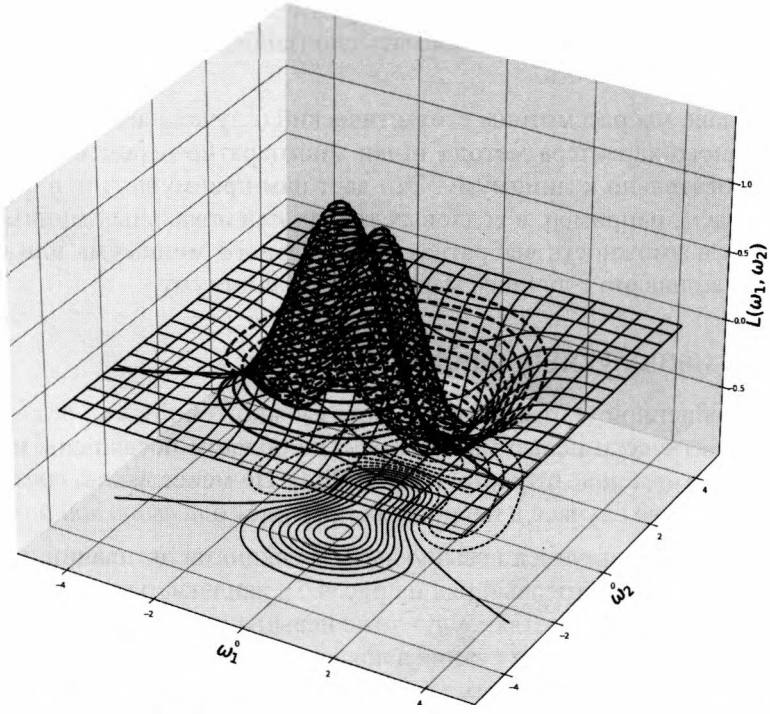


Рис. 4.11. График первой трехмерной невыпуклой функции с ее множествами уровней. Векторы градиента существуют не в $\mathbb{R}^3$, а в том же пространстве ($\mathbb{R}^2$), что и множества уровней

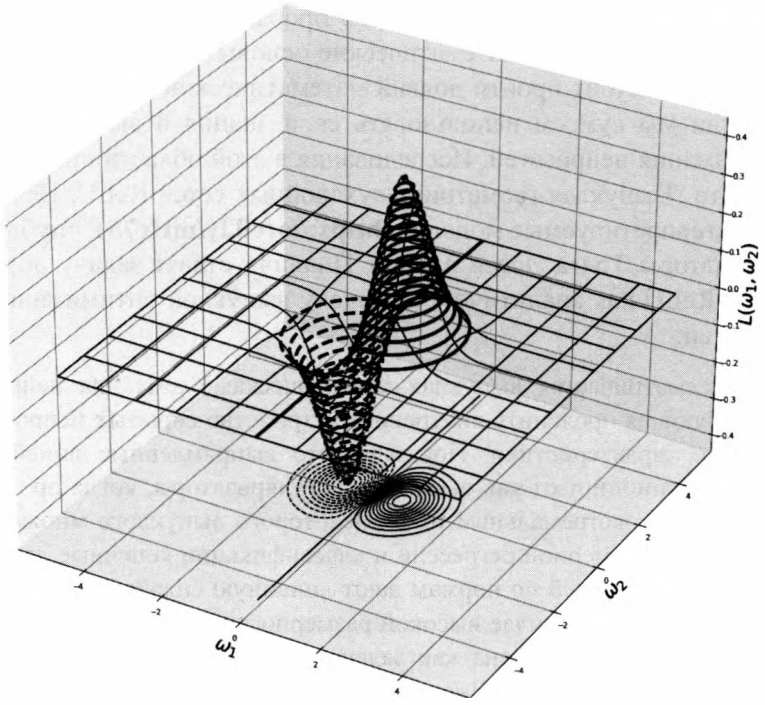


Рис. 4.12. График второй трехмерной невыпуклой функции с ее множествами уровней. Векторы градиента существуют не в $\mathbb{R}^3$, а в том же пространстве ($\mathbb{R}^2$), что и множества уровней

В случаях выпуклых функций активации, например функции ReLU (второй ряд на рис. 4.5), *и* выпуклых функций потерь все равно можно в конечном итоге выйти на задачу *невыпуклой* оптимизации, поскольку в результате композиция двух выпуклых функций необязательно будет выпуклой. В случае неубывающей выпуклой функции потерь результат будет выпуклым.

Самые распространенные функции потерь для нейросетей, например, такие как среднеквадратическая ошибка, перекрестная энтропия и потеря шарнира, являются выпуклыми, но не неубывающими.

Очень важно ознакомиться с основными понятиями из области выпуклой оптимизации. Для начала просто запомним, что выпуклость заменяет линейность, когда линейность слишком упрощена или недоступна, а затем приведем некоторые понятия, связанные с ИИ, глубоким обучением и обучением с подкреплением. Мы рассмотрим их в обсуждении исследования операций в *главе 10*:

- максимум линейной функций — выпуклый;
- минимаксный и максиминный критерии;
- седловые точки;
- игра двух игроков с нулевой суммой;
- двойственность.

Выпуклая оптимизация является достаточно проработанной и понятной областью (по крайней мере, лучше, чем математические основы нейросетей), в то время как нейросетям еще предстоит пройти долгий математический путь, поэтому будет замечательно, если мы сумеем использовать свои знания о выпуклости для более глубокого понимания нейросетей. Исследования в этой области продолжаются. Например, в работе "Выпуклая геометрия двухслойных сетей ReLU. Неявное автокодирование и интерпретируемые модели" (**https://oreil.ly/mHc7S**), опубликованной в 2020 году, ее авторы Толга Эрген и Мерт Пиланчи ставят задачу обучения двухслойных сетей ReLU как аналитическую задачу выпуклой оптимизации. Приведем аннотацию статьи:

> "В статье рассматривается выпуклая аналитическая схема для нейронных сетей ReLU, позволяющая прояснить внутреннее устройство скрытых нейронов и их пространственные характеристики. Показано, что выпрямленные линейные блоки в нейросетях функционируют как выпуклые регуляризаторы, когда простые решения поощряются через экстремальные точки некоторого выпуклого множества. Доказано, что в случае одномерной регрессии и классификации конечные двухслойные сети ReLU с регуляризацией по нормам дают линейную сплайн-интерполяцию. Показано, что в более общем случае высокой размерности задача обучения двухслойных сетей может быть представлена как задача выпуклой оптимизации с бесконечно большим числом ограничений. Затем предложены семейство выпуклых релаксаций для аппроксимации решения и алгоритм секущей плоскости для повышения релаксаций. Выведены условия точности релаксации и приведены простые замкнутые формулы для оптимальных весов нейросети в определенных случаях. Полученные результаты показывают, что скрытые в сети ReLU нейроны можно интерпретировать как выпуклые автокодировщики входного слоя. Также для нейронных сетей установлена связь с эквивалентностью $l_0 - l_1$ по аналогии с решениями минимальной кардинальности при обнаружении со сжатием. Исчерпывающие результаты экспериментов показали, что предложенный подход позволяет получить точные интерпретируемые модели".

Стохастический градиентный спуск

На сегодняшний день обучение нейросети прямой связи выглядит следующим образом:

1. Задать начальный для обучающей функции набор весов $\vec{\omega}^0$.
2. Оценить эту обучающую функцию по *всем* точкам данных в обучающем поднаборе.
3. Вычислить потери по *всем* точкам данных в обучающем поднаборе, сравнив их истинные метки с предсказаниями обучающей функции.
4. Проделать это для *всех* данных обучающего подмножества.
5. Усреднить все потери. Полученное среднее значение и есть функция потерь.

6. Оценить градиент полученной функции потерь при данном начальном наборе весов.
7. Выбрать следующий набор весов согласно правилу крутого спуска.
8. Повторять до тех пор, пока не получится сходимость, или остановиться после определенного количества итераций, определяемого производительностью обучающей функции на валидационном наборе.

Проблема этого процесса заключается в том, что в случае большого обучающего поднабора данных с тысячами точек и нейросети с тысячами весов оценить обучающую функцию, функцию потерь и градиент функции потерь по *всем* точкам данных в обучающем поднаборе становится слишком дорого. Исправить ситуации позволит рандомизация процесса — выбор случайным образом незначительной части обучающего поднабора для оценки обучающей функции, функции потерь и ее градиента на каждом шаге. Это значительно сокращает вычислительные затраты.

Необходимо повторять случайную выборку (в принципе, с заменой, но на практике — без замены) до тех пор, пока нам не удастся достичь сходимости, или остановиться после определенного количества итераций, определяемого производительностью обучающей функции на проверочном наборе. Каждый проход по всему обучающему поднабору называется *эпохой*.

Стохастический градиентный спуск доказал свою высокую эффективность и прочно вошел в практику обучения нейросетей.

Инициализация весов $\vec{\omega}_0$ в оптимизации

Мы уже выяснили, что инициализация с нулевыми весами или с одинаковыми весами — это очень плохая идея. Следующим логическим шагом и традиционной практикой (до 2010 года) был бы случайный выбор весов для исходного $\vec{\omega}_0$ из равномерного распределения на малых отрезках, например [–1; 1], [0; 1] или [–0,3; 0,3], либо из гауссова распределения с заранее выбранными средним и дисперсией. И хотя этот вопрос не изучался подробно, из эмпирических данных следует, что совершенно неважно, откуда берутся начальные веса — из равномерного или гауссова распределения, но, по всей видимости, масштаб начальных весов играет не последнюю роль как в успешном исходе оптимизации, так и в способности сети правильно обобщаться на незнакомые данные. Оказывается, что некоторые варианты в этом отношении лучше других. Сегодня можно говорить о двух самых оптимальных из них, которые зависят от выбора функции активации: сигмоида или ReLU.

Инициализация Хавьера Глорота (Xavier Glorot).

Начальные веса выбираются из равномерного распределения на отрезке $\left[-\frac{\sqrt{6}}{\sqrt{n+m}}; \frac{\sqrt{6}}{\sqrt{n+m}}\right]$, где n — количество входов в узел (например, количество узлов на предыдущем слое), m — количество выходов из слоя (например, количество узлов на текущем слое).

Инициализация Кайминга Хе (Kaiming He).

Начальные веса выбираются из гауссовского распределения с нулевым средним и дисперсией $2/n$, где n — количество входов в узел.

Методы регуляризации

Регуляризация позволяет выбрать оптимальные значения весов обучающей функции и в то же время избежать чрезмерной подгонки данных. Для того чтобы наша обученная функция могла хорошо обобщать данные, нужно, чтобы она улавливала сигнал без помех. Здесь мы рассмотрим четыре простых, но достаточно распространенных метода регуляризации, применяемых в обучении нейросетей: отсев, раннюю остановку, пакетную нормализацию и регуляризацию сокращения веса (гребневая, лассо, эластичная сеть).

Отсев

Отсев (исключение), или дропаут (dropout), — это удаление нескольких случайно выбранных нейронов из каждого слоя в процессе обучения. Как правило, случайным образом исключается около 20% узлов входного слоя и около половины узлов каждого скрытого слоя. Узлы выходного слоя не исключаются. Отсев узлов отчасти вторит принципу генетического воспроизводства, когда исключается половина родительских генов и происходит небольшая случайная мутация. Благодаря этому можно одновременно обучать разные сети (с разным количеством узлов на каждом слое) и усреднять их результаты, что дает в большинстве случаев более достоверные результаты.

Один из способов отсева заключается во введении гиперпараметра p для каждого слоя, задающего вероятность исключения каждого отдельного узла этого слоя. Напомним основные операции, происходящие в каждом узле: линейная комбинация выходов узлов предыдущего слоя с последующей активацией. Применяя метод исключения, каждый выход узлов предыдущего слоя (начиная с входного) умножается на случайное число r, которое с вероятностью p может быть либо 0, либо 1. Таким образом, когда r узла принимает значение 0, этот узел, по сути, исключается из сети, в результате чего при настройке весов на один шаг градиентного спуска приходится компенсировать недостаток за счет оставшихся узлов. Более подробно мы рассмотрим это в *разд. "Обратное распространение ошибки в деталях" далее в этой главе.*

Более глубокое математическое исследование приводится в работе (**https://oreil.ly/uvwNf**), опубликованной в 2015 году, где исключение связывается с байесовской аппроксимацией неопределенности модели.

Ранняя остановка

Поскольку в процессе обучения, в частности в процессе градиентного спуска, веса обновляются, мы должны после каждой эпохи оценивать ошибку, допущенную обучающей функцией при текущих весах на валидационном поднаборе данных.

По мере обучения модели на обучающих данных ошибка должна уменьшаться, однако после определенного количества эпох она начинает увеличиваться, свидетельствуя о том, что обучающая функция начала чрезмерно подгонять обучающие данные и не может правильно обобщить их на валидационные данные. Как только мы обнаружим подобное увеличение в предсказании модели по валидационному поднабору, нужно будет прекратить обучение и возвратиться к набору весов, при котором наблюдалась наименьшая ошибка до того момента, пока она не стала увеличиваться.

Пакетная нормализация каждого слоя

Основная идея заключается в нормализации входов каждого слоя сети. Это означает, что среднее значение входов каждого слоя будет равно 0, а дисперсия — 1. Как правило, это достигается вычитанием среднего значения и делением на дисперсию для каждого из входов слоя. В ближайшее время мы рассмотрим это более подробно. Такие действия целесообразно выполнять на каждом скрытом слое по той же причине, что и на изначальном входном слое.

Применение *пакетной нормализации* часто устраняет необходимость в использовании исключения и снижает требования к инициализации. Это ускоряет процесс обучения и защищает от взрывающихся и затухающих градиентов. Кроме того, она обладает дополнительным преимуществом в виде регуляризации.

Стоимость всех этих преимуществ не слишком высока, т. к. обычно на каждом слое обучаются всего два дополнительных параметра — масштабирование и сдвиг.

Этот метод приведен в работе Иоффе и Сегеди (Szegedy), опубликованной в 2015 году (**https://oreil.ly/pZwV0**). В аннотации описывается процесс пакетной нормализации и решаемые им задачи (в скобках — мои комментарии):

> "Обучение глубоких нейронных сетей осложняется тем, что в процессе обучения распределение входов каждого слоя меняется по мере изменения параметров предыдущих слоев. Это замедляет обучение, требуя более низких значений скорости и тщательной инициализации параметров, и делает заведомо трудным обучение моделей с насыщающимися нелинейностями (например, сигмоидальные функции активации, показанные на рис. 4.5, которые становятся практически постоянными, выдавая одно и то же значение при большой входной величине. В итоге в процессе обучения нелинейность становится бесполезной, и сеть перестает обучаться на последующих слоях). Мы называем это явление (изменение распределения входов каждого слоя) внутренним ковариационным сдвигом и решаем проблему путем нормализации входов слоев. Сильная сторона нашего метода заключается в том, что нормализация является частью архитектуры модели и выполняется для каждого

обучающего мини-пакета. Пакетная нормализация позволяет применять гораздо более высокие скорости обучения и менее тщательно подходить к инициализации, а в некоторых случаях исключает необходимость в процедуре отсева. С помощью пакетной нормализации мы получаем в современной модели классификации изображений ту же точность, выполняя в 14 раз меньше шагов обучения, и значительно превосходим исходную модель по величине. Применяя ансамбль сетей с пакетной нормализацией, мы превзошли лучший по классификации ImageNet результат 4,82% ошибки в пяти топовых тестах, что точнее оценок живых экспертов".

Пакетная нормализация часто реализуется в архитектуре сети либо на ее собственном слое до или после активации. Как правило, *процесс обучения* включает следующие шаги:

1. Из обучающих данных выбираем пакет размером b. В нем каждая точка данных имеет вектор признаков $\vec{x}_i$, следовательно, весь пакет будет иметь векторы признаков $\vec{x}_1, \vec{x}_2, ..., \vec{x}_b$.
2. Вычисляем вектор, записи которого представляют средние значения каждого признака в этом пакете: $\vec{\mu} = \frac{\vec{x}_1 + \vec{x}_2 + ... + \vec{x}_b}{b}$.
3. Вычисляем дисперсию по всему пакету: вычитаем $\vec{\mu}$ из каждого $\vec{x}_1, \vec{x}_2, ..., \vec{x}_b$, вычисляем норму l^2, складываем и делим на b.
4. Нормализуем каждое из $\vec{x}_1, \vec{x}_2, ..., \vec{x}_b$ путем вычитания среднего значения и деления на квадратный корень дисперсии.
5. Масштабируем и сдвигаем по обучаемым параметрам, которые можно инициализировать и обучить методом градиентного спуска аналогично тому, как обучаются веса обучающей функции.
6. Делаем то же самое для входа каждого из последующих слоев.
7. Повторяем со следующим пакетом.

При тестировании и прогнозировании у нас нет пакетных данных для обучения, а параметры каждого слоя уже известны. При этом шаг пакетной нормализации уже заложен в формулу обучающей функции. В процессе обучения мы меняли их для *каждого пакета* обучающих данных, что, в свою очередь, слегка меняло формулу функции потерь для них. Однако смысл нормализации отчасти заключался в том, чтобы не менять формулу функции потерь слишком сильно, т. к. это ведет к изменению местоположения ее минимумов, из-за чего мы будем вечно преследовать движущуюся цель. Но в процессе обучения мы исправили это с помощью пакетной нормализации, и теперь хотим провести валидацию, тестирование и прогнозирование. Какие среднее векторное и дисперсию мы используем для конкретной тестируемой/прогнозируемой точки данных? Воспользуемся ли мы средним и дисперсией признаков исходного набора данных? Нам придется принимать подобные решения.

Регулирование размера весов наложением штрафа на их норму

Другой способ регуляризации обучающей функции, позволяющий избежать чрезмерной подгонки данных, заключается во введении *конкурирующего члена* в задачу минимизации. Вместо решения задачи для набора весов $\vec{\omega}$, минимизирующего *только* функцию потерь:

$$\min_{\vec{\omega}} L(\vec{\omega}),$$

введем новый член $\alpha\|\vec{\omega}\|$ и решим задачу для набора весов $\vec{\omega}$, который минимизирует

$$\min_{\vec{\omega}} L(\vec{\omega}) + \alpha\|\vec{\omega}\|.$$

Например, для функции потерь среднеквадратической ошибки, обычно используемой в задачах регрессии, задача минимизации имеет вид:

$$\min_{\vec{\omega}} \frac{1}{m}\sum_{i=1}^{m}\left|y_{\text{предсказ}}(\vec{\omega}) - y_{\text{истина}}\right|^2 + \alpha\|\vec{\omega}\|.$$

Напомним, что пока мы установили два способа решения этой задачи минимизации.

Минимум достигается в точках, где производная (градиент) равна нулю.

Таким образом, минимизация $\vec{\omega}$ должна удовлетворять $\nabla L(\vec{\omega}) + \alpha\nabla(\|\vec{\omega}\|) = 0$. Затем мы решаем это уравнение для $\vec{\omega}$, если у нас есть возможность получить замкнутую форму для решения. В случае линейной регрессии (которую можно рассматривать как чрезвычайно упрощенную нейросеть с одним слоем и нулевой нелинейной функцией активации) такая возможность есть, и для такого *регуляризованного* случая формула для минимизации $\vec{\omega}$ имеет вид:

$$\vec{\omega} = \left(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{B}\right)^{-1}\mathbf{X}^T\vec{y}_{\text{истина}},$$

где столбцы $\mathbf{X}$ — столбцы признаков данных, дополненные вектором единиц, $\mathbf{B}$ — матрица тождеств (в случае гребневой регрессии, о которой речь пойдет ниже). Решение в замкнутой форме для очень простой задачи линейной регрессии с регуляризацией позволяет оценить регуляризацию сокращения веса и убедиться в ее важности. Мы не будем инвертировать матрицу $\mathbf{X}^T\mathbf{X}$ в *нерегуляризованном* решении и беспокоиться о ее плохой обусловленности (например, из-за сильно коррелированных входных признаков) и возникающих нестабильностях. Вместо этого мы инвертируем $\left(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{B}\right)$ в *регуляризованном* решении. Добавление члена $\alpha\mathbf{B}$ эквивалентно добавлению небольшого положительного члена в знаменатель скалярного числа, что позволяет избежать деления на ноль. Так что теперь вместо $1/x$, где x рискует оказаться равным нулю, мы используем $1/x + \alpha$, где α — положительная константа. Напомним, что обратная матрица — это аналог деления скалярных чисел.

Градиентный спуск.

Мы используем градиентный спуск или его разновидности, например стохастический градиентный спуск, когда у нас нет возможности получить решения в замкнутой форме для уравнения, в котором производная равна нулю, и когда наша задача настолько велика, что вычисление производных второго порядка потребует огромных затрат.

Методы регуляризации сокращения весов

Существуют три распространенных метода регуляризации, контролирующих размер весов, которые мы ищем на протяжении всего повествования в книге.

Гребневая регрессия.

Штрафуем норму l^2 для $\vec{\omega}$. В этом случае мы добавляем к функции потерь член $\alpha\sum_{i=1}^{n}|\omega_i|^2$, а затем минимизируем.

Регрессия лассо.

Штрафуем норму l^2 для $\vec{\omega}$. В этом случае к функции потерь добавляем член $\alpha\sum_{i=1}^{n}|\omega_i|$, затем минимизируем.

Эластичная сеть.

Это средний случай между гребневой регрессией и лассо-регрессией. Вводим один дополнительный гиперпараметр γ, способный принимать любое значение от нуля до единицы, и добавляем к функции потерь член, объединяющий через γ как гребневую регрессию, так и регрессию лассо: $\gamma\alpha\sum_{i=1}^{n}|\omega_i|^2+(1-\gamma)\alpha\sum_{i=1}^{n}|\omega_i|$. Если $\gamma=0$, то это лассо-регрессия, если $\gamma=1$ — гребневая регрессия, а если между нулем и единицей, то это можно считать некой золотой серединой.

В каких случаях используются простая линейная регрессия, гребневая регрессия, лассо-регрессия и эластичная сеть?

Здесь можно слегка запутаться и сбиться с толку множеством вариантов создания моделей машинного обучения, но не стоит отчаиваться! Пока математический анализ не скажет нам точно, какие варианты при каких обстоятельствах лучше других (или пока мы не придем к нему в результате математических вычислений и экспериментов), можно думать об огромном количестве доступных вариантов аналогично тому, как мы думаем, например, о ремонте жилья, когда нам приходится выбирать из множества доступных стройматериалов, конструкций и архитектурных решений, прежде чем мы получим конечный результат.

Поскольку мы заняты капитальным ремонтом дома, наши решения будут судьбоносными и более значимыми, чем при косметическом ремонте. Они влияют на *качество* и *функциональность* конечного результата, но тем не менее это — выбор.

Расслабьтесь, ведь существует не один путь снять покровы с искусственного интеллекта.

- Регуляризация всегда идет на пользу. В целом будет целесообразно добавить член, контролирующий размеры весов и конкурирующий с минимизацией функции потерь.
- Хорошим выбором часто оказывается гребневая регрессия, т. к. норма l^2 является дифференцируемой. Ее минимизация более устойчива, чем минимизация нормы l^1.
- Если мы решили использовать норму l^1, то несмотря на то, что в точке 0 она не дифференцируема, можно все равно определить ее *субдифференциал* или *субградиент* в точке 0. Например, можно задать его равным нулю. Заметим, что $f(x)=|x|$ дифференцируема при $x\neq 0$: она имеет производную 1 при $x>0$ и –1 при $x<0$; единственной проблемной точкой является $x=0$.
- Если предположить, что полезными являются только несколько признаков, то в качестве этапа предварительной обработки данных лучше использовать либо регрессию лассо, либо эластичную сеть для отсева менее важных признаков.
- В большинстве случаев эластичная сеть считается предпочтительнее лассо-регрессии, поскольку последняя может повести себя некорректно, если количество признаков превысит количество обучающих экземпляров или если несколько признаков сильно коррелируют между собой.

Наложение штрафа на нормы l^2 и l^1

Наша задача — найти $\vec{\omega}$, которое решит задачу минимизации:

$$\min_{\vec{\omega}} L(\vec{\omega}, \vec{\omega}_0) + \alpha\|\vec{\omega}\|.$$

Первый член направлен на снижение потерь $L(\vec{\omega}, \vec{\omega}_0)$. Второй — на уменьшение значений координат $\vec{\omega}$ вплоть до нулей. Вид нормы, которую мы выбираем для $\|\vec{\omega}\|$, определяет путь $\vec{\omega}$ к $\vec{0}$.

Если мы используем норму l^1, то координаты $\vec{\omega}$ будут уменьшаться, причем многие из них могут преждевременно исчезнуть, раньше других обратившись в ноль. То есть норма l^1 способствует разреженности: когда вес исчезает, уничтожается вклад в обучающую функцию связанного с ним признака.

График справа на рис. 4.13 показывает множества уровней ромбовидной формы $\|\vec{\omega}\|_{l^1}=|\omega_1|+|\omega_2|$ в двух измерениях (если бы имелось только две функции), а именно $|\omega_1|+|\omega_2|=c$ для различных значений c. Если алгоритм минимизации идет по пути крутого спуска, например градиентного, то мы должны двигаться в направлении, перпендикулярном множествам уровней, и, как показывает стрелка на графике, ω_2 довольно быстро становится равным нулю, поскольку, двигаясь перпендикулярно множествам уровней ромбовидной формы, мы обязательно заденем одну

из координатных осей, фактически уничтожив соответствующий признак. Тогда ω_1 будет стремиться к нулю вдоль горизонтальной оси.

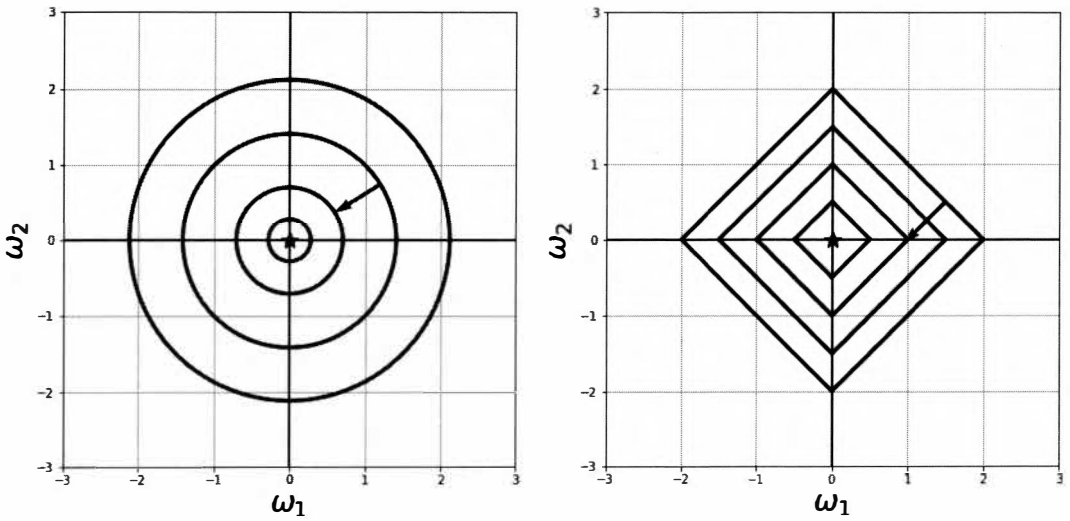


Рис. 4.13. На графике слева показаны круговые множества уровней нормы l^2 для $\vec{\omega}$, а также направление градиентного спуска к минимуму в точке (0; 0). На графике справа показаны ромбовидные множества уровней l^1 для $\vec{\omega}$, а также направление градиентного спуска к минимуму в точке (0; 0)

С нормой l^2 размеры весов уменьшаются без обязательного уничтожения. Левый график на рис. 4.13 показывает круговые множества уровней $\|\vec{\omega}\|_{l^2} = \omega_1^2 + \omega_2^2$ в двух измерениях, а именно $|\omega_1^2| + |\omega_2^2| = c$ для различных значений c. Как можно заметить, движение по траектории, перпендикулярной круговым множествам уровней, к минимуму в точке (0; 0) уменьшает значения ω_1 и ω_2, причем ни один из них не становится нулем раньше другого.

Выбор нормы зависит от конкретного случая. Важно отметить, что во всех случаях мы не регуляризируем веса смещения $\vec{\omega}_0$. Поэтому в данном разделе мы записали их отдельно в функции потерь $L(\vec{\omega}, \vec{\omega}_0)$.

Роль параметра α в регуляризации

Задача минимизации с регуляризацией сокращения веса имеет вид:

$$\min_{\vec{\omega}} L(\vec{\omega}) + \alpha \|\vec{\omega}\|.$$

Для понимания роли гиперпараметра α в регуляризации отметим следующее.

- Между первым членом, где функция потерь $L(\vec{\omega})$ выбирает ω, подгоняющие обучающую функцию к обучающим данным, и вторым членом, заботящимся

только о том, чтобы значения ω были малы, возникает конкуренция. Эти две задачи необязательно совпадают. Значения ω, уменьшающие первый член, могут увеличить второй, и наоборот.

- Если значение α будет большим, то процесс минимизации будет компенсировать это, значительно уменьшая значения ω, не обращая внимания на то, что малые значения ω уменьшат и первый член. Таким образом, чем больше α, тем важнее минимизация второго, а не первого члена, поэтому итоговая модель может не совсем подойти к данным (большая погрешность), хотя иногда это желательно (низкая дисперсия) для того, чтобы она хорошо обобщалась на неизвестные данные.
- Если же, напротив, значение α будет небольшим (скажем, близким к нулю), то можно выбрать большие значения ω, и тогда более важной станет минимизация первого члена. В данном случае минимизация приведет к таким значениям ω, при которых первый член будет удовлетворять требования, поэтому данные хорошо впишутся в модель (низкое смещение), но при этом может оказаться высокой дисперсия. В этом случае модель будет правильно работать на известных данных (она предназначена для верного подгона путем минимизации $L(\vec{\omega})$), но может плохо обобщаться на неизвестные данные.
- При $\alpha \to 0$ можно математически доказать, что решение регуляризованной задачи сходится с решением нерегуляризованной задачи.

Примеры гиперпараметров в машинном обучении

Мы уже познакомились со многими гиперпараметрами, входящими в модели машинного обучения. Перечислим все те, что вошли в нашу модель, и укажем их значения. При этом не будем забывать, что их настройка повышает эффективность моделей. В большинстве случаев можно воспользоваться рекомендуемыми значениями. Как правило, в библиотеках и программных пакетах машинного обучения они стоят по умолчанию. И все же при наличии свободного времени и ресурсов будет неплохо на этапе валидации модели поэкспериментировать с разными значениями. К гиперпараметрам относятся:

- скорость обучения при градиентном спуске;
- коэффициенты сокращения веса, например, как в гребневой регуляризации, регуляризации лассо и эластичной сети;
- количество эпох до прекращения обучения;
- размер данных, разбиваемых на обучающие, проверочные и тестовые поднаборы;
- размеры мини-пакетов при стохастическом градиентном спуске и его разновидностях;
- коэффициенты ускорения в импульсных методах;

- архитектура нейросети: количество слоев, количество нейронов на каждом слое, поведение на каждом слое (пакетная нормализация, тип функции активации), тип регуляризации (исключение, гребневая, лассо), тип сети (прямой связи, плотная, сверточная, генеративно-состязательная, рекуррентная), тип функций потерь и т. д.

Цепное правило и обратное распространение ошибки: вычисление $\nabla L\left(\vec{\omega}^{i}\right)$

Пришло время заняться черной работой и вычислить кое-что важное: градиент функции потерь, а именно $\nabla L\left(\vec{\omega}^{i}\right)$. Каким бы способом мы ни решили найти оптимальные веса — градиентным спуском, стохастическим градиентным спуском, мини-пакетным градиентным спуском или любым другим вариантом градиентного спуска — нам все равно не удастся избежать вычисления этой величины. Напомним, что в формулу функции потерь включена обучающая функция нейросети, состоящая, в свою очередь, из последовательных линейных комбинаций и композиций функций активации. В этом случае нам придется грамотно применить цепное правило. Еще при обсуждении исчислений мы пользовались цепным правилом одной переменной для производных, и сейчас нам придется каким-то образом перейти к цепному правилу нескольких переменных (понятие "нескольких" в некоторых случаях означает миллиарды).

Именно слоистая архитектура нейросети вынуждает нас остановиться и задуматься над тем, как конкретно мы будем вычислять одну производную функции потерь. Основополагающим компонентом здесь будет мощнейший алгоритм *обратного распространения ошибки* (называемый также *автоматическим дифференцированием в обратном режиме*).

Прежде чем мы запишем формулы, кратко опишем шаги обучения нейросети.

- Обучающая функция является функцией $\vec{\omega}$, поэтому результат работы нейросети после прохождения через нее точки данных (что равносильно оценке функции обучения в точке данных) будет выглядеть как: $\text{результат} = \text{функция}\left(\vec{\omega}\right)$. Он состоит из линейных комбинаций выходов узлов, за которыми следуют композиции с функциями активации, повторяющиеся на всех слоях сети. Выходной слой может иметь или не иметь функции активации, а также может состоять из одного или нескольких узлов в зависимости от конечной задачи сети.
- Функция потерь позволяет определить, насколько сильно результат обучающей функции расходится с истинным значением.
- Инициализируем обучающую функцию *случайным набором* весов $\vec{\omega}^{0}$ по описанным в предыдущих разделах правилам. Затем вычисляем потери, или ошибку вследствие использования этих конкретных значений весов. Это называется прямым проходом точки данных по сети.

- Мы хотим перейти к следующему набору весов $\vec{\omega}^1$, дающему меньшую ошибку. Движемся в направлении, противоположном вектору градиента функции потерь.
- Но как мы можем эффективно применить цепное правило со многими переменными в поиске градиента и его оценки при текущем наборе весов, учитывая, что обучающая функция встроена в функцию потерь, имеет слоистую структуру, обусловленную архитектурой нейросети, и высокую размерность?

Ответ заключается в том, что мы отправляем точку данных обратно по сети, вычисляя градиент *в обратном направлении* от выходного слоя до входного и оценивая по пути вклад каждого узла в ошибку. По сути, мы вычисляем функции $\frac{\partial L}{\partial \text{узловые функции}}$, а затем соответствующим образом настраиваем веса, обновляя их от $\vec{\omega}^0$ до $\vec{\omega}^1$. Процесс продолжается по мере поступления в сеть все новых и новых точек данных, обычно пакетами. Когда сеть просматривает весь обучающий набор данных, отсчитывается одна *эпоха*.

Метод обратного распространения ошибки практически не отличается от того, как обучается наш мозг

Когда мы сталкиваемся с новым математическим понятием, нейроны в нашем мозгу устанавливают определенные связи. В следующий раз, когда встречаем то же самое понятие, связь между этими нейронами укрепляется. Аналогично значение ω ребра, соединяющего нейроны в нейросети, увеличивается. Когда мы снова и снова видим одно и то же понятие, оно становится частью модели нашего мозга. Она не изменится, пока мы не узнаем новую информацию, которая отменит предыдущую. В этом случае связь между нейронами ослабнет. Так же и в нейросети, значение ω, связывающее нейроны, уменьшается. Поэтому подстройкой этого значения с помощью минимизации функции потерь можно установить правильные связи между нейронами.

В 1949 году вышла книга нейробиолога Дональда Хебба "Организация поведения: Нейропсихологическая теория" (Donald Hebb "Donald Hebb mentions in his 1949 book The Organization of Behavior: A Neuropsychological Theory"). Приведем цитату (в моем вольном изложении): *"Если биологический нейрон будет часто провоцировать другой нейрон, то связь между ними будет усиливаться. Другими словами, возбужденные клетки соединяются друг с другом"*.

Аналогичным образом вычислительная модель нейросети учитывает ошибку в конечном результате. Поскольку компьютеры понимают только числа, значение ω ребра будет увеличиваться, если узел способствует уменьшению ошибки, и уменьшаться, если узел способствует увеличению функции ошибки. Таким образом, увеличивая соответствующие ω, алгоритм обучения нейросети усиливает связи, уменьшающие ошибку, а уменьшая значения ω, он ослабляет связи, увеличивающие ошибку.

Почему лучше использовать обратное распространение ошибки

Обратное распространение ошибки вычисляет производную обучающей функции по каждому узлу, двигаясь по сети *в обратном направлении*. Тем самым измеряется вклад каждого узла как в обучающую функцию, так и в функцию потерь $L(\vec{\omega})$.

Самой важной формулой для запоминания является цепное правило, заимствованное из исчисления. Оно вычисляет производные *цепных функций* (или композиций функций). Цепное правило исчисления в основном касается функций, зависящих только от одной переменной ω; например, для трех цепных функций производная по ω имеет вид:

$$\frac{d}{d\omega}f_3\left(f_2\left(f_1(\omega)\right)\right)=\left\{\frac{d}{d\omega}f_1(\omega)\right\}\left\{\frac{d}{df_1}f_2\left(f_1(\omega)\right)\right\}\left\{\frac{d}{f_2}f_3\left(f_2\left(f_1(\omega)\right)\right)\right\}.$$

В нейросетях нам нужно применить цепное правило к функции потерь, зависящей от матриц и векторов переменных $\mathbf{W}$ и $\vec{\omega}_0$. Поэтому придется обобщить записанное правило до *цепного правила для множества переменных*. Простейший способ сделать это — пройти структуру сети, вычисляющую производные, в обратном направлении — от выходного слоя до входного.

Если, напротив, мы решим вычислять производные в прямом направлении по сети, то не узнаем, внесут ли они в конечный результат вклад по каждой переменной, т. к. неизвестно, соединятся ли они в графе сети. Но даже при полной связности графа веса более глубоких слоев не присутствуют в более поздних, так что вычисление их производных в последних — это пустая трата времени.

При вычислении производных в обратном направлении по сети мы начинаем с выхода и идем по ребрам графа сети назад, вычисляя производные в каждом узле. Вклад каждого узла вычисляется только по ведущим к нему и выходящим из него ребрам. С вычислительной точки зрения это значительно дешевле, поскольку сейчас мы точно знаем, как и когда узлы вносят свой вклад в результат работы сети.

В линейной алгебре умножение матрицы на вектор гораздо дешевле умножения матриц друг на друга. Мы всегда должны избегать последнего, поскольку вычислить $\mathbf{A}(\mathbf{Bv})$ дешевле, чем $(\mathbf{AB})\mathbf{v}$, хотя теоретически эти две матрицы совершенно одинаковы. В случае больших матриц и векторов это простое наблюдение позволяет сэкономить огромные средства.

Обратное распространение ошибки в деталях

Сделаем паузу и порадуемся тому, что существуют программные пакеты, позволяющие нам навсегда избавиться от необходимости самостоятельно выполнять представленные здесь вычисления. Также не забудем поблагодарить их создателей. Итак, приступим к вычислениям.

Для нейросети с h скрытыми слоями можно записать функцию потерь как функцию обучения, которая, в свою очередь, является функцией всех присутствующих в сети весов:

$$L = L\left(g\left(\mathbf{W}^1, \vec{\omega}_0^1, \mathbf{W}^2, \vec{\omega}_0^2, ..., \mathbf{W}^h, \vec{\omega}_0^h, \mathbf{W}^{h+1}, \vec{\omega}_0^{h+1}\right)\right).$$

Вычисляем частные производные L в обратном порядке: начинаем с $\frac{\partial L}{\partial \mathbf{W}^{h+1}}$ и $\frac{\partial L}{\partial \vec{\omega}_0^{h+1}}$ возвращаемся к $\frac{\partial L}{\partial \mathbf{W}^1}$ и $\frac{\partial L}{\partial \vec{\omega}_0^1}$. При этом производные берутся по каждой записи соответствующей матрицы или вектора.

Для простоты предположим, не теряя при этом общности, что сеть представляет собой регрессионную сеть, предсказывающую одно числовое значение, поэтому обучающая функция g является скалярной (не векторной). Предположим также, что по всей сети мы пользуемся одной и той же функцией активации f для каждого нейрона. Поскольку речь идет о регрессии, выходной нейрон не активируется.

Найдем производные по весам, направленным на выходной слой. Функция потерь имеет вид:

$$L = L\left(\mathbf{W}^{h+1}\vec{s}^h + \omega_0^{h+1}\right),$$

так что

$$\frac{\partial L}{\partial \omega_0^{h+1}} = 1 \times L'\left(\mathbf{W}^{h+1}\vec{s}^h + \omega_0^{h+1}\right)$$

и

$$\frac{\partial L}{\partial W^{h+1}} = \left(\vec{s}^h\right)^T L'\left(\mathbf{W}^{h+1}\vec{s}^h + \omega_0^{h+1}\right).$$

Напомним, что $\vec{s}^h$ — выход последнего слоя, следовательно, он зависит от всех весов предыдущих слоев, а именно, $\left(\mathbf{W}^1, \vec{\omega}_0^1, \mathbf{W}^2, \vec{\omega}_0^2, ..., \mathbf{W}^h, \vec{\omega}_0^h\right)$.

Для вычисления производных по весам, указывающим на последний скрытый слой, приведем их в явном виде в формуле функции потерь:

$$L = L\left(\mathbf{W}^{h+1}\left(f\left(\mathbf{W}^h\vec{s}^{h-1} + \vec{\omega}_0^h\right)\right) + \omega_0^{h+1}\right),$$

так что

$$\frac{\partial L}{\partial \vec{\omega}_0^h} = \vec{1}\left(\mathbf{W}^{h+1}f'\left(\mathbf{W}^h\vec{s}^{h-1} + \vec{\omega}_0^h\right)\right)L'\left(\mathbf{W}^{h+1}\left(f\left(\mathbf{W}^h\vec{s}^{h-1} + \vec{\omega}_0^h\right)\right) + \omega_0^{h+1}\right)$$

и

$$\frac{\partial L}{\partial \vec{\omega}_0^h} = \vec{s}^{h-1}\left(\mathbf{W}^{h+1}f'\left(\mathbf{W}^h\vec{s}^{h-1} + \vec{\omega}_0^h\right)\right)L'\left(\mathbf{W}^{h+1}\left(f\left(\mathbf{W}^h\vec{s}^{h-1} + \vec{\omega}_0^h\right)\right) + \omega_0^{h+1}\right).$$

Напомним, $\vec{s}^{h-1}$ — выход скрытого слоя перед последним скрытым слоем, следовательно, он зависит от всех весов предыдущих слоев, а именно, $\left(\mathbf{W}^1, \vec{\omega}_0^1, \mathbf{W}^2, \vec{\omega}_0^2, ..., \mathbf{W}^{h-1}, \vec{\omega}_0^{h-1}\right)$.

Систематически продолжаем процесс, пока не дойдем до входного слоя.

Оценка значимости признаков входных данных

Одной из задач аналитиков данных является оценка значимости входных переменных (признаков данных) по отношению к выходной, или целевой, переменной.

Основной вопрос, на который необходимо здесь ответить: каким будет относительное изменение выходной переменной при изменении значения определенной входной переменной?

Например, если на определенный автобусный маршрут добавить еще один автобус, повлияет ли это на общее количество пассажиров?

С точки зрения математики, это вопрос о производной, который звучит так: найти частную производную на выходе по входной переменной.

В статистике есть много работ, посвященных значимости переменных в линейных моделях (анализ чувствительности). Публикаций на тему нелинейной модели, например, как наших моделей нейросети, значительно меньше. Мы не можем на основе нелинейных моделей делать предсказания, а затем использовать анализ значимости переменных, предназначенный для линейных моделей. Многие аналитики данных, используя в процессе анализа встроенные программные пакеты, попадают в эту ловушку. Это еще одна причина, по которой необходимо стремиться к глубокому пониманию допущений моделей, на которые мы полагаемся в своих бизнес-решениях.

Итоги и перспективы

Эта глава представляет собой наш официальный переход в эру глубокого обучения в области искусственного интеллекта. Если в *главе 3* были представлены традиционные, но все еще весьма полезные модели машинного обучения, то в *главе 4* к нашему арсеналу добавились нейросети.

В обеих главах модели представлены в рамках общей математической конструкции: функция обучения, функция потерь и оптимизация, причем каждая из них была адаптирована к конкретной задаче и модели.

Применение нелинейных функций активации к каждому нейрону во всех слоях нейросети позволяет обучающей функции улавливать в данных сложные признаки, которые невозможно описать с помощью явной формулы нелинейной функции. Математический анализ, в частности универсальные теоремы аппроксимации для

нейронных сетей, подтверждает эту интуицию и предоставляет теоретическую базу, обосновывающую бурный успех нейросетей. Однако эти теоремы все еще не дают нам возможности построить специальные сети, приспособленные к конкретным задачам и наборам данных, поэтому нам приходится экспериментировать с различными архитектурами, методами регуляризации, гиперпараметрами, пока мы не получим модель нейросети, способную хорошо работать на неизвестных, новых данных.

Нейросети хорошо подходят для решения крупных задач с огромными массивами данных. В таких случаях будут нужны эффективные и вычислительно недорогие методы оптимизации, при том, что все вычисления в таких масштабах можно считать дорогими.

Одним из распространенных методов оптимизации считается стохастический градиентный спуск, а его двигателем — алгоритм обратного распространения ошибки. Или, если подробнее, алгоритм обратного распространения ошибки вычисляет градиент функции потерь (или целевой функции, когда мы добавляем регуляризацию сокращения веса) при текущем выборе веса.

Ключевой задачей оптимизации остается понимание ландшафта целевой функции, и, как показывает практика, задачи выпуклой оптимизации решаются легче, чем невыпуклой. В моделях нейросети используются, как правило, невыпуклые функции потерь.

Глава 4 — последняя фундаментальная (и достаточно длинная) глава в книге. Теперь мы можем приступать к более специализированным моделям искусственного интеллекта и более глубокой математике, когда возникнет такая необходимость.

Последующие главы не зависят друг от друга, поэтому их можно читать в любой очередности, которая покажется наиболее подходящей для вашей конкретной области применения.

Наконец, подведем итоги представленной в этой главе математики, которую нам предстоит изучить более подробно по мере продвижения в этой области.

Вероятность и мера.

Необходимы для доказательства универсальных теорем аппроксимации, которые мы рассмотрим в *главе 11*. Вероятность и мера также связаны с анализом неопределенности при исключении.

Статистика.

Шаги стандартизации входных данных при пакетной нормализации на каждом слое нейросети и итоговая реорганизация соответствующих распределений.

Оптимизация.

Градиентный спуск, стохастический градиентный спуск, выпуклые и невыпуклые ландшафты.

Исчисления на основе линейной алгебры.

Алгоритм обратного распространения ошибки — это цепное правило из области исчислений, примененное к функциям матриц переменных.

ГЛАВА 5

Сверточные нейронные сети и компьютерное зрение

Они. Могут. Видеть.
— Хала

Сверточные нейросети произвели революцию в области компьютерного зрения и обработки естественного языка. Сферы их применения безграничны (несмотря на связанные с этим этические проблемы, как, например, в вопросах наблюдения, автоматического оружия и т. д.). К ним относятся самоуправляемые автомобили, умные беспилотники, распознавание лиц, распознавание речи, медицинская визуализация, генерация звука, генерация изображений, робототехника и т. д.

В этой главе мы начнем с простых определений и трактовок *свертки* и *кросс-корреляции* и обратим внимание на то, что в терминологии машинного обучения эти две несколько разные математические операции часто смешиваются. Впрочем, мы тоже, грешным делом, занимаемся этим, но на то у нас есть веские причины.

Затем мы применим свертку для фильтрации данных с решетчатой структурой, для чего она идеально подходит, например, для данных временны́х рядов (одномерных), аудиоданных (одномерных), изображений (двумерных, если изображения полутоновые, и трехмерных, если они цветные, причем дополнительное измерение соответствует красному, зеленому и синему каналам). С одномерными данными мы применяем одномерные свертки, с двумерными — двумерные (для простоты и краткости в этой главе мы не будем заниматься трехмерными свертками, соответствующими трехмерным цветным изображениям, которые называются *тензорами*). Другими словами, мы адаптируем сеть к форме данных — процедура, внесшая огромный вклад в успех сверточных нейросетей. Это отличается от принудительной адаптации данных к форме входного сигнала сети, например от сжатия двумерного изображения в один длинный вектор, чтобы приспособить его к сети, принимающей на вход только одномерные данные. В последующих главах мы увидим, что то же самое относится и к успешной работе графовых нейросетей на данных с графоподобной структурой.

Далее мы включаем свертку в архитектуру нейросети с прямой связью. Свертка действует так, что сеть становится связанной не полностью, а *локально*. С точки зрения математики, матрица, содержащая веса для каждого слоя, не является *плотной* (поэтому большинство весов для сверточных слоев равны нулю). Кроме того, в отличие от полносвязных нейронных сетей, где каждому входу присваива-

ется свой вес, здесь веса имеют одинаковые значения (*совместное использование весов*). Поэтому матрица, содержащая веса сверточного слоя, в большинстве своем нулевая, а ненулевые части локализованы и совместно используют одинаковые значения. Это прекрасно подходит для работы с изображениями или аудиоданными, т. к. большинство информации содержится локально. Более того, это значительно сокращает количество весов, которые нужно сохранять и вычислять на этапе оптимизации, так что сверточные нейросети представляют собой идеальное решение для данных с огромным количеством входных признаков (напомним, что в изображениях признаком считается каждый пиксел).

Далее мы рассмотрим *пулинг* (объединение) — еще один слой, характерный для архитектуры сверточных нейросетей. Как и в предыдущей главе, многослойная структура и нелинейность каждого слоя позволяют извлекать из изображений все более сложные признаки, значительно расширяя таким образом возможности компьютерного зрения.

Как только мы поймем базовую анатомию сверточной нейросети, можно будет легко применить ту же самую математику к задачам обработки естественного языка, например, при анализе настроения, распознавании речи, генерации аудио и др. Одно и то же математическое решение задач компьютерного зрения и обработки естественного языка можно сравнить с впечатляющей способностью нашего мозга физически изменяться в зависимости от обстоятельств, личного опыта и способов мышления (виртуальная симуляция мозга). Даже когда некоторые участки мозга повреждены, другие участки могут взять на себя новые функции. Например, при нарушении зрения области мозга, отвечающие за зрение, могут начать выполнять задачи, связанные со слухом или запоминанием. В нейронауке это называется *нейропластичностью*. Мы пока еще слишком далеки от полного понимания того, как устроен и функционирует наш мозг, но самое простое объяснение этого феномена заключается в том, что каждый нейрон мозга выполняет одну и ту же базовую функцию, аналогично тому, как нейроны в нейросети выполняют один базовый математический расчет (хотя фактически два — линейная комбинация и последующая активация), а различные нейронные связи в нескольких слоях создают видимую сложность восприятия и поведения.

В основе сверточных нейросетей лежит неокортекс мозга, отвечающий за зрение. Успех в классификации изображений (AlexNet2012, **https://oreil.ly/QxjXD**) в 2012 году вернул искусственный интеллект в основное русло, воодушевил множество людей и, в конце концов, привел нас сюда. Поэтому было бы неплохо в качестве иллюстрации к этой главе почитать в свободное время что-нибудь о зрительных зонах неокортекса и проследить аналогию со сверточными нейросетями, предназначенными для компьютерного зрения.

Свертка и кросс-корреляция

Свертка и кросс-корреляция несколько отличаются друг от друга, т. к. оценивают разные параметры сигнала, который может быть цифровым изображением, цифро-

вым аудио- или иным сигналом. Но при использовании симметричной функции k, которая называется *фильтром* или *ядром*, они выполняются абсолютно одинаково. Если коротко: свертка *переворачивает* фильтр и ведет им по функции, а кросс-корреляция ведет фильтром, *не переворачивая* его. Безусловно, при симметричном фильтре свертка и кросс-корреляция в точности совпадут. Преимуществом переворачивания ядра является то, что операция свертки становится коммутативной, и это, в свою очередь, хорошо подходит для записи теоретических доказательств. Но в контексте нейросетей коммутативность не важна сразу по трем причинам.

- Во-первых, операция свертки выполняется в нейросети обычно не одна, а в комплексе с другими нелинейными функциями, так что мы теряем коммутативность независимо от того, переворачиваем мы ядро или нет.
- Во-вторых, нейросеть *тренируется* значениям записей в ядре в процессе обучения, т. е. она обучается правильным значениям в правильных местах, и переворачивание уже не требуется.
- В-третьих, что немаловажно в практической реализации, в сверточных сетях часто используется *многоканальная свертка*, например, на входе может быть цветное изображение с красно-зелено-синими (или RGB) каналами и даже видео с каналами пространства RGB и одним временны́м каналом. Кроме того, свертка выполняется в пакетном режиме, т. е. входные векторы, изображения, видео или другие типы данных принимаются пакетами, и одновременно с этим выполняются параллельные операции свертки. И даже с учетом переворачивания ядра мы не можем утверждать об их коммутативности, если у них не будет одинакового количества входных и выходных каналов. Но такое на практике встречается довольно редко, т. к. выходы нескольких каналов обычно полностью или частично суммируются, что и дает разницу в их количестве.

В силу этих причин большинство библиотек машинного обучения не переворачивают ядро при свертке, реализуя, по сути, кросс-корреляцию, но представляя ее как свертку. Мы сделаем то же самое.

Операция свертки между двумя действительными функциями k (фильтр) и f определяется как:

$$(k * f)(t) = \int_{-\infty}^{\infty} f(s)k(-s+t)ds = \\ = \int_{-\infty}^{\infty} (-s+t)k(s)ds.$$

Дискретный аналог для дискретных функций:

$$(k * f)(n) = \sum_{s=-\infty}^{\infty} f(s)k(-s+n) = \\ = \sum_{s=-\infty}^{\infty} f(-s+n)k(s).$$

Операция кросс-корреляции между двумя действительными функциями k (фильтр) и f определяется как:

$$(k \star f)(t) = \int_{-\infty}^{\infty} f(s)k(s+t)ds =$$

$$= \int_{-\infty}^{\infty} f(s-t)k(s)ds.$$

Дискретный аналог:

$$(k \star f)(n) = \sum_{s=-\infty}^{\infty} f(s)k(s+n) =$$

$$= \sum_{s=-\infty}^{\infty} f(s-n)k(s).$$

Отметим, что формулы, определяющие свертку и кросс-корреляцию, выглядят абсолютно одинаково, за исключением того, что в свертке место s занимает $-s$. Это означает переворачивание соответствующей функции (перед сдвигом). Также важно отметить, что индексы, входящие в интеграл и сумму свертки, складываются с t или n, чего нельзя сказать о кросс-корреляции. Таким образом, свертка считается коммутативной в том смысле, что $(f * k)(n) = (k * f)(n)$, а кросс-корреляция — необязательно. Мы рассмотрим коммутативность подробнее чуть дальше.

Каждый раз, когда мы сталкиваемся с новым математическим объектом, то прежде чем приступать к его детальному изучению, было бы неплохо сделать паузу и задать себе несколько вопросов. Так у нас получится заложить прочный математический фундамент, не углубляясь в сложную техническую математику.

С каким математическим объектом мы имеем дело?

При свертке и кросс-корреляции в дискретном случае мы имеем дело с бесконечными суммами и в случае континуума — с интегралами по бесконечным областям. Отсюда вытекает следующий вопрос.

Учитывая, что мы суммируем по бесконечным областям, какими функциями можно воспользоваться, чтобы вычисления не разрастались до бесконечности?

Другими словами, какие функции хорошо оперируют бесконечными суммами и интегралами? Начнем с простых ответов, например, когда f и k записываются в компактной форме (имеют нули только в конечной части области), или когда для того, чтобы бесконечная сумма или интеграл сошлись, f и k достаточно быстро сокращаются. Чаще всего подобных простых решений бывает достаточно для таких задач, как обработка изображений и аудиоданных, а также фильтров, которые при этом используются. Но если простой ответ не подходит к нашему конкретному случаю, мы ищем более обобщенные ответы. Они представлены в виде теорем и доказательств. Не стоит искать наиболее обобщенные ответы с самого начала, ведь в большинстве случаев они опираются на большую математическую теорию, формирование и воплощение которой заняло не одно столе-

тие в поисках ответов на бесконечное множество вопросов. Выбрать с самого начала максимально обобщенный способ будет явным перебором вопреки логике и истории, отнимающим к тому же много времени и ресурсов. Это не тот путь, по которому естественным образом развиваются математика и анализ. Более того, когда нам встречается человек, говорящий на самом обобщенном техническом языке вне соответствующего контекста и без предпосылок к такому уровню обобщенности и сложности, мы просто перестаем воспринимать его и спокойно живем дальше, пока он не запутал нас окончательно.

Как используется этот математический объект?

Операция свертки имеет широкое применение, начиная от чисто теоретической математики и заканчивая прикладными науками, технической разработкой, проектированием систем. В математике свертка встречается в таких областях, как дифференциальные уравнения, теория меры, теория вероятностей, статистика, анализ, численная линейная алгебра. В прикладных науках и технике она используется в акустике, спектроскопии, обработке изображений и компьютерном зрении, а также в обработке сигналов при разработке и реализации фильтров с конечной импульсной характеристикой.

Чем полезен этот математический объект конкретно в моей области знаний и как его применять в моем конкретном случае?

В контексте ИИ мы будем пользоваться сверткой для создания сверточных нейросетей как для одномерных текстовых и аудиоданных, так и для двумерных изображений. Те же самые идеи применимы к любому типу высокоразмерных данных, в которых большинство информации содержится локально. Мы используем сверточные нейросети в двух ситуациях: для *понимания* и *генерации* изображений, текстов и аудиоданных.

Кроме того, в контексте данных и их распределений мы используем результат, связанный с распределением вероятности суммы двух независимых случайных величин. Если μ и ν — вероятностные меры по топологической группе $(\mathbb{R}, +)$, X и Y — две независимые случайные величины, распределения которых равны μ и ν, то свертка $\mu * \nu$ является вероятностным распределением суммы случайных величин $X + Y$. Подробнее об этом мы поговорим в *главе 11*, посвященной вероятности.

Как появился этот математический объект?

Будет не лишним потратить немного дополнительного времени, чтобы узнать историю и хронологию того, когда, как и почему впервые появился интересующий нас объект, а также основные связанные с ним результаты. Другими словами, вместо изучения математического объекта с помощью ряда сухих лемм, положений и теорем, лишенных, как правило, всякого контекста, мы знакомимся с его историей, прослеживаем взлеты и падения, пережитые математиками на пути к его появлению. Самое ценное, что можно подчерпнуть отсюда: математика органично развивается по мере того, как мы пытаемся найти ответы на различ-

ные вопросы, установить связи и глубже понять то, чем мы должны пользоваться. Современный математический анализ сложился из попыток ответить на довольно простые вопросы, связанные с синусным и косинусным рядами Фурье (разложение функции на частотные составляющие), которые для многих типов функций оказались не совсем простыми. Например, в каких случаях можно чередовать интеграл и бесконечную сумму, и вообще, что такое интеграл и что такое dx?

Для большинства людей, особенно для тех, кто боится математики, может стать неожиданностью, что на пути к пониманию некоторые крупнейшие представители математики, в том числе отцы и матери целых областей, допускали многочисленные ошибки, которые впоследствии исправляли либо они сами, либо другие люди, пока теория не обрела свою окончательную форму.

Одно из первых применений интеграла свертки появилось в 1754 году в работе Даламбера, посвященной выводу теоремы Тейлора. Позже, в 1797–1800 годах, его использовал Сильвестр Франсуа Лакруа в книге "Трактат о дифференциалах и рядах" (Sylvestre François Lacroix "Treatise on Differences and Series"), входящей в энциклопедическую серию "Элементарный трактат по дифференциальному и интегральному исчислению" ("An Elementary Treatise on Differential Calculus and Integral Calculus"). Очень скоро операции свертки появились в работах таких известных математиков, как Лаплас, Фурье, Пуассон, Вольтерра. Общим здесь является то, что все эти исследования связаны с интегралами, производными и рядами функций. Другими словами, мы имеем исчисление и, опять же, разложение функций на частотные составляющие (ряд и преобразование Фурье).

Какие важнейшие операции, манипуляции и/или теоремы, связанные с этим математическим объектом, необходимо знать, прежде чем вдаваться в детали?

Вещи не становятся известными, или вирусными, если они не представляют большой ценности для множества людей. Свертка является очень простой, но в то же время довольно ценной операцией, которая легко обобщается на более сложные математические объекты, такие как меры и распределения. Она коммутативна, ассоциативна, дистрибутивна по композиции и скалярному умножению; ее интеграл становится произведением, а производная преобразуется в дифференцирование только одной из составляющих ее функций.

Инвариантность и эквивариантность переноса

Эти свойства сверточных нейросетей позволяют выявлять схожие признаки в разных частях изображения. То есть паттерн, присутствующий в одной области изображения, легко распознается в других областях. Основная причина заключается в том, что на сверточном слое нейросети мы выполняем свертку с одним и тем же фильтром (его еще называют ядром или шаблоном) по всему изображению, улавливая одни и те же паттерны (например, ребра, а также горизонтальные, вертикальные и диагональные ориентации). Фильтр имеет фиксированный набор весов. На-

помним, что это не относится к полносвязным нейросетям, когда для разных пикселов изображения нам пришлось бы использовать разные веса. Поскольку в фильтрации изображений применяется не матричное умножение, а свертка, у нас появляется преимущество *инвариантности переноса*, т. к. чаще всего бывает важно только наличие паттерна, независимо от его расположения. На языке математики инвариантность переноса выглядит следующим образом:

$$trans_a(k) * f = k * trans_a(f) = trans_a(k * f)(t),$$

где $trans_a$ — перенос функции по a. В контексте искусственного интеллекта это означает, что при наличии фильтра, предназначенного для выделения определенного признака изображения, его свертка с переведенным изображением (в горизонтальном или вертикальном направлениях) аналогична фильтрации изображения с его *последующим* переносом. Это свойство иногда называется *эквивариантностью переноса*, а инвариантность переноса приписывается пулинг-слою, который часто встроен в архитектуру сверточных нейросетей. Мы обсудим это позже в текущей главе. В любом случае тот факт, что мы применяем один фильтр (один набор весов) ко всему изображению на каждом слое, означает, что при наличии определенного паттерна мы выявляем его в разных областях изображения. То же самое относится и к аудиоданным, и к любым другим данным с решетчатой структурой.

Свертка в обычном пространстве — это произведение в частотном пространстве

Преобразование Фурье от свертки двух функций представляет собой произведение преобразований Фурье каждой функции с точностью до масштаба. Проще говоря, преобразование Фурье разлагает функцию на ее частотные компоненты (подробнее о преобразовании Фурье мы поговорим в *главе 13*). Поэтому операция свертки не создает новых частот, а частоты, присутствующие в функции свертки, являются произведением частот составляющих ее функций.

Занимаясь математикой, мы постоянно пополняем арсенал полезных инструментов для решения различных задач, поэтому в зависимости от области нашей специализации именно на этом этапе мы переходим к изучению смежных, но более сложных результатов, в числе которых *циклические свертки периодических функций*, предпочитаемые алгоритмы вычисления сверток и др. Оптимальным способом детально рассмотреть свертку сквозь призму ИИ является проектирование сигналов и систем, о чем мы поговорим в следующем разделе.

Свертка с точки зрения проектирования системы

Вокруг нас системы, каждая из которых взаимодействует с окружающей средой и предназначена для выполнения определенной задачи. В качестве примера можно

привести системы отопления, вентиляции и кондиционирования воздуха в зданиях, адаптивные системы круиз-контроля в автомобилях, городские транспортные системы, системы орошения, различные системы связи, системы безопасности, навигации, центры обработки данных и т. д. Некоторые системы взаимодействуют друг с другом, другие — нет. Какие-то могут быть очень большими, какие-то — маленькими и простыми, скажем термостат, который с помощью датчиков получает сигналы из окружающей среды, обрабатывает их и выдает другие сигналы, например на приводные устройства.

Операция свертки применяется в проектировании и анализе простейших систем, которые обрабатывают входной и вырабатывают выходной сигналы при наложении на них двух определенных ограничений: *линейность* и *инвариантность по времени*. Линейность и инвариантность по времени становятся линейностью и *инвариантностью переноса, или сдвига,* в случае *сигналов, зависящих от пространства*, такими как изображения, а не *от времени*, например электрические или звуковые сигналы. Важно отметить, что видеоизображение зависит как от пространства (два или три пространственных измерения), так и от времени. В таком контексте линейность относится к выходу масштабированного сигнала (усиленного или ослабленного) и выходу двух наложенных сигналов. Инвариантность по времени и инвариантность переноса связаны с выходом как задержанного (зависящего от времени), так и транслированного, или сдвинутого (зависящего от пространства), сигналов. Мы рассмотрим их подробно в следующем разделе.

Как линейность, так и инвариантность переноса или инвариантность по времени — довольно мощные характеристики. Они позволяют найти выход системы для *любого сигнала* только при известном выходе простого *импульсного сигнала*, который называется *импульсной характеристикой системы*. Можно получить выход системы на *любой входной сигнал*, просто свернув этот сигнал с импульсной характеристикой системы. Таким образом, наложением условий линейности и инвариантности по времени или переносу можно значительно упростить анализ систем обработки сигналов.

И здесь назревает вопрос: насколько реальны такие чудодейственные условия? Другими словами, насколько распространены линейные и инвариантные по времени/переносу системы, или хотя бы приблизительно линейные и приблизительно инвариантные по времени/переносу системы? А может быть, большинство реалистичных систем — нелинейные и сложные? К счастью, мы лично участвуем в проектировании систем, поэтому можем просто решить проектировать системы с такими свойствами. В качестве примера можно привести электрическую цепь, состоящую из конденсаторов, резисторов, индукторов и линейных усилителей. По сути, это математически эквивалентно идеальной механической системе, состоящей из пружины, массы и демпфера. Другие актуальные для нас примеры — обработку и фильтрацию различных типов сигналов и изображений — мы рассмотрим в следующих разделах.

Свертка и импульсная характеристика линейных и инвариантных по переносу систем

Сформулируем понятия линейной системы и инвариантной по времени/переносу системы, а затем рассмотрим, как естественным образом возникает операция свертки при попытке количественно оценить реакцию системы, обладающей этими свойствами, на *любой* сигнал. С точки зрения математики, система — это функция H, принимающая входной сигнал x и выдающая выходной сигнал y. Сигналы x и y могут зависеть как от времени или пространства (одно- или многомерного), так и от обоих этих параметров. Если мы обеспечим линейность такой функции, то можем утверждать следующее:

1. Выход масштабированного входного сигнала есть не что иное, как масштабирование исходного выходного сигнала: $H(ax) = aH(x) = ay$.
2. Выход двух наложенных сигналов есть не что иное, как наложение двух исходных выходов: $H(x_1 + x_2) = H(x_1) + H(x_2) = y_1 + y_2$.

Если мы обеспечим инвариантность по времени/переносу, то можно утверждать, что выход задержанного/переведенного/сдвинутого сигнала есть не что иное, как задержанный/переведенный/сдвинутый исходный выход: $H(x(t - t_0)) = y(t - t_0)$.

Можно использовать эти условия, если представить любой произвольный сигнал, дискретный или непрерывный, в виде наложения множества *импульсных сигналов* различной амплитуды. Таким образом, если можно измерить выход системы на один импульсный сигнал, что называется *импульсной характеристикой системы*, то этой информации будет достаточно для оценки отклика системы на любой другой сигнал. Это ложится в основу весьма обширной теории. В этой главе разберем только дискретный случай, т. к. сигналы, которые рассматриваются применительно к искусственному интеллекту (например, обработка естественного языка, взаимодействие человека и машины, компьютерное зрение), будь то одномерные звуковые сигналы или двух- или трехмерные изображения, — это дискретные сигналы. Аналогично мы поступаем в случае непрерывных сигналов, за исключением того, что рассматриваем бесконечно малые, а не дискретные шаги, используем не суммы, а интегралы, а также выполняем условия непрерывности (или любые другие условия, необходимые для четкого определения интегралов). По сути, в случае непрерывных сигналов задача дополнительно усложняется необходимостью верно определить *импульс* математически обоснованным способом, т. к. он не является функцией в обычном смысле этого слова. К счастью, существует множество математических способов его четкого определения, например теория распределений, или определение его как оператора, воздействующего на обычные функции, или как меры с помощью интегралов Лебега. Однако я предпочитаю не касаться таких понятий, как меры, интегралы Лебега, и не вникать глубоко в математическую теорию до тех пор, пока нам действительно не понадобится их дополнительная функциональность, так что дискретного случая будет вполне достаточно.

Определим единичный импульс $\delta(k)$ как ноль для каждого ненулевого k и как единицу при $k=0$, а его отклик определим как $H(\delta(k))=h(k)$. Тогда $\delta(n-k)$ будет нулем для каждого k и единицей для $k=n$. Это представляет собой единичный импульс, расположенный в точке $k=n$. Таким образом, $x(k)\delta(n-k)$ — импульс амплитуды $x(k)$ в точке $k=n$. Можно записать входной сигнал $x(n)$ в виде:

$$x(n)=\sum_{k=-\infty}^{\infty}x(k)\delta(n-k).$$

Это уравнение кажется несколько запутанным способом записи сигнала, и это так, но оно очень полезное, поскольку указывает на то, что любой дискретный сигнал можно выразить как бесконечную сумму единичных импульсов, правильно масштабируемых в нужных местах. Теперь, используя предположения о линейности и инвариантности переноса H, можно записать реакцию системы на сигнал $x(n)$ в таком виде:

$$\begin{aligned}H(x(n))&=H\left(\sum_{k=-\infty}^{\infty}x(k)\delta(n-k)\right)=\\&=\sum_{k=-\infty}^{\infty}x(k)H(\delta(n-k))=\\&=\sum_{k=-\infty}^{\infty}x(k)h(n-k)=\\&=(x*h)(n)=\\&=y(n).\end{aligned}$$

Таким образом, линейная инвариантная по переносу система полностью описывается импульсной характеристикой $h(n)$. Но есть и другой способ, не зависящий от линейных и инвариантных по переносу систем, который нам будет весьма полезен в следующих разделах. Утверждение:

$$y(n)=(x*h)(n)$$

говорит о том, что сигнал $x(n)$ можно преобразовать в сигнал $y(n)$ после его свертки с фильтром $h(n)$. Таким образом, при тщательной настройке фильтра $h(n)$ можно получить $y(n)$ с некими желаемыми характеристиками или *извлечь* из сигнала $x(n)$ *конкретные признаки*, например все ребра. Более того, применяя различные фильтры $h(n)$, можно из одного и того же сигнала $x(n)$ извлечь различные признаки. Мы рассмотрим эти понятия подробнее в следующих разделах, а пока запомним: по мере прохождения информации, например сигналов или изображений, по сверточной нейросети, на каждом сверточном слое извлекаются (отображаются) различные признаки.

Прежде чем мы оставим линейные и инвариантные по времени/переносу системы, отметим, что они имеют очень простой отклик на синусоидальные входные сигналы. Если на вход системы подается синусоидальная волна с заданной частотой, то на выходе также будет синусоидальная волна с той же частотой, но, возможно, с другой амплитудой и фазой. Кроме того, знание импульсной характеристики системы позволяет вычислить ее частотную характеристику, которая является характеристикой системы для синусоидальных волн на всех частотах, и наоборот. То есть определение импульсной характеристики системы позволяет вычислить ее частотную характеристику, а определение частотной характеристики — ее импульсную характеристику, которая, в свою очередь, полностью определяет реакцию системы на любой произвольный сигнал. Эта связь весьма полезна как с теоретической, так и с прикладной точки зрения; она подводит нас вплотную к преобразованиям Фурье и представлениям сигналов в частотной области. Проще говоря, частотная характеристика линейной инвариантной по переносу системы — это простое преобразование Фурье ее импульсной характеристики. Мы не будем углубляться в вычислительные детали, поскольку эти понятия не имеют существенного значения для остальной части книги; тем не менее важно знать об этих связях и понимать, как вещи, казалось бы, из разных областей, объединяются и соотносятся друг с другом.

Свертка и одномерные дискретные сигналы

Рассмотрим операцию свертки и выясним, как она создает новый сигнал из входного наложением на него фильтра (ядра). Мы не будем переворачивать ядро, т. к. установили, что с точки зрения нейросети это не имеет никакого значения. Начинаем с одномерного дискретного сигнала $x(n)$, затем свертываем его с фильтром (ядром) $k(n)$ и получаем новый сигнал $z(n)$. Продемонстрируем, как получить $z(n)$ по одной записи, двигая $k(n)$ вдоль $x(n)$. Для простоты пусть $x(n)=(x_0, x_1, x_2, x_3, x_4)$ и $k(n)=(k_0, k_1, k_2)$. В этом примере входной сигнал $x(n)$ имеет только пять записей, а ядро — три. На практике, например при обработке сигналов, фильтрации изображений или в нейросетях ИИ, входной сигнал $x(n)$ на порядки больше фильтра $k(n)$. Мы увидим это очень скоро, но пример здесь приведен только для иллюстрации. Вспомним формулу для дискретной кросс-корреляции, которая представляет собой свертку без переворачивания ядра:

$$(k \star f)(n) = \sum_{s=-\infty}^{\infty} x(s)k(s+n) =$$

$$= \ldots + x(-1)k(-1+n) + x(0)k(n) + x(1)k(1+n) + x(2)k(2+n) + \ldots$$

Поскольку ни $x(n)$, ни $k(n)$ не имеют бесконечно большого числа записей, а сумма бесконечна, то мы делаем вид, что записи равны нулю во всех случаях, когда индексы не определены. Новый отфильтрованный сигнал, полученный в результате

свертки, будет иметь только нетривиальные записи с индексами –4, –3, –2, –1, 0, 1, 2. Запишем каждую из них:

$$(k \star f)(-4) = x_4 k_0;$$
$$(k \star f)(-3) = x_3 k_0 + x_4 k_1;$$
$$(k \star f)(-2) = x_2 k_0 + x_3 k_1 + x_4 k_2;$$
$$(k \star f)(-1) = x_1 k_0 + x_2 k_1 + x_3 k_2;$$
$$(k \star f)(0) = x_0 k_0 + x_1 k_1 + x_2 k_2;$$
$$(k \star f)(1) = x_0 k_1 + x_1 k_2;$$
$$(k \star f)(2) = x_0 k_2.$$

Эту операцию будет проще понять, если мы мысленно представим себе, как фиксируем сигнал $x(n) = (x_0, x_1, x_2, x_3, x_4)$ и справа налево перемещаем по нему фильтр $k(n) = (k_0, k_1, k_2)$. В сжатом виде это можно описать с помощью системы представления, или нотации, линейной алгебры, умножая входной вектор $x(n)$ на матрицу специального вида — *матрицу Тёплица*, которая содержит веса фильтров. Подробнее об этом мы расскажем далее в этой главе.

Когда мы двигаем фильтр вдоль сигнала, выходной сигнал достигнет пика в точках, где $x(n)$ и $k(n)$ совпадают, поэтому можно сконструировать фильтр таким образом, чтобы он улавливал определенные паттерны в $x(n)$. Таким образом, свертка (в неперевернутом виде) позволяет оценить сходство между сигналом и ядром.

Можно и по-другому взглянуть на кросс-корреляцию, или неперевернутую свертку: каждая запись выходного сигнала представляет собой *средневзвешенное значение* записей входного сигнала. Таким образом, мы подчеркиваем как линейность этого преобразования, так и то, что можем выбирать веса в ядре таким образом, чтобы выделить определенные признаки на фоне остальных. Нагляднее всего это видно на примере обработки изображений, о которой речь пойдет далее, но для этого необходимо написать формулу свертки (неперевернутой) в двух измерениях. В нотации линейной алгебры дискретная двумерная свертка сводится к умножению двумерного сигнала на матрицу специального вида — *дважды блочно-циркулянтную матрицу*, которую мы также рассмотрим далее в этой главе. Важно запомнить, что умножение на матрицу является линейным преобразованием.

Свертка и двумерные дискретные сигналы

Операция свертки (неперевернутой) в двухмерном пространстве имеет вид:

$$(k \star x)(m, n) = \sum_{q=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} x(m+q, n+s) k(q, s).$$

Например, запись (2, 1) свертки (неперевернутой) между **A** — матрицей 4×4 и **K** — ядром 3×3:

$$\mathbf{A}\star\mathbf{K}=\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03}\\ a_{10} & a_{11} & a_{12} & a_{13}\\ a_{20} & \boxed{a_{21}} & a_{22} & a_{23}\\ a_{30} & a_{31} & a_{32} & a_{33}\end{pmatrix}\star\begin{pmatrix} k_{00} & k_{01} & k_{02}\\ k_{10} & \boxed{k_{11}} & k_{12}\\ k_{20} & k_{21} & k_{22}\end{pmatrix}$$

— это $z_{21}=a_{10}k_{00}+a_{11}k_{01}+a_{12}k_{02}+a_{20}k_{10}+a_{21}k_{11}+a_{22}k_{12}+a_{30}k_{20}+a_{31}k_{21}+a_{32}k_{22}$.

Для наглядности представим, как помещаем ядро **K** точно поверх матрицы **A** с центром k_{11} поверх a_{21}, т. е. записи **A** с нужным индексом, затем перемножаем записи, находящиеся друг на друге, и складываем все результаты. Отметим, что мы вычислили только одну запись выходного сигнала, т. е. если бы мы работали с изображениями, то это было бы значение только одного пиксела отфильтрованного изображения. Но нам нужны и все остальные! Для этого требуется выяснить, какие индексы относятся к действительным сверткам. Под действительными мы понимаем *полные*, т. е. учитывающие все записи ядра при вычислении. Слово *"действительные"* несколько вводит в заблуждение, т. к. все записи будут действительными, если мы заполним границы матрицы **A** нулями. Для того чтобы найти индексы, учитывающие все ядро, вспомним, как мы мысленно помещали **K** точно поверх **A**, причем центр **K** располагался в индексе, который мы хотели вычислить. При таком расположении остаток **K** не должен выходить за границы **A**, поэтому для нашего примера хорошими индексами будут считаться (1, 1), (1, 2), (2, 1), (2, 2), и в результате мы получаем:

$$\mathbf{Z}=\begin{pmatrix} z_{11} & z_{12}\\ z_{21} & z_{22}\end{pmatrix}.$$

Это означает, что при работе с изображениями отфильтрованное изображение **Z** будет меньше по размеру, чем оригинальное изображение **A**. Если мы хотим получить одинаковый размер, то до наложения фильтра нужно добавить нули к оригинальному изображению. Для нашего примера достаточно одного слоя нулей по всей границе **A**, но если **K** будет больше, тогда потребуется больше слоев нулей. Ниже приведена матрица **A**, дополненная одним слоем нулей:

$$\mathbf{A}_{\text{дополненная}}=\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0\\ 0 & a_{00} & a_{01} & a_{02} & a_{03} & 0\\ 0 & a_{10} & a_{11} & a_{12} & a_{13} & 0\\ 0 & a_{20} & a_{21} & a_{22} & a_{23} & 0\\ 0 & a_{30} & a_{31} & a_{32} & a_{33} & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{pmatrix}.$$

И хотя дополнение нулями считается самым простым и распространенным методом, есть и другие способы сохранить размер выходного изображения.

Отражение.

Можно добавить слой или несколько слоев не с нулями, а с такими же значениями, как у граничных пикселов изображения, и тех, что под ними, и так далее, если нужно больше. То есть вместо выключения пикселов за границей изображения можно расширить изображение за счет тех же пикселов, что уже присутствуют на его границе или в ее окрестности.

Заворачивание.

Применяется в случаях периодических сигналов. С точки зрения математики, это циклическая свертка, циркулянтная матрица, дискретное преобразование Фурье, дающее собственные значения циркулянтной матрицы, и матрица Фурье, содержащая собственные векторы в качестве столбцов. Мы не будем углубляться в эту тему, но напомним, что периодичность обычно упрощает ситуацию и наводит на мысль о периодических волнах, которые, в свою очередь, приводят к Фурье.

Многоканальность.

Можно применить более одного *канала* независимых фильтров (весовых матриц), каждый из которых дискретизирует оригинальный вход, после чего объединить их выходы.

Фильтрация изображений

На рис. 5.1 приведен пример, как при свертке одного и того же изображения с разными ядрами извлекаются различные признаки изображения.

Например, третье ядро в таблице имеет в центре 8, а остальные записи равны –1. Это означает, что данное ядро делает текущий пиксел в 8 раз интенсивнее, а затем вычитает из него значения всех окружающих его пикселов. Если мы находимся в однородной области изображения, т. е. все пикселы равны или очень близки по значению, то процесс выдаст ноль, возвращая черный, или выключенный, пиксел. Но когда этот пиксел находится на границе ребра, например на границе глаза или морды оленя, выход свертки будет иметь ненулевое значение, т. е. это будет яркий пиксел. Если мы применим этот процесс ко всему изображению, то получим новое изображение со множеством ребер, трассированных яркими пикселами, в то время как остальная часть изображения будет темной.

Как видно из таблицы, выбор ядра, способного выявлять ребра, размывать и т. д., не является однозначным. В этой же таблице приведены двумерные дискретные гауссовы фильтры размытия. При дискретизации одномерной гауссовой функции ее симметрия не нарушается, но при дискретизации двумерной гауссовой функции мы теряем ее радиальную симметрию, т. к. вынуждены аппроксимировать ее естественную круглую или эллиптическую форму квадратной матрицей. Заметим, что гауссова функция имеет пик в центре и затухает по мере удаления от центра. Более того, площадь под его кривой (поверхность для двух измерений) равна единице. Это дает общий эффект усреднения и сглаживания (удаления помех) при его свертке с другим сигналом. За это приходится платить удалением резких ребер, что как раз и является размытием (можно представить, как резкое ребро заменяется плавно убывающим средним значением самого себя и всех окружающих пикселов на рас-

стоянии нескольких стандартных отклонений от центра). Чем меньше стандартное отклонение (или дисперсия, которая представляет собой квадрат стандартного отклонения), тем больше деталей можно сохранить в изображении.

Операция	Ядро ω	Результат $g(x, y)$
Идентификация	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Выделение контуров	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Повышение резкости	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Размытие по рамке (нормализованное)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Размытие по Гауссу 3×3 (аппроксимация)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Размытие по Гауссу 5×5 (аппроксимация)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Нерезкое маскирование 5×5. На основе гауссова размытия с эффектом 1 и порогом 0 = (без маски изображения)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Заменим пиксел, например, a_{22} на a_{11}-a_{13}-a_{31}+a_{33}. В случае отсутствия ребер эта операция погасит (сделает черным) множество пикселов.

Делаем пиксел очень интенсивным, а затем вычитаем сумму пикселов по вертикали и горизонтали в его окрестности.

Каждый пиксел становится средним значением всех окружающих его пикселов, поэтому изображение получается размытым.

Рис. 5.1. Обработка изображения различными фильтрами (источник изображения: **https://oreil.ly/S2Nfu**[1])

[1] Аналогичное описание можно найти здесь: **http://ru.ruwiki.ru/wiki/Цифровая_обработка_изображений**. — *Прим. ред.*

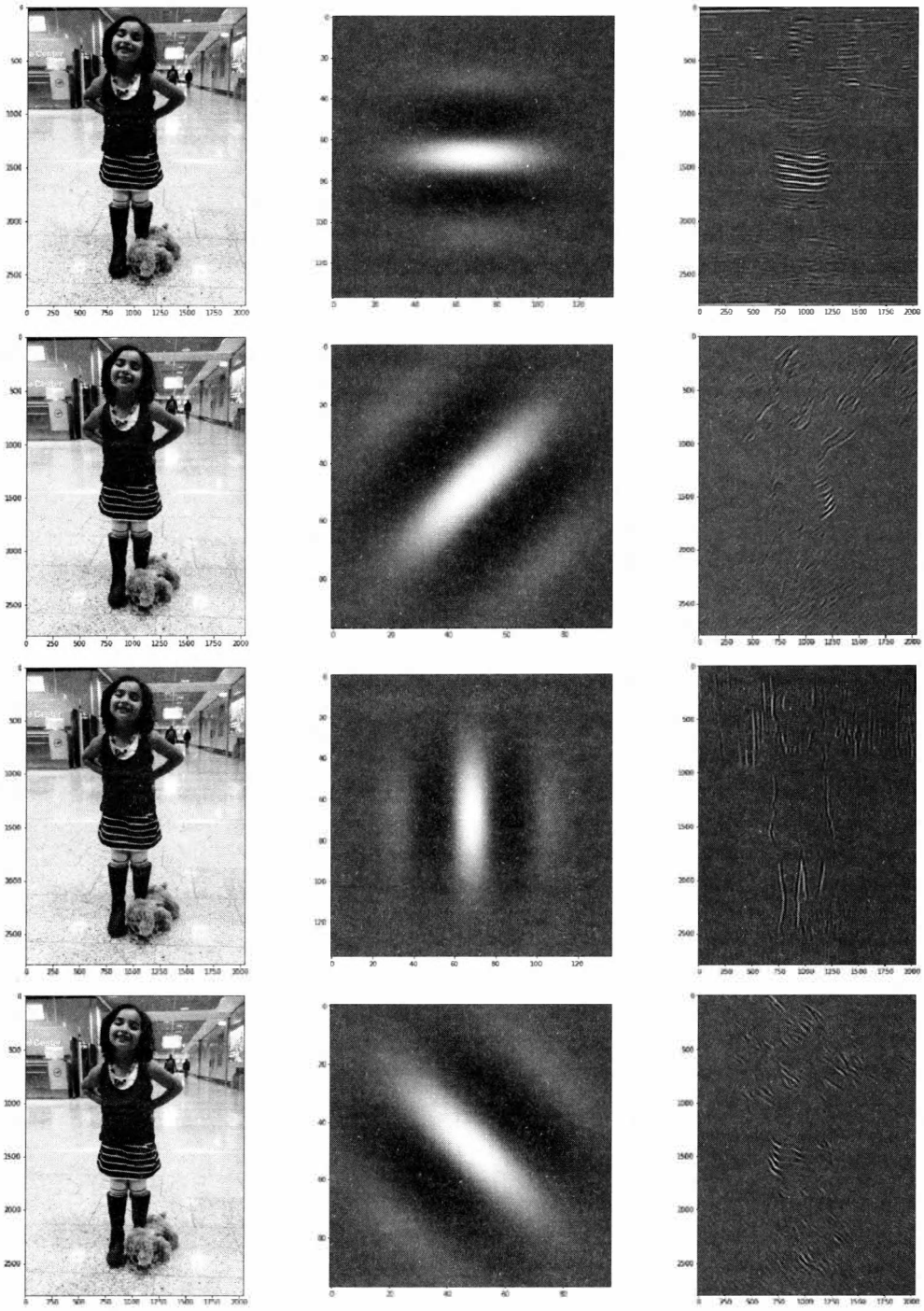


Рис. 5.2. Применение фильтров Габора к одному и тому же изображению; фильтры предназначены для выявления различных текстуры и ориентации изображения. Подробности изображений см. на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**)

Еще один замечательный пример приведен на рис. 5.2. На нем изображение справа — результат свертки изображения слева с фильтром (ядром), показанным в середине каждого ряда. Такие фильтры называются фильтрами Габора (**https://oreil.ly/LRVLi**). Они предназначены для выделения определенных паттернов, текстур, признаков изображения и работают по аналогии с фильтрами, функционирующими в зрительной системе человека.

Наши глаза замечают области изображения, в которых происходит изменение, т. е. создают контраст. Они выявляют ребра (горизонтальные, вертикальные, диагональные) и градиенты (оценивая крутизну изменения). В моделях применяются фильтры, способные делать то же самое математически, скользя с помощью свертки по всему сигналу. При отсутствии изменений в сигнале они выдают ровные гладкие результаты (нули или числа, близкие друг к другу по значению), а когда выявляют ребро или градиент, совпадающий с ядром, получаются пики.

Карта признаков

Сверточная нейросеть *обучает* ядра на основе данных, оптимизируя функцию потерь так же, как и полносвязная сеть. Неизвестные веса, входящие в формулу функции обучения, — это значения каждого ядра на каждом сверточном слое, смещения и веса, относящиеся к любому полносвязному слою в архитектуре сети. Выход сверточного слоя (содержащего нелинейную функцию активации) называется *картой признаков*, а обученные ядра — детекторами признаков. Можно часто наблюдать, что ранние слои нейросетей (близкие к входному слою) усваивают низкоуровневые признаки, например ребра, а поздние слои — более высокоуровневые признаки, такие как формы. Это вполне закономерно, поскольку на каждом новом слое мы составляем нелинейную функцию активации, поэтому по мере увеличения количества слоев сложность сети возрастает, а значит, повышается и ее способность отражать более сложные признаки.

Как составлять карты признаков.

Карты признаков позволяют вскрыть черный ящик и непосредственно наблюдать то, что выявляет обученная сеть на каждом сверточном слое. Когда сеть еще обучается, карты признаков позволяют точно определить источники ошибок и соответствующим образом скорректировать модель. Предположим, что на вход обученной сверточной нейросети подается изображение. На первом уровне ядро с помощью свертки проходит по всему изображению и выдает новое отфильтрованное изображение. Затем последнее пропускается через нелинейную функцию активации, и получается еще одно изображение. Наконец, оно проходит через пулинг-слой, о котором мы расскажем далее, и мы получаем окончательный результат работы сверточного слоя. Это изображение будет отличаться от изначального и, возможно, иметь другие размеры, но если сеть хорошо обучена, она выделит на выходном изображении важные признаки исходного изображения, такие как ребра, текстура и т. д. Выходное изображение представляет собой матрицу чисел или тензор чисел (трехмерный для цветных изображений или четырехмерный, если мы работаем с партиями изображений или с видео-

данными, где есть еще одно дополнительное измерение для временны́х рядов). Их можно легко представить в виде карт признаков, сославшись на библиотеку Matplotlib в Python, где каждая запись в матрице или тензоре сопоставляется с интенсивностью пиксела, находящегося в том же месте, что и запись матрицы. На рис. 5.3 показаны карты признаков на разных сверточных слоях сверточной нейросети.

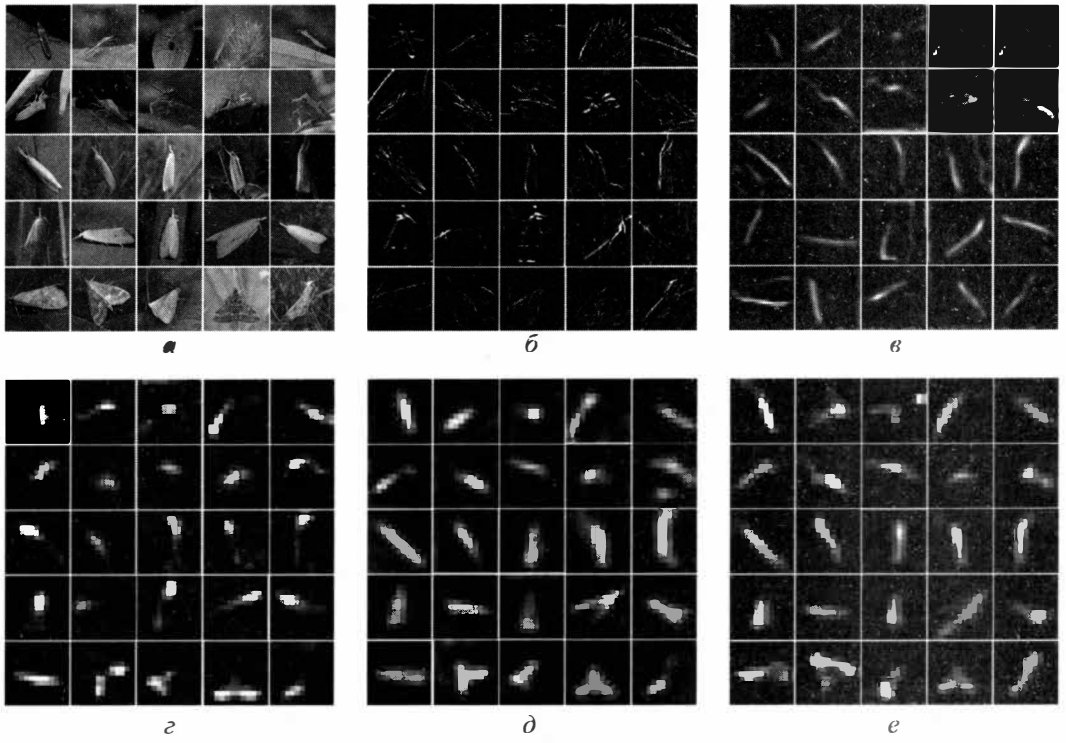


Рис. 5.3. Карты признаков на различных сверточных слоях сверточной нейросети (источник изображения: **https://oreil.ly/WvY1l**)

Нотация линейной алгебры

Теперь мы знаем, что свертка — это самая базовая операция в сверточной нейросети. При заданном фильтре k (он может быть одномерным, двумерным, трехмерным или иным многомерным) свертка обрабатывает входной сигнал, скользя по нему фильтром. Эта операция является линейной, т. е. каждый выходной сигнал представляет собой линейную комбинацию (по весам в k) входных компонентов, так что ее можно удобно записать в виде матричного умножения. Для нас это важно, поскольку мы все равно должны написать обучающую функцию, представляющую сверточную нейросеть, так, чтобы ее можно было легко вычислять и дифференцировать. Математическая структура задачи остается такой же, как и для всех моделей машинного обучения, с которыми мы уже познакомились в книге.

Обучающая функция.

В случае сверточных нейросетей это, как правило, линейные комбинации компонентов входного сигнала, которые складываются с функциями активации, затем с функцией пулинг-слоя (о которой речь пойдет ниже) на нескольких слоях разного размера и с разными связями, и завершаются в итоге логистической функцией, функцией машин опорных векторов или другими функциями в зависимости от конечной цели сети (классификация, сегментация изображения, генерация данных и т. д.).

В отличие от полносвязных нейросетей, линейная комбинация выполняется теперь с весами, относящимися к фильтру, т. е. они не все будут разными (если только вместо сверточных слоев мы не используем локально связанные слои).

Более того, размеры фильтров, как правило, на порядки меньше входных сигналов, поэтому если мы выразим операцию свертки в матричной нотации, большинство весов в матрице будут равны нулю. Напомним, что каждому входному признаку на каждом слое сети соответствует свой вес. Например, если на вход поступает цветное изображение, то каждый пиксел в каждом канале будет иметь свой вес. Но, применив сверточный слой, мы получим только несколько уникальных весов и много-много нулей. Это существенно упрощает требования к хранению, сокращает время вычислений и вместе с тем позволяет учесть важные локальные взаимодействия.

При этом в архитектуре сверточной нейросети часто встречаются полносвязные слои (с разным весом для каждой связи) в окрестности выходного слоя. Это можно рассматривать как дистилляцию признаков — после выявления важных и локально зависимых признаков, чья сложность возрастает с каждым слоем, они комбинируются для прогнозирования. То есть когда на полносвязный слой поступает информация, она уже дистиллирована до наиболее важных компонентов, которые, в свою очередь, выступают в качестве уникальных признаков, участвующих в итоговом предсказании.

Функция потерь.

Эта функция аналогична всем функциям потерь, рассмотренным в предыдущих главах, и всегда представляет собой оценку ошибки, которую вносят предсказания сети и истинные данные.

Оптимизация.

Здесь снова используется стохастический градиентный спуск, что связано с огромным размером задачи и количеством входящих в нее переменных. Как обычно, требуется оценить одну производную функции потерь, содержащую одну производную обучающей функции. Мы вычисляем производную по всем неизвестным весам, относящимся к фильтрам всех слоев и каналов сети, и связанных с ними смещений. В плане вычислений алгоритм обратного распространения ошибки по-прежнему выступает тяговой силой процесса дифференцирования.

Линейная алгебра и вычислительная линейная алгебра обладают всеми необходимыми инструментами для создания обучаемых сверточных нейросетей. В общем случае плотная матрица (в основном ненулевые записи) считается наихудшей в

плане вычислений, поскольку она не имеет в своих записях ни явной структуры, ни паттерна (еще хуже, если она — недиагонализируемая матрица и т. д.). Но в случае разреженной матрицы (в основном нули), или матрицы с определенным паттерном записи (диагональная, трехдиагональная, круговая и т. д.), или того и другого вместе, мы оказываемся в благоприятном вычислительном пространстве, если только знаем, как использовать особую структуру матрицы в своих интересах. Исследователей, изучающих вычисления и алгоритмы больших матриц, можно по праву считать элитой своей области, поскольку без их работы у нас была бы только голая теория, по масштабу трудно реализуемая на практике.

В одном измерении операцию свертки можно представить с помощью специальной матрицы, которая называется *матрицей Тёплица*, а в двух измерениях — с помощью другой специальной матрицы, которая называется *дважды блочно-циркулянтной матрицей*. Остановимся только на этих двух, но с тем, чтобы понять, что в целом матричная нотация является наилучшим способом представления, и было бы глупо не выявить и в максимальной степени не использовать свойственную нашим матрицам структуру. Другими словами, если сначала решать самые общие задачи, то это может оказаться пустой тратой времени, а в нашей жизни время — ценный товар. В качестве неплохого компромисса между самой общей и самой конкретной матрицей можно добавить к полученным результатам анализ сложности, например вычисление порядка метода ($O\left(n^3\right)$ или $O\left(n\log n\right)$ и т. д.), чтобы была очевидна разница между применением тех или иных методов.

За примером обратимся к электронной книге "Глубокое обучение" ("Deep Learning", с. 334 в главе 9) Яна Гудфеллоу и соавт. (**https://www.deeplearningbook.org/**), рассказывающей об эффективности свертки и множества нулей в матрице по сравнению с обычным матричным умножением для обнаружения вертикальных ребер на определенном изображении. Свертка — это чрезвычайно эффективный способ описания преобразований, которые применяют одно и то же линейное преобразование небольшой локальной области ко всему входному сигналу.

Изображение справа на рис. 5.4 было сформировано путем вычитания из каждого пиксела исходного изображения значения соседнего слева пиксела. Оно показывает силу всех вертикально ориентированных ребер на входном изображении, что может оказаться полезным при обнаружении объектов.

> "Оба изображения имеют высоту 280 пикселов. Ширина входного и выходного изображений составляет, соответственно, 320 и 319 пикселов. Это преобразование можно описать ядром свертки, содержащим два элемента, при этом для вычисления свертки потребуется $319\times 280\times 3 = 267\,960$ операций с плавающей запятой (два умножения и одна композиция на выходной пиксел).
>
> Для описания того же преобразования с помощью матричного умножения потребуется $320\times 280\times 319\times 280$, или более восьми миллиардов записей в матрице, благодаря чему свертка в четыре миллиарда раз эффективнее отображает это преобразование. Простой алгоритм матричного умножения выполняет более шестнадцати миллиардов операций с плавающей точкой, и поэтому свертка примерно в 60 000 раз эффективнее

с вычислительной точки зрения. Безусловно, большинство записей матрицы будут нулевыми. Если мы будем сохранять только ненулевые записи матрицы, то как для умножения, так и для свертки потребуется одинаковое количество операций с плавающей точкой. В матрице по-прежнему должно быть $2 \times 319 \times 280 = 178\ 640$ записей".

— Ян Гудфеллоу и др. "Глубокое обучение"

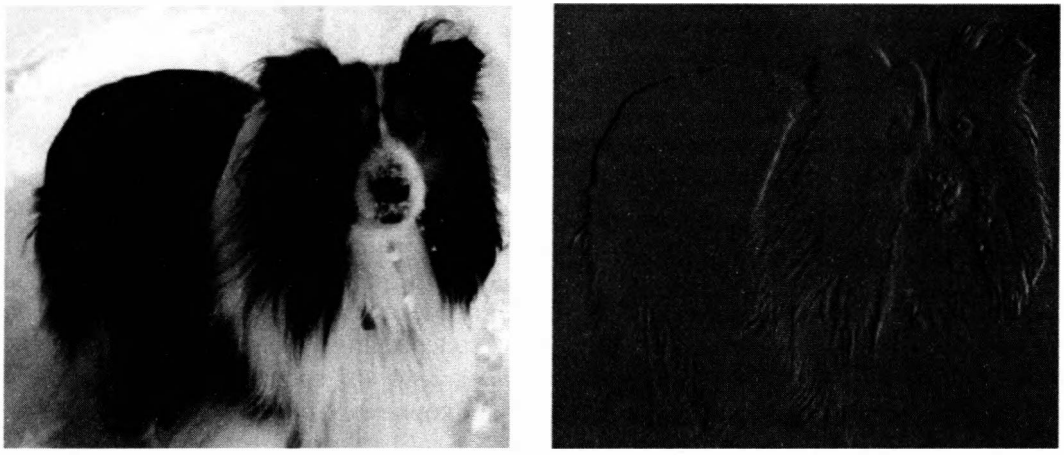


Рис. 5.4. Выявление вертикальных ребер на изображении (источник изображения: **https://www.deeplearningbook.org/**)

Одномерный случай: умножение на матрицу Тёплица

Ленточная матрица Тёплица имеет вид:

$$\mathbf{T} = \begin{pmatrix} k_0 & k_1 & k_2 & 0 & 0 & 0 & 0 \\ 0 & k_0 & k_1 & k_2 & 0 & 0 & 0 \\ 0 & 0 & k_0 & k_1 & k_2 & 0 & 0 \\ 0 & 0 & 0 & k_0 & k_1 & k_2 & 0 \\ 0 & 0 & 0 & 0 & k_0 & k_1 & k_2 \end{pmatrix}.$$

Умножая эту матрицу на одномерный сигнал $x = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$, мы получим точный результат свертки одномерного фильтра $k = (k_1, k_2, k_3)$ с сигналом x, а именно, матрицу $\mathbf{T}x^T = k \star x$. При умножении наблюдается *скользящее движение* фильтра вдоль сигнала.

Двумерный случай: умножение на дважды блочно-циркулянтную матрицу

Двумерный аналог состоит из двумерной свертки и фильтрации изображений. Теперь мы умножаем не на матрицу Тёплица, а на *дважды блочно-циркулянтную*

матрицу, каждая строка которой представляет собой циклический сдвиг заданного вектора. Запись такой матрицы и ее эквивалентности двумерной свертке — замечательное упражнение по линейной алгебре. В глубоком обучении мы по итогу изучаем веса, которые являются записями этих матриц. Используя нотацию линейной алгебры (в выражениях матрицы Тёплица или циркулянтной матрицы), можно найти компактные формулы для производных функции потерь по этим весам.

Пулинг

Одним из этапов, характерным практически для всех сверточных нейросетей, является *пулинг*, или объединение. Как правило, его выполняют после фильтрации входных данных с помощью свертки, а затем пропускают через нелинейную функцию активации. Различают несколько видов пулинга, хотя все они выполняют одну и ту же задачу — они заменяют выход в определенном месте на суммарную статистику ближайших выходов. В качестве примера в случае изображений можно привести замену четырех пикселов одним, содержащим максимальное значение четырех изначальных (*максимальный пулинг*), или их средним значением, или средневзвешенным значением, или квадратным корнем суммы их квадратов и т. д.

На рис. 5.5 показано, как действует максимальный пулинг.

Рис. 5.5. Максимальный пулинг (источник изображения: **https://oreil.ly/y9x9P**)

По сути, он уменьшает размерность и обобщает целые окрестности выходов, жертвуя при этом мелкими деталями. Так что пулинг плохо подойдет в случаях прогнозирования, когда важны мелкие детали. Тем не менее он обладает множеством преимуществ:

- обеспечивает приблизительную инвариантность (неизменность) к малым пространственным преобразованиям входного сигнала. Это пригодится в тех случаях, когда наличие какого-либо признака важнее его точного расположения;
- существенно повышает статистическую эффективность сети;
- повышает вычислительную эффективность и смягчает требования к памяти сети за счет уменьшения количества входов на следующий слой;
- позволяет работать со входами разного размера, контролируя размер объединенных в пулинг окрестностей и размер выхода после объединения.

Сверточная нейросеть для классификации изображений

Если мы возьмемся перечислять всевозможные архитектуры и вариации нейросетей, то рискуем отвлечься от основной цели книги — понимания математики, лежащей в основе различных моделей. Но мы можем рассмотреть существенные компоненты и то, как они объединяются для решения задач ИИ, на примере классификации изображений для компьютерного зрения.

В процессе обучения мы по-прежнему выполняем описанные в *главе 4* шаги:

1. Инициализируем случайные веса (согласно процедурам инициализации, описанным в *главе 4*).
2. Пропускаем пакет изображений через сверточную сеть и выводим класс изображения.
3. Оцениваем функцию потерь для этой конкретной выборки весов.
4. Выполняем обратное распространение ошибки по сети.
5. Подгоняем веса, способствующие возникновению ошибки (стохастический градиентный спуск).
6. Повторяем определенное количество итераций или вплоть до достижения сходимости.

К счастью, нам не придется самим выполнять все эти действия. В библиотеке Keras для Python содержится большое количество предобученных моделей, т. е. их веса уже определены, и нам останется только оценить обученную модель на своем наборе данных.

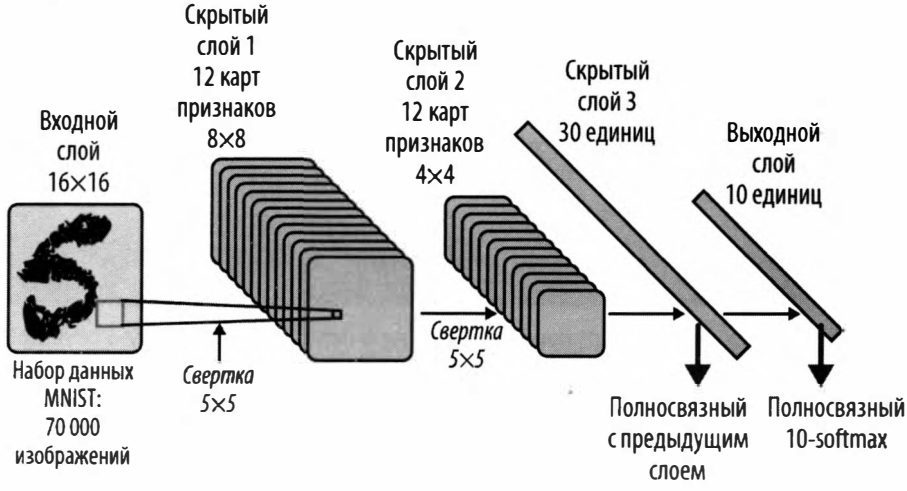


Рис. 5.6. Архитектура LeNet-1 (1989)

Единственное, что мы можем и должны делать, — наблюдать и изучать архитектуру успешных и победоносных сетей. На рис. 5.6 представлена простая архитектура

сети LeNet-1 (**https://oreil.ly/kKgqL**), предложенная ЛеКуном и соавт. (1989), а на рис. 5.7 — архитектура сети AlexNet (2012).

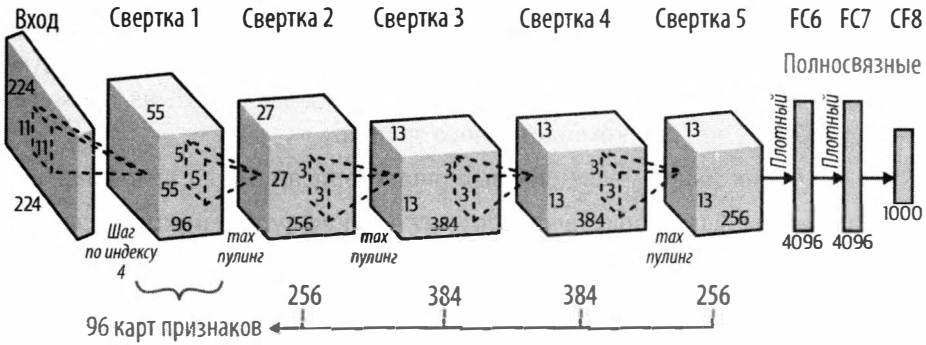


Рис. 5.7. Архитектура AlexNet (2012) с огромным количеством весов (62,3 млн). Адаптировано из: **https://oreil.ly/eEWgJ**

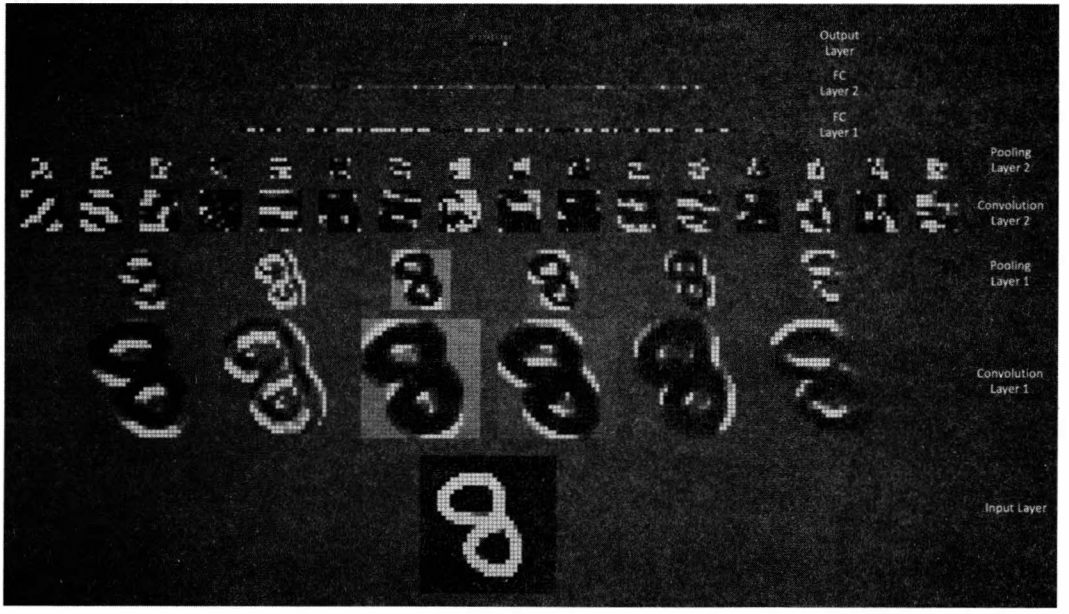


Рис. 5.8. Прогон изображения рукописной цифры 8 через предобученную сеть LeNet-1 (источник изображения: **https://oreil.ly/6vgOc**)

В качестве хорошего упражнения можно попробовать подсчитать количество весов, входящих в обучающую функцию сетей LeNet-1 и AlexNet. Важно отметить: чем больше единиц на каждом слое (картах признаков), тем больше весов. Когда я попыталась подсчитать веса в LeNet-1 с архитектурой, представленной на рис. 5.6, у меня получилось 9484 весов, хотя в опубликованной статье говорится о 9760 весах, так что не понимаю, куда подевались остальные веса. Если вы найдете их, пожалуйста, сообщите мне. В любом случае суть в том, что требуется решить задачу

оптимизации в $\mathbb{R}^{9760}$. Выполним те же вычисления для AlexNet, показанной на рис. 5.7. В этом случае имеется 62,3 млн весов, так что возникает задача оптимизации в $\mathbb{R}^{62,3\text{ млн}}$. Еще одно потрясающее число — для одного прямого прохода по сети потребуется 1,1 млрд вычислительных единиц.

На рис. 5.8 приведена замечательная иллюстрация того, как изображение рукописной цифры 8 проходит через предобученную сеть LeNet-1 и в конечном итоге правильно классифицируется как 8.

И в завершение, если вам кажется, что архитектура была выбрана произвольно, т. е. если у вас возникает вопрос "Можно ли достичь аналогичной производительности с помощью более простой архитектуры?", — добро пожаловать в клуб! Все сообщество задается тем же самым вопросом.

Итоги и перспективы

В этой главе мы говорили о свертке — важнейшем компоненте сверточных нейросетей. Сверточные нейросети применяются в компьютерном зрении, машинной обработке звука и других приложениях искусственного интеллекта.

Мы рассмотрели свертку с точки зрения проектирования систем, а затем на примере фильтрации одномерных и двумерных сигналов. Также мы разобрали эквивалент операции свертки в линейной алгебре (умножение на матрицы со специальной структурой) и завершили главу примером классификации изображений.

В книге нам будут часто встречаться сверточные нейросети, поскольку они стали опорой множества систем искусственного интеллекта, в том числе компьютерного зрения и естественного языка.

ГЛАВА 6

Сингулярное разложение: обработка изображений, обработка естественного языка и социальные сети

Покажите мне главное, только главное.
— Хала

Сингулярное разложение — это математическая операция из линейной алгебры, которая широко применяется в области науки о данных, машинного обучения и искусственного интеллекта. Именно она лежит в основе анализа главных компонент (в анализе данных) и латентно-семантического анализа (в обработке естественного языка). Эта операция преобразует плотную матрицу в диагональную. В линейной алгебре диагональные матрицы считаются особенными и крайне желательными. При умножении на них они ведут себя как скалярные числа, но только растягиваются или сжимаются в определенных направлениях.

При вычислении сингулярного разложения матрицы мы получаем дополнительный бонус — выявление и количественную оценку действия матрицы на само пространство: поворот, отражение, растяжение и/или сжатие. При этом не происходит искривления (изгиба) пространства, поскольку эта операция является линейной (в конце концов, она называется линейной алгеброй). Экстремальное растяжение или сжатие в одном направлении по сравнению с другими влияет на устойчивость любых вычислений с участием матрицы, поэтому наличие такой меры позволяет напрямую регулировать чувствительность вычислений к различным возмущениям, например к шумовым измерениям.

Сила сингулярного разложения заключается в том, что его можно применить к любой матрице. Это обстоятельство, а также обширное применение в области ИИ заслуживает отдельной главы в книге. В следующих разделах мы в общих чертах, не углубляясь в детали, рассмотрим сингулярное разложение, а также его применение в обработке изображений, обработке естественного языка и в социальных сетях.

В случае заданной матрицы **C** (изображение, матрица данных и т. д.) подробности вычисления ее сингулярного разложения опускаются. В большинстве учебников по линейной алгебре так делается при изложении теоретического метода, основанного

на вычислении собственных векторов и собственных значений симметричных матриц $\mathbf{C}^T\mathbf{C}$ и $\mathbf{C}\mathbf{C}^T$, которые для нас являются ковариационными матрицами данных (если данные центрированы). Хотя понимание теории по-прежнему не теряет своей значимости, предлагаемый ею метод вычисления сингулярного разложения малопригоден с точки зрения эффективности вычислений и особенно невозможен в случае больших матриц, которые встречаются в львиной доле реалистичных задач. К счастью, мы живем в такое время, когда подобные вычисления легко выполняются с помощью программных пакетов. В качестве примера достаточно привести метод `numpy.linalg.svd` из *библиотеки NumPy* для Python. Далее в этой главе мы кратко рассмотрим численные алгоритмы, заложенные в эти программные пакеты. Причем основное внимание мы уделим пониманию того, как работает сингулярное разложение и почему оно необходимо для сокращения объема памяти и снижения вычислительных требований к задаче без потери важной информации. Мы также разберем его роль в кластеризации данных.

Факторизация матриц

Скалярное число можно факторизовать несколькими способами, например записать число $12 = 4 \times 3$, $12 = 2 \times 2 \times 3$ или $12 = 0{,}5 \times 24$. Какой из них лучше, зависит от конкретного случая. То же самое применимо и к числовым матрицам. В линейной алгебре существует множество разных способов факторизации матриц. Идея заключается в том, чтобы разложить объект на более мелкие составляющие, и тогда они дадут нам представление о функции и действии самого объекта. Такое разбиение также позволит нам узнать, какие компоненты содержат наибольшее количество информации и, следовательно, представляют наибольшую ценность на фоне остальных. В этом случае целесообразно отбросить менее важные компоненты и создать объект меньшего размера с аналогичной функцией. Меньший объект может быть не столь подробным, как изначальный, содержащий все компоненты; тем не менее он несет в себе довольно значимую информацию из начального объекта, так что его применение при меньшем размере дает свои преимущества. Такое сингулярное разложение называется факторизацией матриц. Ее формула имеет вид

$$\mathbf{C}_{m \times n} = \mathbf{U}_{m \times n} \Sigma_{m \times n} \mathbf{V}^T_{m \times n},$$

где матрица $\mathbf{C}$ раскладывается на три составляющие матрицы: $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^T$. $\mathbf{U}$ и $\mathbf{V}$ — квадратные матрицы с ортогональными строками и столбцами. $\mathbf{\Sigma}$ — диагональная матрица, имеющая ту же форму, что и $\mathbf{C}$ (см. рис. 6.1).

Начнем с умножения матриц. Пусть $\mathbf{A}$ — матрица с 3 строками и 3 столбцами:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}_{3 \times 3},$$

а **B** — матрица с 3 строками и 2 столбцами:

$$\mathbf{B} = \begin{pmatrix} 1 & 3 \\ 4 & -2 \\ 0 & 1 \end{pmatrix}_{3\times 2}.$$

Тогда $\mathbf{C} = \mathbf{AB}$ — матрица с 3 строками и 2 столбцами:

$$\mathbf{C}_{3\times 2} = \mathbf{A}_{3\times 3}\mathbf{B}_{3\times 2} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}_{3\times 3} \begin{pmatrix} 1 & 3 \\ 4 & -2 \\ 0 & 1 \end{pmatrix}_{3\times 2} = \begin{pmatrix} 9 & 2 \\ 24 & 8 \\ 39 & 14 \end{pmatrix}_{3\times 2}.$$

Можно считать, что **C** факторизуется произведением **A** и **B** подобно тому, как число $12 = 4\times 3$. Предыдущая факторизация **C** не имеет значения, поскольку ни **A**, ни **B** не являются матрицами специального типа. Весьма ценным в факторизации **C** является ее сингулярное разложение. Любая матрица имеет сингулярное разложение. Вычислим его на языке Python (код см. в соответствующем блокноте Jupyter):

$$\begin{aligned}\mathbf{C}_{3\times 2} &= \mathbf{U}_{3\times 3}\mathbf{\Sigma}_{3\times 2}\mathbf{V}^T_{2\times 2} = \\ &= \begin{pmatrix} -0{,}1853757 & 0{,}8938507 & 0{,}4082482 \\ -0{,}5120459 & 0{,}2667251 & -0{,}8164965 \\ -0{,}8387161 & -0{,}3604005 & 0{,}4082482 \end{pmatrix} \times \\ &\times \begin{pmatrix} 49{,}402266 & 0 \\ 0 & 1{,}189980 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0{,}9446411 & -0{,}3281052 \\ 0{,}3281052 & -0{,}9446411 \end{pmatrix}.\end{aligned}$$

В этом разложении нужно обратить внимание на следующее: строки $\mathbf{V}^T$ — это *правые сингулярные векторы* (сами столбцы **V**), столбцы **U** — *левые сингулярные векторы*, а диагональные записи $\mathbf{\Sigma}$ — *сингулярные значения*. Сингулярные значения всегда положительны и всегда расположены в порядке убывания вдоль диагонали $\mathbf{\Sigma}$. Отношение наибольшего сингулярного значения к наименьшему называется *числом обусловленности* к матрицы. В нашем случае всего два сингулярных значения, и $k = \dfrac{49{,}402266}{1{,}189980} = 41{,}515207$. Это число играет важную роль в вычислительной устойчивости с участием матрицы. Хорошо обусловленными считаются матрицы, имеющие небольшие числа обусловленности.

Левые сингулярные векторы ортогональны (ортогональны друг другу и имеют длину 1). Точно так же ортогональны и правые сингулярные векторы.

Качественные свойства изображения оцениваются быстрее, чем бесконечные массивы чисел. С помощью Python можно легко представить матрицы в виде изображений (и наоборот, изображения хранятся в виде матриц чисел): значение записи матрицы соответствует интенсивности соответствующего пиксела. Чем больше число, тем ярче пиксел. Меньшие числа в матрице отображаются как более темные

пикселы, а большие — как более яркие. На рис. 6.1 показано рассмотренное выше сингулярное разложение. Можно заметить, что диагональная матрица $\mathbf{\Sigma}$ имеет такую же форму, как $\mathbf{C}$, а ее диагональные записи расположены в порядке убывания, причем самый яркий пиксел, соответствующий наибольшему *сингулярному значению*, находится в левом верхнем углу.

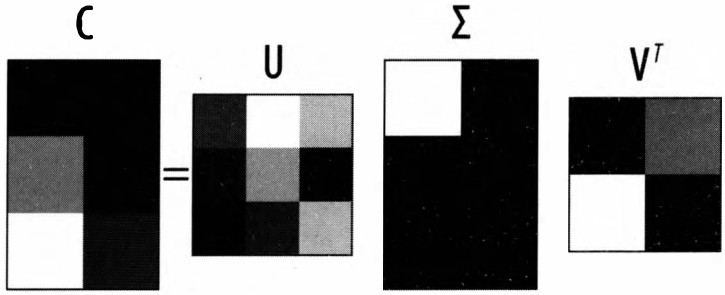


Рис. 6.1. Визуализация сингулярного разложения

На рис. 6.2 и 6.3 показаны сингулярные разложения двух прямоугольных матриц $\mathbf{A}$ и $\mathbf{B}$, где $\mathbf{A}$ — широкая, а $\mathbf{B}$ — высокая матрицы:

$$\mathbf{A} = \begin{pmatrix} -1 & 3 & -5 & 4 & 18 \\ 1 & -2 & 4 & 0 & -7 \\ 2 & 0 & 4 & -3 & -8 \end{pmatrix}_{3\times 5} = \mathbf{U}_{3\times 3}\mathbf{\Sigma}_{3\times 5}\mathbf{V}^T_{5\times 5};$$

$$\mathbf{B} = \begin{pmatrix} 5 & 4 \\ 4 & 0 \\ 7 & 10 \\ -1 & 8 \end{pmatrix}_{4\times 2} = \mathbf{U}_{4\times 4}\mathbf{\Sigma}_{4\times 2}\mathbf{V}^T_{2\times 2}.$$

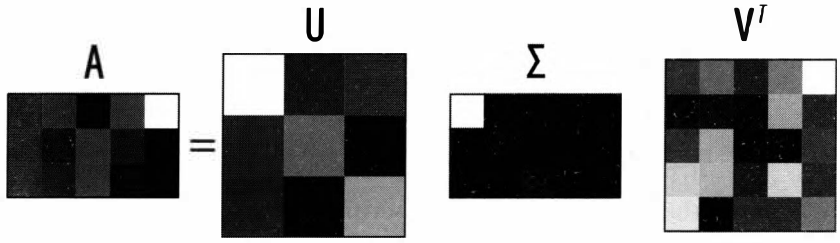


Рис. 6.2. Визуализация сингулярного разложения широкой прямоугольной матрицы. Последние два столбца $\mathbf{\Sigma}$ — сплошные нули (черные пикселы), что позволяет сократить объем памяти, удалив два последних столбца $\mathbf{\Sigma}$ вместе с двумя последними строками $\mathbf{V}^T$

На рис. 6.2 мы видим, что два последних столбца $\mathbf{\Sigma}$ заполнены одними нулями (черные пикселы), следовательно, в целях экономии можно удалить эти два столбца вместе с двумя последними строками $\mathbf{V}^T$ (умножение на диагональную матрицу слева *см. в следующем разделе*). Аналогично на рис. 6.3 две последние строки $\mathbf{\Sigma}$ —

сплошные нули (черные пикселы), а значит, в целях экономии можно избавиться от этих двух строк вместе с двумя последними столбцами **U** (умножение на диагональную матрицу справа *см. в следующем разделе*). Благодаря сингулярному разложению мы уже сэкономили место (важно отметить, что обычно хранятся только диагональные записи **Σ**, а не вся матрица с полным набором нулей).

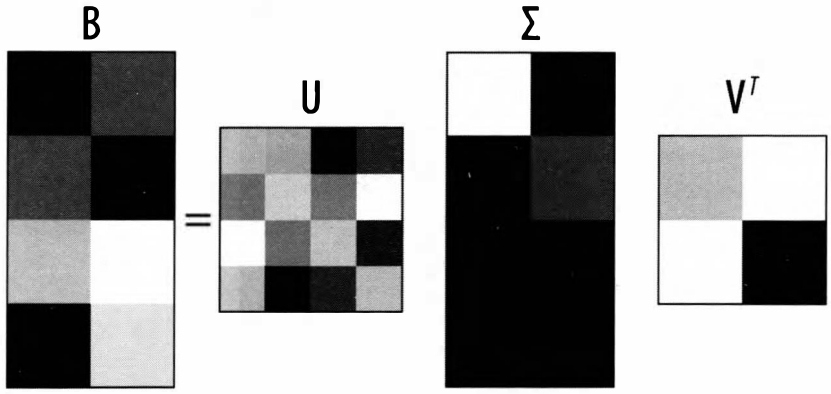


Рис. 6.3. Визуализация сингулярного разложения высокой прямоугольной матрицы. Последние две строки **Σ** — сплошные нули (черные пикселы), что позволяет сократить объем памяти, удалив две последние строки **Σ** вместе с двумя последними столбцами **U**

Диагональные матрицы

При умножении вектора на скалярное число, например на 3, получается новый вектор в том же направлении с той же ориентацией, но его длина в три раза увеличивается. При умножении того же вектора на другое скалярное число, например –0,5, получается другой вектор, опять в том же направлении, но на этот раз его длина будет вдвое меньше, а ориентация изменится на противоположную. Умножение на скалярное число — простейшая операция, и хорошо бы иметь такие матрицы, которые так же просто применялись бы к векторам (другими словами, умножались на них). Будь наша жизнь одномерной, мы имели бы дело только со скалярными числами, но поскольку наша жизнь, впрочем, как и интересующие нас приложения, имеют более высокую размерность, нам предстоит работать с диагональными матрицами (рис. 6.4). И это неплохая работа.

Умножение на диагональную матрицу соответствует растяжению или сжатию в определенных направлениях в пространстве, а на смену ориентации указывает отрицательное число на диагонали. Как известно, большинство матриц не являются диагональными. Сила сингулярного разложения заключается в том, что оно позволяет определить направления в пространстве, по которым матрица *ведет себя как* (пусть и в широком смысле) диагональная матрица. Как правило, диагональная матрица растягивается/сжимается в тех же направлениях, что и координаты вектора. С другой стороны, если матрица не растягивается/сжимается в тех же направле-

ниях, что и координаты, то она не считается диагональной. *После* изменения координат она будет растягиваться/сжиматься в других направлениях. С помощью сингулярного разложения можно добиться требуемого изменения координат (*правые сингулярные векторы*), по которым будут растягиваться/сжиматься векторы (*левые сингулярные векторы*), а также получить величину растяжения/сжатия (*сингулярные значения*). Подробнее об этом мы расскажем в следующем разделе, но сначала разберем умножение на диагональные матрицы слева и справа.

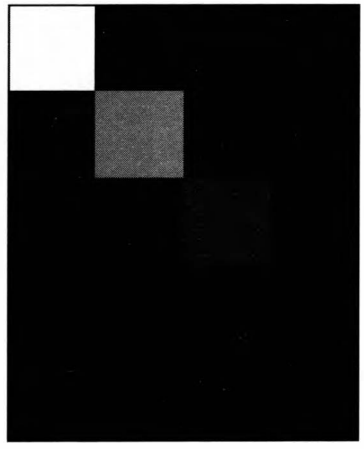


Рис. 6.4. Изображение диагональной матрицы 5 × 4 с диагональными записями: 10 (самый яркий пиксел), 6, 3, 1 (самый темный пиксел, кроме нулей)

При умножении матрицы **A** на диагональную матрицу **Σ** справа, **AΣ**, мы масштабируем столбцы **A**, например, на σ:

$$\mathbf{A\Sigma} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} \sigma_1 a_{11} & \sigma_2 a_{12} \\ \sigma_1 a_{21} & \sigma_2 a_{22} \\ \sigma_1 a_{31} & \sigma_2 a_{32} \end{pmatrix}.$$

При умножении **A** на **Σ** слева, **ΣA**, мы масштабируем строки **A**, например, на σ:

$$\mathbf{\Sigma A} = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} = \begin{pmatrix} \sigma_1 a_{11} & \sigma_1 a_{12} \\ \sigma_2 a_{21} & \sigma_2 a_{22} \\ \sigma_3 a_{31} & \sigma_3 a_{32} \end{pmatrix}.$$

Матрицы как линейные преобразования, действующие в пространстве

Один из способов представления матриц — в виде линейных преобразований (без искривления), которые действуют как на векторы в пространстве, так и на само пространство. Если искривление не допускается, поскольку оно превращает опера-

цию в нелинейную, то какие действия допустимы? К ним относятся поворот, отражение, растяжение и/или сжатие — операции, которые не вызывают искривления. Данную концепцию отражает сингулярное разложение $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. В случае, когда $\mathbf{A}$ действует на вектор $\vec{v}$, рассмотрим пошагово умножение $\mathbf{A}\vec{v} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\vec{v}$:

1. Сначала $\vec{v}$ поворачивается/отражается благодаря ортогональной матрице $\mathbf{V}^T$.
2. Затем из-за диагональной матрицы $\mathbf{\Sigma}$ она растягивается/сжимается по особым направлениям.
3. Наконец, она снова поворачивается/отражается из-за другой ортогональной матрицы $\mathbf{U}$.

Отражения и повороты не меняют пространство, поскольку сохраняют размеры и симметрии (можно представить, как мы поворачиваем предмет или смотрим на его отражение в зеркале). Степень растяжения и/или сжатия, закодированная в диагональной матрице $\mathbf{\Sigma}$ (через ее сингулярные значения по диагонали), является весьма информативной в отношении воздействия $\mathbf{A}$.

Ортогональная матрица

Ортогональная матрица имеет ортогональные строки и ортогональные столбцы. Она никогда не растягивает и не сжимает, а только поворачивает и/или отражает, т. е. при воздействии на объекты изменяет не их размер и форму, а только направление и/или ориентацию. Как и многое в математике, эти названия вносят путаницу. Матрица называется *ортогональной*, хотя ее строки и столбцы являются *ортонормальными*, что означает "ортогональными *и* равными единице". Еще один полезный факт: если $\mathbf{C}$ — ортогональная матрица, то $\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{I}$, т. е. обратная этой матрице — ее транспонирование. Как правило, вычисление обратной матрицы — весьма дорогостоящая операция, но в случае ортогональных матриц будет достаточно поменять строки на столбцы.

Проиллюстрируем эти понятия с помощью двумерных матриц, которые легко визуализировать. В следующих подразделах мы рассмотрим:

- воздействие матрицы $\mathbf{A}$ на правые сингулярные векторы, которые являются столбцами $\vec{v}_1$ и $\vec{v}_2$ матрицы $\mathbf{V}$. Они переводятся в кратные левые сингулярные векторы $\vec{u}_1$ и $\vec{u}_2$, являющиеся столбцами $\mathbf{U}$;
- воздействие $\mathbf{A}$ на стандартные единичные вектора $\vec{e}_1$ и $\vec{e}_2$. Также отметим, что единичный квадрат преобразуется в параллелограмм;
- воздействие $\mathbf{A}$ на общий вектор $\vec{x}$. Это позволит понять матрицы $\mathbf{U}$ и $\mathbf{V}$ как повороты или отражения в пространстве;
- воздействие $\mathbf{A}$ на единичную окружность. Мы видим, что $\mathbf{A}$ преобразует единичную окружность в эллипс, главные оси которого проходят вдоль левых сингулярных векторов ($\vec{u}$), а длины главных осей — сингулярные значения (σ). По-

скольку сингулярные значения упорядочены от наибольшего к наименьшему, то $\vec{u}_1$ определяет направление с наибольшим отклонением, а $\vec{u}_2$ — направление со вторым по величине отклонением и т. д.

Воздействие A на правые сингулярные векторы

Пусть **A** — матрица размера 2×2:

$$\mathbf{A} = \begin{pmatrix} 1 & 5 \\ -1 & 2 \end{pmatrix}.$$

Ее сингулярное разложение $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ имеет вид:

$$\mathbf{A} = \begin{pmatrix} 0{,}93788501 & 0{,}34694625 \\ 0{,}34694625 & -0{,}93788501 \end{pmatrix} \times$$
$$\times \begin{pmatrix} 5{,}41565478 & 0 \\ 0 & 1{,}29254915 \end{pmatrix} \begin{pmatrix} 0{,}10911677 & 0{,}99402894 \\ 0{,}99402894 & -0{,}10911677 \end{pmatrix}.$$

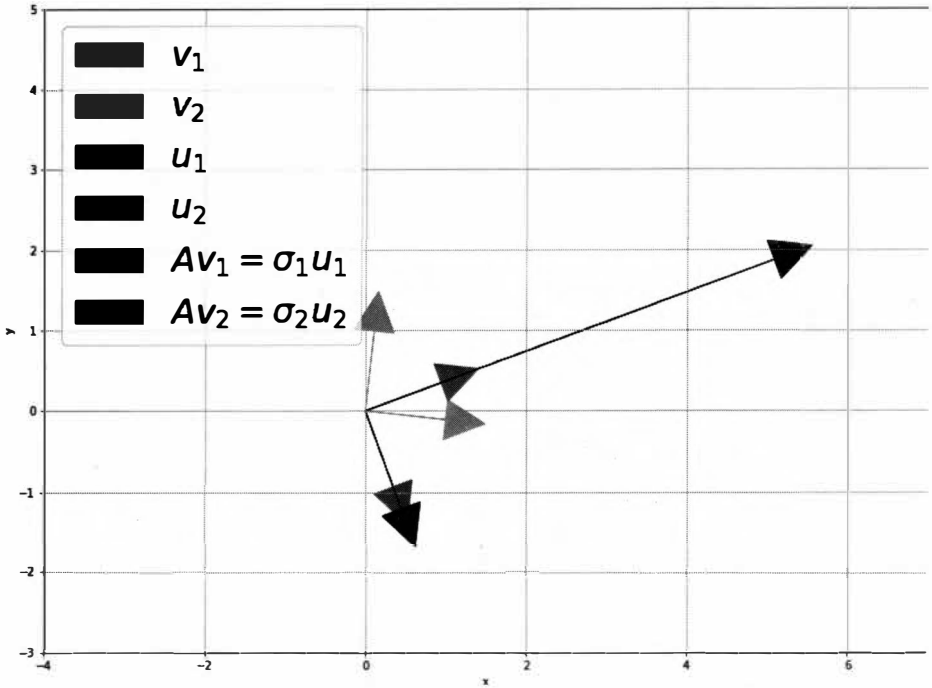


Рис. 6.5. Матрица **A** переводит правые сингулярные векторы в кратные левые сингулярные векторы: $\mathbf{A}v_1 = \sigma_1 u_1$ и $\mathbf{A}v_2 = \sigma_2 u_2$

Выражение $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ эквивалентно выражению:

$$\mathbf{AB} = \mathbf{U}\Sigma,$$

т. к. достаточно умножить $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ на $\mathbf{V}$ справа и воспользоваться тем, что $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ в силу ортогональности $\mathbf{V}$.

Можно представить $\mathbf{AV}$ как матрицу $\mathbf{A}$, воздействующую на каждый столбец матрицы $\mathbf{V}$. Поскольку $\mathbf{AV} = \mathbf{U}\Sigma$, то воздействие $\mathbf{A}$ на ортонормальные столбцы $\mathbf{V}$ равносильно растяжению/сжатию столбцов $\mathbf{U}$ на сингулярные значения. То есть:

$$\mathbf{A}\vec{v}_1 = \sigma_1\vec{u}_1$$

и

$$\mathbf{A}\vec{v}_2 = \sigma_2\vec{u}_2 .$$

Это изображено на рис. 6.5.

Воздействие A на стандартные единичные векторы и определяемый ими единичный квадрат

Матрица $\mathbf{A}$ направляет стандартные единичные векторы в свои столбцы и преобразует единичный квадрат в параллелограмм. При этом искривления (изгиба) пространства не происходит. На рис. 6.6 показано такое преобразование.

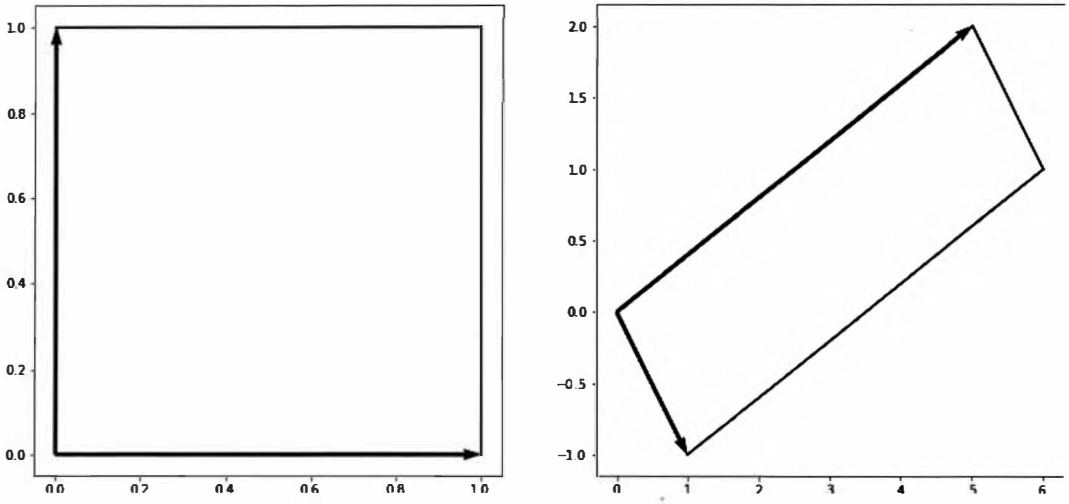


Рис. 6.6. Преобразование стандартных единичных векторов

Воздействие A на единичную окружность

На рис. 6.7 показано, как матрица $\mathbf{A}$ преобразует единичную окружность в эллипс. Главные оси проходят по осям u, а длины главных осей равны осям σ. Опять же,

поскольку матрицы представляют собой линейные преобразования, происходит не искривление, а поворот/отражение и растяжение/сжатие пространства.

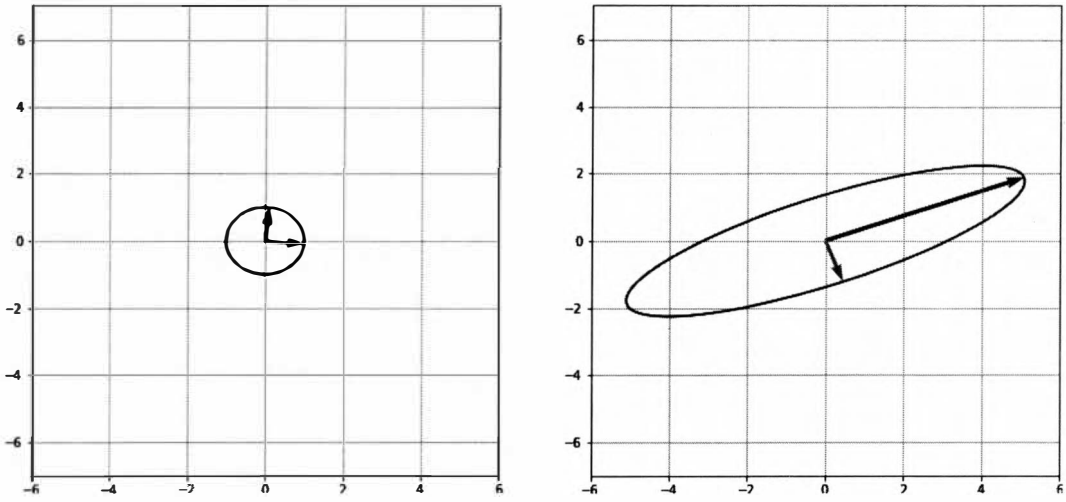


Рис. 6.7. Матрица **A** переводит единичную окружность в эллипс с главными осями вдоль левых сингулярных векторов и длинами главных осей, равными сингулярным значениям

Описанное выше воздействие наблюдается в сингулярном разложении.

Можно геометрически проиллюстрировать превращение окружности в эллипс в виде полярного разложения:

$$\mathbf{A} = \mathbf{QS}.$$

Преобразование окружности в эллипс методом сингулярного разложения

На рис. 6.8 показаны четыре подграфа, иллюстрирующие этапы приведенного выше преобразования окружности в эллипс:

1. Сначала умножаем единичную окружность и векторы $\vec{v}_1$ и $\vec{v}_2$ на $\mathbf{V}^T$. Так как $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, получаем $\mathbf{V}^T\vec{v}_1 = \vec{e}_1$ и $\mathbf{V}^T\vec{v}_2 = \vec{e}_2$. Таким образом, правые сингулярные векторы сначала *выпрямляются*, выравниваясь со стандартными единичными векторами.
2. Затем умножаем на **Σ**. При этом происходит только растяжение/сжатие стандартных единичных векторов на σ_1 и σ_2 (растяжение или сжатие зависит от того, больше или меньше единицы величина сингулярного значения).
3. В итоге умножаем на **U**. Это позволяет либо отразить эллипс вдоль линии, либо повернуть его на определенную величину по часовой стрелке или против нее. Об этом мы подробно поговорим в следующем подразделе.

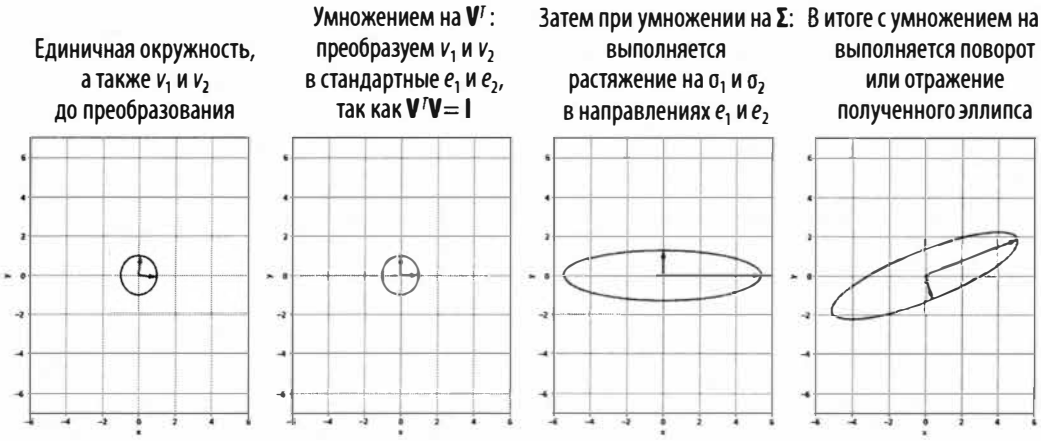


Рис. 6.8. Этапы преобразования единичной окружности в эллипс методом сингулярного разложения

Матрицы поворота и отражения

Матрицы $\mathbf{U}$ и $\mathbf{V}^T$, входящие в сингулярное разложение $\mathbf{A}=\mathbf{U}\Sigma\mathbf{V}^T$, являются ортогональными матрицами. Их строки и столбцы ортогональны, а их обратная величина равна их транспонированию. В двумерном пространстве они могут быть матрицами вращения или отражения (относительно прямой).

Матрица поворота

Матрица, поворачивающаяся по часовой стрелке на угол θ, имеет вид:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Транспонирование матрицы поворота — это поворот в противоположном направлении. Так, если матрица поворачивает по часовой стрелке на угол θ, то ее транспонирование поворачивается против часовой стрелки на угол θ и имеет вид:

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Матрица отражения

Матрица отражения относительно прямой L, составляющей угол θ с осью x, имеет вид:

$$\begin{pmatrix} \cos(2\theta) & \sin(2\theta) \\ \sin(2\theta) & -\cos(2\theta) \end{pmatrix}.$$

Наклон прямой L равен $\operatorname{tg}\theta$, она проходит через начало координат, значит, ее уравнение $y=(\operatorname{tg}\theta)x$. В операции отражения эта прямая действует как зеркало. На

рис. 6.9 показаны две прямые, относительно которых отражаются матрицы $\mathbf{V}^T$ и $\mathbf{U}$, а также вектор $\vec{x}$ и его последующее преобразование.

Определитель матрицы поворота равен 1, а определитель матрицы отражения равен –1.

В более высоких размерностях матрицы поворота и отражения выглядят иначе. Каждый раз нам нужно четко понимать, с каким объектом мы имеем дело. Если речь идет о повороте в трехмерном пространстве, то вокруг какой оси? Если отражение, то относительно какой плоскости? Если у вас возникнет желание узнать об этом подробнее, то сейчас самое время почитать об ортогональных матрицах и их свойствах.

Воздействие A на общий вектор $\vec{x}$

Мы разобрали воздействие $\mathbf{A}$ на правые сингулярные векторы (они преобразовываются в левые сингулярные векторы), стандартные единичные векторы (преобразовываются в столбцы $\mathbf{A}$), единичный квадрат (преобразовывается в параллелограмм) и единичную окружность (преобразовывается в эллипс с главными осями вдоль левых сингулярных векторов, длины которых равны сингулярным значениям). Наконец, мы разберем воздействие $\mathbf{A}$ на общий неособый вектор $\vec{x}$. Он преобразовывается в другой неособый вектор $\mathbf{A}\vec{x}$. Впрочем, разложение этого преобразования на этапы методом сингулярного разложения является информативным.

Напомним матрицу $\mathbf{A}$ и ее сингулярное разложение:

$$\mathbf{A} = \begin{pmatrix} 1 & 5 \\ -1 & 2 \end{pmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T =$$

$$= \begin{pmatrix} 0{,}93788501 & 0{,}34694625 \\ 0{,}34694625 & -0{,}93788501 \end{pmatrix} \times$$

$$\times \begin{pmatrix} 5{,}41565478 & 0 \\ 0 & 1{,}29254915 \end{pmatrix} \begin{pmatrix} 0{,}10911677 & 0{,}99402894 \\ 0{,}99402894 & -0{,}10911677 \end{pmatrix}.$$

Как $\mathbf{U}$, так и $\mathbf{V}^T$ в этом сингулярном разложении являются матрицами отражения. На рис. 6.9 показаны прямые $L_{\mathbf{U}}$ и $L_{\mathbf{V}^T}$, служащие зеркалами для этих отражений. Их уравнения легко находятся из соответствующих матриц $\cos(2\theta)$ и $\sin(2\theta)$ в первой строке, так что их можно использовать в вычислении наклона $\operatorname{tg}\theta$. Тогда уравнение прямой, вдоль которой отражается $\mathbf{V}^T$, имеет вид $y = \left(\operatorname{tg}\theta_{\mathbf{V}^T}\right)x = 0{,}8962347008436108x$, а уравнение прямой, вдоль которой отражается $\mathbf{U}$, будет $y = \left(\operatorname{tg}\theta_{\mathbf{U}}\right)x = 0{,}17903345403184898x$. Так как $\mathbf{A}\vec{x} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\vec{x}$, сначала $\vec{x}$ отражается вдоль прямой $L_{\mathbf{V}^T}$, после чего получаем $\mathbf{V}^T\vec{x}$. Затем, при умножении на $\mathbf{\Sigma}$ слева, первая координата $\mathbf{V}^T\vec{x}$ растягивается по горизонтали по первому син-

гулярному значению, а вторая координата — по второму, что дает в итоге $\Sigma \mathbf{V}^T \vec{x}$. Наконец, при умножении на **U** вектор $\Sigma \mathbf{V}^T \vec{x}$ отражается вдоль прямой $L_{\mathbf{U}}$, что дает $\mathbf{A}\vec{x} = \mathbf{U}\Sigma\mathbf{V}^T\vec{x}$. Рисунок 6.9 иллюстрирует этот процесс.

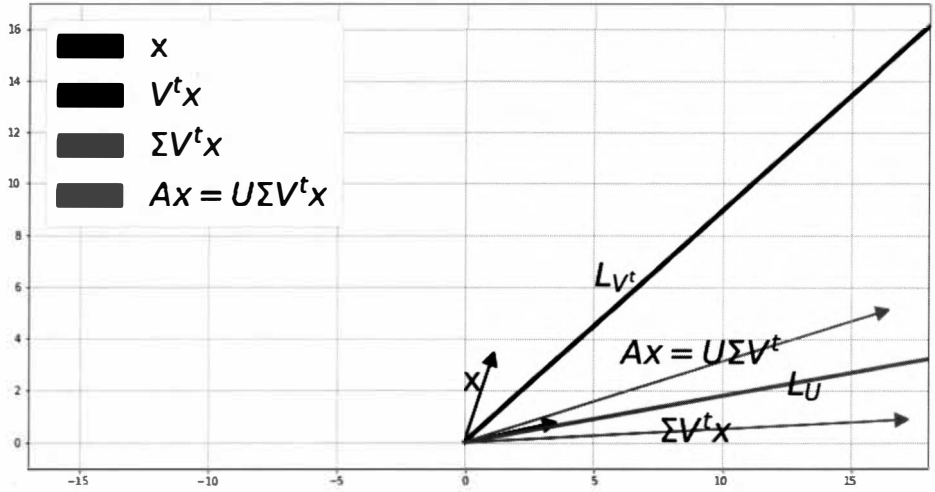


Рис. 6.9. Воздействие матрицы **A** на общий вектор $\vec{x}$.
Преобразование выполняется поэтапно методом сингулярного разложения

Три способа умножения матриц

В эпоху больших данных остро встает вопрос об эффективных алгоритмах умножения матриц. Теоретически существует три способа умножения двух матриц $\mathbf{A}_{m\times n}$ и $\mathbf{B}_{n\times s}$.

Строково-столбцовый метод.

Производит одну запись $(ab)_{ij}$ за один раз, взяв точечное произведение *i*-й строки **A** на *j*-й столбец **B**:

$$(ab)_{ij} = \mathbf{A}_{\text{строка}_i}\mathbf{B}_{\text{столбец}_j} = \sum_{k=1}^{n} a_{ik}b_{kj}.$$

Метод "столбец — столбцы".

Производит один столбец $(\mathbf{AB})_{\text{столбец}_i}$ путем линейного объединения столбцов **A** с использованием записей *i*-го столбца **B**:

$$(\mathbf{AB})_{\text{столбец}_i} = b_{1i}\mathbf{A}_{\text{столбец}_1} + b_{2i}\mathbf{A}_{\text{столбец}_2} + \ldots + b_{ni}\mathbf{A}_{\text{столбец}_n}.$$

Столбцово-строковый метод.

Производит *ранг матрицы по одному экземпляру* произведения, умножая первый столбец **A** на первую строку **B**, второй столбец **A** на вторую строку **B** и т. д. Затем складывает все матрицы первого ранга вместе, и получается итоговое **AB**:

$$\mathbf{AB} = \mathbf{A}_{\text{столбец}_1}\mathbf{B}_{\text{строка}_1} + \mathbf{A}_{\text{столбец}_2}\mathbf{B}_{\text{строка}_2} + \ldots + \mathbf{A}_{\text{столбец}_n}\mathbf{B}_{\text{строка}_n}.$$

Но как это поможет понять выгоду сингулярного разложения? Произведение $\mathbf{A} = \mathbf{U\Sigma V}^T$ сингулярного разложения можно представить в виде суммы матриц первого ранга с помощью *столбцово-строкового метода* умножения матриц. Умножаем матрицу $\mathbf{U\Sigma}$ (которая масштабирует каждый столбец $\mathbf{U}_{\text{столбец}_i}$ из **U** на σ_i) на $\mathbf{V}^T$:

$$\mathbf{A} = \mathbf{U\Sigma V}^T = \sigma_1\mathbf{U}_{\text{столбец}_1}\mathbf{V}^T_{\text{строка}_1} + \sigma_2\mathbf{U}_{\text{столбец}_2}\mathbf{V}^T_{\text{строка}_2} + \ldots + \sigma_r\mathbf{U}_{\text{столбец}_r}\mathbf{V}^T_{\text{строка}_r},$$

где r — количество ненулевых сингулярных значений **A** (также называется *рангом* **A**).

Это выражение замечательно тем, что оно разбивает **A** на сумму матриц первого ранга, расположенных в порядке убывания их значимости, поскольку σ расположены в порядке убывания. Более того, оно дает прямой способ аппроксимации **A** матрицами более низкого ранга, убрав нижние сингулярные значения. Теорема Эккарта — Янга — Мирского (**https://oreil.ly/5eYKY**) утверждает, что это действительно *лучший способ* найти низкоранговую аппроксимацию **A**, когда *близость* аппроксимации измеряется с помощью нормы Фробениуса (представляющей собой квадратный корень суммы квадратов сингулярных значений, **https://oreil.ly/Yev1c**) для матриц. Далее в этой главе мы воспользуемся разложением **A** первого ранга для сжатия цифровых изображений.

Алгоритмы умножения матриц

Поиск эффективных алгоритмов умножения матриц — важная, но на удивление трудная задача. В алгоритмах умножения матриц ценной считается экономия даже на одной операции умножения (экономия на сложении не столь значительна). Недавно компания DeepMind разработала алгоритм AlphaTensor (2022 г., **https://oreil.ly/HZPbd**) для автоматического поиска более эффективных алгоритмов умножения матриц. Это важнейшая веха, т. к. умножение матриц является фундаментальной частью огромного количества технологий, включая нейронные сети, компьютерную графику, научные вычисления.

Большая картина

До сих пор мы рассматривали сингулярное разложение матрицы $\mathbf{A} = \mathbf{U\Sigma V}^T$ с точки зрения воздействия **A** на пространство и с точки зрения аппроксимации **A** матрицами более низкого ранга. Прежде чем мы перейдем к приложениям ИИ, взглянем на ситуацию с высоты птичьего полета и рассмотрим общую картину.

В зависимости от конкретного случая использования матрицы действительных чисел мы хотим выяснить следующее.

- Если матрица представляет собой интересующие нас данные, например изображения или табличные данные, то какие компоненты этой матрицы (данных) будут самыми важными?
- По каким важным направлениям распределяются в основном данные (направления с наибольшим разбросом данных)?
- Если представить матрицу $\mathbf{A}_{m\times n}$ как преобразование из исходного пространства $\mathbb{R}^n$ в целевое пространство $\mathbb{R}^m$, то как эта матрица повлияет на векторы в $\mathbb{R}^n$? На какие векторы они направлены в $\mathbb{R}^m$?
- Как эта матрица влияет на само пространство? Поскольку это линейное преобразование, мы знаем, что искривления пространства не происходит, зато есть растяжение, сжатие, поворот и отражение пространства.
- Многие физические системы можно представить в виде системы линейных уравнений $\mathbf{A}\vec{x} = \vec{b}$. Как решить эту систему (найти $\vec{x}$)? Какой способ наиболее эффективен в зависимости от свойств **A**? Если решения нет, то существует ли приближенное решение, удовлетворяющее нашим целям? Отметим, что здесь мы ищем неизвестный вектор $\vec{x}$, который при воздействии **A** на него преобразуется в $\vec{b}$.

На все эти вопросы можно ответить с помощью сингулярного разложения. Первые два вопроса относятся к самой матрице, а вторые два связаны с эффектом умножения матрицы на векторы (матрица воздействует на пространство и векторы в этом пространстве). Последний вопрос связан с очень важной проблемой решения систем линейных уравнений и встречается во многих приложениях.

Таким образом, матрицу чисел можно разобрать двумя способами:

- Каковы ее внутренние свойства?
- Каковы ее свойства с точки зрения преобразования?

Эти два вопроса связаны между собой, поскольку внутренние свойства матрицы влияют на то, как она воздействует на векторы и на пространство.

Нужно иметь в виду следующие свойства.

- **A** переводит ортонормальные векторы v_i (правые сингулярные векторы) исходного пространства в скалярно кратные ортонормальные векторы u_i (левые сингулярные векторы) целевого пространства:

$$\mathbf{A}v_i = \sigma_i u_i.$$

- В случае квадратной матрицы абсолютное значение ее определителя равно произведению всех ее сингулярных значений: $\sigma_1\sigma_2\cdots\sigma_r$.

- Число обусловленности матрицы относительно нормы l^2, являющейся обычным расстоянием в евклидовом пространстве, представляет собой отношение наибольшего сингулярного значения к наименьшему сингулярному значению:

$$k = \frac{\sigma_1}{\sigma_r}.$$

Число обусловленности и вычислительная устойчивость

Число обусловленности имеет большое значение в плане вычислительной устойчивости.

- Число обусловленности определяет, насколько **A** растягивает пространство. Если число обусловленности слишком большое, то пространство довольно сильно растягивается в одном направлении относительно другого направления, и проводить вычисления в таком сильно растянутом пространстве может быть опасно. Решение $\mathbf{A}\vec{x} = \vec{b}$, когда **A** имеет большое число обусловленности, делает решение $\vec{x}$ неустойчивым в том смысле, что оно чрезвычайно чувствительно к возмущениям $\vec{b}$. Небольшая ошибка в $\vec{b}$ приведет к решению $\vec{x}$, которое будет сильно отличаться от решения без ошибки в $\vec{b}$. Эту неустойчивость легко представить геометрически.
- Числовое решение $\mathbf{A}\vec{x} = \vec{b}$ (например, методом исключения Гаусса) и итерационные методы хорошо работают, когда участвующие матрицы имеют приемлемые (не очень большие) числа обусловленности.
- Одна особенность матрицы с чересчур большим числом обусловленности заключается в том, что она настолько растягивает пространство, что практически сворачивается в пространство меньшей размерности. Самое интересное, что если мы решим отбросить это очень маленькое сингулярное значение и, следовательно, будем работать в свернувшемся пространстве меньшей размерности, то наши вычисления станут совершенно нормальными. Таким образом, на границах экстремальности лежит нормальность, только теперь нормальность лежит в более низкой размерности.
- Во многих итерационных численных методах, в том числе и в весьма значимом методе градиентного спуска, в анализе участвуют матрицы. Если число обусловленности этих матриц слишком велико, то итерационный метод может не сойтись к решению. Число обусловленности определяет скорость сходимости итерационных методов.

Компоненты сингулярного разложения

В этой главе мы разбирали только одну формулу: $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Для вычисления записей **U**, **Σ**, **V** использовался Python, но что это за записи? Ответ будет кратким,

если мы, конечно, знаем, что такое собственные векторы и собственные значения, о чем мы поговорим в следующем разделе. Пока же перечислим компоненты **U**, **Σ**, **V**.

- Столбцы **V** (правые сингулярные векторы) являются ортонормальными собственными векторами симметричной матрицы $\mathbf{A}^T\mathbf{A}$.
- Столбцы **U** (левые сингулярные векторы) являются ортонормальными собственными векторами симметричной матрицы $\mathbf{A}\mathbf{A}^T$.
- Сингулярные значения $\sigma_1, \sigma_2, ..., \sigma_r$ — это квадратные корни собственных значений $\mathbf{A}^T\mathbf{A}$ или $\mathbf{A}\mathbf{A}^T$. Сингулярные значения неотрицательны и расположены в порядке убывания. Сингулярные значения могут быть нулевыми.
- $\mathbf{A}v_i = \sigma_i u_i$.

Каждая вещественная симметричная положительная полубесконечная (с неотрицательными собственными значениями) матрица диагонализируема $\mathbf{S} = \mathbf{PDP}^{-1}$, что означает ее подобие диагональной матрице **D** при рассмотрении в другом наборе координат (столбцов **P**). $\mathbf{A}^T\mathbf{A}$ и $\mathbf{A}\mathbf{A}^T$ являются симметричными положительными полубесконечными матрицами, поэтому они диагонализируемы.

Сингулярное и спектральное разложения

Для того чтобы разобраться в компонентах сингулярного разложения, важно больше узнать о симметричных матрицах. Это также позволяет увидеть разницу между сингулярным разложением $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$ и спектральным разложением $\mathbf{A} = \mathbf{PDP}^{-1}$, если таковое существует.

Сингулярное разложение выполняется во всех случаях, но спектральное разложение — только в случае особых матриц, которые называются диагонализируемыми. Прямоугольные матрицы не бывают диагонализируемыми. Квадратные матрицы могут и быть, и не быть диагонализируемыми. Если квадратная матрица диагонализируема, то сингулярное и спектральное разложения *не равны*, если только матрица не является симметричной и не имеет неотрицательных собственных значений.

Для наглядности построим иерархию матриц по степени их предпочтительности.

1. Лучшими и самыми простыми матрицами являются квадратные диагональные матрицы с одинаковыми номерами по диагонали.
2. Вторыми по качеству считаются квадратные диагональные матрицы **D**, которые необязательно имеют одинаковые номера по диагонали.
3. Третьими в этой иерархии считаются симметричные матрицы. Они имеют вещественные собственные значения и ортогональные собственные векторы. По типу они ближе всего к диагональным в том смысле, что диагонализируемы

$\mathbf{S} = \mathbf{PDP}^{-1}$, или подобны диагональной матрице после изменения базиса. Столбцы **P** (собственные векторы) ортогональны.

4. К четвертым относятся квадратные матрицы, диагонализируемые $\mathbf{A} = \mathbf{PDP}^{-1}$.
5. Пятые — это все остальные. Они не диагонализируемы, т. е. не существует такого изменения базиса, которое сделало бы их диагональными; однако имеется еще один способ сделать их похожими на диагональную матрицу с помощью сингулярного разложения $\mathbf{A} = \mathbf{U\Sigma V}^T$. Здесь **U** и **V** отличны друг от друга и имеют ортонормированные столбцы и строки. Их обратная величина довольно проста — она равна их транспонированию. Сингулярное разложение работает как с квадратными, так и с неквадратными матрицами.

В случае матрицы **A** матрицы $\mathbf{A}^T\mathbf{A}$ и $\mathbf{AA}^T$ являются симметричными и положительно полубесконечными (т. е. их собственные значения неотрицательны), поэтому они диагонализуются двумя базисами ортогональных собственных векторов. При делении на норму этих ортогональных собственных векторов они становятся ортонормальными. Это, соответственно, столбцы **V** и **U**.

Матрицы $\mathbf{A}^T\mathbf{A}$ и $\mathbf{AA}^T$ имеют совершенно одинаковые неотрицательные собственные значения $\lambda_i = \sigma_i^2$. Расположив их квадратные корни в порядке убывания (сохраняя соответствующий порядок собственных векторов в **U** и **V**), в сингулярном разложении мы получим диагональную матрицу **Σ**.

А если исходная матрица симметрична? Как связано сингулярное разложение $\mathbf{A} = \mathbf{U\Sigma V}^T$ с ее диагонализацией $\mathbf{A} = \mathbf{PDP}^{-1}$? Столбцы матрицы **P**, являющиеся собственными векторами симметричной матрицы **A**, ортогональны. При делении на их длину они становятся ортонормированными. Сложив эти ортонормальные собственные векторы в матрицу в порядке убывания абсолютной величины собственных значений, получаем как **U**, так и **V** для сингулярного разложения. Теперь, если все собственные значения симметричной матрицы **A** окажутся неотрицательными, сингулярное разложение этой положительной полубесконечной симметричной матрицы будет совпадать с ее собственным разложением при условии, что мы нормализуем ортогональные собственные векторы в **P**, упорядочив их в порядке убывания относительно неотрицательных собственных значений. Поэтому в данном случае $\mathbf{U} = \mathbf{V}$. А что если некоторые (или все) собственные значения отрицательны? Тогда $\sigma_i = |\lambda_i| = -\lambda_i$, но теперь нужно быть осторожными с соответствующими собственными векторами $\mathbf{A}v_i = -\lambda_i v_i = \lambda_i(-v_i) = \sigma_i u_i$. В результате **U** и **V** в сингулярном разложении становятся неравными. Итак, сингулярное разложение симметричной матрицы, имеющей несколько отрицательных собственных значений, можно легко получить из ее спектрального разложения, хотя это не совсем то же самое.

А что если исходная матрица не симметричная, а диагонализируемая? Как ее сингулярное разложение $\mathbf{A} = \mathbf{U\Sigma V}^T$ связано с ее диагонализацией $\mathbf{A} = \mathbf{PDP}^{-1}$? В этом случае собственные векторы **A**, которые являются столбцами **P**, в целом не ортогональны, поэтому сингулярное разложение такой матрицы не связано с ее спектральным разложением.

Вычисление сингулярного разложения

Как Python и другие программы численно вычисляют сингулярное разложение матрицы? Какие численные алгоритмы скрыты под капотом? Если кратко, то это — *QR-разложение*, *отражения Хаусхолдера*, *итерационные алгоритмы* для собственных значений и собственных векторов.

Теоретически для вычисления сингулярного разложения общей матрицы или собственных значений и собственных векторов квадратной матрицы требуется задать многочлен, равный 0, для решения собственных значений, а затем составить линейную систему уравнений для решения собственных векторов. Такой подход далеко не всегда можно применить на практике. Проблема нахождения нулей полинома довольно чувствительна к любым изменениям в коэффициентах полинома, поэтому вычислительная задача подвержена ошибкам округления, которые присутствуют в коэффициентах. Требуются устойчивые численные методы нахождения собственных векторов и собственных значений без необходимости численно вычислять нули полинома. Кроме того, нужно убедиться, что матрицы, участвующие в линейных системах уравнений, хорошо обусловлены, иначе такие популярные методы, как исключение Гаусса (*LU-разложение*), не будут работать.

Большинство численных реализаций сингулярного разложения стараются избежать вычисления $\mathbf{A}\mathbf{A}^T$ и $\mathbf{A}^T\mathbf{A}$. Это согласуется с одной из идей книги: вместо умножения матриц друг на друга лучше умножать матрицы на векторы. В распространенном численном методе сингулярного разложения используется алгоритм, который называется *отражениями Хаусхолдера*, для преобразования матрицы в бидиагональную матрицу (иногда перед этим выполняется *QR-разложение*), а затем применяются итерационные алгоритмы для нахождения собственных значений и собственных векторов. Разработка таких методов и их адаптация к типам и размерам матриц, встречающимся в приложениях, входит в зону ответственности численной линейной алгебры. В следующем подразделе мы рассмотрим итерационный метод вычисления одного собственного значения и соответствующего ему собственного вектора заданной матрицы.

Численное вычисление собственного вектора

Собственный вектор квадратной матрицы $\mathbf{A}$ — это ненулевой вектор, который при умножении на $\mathbf{A}$ не меняет своего направления; вместо этого он масштабируется только на собственное значение λ:

$$\mathbf{A}v = \lambda v.$$

Приведем итерационный алгоритм, который является простым численным методом нахождения собственного вектора матрицы, соответствующего ее максимальному собственному значению:

1. Начинаем со случайного единичного вектора (длины 1) v_0.
2. Умножаем на $\mathbf{A}$: $v_{i+1} = \mathbf{A}v_i$.

3. Делим на длину v_{i+1}, чтобы размер векторов не был слишком большим.

4. Останавливаемся, достигнув сходимости.

Этот итерационный метод достаточно прост, но все же у него есть один недостаток — он находит только один собственный вектор матрицы, соответствующий ее максимальному собственному значению. Таким образом, он находит направление, которое при использовании **A** растягивается сильнее всего.

Например, рассмотрим матрицу $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 2 & -3 \end{pmatrix}$. Начнем с вектора $\vec{v}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ и применим описанный выше алгоритм. После 28 итераций алгоритма получаем вектор $\vec{v} = \begin{pmatrix} -0,38268343 \\ 0,92387953 \end{pmatrix}$. Код можно посмотреть в блокноте Jupyter, а вывод приведем здесь:

```
[1, 0]
[0.4472136   0.89442719]
[ 0.78086881 -0.62469505]
[-0.1351132   0.99083017]
[ 0.49483862 -0.86898489]
[-0.3266748   0.9451368]
[ 0.40898444 -0.91254136]
[-0.37000749  0.92902877]
[ 0.38871252 -0.92135909]
[-0.37979817  0.92506937]
[ 0.3840601  -0.9233081]
[-0.38202565  0.92415172]
[ 0.38299752 -0.92374937]
[-0.38253341  0.92394166]
[ 0.38275508 -0.92384985]
[-0.38264921  0.92389371]
[ 0.38269977 -0.92387276]
[-0.38267563  0.92388277]
[ 0.38268716 -0.92387799]
[-0.38268165  0.92388027]
[ 0.38268428 -0.92387918]
[-0.38268303  0.9238797]
[ 0.38268363 -0.92387945]
[-0.38268334  0.92387957]
[ 0.38268348 -0.92387951]
[-0.38268341  0.92387954]
[ 0.38268344 -0.92387953]
[-0.38268343  0.92387953]
[ 0.38268343 -0.92387953]
```

```
 v= [-0.38268343  0.92387953]
Av= [ 1.46507563 -3.53700546]
$\lambda=$ -3.828427140993716
```

На рис. 6.10 показана эта итерация. Важно отметить, что все векторы имеют длину 1, и направление вектора не меняется, когда алгоритм сходится, следовательно, захватывается собственный вектор матрицы **A**. В течение нескольких последних итераций знак продолжает колебаться, так что вектор продолжает менять ориентацию, и собственное значение должно быть отрицательным. Действительно, оно равно $\lambda = -3{,}828427140993716$.

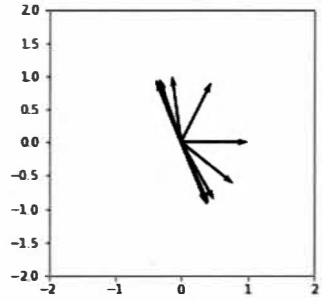


Рис. 6.10. Начинаем с $\vec{v}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, затем умножаем на **A** и нормализуем до сходимости к собственному вектору

Псевдоинверсия

Многие физические системы могут быть представлены (или аппроксимированы) системой линейных уравнений $\mathbf{A}\vec{x} = \vec{b}$. Если $\vec{x}$ — интересующий нас неизвестный вектор, то его можно найти делением на матрицу **A**. Матричный эквивалент деления — это нахождение обратной $\mathbf{A}^{-1}$, поэтому решение будет $\vec{x} = \mathbf{A}^{-1}\vec{b}$. Матрица, имеющая свою противоположность, называется *обратной матрицей*. Это квадратные матрицы с ненулевым определителем (*определитель* — это произведение собственных значений; произведение сингулярных значений и определитель будут иметь абсолютно одинаковое значение). Но как быть с системами, чьи матрицы прямоугольные? А как насчет систем с необратимыми матрицами? Или имеющих квадратные и обратные матрицы, но почти необратимые (их определитель очень близок к нулю)? Нам все еще важно найти решения для таких систем. Сила сингулярного разложения заключается в том, что оно подходит для любой матрицы, в том числе и для упомянутых выше, и позволяет инвертировать любую матрицу.

Для любой матрицы и ее сингулярного разложения $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ можно определить ее *псевдоинверсию* как:

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T,$$

где Σ^+ получается из Σ путем инвертирования всех диагональных записей, кроме нулевых (или очень близких к нулю, если матрица плохо обусловлена).

Это позволяет найти решения любой системы линейных уравнений $\mathbf{A}x = b$, и в нашем случае это будет $x = \mathbf{A}^+ b$.

При наличии обратной матрицы *псевдоинверсия* полностью совпадает с ней.

Применение сингулярного разложения к изображениям

Наконец, мы подошли к практическому применению сингулярного разложения. Начнем со сжатия изображений. Цифровые изображения хранятся в виде матриц чисел, где каждое число соответствует интенсивности пиксела. Мы будем использовать сингулярное разложение для уменьшения объема памяти изображения, не теряя при этом самой важной информации. Все, что нужно сделать, — отбросить незначительные сингулярные значения вместе с соответствующими столбцами $\mathbf{U}$ и строками $\mathbf{V}^T$. Математическое выражение, которое нам в этом поможет, имеет вид:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T = \sigma_1 \mathbf{U}_{\text{столбец}_1} \mathbf{V}^T_{\text{строка}_1} + \sigma_2 \mathbf{U}_{\text{столбец}_2} \mathbf{V}^T_{\text{строка}_2} + \ldots + \sigma_r \mathbf{U}_{\text{столбец}_r} \mathbf{V}^T_{\text{строка}_r}.$$

Напомним, что σ упорядочены от наибольшего значения к наименьшему, поэтому идея заключается в том, чтобы оставить несколько первых больших σ и отбросить остальные σ, которые в любом случае будут малы.

Рассмотрим снимок, приведенный на рис. 6.11. Код и подробную информацию можно посмотреть на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**). Каждое цветное изображение имеет три канала: красный, зеленый, синий (рис. 6.12 и 6.13). Каждый канал — это такая же матрица чисел, как и те, что мы рассматривали ранее в текущей главе.

Каждый канал изображения на рис. 6.11 представляет собой матрицу размера 960×714, так что для хранения всего изображения потребуется $960 \times 714 \times 3 = 2\,056\,320$ чисел. Можно только представить, какими будут требования к хранению потокового видео, содержащего множество кадров изображения. Нам нужен такой механизм сжатия, чтобы не израсходовать всю память. Вычисляем сингулярное разложение для каждого канала (см. рис. 6.14, на котором показано сингулярное разложение для красного канала). Затем мы выполняем массивное сокращение, сохраняя для каждого канала только первые 25 сингулярных значений (из 714), 25 столбцов $\mathbf{U}$ (из 960) и 25 строк $\mathbf{V}^T$ (из 714). Мы видим существенную экономию памяти для каждого канала: $\mathbf{U}$ теперь имеет размер 960×25, $\mathbf{V}^T$ равно 25×714, и нам нужно хранить только 25 сингулярных значений (нули диагональ-

ной матрицы $\mathbf{\Sigma}$ хранить не нужно). В сумме это дает 41 875 чисел для каждого канала, следовательно, для всех трех каналов нам нужно хранить $41\,875 \times 3 = 125\,625$ чисел, что дает колоссальное сокращение требуемой памяти на 93%.

Собираем изображение в каждом канале по очереди, перемножая сжатые $\mathbf{U}$, $\mathbf{\Sigma}$ и $\mathbf{V}^T$:

$$\text{канал}_{\text{сжатый}} = \mathbf{U}_{960\times 25}\mathbf{\Sigma}_{25\times 25}\left(\mathbf{V}^T\right)_{25\times 714}.$$

На рис. 6.15 показан результат умножения для каждого красного, зеленого и синего каналов.

Наконец, мы накладываем слои на сокращенные каналы и получаем уменьшенное изображение (рис. 6.16). Очевидно, что в процессе потерялось много деталей, но это тот компромисс, с которым нам придется смириться.

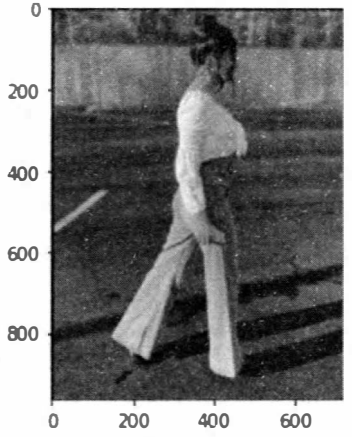


Рис. 6.11. Цифровое цветное изображение размером 960 × 714 × 3

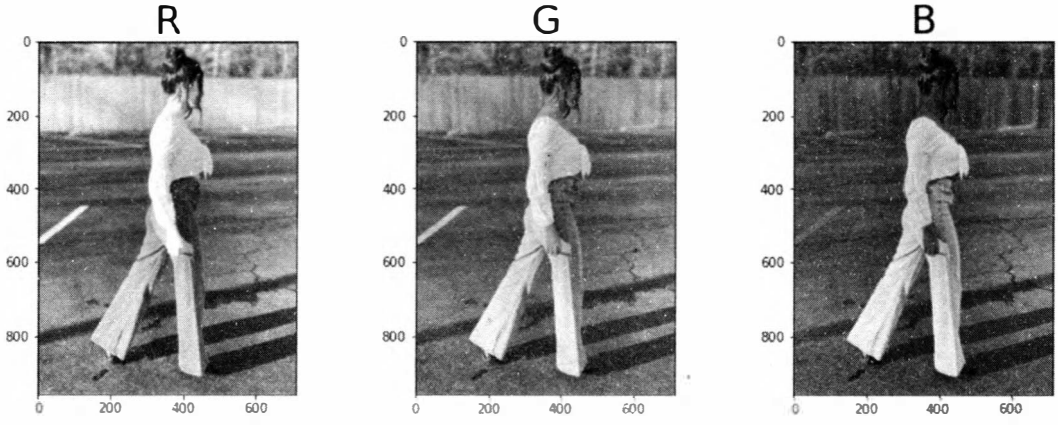


Рис. 6.12. Красный, зеленый и синий каналы цифрового изображения. Каждый канал имеет размер 960 × 714

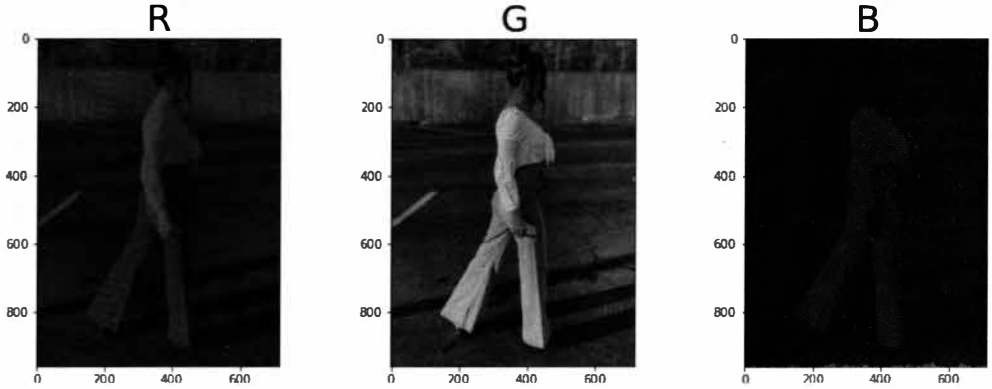


Рис. 6.13. Отображение красного, зеленого и синего оттенков для трех каналов цифрового изображения. Размер каждого из них равен 960 × 714 × 3. См. цветное изображение на GitHub (**https://oreil.ly/xhF8A**)

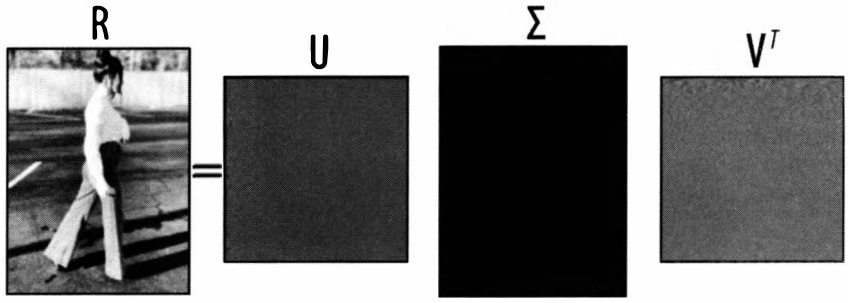


Рис. 6.14. Сингулярное разложение красного канала. Имеется 714 ненулевых сингулярных значений, но среди них мало существенных. Несмотря на то что диагональная матрица **Σ** выглядит полностью черной, на ее диагонали есть ненулевые сингулярные значения. Для отображения на таком уровне разрешения пикселы недостаточно яркие

Рис. 6.15. Красный, зеленый и синий каналы после понижения ранга. Для каждого канала мы сохранили только первые 25 сингулярных значений, первые 25 столбцов **U** и первые 25 строк $\mathbf{V}^T$

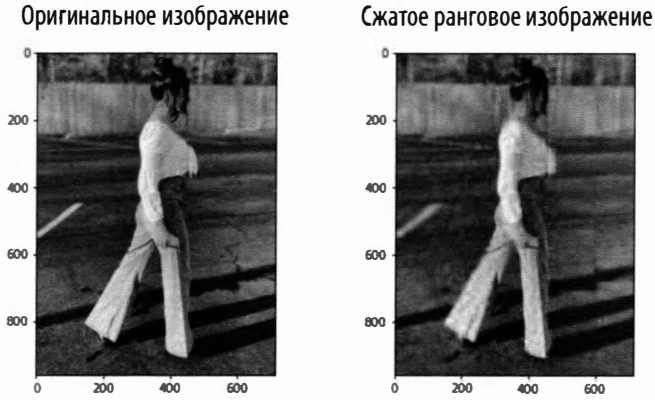


Рис. 6.16. Оригинальное изображение с 714 сингулярными значениями в сравнении с сжатым ранговым изображением, содержащим только 25 сингулярных значений. Оба изображения по-прежнему имеют размер 960 × 714 × 3, но требуют разного объема памяти

С передовыми методами сжатия изображений можно познакомиться в этой публикации: **https://oreil.ly/7dsZ3**.

Метод главных компонент и снижение размерности

Метод главных компонент широко применяется для анализа данных. Он используется для снижения размерности и кластеризации в неконтролируемом машинном обучении. В двух словах, это сингулярное разложение, выполняемое на матрице данных $\mathbf{X}$ после *центрирования* данных, что означает вычитание среднего значения каждого признака из каждого столбца признаков (каждого столбца $\mathbf{X}$). Главные компоненты — правые сингулярные векторы, которые являются строками $\mathbf{V}^T$ в уже знакомом нам разложении $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$.

Статистики любят описывать метод главных компонент (**https://oreil.ly/HdOrY**), используя язык дисперсии, или вариации данных, и *некоррелированности* данных. В итоге они оперируют собственными векторами *ковариационной матрицы* данных. Приведем известное описание метода главных компонент в статистике:

> "Этот метод позволяет снизить размерность набора данных, сохранив при этом как можно больше дисперсии, или статистической информации. Сохранение максимально возможной дисперсии означает поиск новых признаков, которые являются линейными комбинациями признаков из набора данных, последовательно максимизирующими дисперсию и не коррелирующими друг с другом".

Эти описания (правых сингулярных векторов центрированных данных и собственных векторов ковариационной матрицы) полностью совпадают, т. к. строки $\mathbf{V}^T$ — это собственные векторы матрицы $\mathbf{X}^T_{\text{центрированная}}\mathbf{X}_{\text{центрированная}}$, которая, в свою очередь, является ковариационной матрицей данных. Более того, термин

"некоррелированность" в статистике соответствует диагонализации в математике и линейной алгебре, а метод сингулярного разложения гласит, что любая матрица *ведет себя как* диагональная, т. е. как $\mathbf{\Sigma}$, если записать ее в новом наборе координат, а именно в строках $\mathbf{V}^T$, которые являются столбцами $\mathbf{V}$.

Рассмотрим это подробнее. Предположим, что $\mathbf{X}$ — центрированная матрица данных, а ее сингулярное разложение имеет вид $\mathbf{X} = \mathbf{U\Sigma V}^T$. Это то же самое, что $\mathbf{XV} = \mathbf{U\Sigma}$, или $\mathbf{XV}_{\text{столбец}_i} = \sigma_i \mathbf{U}_{\text{столбец}_i}$, если мы решаем выражение по столбцам. Отметим, что $\mathbf{XV}_{\text{столбец}_i}$ — это простая линейная комбинация признаков данных с использованием записей конкретного столбца $\mathbf{V}$. Теперь, по аналогии с тем, что мы уже выполняли на протяжении этой главы, можно отбросить менее значимые компоненты, т. е. столбцы $\mathbf{V}$ и $\mathbf{U}$, соответствующие меньшим сингулярным значениям.

Предположим, что в данных содержатся 200 признаков, но только 2 сингулярных значения являются важными, поэтому мы решили сохранить лишь первые 2 столбца $\mathbf{V}$ и первые 2 столбца $\mathbf{U}$. Таким образом, мы сократили размерность признаков с 200 до 2. Первый новый признак представляет собой линейную комбинацию всех исходных 200 признаков с использованием записей первого столбца $\mathbf{V}$, что равно $\sigma_1 \mathbf{U}_{\text{столбец}_1}$, а второй новый признак представляет собой линейную комбинацию всех исходных 200 признаков с использованием записей второго столбца $\mathbf{V}$, что равно $\sigma_2 \mathbf{U}_{\text{столбец}_2}$. Теперь подумаем об отдельных точках данных. Точка данных в матрице данных $\mathbf{X}$ имеет 200 признаков. Это значит, что для построения графика этой точки данных потребуется 200 осей. Однако, учитывая, что ранее мы сократили размерность, используя только первые две главные компоненты, у этой точки данных теперь будут только две координаты, соответствующие записям $\sigma_1 \mathbf{U}_{\text{столбец}_1}$ и $\sigma_2 \mathbf{U}_{\text{столбец}_2}$. Таким образом, если это была третья точка данных в наборе данных, то ее новыми координатами будут третья запись $\sigma_1 \mathbf{U}_{\text{столбец}_1}$ и третья запись $\sigma_2 \mathbf{U}_{\text{столбец}_2}$. Теперь можно легко построить график этой точки данных в двумерном пространстве, в отличие от графика в исходном 200-мерном пространстве.

Нам нужно выбрать, сколько сингулярных значений (и, соответственно, главных компонент) мы будем сохранять. Чем больше мы сохраним, тем более верным будет исходный набор данных, но и размерность, безусловно, будет выше. Такое решение об *усечении* (нахождение оптимального порога для усечения сингулярных значений) является предметом постоянных исследований. Одним из распространенных методов является определение желаемого ранга заранее или сохранение определенной дисперсии в исходных данных. Другие методы предполагают построение графика всех сингулярных значений, наблюдение очевидного изменения на графике и принятие решения об усечении в этом месте с целью отделить от шума важные паттерны в данных.

Важно не только центрировать данные, но и стандартизировать их — вычесть среднее значение каждого признака и разделить на стандартное отклонение. Причина в том, что сингулярное разложение чувствительно к масштабу измерений при-

знаков. Когда мы стандартизируем данные, мы в итоге работаем не с ковариационной, а с корреляционной матрицей. Нужно запомнить, чтобы избежать путаницы: мы выполняем сингулярное разложение на стандартизированном наборе данных, после чего главные компоненты становятся столбцами **V**, а новые координаты точек данных — записями $\sigma_i \mathbf{U}_{\text{столбец}_i}$.

Метод главных компонент и кластеризация

В предыдущем разделе мы увидели, как с помощью метода главных компонент можно уменьшить количество признаков в данных, получив новый набор признаков, расположенных в иерархическом порядке с точки зрения вариативности данных. Это невероятно полезно для визуализации данных, поскольку их можно визуализировать только в двух или трех измерениях. Важно уметь визуализировать паттерны и корреляции в высокоразмерных, например, генетических данных. Иногда в пространстве сокращенной размерности, определяемом главными компонентами, можно наблюдать кластеризацию данных по категориям. Например, если в набор данных включены пациенты как с онкологией, так и без нее, а также их генетические проявления (как правило, тысячи), мы можем заметить, что при построении данных в пространстве первых трех главных компонент пациенты с онкологией группируются отдельно от пациентов без онкологии.

Приложение для социальных сетей

В рамках той же тематики метода главных компонент и кластеризации в недавней публикации (декабрь 2020 г., **https://oreil.ly/H1sLr**) Дэна Виленчика (Dan Vilenchik) представлено замечательное приложение из социальных сетей — неконтролируемый подход к определению характеристик пользователей в социальных сетях. Ниже приведены выдержки из доклада автора на эту тему, а также аннотация его статьи.

> "Осмысление данных, которые автоматически собираются с онлайн-платформ, таких как социальные сети или платформы электронного обучения, представляется сложной задачей — данные массивны, многомерны, искажены и неоднородны (состоят из по-разному ведущих себя субъектов). В этом докладе мы сосредоточимся на центральной задаче, общей для всех социальных онлайн-платформ, — задаче определения характеристик пользователей. Например, автоматическая идентификация спамера или бота в Twitter, или студента, отсоединившегося от платформы электронного обучения.
>
> Онлайн-каналы социальных сетей играют центральную роль в нашей жизни. Определение характеристик пользователей в социальных сетях — давний вопрос, восходящий к 50-м годам, когда Кац и Лазарсфельд изучали влияние в „Массовой коммуникации". В эпоху машинного обучения эта задача обычно ставится как задача

> контролируемого обучения, где необходимо предсказать целевую переменную: возраст, пол, политические взгляды, доход и т. д. В данной работе мы изучаем, чего можно достичь с помощью неконтролируемого способа. В частности, мы используем метод главных компонент, чтобы понять, какие базовые паттерны и структуры свойственны тем или иным платформам социальных сетей и почему. Мы пришли к парадоксу Симпсона, способному дать нам более глубокое понимание процесса характеристики пользователей на таких платформах на основе данных".

Идея метода главных компонент для создания кластеров с максимальной дисперсией в данных будет неоднократно встречаться нам на протяжении всей книги.

Латентно-семантический анализ

Латентно-семантический анализ данных естественного языка (документов) похож на метод главных компонент числовых данных.

В этом случае мы хотим проанализировать связи между набором документов и содержащимися в них словами. Гипотеза распределения для латентного семантического анализа гласит, что слова, имеющие схожее значение, встречаются в схожих фрагментах текста, а значит, и в схожих документах. Компьютеры понимают только числа, поэтому прежде, чем приступать к анализу документов со словами, необходимо придумать их числовое представление. Одним из таких представлений является матрица *подсчета слов* $\mathbf{X}$, где столбцы представляют уникальные слова (например, яблоко, апельсин, собака, город, интеллект и т. д.), а строки — документ. Такая матрица очень большая, но довольно разреженная (имеет много нулей). В ней слишком много слов (это и есть признаки), поэтому нам нужно уменьшить размерность признаков, сохранив при этом структуру сходства между документами (точками данных). Теперь мы знаем, что делать: выполните разложение по сингулярным значениям матрицы количества слов, $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$, затем отбросьте меньшие сингулярные значения вместе с соответствующими столбцами из $\mathbf{U}$ и строками из $\mathbf{V}^T$. Теперь мы можем представить каждый документ в низкоразмерном пространстве (линейных комбинаций слов) точно так же, как анализ главных компонент позволяет представить данные в низкоразмерном пространстве признаков.

После сокращения размерности можно сравнить документы с помощью *косинусного сходства* — вычислением косинуса угла между двумя векторами, представляющими документы. Если косинус близок к 1, то документы направлены в одну сторону в пространстве слов и, следовательно, представляют собой достаточно похожие документы. Если косинус близок к 0, то векторы, представляющие документы, ортогональны друг другу и, следовательно, сильно отличаются друг от друга.

В самом начале поисковая система Google больше напоминала индекс, затем она эволюционировала и стала принимать запросы на естественном языке. То же самое можно сказать и об автозаполнении в смартфонах. Латентно-семантический анализ

сжимает смысл предложения или документа в вектор, и когда он интегрируется в поисковую систему, это значительно повышает качество работы, позволяя находить именно те документы, которые мы ищем.

Рандомизированное сингулярное разложение

В этой главе мы специально избегали вычислений сингулярного разложения, т. к. это дорого. Однако мы упоминали, что в распространенных алгоритмах используется разложение матрицы, которое называется *QR-разложением* (позволяющее получить ортонормированный базис для столбцов матрицы данных), затем отражения Хаусхолдера для преобразования в бидиагональную матрицу и, наконец, итерационные методы для вычисления нужных собственных векторов и собственных значений. К сожалению, для постоянно растущих наборов данных матрицы оказываются слишком большими даже для этих эффективных алгоритмов. В этом случае единственное спасение — *рандомизированная линейная алгебра*. Эта область предлагает чрезвычайно эффективные методы разложения матриц, опираясь на *теорию случайной выборки*. Рандомизированные численные методы творят чудеса, обеспечивая точные разложения матриц и в то же время будучи гораздо дешевле детерминированных методов. Рандомизированное сингулярное разложение выбирает пространство столбцов большой матрицы данных $\mathbf{X}$, вычисляет *QR-разложение* выбранной (гораздо меньшей) матрицы, проецирует $\mathbf{X}$ на меньшее пространство ($\mathbf{Y} = \mathbf{Q}^T\mathbf{X}$, следовательно, $\mathbf{X} \approx \mathbf{QY}$), затем вычисляет сингулярное разложение $\mathbf{Y}$ ($\mathbf{Y} = \mathbf{U}\Sigma\mathbf{V}^T$). Ортогональная матрица $\mathbf{Q}$ аппроксимирует пространство столбцов $\mathbf{X}$, так что матрицы Σ и $\mathbf{V}$ одинаковы для $\mathbf{X}$ и $\mathbf{Y}$. Чтобы найти $\mathbf{U}$ для $\mathbf{X}$, мы можем вычислить его из $\mathbf{U}$ для $\mathbf{Y}$ и $\mathbf{QU}_\mathbf{X} = \mathbf{QU}_\mathbf{Y}$.

Как и все рандомизированные методы, они должны сопровождаться границами ошибок в плане ожидания того, насколько далека исходная матрица $\mathbf{X}$ от выборочной $\mathbf{QY}$. У нас есть такие границы ошибок, но мы отложим их до *главы 7*, где рассматриваются большие случайные матрицы.

Итоги и перспективы

Самая ценная формула в этой главе имеет вид:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T = \sigma_1\mathbf{U}_{\text{столбец}_1}V^T_{\text{строка}_1} + \sigma_2\mathbf{U}_{\text{столбец}_2}V^T_{\text{строка}_2} + \ldots + \sigma_r\mathbf{U}_{\text{столбец}_r}\mathbf{V}^T_{\text{строка}_r}.$$

Это эквивалентно $\mathbf{XV} = \mathbf{U}\Sigma$ и $\mathbf{XV}_{\text{столбец}_i} = \sigma_i\mathbf{U}_{\text{столбец}_i}$.

Сила сингулярного разложения заключается в том, что оно дает возможность уменьшить ранг матрицы без потери существенной информации. Это позволяет сжимать изображения, сокращать размерность пространства признаков набора данных, вычислять сходство документов при обработке естественного языка.

Мы обсудили метод главных компонент, латентно-семантический анализ и структуру кластеризации, свойственную пространству главных компонент. Мы рассмотрели примеры использования анализа главных компонент в качестве метода неуправляемой кластеризации онкобольных в соответствии с экспрессией их генов, а также для характеристики пользователей социальных сетей.

В конце главы мы рассмотрели рандомизированное сингулярное разложение, подчеркнув идею, идущую лейтмотивом через всю книгу: если объекты слишком велики, выполните дискретизацию. *Случайность — довольно надежная штука!*

Если у вас возникнет желание изучить этот вопрос подробнее, можно прочитать о тензорных разложениях и массивах данных N-way, а также о важности выравнивания данных для корректной работы сингулярного разложения. Я узнала об этом из книги "Глубокое обучение" (**https://www.deeplearningbook.org/**). Другие популярные примеры можно найти в публикациях о собственном лице в контексте распознавания лиц.

ГЛАВА 7

Искусственный интеллект для естественного языка и финансов: векторизация и временные ряды

Они. Могут. Читать.
— *Хала*

Одной из особенностей человеческого интеллекта является овладение языком уже в самом раннем возрасте — это понимание письменной и устной речи, письменное и устное выражение мыслей, разговор между двумя или более людьми, перевод с одного языка на другой, а также использование языка для выражения сочувствия, передачи эмоций, обработки визуальных и аудиоданных, воспринимаемых из окружающей среды. Оставив в стороне философский вопрос о сознании, если машины обретут способность выполнять эти языковые задачи, расшифровывая смысл слов, на уровне, близком к или выше человеческого, то это станет главным двигателем на пути к общему ИИ. Эти задачи относятся к таким областям, как *обработка естественного языка*, *вычислительная лингвистика*, *машинное обучение* и/или *вероятностное моделирование языка*. Эти области безграничны, и много специалистов бесцельно блуждают в дымке разнообразных моделей, сулящих большие перспективы. Мы не должны заблудиться. В этой главе мы постараемся охватить целиком всю область обработки естественного языка, обеспечивая обзор с высоты птичьего полета и не углубляясь в дебри.

В нашей деятельности мы руководствуемся следующими вопросами:

- Какого рода задача стоит перед нами? Другими словами, какова наша цель?
- С каким типом данных мы имеем дело? Какой тип данных нам нужно собрать?
- Какие современные модели решают аналогичные задачи и работают с аналогичными типами данных? Если таковых не существует, придется придумывать модели самим.
- Как обучать эти модели? В каких форматах они получают данные? В каких форматах выдают результаты? Обладают ли они структурой, включающей функцию обучения, функцию потерь (или целевую функцию) и оптимизацию?
- Каковы преимущества и недостатки различных моделей по сравнению с другими?

- Имеются ли доступные пакеты или библиотеки Python для реализации моделей? К счастью, большинство моделей выпускаются сегодня с реализацией на Python и достаточно простыми API. Более того, существует множество предобученных моделей, доступных для загрузки и готовых к использованию в приложениях.
- Сколько вычислительной инфраструктуры нам потребуется для обучения и/или развертывания этих моделей?
- Можем ли мы что-то улучшить? Всегда есть возможности для совершенствования.

Также нужно *извлечь математические данные из наиболее эффективных моделей*. К счастью, это самая простая часть работы, поскольку в основе многих моделей лежит схожая математика, даже если они относятся к разным типам задач или к разным областям применения, например предсказание следующего слова в предложении или предсказание поведения фондового рынка.

В этой главе мы рассмотрим такие современные модели, как:

- трансформеры или модели внимания (представлены в 2017 году). Базовая математика здесь предельно проста — точечное произведение двух векторов;
- рекуррентные нейросети с долгой краткосрочной памятью (представлены в 1995 году). Базовая математика — *обратное распространение ошибки во времени*. Мы рассматривали обратное распространение в *главе 4*, но для рекуррентных сетей мы берем производные по времени;
- сверточные нейронные сети (представлены в 1989 году) для данных временны́х рядов. Базовая математика — *операция свертки*, которую мы обсуждали в *главе 5*.

Эти модели замечательно подходят для *данных временны́х рядов*, т. е. данных, которые появляются последовательно во времени. В качестве примеров данных временны́х рядов можно привести фильмы, аудиофайлы, такие как музыка и диктофонные записи, данные финансовых рынков, климатические данные, данные динамических систем, документы и книги.

Здесь можно задаться вопросом: почему документы и книги считаются зависящими от времени, даже если они уже написаны и просто *существуют*? Почему изображение не зависит от времени, а книга и в целом чтение и письмо зависят? Разгадка довольно проста.

- Читая книгу, мы осмысливаем сначала слово, затем фразу, предложение, абзац и т. д. Так мы усваиваем идеи и тематику всей книги.
- То же самое происходит, когда мы пишем документ, выводя по одному слову за раз, хотя вся идея, которую мы пытаемся выразить, уже *зашифрована* в нем до того, как мы записываем *последовательно* слова на бумагу.
- Когда мы подписываем изображение, само изображение не зависит от времени, но надпись (вывод) зависит.
- Когда мы резюмируем статью, отвечаем на вопрос или переводим с одного языка на другой, итоговый текст зависит от времени. Входной текст может зависеть

от времени, если он обрабатывается с помощью рекуррентной нейронной сети, или быть стационарным, если он обрабатывается сразу с помощью трансформера или сверточной модели.

До 2017 года самые популярные модели машинного обучения для обработки данных временны́х рядов были основаны либо на *сверточных нейросетях*, либо на *рекуррентных нейросетях с долгой краткосрочной памятью*. В 2017 году на смену им пришли *трансформеры*, позволившие полностью отказаться от рекуррентности в некоторых областях применения. Вопрос о том, устарели ли рекуррентные нейросети, остается открытым, но, учитывая, что в сфере ИИ изменения происходят ежедневно, довольно сложно предположить, какие модели выдержат испытание временем, а какие — нет. Более того, рекуррентные нейронные сети лежат в основе многих движков ИИ и до сих пор являются предметом активных исследований.

В этой главе мы ответим на следующие вопросы:

- Как преобразовать текст на естественном языке в числовые величины, которые сохранят смысл? Машины понимают только числа, и нам нужно обрабатывать естественный язык с помощью этих машин. Мы должны либо *векторизовать* выборки текстовых данных, либо *встроить* их в конечномерные векторные пространства.
- Как уменьшить размерность векторов, которая изначально была гигантской, что требовалось для представления естественного языка? Например, во французском языке около 135 000 отдельных слов, так что нам не обойтись без прямого кодирования слова во французском предложении с помощью векторов, имеющих по 135 000 записей в каждом.
- Рассматривает ли модель (в качестве своего входа и/или выхода) наши естественно-языковые данные как зависящую от времени последовательность, поступающую в нее по одному термину за раз, или как стационарный вектор, используемый сразу?
- Как именно работают различные модели обработки естественного языка?
- Почему в этой главе мы говорим, в том числе, о финансах?

Попутно мы обсуждаем типы приложений для естественного языка и финансов, для которых хорошо подходят наши модели. Мы делаем акцент не на программировании, а на математике, поскольку для таких моделей (особенно для языковых приложений) требуется значительная вычислительная инфраструктура. Например, переводчик DeepL Translator (**https://oreil.ly/9NZ6b**) генерирует переводы с помощью суперкомпьютера, работающего на гидроэнергии из Исландии, производительность которого достигает 5,1 петафлопс. Мы также отмечаем, что индустрия чипов, специализирующихся на ИИ, переживает небывалый подъем. Среди лидеров — NVIDIA, тензорный процессор Google (Google TPU), AWS Inferentia, AMD Instinct GPU и такие стартапы, как Cerebras и Graphcore. В то время как обычные чипы с трудом поспевали за законом Мура, который предсказывал удвоение вычислительной мощности каждые 18 месяцев, чипы, специализирующиеся на ИИ, значительно опередили этот закон.

И хотя в данной главе мы не будем писать код, важно отметить, что бóльшая часть программирования может быть выполнена с помощью библиотек TensorFlow и Keras для Python.

На протяжении всего повествования необходимо понимать, на каком этапе мы находимся — на этапе *обучения* модели или на этапе *предсказания* (использования предварительно обученной модели для решения задач).

Кроме того, важно различать, нужны ли для обучения нашей модели маркированные данные, например английские предложения с их французскими переводами в качестве меток, или она может обучаться на немаркированных данных, например вычислять значения слов из контекста.

Искусственный интеллект в обработке естественного языка

Приложения для обработки естественного языка встречаются повсюду. Эта технология интегрирована во многие аспекты нашей жизни, так что сегодня мы воспринимаем ее как должное, пользуясь приложениями в смартфонах, цифровыми календарями, цифровыми домашними помощниками типа Siri, Alexa и т. д. Приведенный ниже список частично адаптирован из замечательной книги "Обработка естественного языка в действии" Хобсона Лейна, Ханнеса Хапке и Коула Ховарда (Hobson Lane, Hannes Hapke, Cole Howard. Natural Language Processing in Action. — Manning, 2019), в которой отмечается актуальность обработки естественного языка:

- поиск и извлечение информации: веб-поиск, работа с документами, автозаполнение, чат-боты;
- электронная почта: спам-фильтры, классификация писем, расстановка приоритетов писем;
- редактирование: проверка орфографии, грамматики, рекомендации по стилю;
- анализ настроений: отзывы о продуктах, обслуживание клиентов, мониторинг морального состояния сообщества;
- общение: чат-боты, цифровые помощники типа Amazon и Alexa, составление расписаний;
- написание текстов: индексирование, алфавитный указатель, оглавление;
- анализ текста: резюмирование, извлечение информации, например анализ финансовых потоков избирательных кампаний и данных естественного языка (поиск связей между политическими донорами), подбор резюме для поиска работы, медицинская диагностика;
- право: юридические выводы, поиск прецедентов, классификация судебных повесток;
- новости: обнаружение событий, проверка фактов, составление заголовков;

- установление авторства: обнаружение плагиата, литературная экспертиза, обучение стилю;
- прогнозирование поведения: финансовые приложения, прогнозирование выборов, маркетинг;
- творческое письмо: сценарии фильмов, поэзия, тексты песен, боты для написания финансовых и спортивных новостей;
- создание подписей к изображениям: компьютерное зрение в сочетании с обработкой естественного языка;
- перевод иностранных языков: платформы Google Translate и DeepL Translate.

Несмотря на впечатляющие успехи на протяжении последних десяти лет, машины все еще далеки от овладения естественным языком. Дело в том, что процессы, связанные с этим, достаточно утомительны и требуют тщательного статистического учета и существенной памяти, аналогично тому, как человеку требуется память для овладения языками. Так что в этой области еще достаточно места для инноваций и вклада в развитие.

В последнее время языковые модели перешли от ручного кодирования к управлению данными. Они уже не реализуют жестко заданные логические и грамматические правила, а полагаются на выявление статистических связей между словами. Несмотря на утверждения одной из лингвистических школ, что грамматика является врожденным свойством человека, или, другими словами, *жестко закодирована* в нашем мозге, люди обладают поразительной способностью осваивать новые языки, даже с новыми, прежде неизвестными правилами грамматики. По личному опыту могу сказать, что попытки выучить грамматику нового языка, вероятно, затрудняют процесс обучения, хотя не буду настаивать на этом.

Одна из основных проблем заключается в том, что данные для естественного языка отличаются чрезвычайной высокой размерностью.

Тысячи языков содержат миллионы слов. Существуют огромные *корпусы документов*, например полные собрания сочинений разных авторов, миллиарды твитов, статьи в Википедии, новостные статьи, комментарии в Facebook, рецензии на фильмы и т. д. И первейшая задача — сократить количество измерений для эффективного хранения, обработки и вычислений, избежав при этом потери важной информации. В сфере ИИ это уже давно является общей проблемой, и как знать, сколько математических инноваций никогда бы не увидели свет, не будь у нас неограниченных возможностей хранения и подобающей вычислительной инфраструктуры.

Подготовка данных естественного языка к машинной обработке

Для обработки любой задачи, связанной с естественном языком, в первую очередь машина должна разбить текст на структурные элементы, сохраняющие смысл, намерения, контекст, темы, информацию и чувства. Для этого нужно установить со-

ответствие между словами и числовыми метками, используя такие процессы, как *токенизация*, *стемминг* (например, присвоение единственному и множественному числам слова одинаковой метки), *лемматизация* (связывание нескольких слов с похожим значением), *нормализация регистра* (например, присвоение словам с одинаковым написанием с заглавными и строчными буквами одинаковых меток) и др. Такие соответствия предназначены не для отдельных букв, составляющих слова, а для целых слов, пар или более слов (2-грамм или *n*-грамм), пунктуации, значимых капитализаций и т. д., несущих в себе смысл. Так создается *словарь*, или *лексикон*, числовых токенов, соответствующих заданному корпусу документов на естественном языке. В этом смысле создаваемый словарь, или лексикон, напоминает словарь для Python, где каждый отдельный объект структурного элемента естественного языка имеет свой уникальный токен.

Последовательность *n*-грамм — это последовательность из *n* слов, которые, будучи упорядоченными между собой, несут значение, отличное от значения каждого слова в отдельности. Например, 2-грамма — это пара слов, смысл которых вне пары меняется, например "мороженое"[1] или "не было", поэтому все 2-граммы целиком получают один числовой токен, сохраняя значение двух слов в их правильном контексте. Аналогичным образом 3-грамма — это тройка упорядоченных слов, например Джон Фицджеральд Кеннеди, и т. д. *Парсер* для естественного языка — это то же самое, что компилятор для компьютера. Не переживайте, если эти новые термины поначалу будут сбивать вас с толку. Для математических целей требуются только числовые токены, связанные с уникальными словами, *n*-граммами, эмодзи, пунктуацией и т. д., и итоговый словарь корпуса документов на естественном языке. Они сохраняются в словаре как объекты, благодаря чему можно легко переключаться между текстом и числовыми токенами.

Оставим подробности *токенизации*, *стемминга*, *лемматизации*, *парсинга* и прочих инструментов обработки данных естественного языка специалистам в области компьютерных наук в сотрудничестве с лингвистами. По сути, такое сотрудничество начинает терять свою актуальность по мере того, как модели развивают способность выявлять закономерности непосредственно из данных, вследствие чего постепенно пропадает необходимость в ручной кодировке лингвистических правил в моделях естественного языка. Отметим также, что хотя *стемминг* и *лемматизация* включены не во все системы обработки естественного языка, во всех них предусмотрена *токенизация*. Для их производительности решающее значение имеет качество токенизации текстовых данных. Это первый шаг, содержащий в себе базовые структурные элементы, которые представляют данные, вводимые в модели. Качество данных и способ их токенизации влияют на результаты работы всего цикла обработки естественного языка. В производственных приложениях используется *парсер spaCy*, выполняющий сегментацию предложений, токенизацию и множество других действий одновременно.

После токенизации и создания работоспособного словаря (коллекции числовых токенов и сущностей, которым они соответствуют в тексте на естественном языке)

[1] Англ. ice cream — мороженые сливки. — *Прим. перев.*

нужно представить всю документацию на естественном языке с помощью векторов чисел. Документы бывают как слишком длинными, например книжная серия, так и слишком короткими, например твит в Twitter либо простой поисковый запрос в Google или DuckDuckGo. Можно представить корпус из миллиона документов как набор из миллиона числовых векторов или матрицу с миллионом столбцов. Длина этих столбцов будет равна выбранному словарю или меньше, если мы решим дополнительно *сжать* эти документы. Говоря языком линейной алгебры, длина этих векторов — это размерность *векторного пространства*, в которое *вложены* наши документы.

Весь смысл этого процесса заключается в получении числовых векторных представлений документов для выполнения математических операций над ними — теперь на авансцену выходит линейная алгебра со своим арсеналом линейных комбинаций, проекций, точечных произведений, сингулярных разложений. Но есть одна оговорка: для приложений, связанных с естественным языком, длина векторов, представляющих документы или размер словаря, непомерно велика для выполнения каких-либо полезных вычислений с ними. Проклятие размерности превращается в реальность.

Проклятие размерности

При увеличении числа размерностей векторы экспоненциально удаляются друг от друга с точки зрения евклидова расстояния. В качестве примера обработки естественного языка можно привести сортировку документов на основе их *расстояния* до другого документа, например поискового запроса. Эта простая операция становится невозможной с превышением 20 измерений или около того, если для измерения близости документов пользоваться евклидовым расстоянием (подробнее см. статью в Википедии "Проклятие размерности", **https://oreil.ly/575IJ**). Таким образом, в приложениях, связанных с естественным языком, нужно использовать другую меру для измерения расстояния между документами. В ближайшее время мы обсудим *косинусное сходство*, измеряющее не евклидово расстояние, а *угол* между двумя векторами документов.

Поэтому основной движущей силой в моделях обработки естественного языка является представление документов с помощью более коротких векторов, передающих основные темы и сохраняющих смысл. Можно только вообразить, сколько потребуется уникальных токенов или их комбинаций, чтобы представить эту книгу, сохранив при этом основную суть.

Подводя итог, кратко обозначим алгоритм работы системы обработки естественного языка:

1. От текста — к числовым токенам, затем к оптимальному словарю для всего корпуса документов.
2. От документов с токенами — к высокоразмерным векторам чисел.

3. От высокоразмерных векторов чисел к более низкоразмерным векторам тематик с использованием таких инструментов, как *прямая проекция на меньшее подмножество словарного пространства* (просто отбрасывая часть словаря, делая соответствующие записи нулевыми), *латентно-семантический анализ* (проекция на специальные векторы, определяемые специальными линейными комбинациями векторов документов), *word2vec*, *doc2vec*, *векторы мысли*, *латентное размещение Дирихле* и др. Мы обсудим их вкратце позднее.

Как это обычно бывает в математическом моделировании, существует более одного способа представить документ в виде вектора чисел. Мы сами выбираем векторное пространство, в котором существуют или в которое *встроены* документы. Каждое векторное представление имеет свои преимущества и недостатки в зависимости от задач обработки естественного языка.

И среди них можно отметить наиболее простые.

Статистические модели и логарифмическая функция

Когда представление документа в виде вектора чисел начинается с подсчета количества появлений определенного термина в документе, модель векторизации документов считается статистической, т. к. она опирается на частоту.

Оперируя частотой терминов, в вычислениях лучше не применять грубые подсчеты, а воспользоваться *логарифмической функцией*. Она отлично подойдет при работе с величинами, которые могут быть как чрезмерно большими, так и чрезмерно малыми, или экстремально изменяться в масштабе. Рассматривая такие экстремальные значения или вариации в логарифмическом масштабе, мы возвращаем их в нормальную область.

Например, число 10^{23} является гигантским, но $\log\left(10^{23}\right) = 23\log 10$ — уже нет.

Аналогично, если слово "акула" встречается в 2 документах корпуса из 20 млн документов (20 млн/2 = 10 млн), а "кит" — в 20 документах того же корпуса (20 млн/20 = 1 млн), то разница составит 9 млн, что кажется чрезмерно завышенным для слов, которые встречаются, соответственно, в 2 и 20 документах. Вычисляя те же величины в *логарифмическом масштабе*, получаем, соответственно, $7\log 10$ и $6\log 10$ (основание логарифма неважно), что уже не кажется столь чрезмерным и больше соответствует встречаемости терминов в корпусе.

Необходимость использования логарифмической функции при работе, в частности, над подсчетом слов подкрепляется *законом Ципфа*. Закон гласит, что подсчет терминов в корпусе естественного языка естественным образом подчиняется степенно́й зависимости, следовательно, их лучше ограничить логарифмической функцией, преобразующей различия частоты терминов в линейную шкалу. Обсудим это далее.

Закон Ципфа для подсчета терминов

В обработке естественного языка закон Ципфа (**https://oreil.ly/hF3Ab**) работает с подсчетом слов. Он настолько интересен и увлекателен, что я не устояла перед искушением применить его в собственной книге. Трудно представить, что когда я пишу каждое слово в этой книге, мое уникальное количество слов действительно подчиняется какому-то закону. Неужели мы сами, равно как и то, как мы формулируем свои идеи и мысли, *настолько предсказуемы*? Оказывается, закон Ципфа распространяется не только на подсчет слов в корпусах документов, но и на множество окружающих нас вещей.

Закон Ципфа гласит: *в корпусе естественного языка, в котором термины упорядочены по частоте, частота первого элемента в два раза больше частоты второго элемента, в три раза — третьего и т. д.* То есть частота появления элемента в корпусе связана с его рангом: $f_1 = 2f_2 = 3f_3 = ...$

Мы можем проверить применимость закона Ципфа, построив график зависимости частоты терминов от их рангов и верифицируя *степенной закон*: $f_r = f(r) = f_1 r^{-1}$. Для верификации степенного закона проще построить график *в двойном логарифмическом масштабе*, отобразив $\log f_r$ относительно $\log r$. Если на логарифмическом графике получается прямая линию, тогда $f_r = f(r) = f_1 r^{\alpha}$, где α — наклон прямой.

Векторные представления документов на естественном языке

Перечислим наиболее распространенные векторные представления документов в современных моделях обработки естественного языка. Первые два — *частота термина* и *обратная частота термина документа* (term frequency — inverse document frequency, TF-IDF), являются статистическими представлениями, т. к. базируются на частоте, опираясь на подсчет появлений слов в документах. Хотя они немного сложнее, чем простое двоичное представление, определяющее наличие или отсутствие определенных слов, тем не менее они довольно поверхностны и ограничиваются простым *подсчетом слов*. И даже учитывая такую поверхностность, они необычайно полезны для таких приложений, как фильтрация спама и анализ настроений.

Векторное представление частоты термина документа или мешка слов

Представим документ с помощью *мешка слов*, сбрасывая порядок появления слов в документе. Несмотря на то что порядок слов содержит важную информацию о содержании документа, его нарушение позволяет верно аппроксимировать короткие предложения и фразы.

Предположим, что мы хотим поместить документ в *пространство словаря* из 10 000 токенов. Тогда вектор, представляющий этот документ, будет иметь 10 000 записей, каждая из которых подсчитывает, сколько раз каждый конкретный токен встречается в документе. По понятным причинам это называется векторным представлением документа в виде *частоты терминов*, или *мешка слов*, где каждая запись является неотрицательным целым числом.

Например, поисковый запрос в Google "прогноз погоды на завтра" векторизуется целиком как нули, кроме единиц в токенах, обозначающих слова "прогноз", "погоды", "на" и "завтра", если они есть в словаре. Затем этот вектор *нормализуется* делением каждой записи на общее количество терминов в документе, чтобы длина документа не искажала анализ. То есть, если в документе 50 000 терминов и термин "кошка" упоминается 100 раз, а в другом документе всего 100 терминов и термин "кошка" упоминается 10 раз, то очевидно, что для второго документа слово "кошка" важнее, чем для первого, и простой подсчет слов без нормализации не смог бы этого отразить.

Наконец, в некоторых классах обработки естественного языка берется логарифм каждого термина в векторе документов по причинам, о которых говорилось в предыдущих двух разделах.

Векторное представление частоты термина и обратной частоты термина документа

В этом случае для каждой записи вектора, представляющего документ, мы попрежнему подсчитываем количество появлений токена в документе, *а затем делим на количество документов в корпусе, где он появляется*.

Идея заключается в том, что если термин появляется много раз в одном документе и не очень часто в других, то этот термин будет считаться важным для данного документа и получит более высокий балл в соответствующей записи вектора, представляющего этот документ.

Во избежание деления на ноль, если термин не встречается ни в одном документе, то к знаменателю принято добавлять единицу. Например, обратная частота термина *"кошка"* равна:

$$\text{IDF для } \textit{"кошка"} = \frac{\text{количество документов в корпусе}}{\text{количество документов, содержащих } \textit{кошка}, +1}.$$

Безусловно, применяя векторное представлении частоты термина и обратной частоты термина документа (или TF-IDF), можно утверждать, что записи вектора документов — это неотрицательные рациональные числа, каждое из которых представляет собой меру *важности* конкретного токена для документа. Наконец, по тем же причинам, что и в предыдущем разделе, берем логарифм каждой записи в этом векторе.

Существует множество разнообразных методов TF-IDF, важных для информационно-поисковых систем, например Okapi BM25 (**https://oreil.ly/xrntI**) и Molino 2017 (**https://oreil.ly/jT56u**).

Векторное представление тематики документа, определенной с помощью скрытого семантического анализа

Векторы TF-IDF считаются очень высокоразмерными (столько же, сколько токенов в корпусе, так что их количество может исчисляться миллионами), разрежены и не получают дополнительного смысла при сложении или вычитании друг из друга. Для наших целей потребуются более компактные векторы, в сотни измерений или меньше, т. е. значительное урезание миллионных измерений. Помимо преимущества уменьшения размерности, эти векторы несут не только подсчет слов и статистику, но также и определенный смысл. Они называются *тематическими векторами*. То есть мы фокусируемся не на *статистике слов в документах*, а на *статистике связей между словами в документах и во всех корпусах*. Полученные таким образом тематики — это линейные комбинации подсчета слов.

Сначала применяем TF-IDF ко всей матрице **X** корпуса документов и получаем *тематическое пространство*. Применение TF-IDF в данном случае означает, что мы, пользуясь инструментарием линейной алгебры, вычисляем *сингулярное разложение матрицы* TF-IDF, т. е. $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. О сингулярном разложении говорится в *главе 6*, поэтому сейчас только рассмотрим, как оно применяется для создания тематического пространства корпуса. Сингулярное разложение из линейной алгебры в обработке естественного языка называется *латентно-семантическим анализом* (ЛСА, **https://oreil.ly/Ul0nb**). Далее мы будем использовать эти два термина как синонимы.

Следует обратить внимание на то, что представляют собой столбцы матрицы **X** TF-IDF корпуса: словесные токены или документы. В различных публикациях и программных пакетах используется либо одно, либо другое, поэтому нужно быть внимательнее и разобраться, с чем нам здесь предстоит работать для получения тематического пространства: с матрицей или ее транспонированием. В этом разделе мы говорим о таком представлении, в котором строки — это все слова (токены для слов, *n*-граммы и т. д.) всего корпуса, а столбцы — векторные представления TF-IDF для каждого документа в корпусе. Это несколько отличается от обычного представления матрицы данных, где признаки (слова в каждом документе) находятся в столбцах, а экземпляры (документы) — в строках. Причину такого перехода мы раскроем в ближайшее время. Тем не менее это не расходится с нашим представлением документов как векторов-столбцов.

Получив новый документ с его векторным представлением TF-IDF, преобразуем его в более компактный *тематический вектор*, *проецируя его на* тематическое пространство, полученное в результате сингулярного разложения матрицы TF-IDF корпуса. *Проецирование* в линейной алгебре — это обычное вычисление *точечного произведения* между соответствующими векторами и сохранение полученных скалярных чисел в записи нового *проецируемого* вектора:

- вектор TF-IDF документа, в котором количество записей соответствует количеству токенов во всем корпусе;

- *векторы весов тематик*, которые являются столбцами матрицы **U**, полученной в результате сингулярного разложения TF-IDF матрицы $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Опять же, каждый вектор весов тематик имеет количество записей, равное количеству токенов в корпусе. Изначально количество векторов весов тематик также соответствует количеству токенов во всем корпусе (столбцы **U**). *Весá* в столбце **U** говорят о вкладе, вносимом определенным токеном в тематику: большой вклад, если это положительное число, близкое к 1, неоднозначный вклад, если оно близко к 0, и даже отрицательный вклад, если это отрицательное число, близкое к –1. Важно отметить, что записи **U** — это всегда числа между –1 и 1, следовательно, интерпретируем их как весовые коэффициенты токенов нашего корпуса.

Выбор тематики и снижение размерности

Здесь можно задаться вопросом: если в каждом корпусе документов количество записей соответствует количеству токенов в нем, а количество векторов весов тематик равно количеству токенов, то где же здесь экономия и когда произойдет сжатие или снижение размерности? Продолжаем.

Цель 1.

Вычислить, сколько тематик содержится в нашем документе. Это простое точечное произведение между вектором TF-IDF документа и столбцом **U**, соответствующим интересующей нас тематике. Записываем его как первое скалярное число.

Цель 2.

Вычисляем, сколько *других* тематик содержится в документе. Это точечное произведение между вектором TF-IDF документа и столбцом **U**, соответствующим какой-либо другой интересующей нас тематике. Запишем его как второе скалярное число.

Цель 3.

Повторяем эти операции для одной или нескольких тематик (соответствует количеству столбцов **U**, что равно общему количеству токенов в корпусе), записывая скалярное число каждого точечного произведения. Теперь понятно, что "тематика" в таком контексте означает вектор-столбец, содержащий веса от –1 до 1, присвоенные каждому токену в корпусе документов.

Цель 4.

Снижаем размерность, сохраняя только нужные нам тематики. То есть, если мы решим оставить лишь две тематики, то *сжатым векторным представлением* нашего документа будет *двумерный вектор*, содержащий два скалярных числа, полученных с помощью двух точечных произведений вектора TF-IDF документа на векторы весов двух тематик. Таким образом, мы снизили размерность документа, возможно, с *миллионов* до всего двух. Согласитесь, это круто!

Цель 5.

Выбираем *правильные тематики* для представления наших документов. Именно здесь сингулярное разложение начинает творить чудеса. Столбцы **U** организова-

ны в порядке от самой важной тематики в корпусе к самой незначительной. Говоря языком статистики, столбцы организованы от тематики с *наибольшей дисперсией* в корпусе и, следовательно, содержащей больше информации, к тематике с наименьшей дисперсией, а значит, содержащей мало информации. Мы рассмотрим, как связаны дисперсия и сингулярное разложение, в *главе 10*. Таким образом, если мы решим спроецировать документ высокой размерности только на первые несколько векторов-столбцов **U**, от нас точно не ускользнет вариативность всевозможных тематик в корпусе и оценки их количества в нашем документе.

Цель 6.

Отметим, что *это — все еще статистический метод определения тематик в документе*. Мы начали с TF-IDF матрицы корпуса путем простого подсчета появлений токенов в документах. В этом смысле тематика определяется только на основании того, что в документах, относящихся к похожим вещам, используются похожие слова. Это отличается от выделения тематик на основе значений используемых в них слов. То есть, в случае когда одна и та же тема обсуждается в двух документах, при этом используется совершенно разная лексика, они будут находиться далеко друг от друга в тематическом пространстве. Исправить это можно сохранением слова со словами, имеющими схожее значение, — метод word2vec, который мы обсудим далее в этой главе.

И здесь возникает ряд вопросов.

Вопрос 1.

Что произойдет, если добавить в корпус еще один документ? К счастью, для получения тематического вектора документа не нужно обрабатывать заново весь корпус — мы просто проецируем его на существующее тематическое пространство корпуса. Но, конечно, это не работает при добавлении нового документа, не имеющего ничего общего с нашим корпусом, например добавление статьи по чистой математике в корпус любовных сонетов Шекспира. В этом случае статья по математике будет представлена кучей нулей или близких к нулю записей, что не позволит правильно отразить идеи статьи.

Вопрос 2.

Что означает матрица $\mathbf{V}^T$ в сингулярном разложении $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ с точки зрения обработки естественного языка нашего корпуса? Матрица $\mathbf{V}^T$ имеет количество строк и столбцов, равное количеству документов в корпусе. Она представляет собой матрицу "документ — документ" и дает значение, общее для документов.

Вопрос 3.

Сохранятся ли большие расстояния между документами при переходе к тематическому пространству более низкой размерности с помощью латентно-семантического анализа? Да, поскольку сингулярное разложение нацелено на *максимизацию* дисперсии всех документов в корпусе.

Вопрос 4.

Сохранятся ли малые расстояния, т. е. сохранит ли латентно-семантический анализ *тонкую структуру* документа, которая отличает его от других, *не слишком отличающихся* документов? Нет. Латентное размещение Дирихле, о котором пойдет речь ниже, справляется с этой задачей лучше.

Вопрос 5.

Можно ли усовершенствовать латентно-семантический анализ, чтобы он также удерживал близкие векторы документов вместе в тематическом пространстве низкой размерности? Да. Можно направлять векторы с помощью дополнительной информации, или *метаданных* документов (например, сообщения с одним и тем же отправителем или наложение штрафа с помощью функции стоимости), так, чтобы метод выводил тематические векторы, сохраняющие, в том числе, *близость*.

Подводя итог, можно сказать, что латентно-семантический анализ оптимально отбирает тематики, максимизируя их разнообразие во всем корпусе. Матрица **U**, полученная в результате сингулярного разложения матрицы TF-IDF, играет крайне важную роль. Она возвращает направления, вдоль которых дисперсия будет максимальной. Как правило, мы избавляемся от тематик с наименьшей дисперсией, отбрасывая последние столбцы **U**. Это похоже на то, как мы вручную избавляемся от *стоп-слов* (или *шумовых слов*) при подготовке текста, причем латентно-семантический анализ выполняет это самым оптимальным способом. Матрица **U** имеет столько же строк и столбцов, сколько и наш словарный запас. Такая кросс-корреляция между словами и тематиками основана на совместной встречаемости слов в одном и том же документе. При умножении нового документа на **U** (его проекции на столбцы **U**) мы получаем количество каждой тематики в документе. Мы можем обрезать матрицу **U** как угодно и отбросить менее важные тематики, сократив размерность до нужного нам количества тематик.

Недостатки латентно-семантического анализа

Создаваемые им тематические пространства, или столбцы **U**, — это обычные *линейные комбинации* токенов, сведенные так, чтобы они охватывали как можно больше различий в употреблении токенов из словаря. Это необязательно приводит к образованию словосочетаний, хоть как-то значимых для человека. Обидно. Такие недостатки можно устранить с помощью word2vec, о чем мы поговорим ниже.

Наконец, тематические векторы, полученные с помощью латентно-семантического анализа, — это простые линейные преобразования, выполненные над векторами TF-IDF. Без них не обойтись в семантическом поиске, кластеризации документов и рекомендательных движках на основе контента. Все это можно получить, измерив расстояния между тематическими векторами, о чем мы поговорим далее в этой главе.

Тематическое векторное представление документа, определенное с помощью латентного размещения Дирихле

В отличие от тематических векторов, полученных в результате латентно-семантического анализа, при *латентном размещении Дирихле* (latent Dirichlet allocation, LDA; **https://oreil.ly/QsSPp**) для того, чтобы получить тематический вектор, приходится обрабатывать заново весь корпус с добавлением каждого нового документа. Более того, для объединения слов в тематики здесь применяется нелинейный статистический подход, опирающийся на распределение Дирихле для частоты слов. В результате метод становится еще точнее, чем латентно-семантический анализ, в плане статистики распределения слов по тематикам. Таким образом, можно сказать, что это алгоритм распределения слов по тематикам, основанный на их совместном появлении в документе, и что для нас имеет значение, каким способом тематики распределяются по документам.

Для обучения этому нелинейному методу потребуется больше времени, чем для обучения линейному латентно-семантическому анализу. Поэтому он непрактичен для приложений, связанных с корпусами документов, даже несмотря на объяснимость. Но зато его можно использовать для обобщения отдельных документов, где каждое предложение становится собственным *документом*, а материнский документ — корпусом.

Метод латентного размещения Дирихле был впервые представлен генетиками в 2000 году для определения состава населения, а в 2003 году он был применен в обработке естественного языка. Приведем его исходные положения.

- Начинаем с грубого подсчета слов (а не с нормализованных векторов TF-IDF), причем последовательность слов для передачи их смысла пока не создается. Здесь мы все еще опираемся на моделирование статистики слов для каждого документа, но на этот раз включаем распределение слов в модель в явном виде.
- Документ — это линейная комбинация произвольного числа тематик (необходимо задать это число заранее, чтобы метод распределил по нему токены документа).
- Каждая тематика может быть представлена определенным распределением слов на основе частоты термина.
- Вероятность появления конкретной тематики в документе соответствует распределению вероятности Дирихле.
- Вероятность того, что конкретное слово будет отнесено к теме, также определяется распределением вероятности Дирихле.

В результате *тематические векторы*, полученные методом размещения Дирихле, являются разреженными, что указывает на четкое разделение между тематиками в плане того, какие объясняющие их слова в них содержатся.

В размещении Дирихле слова, которые часто встречаются вместе, приписываются одним и тем же тематикам. Таким образом, при переходе в тематическое пространство более низкой размерности этот метод сохраняет близкие друг у другу токены.

Латентно-семантический анализ, в свою очередь, при таком переходе сохраняет разнесенные друг от друга токены, следовательно, он лучше подходит для задач классификации, где разделение на классы сохраняется даже при переходе к более низкоразмерному пространству.

Тематическое векторное представление документа, определенное с помощью латентного дискриминантного анализа

В отличие от латентно-семантического анализа и латентного размещения Дирихле, которые разбивают документ на нужное нам количество тематик, *латентный дискриминантный анализ* разбивает документ *только на одну тематику*, например спам, настроение и т. д. Это отлично пригодится в бинарной классификации, например в классификации сообщений на спам и не спам, или классификации отзывов на положительные и отрицательные. Вместо максимизации разделения всех векторов в новом тематическом пространстве с помощью латентно-семантического анализа, можно применить латентный дискриминантный анализ и разделить только центроиды векторов, принадлежащих каждой категории.

Но как определить вектор, представляющий одну тематику? В случае когда документ размечен TF-IDF векторами как спам и не спам, мы вычисляем центроид каждой категории, в результате чего вектор проходит по линии, соединяющей два центроида (рис. 7.1).

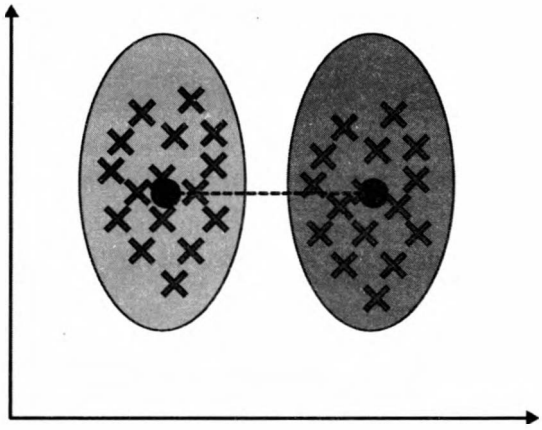


Рис. 7.1. Латентный дискриминантный анализ

Теперь каждый новый документ можно спроецировать на это одно измерение. Координата документа вдоль этой линии — это точечное произведение его TF-IDF на вектор направления линии центроидов. Весь документ (с миллионами измерений) теперь сжат в одно число в одном измерении (вдоль одной оси) с двумя центроидами и средними точками. Можно классифицировать документ по принадлежности к той или иной категории в зависимости от его расстояния до каждого центроида вдоль этой линии. Важно отметить, что граница принятия решения для разделения на категории с помощью этого метода является линейной.

Смысловые векторные представления слов и документов, определяемые встроенными нейросетями

Предыдущие модели векторизации документов на естественном языке учитывали только линейные связи между словами, а применяя латентное размещение Дирихле, нам пришлось положиться на человеческое суждение при выборе параметров модели и выделении признаков. Мы знаем, что сила нейросетей заключается в их способности улавливать нелинейные связи, извлекать признаки и автоматически находить подходящие параметры модели. Теперь мы будем использовать нейросети для создания векторов, представляющих отдельные слова и термины, а также применим аналогичные методы, чтобы получить векторы, представляющие значения целых абзацев. Поскольку в этих векторах закодировано как значение, так и логическое и контекстуальное использование каждого термина, с их помощью можно рассуждать, выполняя обычные операции сложения и вычитания векторов.

Векторное представление word2vec отдельных терминов с учетом атрибутов непрерывности

Используя векторы TF или TF-IDF в качестве отправной точки для тематических векторов, мы игнорируем ближайший контекст слов и его влияние на их значение. Векторы слов решают эту проблему. Вектор слова — это числовое векторное представление значения слова, так что каждый термин в корпусе становится семантическим вектором. Такое векторное представление отдельных слов числами с плавающей точкой позволяет делать семантические запросы и логические рассуждения.

Векторные представления слов обучаются с помощью нейросети. Обычно они имеют от 100 до 500 измерений, кодирующих количество каждого смыслового измерения, которое несет в себе слово. При обучении векторной модели слов текстовые данные не размечены. После обучения два термина будут определяться как близкие или далекие друг от друга по значению путем сравнения их векторов с помощью определенных метрик близости. Самой распространенной из них считается косинусное сходство, о котором пойдет речь далее.

В 2013 году компания Google представила алгоритм word2vec, обученный на новостной ленте Google, содержащей 100 млрд слов. Итоговая предобученная модель word2vec содержит 300 размерных векторов для 3 млн слов и фраз. Ее можно бесплатно скачать на странице архива кода Google для word2vec (**https://oreil.ly/qvGQo**).

Вектор, построенный с помощью word2vec, отражает гораздо больше смысла слова, чем тематические векторы, рассмотренные ранее в этой главе. Весьма примечательна в этом плане аннотация к статье "Эффективная оценка представлений слов в векторном пространстве" (Миколов Т. и др., 2013; **https://oreil.ly/lGT6i**):

> "Предлагаем вашему вниманию две новейшие архитектуры моделей для вычисления непрерывных векторных представлений слов из очень больших наборов данных. Качество этих представлений измеряется в задаче сходства слов, а результаты сравниваются с предыдущими наиболее эффективными методами, основанными на различных типах нейросетей. Мы показываем значительное повышение точности при

гораздо меньших вычислительных затратах, т. е. на обучение высококачественных векторов слов из набора данных объемом 1,6 млрд слов уходит менее одного дня. Кроме того, мы демонстрируем высочайшую эффективность этих векторов на тестовом наборе для измерения синтаксического и семантического сходства слов".

Месяц спустя в статье "Распределенные представления слов и фраз и их композиционность" (Миколов Т. и др., 2013; **https://oreil.ly/RyijL**) рассматривалось представление словосочетаний, которые означают нечто иное, чем их отдельные компоненты, например "Air Canada":

"Предложенная недавно модель непрерывных скип-грамм является эффективным методом обучения высококачественных распределенных векторных представлений, которые отражают большое количество точных синтаксических и семантических связей между словами. В этой статье мы представляем несколько расширений, которые повышают как качество векторов, так и скорость обучения. За счет подвыборки частых слов мы обеспечиваем значительное ускорение, а также обучение более регулярным представлениям слов. Мы также описываем простую альтернативу иерархическому softmax — отрицательную выборку. К непреодолимым ограничениям представлений слов можно отнести их индифферентность к порядку слов и неспособность представлять идиоматические выражения. Например, чтобы получить верное значение „Air Canada", простой комбинации слов „Canada" и „Air" будет недостаточно. На этом примере мы демонстрируем простой метод поиска фраз в тексте и показываем возможность обучения эффективным векторным представлениям миллионов фраз".

В публикации "Лингвистические закономерности в представлениях слов в непрерывном пространстве" (Миколов Т. и др., 2013; **https://oreil.ly/vKzgZ**) рассматриваются представления word2vec и показано, как смысловые векторы кодируют логические закономерности для слов и как это позволяет нам отвечать на регулярно возникающие вопросы аналогии:

"Недавно языковые модели с непрерывным пространством показали превосходные результаты в различных задачах. В статье рассматриваются представления слов в векторном пространстве, которые в неявном виде обучаются с помощью весов входного слоя. Мы выяснили, что такие представления поразительно отражают синтаксические и семантические закономерности языка, и что каждое отношение характеризуется векторным смещением, специфичным для данного отношения. Таким образом, на основе смещений между словами можно строить векторно-ориентированные рассуждения. Например, модель автоматически обучается отношениям между мужчиной и женщиной, и с помощью индуцированных векторных представлений "король – мужчина + женщина" получается вектор, очень близкий к "королеве". Показано, что векторы слов улавливают синтаксические закономерности с помощью вопросов на синтаксическую аналогию (прилагаются к статье), верно ответив почти на 40% вопросов. На примере ответов на вопросы задания 2 SemEval-2012 продемонстрировано, как с помощью метода векторного смещения векторы слов улавливают семантические закономерности. Примечательно, что этот метод превосходит лучшие предыдущие системы".

Начиная с 2013 года производительность word2vec стала значительно улучшаться вследствие обучения на гораздо более объемных корпусах.

Метод word2vec берет одно слово и присваивает ему вектор атрибутов, например место, животное, город, позитивность (чувственность), яркость, пол и т. д. Каждый атрибут — это размерность, отражающая, сколько атрибутов содержит значение слова.

Векторы значений слов и атрибутов кодируются не вручную, а в процессе обучения, когда модель узнает значение слова из его окружения — пять или более близких слов в одном предложении. Этим метод отличается от латентно-семантического анализа, где тематики изучаются только по словам, появляющимся в одном документе, причем необязательно близко друг к другу. В приложениях, связанных с небольшими документами и высказываниями, вставки word2vec фактически заменили тематические векторы, полученные с помощью латентно-семантического анализа. Векторы слов можно также использовать для получения кластеров слов из огромных наборов данных с помощью кластеризации k средних поверх векторных представлений слов. Более подробную информацию можно найти на странице архива кода Google для word2vec (**https://oreil.ly/7tqd8**).

Преимущество представления слов в виде векторов, которые несут некоторый смысл (а не подсчитывают), заключается в том, что с их помощью можно рассуждать. Например, как уже говорилось, если мы вычтем вектор, представляющий "мужчину", из вектора, представляющего "короля", и добавим вектор, представляющий "женщину", то получим вектор, очень близкий к вектору, представляющему слово "королева". Другой пример — связь между словами в единственном и множественном числе. Если вычесть векторы, представляющие единственное число слов, из векторов, представляющих их множественные числа, то получатся векторы, примерно одинаковые для всех слов.

Здесь возникают вопросы: как вычислить вставки word2vec? То есть, как обучить модель word2vec? Что такое обучающие данные, архитектура нейросети, ее вход и выход? Модели word2vec обучаются на неглубоких нейросетях с одним скрытым слоем. На вход подается большой корпус текстов, и на выходе мы получаем векторы из нескольких сотен измерений, по одному на каждый уникальный термин в корпусе. Слова с общим лингвистическим контекстом имеют *близкие* векторы.

Существуют два алгоритма обучения word2vec, но здесь мы не будем подробно описывать их.

Тем не менее сегодня у нас есть четкое представление, как функционируют нейросети, особенно неглубокие с одним скрытым слоем. Приведем эти алгоритмы.

Непрерывный мешок слов.

Предсказывает текущее слово из окна слов окружающего контекста; порядок контекстных слов не влияет на предсказание.

Непрерывная скип-грамма.

Использует текущее слово для предсказания окна слов окружающего контекста; близлежащие контекстные слова алгоритм оценивает сильнее, чем более удаленные.

Оба алгоритма изучают векторное представление термина, подходящее для предсказания других терминов в предложении. Очевидно, что непрерывный мешок слов быстрее, чем непрерывная скип-грамма, но скип-грамма лучше работает с редкими словами.

Более подробную информацию можно посмотреть в учебном пособии "Удивительная сила векторов слов" (Кольер Э., 2016; **https://oreil.ly/mDBZf**), на странице Википедии, посвященной word2vec (**https://oreil.ly/rwL93**), и в трех оригинальных статьях на эту тему: "Эффективная оценка представлений слов в векторном пространстве" (Миколов Т. и др., 2013; **https://oreil.ly/1PeLI**), "Распределенные представления слов и фраз и их композиционность" (Миколов Т. и др., 2013; **https://oreil.ly/XDZzr**) и "Лингвистические закономерности в представлении слов в непрерывном пространстве" (Миколов Т. и др., 2013; **https://oreil.ly/vKzgZ**).

Модель word2vec, обученная на новостях Google, содержит 3 млн слов, каждое из которых представлено вектором из 300 измерений. Для ее скачивания потребуется 3 Гбайт свободной памяти. Однако в случае ограниченного объема памяти или когда важна только часть слов, существуют способы обойтись без загрузки всей предобученной модели.

Визуализация векторов, представляющих слова

Векторы слов имеют крайне высокую размерность (100–500 измерений), но человек способен воспринимать только двух- и трехмерные векторы, поэтому необходимо спроецировать высокоразмерные векторы на эти пространства значительно меньшей размерности, сохранив при этом их самые важные характеристики. Мы уже знаем, что за нас это сделает сингулярное разложение (анализ главных компонент), выдав векторы, по которым можно проецировать в порядке убывания важности, или направления, вдоль которых данная коллекция слов-векторов изменяется больше всего. То есть сингулярное разложение гарантирует, что эта проекция даст наилучшее представление векторов слов, располагая их как можно дальше друг от друга.

В сети можно найти множество замечательных примеров. В публикации "Векторы распределения слово-тема в видеоколлекции MOOC (массовый открытый онлайн-курс)" (Кастрати З. и др., 2020; **https://oreil.ly/ktDdy**) авторы пользуются набором данных из сферы образования с транскриптами 12 032 видеолекций 200 курсов, собранных на платформе Coursera (**https://www.coursera.org/**), для генерации векторов слов с помощью модели word2vec и *тематических векторов документов* с *помощью латентного размещения Дирихле*. Набор данных содержит 878 тыс. предложений и более 79 млн токенов. Объем словарного запаса составляет более 68 тыс. уникальных слов. Каждый видеотранскрипт имеет свою длину в диапазоне от 228 до 32 767 токенов, в среднем 6622 токена на видеотранскрипт. Авторы использовали реализацию word2vec и латентного размещения Дирихле в пакете Gensim для Python. На рис. 7.2 показана трехмерная визуализация подмножества векторов слов с помощью анализа главных компонент.

Важно запомнить, что векторы слов и тематические векторы документов не являются самоцелью. Они — средство достижения цели, которая часто представлена как задача обработки естественного языка, например классификация в определенных предметных областях (как массовые открытые онлайн-курсы из примера), сравнительный анализ и анализ производительности существующих и новых моделей, трансферное обучение, рекомендательные системы, контекстный анализ, насыщение короткого текста тематикой, персональное обучение, организация удобного для поиска и максимально наглядного контента. Мы рассмотрим эти задачи в ближайшее время.

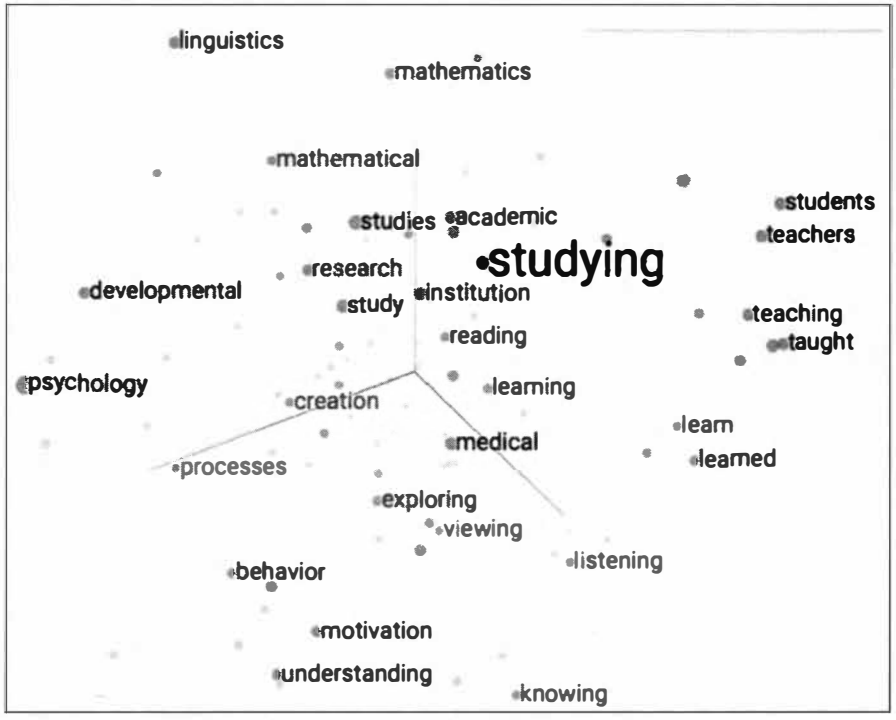


Рис. 7.2. Трехмерная визуализация векторов слов с помощью первых трех главных компонент. В данном примере выделен вектор, представляющий слово *studying* (учеба) и соседних с ним слов: *academic* (академик), *studies* (учеба), *institution* (учебное заведение), *reading* (чтение) и т. д. (источник изображения: **https://oreil.ly/Pj0GF**)

Векторное представление fastText от Facebook числовой характеристики отдельной *n*-граммы

Программа fastText платформы Facebook похожа на word2vec, только вместо представления полных слов или *n*-грамм в виде векторов она обучена выдавать векторное представление для каждой *характеристики n*-граммы. Благодаря этому fastText способна работать с редкими, неправильно написанными и даже неполными словами, которые часто встречаются в сообщениях социальных сетей. Во время обуче-

ния алгоритм скип-граммы word2vec тренируется предсказывать окружающий контекст данного слова. Аналогичным образом алгоритм fastText для числовых характеристик *n*-грамм обучается предсказывать характеристики *n*-грамм окружающего слова, обеспечивая большую детализацию и гибкость. Например, он будет выдавать не целое слово "чудесный", но представит его в виде векторов 2- и 3-грамм: "чу", "чуд", "уд", "уде", "де", "дес", "ес", "есн", "сн", "сны", "ны", "ный" и "ый".

Facebook выпустила свои предобученные модели fastText для 294 языков, обученные на доступных коллекциях Википедии для этих языков. Они включают огромный диапазон — от абхазского до зулу, а также редкие языки, на которых говорит всего несколько человек. Безусловно, эти модели имеют разную точность в зависимости от языка, а также от доступности и качества обучающих данных.

Векторное представление документа doc2vec или par2vec

А как насчет семантического представления документов? В предыдущих разделах нам удалось представить целые документы в виде тематических векторов, но word2vec дает векторное представление отдельных слов или фраз. Возможно ли расширить модель word2vec так, чтобы представлять целые документы как несущие смысл векторы? В статье "Распределенные представления предложений и документов" (Ле и др., 2014; **https://oreil.ly/Y1NzF**) говорится именно об этом: алгоритм неконтролируемо обучает плотные векторы фиксированной длины из фрагментов текста переменной длины, таких как предложения, абзацы и документы. В "Учебнике Doc2Vec с использованием Gensim" (Клинтберг А., 2015; **https://oreil.ly/tWRcj**) рассматривается процесс реализации на Python, в результате которого каждый полный документ в заданном корпусе получает вектор фиксированного размера.

Глобальный вектор или векторное представление слов

Существуют и другие способы создания векторов, представляющих значения слов. Глобальный вектор, или GloVe (2014, **https://oreil.ly/8P32U**), — это модель, создающая такие векторы методом сингулярного разложения. Она обучается только на ненулевых записях глобальной матрицы совместной встречаемости слов, содержащей данные о частоте встречаемости слов друг с другом по всему корпусу.

GloVe — это, по сути, *лог-билинейная модель*[2] (log-bilinear model, LBL model) со взвешенной задачей наименьших квадратов. Лог-билинейную модель можно отнести, пожалуй, к самым простым нейронным языковым моделям. Учитывая предшествующие $n - 1$ слов, эта модель вычисляет начальное векторное представление для следующего слова, выполняя обычную линейную комбинацию векторных представлений этих $n - 1$ слов. Затем вычисляется вероятность появления следующего слова с учетом $n - 1$ предшествующих слов на основе сходства (скалярное произве-

[2] Такие модели называются лог-билинейными (или log-билинейными, или билинейными логарифмическими, или билинейно-логарифмическими, или логарифмически-билинейными) потому, что функция становится линейной по векторам слов и контекстов после логарифмирования. — *Прим. ред.*

дение) между векторным представлением линейной комбинации и представлениями всех слов в словаре:

$$\text{Prob}\left(w_n = w \middle| w_1, w_2, ..., w_{n-1}\right) = \\ = \frac{\exp\left(w^T_{\text{словарь}_1} w\right)}{\exp\left(w^T_{\text{словарь}_1} w\right) + \exp\left(w^T_{\text{словарь}_2} w\right) + ... + \exp\left(w^T_{\text{словарь}_{\text{словарь}_{\text{размер}}}} w\right)}.$$

Базовая интуиция, на которую опирается модель "глобальный вектор", заключается в простом наблюдении, что соотношения вероятностей совпадения слов потенциально кодируют определенный смысл. В примере на сайте проекта GloVe рассматриваются вероятности совпадения целевых слов *ice* (лед) и *steam* (пар) с различными проверочными словами из словаря. В табл. 7.1 приведены фактические вероятности по корпусу, состоящему из 6 млрд слов.

Таблица 7.1. *Таблица вероятностей появления слов "лед" и "пар" вместе со словами "твердый", "газ", "вода" и "шаблон"*

Вероятность и отношения	*k* = твердый	*k* = газ	*k* = вода	*k* = шаблон
$P(k \mid \text{лед})$	$1{,}9 \times 10^{-4}$	$6{,}6 \times 10^{-5}$	$3{,}0 \times 10^{-3}$	$1{,}7 \times 10^{-5}$
$P(k \mid \text{пар})$	$2{,}2 \times 10^{-5}$	$7{,}8 \times 10^{-4}$	$2{,}2 \times 10^{-3}$	$1{,}8 \times 10^{-5}$
$P(k \mid \text{лед})/P(k \mid \text{пар})$	8,9	$8{,}5 \times 10^{-2}$	1,36	0,96

Из табл. 7.1 очевидно, что, как и предполагалось, слово "лед" чаще появляется вместе со словом "твердый", чем со словом "газ", а слово "пар" — наоборот. Слова "лед" и "пар" чаще встречаются с общим для них словом "вода", чем с не связанным с ними словом "шаблон". Вычисляя отношение вероятностей, можно устранить шум от недискриминационных слов, например "вода", так что значения намного больше 1 хорошо коррелируют со свойствами, характерными для льда, а значения намного меньше 1 хорошо коррелируют со свойствами, характерными для пара. Таким образом, отношение вероятностей кодирует некую приближенную форму смысла, соответствующую абстрактному понятию термодинамической фазы.

Цель обучения GloVe — обучиться векторам слов так, чтобы их точечное произведение равнялось логарифму вероятности совместного появления слов. Так как логарифм соотношения равен разности логарифмов, в векторном пространстве слов учитывается разность векторов. Поскольку в таких соотношениях может содержаться определенный смысл, данная информация также кодируется в виде разности векторов. По этой причине полученные векторы слов отлично справляются с задачами аналогии слов, например с задачами из пакета word2vec.

Алгоритмы сингулярного разложения оптимизировались в течение десятилетий, так что в обучении GloVe имеет преимущество перед word2vec, являющейся нейросетью и опирающейся на градиентный спуск и обратное распространение при минимизации ошибок. Для обучения собственных векторов слов из нужного корпу-

са лучше воспользоваться моделью GloVe, а не word2vec, даже при том, что word2vec первой выполнила семантические и логические рассуждения со словами, т. к. GloVe обладает более высокой скоростью обучения, демонстрирует лучшую производительность оперативной памяти и процессора, дает более точные результаты, чем word2vec, даже по небольшим корпусам документов.

Косинусное сходство

До сих пор мы работали в этой главе только над одной задачей — преобразованием документа с текстом на естественном языке в числовой вектор. Документ может состоять из одного слова, одного предложения, абзаца, нескольких абзацев и больше. Мы выяснили несколько способов получения векторов, среди которых какие-то семантически репрезентативны для наших документов больше, чем остальные.

Получив векторное представление документа, можно использовать его в таких моделях машинного обучения, как алгоритмы классификации, кластеризации и др. Например, кластеризация векторов документов в корпусе с помощью алгоритма кластеризации *k средних* для создания классификатора документов. Мы также можем определить *семантическое сходство* документа с остальными документами в поисковых системах, информационно-поисковых системах и в ряде других приложений.

Мы выяснили, что вследствие проклятия размерности не имеет смысла измерять евклидово расстояние между двумя векторами документов крайне высокой размерности, т. к. они окажутся достаточно далеки друг от друга только в силу обширного пространства, в котором они обитают. Так как же определить, насколько *близкими* или *далекими*, *похожими* или *разными* будут векторы, представляющие документы? Один из успешных проверенных способов — косинусное сходство, который заключается в измерении косинуса угла между двумя векторами документов. Для этого нужно воспользоваться скалярным произведением векторов, каждый из которых нормализован на свою длину (предварительная нормализация векторов документов уже дала бы длину, равную единице):

$$\cos\left(\text{угол между } \overrightarrow{doc_1} \text{ и } \overrightarrow{doc_2}\right) = \frac{\overrightarrow{doc_1}^T \overrightarrow{doc_2}}{\text{длина}\left(\overrightarrow{doc_1}\right) \text{длина}\left(\overrightarrow{doc_2}\right)}.$$

На рис. 7.3 показаны три документа, представленные в двумерном векторном пространстве.

Нас интересуют углы между ними.

Косинус угла — это всегда число в диапазоне от –1 до 1. Если два вектора документов идеально выровнены и направлены в одну сторону по всем измерениям, то их косинусное сходство равно 1; если они полностью противоположны друг другу по всем измерениям, то их косинусное сходство равно –1; если они ортогональны друг другу, то их косинусное сходство равно 0.

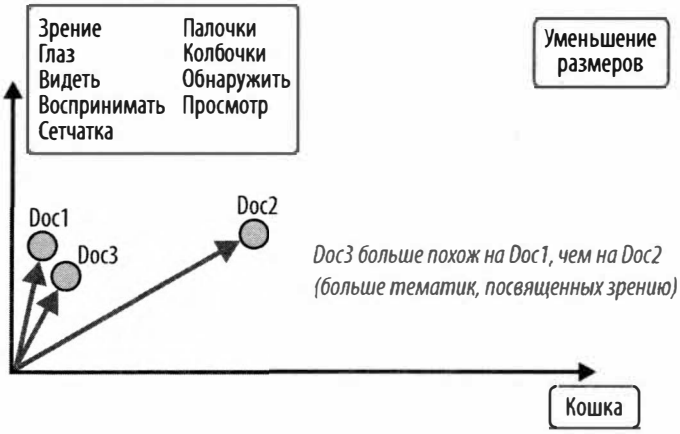


Рис. 7.3. Три документа, представленные в двумерном векторном пространстве

Приложения для обработки естественного языка

Бóльшая часть этой главы была посвящена преобразованию заданного документа с текстом на естественном языке в вектор чисел. Мы выяснили, что существует множество способов получения векторов документов, каждый из которых приводит к различным представлениям (и, следовательно, выводам) или подчеркивает определенные аспекты данных естественного языка по сравнению с другими. Для людей, приходящих в область ИИ, связанную с обработкой естественного языка, это один из самых сложных барьеров, особенно если они имеют количественное образование, где сущности, с которыми они работают, являются числовыми, созревшими для математического моделирования и анализа. Теперь, когда мы преодолели этот барьер, вооружившись конкретными векторными представлениями для данных естественного языка, мы можем математически осмыслить популярные приложения. Важно понимать, что существует несколько способов выполнения каждого из перечисленных ниже действий. Традиционные подходы — это жестко закодированные правила, присваивающие баллы словам, пунктуациям, эмодзи и т. д., а затем полагающиеся на их наличие в выборке данных для получения результата. Современные подходы опираются на различные модели машинного обучения, которые, в свою очередь, основаны на (как правило) размеченных наборах обучающих данных. Для того чтобы преуспеть в этой области, необходимо время, чтобы попробовать различные модели на одной и той же задаче, сравнить производительность и досконально понять каждую модель с ее сильными и слабыми сторонами, а также математическим обоснованием ее успехов и неудач.

Анализ настроений

Приведем самые распространенные подходы к выявлению настроений из текста на естественном языке.

Жестко закодированные правила.

VADER (Valence Aware Dictionary and sEntiment Reasoner) — успешный алгоритм, в котором токенизатор должен тщательно обработать знаки препинания и эмодзи, передающие большое количество настроений. Однако здесь нам придется вручную компилировать тысячи слов с указанием оценки настроения, в то время как машина делает это автоматически.

*Наивный байесовский классификатор (**https://oreil.ly/uzJDd**).*

Это набор классифицирующих алгоритмов, основанных на теореме Байеса из теории вероятностей — правиле принятия решений для классификации по максимальному правдоподобию. Мы рассмотрим это в *главе 11*.

Латентный дискриминантный анализ.

В предыдущем разделе мы узнали, как классифицировать документы на два класса с помощью латентного дискриминантного анализа. Вкратце напомним, что мы начинаем с данных, размеченных на два класса, затем вычисляем центроид каждого класса и находим соединяющее их направление. Проецируем каждый новый экземпляр данных вдоль этого направления и классифицируем его в зависимости от близости к тому или иному центроиду.

Латентно-семантический анализ.

Кластеры векторов документов, сформированные с помощью латентно-семантического анализа, могут быть использованы для классификации. В идеале в тематических пространствах латентно-семантического анализа положительные отзывы группируются на расстоянии от отрицательных. Имея набор отзывов, размеченных как положительные или отрицательные, сначала вычисляем их тематические векторы с помощью латентно-семантического анализа. Для того чтобы классифицировать новый отзыв, вычисляем тематический вектор, а затем косинусное сходство с положительным и отрицательным тематическими векторами. Теперь классифицируем отзыв как положительный, если он больше похож на положительный тематический вектор, и как отрицательный, если он больше похож на отрицательный тематический вектор.

Трансформеры, сверточная нейросеть, рекуррентная нейросеть с долгой краткосрочной памятью.

Все эти современные методы машинного обучения требуют передачи документа в векторной форме в нейросеть с определенной архитектурой. Мы рассмотрим их в ближайшее время.

Фильтрация спама

С точки зрения математики, фильтрация спама представляет собой задачу классификации, схожую с рассмотренным ранее анализом настроения, когда настроение документа является либо положительным, либо отрицательным.

Таким образом, к фильтрации спама применимы те же методы классификации настроений. В любом случае независимо от того, как созданы векторы документов, их

можно использовать для предсказания того, является ли пост в соцсети спамом или нет, предсказания вероятности получения лайков и т. д.

Поиск и извлечение информации

И снова, независимо от того, как мы получили числовые векторы, представляющие документы, их можно использовать в задачах поиска и извлечения информации. Поиск может опираться на индекс или семантику.

Полнотекстовый поиск.

Поиск документа по слову или части слова, которое он содержит. Поисковые системы разбивают документы на слова, которые могут быть проиндексированы, аналогично указателям в конце учебной литературы. Безусловно, для выявления орфографических ошибок и опечаток потребуется огромное количество отслеживаний, а порой и угадываний. С этим неплохо могут справиться индексы (в случае их наличия).

Семантический поиск.

В этом случае при поиске документов учитывается значение слов как в запросе, так и в запрашиваемых документах.

Далее приведем общие подходы к поиску и извлечению информации.

На основе косинусного сходства между TF-IDF документов.

Хорошо подойдет для корпусов, содержащих миллиарды документов. Любая поисковая система с миллисекундным временем отклика использует базовую матрицу TF-IDF.

На основе семантики.

Косинусное сходство между тематическими векторами документов, полученными путем латентно-семантического анализа (в корпусах, содержащих миллионы документов) или латентного размещения Дирихле (в гораздо меньших корпусах). Это похоже на классификацию сообщений на спам и не спам при использовании латентно-семантического анализа, за исключением того, что теперь мы вычисляем косинусное сходство между тематическим вектором нового документа и *всеми* тематическими векторами базы данных, возвращая наиболее похожие на наш документ.

На основе итерации собственных векторов.

Связан с алгоритмами ранжирования результатов поиска, например алгоритм PageRank (мы рассмотрим его в *разд. "Пример: алгоритм PageRank" главы 9*). Приведем довольно примечательную выдержку из статьи "Роль алгоритмов ранжирования в информационном поиске" (Чоудхари и Бурдак, 2012; **https://oreil.ly/jEbMX**):

"В поисковой системе есть три важных компонента. Это краулер, индексатор и алгоритм ранжирования. Краулер (также называется роботом или пауком) путешествует по сети и загружает веб-страницы. Загруженные страницы от-

правляются в модуль индексации, который анализирует их и строит индекс на основе ключевых слов в тексте. Как правило, индекс опирается на ключевые слова. Когда в интерфейс поисковой системы вводится поиск по ключевым словам, компонент обработки запросов сопоставляет ключевые слова запроса с индексом и возвращает URL. А в результате работы алгоритма ранжирования наиболее релевантные страницы появляются вверху, а менее релевантные — внизу поисковой страницы".

Семантический поиск и запросы с помощью векторов слов (word2vec или GloVe).

Рассмотрим такой поиск на примере из книги "Обработка естественного языка в действии"[3], где он сформулирован таким образом: "Она изобрела что-то связанное с физикой в Европе в начале XX века". Когда мы вводим это поисковое предложение в Google или Bing, прямого ответа "Мария Кюри", возможно, не будет. Скорее всего, поисковая система Google выдаст нам только ссылки на списки известных физиков, как мужчин, так и женщин. Просмотрев несколько страниц, мы найдем ответ — Мария Кюри. Google примет это к сведению и в следующий раз уточнит результаты поиска. Теперь, используя векторы слов, можно выполнить простую арифметику над векторами слов, представляющими женщина+Европа+физик+ученый+знаменитость, после чего мы получим новый вектор, близкий по косинусному сходству к вектору, обозначающему Марию Кюри, и, как говорят французы, вуаля! Ответ получен. Мы даже можем вычесть из векторов слов гендерную предвзятость в естественных науках простым вычитанием вектора, обозначающего мужчину, самца и т. д., так что можно искать вектор слов, наиболее близкий к варианту женщина+Европа+физика+ученый-мужчина-2*мужчина.

Поиск на основе запросов по аналогии.

Для вычисления поискового запроса типа "Они — то же самое в музыке, что Мария Кюри — в науке", нам достаточно выполнить простую векторную арифметику над векторами слов, представляющих Мария Кюри-наука+музыка.

Для иллюстрации индексирования и семантического поиска приведем выдержку из книги "Обработка естественного языка в действии":

"Традиционные подходы к индексированию используют двоичные векторы появления слов, дискретные векторы (мешок слов), разреженные векторы чисел с плавающей точкой (TF-IDF), низкоразмерные векторы чисел с плавающей точкой (например, трехмерные данные геоинформационных систем). Но такие высокоразмерные векторы с плавающей точкой, как тематические векторы из латентно-семантического анализа или латентного размещения Дирихле, представляют собой сложную задачу. Обратные индексы работают с дискретными или двоичными векторами, т. к. индекс должен содержать только запись для каждого ненулевого дискретного измерения. Значение этого измерения либо присутствует, либо отсутствует в ссылаемом векторе или документе. Поскольку TF-IDF — это разреженные, в основном нулевые векторы, нам не нужна запись в индексе для большинства измере-

[3] Lane H., Howard C., Hapke H. Natural Language Processing in Action. — Manning, 2019. — 544 p.

ний множества документов. В результате латентно-семантического анализа и латентного размещения Дирихле получаются непрерывные плотные тематические векторы высокой размерности, в которых практически не бывает нулей. Более того, алгоритм семантического анализа не позволяет получить эффективный индекс для масштабируемого поиска. Картину усугубляет проклятие размерности, не позволяющее добиться точности индекса. Одним из решений проблемы высокоразмерных векторов является их индексация с помощью локально-чувствительного хеширования, например почтового индекса, которым обозначена область гиперпространства. Такой хеш напоминает обычный хеш — он дискретен и зависит только от значений в векторе. Но даже это не будет идеально работать, как только мы превысим примерно 12 измерений. Точный семантический поиск не подойдет для большого корпуса, например для поиска в Google или даже для семантического поиска в Википедии. В случае векторов высокой размерности главное — это не стремиться к идеальному индексу или алгоритму скрытого хеширования, а *довольствоваться имеющимся*. В настоящее время есть несколько доступных реализаций некоторых эффективных и точных алгоритмов аппроксимации ближайших соседей, использующих для эффективной реализации семантического поиска скрытое семантическое хеширование. Технически эти решения индексирования или хеширования не могут гарантировать нахождения лучших соответствий нашему семантическому поисковому запросу. Но с их помощью, если мы готовы слегка пожертвовать точностью, можно получить подробный список близких совпадений почти так же оперативно, как обычный обратный индекс по вектору TF-IDF или вектору мешка слов. Модели нейросетей подстраивают концепции тематических векторов так, чтобы векторы, связанные со словами, были более точными и полезными, а значит, улучшали поиск".

Машинный перевод

Задача состоит в том, чтобы перевести последовательность токенов любой длины (например, предложение или абзац) в последовательность любой длины на другом языке. Архитектура "кодер — декодер", которая рассматривается в контексте трансформеров и рекуррентных нейронных сетей, хорошо зарекомендовала себя при решении задач перевода. Она отличается от архитектуры автокодировщика.

Создание подписей к изображениям

В этой технология компьютерное зрение сочетается с обработкой естественного языка.

Чат-боты

Это конечное применение обработки естественного языка. Для чат-бота требуется более одного вида обработки: разбор языка, поиск, анализ, генерация ответов, ответ на запросы и их выполнение. Кроме того, требуется база данных для хранения прошлых высказываний и ответов.

Другие приложения

В числе других областей применения можно отметить *распознавание именованных объектов* (**https://oreil.ly/5Mg9k**), *концептуальный фокус* (**https://oreil.ly/QYDOn**), извлечение релевантной информации из текста (например, дат), генерация языка. Мы рассмотрим их в *главе 8*.

Трансформеры и модели внимания

Трансформеры и модели внимания являются самыми современными для таких приложений обработки естественного языка, как машинный перевод, ответы на вопросы, генерация языка, распознавание именованных объектов, создание подписей к изображениям и чат-боты (по состоянию на 2022 год). В настоящее время они лежат в основе крупных языковых моделей: BERT (Bidirectional Encoder Representations from Transformers — двунаправленные кодирующие представления от трансформеров; **https://oreil.ly/sP7uM**) компании Google, а также GPT-2 (**https://oreil.ly/9rJQo**) и GPT-3 (**https://oreil.ly/NzDKo**) (Generative Pretrained Transformer — генеративный предобученный трансформер) компании OpenAI.

Трансформеры обходят рекуррентные и сверточные архитектуры, выступавшие в качестве основных архитектур в приложениях обработки естественного языка вплоть до 2017 года, когда в статье "Все, что вам нужно, — это внимание" (Васвани А. и др., 2017; **https://oreil.ly/AUAI9**) была представлена первая модель трансформера.

Архитектуры рекуррентных и сверточных нейросетей до сих пор используются (и хорошо работают) в некоторых приложениях для обработки естественного языка, а также в других приложениях, например финансовых. Подробнее об этих моделях мы поговорим далее в этой главе. Однако от них отказались в обработке естественного языка по следующим причинам.

- В случае коротких входных последовательностей токенов естественного языка слои внимания, задействованные в трансформационных моделях, работают быстрее, чем рекуррентные слои. Даже при длинных последовательностях можно модифицировать слои внимания так, чтобы они фокусировались только на определенных областях входного сигнала.
- Количество последовательных операций, необходимых рекуррентному слою, зависит от длины входной последовательности. Это количество сохраняется и для слоя внимания.
- В сверточных нейросетях ширина ядра напрямую влияет на долгосрочные зависимости между парами входов и соответствующих выходов. Для отслеживания долгосрочных зависимостей требуются большие ядра или стеки сверточных слоев, что повышает вычислительные затраты использующей их модели естественного языка.

Архитектура трансформера

Трансформеры являются неотъемлемой частью огромных языковых моделей, таких как GPT-2, GPT-3, ChatGPT, Google BERT (которая обучает языковую модель, просматривая последовательные текстовые данные слева направо и справа налево) и трансформер By Дао (**https://oreil.ly/Submn**). Эти модели довольно массивны: GPT-2 имеет около 1,5 млрд параметров, обученных на миллионах документов, взятых с 8 млн веб-сайтов со всего Интернета. GPT-3 имеет 175 млрд параметров, обученных на еще большем наборе данных. Трансформер By Дао имеет 1,75 трлн параметров, что требует огромного количества вычислительных ресурсов для обучения и вывода.

Изначально трансформеры предназначались для задач перевода, поэтому они имеют *структуру кодера-декодера*. На рис. 7.4 показана архитектура модели трансформера, первоначально представленная в статье "Все, что вам нужно, — это внимание" (Васвани А. и др., 2017; **https://oreil.ly/S3vEz**). Однако каждый кодер и декодер — это отдельный модуль, поэтому их можно использовать отдельно для выполнения различных задач. Например, можно использовать только один кодер для выполнения задачи классификации, скажем *частеречная разметка* или *тегирование частей речи*, т. е. после ввода предложения "Я люблю готовить на своей кухне" на выходе мы получаем класс для каждого слова: "я" — местоимение; "люблю" — глагол; "готовить" — глагол и т. д.

На вход модели полного трансформера (с кодером и декодером) подается последовательность токенов естественного языка произвольной длины, например вопрос чат-боту, абзац на английском языке, требующий перевода на французский, или краткое изложение в виде заголовка. На выходе — другая последовательность токенов естественного языка, также произвольной длины, например ответ чат-бота, переведенный абзац на французский или заголовок.

Но не нужно путать этап обучения с этапом вывода модели.

На этапе обучения.

> Модель получает данные и метки, например английское предложение (выборка входных данных) и его французский перевод (метка), и обучается отображению входа на целевую метку, что, как ожидается, хорошо обобщается на словари и грамматику обоих языков.

На этапе вывода.

> Модель получает только английское предложение и выводит французский перевод. Трансформеры выводят французское предложение по одному новому токену за раз.

На рис. 7.4 показано, как кодер, расположенный в левой части архитектуры трансформера, получает на вход токены, например английское предложение *"How was your day?"* ("Как у тебя прошел день?), и создает несколько числовых векторных представлений для каждого токена этого предложения, а также их линейные преобразования в новые направления, отражающие выравнивание с другими векторами слов, кодируя контекстуальную информацию токена в предложении. Причиной ли-

нейных преобразований (умножения векторов слов на весовые матрицы) служит стремление кодера узнать важные направления, вдоль которых векторы конкретных слов в предложении выравниваются с другими (контекстуальное выравнивание).

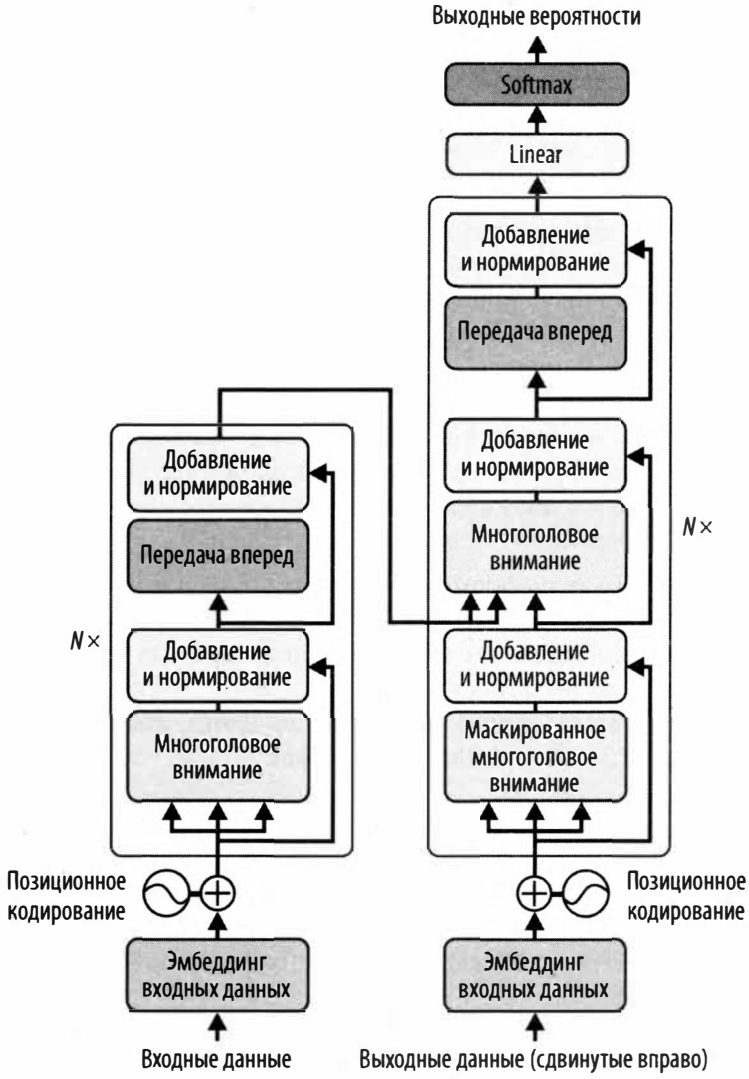


Рис. 7.4. Простая архитектура кодера-декодера в модели трансформатора (источник изображения: **https://oreil.ly/q1FOp**)

Декодер, расположенный в правой части архитектуры на рис. 7.4, получает эти векторы на входе вместе со своим выходом с предыдущего временно́го этапа. В итоге он генерирует на выходе токены, в нашем случае — французский перевод входного предложения *"Comme se passe ta journée"* (см. рис. 7.5). Фактически декодер вычисляет *вероятность* каждого слова во французском словаре (скажем, 50 000 токенов) с помощью функции softmax, а затем с наибольшей вероятностью выдает токен. Но

вычисление softmax для словаря такой высокой размерности обойдется слишком дорого, поэтому декодер использует *выборочный softmax*, вычисляющий вероятность для каждого токена в случайной выборке французского словаря на каждом шаге. На этапе обучения он должен включить в эту выборку целевой токен, но на этапе вывода целевой токен отсутствует.

Для фиксации долгосрочных зависимостей в последовательностях токенов трансформеры используют процесс, который называется *вниманием*. Слово "последовательность" может сбить с толку, особенно математиков, которые четко различают понятия *"последовательность"*, *"серия"*, *"вектор"* и *"список"*. Последовательности обрабатываются по одному термину за раз, т. е. обрабатывается один термин, далее следующий, затем следующий и так далее, пока не будет обработан весь входной сигнал. Трансформеры обрабатывают входные токены непоследовательно. Они обрабатывают их все вместе, параллельно. Это отличается от того, как рекуррентные нейронные сети обрабатывают входные токены, которые должны подаваться последовательно, что, по сути, запрещает параллельные вычисления. Пользуясь терминологией, предложение на естественном языке следует называть *вектором*, если мы обрабатываем его с помощью модели трансформера, или *матрицей*, т. к. каждое слово в предложении представлено в виде собственного вектора, или *тензором*, если мы обрабатываем партию предложений за раз, что позволяет архитектура трансформера. Если мы хотим обработать то же самое предложение с помощью модели рекуррентной нейросети, то мы должны называть его *последовательностью*, поскольку эта модель потребляет входные данные последовательно, по одному токену за раз. При обработке с помощью сверточной нейросети мы снова называем его вектором (или матрицей), поскольку сеть потребляет его целиком, а не разбивает по одному токену за раз.

Преимуществом является то, что модели не нужно последовательно обрабатывать входные данные, поскольку такие архитектуры допускают параллельную обработку. Однако, несмотря на то что благодаря параллелизации трансформеры становятся вычислительно эффективными, они не могут в полной мере использовать преимущества последовательной природы естественного языка и информации, закодированной в этой последовательности. Вспомним, как человек обрабатывает текст. Новые модели трансформеров пытаются выполнить то же самое.

Они работают следующим образом:

1. Представим каждое слово из входной последовательности в виде d-мерного вектора.
2. Включаем в модель порядок слов, добавив к вектору слова информацию о его положении (*позиционное кодирование*). Вносим позиционную информацию во входные данные, сопровождая каждый вектор каждого слова вектором позиционного кодирования той же длины. Векторы позиционного кодирования имеют ту же размерность, что и вставки векторов слов (это позволяет складывать оба вектора). Существует множество вариантов позиционного кодирования, часть из которых изучается в процессе обучения, другая часть фиксирована. В большинстве случаев применяются дискретизированные синусные и косинусные функции различной частоты.

3. Позиционно закодированные векторы слов подаются в блок кодера. Кодер обращает внимание на все слова во входной последовательности, независимо от того, предшествуют они рассматриваемому слову или следуют за ним, т. е. кодер трансформера является двунаправленным.
4. Декодер получает на вход собственное предсказанное выходное слово на временно́м шаге $t-1$, а также выходные векторы кодера.
5. Дополнительно входные данные декодера позиционно кодируются.
6. Полученный вход декодера подается на три подслоя. Декодер не воспринимает последующие слова, поэтому на его первом подслое применяется *маскирование*. На втором подслое декодер также получает выход кодера, который теперь позволяет воспринимать все слова во входной последовательности.
7. Выход декодера проходит через полносвязный слой, а затем чрез слой softmax, и предсказывает следующее слово выходной последовательности.

Механизм внимания

Чудесные свойства трансформера во многом обусловлены встроенными *механизмами внимания*. Механизм внимания включает в себя дополнительные бонусы.

Объяснимость.

Указывает, на какие части входного предложения (или документа) модель обратила внимание, генерируя определенный результат (рис. 7.5).

Использование предобученных моделей внимания.

Предобученные модели можно адаптировать к задачам, специфичным для конкретной области. То есть можно подстроить значения их параметров после дополнительного обучения на данных, специфичных для конкретной области.

Повышенная точность моделирования длинных предложений.

Другая ценная особенность механизмов внимания заключается в том, что они позволяют моделировать зависимости в последовательностях токенов естественного языка без учета расстояний между связанными токенами в этих последовательностях.

На рис. 7.5 показан механизм внимания в задаче перевода с английского на французский.

В механизме внимания нет сложной математики, нужно только выполнить масштабированное точечное произведение. Основная цель внимания — выделить наиболее значимые части входной последовательности и определить, насколько сильно они связаны друг с другом на входе, и как они влияют на определенные части выхода. Поскольку слова могут соотноситься друг с другом различными способами, вычисляем эти показатели выравнивания слов по нескольким направлениям.

Под *внутренним вниманием* подразумевается процесс, при котором последовательность векторов вычисляет выравнивание внутри собственных членов. Мы уже знаем, что скалярное произведение измеряет совместимость двух векторов. Можно вычислить простейшие веса внутреннего внимания, найдя скалярные произведения

всех членов последовательности векторов. Например, в предложении "Я люблю готовить на своей кухне" нужно вычислить все скалярные произведения векторов слов, представляющих слова "я", "люблю", "готовить", "на", "своей" и "кухне". Мы полагаем, что скалярное произведение слова "я" на "своей" будет самым большим, так же как и слова "готовить" на слово "кухня". Причем максимальным будет скалярное произведение слова "я" на "я", "люблю" на "люблю" и т. д., т. к. их векторы идеально выровнены между собой, хотя не содержат какой-либо ценной информации.

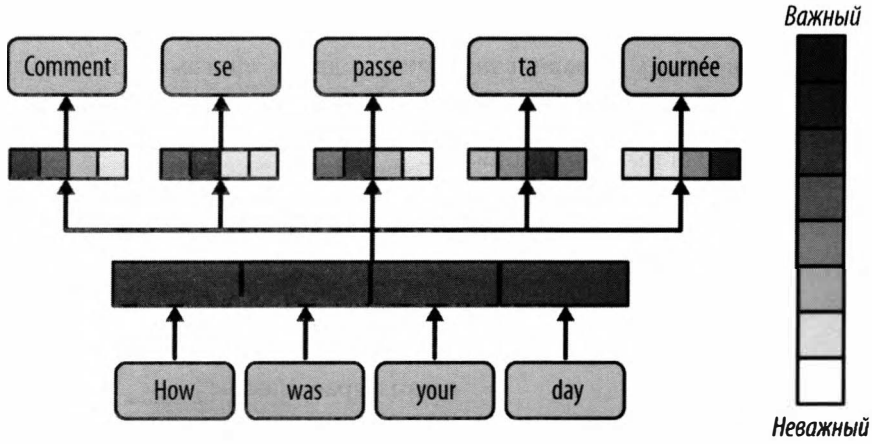


Рис. 7.5. Иллюстрация внимания в задаче перевода: веса, присвоенные входным токенам, показывают, на какие из них модель обратила больше внимания, прежде чем выдать каждый выходной токен (источник изображения: **https://oreil.ly/2SZCW**)

Решение трансформера, позволяющее избежать этих потерь, будет многогранным:

1. Применяем три различных линейных преобразования к каждому вектору входной последовательности (каждому слову предложения), умножая их на три различные весовые матрицы. В результате получаем три различных набора векторов, соответствующих каждому вектору $\vec{w}$ входного слова:
   - вектор *запроса* $\overrightarrow{\text{запрос}} = \mathbf{W}_q \vec{w}$ — вектор, *от которого выполняется отслеживание*;
   - вектор *ключа* $\overrightarrow{\text{ключ}} = \mathbf{W}_k \vec{w}$ — вектор, *к которому выполняется отслеживание*;
   - вектор *значения* $\overrightarrow{\text{значение}} = \mathbf{W}_v \vec{w}$ — для схватывания генерируемого контекста.
2. Получаем показатели выравнивания между векторами запроса и ключа для всех слов в предложении, выполнив их скалярное произведение, масштабированное на обратный квадратный корень длины этих векторов $\sqrt{l}$. Мы применяем такое масштабирование в целях численной устойчивости, чтобы скалярные произведения не становились большими. (Эти скалярные произведения вскоре будут переданы в функцию softmax. Поскольку при большой величине входных данных эта функция имеет слишком маленький градиент, компенсируем это, разделив каждое скалярное произведение на $\sqrt{l}$.) Более того, выравнивание двух век-

торов не зависит от их длин. Поэтому в нашем предложении показатель выравнивания между словами "готовить" и "кухня" будет равен:

$$\text{выравнивание}_{\text{готовить,кухня}} = \frac{1}{\sqrt{l}} \overrightarrow{\text{запрос}}^{T}_{\text{готовить}} \overrightarrow{\text{ключ}}_{\text{кухня}}.$$

Важно отметить, что этот показатель будет отличаться от показателя выравнивания слов "кухня" и "готовить", т. к. векторы запросов и ключей для каждого из них будут разными. Таким образом, полученная матрица выравнивания не является симметричной.

3. Преобразуем показатель выравнивания между двумя словами в предложении в вероятность, передав ее в функцию softmax. Например:

$$\begin{aligned}\omega_{\text{готовить,кухня}} &= \text{softmax}\left(\text{выравнивание}_{\text{готовить,кухня}}\right) = \\ &= \exp\left(\text{выравнивание}_{\text{готовить,кухня}}\right) \Big/ \Big(\Big\{\exp\left(\text{выравнивание}_{\text{готовить,я}}\right) + \\ &\quad + \exp\left(\text{выравнивание}_{\text{готовить,любить}}\right) + \\ &\quad + \exp\left(\text{выравнивание}_{\text{готовить,готовить}}\right) + \\ &\quad + \exp\left(\text{выравнивание}_{\text{готовить,на}}\right) + \\ &\quad + \exp\left(\text{выравнивание}_{\text{готовить,своей}}\right) + \\ &\quad + \exp\left(\text{выравнивание}_{\text{готовить,кухня}}\right)\Big\}\Big).\end{aligned}$$

4. Кодируем контекст каждого слова путем линейной комбинации векторов значений, используя вероятности выравнивания в качестве весов линейной комбинации. Например:

$$\begin{aligned}\text{контекст}_{\text{готовить}} &= \\ &= \omega_{\text{готовить,я}} \overrightarrow{\text{значение}}_{\text{я}} + \omega_{\text{готовить,любить}} \overrightarrow{\text{значение}}_{\text{любить}} + \\ &+ \omega_{\text{готовить,готовить}} \overrightarrow{\text{значение}}_{\text{готовить}} + \omega_{\text{готовить,на}} \overrightarrow{\text{значение}}_{\text{на}} + \\ &+ \omega_{\text{готовить,своей}} \overrightarrow{\text{значение}}_{\text{своей}} + \omega_{\text{готовить,кухня}} \overrightarrow{\text{значение}}_{\text{кухня}}.\end{aligned}$$

Таким образом, нам удалось зафиксировать в одном векторе контекст каждого слова в данном предложении, при этом высокая ценность присвоена словам, с которыми он сочетается больше всего.

Хорошей новостью является то, что можно одновременно вычислить контекстный вектор для всех слов предложения (выборки данных), т. к. векторы, о которых говорилось выше, можно упаковать в матрицы и использовать эффективные параллельные матричные вычисления, чтобы получить контексты для всех терминов сразу.

Мы реализуем все вышеперечисленное в одной *голове внимания*. То есть одна голова внимания производит один контекстный вектор для каждого токена в выборке данных. В нашем случае бы лучше получить несколько контекстных векторов для одного и того же токена, т. к. при усреднении, происходящем на пути к контекст-

ному вектору, часть информации теряется. Идея заключается в том, чтобы иметь возможность извлекать информацию, используя не одно представление, соответствующее одной голове внимания, а разные представления терминов предложения (выборки данных). Для этого мы реализуем *многоголовое*, или *множественное*, *внимание*, выбирая для каждой головы новые матрицы преобразования $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$.

Важно отметить, что в процессе обучения записи матриц преобразования — это параметры модели, которые необходимо получить из выборок обучающих данных. Можно только представить, каким стремительным будет рост числа этих параметров.

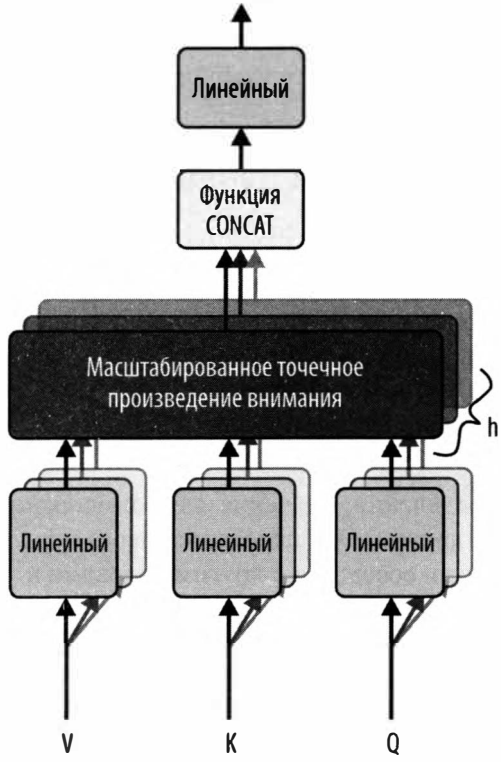


Рис. 7.6. Механизм множественного внимания (источник изображения: **https://oreil.ly/j9ypn**)

На рис. 7.6 показан *механизм множественного внимания*, в котором реализуется h головок, получающих разные линейно преобразованные версии запросов, ключей и значений для генерации h контекстных векторов для каждой лексемы, которые затем объединяются на выходе данных в составе структуры модели *множественного внимания*.

Декодер использует аналогичный механизм внутреннего внимания, но здесь каждое слово может воспринимать только слова перед собой, т. к. текст генерируется слева направо. Кроме того, декодер имеет дополнительный механизм внимания к выходам, которые он получает от кодера.

Трансформеры далеки от совершенства

Несмотря на революцию, которую произвели модели-трансформеры в области обработки естественного языка, они все еще далеки от совершенства. В целом языковые модели — это бездумные имитаторы. Они не понимают ни того, что у них на входе, ни того, что на выходе. Критический разбор, как, например, в этой (**https://oreil.ly/aaj80**) или в этой (**https://oreil.ly/ToNil**) публикации в *MIT Technology Review*, среди прочих, подробно описывают недостатки этих моделей: отсутствие понимания языка, повторы при использовании в генерации длинных отрывков текста и т. д. Тем не менее модель трансформера вызвала прилив сил в обработке естественного языка и сегодня пробивает себе дорогу в другие области ИИ, такие как биомедицина, компьютерное зрение, генерация изображений.

Сверточные нейросети для данных временных рядов

В области обработки естественного языка и финансов термин *"временной ряд"* следует заменить на термин *"временна́я последовательность"*. В математике термин *"ряд"* относится к сложению членов бесконечной *последовательности*. Поэтому, когда наши данные не суммируются, что характерно для всех данных естественного языка и большинства финансовых данных, фактически мы имеем дело не с *рядами*, а с *последовательностями* чисел, векторов и т. д. Что ж, конфликт между словарями неизбежен, даже в таких сильно зависимых друг от друга областях.

Помимо словарных определений, значения слов в основном зависят от того, как они встречаются друг с другом. Это передается через *порядок* слов в предложениях, а также через контекст и соседство с другими словами в предложениях.

Сначала отметим два способа, с помощью которых можно исследовать значения слов и терминов в документах.

Пространственный.

> Исследование предложения целиком как одного вектора токенов независимо от того, как они представлены математически.

Временной.

> Последовательное исследование предложения по одному токену за раз.

Сверточные нейросети, рассмотренные в *главе 5*, пространственно исследуют предложения, перемещая окно фиксированной ширины (ядро или фильтр) по токенам в предложении. Анализируя текстовые данные, сверточная нейросеть полагается на входные данные фиксированной размерности, а рекуррентная нейросеть (о ней речь пойдет далее) — на последовательный ввод токенов, так что входные данные не должны иметь фиксированную длину.

В *главе 5* мы плавно перемещали двумерные окна (ядра или фильтры) по изображениям, а в этой главе мы будем перемещать одномерные ядра по текстовым токенам. Нам уже известно, что каждый токен представлен в виде вектора чисел. Мы можем

использовать либо однократное горячее кодирование, либо векторы слов модели word2vec. При однократном горячем кодировании токены представляются довольно длинным вектором, где 0 — каждое возможное словарное слово, которое нужно включить в корпус, 1 — позиция кодируемого токена. В качестве альтернативы можно использовать обученные векторы слов, созданные моделью word2vec. Таким образом, выборка данных на входе сверточной нейросети представляет собой матрицу, состоящую из векторов-столбцов — по одному столбцу для каждого токена в выборке данных. Если для представления токенов используется word2vec, то каждый вектор-столбец будет содержать от 100 до 500 записей в зависимости от конкретной модели word2vec. Напомним, что для сверточной нейросети в каждой выборке данных должно содержаться одинаковое количество токенов.

Таким образом, одна выборка данных (предложение или абзац) представлена двумерной матрицей, где количество строк равно полной длине вектора слов. В таком контексте говорить, что мы скользим *одномерным* ядром по выборке данных, будет не совсем корректным, но тому есть объяснение. Векторное представление токенов выборки простирается *вниз*, однако фильтр охватывает за один раз всю длину этого измерения. То есть фильтр шириной три токена будет представлять собой матрицу весов с тремя столбцами и количеством строк, равным количеству векторных представлений токенов. Следовательно, под одномерной сверткой здесь понимается свертка только по *горизонтали*, что отличается от двумерной свертки для изображений, где двумерный фильтр перемещается по изображению как по горизонтали, так и по вертикали.

Как показано в *главе 5*, во время прямого прохода значения весов в фильтрах по одной выборке данных будут одинаковыми. Это значит, что процесс можно распараллелить, и именно поэтому сверточные нейросети эффективны в обучении.

Напомним, что сверточные нейросети также могут обрабатывать одновременно более одного канала входных данных, т. е. трехмерные тензоры входных данных, а не только двумерные матрицы чисел. В случае с изображениями это означает одновременную обработку красного, зеленого и синего каналов входного изображения. В обработке естественного языка одна входная выборка — это набор слов, представленных в виде векторов-столбцов, выстроенных рядом друг с другом. Нам известно, что существует множество способов представить одно и то же слово в виде вектора чисел, каждое из которых отражает различные семантические характеристики одного и того же слова. Его векторные представления необязательно будут одинаковой длины. Если ограничить их одинаковой длиной, то каждое представление будет каналом слова, а сверточная нейросеть сможет обрабатывать одновременно все каналы одной выборки данных.

Как отмечается в *главе 5*, эффективность сверточных нейросетей обеспечивается за счет разделения весов, пулинг-слоев, исключения и фильтров малых размеров. Можно запустить модель с фильтрами нескольких размеров, а затем объединить выход каждого фильтра в более длинный вектор мыслей до его передачи в полносвязный последний слой. Безусловно, последний слой сети способен выполнять такие задачи, как классификация настроений, фильтрация спама, генерация текста и др. Они рассматриваются в *главах 5* и *6*.

Рекуррентные нейросети для данных временных рядов

Рассмотрим следующие три предложения.

- Она купила билеты на фильм.
- Она, имея свободное время, купила билеты на фильм.
- Она, слушая отзывы на протяжении двух недель подряд, наконец купила билеты на фильм.

Во всех трех предложениях предикат "купила билеты на фильм" соответствует подлежащему "она". Модель естественного языка может этому обучиться, если она предназначена для работы с долгосрочными зависимостями. Рассмотрим, как различные модели справляются с такими долгосрочными зависимостями.

Сверточные нейросети и долгосрочные зависимости.

Сверточная нейросеть, обладающая узким окном фильтрации в диапазоне трехпяти токенов, сканирующих предложение, может легко обучиться на первом предложении и, возможно, на втором, если позиция предиката изменилась не слишком сильно (пулинг-слои добавляют сети устойчивость к небольшим изменениям). С третьим предложением могут возникнуть проблемы, если только не воспользоваться более крупными фильтрами (что повышает вычислительные затраты и делает сеть больше похожей на полносвязную, чем на сверточную сеть) или не углубить сеть наложением сверточных слоев друг на друга, чтобы по мере продвижения предложения вглубь сети охват расширялся.

Рекуррентные нейросети с модулями памяти.

Абсолютно другой подход заключается в последовательной подаче предложения в сеть, по одному токену за раз, и поддержке *состояния* и *памяти*, хранящей важную информацию в течение определенного времени. Сеть выдает результат после того, как через нее пройдут все токены из предложения. Если такое происходит во время обучения, то с меткой предложения сравнивается лишь результат, полученный после обработки последнего токена, после чего ошибка *распространяется назад во времени* для корректировки весов. Можно сравнить это с тем, как мы запоминаем информацию при чтении длинного предложения или абзаца. Именно так устроены рекуррентные нейросети с длинными блоками краткосрочной памяти.

Модели трансформеров и долгосрочные зависимости.

Модели трансформеров, о которых мы говорили ранее, отменяют свертку и рекурренцию, полагаясь только на внимание для отражения отношения между подлежащим предложения "она" и предикатом "купила билеты на фильм".

Еще одно, чем отличаются рекуррентные модели от сверточных и трансформеров: предполагает ли модель одинаковую длину входных данных по всем выборкам? Нужно ли вводить только предложения одинаковой длины? Трансформеры и сверточные сети полагаются только на выборки данных фиксированной длины, так что выборки нужно предварительно обработать и привести их к одной длине. Рекур-

рентные нейросети, в свою очередь, отлично справляются с входными данными переменной длины, т. к. принимают их фактически по одному токену за раз.

Основная идея рекуррентной нейросети заключается в том, что, обрабатывая новую информацию, она сохраняет прошлую. Как это происходит? В полносвязной нейросети с прямой связью выход нейрона покидает ее и никогда не возвращается назад. В рекуррентной сети выход возвращается в нейрон вместе с новым входом, создавая, по сути, *память*. Такие алгоритмы превосходно подходят для автозаполнения и проверки грамматики. С 2018 года они интегрированы в функцию Gmail Smart Compose.

Как работает рекуррентная нейросеть

Далее рассмотрим этапы обучения рекуррентной нейросети на наборе выборок размеченных данных. Каждая выборка состоит из набора токенов и метки. Как обычно, цель сети — обучение общим признакам и паттернам в данных, что дает по итогу каждую конкретную метку (или выход). В случае последовательного ввода токенов нужно по всем выборкам данных выявить признаки, возникающие тогда, когда в паттернах конкретные токены появляются относительно друг друга:

1. Из набора данных (например, рецензии на фильм, размеченной как положительная, или твита, размеченного как фейк) берем одну выборку данных с токенами и маркерами.
2. Передаем в сеть первый токен $token_0$ из выборки. Запомним, что токены векторизованы, так что, по сути, мы передаем в сеть вектор чисел. Говоря на языке математики, мы оцениваем функцию по вектору этого токена и получаем другой вектор. На данный момент сеть вычислила $f(token_0)$.
3. Передаем в сеть второй токен $token_1$ из выборки, а *также выход первого токена* $f(token_0)$. Теперь сеть оценивает $f(token_1 + f(token_0))$. Это шаг рекурренции, и именно так сеть не забывает о $token_0$ при обработке $token_1$.
4. Передаем в сеть третий токен $token_2$ из выборки, *а также выход предыдущего шага* $f(token_1 + f(token_0))$. Теперь сеть оценивает $f(token_2 + f(token_1 + f(token_0)))$.
5. Продолжаем до тех пор, пока в выборке не закончатся все токены. Предположим, в выборке всего пять токенов, тогда рекуррентная сеть выдаст $f(token_4 + f(token_3 + f(token_2 + f(token_1 + f(token_0)))))$. Отметим, что этот вывод очень похож на вывод полносвязных сетей прямой связи, которые обсуждаются в *главе 4* за исключением того, что при вводе токена из выборки по одному *в один рекуррентный нейрон* этот вывод *разворачивается во времени*, в то время как в *главе 4* при переходе выборки данных от одного слоя нейросети к другому вывод сети *разворачивается в пространстве*. С математической точки зрения они имеют одинаковые формулы, поэтому нам достаточно математики, которую мы рассмотрели в последних трех главах. Вот почему мы так любим математику.

6. В процессе обучения сети давать правильные результаты конечный выход по выборке $f\left(token_4 + f\left(token_3 + f\left(token_2 + f\left(token_1 + f\left(token_0\right)\right)\right)\right)\right)$ сравнивается с истинной меткой путем оценки функции потерь аналогично тому, как мы это делали в *главах 3–5*.
7. Передаем в сеть следующую выборку данных по одному токену и повторяем указанные действия.
8. Обновляем веса точно так же, как в *главе 4*, минимизируя функцию потерь с помощью алгоритма градиентного спуска, где мы вычисляем требуемый градиент (производные по отношению ко всем весам сети) методом обратного распространения ошибки. Как только что отмечено, это в точности та же математика обратного распространения ошибки, которую мы изучали в *главе 4*, только теперь, несомненно, мы *выполняем ее во времени*.

В области финансов, динамики и управления с обратной связью этот процесс называется моделью авторегрессии и скользящего среднего (autoregressive moving average, ARMA; **https://oreil.ly/GEwhG**).

Обучение рекуррентной нейросети может оказаться дорогостоящим, особенно для выборок данных значительной длины, скажем 10 токенов и более, т. к. количество обучаемых весов напрямую зависит от количества токенов в выборках: чем больше токенов, тем *глубже* рекуррентная сеть *во времени*. Помимо вычислительных затрат, такая глубина сопряжена со всеми проблемами, с которыми сталкиваются обычные сети с прямой связью с большим количеством слоев — исчезающими или взрывающимися градиентами, особенно в выборках данных с сотнями токенов, что будет математическим эквивалентом полносвязной нейросети прямой связи с сотнями слоев! Здесь работают те же средства защиты от взрывающихся и исчезающих градиентов, что и для полносвязных сетей.

Управляемые рекуррентные блоки и блоки долгой краткосрочной памяти

В рекуррентных сетях нейронов недостаточно для отражения долгосрочных зависимостей в предложении. Эффект токена размывается и дополняется новой информацией по мере того, как все больше токенов проходит через рекуррентный нейрон. Фактически информация почти полностью исчерпывается только после прохождения двух токенов. Эта проблема решается добавлением в архитектуру сети блоков памяти, которые называются *блоками долгой краткосрочной памяти* (long short-term memory units, LSTM). Они позволяют изучать зависимости по всей выборке данных.

Блоки долгой краткосрочной памяти содержат нейросети, которые можно обучить находить только такую новую информацию, которую требуется сохранить для предстоящего ввода, а также забывать или обнулять информацию, больше не имеющую значения для обучения. Таким образом, блоки долгой краткосрочной памяти обучаются тому, какую информацию следует сохранить, в то время как остальная сеть обучается предсказывать целевую метку.

Здесь нет никакой новой математики сверх того, что мы узнали в *главе 4*, поэтому не будем углубляться в особенности архитектуры блока долгой краткосрочной па-

мяти, или *управляемого блока*. В общих словах, входной токен на каждом временно́м шаге проходит через затворы (функции) забывания и обновления, умножается на ве́са и маски, а затем сохраняется в ячейке памяти. Следующий выход сети зависит от комбинации входного токена и текущего состояния ячейки памяти. Более того, ячейки долгой краткосрочной памяти совместно используют веса, которым они обучены по разным выборкам, поэтому им не нужно заново изучать основную информацию о языке при прохождении токенов из каждой выборки.

Человек может обрабатывать язык на подсознательном уровне, и блоки долгой краткосрочной памяти — это шаг к моделированию данного процесса. Они способны выявить паттерны в языке, что позволяет нам решать более сложные задачи, чем простая классификация, например генерировать язык. На основе изученных распределений вероятностей можно генерировать новые тексты. Это рассматривается в *главе 8*.

Пример данных естественного языка

Изучать модели намного проще на конкретных примерах с реальными данными и с конкретными гиперпараметрами. На сайте Stanford AI (**https://oreil.ly/by2UQ**) можно найти набор данных с рецензиями на фильмы киноагрегатора IMDb (**https://oreil.ly/HbLZX**). Каждая выборка помечена либо 0 (отрицательная рецензия), либо 1 (положительная рецензия). Если мы хотим заняться предварительной обработкой текста на естественном языке, можно начать с исходных текстовых данных. Затем выполняем токенизацию и векторизацию методом *однократного горячего кодирования* выбранного словаря, модели Google word2vec или другой модели. Не забываем разбить данные на обучающий и тестовый наборы. Затем выбираем гиперпараметры, например, длина векторов слов — 300, количество токенов в выборке — около 400, количество мини-пакетов — 32, количество эпох — 2. Для наглядности можно поиграть с этими цифрами и понять, как работают модели.

Финансовый искусственный интеллект

Модели искусственного интеллекта находят широкое применение в финансовой сфере. Мы уже знаем базовую структуру большинства моделей ИИ (за исключением графов, имеющих другую математическую структуру; мы обсудим графовые сети в *главе 9*). Сейчас достаточно лишь упомянуть финансовые приложения для формирования четкого представления, как они моделируются с помощью ИИ. Вдобавок многие финансовые приложения органично переплетаются с приложениями для обработки естественного языка, например принятие маркетинговых решений на основе отзывов покупателей или использование системы обработки естественного языка для прогнозирования экономических тенденций и инициирования крупных финансовых операций исключительно на основе результатов моделей.

Ниже приведены только два из множества примеров применения ИИ в сфере финансов. В моделировании этих задач можно воспользоваться изученным материалом.

- Прогнозирование временны́х рядов фондового рынка. Рекуррентная нейросеть может принимать последовательность входных сигналов и выдавать последовательность выходных. Это можно применять в прогнозировании временны́х рядов, необходимых для котировок акций. При вводе цен за последние n дней сеть выдает цены за последние $n-1$ дней, а также *цену на следующий день*.
- Модель авторегрессии и скользящего среднего (ARMA) в финансах, динамике и управлении с обратной связью.

Фондовый рынок неоднократно фигурирует в книге. Мы обратимся к нему при обсуждении стохастических процессов в *главе 11*, посвященной вероятности.

Итоги и перспективы

В этой главе почти не было новой математики, однако писать ее было труднее всего. Задача состояла в том, чтобы обобщить наиболее важные идеи из всей области обработки естественного языка. Главным барьером, который требовалось преодолеть, был переход от слов к относительно низкоразмерным векторам чисел, несущих смысл. Мы узнали несколько способов, как выполнить это путем векторизации слова за раз либо основных тем длинного документа или целого корпуса, и теперь подача полученных векторов в различные модели машинного обучения с разными архитектурами и целям — это привычное дело.

Исчисление.

Логарифмическая шкала для частот терминов и обратных частот документов.

Статистика.

- Закон Ципфа для подсчета слов.
- Распределение вероятностей Дирихле для предписания слов к темам и тем к документам.

Линейная алгебра.

- Векторизация документов на естественном языке.
- Точечное произведение двух векторов и то, как оно обеспечивает меру сходства или совместимости между представленными векторами сущностями.
- Косинусное сходство.
- Сингулярное разложение, т. е. латентно-семантический анализ.

Вероятность.

- Условные вероятности.
- Лог-билинейная модель.

Данные временны́х рядов.

- Что это означает.
- Как они подаются в модели машинного обучения (как единый массив или по одному элементу за раз).

Модель искусственного интеллекта.

- Трансформер.

ГЛАВА 8

Вероятностные генеративные модели

Искусственный интеллект связывает воедино всю известную мне математику, а я изучаю ее уже много лет.

— Хала

Если когда-нибудь машины будут наделены пониманием окружающего мира и способностью воссоздавать его, как это делаем мы, когда воображаем, мечтаем, рисуем, сочиняем песни, смотрим фильмы или пишем книги, то генеративные модели — один из важных шагов в этом направлении. Для достижения общего искусственного интеллекта необходимо создать соответствующие модели.

Генеративная модель основана на предположении, что входные данные могут быть правильно интерпретированы только в том случае, если модель изучила статистическую структуру, лежащую в основе этих данных. Это в значительной степени аналогично процессу сновидений, который указывает на возможность того, что наш мозг выучил модель, способную виртуально воссоздать окружающую среду.

В этой главе мы пользуемся той же самой математической структурой, включающей функцию обучения, функцию потерь и оптимизацию, что и на протяжении всей книги. Но в отличие от первых нескольких глав, сейчас мы изучаем не детерминированные функции, а вероятностные распределения. По сути, имеются обучающие данные, и нам нужно придумать математическую модель, способную генерировать новые данные, похожие на эти.

Интерес представляют две величины:

- истинное (и неизвестное) совместное распределение вероятностей признаков входных данных $p_{\text{данные}}(\vec{x})$;
- совместное распределение вероятностей признаков данных и параметров модели: $p_{\text{модель}}(\vec{x};\vec{\theta})$.

В идеале мы хотим, чтобы эти два параметра были как можно ближе. На практике мы выбираем значения параметров $\vec{\theta}$, которые в наших конкретных случаях позволяют оптимально работать $p_{\text{модель}}(\vec{x};\vec{\theta})$.

На протяжении всей главы мы придерживаемся трех правил распределения вероятностей:

1. Правило произведения, которое разлагает многомерное совместное распределение вероятностей на произведение условных распределений вероятностей одной переменной.
2. Правило Байеса, позволяющее плавно переходить от одной переменной к другой.
3. Предположения о независимости или условной независимости признаков или латентных (скрытых) переменных, которые еще больше упрощают произведение условных вероятностей одной переменной.

В предыдущих главах мы минимизировали *функцию потерь*. В этой главе аналогичной функцией является *логарифмическая функция правдоподобия*, а процесс оптимизации всегда пытается *максимизировать* это логарифмическое правдоподобие (осторожно, вместо минимизации функции потерь мы максимизируем объективную функцию). В ближайшее время мы поговорим об этом подробнее.

Прежде чем мы углубимся в предмет, поясним, как перевести наши предыдущие детерминированные модели машинного обучения на язык вероятностей. Предыдущие модели обучали функцию, проецирующую характеристики входных данных $\vec{x}$ на выход y (цель или метку), или $f\left(\vec{x};\vec{\theta}\right)=y$. Когда нашей целью была классификация, f возвращала метку y, имеющую наибольшую вероятность. То есть классификатор обучается прямому отображению входных данных $\vec{x}$ на метки классов y; другими словами, он напрямую моделирует *апостериорную вероятность* $p\left(y|\vec{x}\right)$. Мы остановимся на этом позже в текущей главе.

Ценность генеративных моделей

Генеративные модели размывают границы между реальными и сгенерированными компьютером данными. Они совершенствуются и достигают впечатляющих успехов — созданные машиной изображения, в том числе изображения людей, становятся все более реалистичными. Сегодня сложно сказать, является ли изображение модели в индустрии моды реальным человеком или результатом работы генеративной модели машинного обучения.

Цель генеративной модели — использовать машину для генерации новых данных, таких как звуковые сигналы, содержащие речь, изображения, видео или текст на естественном языке. Генеративные модели выбирают данные из изученного распределения вероятностей, причем такие выборки максимально имитируют реальность. Предполагается, что в основе реальных данных лежит некое неизвестное распределение вероятностей, которое мы хотим имитировать (иначе вся наша реальность будет представлять собой случайный хаотический шум, лишенный связ-

ности или структуры), а цель модели — научиться аппроксимировать такое распределение вероятностей на основе обучающих данных.

Собрав большой объем данных из определенной области, мы обучаем генеративную модель генерировать данные, аналогичные собранным. Собранные данные могут представлять собой миллионы изображений или видео, тысячи аудиозаписей или целые корпуса естественного языка.

Генеративные модели полезны для многих приложений, в том числе для дополнения данных, когда их мало, а нужно больше, ввода недостающих значений для изображений с высоким разрешением, моделирования новых данных для обучения с подкреплением или для полуконтрольного обучения, когда доступно лишь несколько меток. Другая область применения — перевод изображений в изображения, например преобразование аэрофотоснимков в карты или преобразование нарисованных от руки эскизов в изображения. Среди других приложений можно перечислить шумоподавление (*денойзинг*), ретуширование (*инпейнтинг*), сверхразрешение и редактирование изображений, например, когда мы делаем на фотографии улыбки шире, скулы выше, а лица стройнее.

Более того, генеративные модели строятся таким образом, чтобы генерировать более одного приемлемого результата, взяв несколько выборок из желаемого распределения вероятностей. Это отличается от детерминированных моделей, которые усредняют выходные данные по различным признакам во время обучения, используя функцию потерь среднеквадратической ошибки или другую усредняющую функцию потерь. Недостатком генеративной модели является то, что она также может брать неудачные выборки.

Один из видов генеративных моделей — *генеративно-состязательные сети* (представленные в 2014 году Яном Гудфеллоу и др.; **https://oreil.ly/pTJZN**) — имеют огромные перспективы и широкий спектр применения, начиная с дополнения наборов данных для завершения маскирования лица человека и заканчивая астрофизикой и физикой высоких энергий, например моделирование наборов данных, подобных тем, что генерируются на Большом адронном коллайдере в ЦЕРНе, или моделирование распределения темной материи и предсказания гравитационного линзирования. Генеративно-состязательные модели устанавливают две нейросети, играющие друг с другом в игру с нулевой суммой (вспомним теорию игр в математике) до тех пор, пока машина сама не сможет отличить реальное изображение от сгенерированного компьютером. Именно поэтому их результаты выглядят довольно близкими к реальности.

В *главе 7*, которая в значительной степени ориентирована на обработку естественного языка, мы уже соприкасались с генеративными моделями, не указывая на них в явном виде. Большинство приложений обработки естественного языка, которые представляют собой не только модель классификации (на спам/не спам, позитивные/негативные настроения и тегирование частей речи), включают в себя генерацию языка, например автозаполнение в смартфонах или электронной почте, машинный перевод, резюмирование текста, чат-боты, создание подписей к изображениям.

Типичная математика генеративных моделей

Генеративные модели воспринимают и представляют мир с помощью распределения вероятностей. То есть цветное изображение — это одна выборка из совместного распределения вероятностей пикселов, которые вместе образуют осмысленное изображение (попробуйте подсчитать размеры такого совместного распределения вероятностей с учетом всех красных, зеленых и синих каналов), звуковая волна — это одна выборка из совместного распределения вероятностей аудиосигналов, которые вместе образуют осмысленные звуки (они также довольно высокой размерности), а предложение — это одна выборка из совместного распределения вероятностей слов или символов, которые вместе представляют собой связные предложения.

И здесь возникает вопрос: как вычислить эти необычайно репрезентативные совместные распределения вероятностей, способные отразить всю сложность окружающего нас мира, но, к сожалению, имеющие довольно высокую размерность?

С точки зрения машинного обучения ответ предсказуем. Начнем с известного нам простого распределения вероятностей, например с гауссова распределения, а затем найдем способ преобразовать его в другое распределение, хорошо аппроксимирующее эмпирическое распределение данных. Но как преобразовать одно распределение в другое? Можно к плотности его вероятности применить детерминированную функцию. Поэтому необходимо выяснить следующее.

Как применить детерминированную функцию к распределению вероятностей и каким будет распределение вероятностей полученной случайной величины?

Мы используем такую формулу преобразования:

$$p_x(\vec{x}) = p_z\left(g^{-1}(\vec{x})\right)\left|\det\left(\frac{\partial g^{-1}\vec{x}}{\partial \vec{x}}\right)\right|.$$

Это очень хорошо описано в книгах по теории вероятностей, и вскоре мы возьмем оттуда все необходимое.

Какую же правильную функцию нужно применить?

Один из способов — натренировать модель, чтобы она *обучилась* этому. Известно, что нейросети способны представить широкий спектр функций, так что можно пропустить начальное простое распределение вероятностей через нейросеть (нейросеть будет формулой искомой детерминированной функции), затем обучить параметры сети путем минимизации ошибки между эмпирическим распределением данных и выходным распределением сети.

Как измерить ошибки между распределениями вероятностей?

Теория вероятностей предоставляет нам некоторые показатели того, как два распределения вероятностей расходятся друг с другом, например *дивергенция Кульбака — Лейблера* (KL). Это также связано с *перекрестной энтропией* из *теории информации*.

Все ли генеративные модели работают таким образом?

И да и нет. *Да* — в том смысле, что все они пытаются обучиться совместному распределению вероятностей, которое предположительно породило обучающие данные. Другими словами, генеративные модели пытаются выучить формулу и параметры совместного распределения вероятностей, максимизирующие вероятность обучающих данных (или вероятность, которую модель приписывает обучающим данным). *Нет* — в том смысле, что нами только намечен *явный способ* аппроксимации желаемого совместного распределения вероятностей. Это одна из научных точек зрения. В общем случае модель, определяющая явную и понятную функцию плотности вероятности, позволяет нам оперировать непосредственно логарифмическим правдоподобием обучающих данных, вычислять его градиент и применять доступные алгоритмы оптимизации в поиске максимума. Существуют и другие модели, которые дают явную, но труднодостижимую функцию плотности вероятности, где для максимизации правдоподобия нужно использовать приближения. Как *приближенно* решить задачу оптимизации? Здесь можно воспользоваться либо детерминированным приближением, опираясь на *вариационные методы* (модели вариационного автоэнкодера), либо стохастическим приближением, опираясь на *методы Монте-Карло с марковскими цепями*. Наконец, существуют *неявные* способы аппроксимации желаемого совместного распределения вероятностей. Неявные модели учатся делать выборки из неизвестного распределения, не определяя его формулу в явном виде. К этой категории относятся *генеративно-состязательные сети*.

В настоящее время можно выделить три наиболее популярных подхода к генеративному моделированию.

Генеративно-состязательные сети.

Это неявные модели плотности.

Вариационные модели, которые дают явную, но трудноразрешимую функцию плотности вероятности.

Аппроксимация решения задачи оптимизации в рамках вероятностных графовых моделей, где нижняя граница максимизируется на логарифмическую вероятность данных, т. к. непосредственная максимизация логарифмической вероятности данных считается трудновыполнимой.

Полностью видимые сети доверия.

Такие модели, как Pixel Convolutional Neural Networks (PixelCNN) 2016 (**https://oreil.ly/DJFM0**) и WaveNet (2016 г., **https://oreil.ly/YODqz**), предоставляют явные и легко реализуемые функции плотности вероятности. Эти модели изучают совместное распределение вероятностей, разлагая его на произведение одномерных распределений вероятностей для каждого отдельного измерения, обусловленных предшествующими ему и изучая каждое из этих распределений по очереди. Такое разложение происходит согласно *правилу произведения*, или цепному правилу для вероятностей. Например, PixelCNN обучает сеть условному распределению вероятностей каждого отдельного пиксела изображения с

учетом предыдущих пикселов (слева и сверху от него), а WaveNet обучает сеть условному распределению вероятностей каждого отдельного звукового сигнала в звуковой волне с учетом предшествующих ему. Недостатки этих моделей в том, что они генерируют образцы только по одной записи за раз и не допускают распараллеливания. Это существенно замедляет процесс генерации. Например, для генерации одной секунды звука WaveNet требуется две минуты вычислительного времени, поэтому мы не можем использовать ее в проведении онлайн-встреч.

Существуют и другие генеративные модели, которые попадают в вышеперечисленные категории, но менее популярны вследствие дорогих вычислений или сложностей с выбором функции плотности и/или ее преобразований. К ним относятся модели, требующие смены переменных, такие как нелинейный анализ независимых компонент (явная и легко реализуемая модель плотности), модели машины Больцмана (явная и труднореализуемая модель плотности со стохастической цепью Маркова для приближенного решения задачи максимизации) и генеративные стохастические сетевые модели (неявная модель плотности, снова зависящая от цепи Маркова для получения приближенного максимального правдоподобия). Мы кратко рассмотрим эти модели в конце текущей главы. На практике и вдали от математической теории и анализа подходы, основанные на использовании цепей Маркова, теряют свою популярность из-за стоимости вычислений и нерасположенности к быстрому схождению.

Переключение мозга с детерминированного на вероятностное мышление

В этой главе мы постепенно переключаем наш мозг с детерминированного мышления на вероятностное. До сих пор для построения прогнозов мы пользовались в книге только детерминированными функциями. Обучающие функции представляли собой линейные комбинации признаков данных, иногда дополненные нелинейными активаторами, функции потерь были детерминированными дискриминаторами между истинными и предсказанными значениями, а методы оптимизации базировались на детерминированных методах градиентного спуска. Стохастичность, или случайность, вводилась только тогда, когда требовалось сделать менее затратными вычисления детерминированных компонентов модели (например, стохастический градиентный спуск или стохастическое сингулярное разложение); когда наборы данных разбивались на обучающие, проверочные и тестовые поднаборы; когда выбирались мини-пакеты; когда обходились некоторые пространства гиперпараметров; или когда оценки выборок данных передавались в функцию softmax, являющуюся детерминированной, а полученные значения интерпретировались как вероятности. Во всех этих случаях стохастичность и связанные с ней распределения вероятностей относились только к конкретным компонентам модели, выступая лишь как средство достижения цели — практической реализации и вычислений детерминированной модели. Они никогда не входили в основной состав модели.

Генеративные модели отличаются от моделей, рассматриваемых в предыдущих главах, тем, что в своей основе они являются вероятностными. Тем не менее у нас все та же структура обучения, потерь и оптимизации, за исключением того, что теперь модель обучается не детерминированной функции, а распределению вероятностей (явно или неявно). Теперь функция потерь измеряет ошибку между истинным и предсказанным распределениями вероятностей (по меньшей мере, для явных моделей плотности), поэтому нужно выяснить, как определить и вычислить не детерминированное значение, а некую функцию ошибки между вероятностями. Также нужно научиться оптимизировать и находить производные в этой вероятностной среде.

В математике гораздо проще оценить (прямая задача) заданную функцию, чем найти ее обратную величину (обратная задача), тем более при наличии доступа только к нескольким показаниям значений функции, например к выборкам данных. В вероятностной постановке прямая задача выглядит следующим образом: осуществить выборку данных при определенном распределении вероятности. Нас интересует обратная задача: учитывая конечное число реализаций (выборок данных) неизвестного нам распределения вероятности, найти распределение вероятности, предположительно породившее их. На ум сразу же приходит одна сложность — проблема уникальности, ведь может существовать более одного распределения, соответствующего нашим данным. Более того, обратная задача обычно намного сложнее, т. к., по сути, чтобы прийти к данным показателям, нам нужно действовать в обратном направлении и отменить процесс, который выполняла прямая функция. Проблема в том, что отменить большинство процессов невозможно, и это то, что больше нас и заложено в законах — Вселенная имеет тенденцию к увеличению энтропии. Помимо сложностей, свойственных решению обратных задач, вероятностные распределения, которые мы обычно пытаемся оценить для приложений ИИ, имеют высокую размерность, множество переменных, и мы даже не можем быть уверенными в том, что наша вероятностная модель учла все переменные (впрочем, это проблематично и для детерминированных моделей). Но не стоит бояться этих сложностей. Представление и операции с высокоразмерными вероятностными распределениями играют важную роль во многих математических, научных, финансовых, инженерных и других приложениях. В общем, нужно углубиться в генеративные модели.

На протяжении всей главы мы будем различать случаи, когда оцениваемое распределение вероятностей задается явной формулой, и когда у нас нет формулы, а вместо этого мы численно генерируем новые выборки данных из неявного распределения. Важно отметить, что в предыдущих главах все детерминированные модели всегда имели явные формулы для функций обучения, в том числе для деревьев решений, а также для полносвязных и сверточных нейросетей. Тогда, оценив эти детерминированные функции по данным, мы могли ответить на такие вопросы, как: каким будет предсказанное значение целевой переменной? В случае вероятностных моделей мы отвечаем на другой вопрос: какой будет вероятность того, что целевая переменная примет определенное значение или будет находиться в определенном интервале? Разница в том, что здесь мы не знаем, как модель объединила перемен-

ные, прежде чем выдать результат, как это было в случае детерминированной функции. В вероятностных моделях мы пытаемся оценить вероятность того, что переменные модели появятся вместе с целевой переменной (их совместную вероятность), в идеале по всем диапазонам переменных. Это даст распределение вероятностей целевой переменной без необходимости явно формулировать, как переменные модели взаимодействуют, прежде чем получить такой результат. Это зависит исключительно от изучения данных.

Оценка максимального правдоподобия

Многие генеративные модели прямо или косвенно опираются на *принцип максимального правдоподобия*. Для вероятностных моделей цель состоит в том, чтобы обучиться распределению вероятностей, которое аппроксимирует истинное распределение вероятностей наблюдаемых данных. Один из способов — задать явное распределение вероятностей $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ с некоторыми неизвестными параметрами $\vec{\theta}$, а затем решить вопрос параметров $\vec{\theta}$, которые делают обучающий набор данных максимально вероятным для наблюдения. То есть нужно найти $\vec{\theta}$, которое *максимизирует правдоподобие* обучающих данных, присваивая таким выборкам высокую вероятность. Если имеется m обучающих точек данных, мы предполагаем, что они отбираются независимо друг от друга, так что вероятность их совместного появления равна произведению вероятностей всех отдельных выборок. Таким образом, мы имеем:

$$\vec{\theta}_{\text{оптим}} = \arg\max_{\vec{\theta}} p_{\text{модель}}\left(\vec{x}^1;\vec{\theta}\right) p_{\text{модель}}\left(\vec{x}^2;\vec{\theta}\right)\cdots p_{\text{модель}}\left(\vec{x}^m;\vec{\theta}\right).$$

Напомним, что каждая вероятность — это число от нуля до единицы. Если мы перемножим все эти вероятности, то получим числа с чрезвычайно малой величиной, что приведет к нестабильности вычислений и риску переполнения (когда машина сохраняет очень маленькое число как ноль, удаляя все значимые цифры). Логарифмическая функция всегда решает эту проблему, преобразуя все числа, чья величина чрезвычайно велика или чрезвычайно мала, обратно в область приемлемых величин. Хорошая новость заключается в том, что преобразование вероятностей в логарифмы не влияет на оптимальные значения $\vec{\theta}$, т. к. логарифмическая функция является возрастающей.

То есть, если $f\left(\vec{\theta}_{\text{оптим}}\right) \geq f\left(\vec{\theta}\right)$ для всех $\vec{\theta}$, то и $\log\left(f\left(\vec{\theta}_{\text{оптим}}\right)\right) \geq \log\left(f\left(\vec{\theta}\right)\right)$ для всех $\vec{\theta}$. Композиция с возрастающими функциями не меняет знака неравенства. Дело в том, что решение максимального правдоподобия становится эквивалентным решению максимального логарифмического правдоподобия. Теперь вспомним, что логарифмическая функция преобразует произведения в суммы, так что мы имеем:

$$\vec{\theta}_{\text{оптим}} = \arg\max_{\vec{\theta}} \log\left(p_{\text{модель}}\left(\vec{x}^1;\vec{\theta}\right)\right) + \log\left(p_{\text{модель}}\left(\vec{x}^2;\vec{\theta}\right)\right) + \ldots + \log\left(p_{\text{модель}}\left(\vec{x}^m;\vec{\theta}\right)\right).$$

Отметим, что это выражение стремится увеличить каждое $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ для каждой выборки данных. То есть оно предпочитает, чтобы значения $\vec{\theta}$ *поднимали* граф $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ над каждой точкой данных $\vec{x}^i$. Однако мы не можем бесконечно поднимать значения. Требуется компенсация в сторону уменьшения, т. к. *гиперплощадь* области под графиком должна равняться 1, зная, что $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ — это распределение вероятностей.

Можно переформулировать наше выражение в терминах ожидания и условных вероятностей:

$$\vec{\theta}_{\text{оптим}} = \arg\max_{\vec{\theta}} \mathbb{E}_{x\sim p_{\text{данные}}} \log\left(p_{\text{модель}}\left(\vec{x}\middle|\vec{\theta}\right)\right).$$

В детерминированных моделях, рассмотренных в предыдущих главах, параметры (или веса) моделей находятся методом минимизации функции потерь, измеряющим ошибку между предсказаниями моделей и истинными значениями, представленными метками данных, или, другими словами, между $y_{\text{модель}}$ и $y_{\text{данные}}$. В этой главе мы уделяем внимание поиску параметров, максимизирующих логарифмическое правдоподобие данных. Было бы неплохо, если бы существовала формулировка максимизации логарифмического правдоподобия, аналогичная минимизации величины, измеряющей ошибку между вероятностными распределениями $p_{\text{модель}}$ и $p_{\text{данные}}$, чтобы отчетливо проследить аналогию между этой и предыдущими главами. К счастью, она есть. Оценка максимального правдоподобия — это минимизация *дивергенции Кульбака — Лейблера* (KL-дивергенция; **https://oreil.ly/Dk4NY**) между породившим данные распределением вероятностей и распределением вероятностей модели:

$$\vec{\theta}_{\text{оптим}} = \arg\min_{\vec{\theta}} \text{Дивергенция}_{\text{KL}}\left(p_{\text{данные}}(\vec{x})\middle\|p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)\right).$$

Если $p_{\text{данные}}$ окажется членом семейства распределений $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ и мы могли бы точно выполнить минимизацию, то получили бы точное распределение, породившее данные, а именно $p_{\text{данные}}$. Однако на практике у нас нет доступа к данным, породившим распределение; фактически это распределение мы пытаемся аппроксимировать. Мы имеем доступ только к m выборкам из $p_{\text{данные}}$. Такие выборки определяют эмпирическое распределение $\hat{p}_{\text{данные}}$, накладывающее массу только на эти m выборок. Сейчас максимизация логарифмического правдоподобия обучающего набора эквивалентна минимизации дивергенции Кульбака — Лейблера между $\hat{p}_{\text{данные}}$ и $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$:

$$\vec{\theta}_{\text{оптим}} = \arg\min_{\vec{\theta}} \text{Дивергенция}_{\text{KL}}\left(\hat{p}_{\text{данные}}(\vec{x})\middle\|p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)\right).$$

Здесь можно запутаться в трех задачах оптимизации, которые с точки зрения математики фактически эквивалентны, просто относятся к разным подразделам и субкультурам математики, статистики, естественных наук и информатики:

- максимизация логарифмического правдоподобия обучающих данных;
- минимизация KL-дивергенции между эмпирическим распределением обучающих данных и распределением модели;
- минимизация функции потерь *перекрестной энтропии* между метками обучающих данных и выходами модели при классификации на несколько категорий при помощи композиции с функцией softmax.

И здесь важно учитывать, что параметры, минимизирующие KL-дивергенцию, совпадают с параметрами, минимизирующими перекрестную энтропию и отрицательное логарифмическое правдоподобие.

Явные и неявные модели плотности

Цель оценки максимального логарифмического правдоподобия (или минимальной KL-дивергенции) — найти распределение вероятностей $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$, которое наилучшим образом объясняет наблюдаемые данные. Генеративные модели используют обученную $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ для генерации новых данных. Существуют два подхода: явный и неявный.

Явная модель плотности.

Определяет формулу распределения вероятностей *в явном виде* в терминах $\vec{x}$ и $\vec{\theta}$, затем находит значения $\vec{\theta}$, максимизирующие логарифмическое правдоподобие выборок обучающих данных, следуя за вектором градиента (частных производных по компонентам $\vec{\theta}$) вверх. Основная проблема здесь в том, чтобы найти формулу для плотности вероятности, которая отражала бы сложность данных, оставаясь в то же время пригодной для вычисления логарифмического правдоподобия по градиенту.

Неявная модель плотности.

Выборка непосредственно из $p_{\text{модель}}\left(\vec{x};\vec{\theta}\right)$ без написания формулы для этого распределения. Генеративные стохастические сети делают это на основе цепей Маркова, которые медленно сходятся и поэтому непопулярны в практических приложениях. Используя такой подход, модель случайным образом преобразует существующую выборку и получает другую выборку из того же распределения. Генеративно-состязательные сети косвенно взаимодействуют с вероятностным распределением модели, не определяя его явно. Они организуют между двумя сетями игру с нулевой суммой, где одна сеть генерирует выборку, а другая действует как классификатор, который определяет, принадлежит ли сгенерированная выборка правильному распределению или нет.

Явно прослеживаемая плотность: полностью видимые сети доверия

Такие модели допускают явную функцию плотности вероятности с удобной оптимизацией логарифмического правдоподобия. Они опираются на *цепное правило вероятности* (**https://oreil.ly/gmC1m**) для разложения совместного распределения вероятностей $p_{\text{модель}}(\vec{x})$ на произведение одномерных распределений вероятностей:

$$p_{\text{модель}}(\vec{x}) = \prod_{i=1}^{n} p_{\text{модель}}(x_i \mid x_1, x_2, \ldots, x_{i-1}).$$

Основным недостатком здесь является то, что выборки должны генерироваться по одному компоненту за раз (один пиксел изображения, или один символ слова, или одна запись дискретной звуковой волны), так что стоимость генерации одной выборки составляет $O(n)$.

Пример: генерация изображений с помощью PixelCNN и машинного звука с помощью WaveNet

PixelCNN (**https://oreil.ly/y1lHU**) обучает сверточную нейросеть, которая моделирует условное распределение каждого отдельного пиксела, учитывая предыдущие пикселы (слева и сверху от целевого пиксела), что показано на рис. 8.1.

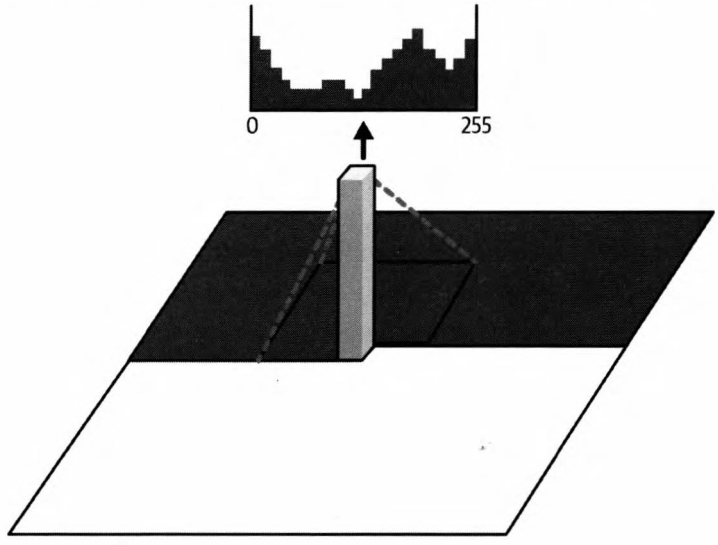


Рис. 8.1. PixelCNN обучается условному распределению n-го пиксела, обусловленному предыдущими $n - 1$ пикселами (источник изображения: **https://oreil.ly/aTZkg**)

WaveNet обучает сверточную нейросеть, которая моделирует условное распределение каждой записи звуковой волны с учетом предыдущих записей. Остановимся

подробно только на WaveNet — она является одномерным аналогом PixelCNN и отражает основные идеи.

Цель WaveNet — генерировать широкополосные необработанные звуковые волны. Поэтому нам нужно узнать совместное распределение вероятностей формы аудиосигнала $\vec{x} = (x_1, x_2, ..., x_T)$ определенного жанра.

Воспользуемся правилом произведения для разложения совместного распределения на произведение распределений с одной переменной, где каждая запись формы аудиосигнала обусловлена предшествующей:

$$p_{\text{модель}}(\vec{x}) = \prod_{t=1}^{T} p_{\text{модель}}(x_t \mid x_1, x_2, ..., x_{t-1}).$$

Одна из трудностей заключается в том, что формы звуковых сигналов имеют очень высокое временно́е разрешение: на 1 секунду звука приходится не менее 16 000 записей (таким образом, один образец данных длиной в минуту представляет собой вектор с $T = 960\,000$ записей). Каждая запись представляет собой один временной шаг дискретизированного необработанного звука и хранится как 16-битное целое число. То есть каждая запись может принимать любое значение от 0 до 65 535. Если мы сохраним этот диапазон, то сеть должна выучить вероятность для каждой записи, поэтому функция softmax должна выдать на выходном уровне 65 536 оценок вероятности для каждой отдельной записи. Общее количество записей, для которых это нужно выполнить, а также вычислительная сложность самой сети становятся весьма дорогими. Для упрощения задачи требуется квантование, что в электронике означает приближение непрерывно изменяющегося сигнала к сигналу, чья амплитуда ограничена заданным набором значений. WaveNet преобразует исходные данные, ограничивая значения записей 256 вариантами от 0 до 255, аналогично диапазону пикселов в цифровых изображениях. Теперь в процессе обучения сеть должна обучиться распределению вероятности каждой записи по этим 256 значениям, учитывая предыдущие записи, и во время генерации звука она осуществляет выборку из выученных распределений по одной записи за раз.

Последняя трудность заключается в том, что если звуковой сигнал представляет собой что-то значимое, то представляющий его вектор имеет дальнодействующие зависимости на нескольких временны́х шкалах. Для того чтобы уловить эти дальнодействующие зависимости, WaveNet использует *расширяемые свертки*. Это одномерные ядра, или фильтры, которые пропускают некоторые записи для охвата более широкого диапазона без увеличения числа параметров (рис. 8.2).

Важно отметить, что сеть не может заглянуть в будущее, поэтому фильтры на каждом слое не могут использовать записи из обучающей выборки, предшествующие целевой записи. В одном измерении мы просто прекращаем фильтрацию раньше на каждом сверточном слое, так что это простой сдвиг по времени. В двух измерениях используются *маскированные фильтры*, которые имеют нули справа и снизу от центральной записи.

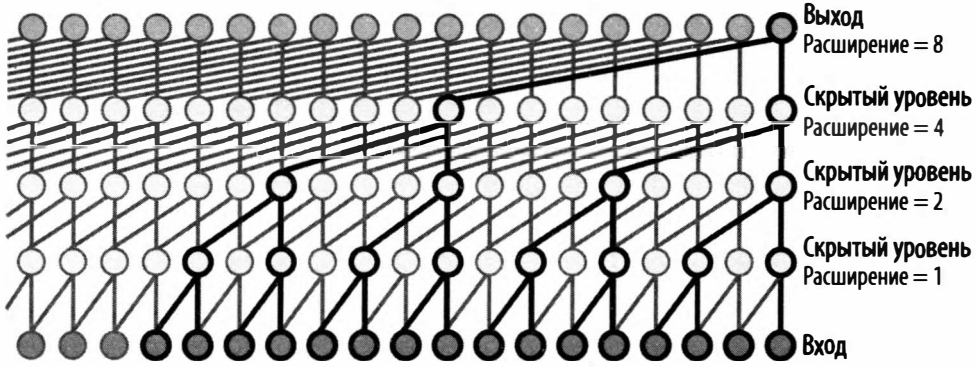


Рис. 8.2. Расширяемая свертка с размером ядра, равным двум. На каждом слое ядро имеет только два параметра, но для большего охвата оно пропускает записи (источник изображения с красивой анимацией: **https://oreil.ly/WiHpx**)

WaveNet изучает в общей сложности T распределений вероятностей, по одному для каждой записи формы звуковой волны, обусловленной предшествующими ей записями: $p_{\text{модель}}(x_1)$, $p_{\text{модель}}(x_2 \mid x_1)$, $p_{\text{модель}}(x_3 \mid x_1, x_2)$, ... и $p_{\text{модель}}(x_T \mid x_1, x_2, ..., x_{T-1})$. В процессе обучения эти распределения могут вычисляться параллельно.

Предположим, что требуется узнать распределение вероятностей 100-й записи, учитывая предыдущие 99 записей. Введем пакеты аудиовыборок из обучающих данных, после чего сверточная сеть, используя только первые 99 записей каждой выборки, вычисляет линейные комбинации (фильтры линейно комбинируются), проходит через нелинейные функции активации от одного слоя к другому, используя некоторые пропускные связи и остаточные слои для борьбы с исчезающими градиентами, и, наконец, пропускает результат через функцию softmax и выводит вектор длины 256, содержащий оценки вероятности для значения 100-й записи. Это и есть распределение вероятностей для 100-й записи, которое выводит модель. После сравнения этого выходного распределения с эмпирическим распределением данных для 100-й записи из обучающего пакета параметры сети корректируются для уменьшения ошибки (уменьшения кросс-энтропии или увеличения правдоподобия). По мере прохождения через сеть большего количества партий данных и эпох распределение вероятностей для 100-й записи с учетом предыдущих 99 будет приближаться к эмпирическому распределению обучающих данных. После обучения значения параметров сохраняются в сети. Теперь можно использовать обученную сеть для генерации машинного звука, по одной записи за раз:

1. Осуществляем выборку значения x_1 из распределения вероятностей $p_{\text{модель}}(x_1)$.
2. Дополняем (x_1) нулями, чтобы установить необходимую длину входного сигнала сети, и пропускаем вектор через сеть. На выходе получаем $p_{\text{модель}}(x_2 \mid x_1)$, откуда можно сделать выборку x_2.

3. Дополняем (x_1, x_2) нулями и пропускаем вектор через сеть. На выходе получаем $p_{\text{модель}}(x_3 \mid x_1, x_2)$, откуда можно сделать выборку x_3.
4. Движемся дальше.

Можно настроить модель WaveNet на определенную идентификацию диктора, чтобы с помощью одной модели генерировать разные голоса.

Существенным недостатком является то, что WaveNet можно обучать параллельно, но использовать ее для генерации звука только последовательно. Это исправлено с помощью Parallel WaveNet (**https://oreil.ly/BJCHO**), доступной онлайн в Google Ассистент, которая поддерживает несколько голосов на английском и японском языках.

Подводя итог и придерживаясь того же математического контекста, что и на протяжении всей главы, можно сказать, что PixelCNN и WaveNet — это модели, нацеленные на обучение совместному распределению вероятностей данных изображений или аудиоданных определенных жанров. Для этого они разлагают совместное распределение на произведение одномерных распределений вероятностей для каждой записи данных, обусловленных всеми предыдущими записями. Для поиска таких одномерных условных распределений они используют сверточную сеть, чтобы обучиться тому, как наблюдаемые записи взаимодействуют друг с другом и создают распределение следующей записи. Таким образом, вход сети является детерминированным, а ее выход — *массовая функция вероятности*. Сама сеть также является детерминированной функцией. Можно рассматривать сеть вместе с ее выходом как распределение вероятностей с настраиваемыми параметрами. По мере обучения выходной сигнал корректируется, пока не достигает допустимого соответствия с эмпирическим распределением обучающих данных. Таким образом, мы не применяем детерминированную функцию к распределению вероятностей и не подстраиваем параметры функции до тех пор, пока не добьемся соответствия с распределением обучающих данных. Так что мы начинаем с явной формулы для распределения вероятностей со многими параметрами (параметрами сети), а затем подстраиваем параметры, пока это явное распределение вероятностей корректно не согласуется с обучающими данными. Эта процедура выполняется для каждого условного распределения вероятностей, соответствующего каждой записи.

Явно прослеживаемая плотность: нелинейный анализ независимых компонент с изменением переменных

Основная идея заключается в том, что имеется случайная переменная, представляющая наблюдаемые обучающие данные $\vec{x}$, и требуется узнать породившую ее исходную случайную переменную $\vec{s}$. Мы предполагаем, что существует детерминированное преобразование $g(\vec{s}) = \vec{x}$, которое является инвертируемым и диффе-

ренцируемым и преобразует неизвестную $\vec{s}$ в наблюдаемую $\vec{x}$. То есть $\vec{s} = g^{-1}(\vec{x})$. Теперь нужно найти соответствующее g, чтобы найти распределение вероятностей $\vec{s}$. Более того, мы предполагаем, что $\vec{s}$ имеет независимые записи, или компоненты, так что ее распределение вероятностей есть не что иное, как произведение распределений ее компонент.

Формула, связывающая вероятностное распределение случайной величины с вероятностным распределением ее детерминированного преобразования, имеет вид:

$$p_s(\vec{s}) = p_x(\vec{x}) \times \text{Jac} = p_x\left(g(\vec{s})\right)\left|\det\left(\frac{\partial g(\vec{s})}{\partial \vec{s}}\right)\right|.$$

Умножение на *якобиан* (Jac) преобразования учитывает изменение объема в пространстве в результате преобразования.

Нелинейный анализ независимых компонент (**https://oreil.ly/0nVvD**) моделирует совместное распределение вероятностей как нелинейное преобразование данных $\vec{s} = g^{-1}(\vec{x})$. Преобразование g изучается таким образом, что g^{-1} проецирует данные в латентное пространство, где они соответствуют факторизованному распределению; т. е. отображение приводит к независимым латентным переменным. Преобразование g^{-1} параметризовано таким образом, чтобы можно было легко вычислить якобиан и обратный якобиан. Оно базируется на *глубокой нейросети*, а его параметры изучаются путем оптимизации логарифмического правдоподобия, что легко выполнимо.

Важно отметить, что условие, согласно которому преобразование g должно быть инвертируемым, означает, что латентные переменные $\vec{s}$ должны иметь ту же размерность, что и признаки данных (длина $\vec{x}$). Это накладывает ограничения на выбор функции g и является недостатком моделей нелинейного анализа независимых компонент.

Для сравнения, генеративно-состязательные сети предъявляют очень мало требований к g и, в частности, позволяют $\vec{s}$ иметь больше размерностей, чем $\vec{x}$.

Явно непрослеживаемая плотность: аппроксимация вариационных автокодировщиков вариационными методами

Детерминированные автокодировщики состоят из кодера, который проецирует данные из пространства x в скрытое пространство z меньшей размерности, и декодера, который, в свою очередь, проецирует данные из пространства z в пространство $\hat{x}$ с целью не допустить потерю информации или уменьшить ошибку реконструкции, что означает сохранение близкими x и $\hat{x}$, например в смысле евклидова расстояния. В этом смысле *анализ главных компонент* на основе сингулярного раз-

ложения $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ можно рассматривать как линейный кодер, где декодер — это простое *транспонирование кодирующей матрицы*. Функции кодирования и декодирования могут быть нелинейными и/или нейросетями.

В случае детерминированных автокодировщиков мы не можем использовать декодер в качестве генератора данных. По крайней мере, если мы сделаем это, то из латентного пространства z нам придется выбрать некоторое z и применить к нему функцию декодера. Вряд ли мы получим хоть одно $\hat{x}$, близкое к тому, как выглядят ожидаемые данные x, если только не выбран z, соответствующий закодированному x в результате чрезмерной подгонки. Нам нужна регуляризация, которая предоставит некоторый контроль над пространством z, что позволит избежать перебора и использовать автокодировщики в качестве генератора данных. Для этого нужно перейти от детерминированного к вероятностному автокодированию.

Вариационные автокодировщики — это вероятностные автокодировщики, которые выдают не отдельные пункты, а распределения вероятностей в латентном пространстве z. Более того, в процессе обучения функция потерь включает дополнительный член регуляризации, который управляет распределением в латентном пространстве. Поэтому функция потерь для вариационных автокодировщиков содержит член реконструкции (например, среднее квадратичное расстояние) и регуляризационный член для управления выводимым кодером распределением. Регуляризационный член может представлять собой KL-дивергенцию гауссова распределения, т. к. в основе лежит предположение, что простые вероятностные модели лучше всего описывают обучающие данные. Другими словами, сложные взаимосвязи могут быть вероятностно простыми. Здесь нужно быть осторожными, поскольку это вносит погрешность: если простое предположение о распределении данных в латентной переменной слишком слабое, это может стать недостатком. То есть когда предположение о предшествующем распределении или предположение о приближенном апостериорном распределении слишком слабо (даже при идеальном алгоритме оптимизации и бесконечном количестве обучающих данных), разрыв между оценкой $\mathscr{L}$ и истинным логарифмическим правдоподобием может привести к тому, что $p_{\text{модель}}$ обучится совершенно иному распределению, отличному от истинного $p_{\text{данные}}$.

С точки зрения математики, мы максимизируем нижнюю границу $\mathscr{L}$ на логарифмическом правдоподобии данных. В науке вариационные методы определяют нижние границы на функционал энергии, который нужно максимизировать, или верхние границы на функционал энергии, который нужно минимизировать. Такие границы обычно легче получить, и они имеют легкодоступные алгоритмы оптимизации, даже если логарифмическое правдоподобие не является таковым. В то же время они дают отличные оценки для искомых оптимальных значений:

$$\mathscr{L}\left(\vec{x}, \vec{\theta}\right) \leq \log p_{\text{модель}}\left(\vec{x}, \vec{\theta}\right).$$

С помощью вариационных методов можно добиться отличного правдоподобия, хотя, по субъективной оценке, сгенерированные выборки считаются менее качествен-

ными. Они также считаются более сложными для оптимизации по сравнению с полностью видимыми сетями доверия. Более того, их математика тоже считается более сложной, чем математика полностью видимых сетей доверия и генеративно-состязательных сетей (о них речь пойдет ниже).

Явно непрослеживаемая плотность: аппроксимация машины Больцмана с помощью цепи Маркова

Машины Больцмана (созданные в 1980-х годах) — это семейство генеративных моделей, которые используют цепи Маркова для обучения генеративных моделей. Такая технология выборки обходится дороже, чем простая выборка мини-пакета из набора данных для оценки функции потерь. Цепи Маркова рассматриваются в контексте обучения с подкреплением в *главе 11*. В плане генерации данных у них много недостатков, из-за которых они вышли из употребления: высокие вычислительные затраты, непрактичность и низкая эффективность при расширении на более высокие размерности, медленная сходимость и отсутствие четкого способа узнать, насколько сошлась модель, даже если теоретически она должна сойтись. Методы цепей Маркова не подойдут для решения таких задач, как генерация ImageNet.

Цепь Маркова имеет оператор перехода q, который кодирует вероятность перехода из одного состояния системы в другое. Он должен быть определен в явном виде. Можно генерировать выборки данных путем многократного построения выборки $x' \sim q(x' \mid x)$, последовательно обновляя x' в соответствии с оператором перехода q. Такой последовательный характер генерации является еще одним недостатком по сравнению с одношаговой генерацией. Методы цепей Маркова иногда могут гарантировать, что x' в конечном итоге сведется к выборке из $p_{\text{модель}}(x)$, даже если сходимость окажется медленной.

В некоторых моделях, таких как глубокие машины Больцмана, применяются как цепь Маркова, так и вариационные аппроксимации.

Неявная плотность — цепь Маркова: генеративная стохастическая сеть

Генеративная стохастическая сеть (Бенджио и др., 2014; **https://oreil.ly/DJI78**) не определяет явно функцию плотности, но использует оператор перехода цепи Маркова, который косвенно взаимодействует с $p_{\text{модель}}(x)$, осуществляя выборку из обучающих данных. Для того чтобы получить выборку $p_{\text{модель}}(x)$, нужно запустить оператор несколько раз. Эти методы по-прежнему страдают от недостатков методов цепи Маркова, о которых упоминалось в предыдущем разделе.

Неявная плотность — направление: генеративно-состязательные сети

В настоящее время наиболее популярными генеративными моделями являются:

- полностью видимые глубокие сети доверия, такие как PixelCNN, WaveNet и их разновидности;
- вариативные автокодировщики, содержащие вероятностную архитектуру "кодер — декодер";
- генеративно-состязательные сети, которые привлекли большое внимание научного сообщества благодаря простоте своей концепции и превосходному качеству генерируемых образцов. Рассмотрим их поближе.

Генеративно-состязательные сети (**https://oreil.ly/2GKF3**) были представлены в 2014 году Яном Гудфеллоу и др. Математика, которая используется в этих сетях, представляет собой замечательную смесь теории вероятностей и теории игр.

У генеративно-состязательных сетей отсутствуют некоторые недостатки, свойственные другим генеративным моделям.

- Одновременная параллельная генерация выборок, а не подача нового пиксела обратно в сеть для предсказания предыдущего, как в PixelCNN.
- Функция генератора имеет мало ограничений. Это является преимуществом по сравнению с машинами Больцмана, для которых лишь немногие распределения вероятностей допускают приемлемую выборку цепей Маркова, и по сравнению с нелинейным анализом независимых компонент, для которого генератор должен быть инвертируемым, а скрытые переменные z должны иметь ту же размерность, что и выборки x.
- Генеративно-состязательным сетям не требуются цепи Маркова. Это является преимуществом по сравнению с машинами Больцмана и генеративными стохастическими сетями.
- В то время как вариационные автокодировщики могут никогда не сойтись с порождающим данные истинным распределением, если они предполагают слишком слабые *априорные* или *апостериорные распределения*, генеративные состязательные сети сходятся с истинным $p_{\text{данные}}$, если имеется бесконечное количество обучающих данных и модель достаточно большая. Более того, генеративно-состязательным сетям не требуются вариационные ограничения, а конкретные семейства моделей, используемые в рамках генеративно-состязательных сетей, как известно, считаются универсальными аппроксиматорами. Таким образом, уже известно, что генеративные состязательные сети являются асимптотически согласованными. В то же время предполагается, что некоторые вариационные автокодировщики являются асимптотически последовательными, но это еще предстоит доказать.

Недостаток генеративно-состязательных сетей заключается в том, что для их обучения требуется найти равновесие Нэша в игре, что сложнее, чем простая оптими-

зация целевой функции. Кроме того, решение имеет тенденцию быть численно неустойчивым. В 2015 году Алек Рэдфорд и др. в работе "Неконтролируемое обучение представлений с помощью глубоких сверточных генеративно-состязательных сетей" (**https://oreil.ly/ARUUD**) исправили эту ситуацию. Такой подход позволил получить более стабильные модели.

В процессе обучения генеративно-состязательные сети создают игру между двумя отдельными сетями — сетью генератора и сетью дискриминатора, которая пытается классифицировать выборки генератора как порожденные либо истинным распределением $p_{\text{данные}}(x)$, либо моделью $p_{\text{модель}}(x)$. Функции потерь этих двух сетей связаны друг с другом, так что дискриминатор сообщает о расхождении между двумя распределениями, а генератор соответствующим образом настраивает свои параметры до точного воспроизведения истинного распределения данных (в теории), чтобы классификации дискриминатора оказались не лучше случайных предположений.

Сеть генератора стремится максимизировать вероятность присвоения дискриминатором неверной метки при классификации, независимо от того, откуда взята выборка — из обучающих данных или из модели, в то время как сеть дискриминатора стремится минимизировать эту вероятность. В этом смысл игры с нулевой суммой для двух игроков, где выигрыш одного игрока — это проигрыш другого. В итоге вместо обычной задачи максимизации или минимизации мы решаем задачу о минимаксе. Уникальное решение действительно существует.

Принцип работы генеративно-состязательной сети

Придерживаясь цели — обучиться распределению вероятностей генератора $p_g\left(\vec{x};\vec{\theta}\right)$ по данным, опишем процесс обучения генеративно-состязательных сетей:

1. Начинаем со случайной выборки $\vec{z}$ из предварительного распределения вероятностей $p_z(\vec{z})$, которое может быть просто равномерным случайным шумом для каждой компоненты $\vec{z}$.
2. Начинаем также со случайной выборки $\vec{x}$ из обучающих данных, т. е. это выборка из вероятностного распределения $p_{\text{данные}}(\vec{x})$, которому пытается обучиться генератор.
3. Применяем к $\vec{z}$ детерминированную функцию $G\left(\vec{z},\vec{\theta}_g\right)$, представляющую генеративную нейросеть. Параметры $\vec{\theta}_g$ — это параметры, которые нужно подстраивать методом обратного распространения ошибки, пока выход $G\left(\vec{z},\vec{\theta}_g\right)$ не будет похож на выборки из обучающего набора данных.
4. Передаем выход $G\left(\vec{z},\vec{\theta}_g\right)$ в другую детерминированную функцию *D*, представляющую *дискриминантную нейросеть*. Теперь мы имеем новый выход $D\left(G\left(\vec{z},\vec{\theta}_g\right),\vec{\theta}_d\right)$, представляющий собой число, более близкое к единице или

нулю, что означает, откуда эта выборка — из генератора или из обучающих данных. Таким образом, для данного входа выход $D\left(G\left(\vec{z}, \vec{\theta}_g\right), \vec{\theta}_d\right)$ должен возвращать от генератора число, близкое к единице. Параметры $\vec{\theta}_d$ — это параметры, которые нужно подстраивать методом обратного распространения ошибки, пока D не будет возвращать неправильную классификацию примерно в половине случаев.

5. Передаем в D также выборку $\vec{x}$ из обучающих данных для оценки $D\left(\vec{x}, \vec{\theta}_d\right)$. Для этого входного сигнала $D\left(\vec{x}, \vec{\theta}_d\right)$ должен возвращать число, близкое к нулю.

6. Какой будет функция потерь для этих двух сетей, которая имеет в своей формуле оба набора параметров $\vec{\theta}_g$ и $\vec{\theta}_d$, а также отобранные векторы $\vec{x}$ и $\vec{z}$? Дискриминантная функция D стремится понять ее правильно для обоих типов входов — $\vec{x}$ и $G\left(\vec{z}, \vec{\theta}_g\right)$. Поэтому параметры $\vec{\theta}_d$ нужно отобрать таким образом, чтобы в случае $G\left(\vec{z}, \vec{\theta}_g\right)$ большому значению присваивалось число, близкое к 1, а в случае $\vec{x}$ большому значению соответствовало число, близкое к 0. В обоих случаях можно использовать отрицательную часть логарифмической функции, т. к. эта функция велика вблизи 0 и мала вблизи 1. Поэтому D требуются параметры $\vec{\theta}_d$, которые максимизируют:

$$\mathbb{E}_{\vec{x} \sim p_{\text{данные}}(\vec{x})}\left[\log D\left(\vec{x}, \vec{\theta}_d\right)\right] + \mathbb{E}_{\vec{z} \sim p_z(\vec{z})}\left[\log\left(1 - D\left(G\left(\vec{z}, \vec{\theta}_g\right), \vec{\theta}_d\right)\right)\right].$$

В то же время G требуются параметры $\vec{\theta}_g$, минимизирующие $\log\left(1 - D\left(G\left(\vec{z}, \vec{\theta}_g\right), \vec{\theta}_d\right)\right)$. В совокупности D и G участвуют в минимаксной игре для двух игроков с функцией ценности $V(D, G)$:

$$\min_{\vec{\theta}_g} \max_{\vec{\theta}_d} \mathbb{E}_{\vec{x} \sim p_{\text{данные}}(\vec{x})}\left[\log D\left(\vec{x}, \vec{\theta}_d\right)\right] + \mathbb{E}_{\vec{z} \sim p_z(\vec{z})}\left[\log\left(1 - D\left(G\left(\vec{z}, \vec{\theta}_g\right), \theta_d\right)\right)\right].$$

Это очень простая математическая структура, в которой настройка сети дискриминаторов позволяет приблизиться к истинному распределению данных, не определяя его явно и ничего не предполагая о нем.

В заключение отметим, что генеративно-состязательные сети весьма перспективны для многих приложений. Одним из примеров является существенное улучшение полуконтролируемого обучения с их помощью. Так, в "Учебном пособии NIPS 2016: генеративно-состязательные сети" (Гудфеллоу Я., 2016; **https://oreil.ly/isXsx**) находим:

"Представляем метод полуконтролируемого обучения с помощью генеративно-состязательных сетей, в котором дискриминатор производит дополнительный выход, указывающий на метку входа. Такой подход позволяет получить лучшие результаты на MNIST, SVHN и CIFAR-10 в условиях очень малого количества разме-

ченных выборок. Например, в MNIST достигнута точность 99,14% при использовании всего 10 размеченных выборок на класс с полносвязной нейросетью — результат, довольно близкий к известным лучшим результатам полностью контролируемых методов с использованием всех 60 000 помеченных выборок. Это выглядит весьма многообещающим, т. к. получить размеченные выборки может оказаться в действительности довольно дорогостоящим предприятием".

Еще одно весьма перспективное применение генеративно-состязательных сетей (и машинного обучения в целом) — моделирование данных для физики высоких энергий. Остановимся на этом подробнее.

Пример: машинное обучение и генеративные сети для физики высоких энергий

Последующее рассуждение опирается на материалы семинара 2020 года "Машинное обучение в реактивной физике" (**https://oreil.ly/Uy3Ah**) и две публикации — "Глубокое обучение и его применение в физике БАК"[1] (Гест Д. и др., 2018; **https://oreil.ly/4nlAj**) и "Графовые генеративно-состязательные сети для генерации разреженных данных в физике высоких энергий" (Кансал Р. и др., 2021; **https://oreil.ly/EHGrK**).

До начавшейся в 2012 году революции глубокого обучения область физики высоких энергий традиционно опиралась в анализах и вычислениях на физические соображения и человеческую интуицию, *расширяемые деревья решений*, ручное конструирование признаков данных и снижение размерности, а также традиционный статистический анализ. Эти методы, хотя и являются глубокими, конечно же, далеки от оптимальных, их трудно автоматизировать или расширить на более высокие измерения. Некоторые исследований показали, что традиционные неглубокие сети, основанные на *высокоуровневых* признаках, инспирированных физикой, превосходят глубокие сети, опирающиеся на более высокоразмерные *низкоуровневые* признаки, которые подвергаются меньшей предварительной обработке. Многие области анализа данных *Большого адронного коллайдера* (**https://oreil.ly/WKudv**) долгое время страдали от недостаточной инженерии признаков и заслуживают пересмотра. Таким образом, область физики высоких энергий является благоприятной почвой для применения машинного обучения. И здесь можно наблюдать значительный прогресс. В этой области используется несколько методов машинного обучения, включая искусственные нейросети, оценка плотности ядра, метод опорных векторов, генетические алгоритмы, расширяемые деревья решений, случайные леса и генеративные сети.

Экспериментальная программа Большого адронного коллайдера исследует самые фундаментальные вопросы современной физики: природа массы, размерность пространства, объединение фундаментальных сил, природа частиц темной материи,

[1] БАК — Большой адронный коллайдер. — *Прим. ред.*

точная настройка Стандартной модели (**https://oreil.ly/SMljD**). Одна из главных целей — изучить самую фундаментальную структуру материи. Частично это подразумевает поиск и изучение экзотических частиц, таких как *топ-кварк* и *бозон Хиггса*, образующихся при столкновениях на таких ускорителях, как Большой адронный коллайдер. К конкретным задачам и проблемам относятся реконструкция масс, субструктура струй, классификация вкусов струй. Например, можно идентифицировать струи из тяжелых (c, b, t) или легких (u, d, s) кварков, глюонов, а также W-, Z-, H-бозонов.

Проведение экспериментов с частицами высоких энергий и сбор полученных данных требуют колоссальных затрат. Собранные данные огромны с точки зрения количества столкновений и сложности каждого столкновения. Кроме того, большинство событий в ускорителях не приводит к образованию интересных частиц (сигнальные частицы по сравнению с фоновыми). Так как сигнальные частицы встречаются редко, необходимы высокие скорости передачи данных. Например, детекторы Большого адронного коллайдера оснащены $O(108)$ датчиками для регистрации большого количества частиц, образующихся после каждого столкновения. В связи с этим крайне важно извлекать максимум информации из экспериментальных данных (модели регрессии и классификации), точно выбирать и идентифицировать события для эффективных измерений, а также создавать надежные методы моделирования новых данных, аналогичных данным, полученным в ходе экспериментов (генеративные модели). Данные физики высоких энергий характеризуются высокой размерностью, а также сложной топологией многих сигнальных событий.

Рассматриваемая тема связана с нашей главой природой столкновений и взаимодействия их продуктов с детекторами Большого адронного коллайдера. Это квантово-механические явления, и, следовательно, наблюдения, возникающие в результате конкретного взаимодействия, в основе своей вероятностны. Поэтому анализ полученных данных нужно проводить в статистических и вероятностных выражениях.

В этой главе мы задаемся целью узнать распределение вероятностей $p\left(\vec{\theta} \mid \vec{x}\right)$ параметров модели с учетом наблюдаемых данных. Если данные имеют достаточно низкую размерность, например менее пяти измерений, то задача оценки неизвестной статистической модели по смоделированным выборкам не вызовет особого труда при использовании гистограммы или оценки плотности на основе ядра. Но из-за проклятия размерности не так-то легко перенести эти простые методы на более высокие измерения. Для оценки исходной функции плотности вероятности потребуется N выборок в одном измерении и $O\left(N^d\right)$ — в d измерениях. В итоге, если размерность данных превысит 10 или около того, применение наивных методов оценки распределения вероятностей становится непрактичным или даже невозможным, поскольку для этого потребуется запредельное количество вычислительных ресурсов.

Специалисты в области физики высоких энергий традиционно справлялись с проклятием размерности, уменьшая размерность данных с помощью ряда шагов, кото-

рые действовали как на отдельные события столкновений, так и на коллекции событий. Эти устоявшиеся подходы основывались на созданных вручную специфических особенностях в данных достаточно малого количества для оценки неизвестного распределения вероятности $p(x|\theta)$ при помощи выборок, сгенерированных средствами моделирования. Очевидно, что такого традиционного похода недостаточно в силу обилия сложных данных и тончайших характеристик в редкой области новой физики. Машинное обучение избавляет от необходимости вручную конструировать признаки и снижать размерность, в результате чего можно было упустить важную информацию в высокоразмерных данных нижнего уровня. Более того, структура данных нижнего уровня, полученных непосредственно с датчиков, отлично вписывается в такие популярные модели нейросетей, как сверточные и графовые нейросети, например *проективная башенная структура калориметров* практически во всех современных детекторах физики высоких энергий аналогична пикселам изображения.

Однако, несмотря на успех метода на основе изображений, реальная геометрия детектора не является идеально регулярной, поэтому для представления изображений струй требуется предварительная обработка данных. Кроме того, изображения струй часто бывают сильно разрежены. Проблему неправильной геометрии и разреженности можно решить применением в моделировании данных о частицах *графовых сверточных сетей* вместо обычных сверточных сетей. Графовые сверточные сети расширяют возможности использования сверточных нейросетей с непостоянными выборками данных. Они способны работать с разреженными, инвариантными к перестановкам данными со сложной геометрией. Мы обсудим графовые сети в *главе 9*. Они всегда содержат узлы, ребра и матрицу, кодирующую связи в графе, которая называется *матрицей смежности*. В контексте физики высоких энергий частицы струи представляют собой узлы графа, а ребра кодируют, насколько близко расположены частицы в изученной матрице смежности. В физике высоких энергий графовые сети успешно применяются в решении задач классификации, реконструкции и генерации.

В этой главе мы рассматриваем генеративные модели, или генерацию данных, похожих на заданный набор данных. Большое значение имеет генерация или моделирование данных, которые совпадут с экспериментальными данными, собранными в физике высоких энергий. Авторы публикации "Графовые генеративно-состязательные сети для генерации разреженных данных в физике высоких энергий" (**https://oreil.ly/eVkOP**), используя фреймворк генеративно-состязательных сетей, разработали графовые генеративные модели для моделирования наборов разреженных данных, подобных получаемым на Большом адронном коллайдере в ЦЕРНе (**https://home.cern/**).

Авторы иллюстрируют свой подход обучением и генерацией разреженных представлений изображений рукописных цифр MNIST и струй частиц в протон-протонных столкновениях, подобных тем, что происходят на Большом адронном коллайдере. Модель успешно генерирует разреженные изображения цифр MNIST и данные о струях частиц. Для количественной оценки соответствия реальных и сге-

нерированных данных авторы используют две метрики: графовое расстояние Фреше (**https://oreil.ly/LHron**) и расстояние Вассерштейна (**https://oreil.ly/V2UVS**) на уровне признаков частиц и струй.

Другие генеративные модели

Мы обсудили генеративные модели (по состоянию на 2022 год), но глава будет неполной, если мы не рассмотрим такие модели, как наивная байесовская модель, модель гауссовой смеси и машина Больцмана. Есть еще много других. В связи с этим Ян Лекун (вице-президент и ведущий исследователь в области ИИ компании Meta Platforms, Inc.) предлагает свой взгляд (**https://oreil.ly/fSGCz**) на некоторые модели:

> "В 2000-х годах исследователи в области распознавания речи, компьютерного зрения и обработки естественного языка[2] были одержимы идеей точного представления неопределенности. Это вызвало шквал работ по вероятностным генеративным моделям, таким как скрытые марковские модели в распознавании речи, марковские случайные поля и модели созвездий в компьютерном зрении, а также вероятностные тематические модели в NLP, например, с использованием латентного анализа Дирихле. На семинарах по компьютерному зрению велись дебаты о генеративных и дискриминационных моделях. Предпринимались героические, но пока безуспешные попытки построить системы распознавания объектов с помощью непараметрических байесовских методов. Многие из них опирались на предыдущий опыт работы с байесовскими сетями, факторными графами и другими графовыми моделями. Так мы узнали об экспоненциальном семействе, распространении доверия, зацикленном распространении доверия, вариационном выводе, процессе китайского ресторана, процессе индийского буфета и т. д. Но практически ни одна из этих работ не затрагивала проблему обучения представлений. Предполагалось, что признаки заданы. Структура графовой модели с ее латентными переменными также считалась заданной. Все, что оставалось, — вычислить некое логарифмическое правдоподобие путем линейной комбинации признаков, а затем, используя один из перечисленных выше сложных методов вывода, получить маргинальные распределения по неизвестным переменным, одной из которых является ответ, например категория. По сути, экспоненциальное семейство в значительной степени означает поверхностное, неглубокое, когда логарифмическое правдоподобие можно выразить как линейно параметризованную функцию признаков (или их простых комбинаций). Изучение параметров модели рассматривалось как еще одна задача вариационного вывода. Интересно отметить, что почти ничего из вышесказанного не имеет отношения к современным системам распознавания речи, зрения и NLP. Как оказалось, решение проблемы обучения иерархическим представлениям и сложным функциональным зависимостям было гораздо более важным вопросом, чем возможность выполнять точный вероятностный вывод с помощью неглубоких моделей. Однако это не значит, что точные вероятностные выводы не нужны".

[2] Natural language processing (NLP). — *Прим. ред.*

Далее он продолжает в том же духе:

> "Генеративно-состязательные сети отлично подходят для создания красивых картинок (хотя их и вытесняют диффузионные модели, или „многоступенчатые шумоподавляющие автокодировщики“, как я их называю), но в распознавании и обучении представлениям они оказались большим разочарованием".

Тем не менее все эти модели могут многое дать с точки зрения математики. По моему опыту, мы понимаем и запоминаем математику на гораздо более глубоком уровне, когда видим, как она развивается и используется в конкретных целях, а не просто тренирует нейроны мозга. Многие математики утверждают, что испытывают удовольствие, доказывая теории, которым еще предстоит найти применение. Лично я не отношусь к их числу.

Наивная байесовская модель классификации

Наивная байесовская модель — это довольно простая модель классификации, которую также можно использовать в качестве генеративной модели, т. к. в итоге она вычисляет совместное распределение вероятностей $p(\vec{x}, y_k)$ данных, чтобы определить их классификацию. Обучающие данные содержат признаки $\vec{x}$ и метки y_k. Поэтому можно использовать наивную байесовскую модель для генерации новых точек данных с метками, осуществляя выборку из этого совместного распределения вероятностей. Наивная байесовская модель нацелена на вычисление вероятности класса y_k с учетом признаков данных $\vec{x}$, что является условной вероятностью $p(y_k \mid \vec{x})$. Такое вычисление дорого обойдется в случае данных с большим количеством признаков (высокоразмерных $\vec{x}$), поэтому мы будем использовать правило Байеса и применим обратную условную вероятность, которая, в свою очередь, ведет к совместному распределению вероятностей. То есть:

$$p(y_k \mid \vec{x}) = \frac{p(y_k)p(\vec{x} \mid y_k)}{p(\vec{x})} = \frac{p(\vec{x}, y_k)}{p(\vec{x})}.$$

Наивная байесовская модель делает довольно сильное наивное предположение, которое на практике работает лучше, чем можно было бы ожидать, о том, что признаки данных взаимно независимы при условии метки класса y_k. Это предположение позволяет существенно упростить совместное распределение вероятностей в числителе, особенно при его разложении на произведение условных вероятностей одной переменной. Предположение о независимости признаков в зависимости от метки класса y_k означает:

$$p(x_i \mid x_{i+1}, x_{i+2}, ..., x_n, y_k) = p(x_i \mid y_k).$$

Таким образом, совместное распределение вероятностей складывается из:

$$p(\vec{x}, y_k) = p(x_1 \mid x_2, ..., x_n, y_k)\, p(x_2 \mid x_3, ..., x_n, y_k) \cdots p(x_n \mid y_k)\, p(y_k) =$$
$$= p(x_1 \mid y_k)\, p(x_2 \mid y_k) \cdots p(x_n \mid y_k)\, p(y_k).$$

Теперь нам не составит труда на основе обучающих данных оценить вероятности отдельных признаков, обусловленные каждой категорией данных. Аналогичным образом можно оценить вероятность каждого класса $p(y_k)$ из обучающих данных или предположить, что классы равновероятны, так что

$$p(y_k) = 1/(\text{количество классов}).$$

Важно отметить, что в общем случае генеративные модели находят совместное распределение вероятностей $p(\vec{x}, \vec{y}_k)$ между метками y_k и данными $\vec{x}$. Модели классификации, в свою очередь, вычисляют условные вероятности $p(y_k \mid \vec{x})$. Они сосредоточены на вычислении границ принятия решений между различными классами данных, возвращая класс y_k с наибольшей вероятностью. Таким образом, наивный байесовский классификатор возвращает метку y_* с наибольшим значением $p(y_k \mid \vec{x})$, равным наибольшему значению $p(x_1 \mid y_k)p(x_2 \mid y_k)\cdots p(x_n \mid y_k)p(y_k)$.

Модель гауссовой смеси

В модели гауссовой смеси предполагается, что все точки данных генерируются из смеси конечного числа гауссовых распределений с неизвестными параметрами (средними и ковариационными матрицами). Можно считать, что модели смесей похожи на кластеризацию *k* средних, только здесь включена информация о центрах кластеров (средние гауссианы), а также форму распределения данных в каждом кластере (определяется ковариацией гауссианов). Для определения количества кластеров в данных в моделях гауссовых смесей иногда используется байесовский информационный критерий (**https://oreil.ly/YjP4F**). Также можно ограничить модель, чтобы контролировать ковариации различных гауссианов в смеси: полных, связанных, диагональных, связанных диагональных и сферических (рис. 8.3).

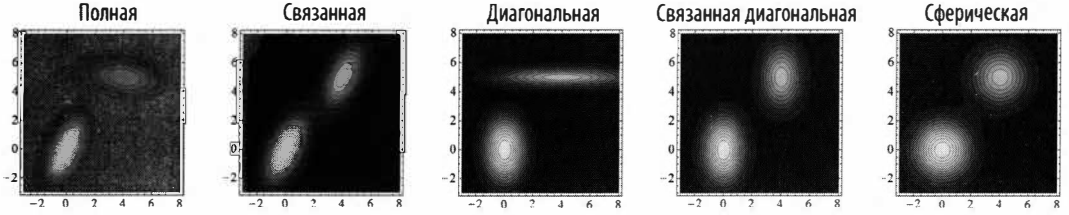


Рис. 8.3. Типы ковариации гауссовой смеси
(источник изображения: **https://oreil.ly/BMF7a**)

Наконец, необходимо максимизировать правдоподобие данных для оценки неизвестных параметров смеси (средние значения и записи ковариационных матриц).

Метод максимального правдоподобия теряет свою актуальность, если в данных присутствуют латентные, или скрытые, переменные (переменные, которые не измеряются и не наблюдаются напрямую). В этом случае для оценки максимального правдоподобия используется *алгоритм максимизации ожиданий* (expectation

maximization — EM-алгоритм, **https://oreil.ly/QpWUo**). Он строится следующим образом:

1. Оцениваем значения скрытых переменных, создав функцию ожидания логарифмического правдоподобия с использованием текущих оценок неизвестных параметров.
2. Выполняем оптимизацию — вычисляем новые параметры, максимизирующие ожидаемое логарифмическое правдоподобие, оцененное на шаге 1.
3. Повторяем шаги 1 и 2 до сходимости.

Мы видим, как модели гауссовой смеси можно применять в качестве кластерных, генеративных или классификационных моделей. В случае кластеризации это основная часть построения модели. При генерации — выборка новых точек данных из смеси после вычисления неизвестных параметров путем максимизации ожиданий. В случае классификации, получив новую точку данных, модель относит ее к гауссиану, к которому она принадлежит с наибольшей вероятностью.

Эволюция генеративных моделей

В этом разделе мы поведаем историю, которая способствовала завершению спящего периода нейросетей и привела в конечном итоге к появлению таких современных вероятностных моделей глубокого обучения, как вариативные автокодировщики, полностью видимые глубокие сети доверия и генеративно-состязательные сети. Перед нами проходит путь от *сетей Хопфилда* к *машинам Больцмана* и *ограниченным машинам Больцмана*. Я испытываю особую симпатию к этим моделям, т. к. помимо исторической ценности и изучения совместного распределения вероятностей признаков данных путем сборки сети базовых вычислительных единиц, они используют математический механизм особенно искусной высокоразвитой области *статистической механики* — моей первоначальной области исследований.

В статистической механике распределения вероятностей определяются в выражениях функций энергии.

Вероятность того, что мы найдем систему в определенном состоянии $\vec{x}$, зависит от ее энергии $E(\vec{x})$ в этом состоянии. Точнее, высокоэнергетические состояния менее вероятны, что проявляется в отрицательном знаке в экспоненте в формуле:

$$p(\vec{x}) = \frac{\exp(-E(\vec{x}))}{Z}.$$

Экспоненциальная функция обеспечивает положительное значение p, а *функция секционирования* Z в знаменателе гарантирует, что сумма (или интеграл при непрерывной переменной x) $p(\vec{x})$ общих состояний $\vec{x}$ равна 1, что делает p достоверным распределением вероятностей. Модели машинного обучения, определяющие таким образом совместные распределения вероятностей, называются, по понятным

причинам, *моделями на основе энергии*. Они отличаются тем, как они назначают энергию в каждом состоянии, т. е. конкретной формулой, которую они используют для $E(\vec{x})$, что, в свою очередь, влияет на формулу для функции секционирования Z. В формуле для $E(\vec{x})$ содержатся параметры модели $\vec{\theta}$, которые требуется вычислить по данным *методом максимального правдоподобия*. По сути, лучше выразить зависимость p, E и Z от $\vec{\theta}$ в формуле совместного распределения вероятностей:

$$p(\vec{x}, \vec{\theta}) = \frac{\exp\left(-E(\vec{x}, \vec{\theta})\right)}{Z(\vec{\theta})}.$$

В большинстве случаев вычислить замкнутую формулу для функции секционирования Z невозможно, что весьма затрудняет применение метода максимального правдоподобия. Точнее, в процессе максимизации логарифмического правдоподобия нужно вычислить его градиент, в том числе по параметру $\vec{\theta}$, так что мы вынуждены вычислять градиент функции секционирования Z по $\vec{\theta}$. В этих вычислениях часто встречается такая величина:

$$\nabla_{\vec{\theta}} \log Z(\vec{\theta}) = \mathbb{E}_{\vec{x} \sim p(\vec{x})}\left(\nabla_{\vec{\theta}} \log\left(\text{нумератор}(\vec{x}, \vec{\theta})\right)\right),$$

где в нашем случае числитель в формуле совместного распределения вероятности на основе энергии равен $\exp\left(-E(\vec{x}, \vec{\theta})\right)$, хотя в разных моделях он может отличаться.

В случаях когда функция секционирования неразрешима, приходится прибегать к таким методам аппроксимации, как стохастическое максимальное правдоподобие и контрастная дивергенция. Другие методы позволяют обойтись без аппроксимации функции секционирования и вычислять условные вероятности без знания этой функции. В них используются отношения условных вероятностей, а также отношения совместного распределения вероятностей на основе энергии, что эффективно отменяет функцию секционирования. Такие методы включают в себя сопоставление оценок, сопоставление отношений и сопоставление оценок шумоподавления.

Другие методы, такие как контрастная оценка шума, выборка по значимости, выборка по мосту или их комбинация, основанная на сильных сторонах каждого из них, аппроксимируют не логарифм градиента, а непосредственно функцию секционирования.

Мы не будем касаться этих методов, но отсылаем заинтересованного читателя к книге Яна Гудфеллоу и др. "Глубокое обучение" (2016 г., **https://oreil.ly/ViEzk**).

Вернемся к сетям Хопфилда и машинам Больцмана. Они представляют собой переход к глубоким нейросетям, обучаемым методом обратного распространения ошибки и лежащие в основе новейших детерминированных и вероятностных моделей глубокого обучения. Эти методы формируют оригинальный (для нейронов) *коннекционистский подход* к обучению произвольных распределений вероятно-

стей — сначала только бинарным векторам нулей и единиц, а затем векторам с произвольными значениями действительных чисел.

Сети Хопфилда

Сети Хопфилда (Hopfield nets) используют преимущества изящной математики статистической механики, отождествляя состояния нейронов искусственной нейросети с состояниями элементов физической системы. Несмотря на то что в конечном итоге сети Хопфилда оказались дорогими в плане вычислений и имеют ограниченное практическое применение, они заложили основу современной эпохи нейросетей, и их стоит изучить хотя бы для того, чтобы оценить историческую эволюцию ИИ. Сети Хопфилда не имеют скрытых блоков, и все их (видимые) блоки соединены друг с другом. Каждая единица может находиться во *включенном* или *выключенном* состоянии (единица или ноль), а все вместе они кодируют информацию обо всей сети (или системе).

Машина Больцмана

Машина Больцмана — это сеть Хопфилда, но с добавлением скрытых блоков. Мы уже знакомы со структурой входных и скрытых блоков в нейросетях, поэтому их не нужно объяснять, но именно с них все и началось. Как и в сети Хопфилда, входные, а также скрытые единицы являются двоичными, а состояния — либо 0, либо 1 (в современных версиях реализованы единицы, принимающие не только двоичные значения, но и значения действительных чисел).

Все машины Больцмана имеют неразрешимую функцию секционирования, поэтому мы аппроксимируем градиент максимального правдоподобия при помощи методов, рассмотренных во введении к этому разделу.

Для обучения машин Больцмана используется только вычислительно интенсивное *семплирование по Гиббсу*. Фамилия Гиббса неоднократно встречается в области статистической механики. Семплирование по Гиббсу дает несмещенные оценки весов сети, но эти оценки имеют высокую дисперсию. В целом существует компромисс между смещением и дисперсией, который подчеркивает преимущества и недостатки методов на их основе.

Ограниченная машина Больцмана (явная плотность и трудноразрешимость)

Машины Больцмана имеют очень низкую скорость обучения из-за множества взаимосвязей между видимыми и скрытыми слоями (вспомним довольно запутанную систему обратного распространения ошибки). Из-за медленного обучения их нельзя использовать в решении практических задач. Здесь приходят на помощь *ограниченные машины Больцмана*, которые ограничивают связи только между разными слоями. То есть в ограниченной машине Больцмана внутри каждого слоя нет связей, благодаря чему можно обновлять одновременно все блоки в каждом слое.

Таким образом, можно собирать статистику совпадений для двух связанных слоев, поочередно обновляя все блоки в каждом слое. На практике можно добиться большей экономии за счет минимальных процедур выборки, например контрастной дивергенции.

Условная независимость

Отсутствие связей внутри каждого слоя означает, что состояния всех блоков в скрытом слое не зависят друг от друга, но зависят от состояний блоков в предыдущем слое. Другими словами, учитывая состояния блоков предыдущего слоя, можно утверждать, что состояние каждого скрытого блока не зависит от состояний других блоков в скрытом слое. Эта условная независимость позволяет факторизовать совместную вероятность состояния скрытого слоя $p\left(\vec{h}\left|\overrightarrow{h_{\text{предыдущий}}}\right.\right)$ как произведение условных вероятностей состояний отдельных скрытых блоков. Например, если в скрытом слое имеются три блока, то $p\left(\vec{h}\left|\overrightarrow{h_{\text{предыдущий}}}\right.\right)=p\left(h_1\left|\overrightarrow{h_{\text{предыдущий}}}\right.\right)p\left(h_2\left|\overrightarrow{h_{\text{предыдущий}}}\right.\right)p\left(h_3\left|\overrightarrow{h_{\text{предыдущий}}}\right.\right)$. Верно и обратное: состояния блоков предыдущего слоя условно независимы друг от друга, учитывая состояния текущего слоя. Эта условная независимость означает, что вместо того, чтобы итеративно обновлять блоки в течение длительного времени, можно делать выборку их состояний.

Универсальная аппроксимация

Ограниченные связи в ограниченных машинах Больцмана позволяют складывать их, т. е. иметь несколько скрытых слоев, способных извлекать более сложные признаки. Теперь мы можем увидеть, как постепенно формировалась архитектура современной многослойной искусственной нейросети. Напомним, что в *главе 4* мы обсуждали универсальную аппроксимацию нейросетей для широкого спектра детерминированных функций. В этой главе мы хотим, чтобы наши сети представляли (или обучались) не детерминированные функции, а совместные распределения вероятностей. В 2008 году Ле Ру и Бенджио доказали, что машины Больцмана могут аппроксимировать с произвольной точностью любое дискретное распределение вероятностей. Это также применимо к ограниченным машинам Больцмана. Более того, в определенных мягких условиях каждый дополнительный скрытый слой увеличивает значение функции логарифмического правдоподобия, что позволяет приблизить распределение модели к истинному совместному распределению вероятностей обучающего набора.

В 2015 году Элдан и Шамир эмпирически убедились, что увеличение количества слоев нейросетей экспоненциально более ценно, чем увеличение ширины слоев сети по количеству блоков в каждом слое (глубина против ширины). Мы также знаем из практики (без доказательств), что можно обучить сеть с сотнями скрытых слоев, где более глубокие слои представляют признаки более высокого порядка. Исторически сложилось так, что для обучения глубоких сетей требовалось преодолеть проблему исчезающих градиентов.

Исходный автокодировщик

Архитектура автокодировщика предназначена для сжатия информации о входе в его низкоразмерные скрытые слои. Скрытые слои должны сохранять то же количество информации, что и входные слои, даже если в них меньше блоков, чем во входных. Мы уже обсуждали современные вариационные автокодировщики, обеспечивающие эффективный метод обучения автокодировочных сетей. В процессе обучения каждый вектор сопоставляется с самим собой (без контроля), а сеть пытается обучиться наилучшему *кодированию*. При этом входной и выходной слои должны иметь одинаковое количество блоков. Машина Больцмана с определенным количеством входных блоков, меньшим количеством скрытых блоков и выходным слоем с тем же количеством блоков, что и во входном слое, описывает оригинальную архитектуру сетевого автокодировщика. С исторической точки зрения это очень важно, поскольку автокодировщик представляет собой один из первых примеров успешного обучения сети коду, заложенному в состояниях скрытых блоков, для представления своих входов. Благодаря этому можно заставить сеть сжимать свои входы в скрытый слой с минимальной потерей информации. Теперь это является неотъемлемой частью нейросетей, воспринимаемой нами как должное. Архитектура автокодировщиков с машинами Больцман и без них (с совместным распределением вероятности на основе энергии и без него) по-прежнему играет важную роль в мире глубокого обучения.

Ранее в этой главе мы обсуждали вариационных автокодировщиков. С исторической точки зрения они синтезируют идеи автокодировщиков машин Больцмана, глубоких автокодировочных сетей, *шумоподавляющих автокодировщиков* и информационного узкого места (Тишби и др., 2000), которые берут свое начало в идее анализа через синтез (Селфридж, 1958). Вариационные автокодировщики используют для обучения быстрые вариационные методы. В контексте компромисса между смещением и дисперсией вариационные методы дают смещенные оценки для весов сети с низкой дисперсией.

Вероятностное языковое моделирование

Естественная связь между этой главой и *главой 7*, почти полностью посвященной исключительно обработке естественного языка и различным способам извлечения смысла из данных естественного языка, состоит в том, чтобы рассмотреть основы вероятностных языковых моделей, а затем выделить опирающиеся на эти основы модели из *главы 7*.

Эта глава начинается с метода максимального правдоподобия. Одна из причин, по которой он появляется всегда, когда нужно оценить распределения вероятностей, заключается в том, что распределение вероятностей, полученное методом максимального правдоподобия, поддерживается математической теорией при соблюдении определенных условий: оценка максимального правдоподобия сходится к истинному распределению $p_{\text{данные}}(\vec{x})$, породившему данные, в пределе, когда число

выборок данных уходит в бесконечность (т. е. при условии, что имеется тонна данных), и при условии, что распределение вероятностей модели $p_{\text{модель}}\left(\vec{x}, \vec{\theta}\right)$ уже включает истинное распределение. То есть в пределе, когда количество выборок уходит в бесконечность, параметры модели $\vec{\theta}^*$, максимизирующие вероятность данных, удовлетворяют $p_{\text{модель}}\left(\vec{x}, \vec{\theta}^*\right) = p_{\text{данные}}\left(\vec{x}\right)$.

В языковых моделях обучающие данные — это образцы текста из некоторого корпуса и/или жанра, и нужно узнать их распределение вероятностей для генерации похожих текстов. Важно помнить, что истинное распределение данных, скорее всего, *не входит* в семейство распределений, представленных $p_{\text{модель}}\left(\vec{x}, \vec{\theta}\right)$, поэтому теоретический результат в предыдущем абзаце может так и не оправдаться на практике; однако это не останавливает нас, и мы обычно довольствуемся моделями, так или иначе пригодными для наших целей. Наша цель — создать модель, которая будет присваивать вероятности фрагментам языка. Если собрать произвольно несколько фрагментов языка, то, скорее всего, получится тарабарщина. Нам же нужно найти распределение предложений, несущих некий смысл. Хорошая языковая модель присваивает высокую вероятность предложениям, имеющим смысл, даже если этих предложений нет среди обучающих данных. Для оценки эффективности языковой модели вычисляется ее *перплексия* на обучающем наборе данных.

Языковые модели основаны на предположении, что распределение вероятности следующего слова зависит от $n-1$ слов, которые ему предшествовали, при некотором фиксированном n, поэтому важно вычислить $p_{\text{модель}}\left(x_n \mid x_1, x_2, ..., , x_{n-1}\right)$. При использовании модели word2vec, которая заключает значение каждого слова в вектор, каждая переменная x будет представлена вектором. Слова с одинаковым значением или часто используемые в схожих контекстах имеют схожие векторные значения. Для предсказания вектора следующего слова на основе векторов предыдущих слов можно воспользоваться трансформером из *главы 7*.

Языковые модели на основе частоты составляют таблицы условных вероятностей, подсчитывая количество совместных появлений слов в обучающем корпусе. Например, можно оценить условную вероятность $p(\text{утро} \mid \text{доброе})$ появления слова "утро" после слова "доброе", подсчитав, сколько раз в корпусе встречается "доброе утро", разделенное на количество появлений в корпусе слова "доброе". То есть:

$$p(\text{утро} \mid \text{доброе}) = \frac{p(\text{доброе, утро})}{p(\text{доброе})}.$$

Такая модель не подойдет в случае очень больших корпусов или неструктурированных текстовых данных, таких как твиты, комментарии в Facebook, SMS-сообщения, где не полностью соблюдаются обычные правила грамматики/орфографии и т. д.

Понятие вероятностной языковой модели можно формализовать следующим образом:

1. Зададим словарный запас V языка. Это может быть набор символов, пробелов, знаков препинания, уникальных слов и/или n-грамм. Математически это конечное дискретное множество, которое включает в себя символ остановки, обозначающий конец мысли или предложения, например точку (хотя точка не всегда означает конец предложения, пример — ее наличие в сокращениях).
2. Определим предложение (которое может быть осмысленным или не быть таковым) как конечную последовательность символов $\vec{x} = (x_1, x_2, ..., x_m)$ из словаря V, заканчивающуюся символом остановки. Каждый x_i может принимать любое значение из словаря V. Можно задать m как максимальную длину предложения.
3. Определим языковое пространство $l^m = \{(x_1, x_2, ..., x_m), x_i \in V\}$ как множество всех предложений длиной меньше или равной m. Подавляющее большинство этих предложений ничего не значит, и нам нужно определить языковую модель, фиксирующую только несущие смысл предложения: высокие вероятности для осмысленных предложений и низкие вероятности для предложений, не имеющих смысла.
4. Пусть $\mathscr{L}$ — коллекция всех поднаборов l^m. Таким образом, учитываются коллекции всех осмысленных и бессмысленных предложений максимальной длины m.
5. Строгая теория вероятностей начинается, как правило, с тройки, которая включает в себя пространство, сигма-алгебру, содержащую некоторые поднаборы этого пространства, и меру вероятности, назначенную каждому члену выбранной сигмы-алгебры (не будем вдаваться в подробности в этой главе). В таком контексте языковая модель будет представлять собой вероятностную тройку, состоящую из языкового пространства l^m, сигма-алгебры, составленной из всех поднаборов языкового пространства $\mathscr{L}$, и вероятностной меры P, которую нужно назначить каждому члену $\mathscr{L}$. Так как наше языковое пространство — дискретное и конечное, то вместо этого будет проще присвоить каждому члену l^m вероятность p, т. е. вероятность каждого предложения $\vec{x} = (x_1, x_2, ..., x_m)$ (т. к. это, в свою очередь, индуцирует вероятностную меру P на коллекцию всех поднаборов $\mathscr{L}$, так что об этом вовсе не стоит беспокоиться в случае языковых моделей). Собственно, это p нам и требуется узнать из обучающих данных. Обычный подход заключается в отборе полного семейства распределений вероятностей $p(\vec{x}; \vec{\theta})$, параметризованное $\vec{\theta}$.
6. Наконец, необходимо оценить параметр $\vec{\theta}$ путем максимизации вероятности обучающего набора данных, содержащего большое количество выборок с предложениями из l^m. Поскольку вероятности осмысленных предложений — это очень маленькие числа, во избежание потери значимости используется логарифм этих вероятностей.

В целях согласованности можно проверить в контексте текущего раздела линейные логарифмические модели, билинейные логарифмические модели GloVe и латентное распределение Дирихле из *главы 7*.

Итоги и перспективы

Мы прошли еще одну основополагающую главу в своем путешествии в базовую математику для современных моделей искусственного интеллекта и перешли от изучения детерминированных функций в предыдущих главах к изучению совместных вероятностных распределений признаков данных. Наша цель — использовать их для генерации новых данных, аналогичных обучающим. Мы узнали, пока без формализации, множество свойств и правил распределений вероятностей. Проведен обзор наиболее актуальных моделей, а также исторической эволюции до сегодняшнего дня. Проведено различие между моделями, предоставляющими явные формулы для своих совместных распределений, и моделями, косвенно взаимодействующими с базовым распределением без явного написания формул. Вычисление логарифма правдоподобия и его градиентов для моделей с явными формулами может быть как простым, так и сложным, и каждый из этих случаев требует собственных методов. При этом цель всегда остается одной и той же — отразить истинное совместное вероятностное распределение данных, подобрав модель, максимизирующую логарифмическое правдоподобие.

Ничего этого не потребуется в случае низкоразмерных данных с одним или двумя признаками. С оценкой вероятностных распределений низкоразмерных данных отлично справляются гистограммы и оценщики плотности ядра. Одним из главных достижений машинного обучения является возможность моделирования высокоразмерных совместных вероятностных распределений на основе большого объема данных.

Все рассмотренные в этой главе подходы имеют свои достоинства и недостатки. Например, вариационные автокодировщики позволяют как проводить обучение, так и эффективно делать байесовский вывод в вероятностных графовых моделях со скрытыми (латентными) переменными. Однако они генерируют выборки более низкого качества. Генеративно-состязательные сети генерируют более качественные выборки, но их сложнее оптимизировать из-за нестабильной динамики обучения. Вместо стабильного максимума или минимума они ищут нестабильную седловую точку. Глубокие сети доверия, такие как PixelCNN и WaveNet, характеризуются стабильным процессом обучения, оптимизируя функцию потерь softmax. Однако они неэффективны при осуществлении выборки и не способны органично встраивать данные в более низкие измерения, как это делают автокодировщики.

Игры с нулевой суммой для двух игроков из теории игр естественным образом вошли в эту главу вследствие настройки генеративно-состязательных сетей.

Заглядывая в следующую главу, посвященную *графовому моделированию*, отметим, что связи в графе нейросети диктуют способ записи условных вероятностей, легко указывая на различные зависимости и условную независимость. В этом мы убедились, когда рассматривали ограниченные машины Больцмана в данной главе. В следующей главе мы сфокусируемся исключительно на графовом моделировании, которого нам удавалось избегать на протяжении добрых трех четвертей книги.

ГЛАВА 9
Графовые модели

Именно этому мы и хотим научиться.
— *Хала*

Графы, диаграммы и сети окружают нас повсюду — это и города с их картами дорог, и аэропорты с их стыковочными рейсами, это электрические и энергетические сети, Всемирная паутина, биологические и молекулярные сети, например наша нервная система, социальные сети, сети террористических организаций, схематические представления математических моделей, искусственные нейросети и многое-многое другое. Их легко распознать — отдельные узлы представляют некие важные для нас сущности, которые затем соединяются направленными или ненаправленными ребрами, указывающими на наличие определенных отношений между связанными узлами.

Данные, имеющие естественную графовую структуру, проще понять с помощью механизма, который использует и сохраняет эту структуру, создавая функции, работающие непосредственно с графами (как бы они ни были математически представлены), а не подает графовые данные в существующие модели машинного обучения, которые искусственно изменяют форму данных перед их анализом. Это неизбежно ведет к потере ценной информации. Именно по этой же причине сверточные нейросети успешно работают с данными изображений, а рекуррентные нейросети — с последовательными данными и т. д.

Графовые модели весьма привлекательны для специалистов по обработке данных и инженеров. Графовые структуры обеспечивают гибкость, которой нет в таких пространствах с фиксированной базовой системой координат, как *евклидовы пространства* или *реляционные базы данных*, где данные и их признаки должны быть жесткой и заранее определенной формы. Более того, графы — это естественная среда, в которой можно исследовать взаимосвязи между точками в наборе данных. До сих пор наши модели машинного обучения использовали данные, представленные в виде изолированных точек. Графовые модели, в свою очередь, используют изолированные точки данных *и связи между ними*, что позволяет глубже понимать и создавать более выразительные модели.

Человеческий мозг воспринимает графические структуры естественным образом — он способен моделировать объекты и их связи. Он также достаточно гибок, что позволяет генерировать новые сети или расширять и модернизировать существующие, например при планировании городов, разработке проектов или постоянном

обновлении транзитных сетей. Более того, мы можем легко переходить от текста на естественном языке к графовым моделям, и наоборот. Когда мы читаем что-то новое, нам кажется естественным сформулировать графическое представление, чтобы лучше понять текст или проиллюстрировать его другим людям. И наоборот, схемы графов можно описать с помощью естественного языка. В настоящее время существуют модели, которые генерируют текст на естественном языке на основе графов знаний, и наоборот. Это называется рассуждениями над графами знаний.

Мы уже достаточно хорошо знакомы со структурными элементами нейросетей, а также с типами данных и задач, для которых они обычно подходят:

- многослойный перцептрон или полносвязная нейросеть (*см. главу 4*);
- сверточные слои (*см. главу 5*);
- рекуррентные слои (*см. главу 7*);
- компоненты "кодер — декодер" (*см. главу 7*);
- состязательные компоненты и игры двух игроков с нулевой суммой (*см. главу 8*);
- вариационные компоненты (*см. главу 8*).

Основными задачами в большинстве случаев являются классификация, регрессия, кластеризация, кодирование и декодирование, а также генерация новых данных, когда модель обучается совместному вероятностному распределению признаков данных.

Мы уже знаем, что можно смешивать и сочетать некоторые компоненты нейросетей для построения новых моделей, ориентированных на решение конкретных задач. Хорошая новость заключается в том, что графовые нейросети используют те же самые компоненты, поэтому в этой главе нам не понадобится изучать новые концепции машинного обучения. Как только мы поймем, как математически представить графовые данные и их признаки так, чтобы передать их в нейросеть для анализа или генерации новых данных сети (графа), мы будем готовы. Поэтому мы не будем вдаваться в разбор всех существующих графовых нейросетей, а сконцентрируемся на простой математической формулировке, популярных приложениях, общих задачах для графовых моделей, доступных наборах данных и методах оценки модели. Наша цель — развить сильную интуицию в понимании сути предмета. И снова, главная проблема, которую нам предстоит решить, — это снижение размерности задачи так, чтобы она поддавалась вычислениям и анализу, сохранив при этом максимальное количество информации. Другими словами, сложно предположить, что в случае сети с миллионами пользователей модели будут принимать на вход векторы или матрицы с миллионами измерений. Требуются эффективные методы представления графовых данных.

Тем, кто захочет глубже и быстрее погрузиться в графовые нейросети, будет неплохо познакомиться для начала с обзорной статьей "Всесторонний обзор графовых нейросетей" (Ву З. и др., 2019; **https://oreil.ly/938pf**), но только после внимательного прочтения этой главы.

Графы: узлы, грани и их признаки

Графы идеально подходят для моделирования любых задач, целью которых является понимание дискретного набора объектов (с акцентом на дискретность, а не на непрерывность) через отношения между ними. Теория графов — относительно молодая дисциплина в дискретной математике и информатике с неограниченным потенциалом применения. Для решения множества нерешенных проблем в этой области нужно больше "свежих мозгов".

Граф (рис. 9.1) состоит из следующих компонентов.

Узлы, или вершины.

Объединены в набор в виде $\text{Узлы} = \{\text{узел}_1, \text{узел}_2, ..., \text{узел}_n\}$. Набор может быть как небольшим — несколько узлов (или даже один узел), так и огромным — миллиарды узлов.

Ребра.

Соединяют два любых узла (это может быть собственное ребро узла или несколько ребер, соединяющих два одинаковых узла) направленным (от одного узла к другому) или ненаправленным способом (ребро не имеет направления от одного узла к другому). Множество ребер — это $\text{Ребра} = \{\text{ребро}_{ij} = (\text{узел}_i, \text{узел}_j)$ так, что есть ребро, направляющее из узла_i в $\text{узел}_j\}$.

Признаки узла.

Каждому узлу_i можно присвоить список, скажем, d признаков (например, возраст, пол и уровень дохода пользователя социальных сетей), объединенных в вектор $\overrightarrow{\text{признаки}_{\text{узел}_i}}$. Затем можно объединить все векторы признаков n узлов графа в матрицу $\textbf{Признаки}_{\text{узлы}}$ размера $d \times n$.

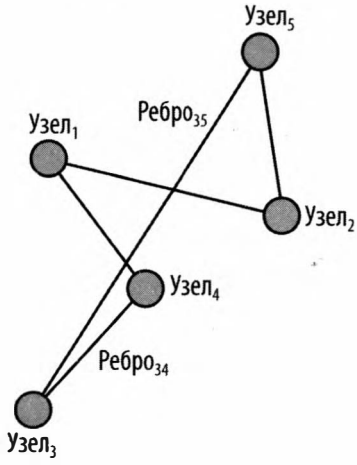


Рис. 9.1. Граф состоит из узлов и соединяющих их направленных или ненаправленных ребер

Признаки ребер.

Аналогично каждому ребру$_{ij}$ можно присвоить список c признаков (например, протяженность дороги, ограничение скорости и показатель, является ли дорога платной/бесплатной), объединенных в вектор $\overrightarrow{\text{признаки}_{\text{ребро}_{ij}}}$. Затем можно объединить все векторы признаков m ребер графа в матрицу $\mathbf{Признаки}_{\text{Ребра}}$ размера $c \times m$.

Эффективность графовых моделей обусловлена их гибкостью и отсутствием необходимости придерживаться жесткой решетчатой структуры. Их узлы можно представить как *плавающие в пространстве* без каких-либо координат. Они удерживаются вместе только за счет соединяющих их ребер. Вместе с тем нам нужно каким-то способом отобразить их внутреннюю структуру. Существуют программные пакеты, визуализирующие графы с помощью наборов узлов и ребер, но на этих красивых (и информативных) картинках невозможно проводить анализ и вычисления. В качестве входных данных для моделей машинного обучения можно воспользоваться двумя популярными представлениями графов: *матрица смежности* и *матрица инцидентности* графа.

Среди прочих полезных представлений спекулятивных графовых алгоритмов можно отметить такие, как *перечисление ребер*, *два линейных массива* и *перечисление последователей*. Все они передают одну и ту же информацию, но отличаются требованиями к хранению и эффективностью поиска, получения и операций с графами. Большинство графовых нейросетей получают на вход матрицу смежности, а также матрицы признаков узлов и ребер. В большинстве случаев до ввода графовых данных в модель необходимо уменьшить размерность (так называемое представление графа или встраивание графа). В других случаях шаг сокращения размерности — это часть самой модели.

Матрица смежности.

Одним из алгебраических способов хранения структуры графа на машине и изучения его свойств является матрица смежности, представляющая собой матрицу размера $n \times n$, которую при наличии ребра, соединяющего вершины узел$_i$ и узел$_j$, можно записать как смежность$_{ij} = 1$, а в отсутствии ребра из вершины узел$_i$ в вершину узел$_j$ — как смежность$_{ij} = 0$. Важно отметить, что это определение учитывает собственное ребро, т. е. ребро из вершины к самой себе, а не множество ребер между двумя разными вершинами, если только в записи матрицы не включены числа 2, 3 и т. д., хотя это может испортить некоторые результаты, полученные с помощью матрицы смежности.

Матрица инцидентности.

Еще один алгебраический способ хранить структуру графа и полную информацию о нем. Здесь мы перечисляем и вершины, и ребра, а затем формулируем матрицу, чьи строки соответствуют вершинам, а столбцы — ребрам. Запись

матрицы инцидентности$_{ij}$ равна 1, если ребро$_j$ соединяет узел$_i$ с другим узлом, и 0 в противном случае. Заметим, что это определение способно учесть множество ребер между двумя различными вершинами, но не собственное ребро из вершины к самой себе. Поскольку в большинстве графов ребер больше, чем вершин, эта матрица обычно довольно широкая и размером больше, чем матрица смежности.

Еще одна матрица, связанная с неориентированным графом, — это *матрица Лапласа* (Laplacian matrix)[1]. Она представляет собой симметричную матрицу размера $n \times n$, где каждый узел имеет соответствующую строку и столбец. Диагональные записи матрицы Лапласа равны степени каждого узла, а недиагональные записи равны нулю в отсутствие ребра между узлами, соответствующими этой записи, и –1 при наличии ребра между ними. Это дискретный аналог непрерывного оператора Лапласа из исчислений и дифференциальных уравнений с частными производными, где дискретизация происходит в узлах графа.

Матрица Лапласа учитывает вторые производные непрерывной (и дважды дифференцируемой) функции, оценивающие вогнутость функции или то, насколько ее значение в точке отличается от значения в окружающих точках. По аналогии с оператором непрерывного лапласиана, матрица Лапласа служит мерой того, насколько значения в одном узле графа отличаются от значений в соседних узлах. Графовый лапласиан применяется при изучении случайных блужданий по графам, а также электрических сетей и сопротивлений. Мы рассмотрим их далее в этой главе.

Из матриц смежности и инцидентности будет несложно вывести простую статистику узлов и ребер, например степени узлов (степень узла — это количество ребер, связанных с этим узлом). Распределение степеней $P(k)$ отражает изменчивость степеней всех узлов. Показатель $P(k)$ — это эмпирическая вероятность того, что узел имеет ровно k ребер. Это может быть интересно для таких сетей, как веб-связь и биологические сети.

Например, если распределение узлов степени k в графе подчиняется степенному закону вида $P(k) = k^{-\alpha}$, то в таких графах будет мало узлов с высокой связностью, или *хабов*, занимающих центральное место в топологии сети и в целом удерживающих ее, и много узлов с низкой связностью, соединенных с хабами.

Можно также добавить зависимость от времени и представить себе динамические графы, свойства которых меняются с течением времени. В настоящее время существуют модели, которые добавляют зависимость от времени к векторам признаков узлов и/или ребер (так что каждая запись этих векторов становится зависимой от времени). Например, в системе GPS, прокладывающей маршруты движения, признак ребер, соединяющих одну точку на карте с другой, меняется со временем в зависимости от дорожной ситуации.

---

[1] Также называется графовым лапласианом, матрицей допуска, матрицей Кирхгофа или дискретным лапласианом. — *Прим. ред.*

Теперь, когда имеется математическая основа для графовых объектов, а также признаки их узлов и ребер, можно вводить эти репрезентативные векторы и матрицы (и метки для контролируемых моделей) в модели машинного обучения и заниматься обычным делом. Чаще всего иметь хорошее представление объектов — это уже полдела.

Другая половина истории — это выразительные возможности моделей машинного обучения в целом, когда можно действительно получить хорошие результаты, не кодируя (и даже не изучая) правила, ведущие к ним. В контексте текущей главы это означает, что можно сразу переходить к графовым нейросетям, оставляя теорию графов *пока* в стороне.

Ориентированные графы

С одной стороны, в ориентированных графах нас интересуют те же свойства, что и в неориентированных, такие как их остовные деревья, принципиальные схемы, множества разрезов, планарность, толщина и др. С другой стороны, ориентированные графы обладают собственными уникальными свойствами, отличными от неориентированных, такими как сильная связность, арборесценция (направленная форма корневого дерева), дециклизация и др.

Пример: алгоритм PageRank

PageRank (**https://oreil.ly/0yqGu**) — это устаревший алгоритм (утратил силу в 2019 году), который компания Google использовала для ранжирования веб-страниц в результатах поиска. Он позволяет оценить важность веб-страницы на основе количества ссылок на нее на других страницах. На языке графов узлы — это веб-страницы, а направленные ребра — ссылки, ведущие с одной страницы на другую. Согласно PageRank, узел считается важным, если на него ссылается много других веб-страниц, т. е. если его входящая степень велика (рис. 9.2).

В качестве конкретного примера, включающего графы, матрицу смежности, линейную алгебру и веб, рассмотрим алгоритм PageRank для предельно упрощенного варианта Интернета, состоящего не из миллиардов, а всего из четырех проиндексированных веб-страниц, как показано на рис. 9.3.

В графе на рис. 9.3 с A связан только узел B; A и D связаны с B; A и D связаны с C; A, B и C связаны с D; A связан с B, C, D; B связан с A и D; C связан с D; D связан с B и C; D связан с B и C.

Представим пользователя Интернета, который во время посещения какой-либо страницы случайно нажимает на ссылку с этой страницы, затем на ссылку с новой страницы и т. д. Таким образом этот пользователь имитирует *случайное блуждание по графу Интернета*.

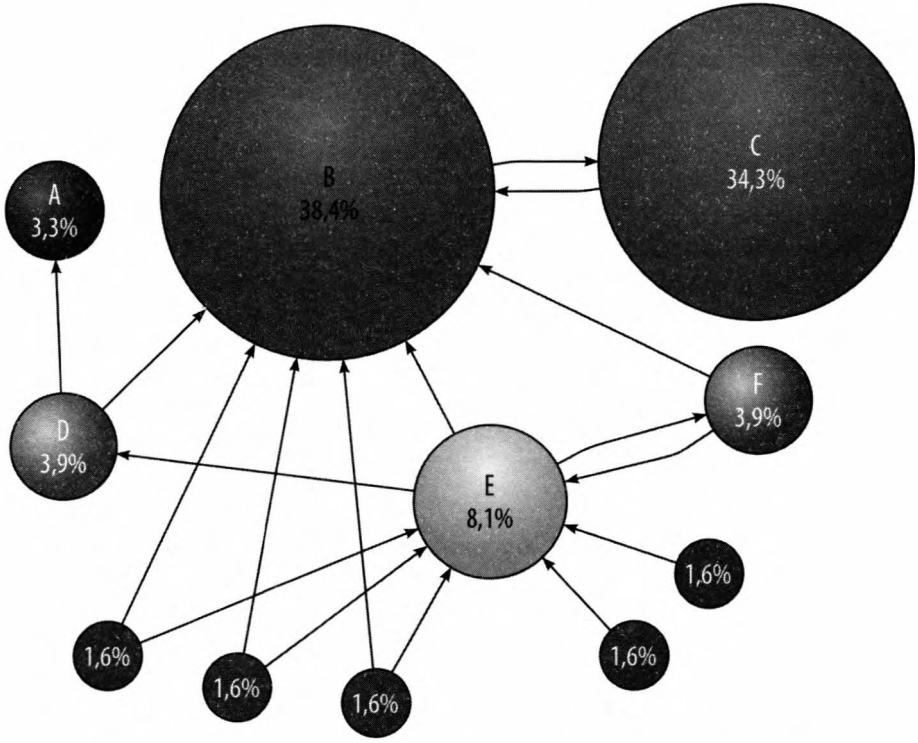


Рис. 9.2. PageRank присваивает более высокий балл страницам, на которые указывает (или ссылается) большее количество страниц (источник изображения: **https://oreil.ly/wMg3p**)

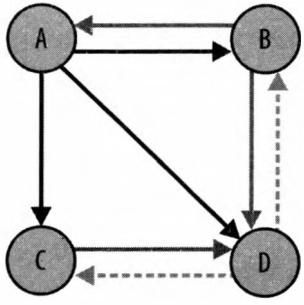


Рис. 9.3. Макет Всемирной паутины, состоящий только из четырех проиндексированных веб-страниц (по материалам Coursera: "Математика для машинного обучения", **https://oreil.ly/gHqYs**)

В целом на графе, представляющем Всемирную паутину, такой случайный пользователь проходит по графу от определенного узла к одному из соседних (или обратно к себе, если имеются обратные ссылки на страницу). В этой главе мы еще не раз коснемся Всемирной паутины и рассмотрим вопросы, раскрывающие характер ее графа. Для случайного блуждания требуется матрица, которую в данном контексте можно назвать *матрицей ссылок*, хотя в действительности это матрица смежности,

взвешенная по степени каждой вершины. Такая матрица случайного блуждания, или матрица ссылок, используется для оценки долгосрочного поведения случайного блуждания по графу. Понятие случайного блуждания по графам проходит лейтмотивом через всю главу.

Вернемся к четырехстраничной Всемирной паутине на рис. 9.3. Если пользователь находится на странице *A*, то существует одна треть вероятности, что он перейдет на страницу *B*, одна треть — на страницу *C* и одна треть — на страницу *D*. Таким образом, *вектор внешних ссылок* страницы *A* равен:

$$\overrightarrow{\text{ссылки}}_A = \begin{pmatrix} 0 \\ 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}.$$

Если пользователь находится на странице *B*, то существует половина вероятности, что он перейдет на страницу *A*, и половина вероятности — на страницу *D*. Таким образом, вектор внешних ссылок страницы *B* равен:

$$\overrightarrow{\text{ссылки}}_B = \begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \end{pmatrix}.$$

Аналогично векторы внешних ссылок страниц *C* и *D* имеют вид:

$$\overrightarrow{\text{ссылки}}_C = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \text{ и } \overrightarrow{\text{ссылки}}_D \begin{pmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{pmatrix}.$$

Объединим векторы ссылок всех веб-страниц и получаем матрицу ссылок:

$$\textbf{Ссылки} = \begin{pmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 1 & 0 \end{pmatrix}.$$

Важно отметить, что столбцы матрицы ссылок — это вероятности *исходящих ссылок*, а строки *A* — вероятности *входящих ссылок*. Как пользователь может оказаться на странице *A*? Он может оказаться только на странице *B*, а оттуда с половиной вероятности он попадет на страницу *A*.

Теперь можно *ранжировать* страницу *A*, сложив ранги всех страниц с ссылками на *A*, каждая из которых взвешена по вероятности того, что пользователь попадет с нее на страницу *A*; т. е. страница, на которую указывает множество высоко ранжи-

рованных страниц, также будет иметь высокий ранг. Таким образом, ранги (rank) всех четырех страниц составят:

$$\text{ранг}_A = 0 \cdot \text{ранг}_A + 1/2 \cdot \text{ранг}_B + 0 \cdot \text{ранг}_C + 0 \cdot \text{ранг}_D;$$
$$\text{ранг}_B = 1/3 \cdot \text{ранг}_A + 0 \cdot \text{ранг}_B + 0 \cdot \text{ранг}_C + 1/2 \cdot \text{ранг}_D;$$
$$\text{ранг}_C = 1/3 \cdot \text{ранг}_A + 0 \cdot \text{ранг}_B + 0 \cdot \text{ранг}_C + 1/2 \cdot \text{ранг}_D;$$
$$\text{ранг}_D = 1/3 \cdot \text{ранг}_A + 1/2 \cdot \text{ранг}_B + 1 \cdot \text{ранг}_C + 0 \cdot \text{ранг}_D.$$

Для того чтобы найти числовое значение ранга каждой веб-страницы, нужно решить систему линейных уравнений, что переносит нас в область линейной алгебры. В матрично-векторной системе представлений можно записать это как:

$$\overrightarrow{\text{ранги}} = \textbf{Ссылки}\ \overrightarrow{\text{ранги}}.$$

Таким образом, вектор, содержащий все ранги всех веб-страниц, является собственным вектором матрицы ссылок графа веб-страниц (где узлы — это веб-страницы, а направленные ребра — связи между ними) с собственным значением 1. Напомним, что в реальности граф веб-страниц огромен, а значит, матрица ссылок будет такой же огромной, и здесь непосредственный интерес представляет разработка эффективных способов нахождения ее собственных векторов.

Вычисление собственных векторов и собственных значений заданной матрицы — это важнейший вклад численной линейной алгебры, имеющий непосредственное применение во многих областях. Многие численные методы нахождения собственных векторов и собственных значений предполагают многократное умножение матрицы на вектор. При работе с огромными матрицами это требует больших затрат, так что для удешевления операций придется пойти на всевозможные уловки. Воспользуемся разреженностью матрицы (многие записи — нули, поэтому перемножать их, чтобы выяснить, что это просто нули, будет пустой тратой времени) — введем случайность, или стохастичность, и окунемся в область высокоразмерных вероятностей и больших случайных матриц (мы познакомимся с ними в *главе 11*, посвященной вероятности). Сейчас еще раз воспользуемся итерационным методом, который рассматривается в *главе 6*, посвященной сингулярным разложениям, — начинаем со случайного вектора $\overrightarrow{\text{ранги}}_0$, а затем итеративно получаем последовательность векторов, умножая на матрицу ссылок:

$$\overrightarrow{\text{ранги}}_{i+1} = \textbf{Ссылки}\ \overrightarrow{\text{ранги}}_i.$$

В нашем случае четырех страниц Всемирной паутины это сводится к вектору:

$$\overrightarrow{\text{ранги}} = \begin{pmatrix} 0{,}12 \\ 0{,}24 \\ 0{,}24 \\ 0{,}4 \end{pmatrix}.$$

Это значит, что страница *D* имеет наивысший ранг, и при запросе поисковой системы с аналогичным содержанием она будет возвращена первой. Затем можно об-

новить диаграмму на рис. 9.3, где размер каждого круга соответствует важности страницы.

Когда функционировал алгоритм PageRank, его практическая реализация включала коэффициент затухания d, число от 0 до 1, как правило, 0,85, которое учитывает только 85-процентную вероятность того, что пользователь перейдет по ссылке со страницы, которую он посещает в данный момент, и 15-процентную вероятность того, что он начнет с совершенно новой страницы, на которой нет ссылок с текущей страницы. Таким образом, итеративный процесс поиска рангов страниц в Интернете модифицируется в более простой способ:

$$\overrightarrow{\text{ранги}}_{i+1} = d \cdot \left(\textbf{Ссылки}\ \overrightarrow{\text{ранги}}_i\right) + \frac{1-d}{\text{общее количество страниц}} \cdot \overrightarrow{\text{единичные}}.$$

В конце концов, можно задаться вопросом: продолжает ли Google искать в Интернете новые веб-страницы и индексировать их, а также проверяет ли он все проиндексированные веб-страницы на наличие новых ссылок? Ответ будет положительным, и далее приведем выдержки из "Углубленного руководства по принципам работы Google Поиска" (**https://oreil.ly/oHw0g**):

> "Google Поиск — это полностью автоматизированная поисковая система, использующая программное обеспечение, так называемых поисковых роботов, которые постоянно сканируют веб и добавляют в индекс новые страницы. Фактически подавляющее большинство страниц появляется в результатах поиска не посредством ввода вручную, а находятся и добавляются автоматически, когда роботы сканируют веб-страницы. [...] Поскольку официального реестра веб-страниц не существует, роботу Google приходится постоянно искать новые страницы и добавлять их к списку известных. Этот процесс называется „обнаружение URL“. О некоторых страницах известно, потому что робот Google посещал их раньше. Другие обнаруживаются при переходе по ссылкам с уже известных страниц (например, по ссылке на новую запись в блоге на главной странице или странице категории). Иногда владельцы сайтов сами присылают нам списки URL, которые нужно просканировать, — так называемые файлы sitemap. [...] Когда пользователь вводит запрос, система находит в индексе и показывает в результатах пользовательского поиска самые подходящие страницы высокого качества. При этом учитываются сотни различных факторов, такие как местоположение, язык, тип устройства пользователя (компьютер или телефон) и многое другое. Например, результаты по запросу „ремонт велосипедов“ будут различаться в зависимости от того, находитесь ли вы в Париже или в Гонконге".

Чем больше собирается данных, тем сложнее будет поиск. В 2015 году Google выпустила RankBrain. Этот алгоритм использует машинное обучение для векторизации текста на веб-страницах аналогично тому, что мы выполняли в *главе 7*. Подобная процедура добавляет контекст и смысл к проиндексированным страницам для более точных результатов поиска. Недостатком этого процесса является большая размерность, связанная с векторами смысла. Для того чтобы избежать трудностей, связанных с проверкой каждого вектора в каждом измерении, Google, прежде чем

вернуть наиболее близкие к запросу веб-страницы, использует алгоритм ближайшего соседа, который возвращает отличные результаты за миллисекунды, — это то, что происходит сейчас.

Инверсия матриц с помощью графов

Многие задачи в прикладных науках подразумевают запись дискретной линейной системы $\mathbf{A}\vec{x} = \vec{b}$ и ее решение, что эквивалентно инверсии матрицы **A** и поиску решения $\vec{x} = \mathbf{A}^{-1}\vec{b}$. Но в случае больших матриц эта операция требует больших вычислительных затрат, а также больших объемов памяти, страдая низкой точностью. Мы постоянно ищем эффективные способы инверсии матриц, усиливая особые характеристики конкретных матриц в некоторых случаях.

Рассмотрим пошагово графо-теоретический метод вычисления обратной матрицы внушительного размера (например, сто строк и сто столбцов):

1. Меняем каждую ненулевую запись в матрице **A** на 1. Получаем бинарную матрицу.
2. Переставляем строки и соответствующие столбцы полученной бинарной матрицы так, чтобы все диагональные записи были равны 1.
3. Рассматриваем полученную матрицу как матрицу смежности ориентированного графа (при этом из графа удаляются самоциклы, соответствующие 1 по диагонали).
4. Разбиваем полученный ориентированный граф на фрагменты.
5. Если фрагмент слишком велик, разбиваем его на более мелкие фрагменты, удаляя соответствующее ребро.
6. Инвертируем меньшие матрицы.
7. Очевидно, в результате получается матрица, обратная по отношению к исходной.

Не вдаваясь в подробности, только отметим, что этот метод настолько изящен, что не мог не попасть в текущую главу.

Графы Кэли для групп: чистая алгебра и параллельные вычисления

Графы Кэли (Cayley graphs) для групп, которые также называются диаграммами Кэли (Cayley diagrams), могут оказаться полезными при проектировании и анализе сетевых архитектур для параллельных компьютеров, проблем маршрутизации и алгоритмов маршрутизации для взаимосвязанных сетей. В статье "Сети процессорных соединений на основе графа Кэли" (Шибель и др., 2011; **https://oreil.ly/zXqYi**) можно в увлекательной форме прочитать о ранних разработках, когда графы Кэли применялись для сетей параллельных вычислений, и узнать, как строятся графы

Кэли, отвечающие определенным расчетным параметрам. Графы Кэли также применялись для классификации данных (**https://oreil.ly/xp5hB**).

Можно представить каждую группу из *n* элементов в виде связного ориентированного графа из *n* узлов, где каждый узел соответствует элементу из группы, а каждое ребро представляет собой умножение на генератор из группы. Ребра размечены (или окрашены) в зависимости от того, на какой генератор из группы выполняется умножение (рис. 9.4). Такой ориентированный граф однозначно определяет группу — каждому произведению элементов в группе соответствует последовательность направленных ребер на графе. Например, граф циклической группы *n* элементов — это направленная цепь из *n* узлов, где каждое ребро представляет собой умножение на один генератор из группы.

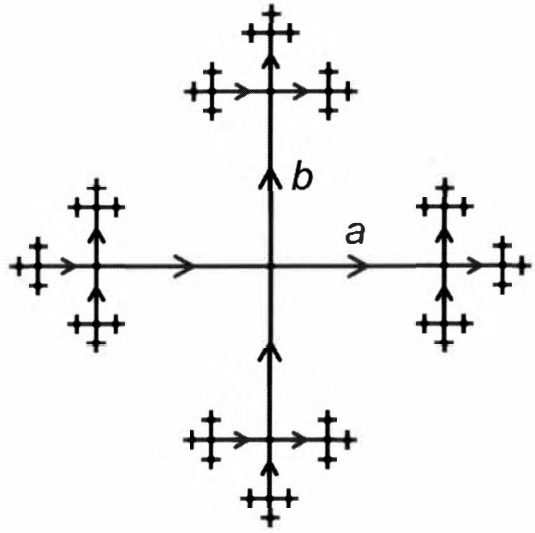


Рис. 9.4. Граф Кэли для свободной группы на двух генераторах *a* и *b*; каждый узел представляет элемент свободной группы, а каждое ребро — умножение на *a* или *b* (источник изображения: **https://oreil.ly/cafsv**)

С точки зрения чистой математики, графы Кэли для групп полезны для визуализации и изучения абстрактных групп, представляя полностью абстрактную структуру и все ее элементы в виде наглядной диаграммы. Благодаря симметрии графов Кэли они могут оказаться полезными при построении более сложных абстрактных объектов. Это центральные инструменты для комбинаторной и геометрической теории групп. Подробную информацию о графах Кэли можно прочитать на странице Wolfram Mathworld (**https://oreil.ly/JCvux**).

Передача сообщений внутри графа

Структура передачи сообщений позволяет моделировать распространение информации в графах, а также четко агрегировать информацию, передаваемую узлами,

ребрами и структурой самого графа, в векторы нужной размерности. Каждый узел в этой структуре обновляется информацией из векторов признаков соседних узлов и соединенных с ними ребер. Графовая нейросеть выполняет несколько раундов передачи сообщений; каждый раунд распространяет информацию об одном узле дальше. Наконец, объединяя латентные признаки каждого отдельного узла, мы получаем его единое векторное представление и представляем весь граф.

Если выражаться точнее, то для конкретного узла выбирается функция, которая принимает на вход вектор признаков узла, вектор признаков одного из соседних узлов (связанных с ним ребром) и вектор признаков ребра, соединяющего его с этим соседним узлом, после чего выдает новый вектор, содержащий в себе информацию от узла, соседа и соединяющего их ребра. Применив ту же функцию ко всем соседям узла и сложив полученные векторы, мы получим *вектор сообщения*. Наконец, обновляем вектор признаков нашего узла, комбинируя его исходный вектор признаков с вектором сообщений при помощи функции обновления, также выбранной нами.

После того как мы выполним данную операцию с каждым узлом в графе, каждый новый вектор признаков узла будет содержать информацию о нем самом, всех его соседях и всех соединяющих его ребрах. Повторим этот процесс еще раз, и тогда самый последний вектор признаков узла будет содержать информацию от него самого, всех его соседей и *соседей его соседа*, а также всех соответствующих связующих ребер. Таким образом, чем больше раундов передачи сообщений, тем больше информации содержит вектор признаков узла от более далеких узлов графа, перемещаясь по одному разделению ребер за раунд. Информация последовательно распространяется по всей сети.

Безграничные возможности применения графов

Приложения для графовых нейросетей и графовых моделей в целом настолько распространены и важны, что я даже немного сожалею, что не начала книгу с графов. В случае графовой модели мы начинаем с ответов на следующие вопросы:

- Что такое узлы?
- Каковы отношения, связывающие два узла, которые устанавливают между ними направленные или ненаправленные ребра?
- Должна ли модель включать векторы признаков узлов и/или ребер?
- Является ли модель динамической, т. е. узлы, ребра и их признаки изменяются со временем, или она статична во времени?
- Что нас интересует? Классификация (например, онкология или ее отсутствие; фейк или не фейк)? Генерирование новых графов (например, для исследования лекарственных препаратов)? Кластеризация? Встраивание графа в более низкоразмерное и структурированное пространство?
- Какие данные доступны или необходимы, насколько они организованы и/или размечены? Нужна ли предварительная обработка?

В этом разделе мы рассмотрим всего несколько приложений, хотя существует множество других, превосходно поддающихся графовой структуре моделирования. Будет полезно прочитать аннотации к публикациям по ссылкам — они помогут выявить общие темы и способы осмысления таких моделей. Список общих задач для графовых нейросетей включает:

- классификацию узлов;
- классификацию графов;
- кластеризацию и обнаружение сообществ;
- генерацию новых графов;
- максимизацию влияния;
- предсказание связностей.

Данные изображения в виде графов

Нам часто встречаются графовые нейросети, протестированные на наборе данных рукописных цифр MNIST (**https://oreil.ly/HQL5F**), который считается одним из эталонных наборов для компьютерного зрения. Рассмотрим, как данные изображения (хранящиеся в виде трехмерных тензоров интенсивностей пикселов по каждому каналу) удается уместить в структуру графа. Каждый пиксел — это узел, а его характеристики — соответствующие интенсивности трех каналов (если это цветное изображение, в противном случае у него только один признак). Ребра связывают каждый пиксел с тремя, пятью или восемью окружающими его пикселами в зависимости от того, где он расположен — в углу, на краю или в центре изображения.

Сети головного мозга

Одна из основных задач нейронауки — изучение сетевой организации мозга. Графовые модели обеспечивают естественную структуру и множество инструментов для анализа сложных сетей мозга как с точки зрения их анатомии, так и функциональности.

Для создания ИИ, не уступающего человеческому, необходимо понять человеческий мозг на многих уровнях. Один из аспектов — это сетевая связность мозга, влияние связности на его функциональность и способы ее воспроизведения, начиная с небольших вычислительных единиц, модульных компонентов и заканчивая полностью независимой и функциональной системой.

Анатомические сети человеческого мозга демонстрируют малую длину пути (сокращение издержек на проводку) наряду с кортикальными *узлами* высокой степени, т. е. высокой кластеризацией. Это происходит как на клеточном уровне, так и в масштабах всего головного мозга. Другими словами, похоже, что сеть мозга организована так, чтобы максимизировать эффективность передачи информации и ми-

нимизировать издержки на соединение. Сеть также демонстрирует модульные и иерархические топологические структуры и функциональные возможности. Топологические структуры сетей мозга и их функциональные возможности взаимозависимы как на коротких, так и на длинных временны́х шкалах. Динамические свойства сетей зависят от их структурной связности, а в более длительном временно́м масштабе на топологическую структуру сети влияет динамика.

И здесь возникает ряд важнейших вопросов. Как сетевые свойства мозга связаны с его когнитивным поведением? Как они связаны с психическими расстройствами? Например, можно рассмотреть такое невропсихиатрическое расстройство, как синдром разобщенности, когда теория графов позволяет количественно определить слабые места, уязвимость и аномалии в сетевых структурах. Фактически теория графов применялась в изучении структурных и функциональных свойств сетей при шизофрении, болезни Альцгеймера и других расстройствах.

Распространение заболеваний

Как показал опыт пандемии COVID-19, крайне важно иметь возможность точно и надежно прогнозировать случаи заболевания в целях смягчения последствий, принятия карантинных мер, разработки политики и многих других факторов принятия решений. В графовой модели в качестве узлов могут рассматриваться как отдельные люди, так и целые географические блоки, а в качестве ребер — случаи контактов между этими людьми или блоками. Недавние модели для предсказания распространения COVID-19, например в работе "Комбинирование графовых нейросетей и пространственно-временных моделей болезней в целях улучшения предсказания еженедельных случаев COVID-19 в Германии" (Фриц и др., 2022; **https://oreil.ly/Tkhy7**), включают данные о мобильности пользователей Facebook, Apple и Google для моделирования взаимодействий между узлами в своих моделях.

Здесь содержится множество весьма полезных в использовании данных. Ресурс Facebook "Данные с пользой" (Data for Good, **https://oreil.ly/MJAwr**) располагает огромным количеством данных о плотности населения, социальной мобильности и моделях передвижения, социальной связанности и др. Отчеты Google о мобильности населения в связи с COVID-19 (**https://oreil.ly/2g0mT**) содержат данные Google Карт и других продуктов, которые отражают тенденции перемещения пользователей с течением времени в зависимости от географии по различным категориям мест, таким как магазины и места отдыха, продуктовые магазины и аптеки, парки, транзитные пункты, рабочие и жилые районы. Той же цели служат и данные о мобильности Apple и Amazon — помощь в борьбе с распространением COVID-19.

Распространение информации

Графы можно использовать для моделирования распространения информации, болезней, слухов, сплетен, компьютерных вирусов, инновационных идей и т. д. Такая модель обычно представляет собой ориентированный граф, где каждый узел соот-

ветствует индивидууму, а ребра размечены информацией о взаимодействии между индивидуумами. Метки ребер, или веса, как правило, представляют собой вероятности. Вес w_{ij} ребра, связывающего узел$_i$ с узлом$_j$, — это вероятность того, что определенный эффект (болезнь, слух, компьютерный вирус) распространится от узла$_i$ к узлу$_j$.

Обнаружение и отслеживание распространения фейков

Графовые нейросети лучше справляются с задачей обнаружения фейков (рис. 9.5), чем методы обработки естественного языка на основе контента. Весьма информативна в этом плане аннотация статьи "Обнаружение фейков в социальных сетях с помощью геометрического глубокого обучения" (Монти Ф. и др., 2019; **https://oreil.ly/HQNTq**):

> "Сегодня социальные сети являются одним из основных источников новостей для миллионов людей по всему миру в силу таких качеств, как низкая стоимость, легкий доступ, быстрое распространение. Но за это приходится платить сомнительной надежностью и большой вероятностью столкнуться с фейками, написанными с целью ввести читателей в заблуждение. Автоматическое обнаружение фейков сопряжено с трудностями, с которыми не могут справиться существующие методы анализа контента. Одна из основных причин заключается в том, что зачастую для интерпретации новостей требуется знание политического и социального контекста, или же „здравого смысла“, чего пока не хватает современным алгоритмам обработки естественного языка. Недавние исследования эмпирически показали, что фейковые и настоящие новости по-разному распространяются в социальных сетях, формируя паттерны распространения, которые можно использовать для автоматического обнаружения фейков. Подходы на основе распространения обладают множеством преимуществ по сравнению с аналогами на основе контента, среди которых можно отметить независимость от языка и лучшую устойчивость к атакам соперника. В работе показана новая модель автоматического обнаружения фейков на основе геометрического глубокого обучения. Лежащие в основе модели алгоритмы являются обобщением классических сверточных нейросетей на графы, что позволяет объединять такие разнородные данные, как контент, профиль и активность пользователя, социальный граф и распространение новостей. Модель обучена и протестирована на новостях, распространявшихся в Twitter, которые были проверены профессиональными организациями, занимающимися проверкой фактов. Эксперименты показали, что структура социальной сети и распространение новостей являются важными характеристиками, позволяющими с высокой точностью (92,7% ROC AUC) определять фальшивые новости. Во-вторых, мы наблюдаем, что фейки можно уверенно обнаружить на ранней стадии, всего через несколько часов после распространения. В-третьих, мы проверяем, насколько устарела модель, на обучающих и тестовых данных, разделенных во времени. Полученные результаты свидетельствуют о том, что подходы на основе распространения информации можно использовать в распознавании фейков в качестве альтернативы или дополнения к подходам на основе контента".

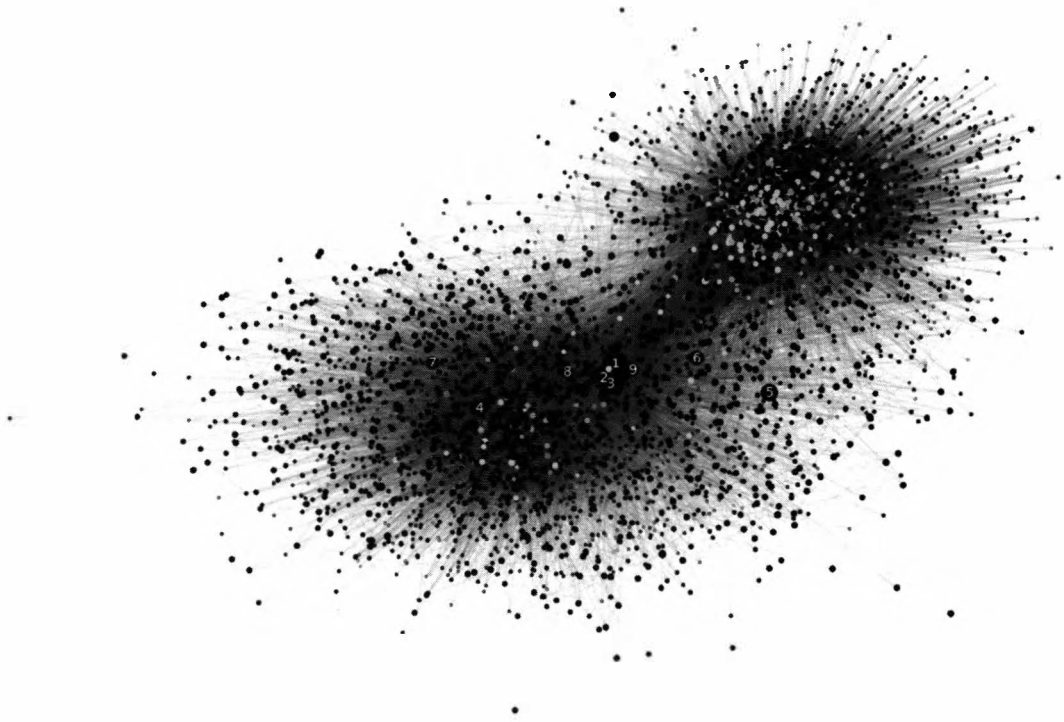


Рис. 9.5. Узлы, распространяющие фейки, помечены красным цветом; люди, думающие одинаково, объединяются в социальные сети (цветную версию изображения см. в источнике: **https://oreil.ly/hfNuf**)

Рекомендательные системы веб-масштаба

С 2018 года сервис Pinterest использует графовую сверточную сеть PinSage (**https://oreil.ly/JQYz4**). Она курирует домашнюю ленту пользователей и предлагает новые и релевантные *пины*. Авторы используют в своей модели *случайные блуждания по графам*, которые мы обсудим далее в этой главе. Приведем полный текст аннотации:

> "Последние достижения в области глубоких нейросетей для данных, имеющих графовую структуру, обеспечили превосходные бенчмарки рекомендательных систем. Однако до сих пор сложной задачей остается практическое применение и масштабирование этих методов для решения задач рекомендации в веб-масштабе, охватывающем миллиарды товаров и сотни миллионов пользователей. В статье рассматривается движок крупномасштабных глубоких рекомендаций, реализованный в Pinterest. Авторами разработан алгоритм графовой сверточной сети (Graph Convolutional Network, GCN) PinSage, который сочетает в себе эффективные случайные блуждания и свертки графов для генерации эмбеддинга, или вставок, узлов (т. е. наименований), включающих как структуру графа, так и информацию о признаках узлов. По сравнению с предыдущими методами GCN новый метод основан на высокоэф-

фективных случайных блужданиях для структурирования сверток, а новая стратегия обучения опирается на все более сложные обучающие примеры в целях повышения надежности и сходимости модели. Алгоритм PinSage развернут на Pinterest и обучен на 7,5 млрд примеров на графе, имеющем 3 млрд узлов, представляющих пины и доски, и 18 млрд ребер. Как показали оффлайн-метрики, изучение пользователей и *A*/*B*-тесты, PinSage генерирует рекомендации более высокого качества, чем его аналоги на основе глубокого обучения и графов. Насколько нам известно, это крупнейшее на сегодняшний день применение эмбеддинга глубоких графов, открывающее дорогу новому поколению рекомендательных систем веб-масштаба на основе графовых сверточных архитектур".

Борьба с онкологией

Авторы статьи "Суперпродукты: машинный интеллект выявляет в продуктах питания молекулы, побеждающие рак" (Веселков К. и др., 2019; **https://oreil.ly/2BfHL**), используя данные о взаимодействии белков, генов и лекарственных препаратов, выявили молекулы, позволяющие предотвратить и победить онкологию. Они также составили карту продуктов питания, наиболее богатых такими молекулами (рис. 9.6). В очередной раз мы видим применение метода случайных блужданий по графам. Приведем аннотацию статьи:

"Последние данные показывают, что до 30–40% раковых заболеваний можно предотвратить только с помощью диеты и образа жизни. Представляем уникальную платформу сетевого машинного обучения сети для выявления предполагаемых молекул, способных победить онкологию с помощью продуктов питания. Такие молекулы выявлены на основе общности их молекулярно-биологических сетей с клинически одобренными противораковыми препаратами. В моделировании действия лекарств на интерактомные сети человека с целью получения геномных профилей активности 1962 одобренных лекарственных препаратов (199 из которых были классифицированы как „противораковые“ с перечнем их основных показаний) использовался алгоритм машинного обучения на основе случайных блужданий по графам (работающий на суперкомпьютерной платформе DreamLab). Для предсказания противораковых молекул с помощью таких „обученных“ профилей активности интерактома применялся контролируемый подход. Протестированная модель предсказала противораковые препараты с точностью классификации 84–90%. В модель загружалась обширная база данных из 7962 биологически активных молекул в составе продуктов питания, что позволило предсказать 110 молекул, способных победить онкологию (определенных по порогу сходства с противораковыми препаратами >70%), с ожидаемой эффективностью, сравнимой с клинически одобренными противораковыми препаратами из различных химических классов, включая флавоноиды, терпеноиды и полифенолы. В результате была составлена „пищевая карта“, где противораковый потенциал каждого ингредиента определялся количеством противораковых молекул. Проведенный анализ позволяет разработать стратегии питания следующего поколения для профилактики и лечения онкологических заболеваний".

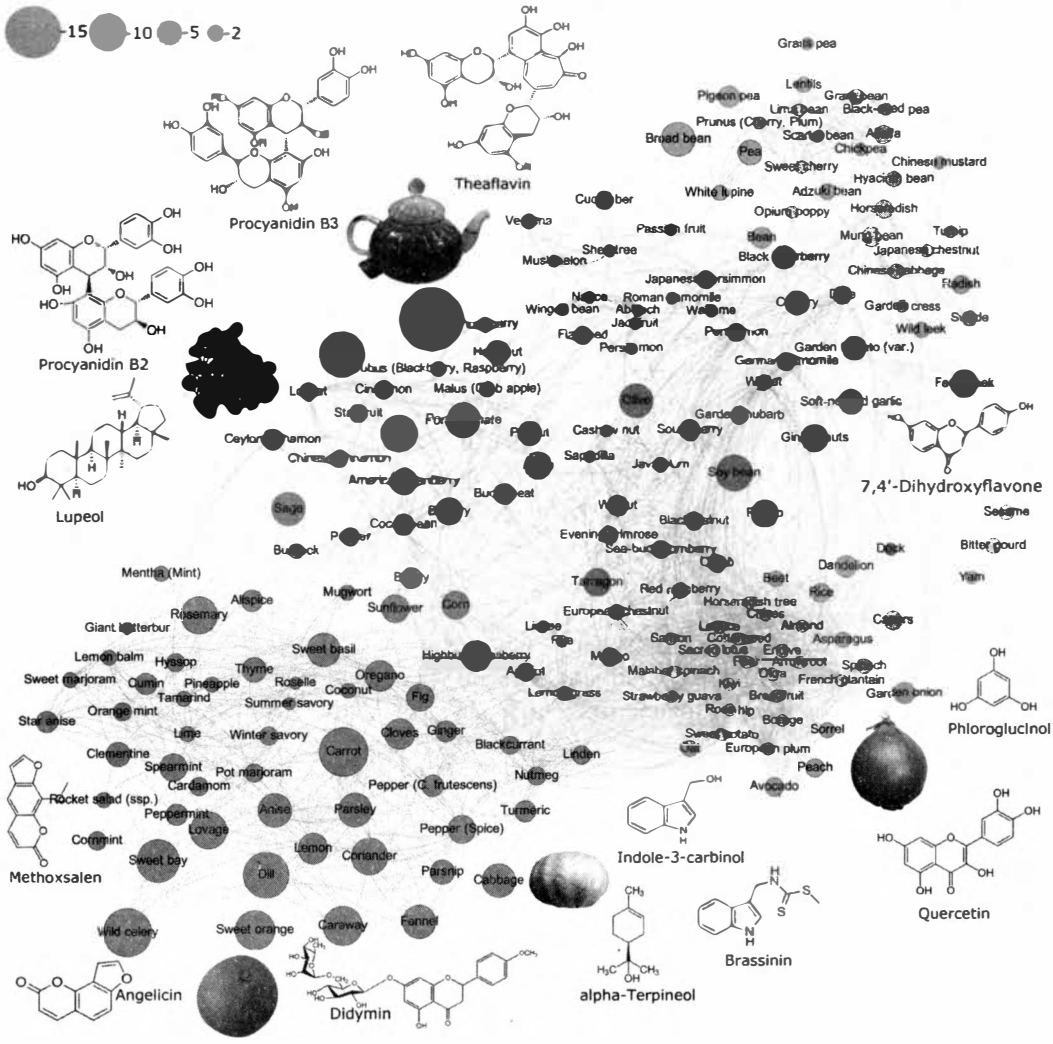


Рис. 9.6. Составленная машинным интеллектом карта противораковых молекул в продуктах питания; чем больше узел, тем разнообразнее набор противораковых молекул (источник изображения: **https://oreil.ly/sGAIp**)

Биохимические графы

Молекулы и химические соединения можно представить в виде графов, где узлы — это атомы, а ребра — химические связи между ними. Наборы данных из этой области химической информатики полезны для оценки эффективности модели классификации. Например, набор данных NCI1 (**https://oreil.ly/LUikV**) содержит около 4100 химических соединений, что полезно для противораковых скринингов, где химические вещества помечаются как положительные или отрицательные в плане противодействия клеточному раку легких. На том же сайте доступны аналогичные размеченные наборы графовых данных о белка́х и других соединениях, а также

публикации, в которых они встречаются, и результаты работы различных моделей на этих наборах данных.

Генерация молекулярных графов для выявления структуры лекарственных препаратов и белков

В предыдущей главе мы узнали, как генеративные сети — вариативные автокодировщики и состязательные сети — обучаются совместным распределениям вероятностей в данных, чтобы генерировать похожие данные для различных целей. Генеративные сети для графов основаны на аналогичных идеях, однако сами по себе они несколько сложнее, чем сети, генерирующие изображения. Генеративные графовые сети генерируют новые графы либо последовательно, выводя узлы и ребра шаг за шагом, либо глобально, сразу выводя матрицу смежности всего графа. Подробности об этом смотрите, например, в обзорной статье по генеративным графовым сетям (Го С., Чжао Л., 2022; **https://oreil.ly/omCsl**).

Сети цитирования

В сетях цитирования узлами могут выступать авторы, а ребрами — их соавторство; или узлами являются статьи, а (направленными) ребрами — цитирование друг друга. Каждая статья имеет направленные ребра, указывающие на статьи, на которые она ссылается. Признаки каждой статьи включают аннотацию, авторов, год, место, название, область исследования и др. Задачи включают кластеризацию узлов, классификацию узлов, предсказание ссылок. К популярным наборам данных для сетей цитирования можно отнести CoRA, CiteSeerX и PubMed. В наборе данных CoRA (**https://oreil.ly/X3J3t**) содержится около 3 тыс. публикаций по машинному обучению, сгруппированные в семь категорий. Каждая статья в сетях цитирования представлена "горячим" вектором, указывающим на наличие или отсутствие слова из предварительно заданного словаря, или вектором TF-IDF. Эти наборы данных постоянно обновляются по мере появления новых публикаций в сети.

Социальные медиасети и прогнозирование общественного воздействия

Социальные медиасети — Facebook, Twitter, Instagram, Reddit — являются отличительной чертой нашего времени (после 2010 года). В качестве примера доступных наборов данных можно привести набор данных Reddit (**https://oreil.ly/uwT6N**). Он представляет собой граф, где узлы — посты, а ребра — связи между двумя постами с комментариями от одного и того же пользователя. Посты также размечены как принадлежащие тому или иному сообществу.

Социальные медиасети оказывают огромное влияние на общество, начиная от рекламы и заканчивая победой на президентских выборах и свержением политических режимов. Одной из важных задач графовой модели, представляющей социальные сети, является предсказание общественного воздействия узлов сети. В данном слу-

чае узлы — это пользователи, а их взаимодействия — ребра. Признаки включают пол, возраст, семейное положение, местоположение, уровень активности пользователей и др. Один из способов количественной оценки общественного воздействия (целевой переменной) заключается в предсказании действий пользователя с учетом действий его ближайших соседей в сети. Например, если друзья пользователя покупают товар, какова вероятность того, что они купят тот же товар через определенный промежуток времени? Общественное воздействие определенных узлов в сети можно предсказать методом случайного блуждания по графам.

Социологические структуры

Социальные диаграммы — это ориентированные графы, представляющие отношения между людьми или группами людей в обществе. Узлы — члены общества или группы, а ориентированные ребра — отношения между ними, например восхищение, ассоциация, влияние и др. Нам нужно узнать связность, разделимость, размер фрагментов и т. д. в этих социальных диаграммах. В качестве примера можно привести антропологические исследования, где ряд племен классифицируется в соответствии со структурой их родства.

Байесовские сети

Мы рассмотрим подробнее байесовские сети позже в текущей главе. Они представляют собой вероятностные графовые модели, цель которых нам хорошо известна из области ИИ — изучить совместное распределение вероятностей признаков набора данных. Байесовские сети рассматривают это совместное распределение вероятностей как произведение распределений одной переменной, *обусловленных только родителями узла* на графе, представляющем связи между признаками данных. То есть узлы — это переменные признаков, а ребра — связи между признаками, которые *мы полагаем* связанными. В качестве примеров применения можно привести фильтрацию спама, распознавание голоса, кодирование и декодирование — и это лишь некоторые из них.

Прогноз трафика

Прогноз трафика — это задача предсказания интенсивности движения с использованием карт дорог, данных о скорости движения, интенсивности трафика. Для отслеживания результатов и сравнения моделей можно воспользоваться контрольными наборами данных о трафике (**https://oreil.ly/LsZ0Q**). Например, набор данных METR-LA (**https://oreil.ly/YU918**) — это пространственно-временной граф, содержащий данные о дорожном движении за четыре месяца, собранные 207 датчиками на автомагистралях округа Лос-Анджелес. Транспортная сеть представляет собой граф, где узлы — датчики, а ребра — участки дорог между датчиками. В определенный момент времени t признаками считаются такие параметры дорожного трафика, как скорость и интенсивность. Задача графовой нейросети — предсказать признаки графа по истечении определенного времени.

В других моделях прогноза трафика применяются байесовские сети (**https://oreil.ly/6mVzy**), например в прогнозе транспортных потоков между соседними транспортными развязками. Модель использует информацию от ближайших развязок для анализа ситуации на конкретных развязках.

Логистика и исследование операций

С помощью графов можно моделировать и решать многие задачи исследования операций, включая транспортные проблемы и сетевые графики. Как правило, речь идет о взвешенных ориентированных графах. Если сеть небольшая, то задачи исследования операций являются, по сути, комбинаторными и всегда тривиальны. Но в случае больших реальных сетей задача состоит в поиске эффективных алгоритмов, способных просеять огромное пространство поиска и быстро исключить из него большие части. Большинство исследовательской литературы посвящено оценке вычислительной сложности таких алгоритмов. Это называется *комбинаторной оптимизацией*. Типичные задачи включают задачу коммивояжера, оптимизацию цепочек поставок, маршрутизацию и тарификацию совместных поездок, поиск работы и др. В их решении используются несколько графовых методов и алгоритмов, среди которых минимальные остовные деревья, кратчайший путь, теорема о максимальном потоке и минимальном разрезе, сопоставление графов. Примеры исследования операций мы рассмотрим позднее.

Языковые модели

Графовые модели актуальны для множества задач, связанных с естественным языком. На первый взгляд эти задачи кажутся разными, но многие из них сводятся к кластеризации, для чего отлично подходят графовые модели.

В любом приложении в первую очередь необходимо выбрать, что конкретно представляют узлы, ребра и признаки. В случае естественного языка такой выбор позволяет выявить скрытые структуры и закономерности как в языке, так и в языковых корпусах.

В случае графовых моделей можно не представлять предложение на естественном языке как последовательность токенов в рекуррентных моделях или как вектор токенов в трансформерах, а встроить предложение в граф (эмбеддинг), после чего применить графовое глубокое обучение (или графовые нейросети).

На рис. 9.7 приведен пример из вычислительной лингвистики — построение диаграмм для синтаксического анализа языка.

Узлы на рисунке — это слова, *n*-граммы или фразы, а ребра — связи между ними, зависящие от грамматики или синтаксиса языка (артикль, существительное, глагол и т. д.). Язык определяется как множество всех строк, которые правильно сгенерированы из языкового словаря по правилам его грамматики. В этом смысле компьютерные языки легче поддаются синтаксическому анализу (они так устроены), чем естественные языки, которые по своей природе гораздо сложнее.

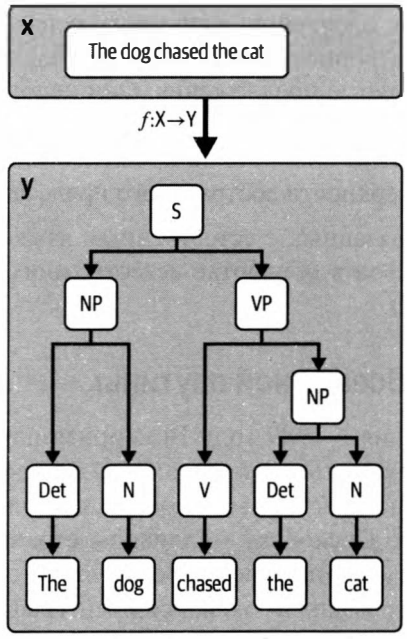


Рис. 9.7. Синтаксический анализ предложения[2]

Синтаксический анализ

Синтаксический анализ (парсинг) означает преобразование потока входных данных в структурированное или формальное представление в целях автоматической обработки. Входными данными для *синтаксического анализатора (парсера)* могут быть предложения, слова или даже символы. На выходе получается древовидная диаграмма, содержащая информацию о функции каждой части входного сигнала. Прекрасным синтаксическим анализатором языкового материала является наш мозг, в то время как компьютеры неплохо выполняют синтаксический разбор языков программирования.

Другие примеры — кластеризация новостей или рекомендации статей. В этих случаях для определения сходства текстов используется встраивание (эмбеддинг) графов текстовых данных. Узлами могут служить слова, а ребрами — семантические отношения между словами или просто их совпадения. Или же узлами могут быть слова и документы, а ребрами — семантические отношения или отношения совместной встречаемости. Признаки узлов и ребер могут включать авторов, темы, временны́е периоды и др. В таких графах естественным образом возникают кластеры.

Другой вид парсинга, не зависящий от синтаксиса или грамматики языка, — это абстрактное представление значения (abstract meaning representation, AMR). Оно опирается на семантическое представление в том смысле, что схожим по значению предложениям должно быть присвоено одно и то же абстрактное представление

[2] Собака преследовала кошку. — *Прим. ред.*

значения, даже если они не идентичны по формулировке. Графы абстрактного представления значения — это корневые размеченные ориентированные ациклические графы, представляющие полные предложения. Они используются в машинном переводе и изучении естественного языка. Существуют общедоступные наборы данных, а также разнообразные пакеты и библиотеки, предназначенные для парсинга, визуализации и формирования поверхности абстрактного представления значения.

О других приложениях, связанных с естественным языком, можно прочитать в обзорной статье "Обзор графов в обработке естественного языка" (Настасе В. и др., 2015; **https://oreil.ly/2Birx**).

Графовая структура Всемирной паутины

С момента своего появления в 1989 году Всемирная паутина сильно разрослась и стала незаменимым инструментом для миллиардов людей по всему миру. С помощью интернет-браузера она обеспечивает доступ к миллиардам веб-страниц, документов и других ресурсов. Поскольку миллиарды страниц связаны друг с другом, изучение графовой структуры Интернета представляет огромный интерес. С точки зрения математики этот громадный экспансивный граф увлекателен сам по себе. Но понимание этого графа важно не только как изящное упражнение для ума — оно позволяет понять алгоритмы сканирования, индексирования и ранжирования веб-страниц (как в алгоритме PageRank, который мы рассматривали ранее в этой главе), поиска сообществ и выявления социальных и других явлений, характеризующих их рост или упадок.

Граф Всемирной паутины имеет:

узлы

веб-страницы, чьи масштабы исчисляются миллиардами;

ребра

направления от одной страницы к другой в масштабах сотен миллиардов.

Нас интересуют следующие аспекты.

- Средняя степень узлов.
- Распределения степеней узлов (как для входящих, так и для исходящих степеней, которые могут сильно различаться). Представляют ли они степенной закон или какие-то другие законы?
- Связность графа — процент связанных пар.
- Средние расстояния между узлами.
- Наличие зависимости наблюдаемой структуры сети от конкретного используемого краулера.
- Особые структуры слабо и сильно связанных компонентов.
- Наличие гигантского сильно связанного компонента. Какова доля узлов, которые могут достигать или быть достижимыми из этого гигантского компонента?

Автоматический анализ компьютерных программ

Графы можно использовать для верификации и обоснования компьютерных программ, теории надежности, в диагностике неисправностей компьютеров и изучении структуры компьютерной памяти. В качестве примера можно привести работу "Графовые нейросети в анализе программ" (Алламанис М., 2021; **https://oreil.ly/ZJyeo**):

> "Цель анализа программы — определение соответствия поведения программы конкретной спецификации. Как правило, именно человек должен определять и настраивать анализ программы, что представляет собой дорогостоящий процесс. Недавно методы машинного обучения показали перспективы вероятностной реализации различных видов анализа программ. Учитывая структурированный характер программ и общность графовых представлений в их анализе, графовые нейросети (graph neural networks, GNN) предлагают оригинальный способ представления, обучения и обоснования программ и широко используются в анализе программ на основе машинного обучения. В статье рассматривается использование графовых нейросетей в анализе программ и выделяются два практических случая применения — вывод типа и выявление злоупотребления переменными".

Структуры данных в вычислительной технике

В области вычислительной техники структура данных — это структура, которая хранит, управляет и организует данные. Существуют различные структуры данных, и обычно они выбираются так, чтобы обеспечить эффективный доступ к данным (чтение, запись, добавление, вывод или хранение отношений и т. д.).

Некоторые структуры данных используют графы для организации данных, вычислительных устройств в кластере, представления потока данных и вычислений, коммуникационной сети. Существуют также графовые базы данных, предназначенные для хранения и запроса графовых данных.

Другие базы данных трансформируют графовые данные в более структурированные форматы (например, реляционные).

Приведем несколько примеров структур графовых данных.

Алгоритм PageRank.

> Мы уже встречались с алгоритмом PageRank, а также со структурой ссылок на веб-сайт, представленной в виде ориентированного графа, где узлы — веб-страницы, а ребра — ссылки с одной страницы на другую. База данных, хранящая все веб-страницы вместе с их структурами ссылок, либо может быть графовой, где граф хранится как есть, используя матрицу связей или матрицу смежности без необходимости в трансформере, либо может быть трансформирована под структуру других неграфических баз данных.

Двоичные деревья поиска для организации файлов в базе данных.

> Двоичные деревья поиска — это упорядоченные структуры данных, эффективные как для случайного, так и для последовательного доступа к записям, а также

для модификации файлов. Свойственный двоичному дереву поиска порядок ускоряет время поиска — на каждом уровне дерева объем данных для сортировки сокращается вдвое. При этом также ускоряется время вставки — мы создаем новый фрагмент в памяти и ссылку на него, а не добавляем узел в структуру данных двоичного дерева как в случае с массивом. Так получается быстрее, чем создавать новый большой массив, а затем вставлять в него данные из меньшего массива.

Информационно-поисковые системы на основе графов.

В некоторых информационно-поисковых системах каждому документу присваивается определенное количество индексных терминов. Можно считать их индикаторами, дескрипторами или ключевыми словами документа. Индексные термины будут представлены в виде узлов графа. Если два индексных термина тесно связаны друг с другом, как, например, индексные *граф* и *сеть*, то после их соединения неориентированным ребром получается довольно большой и, возможно, несвязный *граф подобия*. Максимально связанные подграфы этого графа являются его *компонентами*, которые естественным образом классифицируют документы в этой системе. Для поиска информации в запросе указываются индексные термины, т. е. определенные узлы графа, и система возвращает максимально полный подграф, включающий соответствующие узлы. Таким образом, получается полный список индексных терминов, которые, в свою очередь, определяют нужные нам документы.

Балансировка нагрузки в распределенных сетях

Вычислительный прогресс прошел путь от закона Мура до параллельных и облачных вычислений. В облачных вычислениях наши данные, файлы и машины, работающие с файлами и производящие вычисления на наших данных, располагаются далеко от нас. Они даже находятся вдали друг от друга. С каждым днем приложения становятся все сложнее, а сетевой трафик растет, поэтому нам нужен программный или аппаратный аналог регулировщика сетевого трафика, который распределяет сетевой трафик между несколькими серверами так, чтобы ни один сервер не испытывал большой нагрузки, повышая производительность в плане времени отклика приложений, удобства работы конечных пользователей и т. д.

По мере роста трафика возникает необходимость добавлять все больше устройств, или узлов, чтобы справиться с объемом. Распределение сетевого трафика должно осуществляться с сохранением безопасности и конфиденциальности данных, а также быть способным предсказать узкие места в трафике до их возникновения. Эту задачу решают *балансировщики нагрузки*. Можно легко представить распределенную сеть в виде графа, узлами которого являются подключенные серверы и устройства. Тогда балансировка нагрузки — это проблема распределения трафика на данном графе, и существует множество алгоритмов распределения нагрузки. Все алгоритмы работают на графе сети. Некоторые из них статичны и распределяют

нагрузку без обновления текущего состояния сети в плане нагрузки или неисправных устройств, в то время как другие динамичны, но требуют постоянной связи внутри сети о состоянии узлов. Приведем несколько алгоритмов.

Алгоритм наименьшего количества соединений.

Направляет трафик на сервер с наименьшим количеством активных соединений.

Алгоритм наименьшего времени отклика.

Направляет трафик на сервер с наименьшим количеством активных соединений и наименьшим средним временем отклика.

Циклический алгоритм.

По очереди распределяет нагрузку на серверы. Он направляет трафик на первый доступный сервер, а затем перемещает его в конец очереди.

Алгоритм IP Hash.

Распределяет серверы на основе IP-адреса клиента.

Искусственные нейросети

Искусственные нейросети (artificial neural networks, ANN) — это графы, где узлы представляют собой вычислительные блоки, а ребра — входы и выходы этих блоков. На рис. 9.8 показаны популярные модели искусственных нейросетей в виде графов.

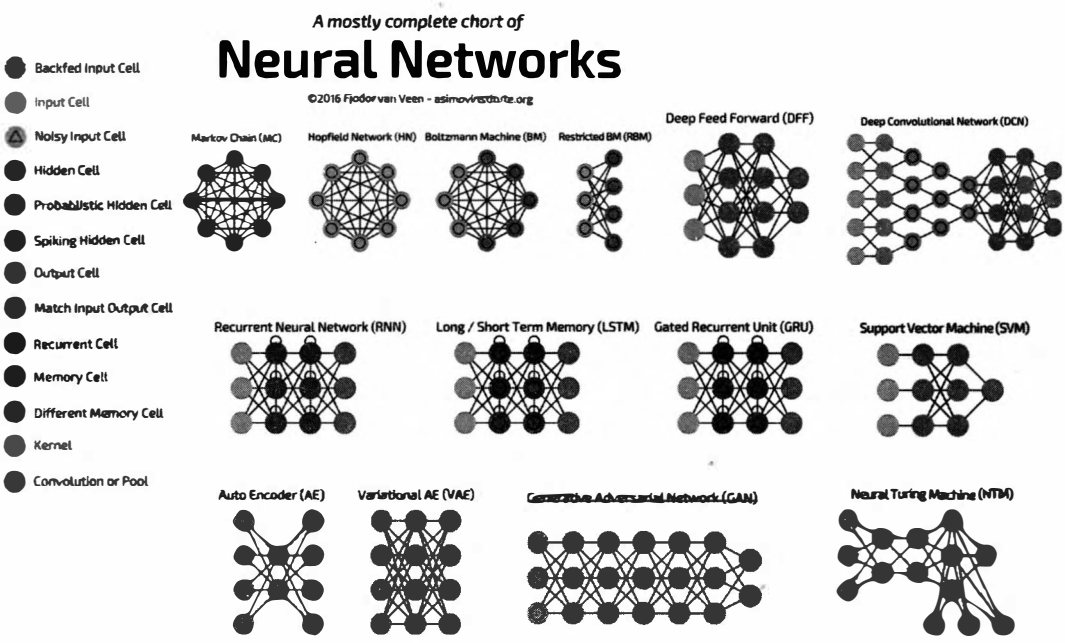


Рис. 9.8. Нейросети в виде графов
(источник изображения: **https://oreil.ly/e6HQp**)

Случайные блуждания по графам

Случайное блуждание по графу (рис. 9.9) означает практически дословно последовательность шагов, которая начинается в некоторой вершине, и на каждом временно́м шаге выбирается соседняя вершина (при помощи матрицы смежности) с вероятностью, пропорциональной весам ребер, куда перемещается процесс.

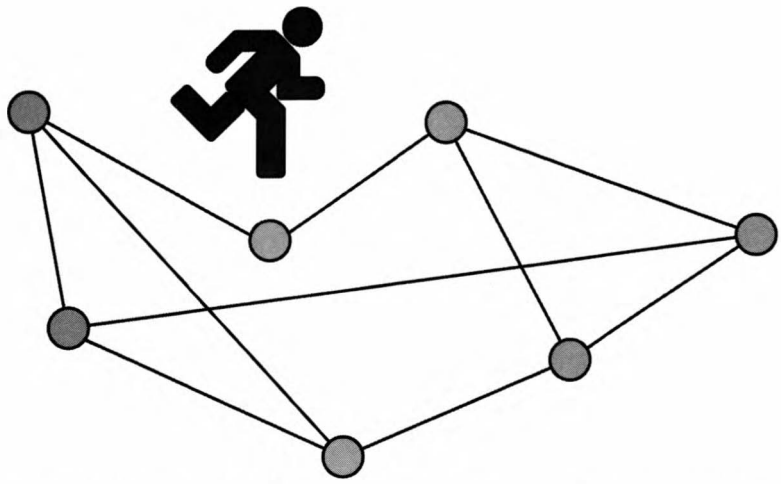


Рис. 9.9. Случайное блуждание по неориентированному графу

В случае невзвешенных ребер для перемещения шага с равной вероятностью будут выбраны все соседние узлы. На любом временно́м шаге блуждание может не переместиться к одному из соседей, а остаться в том же самом узле при наличии собственного ребра или в случае ленивого случайного блуждания с положительной вероятностью. Нам нужно выяснить следующее.

- Порядок прохождения узлов в процессе случайного блуждания. В данном случае начальная точка и структура графа имеют значение в плане того, сколько узлов охватит блуждание и можно ли вообще достичь определенных областей графа. В графовых нейросетях нас интересует обучение представления конкретного узла на основе признаков его соседей. В случае больших графов, когда узлы имеют больше соседей, чем это возможно с точки зрения вычислений, применяется случайное блуждание. Однако нужно соблюдать осторожность, т. к. в разных частях графа процесс случайного блуждания будет происходить с разной скоростью, и без учета этого путем регулировки количества шагов случайного блуждания в зависимости от структуры подграфа могут получиться представления узлов низкого качества наряду с нежелательными результатами по мере продвижения по конвейеру.
- Поведение случайного блуждания, т. е. распределение вероятностей посещенных вершин после определенного количества шагов. Такие характеристики случайного блуждания, как долгосрочное поведение, можно изучить при помощи *спектра графа*, представляющего собой набор собственных значений матрицы

смежности. В целом спектр оператора позволяет понять, что происходит при многократном применении оператора. Случайное блуждание по графу эквивалентно многократному применению нормализованной версии матрицы смежности к узлу начального графа. Каждый раз, применяя *матрицу случайного блуждания*, мы проходим по графу на один шаг дальше.

- Поведение случайного блуждания на различных типах графов (маршруты, деревья, два полносвязных графа, соединенных одним ребром, бесконечные графы и др.).
- Вернется ли случайное блуждание в начальную точку? Если да, то сколько времени займет процесс до возвращения?
- Каково время блуждания до конкретного узла?
- Каково время прохождения всех узлов?
- Как *распространяется* случайное блуждание. То есть распределение влияния конкретных узлов, находящихся в конкретных областях графа. Масштаб этого влияния.
- Возможно ли на основе случайного блуждания разработать алгоритмы, способные достигать нечеткие участки больших графов?

Случайные блуждания и броуновское движение

В пределе, когда размер шагов случайного блуждания стремится к нулю, получается броуновское движение. Броуновское движение моделирует случайные колебания частиц, взвешенных в какой-либо среде, например в жидкости, или колебания цен на деривативы на финансовых рынках. Термин "броуновское движение" часто встречается с термином *"винеровский процесс"*, представляющим собой непрерывный стохастический процесс с четким математическим определением того, как начинается (с нуля) движение (частицы или колебания цен на финансовых рынках), как происходит выборка следующего шага (из нормального распределения с независимыми приращениями), и предположениями о его непрерывности как функции времени (он почти наверняка непрерывен). Еще один связанный с ним термин — *мартингал*. Мы рассмотрим их в *главе 11*, посвященной вероятности.

В этой главе мы уже встречались со случайным блужданием при обсуждении алгоритма PageRank, когда посетитель веб-страницы в Интернете случайно решает перейти с нее на соседнюю. Мы отметили, что поведение долгосрочного блуждания обнаруживается при многократном применении матрицы ссылок графа, которая представляет собой матрицу смежности, нормированную на степени каждого узла. В следующем разделе мы рассмотрим другие варианты использования случайного блуждания в графовых нейросетях.

Случайное блуждание (на ориентированных или неориентированных, взвешенных или невзвешенных графах) применяется для обнаружения сообществ и мак-

симизации влияния в небольших сетях, где будет нужна только матрица смежности графа (в отличие от встраивания узлов в векторы признаков с последующей кластеризацией).

Обучение представлению узлов

Прежде чем реализовывать графовые задачи на машине, необходимо научиться представлять узлы графа в виде векторов, содержащих информацию об их местоположении в графе и признаках, связанных с их местоположением. Вектор представления узла составляется, как правило, из собственных признаков узла и признаков его соседей.

Существуют различные способы объединения и преобразования признаков, или даже выбора того, какие из соседних узлов внесут вклад в представление признаков данного узла. Рассмотрим некоторые из них.

- Традиционные методы представления узлов опираются на сводную статистику подграфов.
- В других методах предполагается одинаковое векторное представление узлов, через которые проходит краткосрочное случайное блуждание.
- Еще одни методы учитывают, что случайное блуждание может по-разному распространяться по различным подструктурам графа, поэтому способ представления узла адаптируется к локальной подструктуре, в которую входит узел, определяя соответствующий радиус влияния для каждого узла в зависимости от топологии подграфа, к которому он принадлежит.
- Есть методы, создающие многомасштабное представление путем умножения вектора признаков узла на значения матрицы случайных блужданий.
- Прочие методы позволяют нелинейно объединять векторы признаков узла и его соседей.

Необходимо также определить, насколько велико окружение узла, из которого он черпает информацию (распределение влияния), т. е. найти диапазон узлов, чьи признаки влияют на представление данного узла. Это аналогично анализу чувствительности в статистике, только здесь нужно определить чувствительность представления узла к изменениям в признаках окружающих его узлов.

В процессе обучения составленный вектор представления узлов вместе с другими признаками данных передается для классификации в другую модель машинного обучения, например в модель машины опорных векторов. В частности, можно выучить вектор признаков каждого пользователя социальной сети и затем передать эти векторы вместе с другими признаками в модель классификации для предсказания того, является ли конкретный пользователь распространителем фейков. Хотя для классификации узлов необязательно опираться на модель машинного обучения. Это можно сделать непосредственно на основе структурных графовых данных, когда можно предсказать класс узла в зависимости от его связи с другими локальны-

ми узлами. Граф может быть размечен лишь частично, и задача состоит в том, чтобы предсказать остальные метки. Более того, шаг представления узлов может быть либо шагом предварительной обработки, либо одной из частей сквозной модели, подобной графовой нейросети.

Задачи для графовых нейросетей

После знакомства с формулировкой графов в линейной алгебре, приложениями графовых моделей, случайными блужданиями по графам и векторными представлениями узлов, которые кодируют особенности узлов вместе с их зонами влияния в графе, необходимо получить представление о том, какие задачи могут решать графовые нейросети. Рассмотрим некоторые из них.

Классификация узлов

Рассмотрим примеры задач классификации узлов.

- Классификация научных статей по определенной дисциплине в сети цитирования научных статей (например, в CiteSeerX или CoRA), где узлы — статьи (заданные как векторы мешков слов), а направленные ребра — ссылки между ними.
- Классификация постов по принадлежности к сообществам в наборе данных Reddit, где узлы — комментарии (заданные в виде векторов слов), а неориентированные ребра — комментарии, размещенные одним и тем же пользователем.
- Набор данных сети белок-белковых взаимодействий (ББВ) содержит 24 графа, где узлы размечены наборами онтологии генов. (Не обращайте внимания на медицинские технические названия, сосредоточьтесь на математике. Математическое моделирование как раз и прекрасно тем, что работает одинаково для всех видов приложений из всех областей, что делает его потенциальным базовым языком Вселенной.) Как правило, из набора данных ББВ-сети 20 графов используются для обучения, 2 графа — для валидации, а остальные — для тестирования, причем каждый из графов соответствует человеческой ткани. Признаки, связанные с узлами, — наборы позиционных генов, наборы мотивных генов, иммунологические показатели. Классифицируем каждый узел в соответствии с набором генных онтологий.
- Сети мониторинга торговли дикорастущими растениями и дикими животными, которые анализируют динамику тенденций, используя и обновляя данные, например Traffic.org (**https://www.traffic.org/**), CITES Wildlife Trade Database (**https://oreil.ly/wj2o1**), USDA Ag Data Commons (**https://oreil.ly/PKD3D**, в этот набор данных входит более миллиона поставок объектов дикой природы, представляющих более 60 биологических классов и более 3,2 млрд живых организмов). Одна из задач классификации торговой сети заключается в том, чтобы классифицировать каждый узел, представляющий участника торговой сделки

(покупателя или продавца) как занимающегося или не занимающегося незаконной торговлей. Ребра в сети представляют торговую операцию между покупателем и продавцом. Признаки узлов содержат личную информацию участников сделки, номера банковских счетов, местоположение и т. д., а признаки ребер — идентификационные номера транзакций, даты, ценники, купленные/проданные виды и т. д. При наличии поднабора участников, размеченных как нелегальные, задача модели заключается в предсказании меток других узлов на основе их связей с остальными узлами (и их признаками) в сети.

Примеры классификации узлов легко поддаются полуконтролируемому обучению, когда только несколько узлов в наборе данных имеют свои метки, и задача состоит в разметке остальных узлов. Чистые размеченные данные — это главное, за что нужно бороться, добиваясь повышения точности, надежности, прозрачности моделей.

Классификация графов

Иногда требуется разметить не отдельные узлы, а весь граф целиком. Например, в наборе данных PROTEINS о структуре белков содержится коллекция химических соединений, каждое из которых представлено в виде графа и размечено как фермент или не фермент. В графовую модель обучения вводятся узлы, ребра, их признаки, структура и разметка каждого графа из набора данных, что дает в результате представление всего графа или эмбеддинга вместо представления одного узла.

Кластеризация и обнаружение сообществ

Кластеризация в графах — важная задача, позволяющая обнаружить в сетях сообщества или группы, например террористические организации. Один из способов заключается в создании представлений узлов и графов с их последующей передачей в традиционные методы кластеризации, например кластеризация k средних. Другие способы создают представления узлов и графов, учитывающие в своей конструкции задачу кластеризации. К ним относятся конструкции *кодера — декодера* и *механизмы внимания* по аналогии с методами, рассмотренными в предыдущих главах. Другие методы являются спектральными, т. е. они опираются на собственные значения графового лапласиана. Следует отметить, что в случаях неграфовых данных одним из методов, применявшихся для кластеризации, был анализ главных компонент, который также является спектральным и опирается на сингулярные значения таблицы данных. В любом случае вычисление собственных значений — дорогостоящая операция, так что задача сводится к поиску обходных путей. В случае графов можно воспользоваться методами из теории графов, например методом *максимального потока и минимального разреза* (мы рассмотрим это позже в текущей главе). Каждый метод имеет свои достоинства и недостатки; например, некоторые из них используют проверенные временем результаты теории графов, но при этом не учитывают признаки узлов и ребер, т. к. теория разрабатывалась без учета каких-либо признаков, не говоря уже о целых кластерах. Посыл заключается в том, чтобы в каждом случае у нас было правильное представление о том, что учитывают и не учитывают конкретные модели.

Генерация графов

Генерация графов крайне важна для исследования лекарственных препаратов, проектирования материалов и ряда других приложений. В традиционных способах генерации графов используются семейства моделей случайных графов, созданные вручную путем простого стохастического процесса генерации. Благодаря своим простым свойствам, эти модели хорошо понятны с точки зрения математики. Однако по той же самой причине они не в состоянии отражать реальные графы с более сложными зависимостями или даже верные статистические свойства, например *распределение с толстыми хвостами* для степеней узлов, которые проявляются во многих реально существующих сетях. Более современные подходы, такие как генеративные графовые нейросети, объединяют представления графов и узлов с генеративными моделями из предыдущей главы. Они способны легко выучить структурную информацию из данных и генерировать сложные графы, например молекулы и соединения.

Максимизация влияния

Максимизация влияния — это подобласть сетевого распространения, когда целью является максимизация распространения через сеть чего-либо, например информации или агитации по вакциноции, причем путем их передачи только нескольким начальным узлам, или *семенам*. Задача состоит в поиске нескольких узлов, оказывающих максимальное влияние. На практике такой подход используется в приложениях, распространяющих различную информацию, например вакансии, новости, рекламу, призывы к вакцинации. Традиционные методы поиска семян заключаются в выборе узлов на основе наибольшей степени, близости, связанности и других свойств структуры графа. В других методах используется дискретная оптимизация, демонстрирующая отличные результаты и оправдывающая существование приближенных оптимизаторов. В более современных подходах применяются графовые и состязательные нейросети, когда другие задачи конкурируют друг с другом с целью максимизации влияния узла, как, например, для охвата определенной части населения, скажем группы меньшинств, которая необязательно сильно связана с естественными узлами графа.

Предсказание ссылок

Когда имеются две вершины графа, какова вероятность того, что они связаны ребром? Важно отметить, что близость в смысле наличия общих соседей необязательно является показателем связности (или взаимодействия). В социальной сети люди, как правило, вращаются в одних и тех же кругах, поэтому два человека, имеющих много общих друзей, скорее всего, будут взаимодействовать и, скорее всего, также будут связаны. Но в некоторых биологических системах, например при исследовании белок-белковых взаимодействий, верным будет обратное — белки, имеющие больше общих соседей, будут взаимодействовать с меньшей вероятностью. Поэтому подсчет показателей сходства на основе таких базовых свойств, как расстояние

между графами, их степени, общие соседи и т. д., не всегда дает правильные результаты. Нужны нейросети для обучения эмбеддингов узлов и графов, а также для классификации связанности узлов. Один из примеров таких сетей приведен в статье "Предсказание ссылок с помощью графовых нейросетей и извлечения знаний" (Чжан З. и др., 2020; **https://oreil.ly/rVvgg**).

Динамические графовые модели

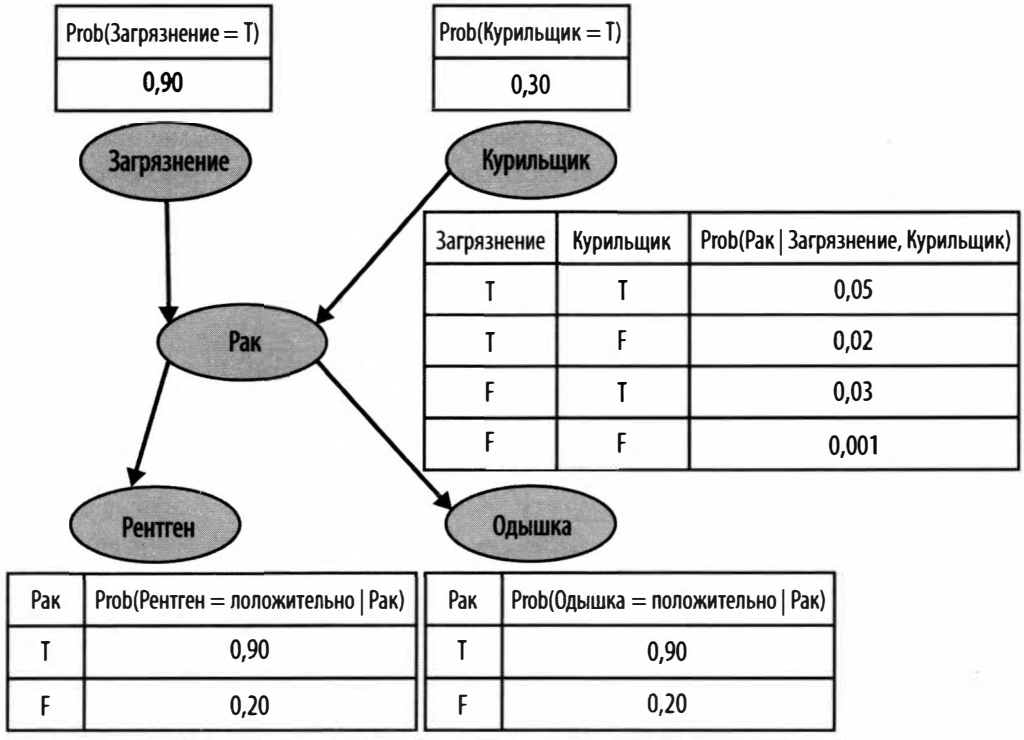
Многие из приложений, рассмотренных в этой главе, значительно выиграют, если мы включим в динамичные по своей сути графовые модели временну́ю зависимость. В качестве примеров можно привести прогноз трафика, балансировку нагрузки в распределенных сетях, моделирование всевозможных систем взаимодействующих частиц, мониторинг незаконной торговли дикими животными. В динамических графовых моделях признакам узлов и ребер разрешено изменяться со временем, а некоторые узлы или ребра можно добавлять или удалять. Такое моделирование позволяет получить информацию типа последних рыночных тенденций и колебаний, новой волны сетевых преступлений, а также новых маршрутов или соединений в транспортных системах.

Размышления о том, как моделировать динамические графы и извлекать из них информацию, не являются чем-то новым; см. например, статью "Динамические графовые модели" (Хэррей Ф. и др., 1997; **https://oreil.ly/YHvpp**). С применением глубокого обучения извлекать знания из подобных систем стало намного проще. Современные подходы к динамическим графам объединяют в себе свертки графов для выявления пространственных зависимостей и рекуррентные или сверточные нейросети для моделирования временны́х зависимостей.

В качестве отличного примера с превосходными результатами высокого разрешения можно привести статью "Обучение моделированию сложной физики с помощью графовых сетей" (Санчес-Гонсалес А. и др., 2020; **https://oreil.ly/Ui3IF**), где динамическая графовая нейросеть используется для моделирования любой системы взаимодействующих частиц в гораздо бо́льших масштабах как по количеству участвующих частиц, так и по времени, в течение которого системе разрешено (численно) эволюционировать, по сравнению с тем, как было раньше. Частицы, например песок или капли воды, — это узлы графа с такими признаками, как местоположение, скорость, давление, внешние силы и т. д., а ребра соединяют частицы, которым разрешено взаимодействовать друг с другом. На вход нейросети подается граф, и на выходе получается граф с теми же узлами и ребрами, только с обновленными признаками местоположения и свойств частиц. На каждом временно́м шаге сеть обучается динамике, или правилу обновления, посредством передачи сообщений. Правило обновления зависит от состояния системы на текущем временно́м шаге и от параметризованной функции, чьи параметры оптимизируются для конкретной цели обучения, зависящей от конкретного приложения, и это является основным шагом в любой нейросети. Целями предсказания при контролируемом обучении выступают средние значения ускорения каждой частицы.

Байесовские сети

Байесовские сети — это графы, которые идеально подходят для работы с неопределенностью, кодируя вероятности математически обоснованным способом. На рис. 9.10 и 9.11 показаны примеры двух байесовских сетей. В *главе 8* было представлено несколько моделей, которые пытаются изучить совместное распределение вероятностей — от явных и легко выполнимых до неявных и трудновыполнимых. Байесовская сеть представляет собой простую явную модель совместного распределения вероятностей, где явно указано, как каждая переменная взаимодействует с другими.



Загрязнение	Курильщик	Prob(Рак | Загрязнение, Курильщик)
T	T	0,05
T	F	0,02
F	T	0,03
F	F	0,001

Рак	Prob(Рентген = положительно | Рак)
T	0,90
F	0,20

Рак	Prob(Одышка = положительно | Рак)
T	0,90
F	0,20

Рис. 9.10. Байесовская сеть[3]

В байесовской сети:

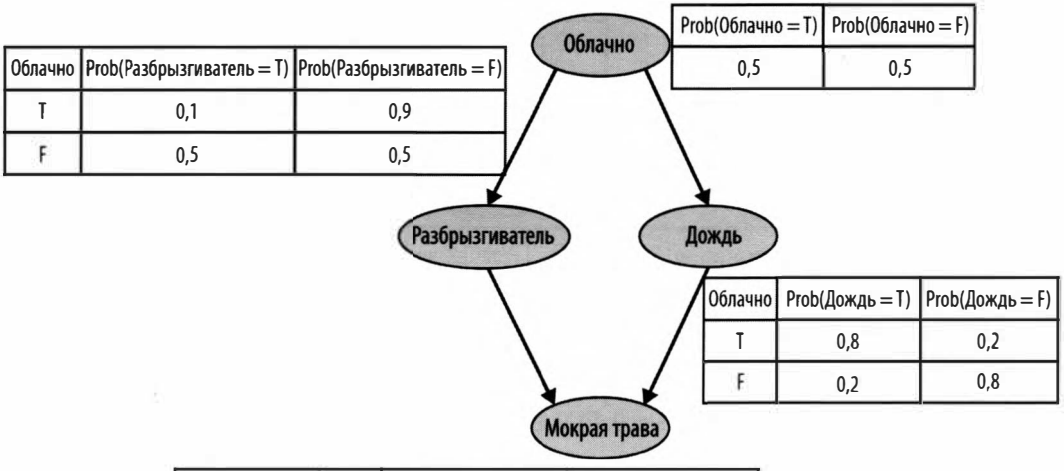
- узлы — переменные, которые, по нашему мнению, должна содержать модель;
- ребра направлены от *родительского* узла к *дочернему* узлу или от *вышестоящего* нейрона к *нижестоящему* в том смысле, что мы знаем зависимость дочерней переменной от наблюдения родительской переменной;

[3] На рисунке Prob (от англ. *probability*) означает вероятность. — *Прим. ред.*

- в графе сети не допускаются циклы;
- строгое следование правилу Байеса: если имеется стрелка от A к B, то $P(B|A)$ — это прямая вероятность, а $P(A|B)$ — обратная вероятность. Допустим, имеется P(доказательства | гипотеза) или P(симптомы | болезнь). По правилу Байеса, обратная вероятность рассчитывается как:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)};$$

- если на переменную не указывает стрелка (если у нее нет родителей), то нужно знать лишь *априорную вероятность* этой переменной, которая вычисляется на основе данных или экспертных знаний (например, 13% женщин в США страдают раком молочной железы);
- в случаях когда имеется больше данных по одной из переменных в модели, или больше *доказательств*, мы обновляем узел, соответствующий этой переменной (условную вероятность), после чего распространяем эту информацию по связям в сети, обновляя условные вероятности в каждом узле двумя разными способами в зависимости от того, в каком направлении распространяется информация — от родительского элемента к дочернему, или наоборот. При этом обновление в каждом направлении достаточно простое — соблюдение правила Байеса.



Prob(Облачно = T)	Prob(Облачно = F)
0,5	0,5

Облачно	Prob(Разбрызгиватель = T)	Prob(Разбрызгиватель = F)
T	0,1	0,9
F	0,5	0,5

Облачно	Prob(Дождь = T)	Prob(Дождь = F)
T	0,8	0,2
F	0,2	0,8

Разбрызгиватель	Дождь	Prob(Мокрая трава = T)	Prob(Мокрая трава = F)
T	T	0,99	0,01
T	F	0,9	0,1
F	T	0,9	0,1
F	F	0,0	1,0

Рис. 9.11. Другая байесовская сеть

Байесовская сеть как компактная таблица условных вероятностей

Что ж, байесовская сеть представляет собой компактную таблицу условных вероятностей. Обычно, когда мы моделируем сценарий реального мира, каждая дискретная переменная может принимать определенные дискретные значения или категории, а каждая непрерывная переменная — любое значение в заданном непрерывном диапазоне. Теоретически можно построить гигантскую таблицу условных вероятностей, которая дает вероятность того, что каждая переменная примет определенное состояние при фиксированных значениях других переменных. В реальности, даже для достаточно малого числа переменных, это невыполнимо и дорого как в плане хранения, так и в плане вычислений. Кроме того, у нас нет доступа ко всей информации, необходимой для построения таблицы. Байесовские сети обходят это препятствие, позволяя переменным взаимодействовать только с несколькими соседями, так что остается лишь вычислить вероятность переменной с учетом состояний переменных, непосредственно связанных с ней в графе сети, как в прямом, так и в обратном направлении. При появлении в сети нового доказательства какой-либо переменной структура графа по правилу Байеса позволяет систематическим, объяснимым, прозрачным способом обновить вероятности всех переменных в сети. Такое *разрежение* сети — это признак байесовских сетей, благодаря которому они получили популярность.

В целом граф байесовской сети задает совместное распределение вероятностей переменных модели (или признаков данных) как произведение локальных условных распределений вероятностей, по одному для каждого узла:

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i \mid \text{родители } x_i).$$

Создание предсказаний при помощи байесовской сети

После создания байесовской сети и определения условных вероятностей (а также их постоянного обновления с помощью дополнительных данных) можно быстро получить результат посредством простого поиска в таблицах распределения условных вероятностей по правилу Байеса или правилу произведения — в зависимости от запроса. Например, вероятность того, что письмо является спамом, учитывая содержащиеся в нем слова, местоположение отправителя, время суток, имеющиеся ссылки, историю взаимодействия между отправителем и получателем и другие значения переменных для обнаружения спама. Или вероятность того, что у пациентки рак молочной железы, учитывая результаты маммографии, семейный анамнез, симптомы, анализы крови и т. д. Преимуществом является то, что для получения результатов не нужно тратить энергию на выполнение больших программ или задействовать огромные кластеры компьютеров. Это значит, что аккумуляторы телефонов и планшетов прослужат дольше, т. к. им не придется тратить много вычислительной мощности на кодирование и декодирование сообщений; вместо этого

байесовские сети позволяют использовать алгоритм прямой коррекции ошибок с *турбодекодированием* (**https://oreil.ly/nu1LF**).

Байесовские сети — это не каузальные сети, а сети доверия

В байесовских сетях, хотя стрелка, направленная от родительской переменной к дочерней, предпочтительно является причинно-следственной, в общем случае таковой *не является*. Это означает лишь то, что можно смоделировать распределение вероятностей дочерней переменной с учетом состояний ее родителя (родителей), и то, что можно применить правило Байеса и найти обратную вероятность — распределение вероятностей родителя с учетом дочернего элемента. Как правило, это более сложное направление, т. к. оно менее интуитивно и труднее поддается наблюдению. Можно рассуждать об этом в том смысле, что вычислить распределение вероятностей характерных черт ребенка, учитывая, что мы знаем черты родителей P(ребенок | мать, отец) еще до рождения ребенка, будет легче, чем сделать вывод о чертах родителей, зная черты ребенка P(отец | ребенок) и P(мать | ребенок). В этом примере байесовская сеть (рис. 9.12) имеет три узла — мать, отец и ребенок, при этом одно ребро направлено от матери к ребенку, а другое — от отца к ребенку.

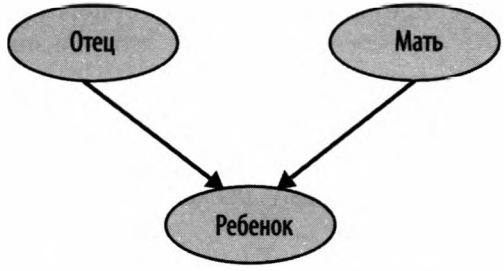


Рис. 9.12. Переменная "ребенок" служит коллайдером в этой байесовской сети

Между матерью и отцом ребро отсутствует, поскольку нет причин для того, чтобы их черты были связаны. Знание черт матери не дает никакой информации о чертах отца, но знание черт матери и ребенка позволяет нам узнать немного больше о чертах отца, или о распределении P(отец | мать, ребенок). Это означает, что черты матери и отца, изначально независимые, становятся условно зависимыми, если знать черты ребенка. Таким образом, байесовская сеть моделирует зависимости между переменными в виде графовой структуры, предоставляя карту того, как переменные *предположительно* связаны друг с другом — их условные зависимости и независимости. Именно поэтому байесовские сети также называются *сетями доверия*.

Что нужно знать о байесовских сетях

Важно запомнить следующее о байесовских сетях.

- Байесовские сети не имеют причинно-следственного направления и ограничены в ответах на причинные вопросы, или вопросы "почему", например "Что стало

причиной определенной болезни?". Однако вскоре мы узнаем, что байесовскую сеть можно использовать для причинного обоснования и для предсказания последствий вмешательства. Независимо от того, как используется сеть, ее обновление, или распространение *доверия*, всегда происходит одинаково.

- Если в некоторых переменных отсутствуют данные, байесовские сети могут справиться с этим, т. к. разработаны для эффективного распространения информации от переменных с бóльшим количеством информации о них к переменным с меньшим количеством информации.

Цепи, вилки и коллайдеры

Конструктивными элементами байесовских сетей (с тремя и более узлами) являются три типа *сочленений*: цепь, вилка, коллайдер, как показано на рис. 9.13.

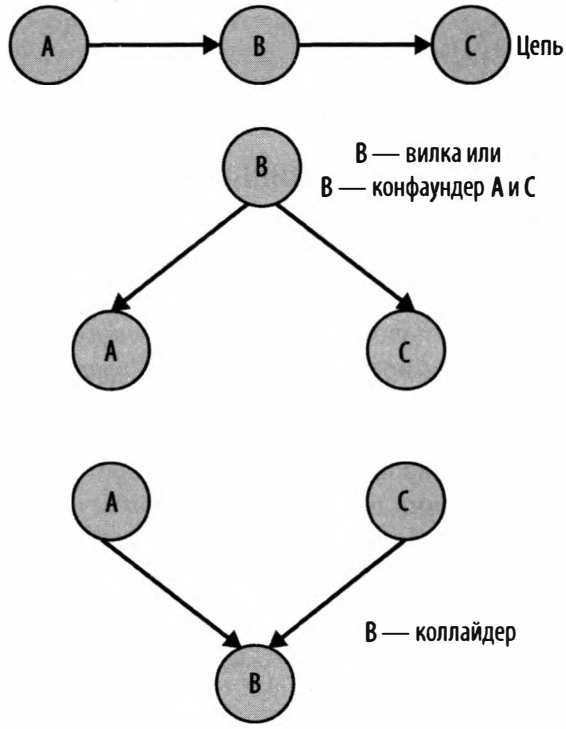


Рис. 9.13. Три типа сочленений в байесовской сети

Цепь: $A \rightarrow B \rightarrow C$.

В этой цепи B является посредником. При известном значении B информация об A не увеличивает и не уменьшает доверие к C. Таким образом, A и C условно независимы, учитывая, что нам известно значение посредника B. Условная независимость позволяет и пользователю, и машине, применяющей байесовскую сеть, сосредоточиться только на значимой информации.

Вилка: $B \rightarrow A$ и $B \rightarrow C$.

Здесь *B* является общим родителем, или *конфаундером*, *A* и *C*. Данные покажут, что *A* и *C* статистически коррелируют, хотя между ними нет причинно-следственной связи. Эту фальшивую корреляцию можно выявить, поставив в зависимость от *конфаундера B*.

Коллайдер: $A \rightarrow B$ и $C \rightarrow B$.

Коллайдеры отличаются от цепей или вилок, если обусловлена переменная в середине. Мы видели это на примере родителей, указывающих на ребенка. Если *A* и *C* изначально независимы, то условие *B* делает их зависимыми! Такая неожиданная и некаузальная передача информации является одной из характеристик обусловленности байесовских сетей — обусловленность коллайдера случайным образом устанавливает зависимость между его родителями.

Еще один момент, заслуживающий внимания, — это ситуация, когда результат и медиатор спутаны. В этом случае зависимость от медиатора будет отличаться от его постоянного значения.

Как создать байесовскую сеть для всех переменных из набора данных?

Графовую структуру байесовской сети можно либо определить вручную, либо выучить из данных с помощью алгоритмов. Алгоритмы байесовских сетей достаточно развиты, также есть коммерческие алгоритмы. После того как структура сети определена, с поступлением в сеть новой информации о некоторой переменной можно легко обновить условные вероятности в каждом узле, следуя диаграмме и распространяя информацию по сети, обновляя доверие к каждой переменной. Изобретатель байесовских сетей Джуда Перл (**https://oreil.ly/J7nSq**) уподобляет этот процесс процессу обновления живой органической ткани и биологической сети нейронов, когда при возбуждении одного нейрона реагирует вся сеть, распространяя информацию от одного нейрона к его соседям.

Наконец, нейросети можно рассматривать как байесовские сети.

Модели, обучающиеся паттернам на основе данных

Важно отметить, что байесовские сети и любые другие модели, изучающие совместные распределения вероятностей признаков данных, включая генеративные и детерминированные модели, рассмотренные в предыдущих главах, только выявляют паттерны в данных и в первую очередь изучают взаимосвязи, а не *причины* этих паттернов. Для того чтобы агент ИИ был действительно разумным и рассуждал как человек, он должен задавать вопросы *"как?"*, *"почему?"*, *"что, если..?"* относительно всего, что он видит и что делает, и искать ответы, как это делают люди в самом раннем возрасте. Фактически у людей возраст "почемучек" наступает в самом начале их развития — это возраст, когда дети сводят с ума своих родителей, спраши-

вая "почему?" обо всем на свете. Агент ИИ должен иметь *каузальную модель*. Эта концепция крайне важна в контексте общего ИИ. Мы рассмотрим ее в следующем разделе и еще раз в *главе 11*, посвященной вероятности.

Графические диаграммы вероятностного каузального моделирования

С первых шагов в статистике мы только и слышали: *корреляция — это не каузальность*. Затем мы то и дело говорили о данных, массивах данных, корреляциях в данных. С этим все понятно — мы уловили суть, но как насчет причинно-следственной связи, т. е. каузации? Что это такое? Как оценивать ее количественно? А мы точно знаем, что означает *"почему"*? Мы воспринимаем причину и следствие на интуитивном уровне, даже в восьмимесячном возрасте. Я вообще считаю, что в мире причин мы действуем на более естественном и интуитивном уровне, чем в мире ассоциаций. *Почему* же тогда (видите?) наши машины, которые, как мы полагаем, в какой-то момент смогут рассуждать так же, как мы, функционируют только на уровне взаимосвязей и регрессий? Именно такую точку зрения в области ИИ обосновывает математик и философ Джуда Перл в своей замечательной книге "Думай „почему?“. Причина и следствие как ключ к мышлению"[4] (2020). Моя любимая цитата из этой книги — "беспричинная корреляция нарушает наш здравый смысл". Идея заключается в том, что необходимо сформулировать и количественно определить, какие корреляции обусловлены причинно-следственными связями, а какие — другими факторами.

Перл строит свои математические каузальные модели с помощью диаграмм (графов), похожих на байесовские сетевые графы, но наделенных схемой вероятностного обоснования на основе *исчисления do*, или вычисления вероятностей *по оператору выполнения do*, в отличие от вычисления вероятностей по *оператору наблюдения observe*, что хорошо знакомо из некаузальных статистических моделей. Основная мысль заключается в следующем.

Наблюдать — это не то же самое, что выполнять. В математической нотации Prob(количество пассажиров автобуса | цветовая кодировка маршрутов) — это не то же самое, что Prob(количество пассажиров автобуса | *do* цветовая кодировка маршрутов)[5].

На основе данных можно сделать первый вывод. Посмотрим на количество пассажиров с учетом цветовой кодировки автобусных маршрутов в определенном городе. Эта вероятность не говорит нам о том, *как* цветовая кодировка маршрутов влияет на количество пассажиров. Вторая вероятность *с оператором do* — совершенно другая, и данные не дают сами ответ без причинно-следственной диаграммы. Раз-

[4] Pearl J., Mackenzie D. The Book of Why: The New Science of Cause and Effect. — Basic Books, 2020. — 432 p. — *Прим. ред.*

[5] Prob — это вероятность (от англ. *probability*). — *Прим. ред.*

ница в том, что с оператором *do* мы намеренно изменяем цветовую кодировку маршрутов и хотим оценить влияние этого изменения на количество пассажиров. Если после такого целенаправленного *выполнения* количество пассажиров увеличивается, и если мы нарисовали правильный граф, включая переменные и их взаимосвязь друг с другом, то можно утверждать, что использование автобусных маршрутов с цветовой кодировкой *вызвало* увеличение количества пассажиров. Если мы не *наблюдаем*, а *выполняем*, то мы вручную блокируем все вероятные естественные причины появления цветовой кодировки маршрутов (например, смена руководства или сезона), способных повлиять на количество пассажиров. Нам не удастся заблокировать все эти причины, если мы будем просто наблюдать за данными. Более того, используя оператор *do*, мы целенаправленно, вручную *задаем* цветовой кодировке *значение* автобусных маршрутов (в отличие от пронумерованных и т. д.).

По правде говоря, пример с автобусными маршрутами не выдуман. В настоящее время я сотрудничаю с департаментом общественного транспорта в Харрисонбурге, штат Вирджиния, решая задачи увеличения количества пассажиров, повышения эффективности, оптимизации работы в условиях ограниченных ресурсов и резкого сокращения населения города в период университетских каникул. В 2019 году транспортный отдел *целенаправленно* сменил номерную кодировку автобусных маршрутов на цветовую и в то же время намеренно поменял гибкое расписание, настроенное под занятия в университете, на фиксированное. Это дало рост пассажиропотока на 18%. Не сомневайтесь — мои студенты, участвующие в работе над проектом этим летом (2022 год), скоро будут рисовать такие диаграммы причинно-следственных связей и писать такие вероятности[6]:

$$P(N_{\text{п}} \mid \text{do цветовая\_кодировка\_маршрута, do фиксированные\_расписания}).$$

Машина, способная распознать паттерн и действовать в соответствии с ним подобно тому, как ящерица наблюдает за летающим вокруг нее жуком, изучая закономерности его полета, а затем ловит и съедает его, — такая машина обладает совсем другим уровнем *интеллекта*, чем машина, способная рассуждать на двух более высоких уровнях, чем простое выявление паттернов:

1. *Если я намеренно выполню это действие, что произойдет с [вставить здесь переменную]?*
2. *Если бы я не выполнил этого действия, произошло бы [переменная приняла определенное значение] все равно?* Если бы Харрисонбург не перешел на цветовую кодировку маршрутов и фиксированные расписания, увеличилось бы количество пассажиров? А если бы изменилась только одна из этих переменных, а не обе?

Одни только данные не могут ответить на эти вопросы. По сути, тщательно выстроенные причинно-следственные диаграммы позволяют различать моменты, когда для ответа на эти вопросы можно использовать одни только данные, и когда на них ответить нельзя *независимо от количества собранных данных*. Пока в машинах не появятся графы, отображающие причинно-следственное обоснование, они оста-

[6] Здесь $N_{\text{п}}$ — количество пассажиров. — *Прим. ред.*

нутся на том же уровне интеллекта, что и ящерицы. Удивительно, но люди выполняют все эти вычисления мгновенно, хотя много раз приходят к неверным выводам и десятилетиями спорят друг с другом о причинах и следствиях. По-прежнему для решения вопросов требуются математика и графы. По сути, граф направляет и подсказывает, какие данные нужно искать и собирать, какие переменные обусловливать и к каким переменным применить оператор *do*. Такое целенаправленное конструирование и обоснование сильно отличаются от культуры сбора больших объемов данных или бесцельного обусловливания любых переменных без разбора.

Теперь мы это знаем, так что можно рисовать диаграммы и создавать модели, которые помогут нам в решении всевозможных вопросов, связанных с обусловленностью. Я выздоровела благодаря врачу или потому, что прошло время и жизнь успокоилась? Все равно нам придется собирать и систематизировать данные, зато теперь этот процесс будет целенаправленным и управляемым.

Машина, обладающая такими способами рассуждения — моделью причинно-следственной диаграммы с перечнем (очень коротким) допустимых операций — сможет отвечать на запросы на всех трех уровнях причинности:

- Коррелируют ли между собой переменные *A* и *B*? Коррелируют ли между собой количество пассажиров и разметка автобусных маршрутов?
- Как изменится переменная *B*, если переменной *A* задать конкретное значение? Увеличится ли количество пассажиров, если намеренно разметить маршруты автобусов цветом?
- Изменится ли переменная *B*, если не задавать определенного значения переменной *A*? Увеличится ли количество пассажиров без перехода на цветовую кодировку автобусных маршрутов?

Нам еще предстоит научиться работать с вероятностными выражениями, в которых используется оператор *do*. Мы выяснили, что наблюдать — это не то же самое, что выполнять; наблюдать — значит *наблюдать* в данных, а выполнять — значит целенаправленно проводить эксперимент для оценки причинно-следственной обусловленности переменных. Это более затратно, чем простой подсчет пропорций, наблюдаемых в данных. Перл установил три правила для операций с вероятностными выражениями, в которых используется оператор *do*. С их помощью можно перейти от выражений с оператором *выполнения* к выражениям с оператором *наблюдения*, где ответы можно получить из данных. Эти правила ценны тем, что позволяют количественно оценить причинно-следственную обусловленность, *наблюдая* ее, обойдясь без *выполнения*. Мы обсудим это в *главе 11*, посвященной вероятности.

Краткая история теории графов

Нельзя оставить эту главу без экскурса в теорию графов и текущее состояние этой области. Она зиждется на простейших основаниях, но при этом прекрасна, она

вдохновляет, в ней применяются весьма перспективные приложения, которые заставили меня пересмотреть весь свой математический путь и срочно попытаться перепрофилироваться.

Словарь теории графов включает в себя графы, узлы, ребра, степени, связность, деревья, остовные деревья, контуры, фундаментальные контуры, векторное пространство графа, ранг и нульмерность (как в линейной алгебре), двойственность, путь, блуждание, линию Эйлера, гамильтонов контур, разрез, сетевой поток, обход, раскраску, нумерацию, ссылки, уязвимость.

Хронология развития теории графов довольно поучительна — ее корни уходят в транспортные системы, карты и географию, электрические цепи и молекулярные структуры в химии.

- В 1736 году Эйлер опубликовал первую работу по теории графов, решив задачу кенигсбергских мостов (**https://oreil.ly/FHXyC**). Затем более ста лет в этой области ничего не происходило.
- В 1847 году, работая над электрическими сетями, Кирхгоф разработал теорию деревьев.
- Вскоре после этого, в 1850-х годах, Кэли открыл деревья в попытке пронумеровать изомеры насыщенных углеводородов C_nH_{2n+2}. Артур Кэли (1821–1895) — один из основателей теории графов. Его имя неразделимо связано с графовыми данными. Совсем недавно сеть CayleyNet (**https://oreil.ly/uKQag**) использовала сложные рациональные функции, называемые полиномами Кэли, в спектральном методе глубокого обучения на графовых данных.
- В то же самое время, в 1850 году, сэр Уильям Гамильтон изобрел игру, ставшую основой *гамильтоновых контуров*, и продал ее в Дублине. Смысл в том, что имеется правильный многогранник из дерева, имеющий 12 граней и 20 углов, причем каждая грань — это правильный пятиугольник, а 3 ребра пересекаются в каждом углу. В 20 углах написаны названия 20 городов: Лондон, Рим, Нью-Йорк, Мумбаи, Дели, Париж и т. д. Нужно найти маршрут по ребрам многогранника, проходящий через каждый из 20 городов ровно один раз (гамильтонов контур). Решение этой конкретной задачи особой сложности не представляет, но до сих пор не было необходимого и достаточного условия для существования такого маршрута в произвольном графе.
- В этот же период времени на лекции Мебиуса (Möbius, 1840-е годы), в письме де Моргана (De Morgan, 1850-е годы) и в статье Кэли в первом томе Трудов Королевского географического общества (1879) появилась самая известная проблема теории графов (решена в 1970 году) — *теорема о четырех красках*. С тех пор она занимает умы многих математиков и приводит к множеству интересных открытий. Она гласит, что четырех красок достаточно для раскраски любой карты на плоскости так, чтобы страны с общими границами имели разные цвета. Интересно то, что если дать себе больше пространства, выйдя за пределы плоской

поверхности, например на поверхность сферы, то у нас действительно появятся решения этой гипотезы.

- К сожалению, на протяжении последующих 70 лет ничего не происходило вплоть до 1920-х годов, когда Кёниг (König) написал первую книгу на эту тему и в дальнейшем опубликовал ее в 1936 году.
- Ситуация изменилась с появлением компьютеров и их растущей способностью исследовать большие проблемы комбинаторного характера. Это подстегнуло интенсивную деятельность как в чистой, так и в прикладной теории графов. К настоящему времени этой теме посвящены тысячи статей и десятки книг, значительный вклад внесли такие авторы, как Клод Берже (Claude Berge), Ойстейн Оре (Oystein Ore), Пол Эрдеш (Paul Erdös), Уильям Татте (William Tutte) и Фрэнк Харари (Frank Harary).

Основные положения теории графов

Рассмотрим основные тематики из теории графов и постараемся взглянуть на них с высоты птичьего полета, не погружаясь в детали.

Остовные деревья и кратчайшие остовные деревья

Они имеют большое значение и используются в протоколах сетевой маршрутизации, алгоритмах кратчайшего пути, алгоритмах поиска. Остовное дерево графа — это подграф, который представляет собой дерево (любые две вершины могут быть соединены одним-единственным путем) и включает в себя все вершины графа. Другими словами, остовные деревья удерживают вершины графа вместе. У одного и того же графа может быть много остовных деревьев.

Наборы разрезов и разрезанные вершины

Любой связный граф можно разбить на части — разъединить, прорезав достаточное количество ребер или, иногда, удалив достаточное количество вершин. Если нам удастся найти такие наборы разрезов в заданном графе, например в коммуникационной, электрической, транспортной или других сетях, то можно разрезать все связи между его разъединенными частями. Обычно нас интересуют наименьшие или минимальные наборы разрезов, выполняющие задачу разъединения графа путем удаления наименьшего количества его ребер или вершин. Это позволяет выявить самые слабые связи в сети. В отличие от остовных деревьев, наборы разрезов разделяют вершины, а не удерживают их вместе. Таким образом, мы справедливо ожидаем тесной связи между остовными деревьями и наборами разрезов. Более того, если граф представляет собой сеть с неким источником (например, жидкостью, трафиком, электричеством или информацией) и поглотителем, где каждое ребро пропускает через себя только определенное количество потока, то существует тес-

ная связь между максимальным потоком, способным двигаться от источника к поглотителю, и разрезом через ребра графа, отсоединяющим источник от поглотителя, с минимальной общей пропускной способностью разрезанных ребер. Здесь работает *теорема о максимальном потоке и минимальном разрезе*, которая гласит, что в гидродинамической сетке максимальное количество потока, проходящего от источника к поглотителю, равно суммарному весу ребер в минимальном разрезе. В математике, когда задача максимизации (максимального потока) нетривиально эквивалентна задаче минимизации (минимального разреза) (например, не только изменяя знак целевой функции), это свидетельствует о двойственности. Действительно, теорема о максимальном потоке и минимальном разрезе для графов — это частный случай теоремы двойственности из линейной оптимизации.

Планарность

Геометрическое представление графа — это планарное или трехмерное представление? То есть возможно ли нарисовать вершины графа и соединить его ребра в одной плоскости так, чтобы они не пересекались друг с другом? Это представляет интерес для таких технологических приложений, как автоматическое подключение комплексных систем, печатных плат, сверхбольших интегральных микросхем. В случае непланарных графов важно знать их толщину и количество пересечений между ребрами. Эквивалентным условием для *планарного графа* является существование *двойственного графа*, где связь между графом и его двойником становится понятной в контексте векторного пространства графа. Здесь линейная алгебра и графы выступают вместе, когда на вопросы о геометрических фигурах отвечают алгебраические и комбинаторные представления, и наоборот. Что касается вопроса о планарности, то нужно рассматривать лишь простые неразделимые графы, все вершины которых имеют три степени или больше. Более того, непланарным считается любой граф, у которого количество ребер в три раза превышает количество вершин минус шесть. По сей день в этой области исследований существует множество нерешенных проблем.

Графы как векторные пространства

Важно понимать граф одновременно как геометрический и алгебраический объект, учитывая соответствие между этими двумя представлениями.

Каждому графу соответствует векторное пространство размерности e над полем целых чисел по модулю 2, где e — количество ребер графа. Так, если граф имеет только три ребра: ребро$_1$, ребро$_2$, ребро$_3$, — то ему соответствует трехмерное векторное пространство, содержащее векторы (0, 0, 0), (1, 0, 0), (0, 1, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), (0, 0, 1), (1, 1, 1). Здесь (0, 0, 0) соответствует нулевому подграфу, в котором не содержится ни одно из трех ребер; (1, 1, 1) соответствует полному графу, содержащему все три ребра; (0, 1, 1) соответствует подграфу, в котором содержатся только ребро$_2$ и ребро$_3$, и т. д.

Поле целых чисел по модулю 2

Поле целых чисел по модулю 2 содержит только два элемента: 0 и 1, а операции + и × выполняются по модулю 2. Фактически они эквивалентны логическим операциям AND (И) и XOR (исключающее ИЛИ) в булевой логике. Векторное пространство должно быть задано над полем и быть замкнутым при умножении его векторов на *скаляры* того же поля. В этом случае скалярами считаются только 0 и 1, а умножение происходит по модулю 2. Таким образом, графы являются прекрасным примером векторных пространств над конечными полями, которые отличаются от обычных вещественных или комплéксных чисел. Размерность векторного пространства графа равна числу ребер e графа, а общее число векторов в этом векторном пространстве будет 2^e. Мы видим, что теорию графов можно сразу же применить к коммутационным схемам (с переключателями "вкл." и "выкл."), цифровым системам и сигналам, т. к. все они работают в поле целых чисел по модулю 2.

Исходя из этого простого соответствия и опираясь на всю область линейной алгебры, будет естественным попытаться понять наборы разрезов, цепи, фундаментальные контуры, остовные деревья и другие важные подструктуры графов, а также отношения между ними в контексте векторных подпространств, базиса, пересечений, ортогональности и размерности этих подпространств.

Реализуемость

Мы уже использовали матрицу смежности и матрицу инцидентности в качестве матричных представлений, полностью описывающих граф. Другие матрицы, такие как матрица цепей, матрица набора разрезов и матрица путей, описывают важные признаки графа. После этого, разумеется, нужно изучить, как все эти матрицы соотносятся и взаимодействуют друг с другом.

Еще одна довольно важная тема — *реализуемость*: какие условия должна выполнять данная матрица, чтобы стать циркулянтной матрицей графа?

Раскраска и сопоставление

Во многих ситуациях нас интересует присвоение меток, или цветов, вершинам и ребрам графа или даже областям планарного графа. Известная задача раскраски графа заключается в том, чтобы цвета, которыми мы хотим пометить каждую вершину, были бы таковыми, что никакие соседние вершины не должны иметь одинакового цвета; более того, нужно выполнить это, используя минимальное количество цветов. Наименьшее количество цветов, необходимое для раскраски графа, называется его *хроматическим числом*. С раскраской связаны такие темы, как разбиение вершин, покрытие и хроматический многочлен.

Сопоставление — это набор ребер, в котором нет двух смежных. Максимальное соответствие — максимальный набор ребер, в котором нет двух смежных. Сопос-

тавление в общем и в двудольном графе имеет множество применений, например, подбор минимального набора классов для удовлетворения требований к выпускникам или подбор рабочих мест в соответствии с предпочтениями сотрудников (в итоге это оказывается задачей о *максимальном потоке и минимальном разрезе*). Для поиска идеальных сопоставлений на больших двудольных графах можно воспользоваться алгоритмами на основе случайного блуждания.

Перечисление

В 1857 году Кэли в процессе подсчета количества изомеров насыщенного углеводорода C_nH_{2n+2} пришел к подсчету количества разных деревьев с n узлами, что послужило его вкладом в теорию графов. Существует множество типов графов, подлежащих *перечислению*, и многие из них стали темами отдельных научных работ. В качестве примеров можно привести перечисление всех корневых деревьев, простых графов, простых диграфов и прочих обладающих специфическими свойствами графов. Перечисление представляет собой обширную область в теории графов. К важнейшим методам перечисления относится *теорема перечисления Полиа*, где требуется найти подходящую *группу перестановок*, а затем получить индекс цикла, что представляется нетривиальным.

Алгоритмы и вычислительные аспекты графов

Алгоритмы и компьютерные разработки представляют огромную ценность для всех, кто работает с моделированием графов. Для традиционных задач теории графов предусмотрены следующие алгоритмы:

- выяснение разделимости графа;
- выяснение связности графа;
- выяснение компонент графа.
- поиск остовных деревьев графа;
- поиск набора фундаментальных контуров;
- поиск набора разрезов;
- поиск кратчайшего пути от конкретной вершины к другой;
- проверка на планарность графа;
- построение графа с особыми свойствами.

Сегодня графовые нейросети комплектуются собственными пакетами с открытым исходным кодом. Как и всегда, для того чтобы алгоритм имел практическую пользу, он должен быть эффективным. Время его работы не должно увеличиваться факториально или даже экспоненциально с ростом числа узлов графа. Оно должно быть полиномиальным, пропорциональным n^k, где k — предпочтительно малое число.

Всем желающим заняться моделированием графов будет полезно ознакомиться как с теорией, так и с вычислительными аспектами графов.

Итоги и перспективы

В этой главе мы кратко рассмотрели различные аспекты графического моделирования, уделив особое внимание примерам, приложениям, а также развитию интуиции. Для читателей, желающих погрузиться глубже, в главе имеется множество ссылок. Главное — не заблудиться бесцельно в дебрях (порой довольно густых) без понимания всей картины, а также текущего состояния области в контексте ИИ.

Мы также познакомились со случайными блужданиями по графам, байесовскими сетями и вероятностными каузальными моделями, что еще больше повернуло наше мышление в сторону вероятностного — главной темы *главы 11*. Я все время старалась ввести в нарратив всевозможные варианты использования вероятности в ИИ, прежде чем мы перейдем к математической главе о вероятности (*см. главу 11*).

В конце главы представлю выдержку из весьма любопытной статьи "Реляционное индуктивное смещение, глубокое обучение и графовые сети" (Батталья П. и др., 2018; **https://oreil.ly/kjZ76**), где приводится аргумент в пользу того, почему сообщество глубокого обучения должно принять графовые сети:

> "В статье рассматривается новый структурный элемент инструментария искусственного интеллекта с сильным реляционным индуктивным смещением — графовая сеть, способная обобщать и расширять различные подходы к оперирующим графами нейросетям и предоставляющая простой интерфейс для операций с структурированными знаниями и формирования структурированного поведения. В работе обсуждается, как графовые сети могут поддерживать релятивное суждение и комбинаторное обобщение, что закладывает основу для более сложных, легко интерпретируемых и гибких паттернов суждения".

ГЛАВА 10
Исследование операций

Большинство ученых обязаны своим величием не столько умению решать проблемы, сколько мудрости в их выборе.
— Е. Брайт Уилсон (1908–1992), американский химик

В этой главе мы рассмотрим интеграцию искусственного интеллекта в область исследования операций, взяв лучшее из этих двух областей для более эффективного и обоснованного принятия решений. Хотя такое вступление звучит как реклама, но именно этим занимается исследование операций. Эта область может значительно продвинуться вперед благодаря достижениям в области машинного обучения.

Исследование операций относится к самым занимательным и стимулирующим областям прикладной математики. Это наука о балансировании различных потребностей и имеющихся ресурсов самыми эффективными с точки зрения времени и затрат способами. Многие задачи исследования операций сводятся к поиску оптимальной точки, чаши Грааля, в которой все будет функционировать гладко и эффективно, без резервного копирования и без отходов, с беспрерывным своевременным обслуживанием, сбалансированными затратами и высокими доходами каждого участника. Многим приложениям так и не удается найти священный Грааль, хотя многие методы исследования операций позволяют приблизиться к нему, по крайней мере, в случае упрощенных моделей сложной реальности. Математическая оптимизация с ограничениями проникает в каждую отрасль, каждую сеть, каждый аспект нашей жизни. При правильном подходе мы наслаждаемся ее преимуществами, при неправильном — страдаем от ее последствий: мировая и региональные экономики до сих пор переживают последствия COVID-19, перебоев в цепочке поставок.

Прежде чем мы рассмотрим, как машинное обучение начинает пробивать себе дорогу в исследование операций, выделим несколько идей, которые должен усвоить каждый, кто захочет работать в этой области. Поскольку этой замечательной теме в книге посвящена всего одна глава, кратко рассмотрим ее основные аспекты.

Теорема об отсутствии бесплатного обеда.

Заставляет нас переключиться на разработку и анализ методов, лучше всего подходящих для конкретного случая, вместо поиска наиболее общих и широко применимых методов, как это свойственно многим математикам. По сути, она практически требует от подобных математиков *остыть* и довольствоваться специализированными решениями для конкретных типов задач.

Оценка сложности задач и асимптотический анализ алгоритмов.

Асимптотический анализ говорит нам о том, что даже суперсовременный гениальный алгоритм окажется бесполезным, если его вычислительные требования резко возрастают с увеличением масштаба задачи. Решения в области исследования операций необходимо масштабировать на большие сценарии с множеством переменных. Оценка сложности, в свою очередь, рассматривает уровень сложности самих задач, а не алгоритмов их решения. Комбинаторные задачи $O(n!)$ считаются крайне неприемлемыми вследствие того, что при очень больших n значение $n!$ больше k^n, но экспоненциальная сложность k^n уже будет крайне неприемлемой!

Важные темы и приложения в исследовании операций.

Их можно найти в любой хорошей книге по исследованию операций. Необходимо всегда держать одну из них под рукой. Переход от конкретного приложения и бизнес-задач к математической формулировке — это навык, без которого невозможно добиться успеха в этой области.

Разнообразные методы и алгоритмы оптимизации.

Это движущая сила решений и программных пакетов для исследования операций.

Программные пакеты.

Они доступны повсеместно, и по причине ограниченного объема книги мы не будем останавливаться в этой главе на алгоритмах и вычислениях.

Резюмируем исследование операций шестью словами — это *математическая формулировка*, *оптимизация*, *алгоритмы*, *программа*, *решения.*

По ходу чтения этой главы можно рассматривать концепции в контексте того, как управляют своими операциями компании, с которыми мы взаимодействуем в повседневной жизни. Рассмотрим, например, логистику Amazon. Amazon — крупнейшая в мире компания в сфере электронной коммерции. В 2022 году ее доля на рынке электронной коммерции США составила 45%. Ежедневно она продает и доставляет миллионы единиц товаров, а каждую секунду совершает продажи на сумму около 5000 долларов. Как Amazon удается добиваться таких результатов? Как компания управляет своими запасами, складами, транспортом, своей суперэффективной системой доставки? Как Amazon формулирует свои подзадачи и объединяет их в одну большую успешную операцию? То же самое можно сказать и о транспортной логистике, например, в компании Uber. Ежедневно Uber осуществляет до 15 млн совместных поездок по всему миру — компания подбирает свободных водителей для ближайших пассажиров, определяет маршруты и время подачи и поездки, устанавливает тарифы на поездки, прогнозирует доходы водителей, а также соотношение спроса и предложения, проводит многочисленные аналитические исследования.

Для исследования операций типичны сложные и сильно взаимосвязанные задачи оптимизации, позволяющие таким массивным системам работать относительно гладко. Более того, многие из этих задач считаются NP-трудными (в терминах вы-

числительной сложности это означает, что они имеют недетерминированный полиномиальный уровень сложности; в переводе на нормальный язык — *очень дорогое вычисление*). Добавим к этому их стохастическую природу и получим ждущие своих решений занимательные математические задачи.

В целом математические методы и алгоритмы исследования операций ежегодно экономят миру миллиарды долларов. Опрос 500 крупнейших компаний США показал, что 85% из них используют линейное программирование (другое название линейной оптимизации, которая занимает значительное место в исследовании операций, и именно поэтому в данной главе много времени уделено симплекс-методу и двойственности). Сейчас самое время заняться этим направлением, воспользовавшись инструментарием ИИ. Награда ждет на многих уровнях: интеллектуальном, финансовом и в виде значимого вклада в общее благо человечества. Так что темы, рассматриваемые в данной главе, ни в коем случае не умаляют значение остальных не менее важных тем в этой области.

Для более основательного погружения в исследование операций (разумеется, после прочтения этой главы) лучше всего обратиться к опыту успешных профессионалов, среди которых:

- проекты победителей и финалистов конкурса Франца Эдельмана "За достижения в области исследования операций и науки управления" (**https://oreil.ly/fuewK**);
- книга Фредерика Хиллера и Джеральда Либермана "Введение в исследование операций"[1].

Отсутствие бесплатного обеда

Теорема об отсутствии бесплатного обеда для оптимизации гласит, что не существует одного конкретного алгоритма оптимизации, который бы работал лучше всего для каждой задачи. Все алгоритмы, выполняющие поиск оптимизатора целевой функции (функции стоимости, функции потерь, функции полезности, функции правдоподобия), имеют схожую производительность при усреднении по всем возможным целевым функциям. Поэтому если на каком-то классе целевых функций какой-то алгоритм работает лучше другого, значит, есть и другие целевые функции, на которых другой алгоритм работает лучше. Не существует одного совершенного алгоритма, который хорошо справлялся бы со всеми типами задач. Поэтому выбор алгоритма определяется конкретной задачей (или областью). В зависимости от области применения можно найти достаточно информации о том, какими алгоритмами пользуются специалисты на практике, как они обосновывают свой выбор, сравнивают их с другими в задачах как высокой, так и разумной размерности, и как постоянно пытаются добиться лучшей производительности, опираясь в большей степени на два критерия: скорость (не требует больших затрат на вычисления) и точность (дает правильные ответы).

[1] Hillier F., Liebertnan G. Introduction to Operations Research. — 11th ed. — McGraw Hill, 2021. — 994 p.

Анализ сложности и нотация O()

Во многих случаях проблема эффективного распределения ограниченных ресурсов при различных ограничениях сводится к разработке эффективных алгоритмов дискретной оптимизации. Линейное программирование, целочисленное программирование, комбинаторная оптимизация, оптимизация в графовых структурах (сетях) — все они взаимосвязаны (иногда это не более чем два разных названия одного и того же) и имеют одну и ту же цель: из *дискретного конечного множества* допустимых вариантов найти оптимизатор — *допустимый набор*. Если допустимый набор изначально не дискретен, то в некоторых случаях его можно свести к дискретному и воспользоваться богатым инструментарием из этой области. И здесь возникает главная проблема — исчерпывающий поиск обычно не поддается решению. Это означает, что если перечислить все доступные варианты на допустимом наборе и оценить целевую функцию на каждом из них, то придется потратить уйму времени, чтобы найти точку (точки), дающую оптимальный ответ. Никто не утверждает, что конечное допустимое — это не значит огромное. Нужны специальные алгоритмы, способные эффективно исключать большие участки пространства поиска. Некоторые алгоритмы позволяют найти точное решение для некоторых задач, в то время как другие находят только приблизительные решения, и остается лишь довольствоваться этим.

Сразу оговорим некоторые различия, т. к. это многих сбивает с толку.

Анализ сложности предназначен для задач, которые мы хотим решить (маршрутизация, задача коммивояжера, задача о рюкзаке и т. д.).

Естественная сложность задачи не зависит от алгоритмов, применяемых для ее решения. По сути, часто сложность говорит нам о том, что для текущих задач можно даже не надеяться на более эффективный алгоритм или на то, что в других случаях нам удастся сделать это лучше. В любом случае анализ сложности задач — это самостоятельная обширная дисциплина, а область исследования операций предоставляет множество *сложных задач*, над которыми стоит поразмышлять. Именно здесь появляются такие термины, как полиномиальная проблема, недетерминированная полиномиальная проблема, недетерминированная полиномиальная полная проблема, недетерминированная полиномиальная проблема, сложная проблема полиномиального времени, дополняющая недетерминированная полиномиальная проблема, дополняющая недетерминированная полиномиальная полная проблема. Эти термины настолько запутаны, что кому-то нужно серьезно пересмотреть их номенклатуру. Мы не будем давать здесь определения каждого из них (в основном потому, что в теории границы между этими классами задач еще не определены), но проведем такое разделение: задачи, которые можно решить за полиномиальное время или меньше, и задачи, для которых нельзя найти точное решение за полиномиальное время независимо от того, какой алгоритм используется, и в этом случае приходится довольствоваться приближенными алгоритмами (например, задача коммивояжера). Важно отметить, что иногда полиномиальное время решения задач может оказаться не совсем замечательным, т. к., например, $O\left(n^{2000}\right)$ не будет столь быстрым.

Асимптотический анализ предназначен для алгоритмов, разработанных для решения этих проблем.

Здесь мы пытаемся оценить количество требуемых алгоритмом операций и количественно оценить его относительно масштаба задачи. В таких случаях используется *нотация большого O*.

Нотация большого O()

Функция $g(n)$ — это $O(f(n))$, если $g(n) \le cf(n)$ для некоторой постоянной c и для всех $n \ge n_0$. Например, $2n+1$ будет $O(n)$, $5n^3 - 7n^2 + 1$ будет $O(n^3)$, $n^2 2^n - 55n^{100}$ будет $O(n^2 2^n)$, а $15n\log n - 5n$ будет $O(n\log(n))$. Следует помнить о постоянной асимптотике в случае $O(1)$, когда количество операций алгоритма не зависит от размера задачи (это замечательно, т. к. означает, что он масштабируется, не беспокоясь о громоздкости задачи).

Для некоторых алгоритмов можно подсчитать точное количество операций: например, чтобы вычислить скалярное произведение (точечное произведение) двух векторов длины n, простой алгоритм использует ровно $2n-1$ умножений и сложений, что делает его $O(n)$. Для умножения двух матриц размера $n \times n$ простой алгоритм вычисления скалярного произведения каждой строки из первой матрицы на каждый столбец из второй матрицы требует ровно $(2n-1)n^2$ операций, так что это будет $O(n^3)$. Обратная матрица также будет $O(n^3)$.

Тому, кто интересуется асимптотическим анализом алгоритмов, быстро становится очевидным, что он несколько сложнее, чем подсчет операций, т. к. иногда приходится выполнять оценки или усреднения по размеру входных данных (значение n), по тому, как считать операции в алгоритме (по каждой строке кода?), причем нельзя не учитывать тот факт, что операции вычисления над большими числами занимают больше времени и памяти, чем операции над меньшими числами. В конечном счете предпочтение отдается алгоритмам, которые выполняются за полиномиальное время или меньше, а не за экспоненциальное время или больше. Продемонстрируем это на простейшем примере.

Полиномиальный алгоритм $O(n^k)$ vs экспоненциальный алгоритм $O(k^n)$

Предположим, что мы работаем на машине, способной выполнять 10^7 операций в секунду (10 млн). Запустим ее на 1000 секунд, что составляет около 16 минут, на двух разных алгоритмах, один из которых экспоненциальный по размеру задачи, скажем $O(2^n)$, а другой — полиномиальный $O(n^3)$. В данном случае размер задачи равен n и обозначает меру размерности входных данных, например количество узлов графа, количество записей матрицы, количество признаков набора данных или количество выборок. Каким будет наибольший размер задачи, с которой каждый алгоритм справится на этой машине за 16 минут?

Для алгоритма с экспоненциальным временем количество требуемых операций составляет максимум (в худшем случае) $c \cdot 2^n = 10^7 \cdot 1000$ при некотором предпочтительно малом c. Таким образом, размер задачи, которую он может решить за 1000 секунд со скоростью 10 млн операций в секунду, составляет $n = 10\log_2 10 - \log_2 c \approx 33$.

Теперь сравним его с алгоритмом полиномиального времени, для которого наихудшим случаем является $cn^3 = 10^7 \cdot 1000$, так что $n = \frac{1}{3\sqrt{c}}\sqrt{10^{10}} \approx 2100$. Это почти на два порядка больше, чем у алгоритма с экспоненциальным временем.

Отсюда следует вывод: при одинаковом аппаратном обеспечении и количестве времени алгоритмы полиномиального времени $O\left(n^k\right)$ позволяют решать гораздо бóльшие задачи, чем алгоритмы экспоненциального времени $O\left(k^n\right)$. Алгоритмы комбинаторного времени $O(n!)$ безнадежны. Более того, нам *всегда* нужно, чтобы k было небольшим. Чем меньше, тем лучше.

Тому, кто привык работать в точной, а не в приближенной или асимптотической области, это рассуждение может показаться неудобным, т. к. иногда в задачах меньшего размера некоторые алгоритмы высшего порядка лучше, чем алгоритмы низшего порядка. Например, если точное количество операций алгоритма $O(n)$ равно $20n - 99$, а алгоритма $O\left(n^2\right)$ равно $n^2 + 1$, то будет верно, что асимптотически (или при достаточно большом n) алгоритм $O(n)$ лучше, чем $O\left(n^2\right)$, и неверно, если n меньше 10, т. к. в этом случае $n^2 + 1 < 20n - 99$. Причем это верно для достаточно маленьких, а не для больших задач.

Далее в этой главе мы обсудим два метода оптимизации — *симплекс-метод* и *метод внутренней точки* для линейной оптимизации (оптимизации, в которой линейны как целевая функция, так и ограничения). Метод внутренней точки — это алгоритм полиномиального времени, а симплекс-метод — экспоненциального времени, поэтому следовало бы ожидать, что все будут использовать более дешевый метод внутренней точки и откажутся от симплекс-метода, но это не так. Симплекс-метод (и двойственный симплекс-метод) по-прежнему широко используется для линейной оптимизации вместо метода внутренней точки, т. к. экспоненциальное время — это наихудший сценарий, а большинство приложений не являются наихудшими. Более того, между алгоритмами обычно возникают компромиссы в плане вычислительных усилий на итерацию, количества необходимых итераций, влияния лучших начальных точек, сходимости алгоритма или необходимости дополнительной помощи ближе к концу, объема вычислений, которые потребует эта дополнительная помощь, а также возможности использования алгоритмом преимуществ параллельной обработки. По этой причине компьютерные пакеты для линейной

оптимизации имеют эффективные реализации как симплекс-метода, так и метода внутренней точки (а также многих других алгоритмов). В конечном итоге мы выбираем то, что лучше всего подходит для нашего конкретного случая.

Оптимизация — сердце исследования операций

Мы вернулись к оптимизации. В машинном обучении оптимизация заключается в минимизации функции потерь для моделей, обучающихся детерминированным функциям, или максимизации функции правдоподобия для моделей, обучающихся вероятностным распределениям. Нам не нужно, чтобы решение точно соответствовало данным, т. к. оно будет плохо обобщаться на неизвестные данные. Поэтому существуют методы регуляризации, ранней остановки и прочие. В машинном обучении модель обучается на имеющихся данных при помощи детерминированной функции или распределения вероятностей, которые являются источником данных (правило или процесс генерирования данных), после чего на основе обученной функции или распределения модель делает выводы. Оптимизация — это только шаг на данном пути, который сводится к минимизации функции потерь с условиями регуляризации или без них. Функции потерь, используемые в машинном обучении, как правило, являются дифференцируемыми и нелинейными, в то время как оптимизация не имеет ограничений. Но в зависимости от задачи можно добавлять ограничения, чтобы *направить* процесс в нужную область.

Методы оптимизации могут либо включать вычисление производных целевой функции $f(\vec{x})$, как, например, довольно популярный в машинном обучении градиентный спуск (стохастический градиентный спуск, Adam и т. д.), либо не включать. Существуют алгоритмы оптимизации, не требующие вычисления производных. Они окажутся весьма полезными, когда объективная функция недифференцируема (например, функции с углами) или формула целевой функции вообще недоступна. Примерами методов оптимизации без производных являются байесовский поиск, алгоритм кукушки, генетические алгоритмы.

Оптимизация, в частности *линейная оптимизация*, занимает центральное место в исследовании операций со времен Второй мировой войны, когда был разработан, например, *симплекс-метод* для решения задач военной логистики и операций. Как обычно, основная цель здесь заключается в минимизации целевой функции (стоимость, расстояние, время и т. д.) при определенных ограничениях (бюджет, сроки, пропускная способность и т. д.):

$$\min_{\text{ограничения}} f(\vec{x}).$$

В типичном курсе по оптимизации исследований операций много времени уделяется линейной оптимизации, целочисленной оптимизации и оптимизации в сетях (графах), т. к. большинство реальных проблем логистики и распределения ресурсов прекрасно укладываются в эти формулировки. Для того чтобы стать успешным исследователем операций, необходимо освоить следующие дисциплины.

Линейная оптимизация.

В этом случае целевая функция и ограничения будут линейными. Здесь фигурируют такие понятия, как симплекс-метод, двойственность, релаксация Лагранжа (**https://oreil.ly/QEZXW**) и анализ чувствительности. В линейных задачах границы нашего мира плоские и состоят из линий, плоскостей и гиперплоскостей. В такой (сверх)многоугольной геометрии, или *многограннике*, часто имеются угловые точки, которые являются кандидатами в оптимизаторы, поэтому существуют систематические способы просеивания этих точек и проверки их на оптимальность (именно этим занимаются симплекс-метод и двойственный симплекс-метод).

Метод внутренней точки.

Применяется для решения масштабных задач линейной оптимизации, недоступных для симплекс-метода. В двух словах, симплекс-метод обходит *границу* пространства поиска (ребра многогранника), проверяет каждый угол на оптимальность, после чего переходит к другому углу на границе. Метод внутренней точки, в свою очередь, проходит *внутри пространства* поиска, добираясь до оптимального угла изнутри, а не с границы.

Целочисленное программирование.

Оптимизация, в которой все элементы оптимизирующего вектора должны быть целыми числами. Иногда они могут быть только нулем или единицей (отправлять грузовик на склад в Огайо или не делать этого). Простейший типичный пример — задача о рюкзаке (задача о ранце, **https://oreil.ly/iT2Wd**). Здесь для решения больших задач целочисленного программирования применяется *метод ветвей и границ*.

Оптимизация в сетях.

Большинство сетевых задач можно переформулировать в задачи линейной оптимизации, где работают симплекс-методы и их специализированные версии, хотя лучше всего, используя структуру сети, воспользоваться таким важным положением теории графов, как теорема о максимальном потоке и минимальном разрезе, чтобы применить самые эффективные алгоритмы. Большинство задач оптимизации в сетях сводится к поиску кратчайшего пути в сети (пути от одного узла к другому с минимальным расстоянием или минимальными затратами), минимального остовного дерева сети (это отлично подходит для оптимизации *проектирования* сетей), максимального потока (от источника к месту назначения или от источника к источнику), минимального потока затрат, многотоварного потока или решению задачи о коммивояжере (поиску циклического маршрута минимальных затрат (или расстояния или веса), который проходит через все узлы сети только один раз (гамильтонов контур)).

Нелинейная оптимизация.

Целевая функция и/или ограничения являются нелинейными. В книге постоянно приводится пример минимизации нелинейных функций потерь для моделей машинного обучения. Во всех случаях они будут нелинейными, и чаще всего здесь

применяются алгоритмы градиентного спуска. В решении небольших задач можно воспользоваться *алгоритмом Ньютона* (вторые производные). В исследовании операций нелинейность целевой функции и/или ограничений может возникать, например, вследствие нефиксированной стоимости доставки товаров из одного места в другое (например, она зависит либо от расстояния, либо от количества товаров) или из-за того, что в потоке через сеть могут содержаться как потери, так и выигрыши. К известным особым видам нелинейной оптимизации относится *квадратичная оптимизация* с линейными ограничениями. Она встречается в таких приложениях, как уравнения сети для электрических цепей и теория упругости, где учитываются смещения, напряжения, деформации, баланс сил в структуре. Мы знаем, как просто найти минимум квадратичной функции $f(x) = sx^2$, где s — положительная константа. Подобная простота прекрасно переходит в более высокие измерения, где целевая функция будет выглядеть как $f(\vec{x}) = \vec{x}^T \mathbf{S} \vec{x}$, где $\mathbf{S}$ — положительная полубесконечная матрица, играющая в высоких измерениях ту же роль, что и положительная константа в одном измерении. Здесь в нашем распоряжении имеется даже *теория двойственности*, которой можно воспользоваться, как и в случае линейной оптимизации. Когда при оптимизации теряется линейность, мы полагаемся на то, что функции являются квадратичными, а ограничения — линейными. Когда это теряется, мы полагаемся на выпуклость функций и/или допустимого набора. Когда теряется выпуклость, нам остается надеяться на то, что наши методы не застрянут в локальных минимумах высокоразмерных ландшафтов и найдут каким-то образом путь к оптимальным решениям.

Динамическое программирование и марковские процессы принятия решений.

Динамическое программирование относится к многоэтапным проектам, когда необходимо принимать решения на каждом этапе, при этом каждое решение порождает непосредственные затраты. Решение на каждом этапе связано с текущим состоянием, а также с правилами перехода в следующее состояние (выбор следующего состояния путем минимизации детерминированной функции или вероятности). Динамическое программирование — это разработка эффективных методов (чаще всего рекурсивных) нахождения *оптимальной последовательности взаимосвязанных решений* для достижения определенной цели. Идея заключается в том, чтобы избежать перечисления всех вариантов для каждого этапа процесса принятия решения с последующим выбором наилучшей комбинации решений. Такой исчерпывающий поиск окажется слишком дорогим для задач с большим количеством этапов принятия решений, каждый из которых имеет множество состояний. Если правила перехода от одного этапа к другому носят вероятностный, а не детерминированный характер, а этапы процесса принятия решений повторяются бесконечно, т. е. проект имеет бесконечное число этапов, то на выходе мы получаем марковский процесс принятия решений (или цепь Маркова). Это процесс, который развивается во времени вероятностным образом. Особое свойство марковского процесса принятия решений заключается в том, что вероятности того, как процесс будет развиваться в будущем, не зависят

от прошлых событий, а только от текущего состояния системы. Цепи Маркова как с дискретным, так и с непрерывным временем моделируют такие важные системы, как системы очередей (**https://oreil.ly/zOLmN**), динамическое управление сигналами светофора в целях минимизации времени ожидания автомобилей, гибкое штатное расписание колл-центров. Здесь важными математическими объектами являются матрицы перехода и решаются задачи определения вероятностей устойчивого состояния. В итоге все сводится к вычислению собственного пространства матрицы перехода.

Стохастические алгоритмы.

Примерами стохастических алгоритмов служат динамическое программирование на базе вероятностного перехода и цепи Маркова, стохастический градиентный спуск, случайные блуждания по графам. Стохастическим считается любой алгоритм, в котором присутствует элемент случайности. Здесь математика переходит на язык вероятностей, ожиданий, стационарных состояний, сходимости и т. д. Другим примером, в котором проявляются стохастические алгоритмы и анализ процессов, является *теория массового обслуживания*, или *теория очередей*, например очередей в приемном покое больницы или на судоремонтной верфи. Она опирается на вероятностные распределения времени прибытия клиентов и времени обслуживания в сервисном центре.

Метаэвристика.

В большинстве задач оптимизации поиск оптимального решения может оказаться непрактичным, поэтому мы (которым все равно нужно принимать решения) прибегаем к *эвристическим методам*, чтобы найти *ответ* (не будем называть его решением), который будет необязательно оптимальным, но вполне пригодным для решения поставленной задачи. Метаэвристика — это общие методы решения, которые обеспечивают руководство по стратегии и общие рамки для разработки эвристических методов, подходящих для определенных семейств проблем. Сложно гарантировать оптимальность ответа, полученного эвристическим методом, зато эвристики ускоряют процесс поиска удовлетворительных решений в случаях, когда оптимальные решения оказываются слишком затратными в плане вычислений или недоступными. Также существует проблема *выполнимости*. Поскольку задачи в исследовании операций почти всегда ограничены, естественным образом возникает вопрос: насколько выполнимы ограничения? То есть является ли допустимый набор непустым? Некоторые задачи исследования операций можно переформулировать в задачи выполнимости.

В процессе решения реальных задач бо́льшая часть работы отделов исследования операций заключается в формулировке конкретных случаев использования и целей так, чтобы уложить их в одну из этих схем оптимизации. Здесь важно распознать особые структуры (например, разреженность задействованных матриц) или подструктуры, которые можно использовать для создания более эффективных алгоритмов. Это принципиально важно в случае сложных и масштабных систем.

Рассуждения об оптимизации

Когда мы сталкиваемся с проблемой оптимизации в математике:

$$\min_{\vec{x} \in \text{некий допустимый набор}} f(\vec{x}),$$

где допустимый набор определен некоторыми ограничениями, которым должен удовлетворять вектор $\vec{x}$ (или он может быть совершенно неограниченным), мы обычно делаем паузу и задаемся таким вопросами:

- Является ли $f(\vec{x})$ линейной функцией?
- Является ли $f(\vec{x})$ выпуклой функцией? Ограничена ли она снизу?
- Является ли минимальное значение конечным или оно стремится к $-\infty$?
- Является ли допустимый набор непустым? То есть существуют ли такие $\vec{x}$, которые действительно выполняют ограничения?
- Является ли допустимый набор выпуклым?
- Имеется ли минимизатор?
- Является ли минимизатор единственным или существуют другие?
- Как найти минимизатор?
- Какое значение имеет минимум?
- Насколько изменится минимизатор и значение минимума, если что-то изменится в ограничениях или в целевой функции?

В зависимости от типа решаемой задачи можно отвечать на эти вопросы независимо друг от друга, т. е. иногда можно ответить только на некоторые из них, но не на другие. Это нормально, т. к. любая информация об оптимизаторе и значении оптимума является ценной.

Рассмотрим распространенные типы задач оптимизации.

Оптимизация: конечные размерности без ограничений

Аналогична оптимизации, которую проходят на лекциях по исчислению, и на оптимизацию, которую проводят при обучении модели машинного обучения, минимизируя функцию потерь. Целевая функция $f(\vec{x})$ является дифференцируемой:

$$\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}).$$

При неограниченной и дифференцируемой оптимизации минимизатор $\vec{x}^*$ выполняет условие $\nabla f(\vec{x}) = 0$. Более того, *гессиан* (матрица вторых производных) положительно полуопределен при $\vec{x}^*$. Обсуждая оптимизацию в машинном обучении, мы остановились на стохастическом градиентном спуске и его вариантах для очень высокоразмерных задач. С задачами меньшей размерности неплохо справляются методы Ньютона (работающие не только с первыми, но и со вторыми производны-

ми). В редких случаях, например при решении таких задач, как функция потерь среднеквадратической ошибки для линейной регрессии, можно получить аналитические решения. Примеры, в которых получаются аналитические решения, как правило, тщательно выстроены (скажем, как все примеры в учебниках по исчислению) и имеют довольно низкую размерность.

Оптимизация: конечные размерности, ограниченные множители Лагранжа

Рассмотрим случай, когда имеется только одно ограничение $g(\vec{x})=b$. Это достаточно хорошо объясняет, что нам нужно. Задача минимизации выглядит следующим образом:

$$\min_{\substack{g(\vec{x})=b \\ x \in \mathbb{R}^d}} f(\vec{x}).$$

Если $f(\vec{x})$ и $g(\vec{x})$ — дифференцируемые функции из $\mathbb{R}^d \to \mathbb{R}$, то можно ввести множители Лагранжа (метод 1797 года) и превратить задачу в *неограниченную*, только в более высоких размерностях (соответствующих новым множителям Лагранжа, введенным в задачу оптимизации). Однако ничто не дается даром. В этом случае мы добавляем к целевой функции кратное ограничение, а затем минимизируем, т. е. ищем точки, в которых градиент равен нулю. Новая целевая функция для задачи без ограничений называется лагранжианом и является функцией как вектора решений $\vec{x}$, так и новой переменной λ, умноженной на ограничение, которое называется *множителем Лагранжа*:

$$\mathscr{L}(\vec{x};\lambda)=f(\vec{x})+\lambda\left(b-g(\vec{x})\right).$$

Если имеется более одного, скажем пять ограничений, то мы вводим множитель Лагранжа для каждого из них, добавляя к задаче оптимизации пять дополнительных измерений, чтобы перевести ее из режима ограничений в режим без ограничений.

Оптимизатор $(\vec{x}^*, \lambda^*)$ задачи без ограничений должен удовлетворять $\nabla\mathscr{L}(\vec{x};\lambda)=0$. Найдем его так же, как и в случае общих задач без ограничений (см. предыдущий пример). Значение $\vec{x}^*$ из $(\vec{x}^*, \lambda^*)$ *будет решением задачи с ограничениями*, которое мы искали изначально. Значит, это точка на гиперповерхности, определяемой ограничением $g(\vec{x}^*)=b$, где значение f будет наименьшим.

Если задача имеет особую структуру, которой можно воспользоваться, например, если f — квадратичная функция, а g — линейное ограничение, или если линейны как f, так и g, то существуют более удобные методы оптимизации с ограничениями как при помощи множителей Лагранжа (которые вводят двойственность), так и без них. К счастью, задачи оптимизации с простыми структурами прекрасно изучены, и не только потому, что они упрощают математику и вычисления, но и потому, что

они постоянно появляются в науке и в реальных приложениях, что несколько подтверждает мою теорию о том, что природа проще, чем кажется математикам. Мы вернемся к множителям Лагранжа для задач с ограничениями в разделе о двойственности, где сосредоточимся исключительно на полностью линейных задачах или квадратичных задачах с линейными ограничениями.

Значение множителей Лагранжа

Важный аспект, который мы должны навсегда запомнить, заключается в том, что множитель Лагранжа λ — это не какой-то бесполезный вспомогательный скаляр, позволяющий превратить задачу с ограничениями в задачу без ограничений. Он имеет значение, весьма ценное для анализа чувствительности, для приложений в области финансов и исследования операций, а также для теории двойственности (все они связаны друг с другом). С точки зрения математики, наблюдая формулу лагранжиана $\mathscr{L}(\vec{x};\lambda)=f(\vec{x})+\lambda\left(b-g(\vec{x})\right)$, можно утверждать, что λ — это скорость изменения лагранжиана в зависимости от b, если бы нам было позволено менять b (значение ограничения в приложениях, где мы заботимся о влиянии наложения или ослабления ограничений). То есть:

$$\frac{\partial\mathscr{L}\left((\vec{x};\lambda,b)\right)}{\partial b}=\frac{\partial f(\vec{x})+\lambda\left(b-g(\vec{x})\right)}{\partial b}=\frac{\partial f(\vec{x})}{\partial b}+\frac{\lambda\left(b-g(\vec{x})\right)}{\partial b}=0+\lambda=\lambda.$$

Более того, оптимальное значение λ^*, соответствующее оптимизатору $\vec{x}^*$, можно интерпретировать как ***предельный эффект*** b на оптимально достижимое значение целевой функции $f(\vec{x}^*)$. Следовательно, если $\lambda^*=2{,}1$, то увеличение b на единицу (сдвиг ограничения на единицу) приведет к увеличению оптимального значения f на 2,1 единицы. Это довольно ценная информация для приложений в области финансов и исследования операций. Попробуем разобраться, почему так происходит. Нужно доказать, что:

$$\frac{df\left(\vec{x}^*(b)\right)}{db}=\lambda^*.$$

Важно отметить, что при оптимизаторе $\vec{x}^*(b)$, который получается при установке градиента лагранжиана на ноль, происходят две вещи: $\nabla f\left(\vec{x}^*(b)\right)=\lambda^*\nabla g\left(\vec{x}^*(b)\right)$ и $g\left(\vec{x}^*(b)\right)=b$. Используя эту информацию и цепное правило для производных (здесь стоит обратиться к учебнику по вычислениям и освоить цепное правило, которое мы постоянно применяем), мы имеем:

$$\frac{df\left(\vec{x}^*(b)\right)}{db}=\nabla f\left(\vec{x}^*(b)\right).\frac{d\vec{x}^*(b)}{db}=\lambda^*\nabla g\left(\vec{x}(b)\right)\frac{d\vec{x}^*(b)}{db}=\lambda^*\frac{dg\left(\vec{x}(b)\right)}{db}=$$

$$=\lambda^*\frac{db}{db}=\lambda^*\times 1=\lambda^*.$$

Другими словами, множитель Лагранжа λ^* — это скорость изменения оптимальных издержек (значения целевой функции) в результате ослабления соответствующего ограничения. В экономике λ^* называется *предельными издержками* с учетом ограничений или *теневой цены*. Далее в этой главе при обсуждении двойственности для обозначения переменных решения двойственной задачи будет использоваться латинская буква *p*.

Оптимизация: бесконечные размерности, вариационное исчисление

Область вариационного исчисления — это область оптимизации, только вместо поиска оптимизирующих *точек* в конечномерных пространствах мы ищем оптимизирующие *функции* в бесконечномерных пространствах.

Конечномерная функция из $\mathbb{R}^d \to \mathbb{R}$ выглядит как $f(\vec{x}) = f(x_1, x_2, ..., x_d)$, а ее градиент (который всегда важен для оптимизации) имеет вид:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{pmatrix}.$$

Направленная производная оценивает изменение *f* или его *вариацию* в направлении некоторого вектора $\vec{n}$:

$$\frac{\partial f(\vec{x})}{\partial \vec{n}} = \lim_{h \to 0} \frac{f(\vec{x} + h\vec{n}) - f(\vec{x})}{h} = \nabla f . \vec{n}.$$

Если допустить зависимость *x* от времени, то мы получим:

$$f'(\vec{x}(t)) = \frac{df(\vec{x}(t))}{dt} = \nabla f . \frac{d\vec{x}(t)}{dt} = \nabla f . \vec{x}'(t).$$

Это выражение пригодится при вычислении *вариаций функционалов бесконечной размерности*. Функционал — это функция, у которой на входе — функция, а на выходе — вещественное число. Таким образом, в случае функционала бесконечной размерности $E(u)$ некоторое пространство функций $\to \mathbb{R}$ отображает функцию *u*, находящуюся в *некотором пространстве функций*, в вещественное число.

В качестве примера функционала можно привести интеграл от непрерывной функции на отрезке [0; 1]. Другой распространенный пример интеграла:

$$E(u(x)) = \int_0^1 \left(u(x)^2 + u'(x)^2 \right) dx.$$

Например, этот функционал отображает функцию x^2 для числа 5/3, которое является значением интеграла.

Аналог этого выражения для производной по времени в конечной размерности (когда допускается зависимость от времени) теперь выглядит:

$$\frac{dE(u(t))}{dt} = \nabla E . \frac{du(t)}{dt} = \nabla E . u'(t) = \langle \nabla E, u'(t) \rangle_{\text{некое пространство функций}}$$

Довольно удобно. Как правило, это позволяет точно определить градиент бесконечной размерности ∇E, если в своих вычислениях нам удастся выделить величину, умноженную на $u'(t)$. В таких вычислениях, как правило, используются интегральные выражения, а *произведение* обычно определяется в *некотором бесконечномерном* представлении, которое также включает интегральные выражения. Именно поэтому вместо обычной точки мы пользуемся нотацией произведения $\langle \nabla E, u'(t) \rangle$.

Рассмотрим один пример, демонстрирующий произведение для функций, находящихся в бесконечномерном пространстве $L^2(D)$, содержащем все функции $u(x)$ с конечным $\int_D |u(x)|^2 dx$ и градиент функционала в представлении $L^2(D)$.

Аналогия между функциями и функционалами оптимизации

Нам не нужно забывать об аналогии между формулировками конечной и бесконечной размерности, т. к. в математике все всегда четко сочетается. В то же время нужно быть осторожными, поскольку переход к бесконечности достаточно массивен, так что многие свойства и методы конечной размерности не выдерживают его.

В конечных измерениях *точка* или *точки оптимизации* удовлетворяют уравнению, составленному с учетом градиента *целевой функции* (альтернативные варианты в книге — функция потерь, функция затрат или функция полезности) равным нулю.

В бесконечных измерениях *функция* или *функции оптимизации* удовлетворяют дифференциальному уравнению, составленному с учетом градиента *целевого функционала* равным нулю, т. е. при условии, что нам каким-то образом удалось определить градиент *функционала*. Для того чтобы найти оптимизатор, нужно либо решить это дифференциальное уравнение, которое называется *уравнением Эйлера — Лагранжа*, либо следовать некоторой схеме оптимизации на ландшафте функционала. Ландшафт функционала бесконечной размерности *визуализировать невозможно*, так что, по иронии судьбы, визуализируем его в итоге как одномерный, где ось x представляет пространство функций u, а ось y — $E(u)$.

Примером схемы оптимизации является градиентный спуск, широко распространенный в машинном обучении конечной размерности. Та же идея градиентного спуска применима к бесконечной размерности — двигаться в направлении максимального увеличения (в случае максимизации) или уменьшения (в случае минимизации).

Безусловно, необходимо определить, что означает градиент для функционалов, заданных на пространствах бесконечной размерности. Оказывается, существует множество способов определения градиентов в зависимости от того, в каких пространствах находятся соответствующие функции (например, в пространстве всех непрерывных функций, пространстве всех функций, имеющих одну непрерывную производную, пространстве функций, интеграл от квадрата которых конечен, и т. д.). Значение градиента остается неизменным — он оценивает *вариацию* функционала в конкретном представлении по аналогии с тем, как градиент функции конечной размерности оценивает вариацию (изменение) функции в определенном направлении.

Можно пропустить остальную часть этого раздела, если вам неинтересны дифференциальные уравнения, т. к. этот раздел не является обязательным для исследования операций. Единственная цель состоит в том, чтобы изучить переход в бесконечную размерность и посмотреть, как можно получить дифференциальные уравнения в процессе минимизации формул, содержащих интегралы (функциональные формулы). Более подробную информацию о приведенных ниже примерах можно найти в PDF-файле по вариационному исчислению на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**).

Пример 1: гармонические функции, энергия Дирихле, уравнение теплопроводности

Гармоническая функция — это функция, сумма всех вторых производных которой равна нулю, например, в двумерном пространстве: $\Delta u = u_{xx} + u_{yy} = 0$. Примерами функций будут $e^x \sin(y)$ и $x^2 - y^2$. В реальной жизни такие функции встречаются в электростатике при моделировании электростатических потенциалов и распределений плотности заряда, а также при моделировании ритмических движений, например ритмического периодического движения струны или бесконечно долгого колебания маятника без трения.

Гармоническая функция минимизирует энергию Дирихле. Если не пытаться искать функции, чьи вторые производные сводятся к нулю (при $\Delta u = 0$ существуют надежные способы выполнить это), и которые удовлетворяют определенным граничным условиям, то можно представить гармоническую функцию в виде минимизатора функционала энергии, который называется функционалом энергии Дирихле:

$$E(u(x)) = \int_D \frac{1}{2} |\nabla u(x)|^2 \, dx.$$

Здесь $u(x)$ принадлежит соответствующему пространству функций, что гарантирует конечность интеграла, а $u(x) = h(x)$ задается на границе ∂D области D.

Когда для поиска минимизатора градиент этого функционала энергии задается равным нулю, получается уравнение (Эйлера — Лагранжа) $\Delta u = 0$, $u = h(x)$ на ∂D, т. е. именно такое дифференциальное уравнение, которому удовлетворяет гармоническая функция. Рассмотрим, как это происходит:

$$\begin{aligned} E'(u(x)) &= \int_D \frac{1}{2}\left(|\nabla u(x)|^2\right)' dx = \\ &= \int_D \nabla u(x) \nabla u'(x) dx = \\ &= -\int_D \Delta u(x) u'(x) dx + \int_{\partial D} \nabla u(x) . \vec{n} u'(x) ds = \\ &= -\int_D \Delta u(x) u'(x) dx + \int_{\partial D} \frac{\partial u(x)}{\partial \vec{n}} u'(x) ds. \end{aligned}$$

Третье равенство получается путем интегрирования по частям, которое переносит производную из одного коэффициента интеграла в другой, добавляя при этом отрицательный знак и пограничный член. Граничный член содержит два коэффициента исходного интеграла, но без производной, которая была перенесена с одного на другой.

Предыдущее выражение справедливо для любого $u'(x)$ в нашем пространстве функций; в частности, оно будет справедливо для случаев, когда $u'(x) = 0$ на границе области, что уничтожает интегральный член на границе, и мы получаем:

$$E'(u(x)) = -\int_D \Delta u(x) u'(x) dx = \left\langle -\Delta u(x), u'(x) \right\rangle_{L^2(D)}.$$

Мы только что определили *произведение* в пространстве функций $L^2(D)$ как интеграл от обычного произведения функций над областью.

Теперь по аналогии с примером конечной размерности запишем:

$$E'(u(x)) = \left\langle \nabla_{L^2(D)} E(u(x)), u'(x) \right\rangle_{L^2(D)}.$$

Сравнивая два последних выражения, можно отметить, что градиент энергии Дирихле $E(u)$ в представлении $L^2(D)$ равен $-\Delta u(x)$, и только в случае гармонических функций он равен нулю.

Уравнение теплопроводности выполняет градиентный спуск для функционала энергии Дирихле. В этом сюжете есть кое-что еще. В природе, где система эволюционирует во времени, первый вопрос обычно звучит так: что движет эволюцией? Интуитивно понятный ответ заключается в том, что система эволюционирует таким образом, чтобы *энергия уменьшалась самым эффективным способом*. Обычно найти формулы для этих *энергетических функционалов* не так-то просто, но, когда это удается, можно получить докторскую степень, как это вышло в моем случае.

Простой пример — это уравнение теплопроводности $u_t = \Delta u$ при $u(x, t) = 0$ на ∂D и некотором начальном условии $u(x, 0) = g(x)$. Оно моделирует диффузию тепла, дыма, атомов на поверхности материала и т. д. В него естественным образом встроена зависимость от времени. Если мы проследим эволюцию решения уравнения тепла (которое может представлять температуру, концентрацию растворителя, газ в комнате и т. д.) во времени $u(x, t)$, то обнаружим, что скользим по ландшафту функционала энергии Дирихле $E(u) = \frac{1}{2}\int_D |\nabla u(x, t)|^2 \, dx$ в направлении наиболее крутого спуска в представлении $L^2(D)$, поскольку, как обсуждалось ранее:

$$u_t = \Delta u = -\nabla_{L^2(D)} E(u).$$

Это значит, что, начиная с некоторого $u(x, 0) = g(x)$, самый быстрый путь к минимизатору энергии Дирихле (который является гармонической функцией) на бесконечномерном ландшафте энергии Дирихле лежит через решение уравнения теплопроводности — через отслеживание пути начального $g(x)$, изменяющегося со временем.

В этом смысле уравнение теплопроводности дает возможность сформулировать схему минимизации для задачи:

$$\min_{u=0 \text{ на } \partial D} \frac{1}{2}\int_D |\nabla u(x, t)|^2 \, dx.$$

Пример 2: кратчайший путь между двумя точками по соединяющей их прямой

Кратчайший путь между двумя точками в $\mathbb{R}^2$ проходит по соединяющей их прямой. Для этого минимизируем длину дуги кривой, соединяющей точки (x_1, y_1) и (x_2, y_2), а именно:

$$\min_{y(x_1)=y_1 \text{ и } y(x_2)=y_2} \int_{x_1}^{x_2} \sqrt{1 + y'(x)^2} \, dx.$$

Как и в предыдущем примере, эта задача представляет собой задачу минимизации функционала, содержащего интегральные выражения некоторых функций и их производных.

Для ее решения запишем уравнение Эйлера — Лагранжа, задав градиент функционала равным нулю. В результате получаем минимизирующую функцию $y(x) = mx + b$, где m и b — соответственно наклон и y-пересечение прямой, соединяющей две заданные точки. Подробности этого примера приведены в PDF-файле, посвященном вариационному исчислению, на странице книги на GitHub (**https://github.com/halanelson/Essential-Math-For-AI**).

Другие ознакомительные примеры вариационного исчисления

Другие вводные примеры вариационного исчисления, которые мы можем решить с помощью минимизации соответствующего функционала энергии (посредством вариационного принципа), включают задачу о минимальной поверхности и изопериметрическую задачу.

Оптимизация в сетях

Мы начнем с оптимизации в сетях *до* симплекс-метода для линейной оптимизации, т. к. большинство людей, несмотря на обилие сетевых структур в приложениях исследования операций и в самой природе, привыкли мыслить в алгебраических категориях (уравнений и функций), а не в терминах графов или сетевых структур. Поэтому необходимо научиться работать с графовыми моделями. Как правило, задачи оптимизации сетевых структур имеют комбинаторный характер $O(n!)$, что далеко не всегда хорошо, значит, нам требуются алгоритмы, которые каким-то образом обходят это и эффективно просеивают пространство поиска. (Запомним, что порядок задачи — это в большинстве случаев наихудший сценарий, и здесь мы довольствуемся приближенными решениями.)

Рассмотрим типичные сетевые задачи, охватывающие широкий спектр реальных приложений. Начнем с одной из старейших и наиболее известных задач — с *задачи о коммивояжере*. Мы живем в эпоху, когда существуют программные пакеты с открытым исходным кодом и облачные вычислительные ресурсы, включающие мощные алгоритмы для решения всех задач, упомянутых в этой главе, поэтому в данном разделе мы сосредоточимся на понимании сетевых задач и приложений на их основе, а не на алгоритмах их решения.

Задача о коммивояжере

Эта знаменитая задача из области исследования операций подходит ко многим реальным ситуациям. Коммивояжер должен посетить несколько городов во время поездки. Учитывая расстояния между городами, в каком порядке он должен ехать, чтобы посетить каждый город ровно один раз и вернуться домой, причем пройденное расстояние должно быть минимальным (рис. 10.1)?

Можно перечислить множество вариантов практического применения: курьерская машина должна доставить посылки со склада по всем адресам минимально затратным способом (по времени или расстоянию); в процессе производства электронных микросхем требуется найти самую эффективную последовательность сверления отверстий на печатной плате.

Представим задачу о коммивояжере как задачу оптимизации на графе, где города — это узлы, между каждой парой городов есть ребра (что делает граф полным), и каждое ребро имеет вес (или атрибут, или признак), представляющий расстояние между двумя городами. Этот граф имеет множество путей, которые проходят через

все города только один раз и возвращаются в тот, с которого мы начинали (*гамильтонов контур*), но нам нужен путь, имеющий минимальное итоговое расстояние.

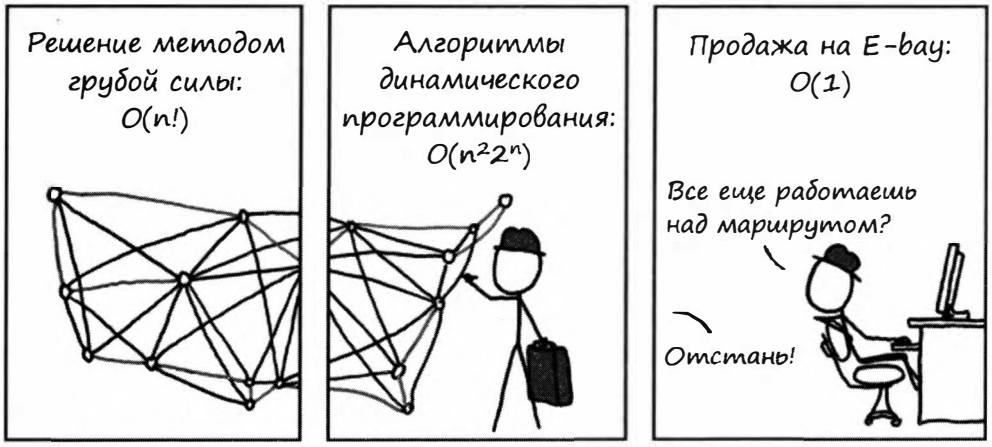


Рис. 10.1. Задача о коммивояжере (источник изображения: **https://xkcd.com/399**)

Это не такая уж простая задача. Общее количество различных гамильтоновых контуров в полном графе из n узлов составляет $(n-1)!/2$. Начиная с любого узла, имеется $n-1$ ребро, чтобы выбрать следующий город для посещения, затем $n-2$ варианта из второго города, $n-3$ из третьего и т. д. Эти варианты независимы, так что в общей сложности имеется $(n-1)!$ вариантов. Их нужно разделить на 2, чтобы учесть симметрию в том смысле, что один и тот же гамильтонов контур можно пройти как вперед, так и назад и при этом получить одинаковое общее пройденное расстояние. Такая задача подсчета представляет собой круговую перестановку с симметрией. Исчерпывающее решение задачи о коммивояжере перечислит все $(n-1)!/2$ гамильтоновых контуров, суммируя пройденное в каждом из них расстояние, а затем выберет наикратчайший. Даже в случае приемлемого значения n это будет слишком дорого; например, чтобы посетить все 50 столиц штатов США (допустим, необходимо минимизировать общую стоимость поездки), придется перебрать $(50-1)!/2 = 3{,}04 \cdot 10^{62}$ варианта! Эффективного алгоритма для задач произвольного размера не существует. Отличные приближенные решения можно получить с помощью эвристических методов. Более того, для огромного количества городов эту задачу оптимально решили замечательные алгоритмы на основе *метода ветвей и обрезки*.

Минимальное остовное дерево

Задача о минимальном остовном дереве идет сразу после задачи о коммивояжере, т. к. люди часто путают эти две задачи. Сейчас самое время устранить эту путаницу. Мы имеем полносвязную сеть с положительными весами, связанными с каж-

дым ребром, которые опять же могут представлять расстояние, время, мощность или стоимость подключения инфраструктуры, например водопроводной магистрали, электрической сети или телефонной линии. Как и в задаче о коммивояжере, необходимо найти набор ребер, содержащий все узлы графа и минимизирующий общий вес. Однако требование здесь будет другим — необходимо убедиться в том, что мы выбираем набор ребер так, чтобы обеспечить путь между любыми двумя парами узлов, т. е. добраться из одного узла графа до любого другого. В задаче о коммивояжере каждый город нужно посетить только один раз, а затем вернуться в исходный, и это означает, что каждый узел не может получить более двух ребер (такого требования нет в случае остовного дерева). Возвращение в последний город в задаче о коммивояжере означает, что имеется лишнее ребро замыкания контура, которое не требуется для остовного дерева. Если убрать это последнее ребро из решения задачи о коммивояжере, то определенно получится остовное дерево, правда, нельзя гарантировать, что оно будет иметь минимальную стоимость. На рис. 10.2 показаны решения задач минимального остовного дерева и коммивояжера для одного и того же графа.

Рис. 10.2. Решение задач о минимальном остовном дереве и о коммивояжере для одного и того же графа

Важно отметить, что если в какой бы то ни было сети имеются узлы, то потребуется только $n-1$ ребро, чтобы между любыми двумя узлами был путь, поэтому для минимального остовного дерева нельзя использовать больше $n-1$ ребер, т. к. это увеличит стоимость. Необходимо выбрать набор ребер, минимизирующий издержки.

Мы уже упоминали такие приложения, как проектирование телекоммуникационных, электрических, маршрутных и транспортных, а также инфраструктурных сетей (трубопроводов). Разработка таких сетей требует больших затрат, поэтому их оптимальное проектирование позволяет сэкономить миллионы долларов.

Кратчайший путь

Простейший вариант задачи о кратчайшем пути заключается в том, что имеются две вершины на графе и требуется соединить их набором ребер так, чтобы общая сумма весов ребер (расстояние, время) была минимальной. Эта задача отличается

от задач о коммивояжере и минимальном остовном дереве тем, что здесь не нужно заботиться о том, чтобы охватить все узлы графа. Единственное, к чему мы стремимся, — это добраться из пункта отправления в пункт назначения наименее затратным способом.

Одно из очевидных применений — поездка из одного пункта назначения в другой с минимальным расстоянием, стоимостью, временем и т. д. Другие менее очевидные, но довольно важные приложения — сетевой график. Вместо пунктов отправления и назначения может быть начало и конец проекта. Каждый узел представляет одно действие, а вес каждого ребра — понесенные издержки или потраченное время, если действие *i* примыкает к действию *j* (в случае ориентированного графа это будут понесенные издержки и потраченное время, если действие *i* произойдет после действия *j*). Задача состоит в выборе последовательности работ, минимизирующей общие издержки.

Другие варианты задачи о кратчайшем пути включают поиск кратчайшего пути от начального узла до *всех остальных узлов* или поиск кратчайших путей между *всеми парами узлов*.

Многие алгоритмы маршрутизации транспортных средств и алгоритмы проектирования сетей включают алгоритмы поиска кратчайшего пути в качестве подпрограмм.

Также можно сформулировать задачу о кратчайшем пути как задачу линейной оптимизации и пользоваться методами из ее инструментария.

Максимальный поток и минимальный разрез

Здесь также имеются пункт отправления и пункт назначения, каждое *направленное* ребро имеет определенную пропускную способность (максимальное количество транспортных средств на маршруте, максимальное количество товаров в пути, максимальное количество материалов или природных ресурсов, например нефти или воды, которые может перекачать трубопровод), и требуется найти набор ребер, максимизирующий *поток* от пункта отправления до пункта назначения. Важно отметить, что все ребра направлены от исходной точки к конечной.

В определении оптимального (максимального потока) набора ребер, соединяющих начало с пунктом назначения, важнейшую роль играет *теорема о максимальном потоке и минимальном разрезе* из теории графов.

Теорема гласит, что максимальный поток из пункта отправления в пункт назначения через направленную сеть равен минимальной сумме весов ребер, необходимой для прерывания любой связи между пунктом отправления и пунктом назначения. То есть можно разрезать сеть и прервать связь между пунктом отправления и пунктом назначения более чем одним способом. Набор ребер, прерывающий связь *и* имеющий наименьший вес, называется минимальным набором ребер. Значение этого набора равно значению максимально возможного потока в сети. Такой вывод скорее интуитивен: что можно максимально передать через ребра сети? Он ограни-

чен сверху пропускной способностью ребер, имеющих решающее значение для соединения пункта отправления с пунктом назначения.

Задачу о максимальном потоке можно переформулировать в линейной оптимизации, и, безусловно, задача о минимальном разрезе будет ее двойником, так что, само собой, они будут иметь одинаковое решение! Мы увидим это далее по ходу главы.

Наконец, предположим, что имеется более одного пункта отправления и более одного пункта назначения, аналогично распределительной сети. Тогда по-прежнему можно максимизировать поток через сеть, решая точно такую же задачу, только теперь добавляется вымышленный *исходный суперпункт*, указывающий на все реальные пункты отправления, и еще один вымышленный *конечный суперпункт*, на который указывают все реальные пункты назначения, имеющие бесконечную емкость. А затем поступим как обычно, решив задачу о максимальном потоке на этом новом графе с двумя новыми вымышленными *суперузлами*.

Максимальный поток и минимальная стоимость

Эта задача похожа на задачу максимального потока, за исключением того, что теперь имеется стоимость, связанная с отправкой потока через каждое ребро, которая пропорциональна количеству единиц потока. Очевидно, что цель состоит в минимизации затрат при осуществлении поставок из всех пунктов отправления во все пункты назначения. Эту задачу можно сформулировать как задачу линейной оптимизации и решить ее с помощью симплекс-метода, оптимизированного для сетей.

Области применения универсальны и крайне важны — это всевозможные распределительные сети с узлами поставок, перевалочными узлами, узлами спроса, цепочки поставок (товаров, крови, ядерных материалов, продуктов питания), сети управления твердыми отходами, координация типов продукции для производства или расходования ресурсов на нужды рынка, управление денежными потоками, проблемы назначения, например назначение сотрудников и временны́х интервалов на конкретные задачи или кандидатов на свободные рабочие места.

Задача о назначениях

Задачу о назначениях также называют проблемой матчинга. Количество назначенных должно соответствовать количеству заданий, каждому может быть назначено только одно задание, и каждое задание может быть выполнено лишь одним назначенным. Поручение задачи i назначенцу j сопряжено с определенными издержками. Цель состоит в том, чтобы выбрать такое соответствие между задачами и назначенцами, которое будет минимизировать общие издержки. Граф такой задачи относится к специальному типу, который называется *двудольным графом*. Такой граф можно разделить на две части, так что все ребра будут идти от одной вершины в первой части к одной вершине во второй части. Задача о назначениях с одинаковыми весами — это задача максимального потока на двудольном графе. Все, что

требуется, — это задать вымышленные суперпункты отправления и назначения и решить эту задачу так же, как решается задача максимального потока в следующем разделе, посвященном линейной оптимизации и двойственности. Для решения этих задач существует множество эффективных алгоритмов.

Метод критического пути для разработки проекта

Метод критического пути (critical path method, CPM) — это метод оптимизации сети, представляющей все виды работ в проекте, общий бюджет, общее ограничение по времени, очередность выполнения работы, сколько времени и затрат требует каждая работа и какие работы могут выполняться одновременно. Например, можно представить проект строительства дома от начала до конца. Отличным инструментом для разработки проекта является метод критического пути с учетом компромиссов между издержками и временем, который гарантирует, что при минимальных суммарных издержках проект уложится в сроки. По аналогии с методом критического пути существует *метод оценки и анализа программ* (Program Evaluation Review Technique, PERT) — инструмент планирования управления проектами, который позволяет рассчитать кратчайший возможный, максимально возможный и наиболее вероятный сроки выполнения проекта.

Задача об *n* ферзях

Прежде чем мы перейдем к линейной оптимизации, симплекс-методу и двойственности, немного отвлечемся и вспомним интересную комбинаторную задачу, которая не оставляла математиков в покое на протяжении 150 лет в основном из-за полного отсутствия структуры — это задача об *n* ферзях (рис. 10.3). В июле 2021 года Майкл Симкин представил ее решение (**https://oreil.ly/Q3soe**). Приведем выдержки из аннотации его статьи с решением, озаглавленным "Число конфигураций *n* ферзей" (**https://oreil.ly/aTPHD**):

> "Задача об *n* ферзях заключается в том, чтобы определить количество способов расставить *n* ферзей на (шахматной) доске $n \times n$ так, чтобы ни один из них не находился под ударом другого. Мы покажем, что существует такая константа $\alpha = 1{,}942 \pm 3 \cdot 10^{-3}$, при которой (количество способов размещения на доске ферзей, не угрожающих друг другу, равно) $\left(1 \pm o(1) ne^{-\alpha}\right)^n \left(\left(1 \pm o(1)\right) ne^{-\alpha}\right)^n$. Константа α характеризуется как решение выпуклой задачи оптимизации в $P\left(\left[-1/2; 1/2\right]^2\right)$ — пространстве вероятностных мер Бореля на квадрате".

На портале GeeksforGeeks (**https://oreil.ly/t5wW3**) приведен простой *алгоритм с отходом* для решения задачи об *n* ферзях. Важно отметить, что решение Симкина количественно определяет общее число приемлемых конфигураций ферзей, в то время как алгоритмы находят только одну или несколько таких конфигураций.

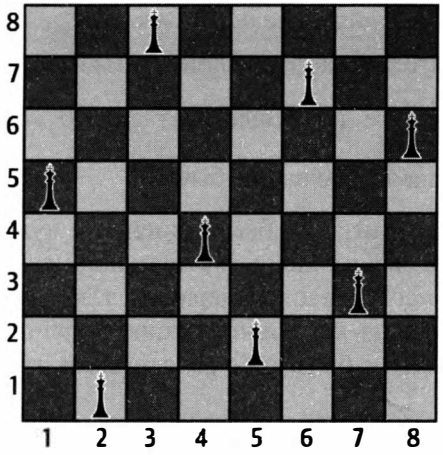


Рис. 10.3. Восемь ферзей расставлены на шахматной доске 8 × 8 так, что ни один из них не находится под ударом другого

Линейная оптимизация

Любая *линейная* или *нелинейная* задача оптимизации в конечной размерности имеет вид:

$$\min_{\substack{g_1(\vec{x})\le 0\\ g_2(\vec{x})\le 0\\ \cdots\\ g_m(\vec{x})\le 0}} f(\vec{x}).$$

Точка $\vec{x}$, удовлетворяющая всем ограничениям, считается *допустимой точкой*. У нас имеются следующие варианты.

Существует только одно оптимальное решение.

Считается, что ландшафт целевой функции имеет только одну минимальную точку.

Существует несколько оптимальных решений.

В этом случае набор оптимальных решений может быть ограниченным или неограниченным.

Оптимальное значение стремится к $-\infty$.

Ландшафт целевой функции бесконечно убывает, так что ни одна из допустимых точек не является оптимальной.

Допустимый набор пуст.

Нас не интересуют целевая функция и ее минимальные значения, т. к. не существует точек, одновременно удовлетворяющих всем ограничениям. Задача минимизации не имеет решения.

Оптимальное значение конечное, но не достижимое.

Отсутствует оптимизатор, даже если допустимый набор не пустой. Например, $\inf_{x \geq 0} \frac{1}{2}$ равно нулю, однако не существует такого конечного x, при котором $1/x = 0$. В линейных задачах такого не бывает.

Для того чтобы задача оптимизации была линейной, нужно, чтобы целевая функция f и все ограничения g были линейными функциями. Линейная оптимизация занимает львиную долю в исследовании операций, т. к. многие задачи исследования операций можно моделировать как минимизацию линейной функции с линейными ограничениями, которые могут быть либо равенствами, либо неравенствами.

Общая и стандартная формы

Линейность — замечательная вещь, позволяющая использовать весь инструментарий линейной алгебры (векторные и матричные вычисления). Существуют две распространенные формы задач линейной оптимизации.

Общая форма.

Используется в развитии теории линейного программирования. В ней отсутствуют ограничения на знаки переменных решения (записи вектора $\vec{x}$):

$$\min_{\mathbf{A}\vec{x} \geq \vec{b}} (\vec{c}.\vec{x}).$$

Допустимый набор $\mathbf{A}\vec{x} \geq \vec{b}$ — многогранник (**https://oreil.ly/1CybB**), который можно представить как пересечение конечного числа полупространств с плоскими границами. Он бывает ограниченным или неограниченным. Вскоре мы рассмотрим примеры.

Стандартная форма.

Применяется для вычислений и разработки таких алгоритмов, как симплекс-метод и метод внутренних точек. Переменные решения должны быть неотрицательными, поэтому нужно искать оптимизаторы только в *первом гипероктанте*, высокоразмерном аналоге первого квадранта, где все координаты неотрицательны. Кроме того, ограничения всегда должны быть равенствами, а не неравенствами, поэтому мы находимся на границе многогранника, а не внутри него. Это задача линейной оптимизации, записанная в стандартной форме:

$$\min_{\substack{\mathbf{A}\vec{x} = \vec{b} \\ \vec{x} \geq \vec{0}}} (\vec{c}.\vec{x}).$$

Существует простой способ интуитивно понять линейную задачу в стандартной форме — составить вектор $\vec{b}$ из столбцов **A** так, чтобы минимизировать стоимость $\vec{c}.\vec{x}$.

От стандартной формы можно легко перейти к общей форме задачи линейной оптимизации. Например, ввести избыточные и фиктивные переменные, чтобы преоб-

разовать общую задачу линейной оптимизации в стандартную форму, причем важно отметить, что в итоге мы получаем одну и ту же задачу в разных измерениях. С вводом переменной для изменения неравенства на равенство, например, вводим s_1, чтобы преобразовать неравенство $x_1 - 3x_2 \geq 4$ в равенство $x_1 - 3x_2 - s_1 = 4$, размерность увеличивается (в данном примере с двух до трех). Это нормально. Одна из замечательных особенностей математики заключается в том, что мы можем моделировать неограниченное количество измерений, хотя живем, по сути, только в трехмерном мире.

Визуализация задачи линейной оптимизации в двух измерениях

Визуализируем двумерную задачу, не имеющую ни общей, ни стандартной формы (но ее можно легко преобразовать в любую форму, что не требуется для такой простой задачи, т. к. можно извлечь минимум, исследуя граф):

$$\min_{\substack{x+2y \leq 3 \\ 2x+2y \leq 3 \\ x \geq 0 \\ y \geq 0}} (-x - y).$$

На рис. 10.4 показаны границы всех ограничений (прямые линии) этой задачи линейной оптимизации, а также полученный допустимый набор. Оптимальное значение объективной функции $-x - y$ равно -2 и достигается в точке (1; 1), которая является одним из углов допустимого набора.

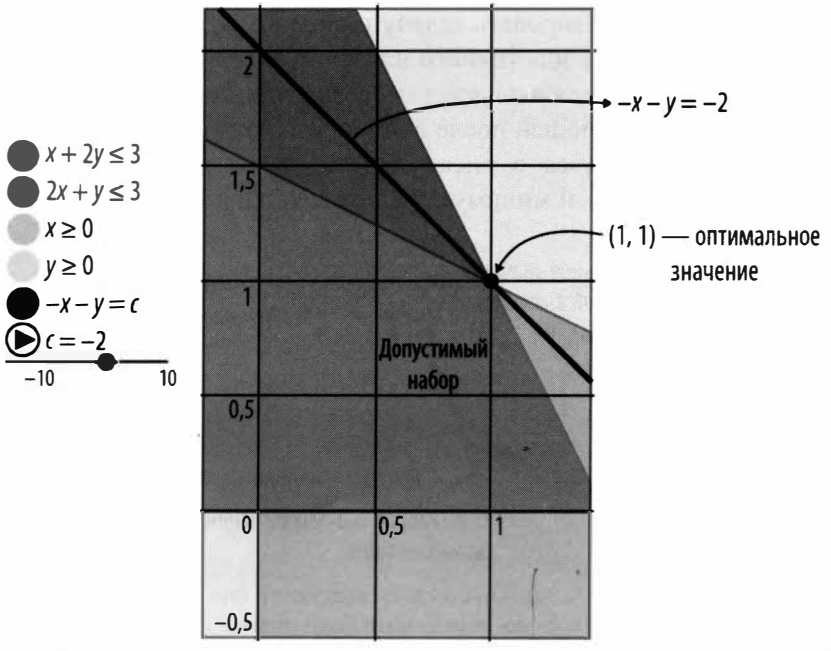


Рис. 10.4. Допустимый набор и оптимальное значение функции $-x - y$ равно -2, достигаемое в угловой точке (1; 1)

В случае задачи без ограничений инфимум[2] $-x-y$ был бы $-\infty$. Ограничения существенно меняют ситуацию. Причем оптимальное значение не случайно находится в одном из углов многоугольника (двумерного многогранника). Если провести прямую $-x-y=c$ для некоторого c, которое располагает часть линии внутри допустимого набора, а затем двигаться в направлении отрицательной части вектора градиента (напомним, что это направление наиболее быстрого спуска), то линия будет двигаться в направлении вектора $-\nabla(-x-y)=-(-1;1)=(1;1)$ (безусловно, это совпадение, что вектор градиента имеет те же координаты, что и оптимизирующая точка, поскольку они совершенно не связаны). Пока часть линии находится внутри допустимого набора, можно *продолжать двигать* и уменьшать c, пока уже будет нельзя продолжать движение, т. к. в этом случае мы выйдем из допустимого набора, попадем в недопустимый и потеряем всю проделанную работу. Это происходит ровно тогда, когда вся линия оказывается за пределами допустимого набора, слегка зависнув в точке (1; 1), которая все еще находится в допустимом наборе.

Итак, мы нашли оптимизатор — точку, в которой значение $-x-y$ будет наименьшим. Скоро мы вернемся к движению по углам допустимых наборов линейных задач, поскольку именно там находятся оптимизаторы.

От выпуклости к линейности

Даже если целевая функция является нелинейной, во многих случаях нам может посчастливиться переформулировать задачу в линейную, а затем использовать методы линейной оптимизации для точного или приближенного к нему решения. Одним из таких случаев является выпуклая целевая функция. В задачах оптимизации выпуклость считается следующей после линейности желательной характеристикой, позволяющей не беспокоиться о риске застрять в локальном минимуме — для выпуклой функции локальный минимум также является и глобальным минимумом.

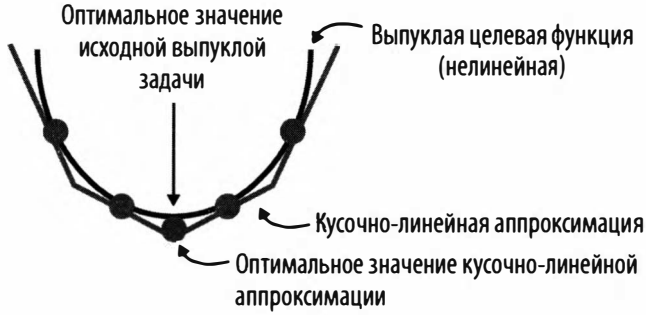


Рис. 10.5. Аппроксимация выпуклой функции кусочно-линейными функциями

[2] Inf (от лат. *infimum*) — точная нижняя граница. — *Прим. ред.*

Выпуклую (дифференцируемую) функцию всегда можно аппроксимировать кусочно-линейной выпуклой функцией, как показано на рис. 10.5. После этого можно превратить задачу оптимизации с кусочно-линейной целевой функцией в задачу с линейной целевой функцией. Однако этот процесс приведет к потере дифференцируемости на первом этапе (функция перестает быть гладкой), а также увеличению размерности на втором этапе. За все приходится платить.

Задача *выпуклой оптимизации* имеет выпуклую целевую функцию и выпуклый допустимый набор. Выпуклая оптимизация представляет собой целую область.

Выпуклая функция

Функция f: $\mathbb{R}^n \to \mathbb{R}$ выпуклая тогда и только тогда, когда

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda)f(y)$$

при всех $x, y \in \mathbb{R}^n$ и $0 \le \lambda \le 1$. Это значит, что отрезок, соединяющий любые две точки на графе f, лежит выше графа f.

Приведем несколько полезных фактов о выпуклых функциях.

- Выпуклая функция не может иметь локальный минимум, который бы не был глобальным минимумом.
- Если функции $f_1, f_2, ..., f_m : \mathbb{R}^n \to \mathbb{R}$ — выпуклые, то $f(x) = \max_i f_i(x)$ также является выпуклой функцией. В этом случае f может потерять гладкость, так что методы оптимизации не смогут использовать производные.
- Функция $f(x) = \max\{m_1 x + d_1, m_2 x + d_2, ..., m_n x + d_n\}$ или, более компактно, $f(x) = \max_{i=1, 2, ..., n} \{m_i x + d_i\}$ является кусочно-линейной, как показано на рис. 10.6.

  Это выпуклая функция, т. к. каждое $m_i x + d_i$ является выпуклым (линейные функции выпуклы и вогнуты одновременно), а максимум выпуклой функции также будет выпуклым.

Теперь можно переформулировать задачи оптимизации с кусочно-линейными выпуклыми целевыми функциями в задачи линейной оптимизации:

$$\min_{\mathbf{A}x \ge b} \left(\max_i m_i . x + d_i\right) \leftrightarrow \min_{\substack{\mathbf{A}x \ge b \\ z \ge m_i . x + d_i}} z.$$

Важно отметить, что, добавив новую переменную принятия решения z, мы увеличили размерность.

Например, функция абсолютной величины $f(x) = |x| = \max\{x, -x\}$ является кусочно-линейной и выпуклой. Задачу оптимизации, в которой функция абсолютной величины включает абсолютные значения переменных решения, можно переформулировать в задачу линейной оптимизации двумя способами (здесь c_i в целевой

функции должны быть неотрицательными, в противном случае целевая функция может оказаться невыпуклой):

$$\min_{\mathbf{A}x\geq b}\sum_{i=1}^{n}c_1|x_i| \leftrightarrow \min_{\substack{\mathbf{A}x\geq b\\ z_i\geq x_i\\ -z_i\geq x_i}}\left(\sum_{i=1}^{n}c_i z_i\right) \leftrightarrow \min_{\substack{\mathbf{A}x^+-\mathbf{A}x^-\geq b\\ x^+,\,x^-\geq 0}}\left(\sum_{i=1}^{n}c_i\left(x_i^+ + x_i^-\right)\right).$$

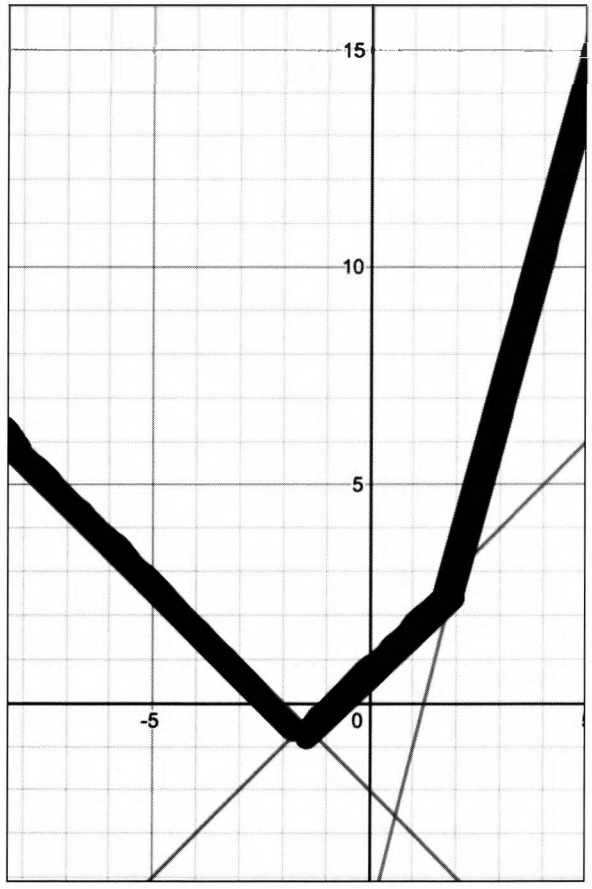


Рис. 10.6. Максимум линейной функции является кусочно-линейным и выпуклым

Геометрия линейной оптимизации

Представим геометрию задачи линейной оптимизации в стандартном виде, поскольку именно такая форма наиболее удобна для алгоритмов поиска минимизатора.

Геометрия — это все, что связано с фигурами, линиями, поверхностями, точками, ребрами, углами и т. д.

Задача в стандартной форме

$$\min_{\substack{\mathbf{A}\vec{x}=\vec{b}\\ \vec{x}\geq\vec{0}}}\left(\vec{c}.\vec{x}\right)$$

включает в себя линейные алгебраические уравнения. Нам нужно понять геометрическую картину, связанную с этими уравнениями, а также процесс минимизации. Вспомним, что линейность подразумевает плоскость, а когда плоские объекты пересекаются друг с другом, они образуют гиперплоскости, линии и/или углы.

Линейные ограничения задачи минимизации задают многогранник. Для нас крайне важны углы этого многогранника. Но как узнать, что многогранник имеет углы? А что если это всего лишь полупространство? Как мы уже говорили, когда мы переходим от общей формы к стандартной, мы перескакиваем на размерность выше. Более того, мы принудительно вводим неотрицательность для переменных решения. Поэтому, даже если многогранник не имеет углов в общем виде, он всегда будет иметь углы в своей более высокоразмерной стандартной форме — многогранник стандартной формы располагается в первом гипероктанте, а значит, не может содержать полных линий. Это хорошо. Существуют теоремы, гарантирующие, что в задаче линейной оптимизации либо оптимальное значение равно $-\infty$, либо существует конечное оптимальное значение, которое можно достичь в одном из углов многогранника. Следовательно, при поиске оптимизатора необходимо сосредоточить внимание на этих углах. Поскольку многие многогранники, связанные с реальными ограничениями, имеют десятки тысяч углов, требуются эффективные способы их просеивания.

Интуитивно понятно, что можно начать с координат одного угла многогранника, а затем проделать путь к оптимальному углу. Но как найти координаты этих углов? Мы пользуемся методами линейной алгебры. Поэтому ограничения лучше выразить в виде линейной системы $\mathbf{A}\vec{x} = \vec{b}$ (при $\vec{x} \geq 0$). На языке линейной оптимизации угол называется *допустимым базисным решением*. Такое алгебраическое название указывает на то, что координаты угла удовлетворяют всем ограничениям (допустимы) и решают линейные уравнения, связанные с неким базисом (полученным из системы $\mathbf{A}\vec{x} = \vec{b}$, в частности, из m столбцов матрицы $\mathbf{A}$).

Взаимосвязь между алгеброй и геометрией

Прежде чем мы рассмотрим симплекс-метод, попробуем связать алгебраические уравнения (или неравенства для задач общего вида) ограничений с геометрическими мысленными образами.

Многогранник.

Все ограничения в целом образуют многогранник. С точки зрения алгебры, многогранник — это множество точек, удовлетворяющих линейной системе $\vec{x} \in \mathbb{R}^n$, так что $\mathbf{A}\vec{x} \geq \vec{b}$ для некоторых $\mathbf{A}_{m \times n}$ и $\vec{b} \in \mathbb{R}^m$.

Внутренность полупространства.

Здесь рассматривается *только одно* неравенство из ограничений, а не вся система $\mathbf{A}\vec{x} \geq \vec{b}$, а именно $\vec{a}_i.\vec{x} > \vec{b}_i$ (строгая часть неравенства одного ограничения неравенства). Оно соответствует всем точкам, лежащим на одной стороне отно-

сительно одной грани многогранника. Поскольку неравенство строгое, мы находимся не на границе полупространства, а внутри него.

Гиперплоскость.

Здесь рассматривается только одно ограничение равенства $\vec{a}_i.\vec{x} = \vec{b}_i$ или только часть равенства в ограничении неравенства. Это граница полупространства $\vec{a}_i.\vec{x} > \vec{b}_i$ или одна грань многогранника.

Активные ограничения.

Когда мы подставляем координаты точки $\vec{x}^*$ в ограничение $\vec{a}_i.\vec{x} \geq \vec{b}_i$ и получаем равенство, т. е. $\vec{a}_i.\vec{x} = \vec{b}_i$, значит, ограничение *активно* в этой точке. Геометрически это помещает $\vec{x}^*$ не внутрь, а на границу полупространства.

Угол многогранника.

Геометрически для образования угла требуется пересечь нужное количество гиперплоскостей. Алгебраически в угловой точке действует нужное количество ограничений. Это *допустимое базисное решение*, которое мы рассмотрим при обсуждении симплекс-метода.

Смежные базисы для нахождения смежных углов.

Это два набора столбцов матрицы **A**, которые имеют все общие столбцы, кроме одного. С их помощью вычисляются координаты смежных углов. В симплекс-методе необходимо геометрически перейти от одного угла к соседнему, а *смежные базисы* позволяют сделать это систематическим алгебраическим способом. Мы также рассмотрим это при обсуждении симплекс-метода.

Вырожденный случай.

Для наглядного представления предположим, что имеются две линии, пересекающиеся в двух измерениях и образующие угол. Если третья линия пересекается с ними в той же точке, или, другими словами, если в одной и той же точке в двух измерениях активно более двух ограничений, то происходит *вырождение*. В *n* измерениях точка $\vec{x}^*$ имеет более *n* проходящих через нее гиперплоскостей, или более *n* активных ограничений. С точки зрения алгебры, в результате такого вырождения с нашим алгоритмом оптимизации происходит следующее: когда мы выбираем другой набор линейно независимых столбцов **A** для решения *другого* допустимого базисного решения (угла) в итоге может получиться то же самое, что и раньше, а это обеспечивает цикличность алгоритма!

Симплекс-метод

Наша цель — разработать алгоритм, который находит оптимальное решение задачи линейной оптимизации в стандартной форме:

$$\min_{\substack{A\vec{x} = \vec{b} \\ \vec{x} \geq \vec{0}}} (\vec{c}.\vec{x}),$$

где **A** — это матрица размера $m \times n$ с m линейно независимыми строками (так что $m \le n$), $\vec{b}$ — вектор размера $m \times 1$, а $\vec{c}$ и $\vec{x}$ — векторы размера $n \times 1$. Без потери общности предполагается, что m строк матрицы **A** линейно независимы, что означает отсутствие избыточности в ограничениях задачи. Это также гарантирует наличие хотя бы одного набора m линейно независимых столбцов **A** (ранг(**A**) = m). Эти линейно независимые столбцы, или базис, нужны нам для того, чтобы начать поиск оптимизатора в определенном углу многогранника и при помощи симплекс-метода переходить от одного угла к другому.

Основная идея симплекс-метода

Начинаем с угла многогранника (который также называется допустимым базисным решением), двигаемся к другому углу в направлении, гарантированно уменьшающем целевую функцию, или стоимость, пока либо не достигнем оптимального решения, либо не обнаружим, что задача не имеет границ, а оптимальная стоимость равна $-\infty$ (это выясняется с помощью определенных условий оптимальности, которые становятся критериями завершения алгоритма). В случае вырожденных задач существует вероятность зацикливания, но можно избежать этого, если делать грамотный выбор (систематический способ выбора) при возникновении связей в процессе.

Симплекс-метод скачет по углам многогранника

Приведем задачу линейной оптимизации в трех измерениях:

$$\min_{\substack{x_1 \le 2 \\ x_3 \le 3 \\ 3x_2 + x_3 \le 6 \\ x_1 + x_3 \le 4 \\ x_1, x_2, x_3 \ge 0}} \left(-x_1 + 5x_2 - x_3\right).$$

На рис. 10.7 показан многогранник, соответствующий семи линейным ограничениям.

Важно отметить, что задача приведена не в стандартной форме. Если мы преобразуем ее в стандартную форму, то получим четыре дополнительных измерения, соответствующих четырем новым переменным, необходимые для преобразования четырех ограничений неравенства в ограничения равенства.

Невозможно визуализировать семимерный многогранник, но этого и не требуется. Можно оперировать симплекс-алгоритмом в семи измерениях и отслеживать ключевые переменные в трех измерениях: (x_1, x_2, x_3). Таким образом, можно проследить путь симплекс-метода от одного угла многогранника к другому, уменьшая значение целевой функции $-x_1 + 5x_2 - x_3$ на каждом шаге, пока не найдем угол с минимальным значением.

Можно пропустить остальную часть этого раздела и сразу перейти к *разд. "Транспортная задача и задача о назначении" далее в этой главе*, если не хотите вдаваться в подробности симплекс-метода и его всевозможных реализаций.

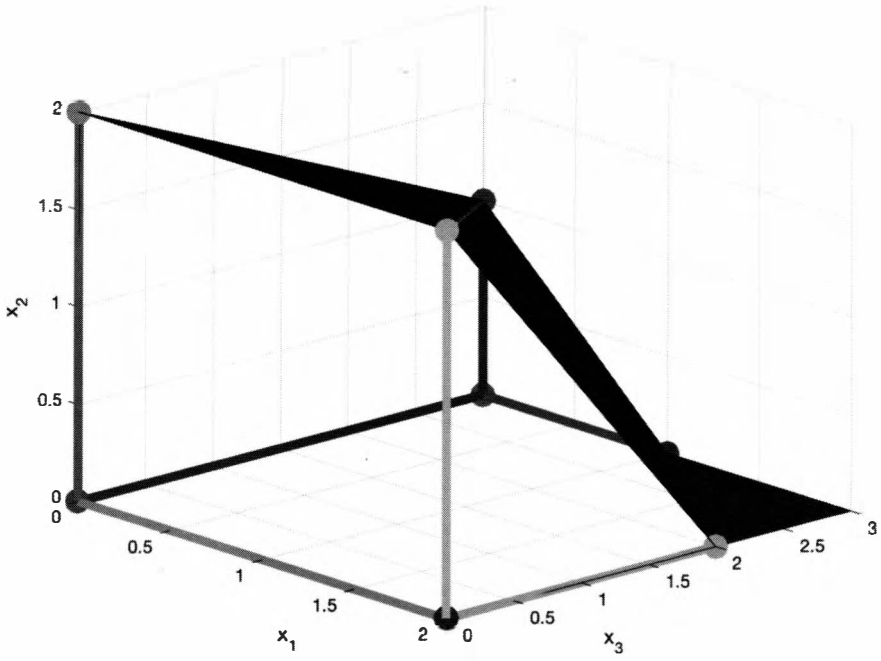


Рис. 10.7. Симплекс-метод движется от одного угла многогранника к другому, пока не найдет оптимальный угол

Шаги симплекс-метода

В случае задачи линейной оптимизации в стандартной форме симплекс-метод работает следующим образом:

1. Начинаем с угла многогранника (допустимое базисное решение $\vec{x}^*$). Как найти координаты этого допустимого базисного решения? Из **A** выбираем ***m*** линейно независимых столбцов $A_1, ..., A_m$. Записываем их в матрицу **B** (базисную матрицу). Решаем $\mathbf{B}\vec{x}_b = \vec{b}$ для $\vec{x}_b$. Если все $\vec{x}_b$ неотрицательны, значит, мы нашли допустимое базисное решение $\vec{x}^*$. Разместим записи $\vec{x}_b$ в соответствующих позициях в $\vec{x}^*$, а остальные сделаем нулевыми. В качестве альтернативного варианта, можно решить $\mathbf{A}\vec{x} = \vec{b}$ для $\vec{x}$, где соответствующие выбранным столбцам записи являются неизвестными, а остальные — нулями. Таким образом, допустимое базисное решение $\vec{x}^*$ имеет нулевые небазисные координаты и базисные координаты $\vec{x}_b = \mathbf{B}^{-1}\vec{b}$. Например, если

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 6 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ и } \vec{b} = \begin{pmatrix} 8 \\ 12 \\ 4 \\ 6 \end{pmatrix},$$

то можно выбрать A_4, A_5, A_6, A_7 в качестве набора базисных столбцов, дающих $\vec{x} = (0, 0, 0, 8, 12, 4, 6)^T$ в качестве допустимого базисного решения (координаты одной вершины многогранника). Можно выбрать A_3, A_5, A_6, A_7 в качестве другого набора базисных столбцов, дающего $\vec{x} = (0, 0, 4, 0, -12, 4, 6)^T$ в качестве базисного решения, но только *не* допустимого базисного, т. к. оно имеет отрицательную координату.

2. Двигаемся из $\vec{x}^*$ в другой угол $\vec{y}^* = \vec{x}^* + \theta^* \vec{d}$. Нам нужно найти направление $\vec{d}$, при котором мы останемся в многограннике (допустимо), увеличим только одну небазисную переменную x_j от нуля до положительного числа, а остальные небазисные переменные останутся равными нулю. В то же время при движении от $\vec{x}^*$ к $\vec{y}^* = \vec{x}^* + \theta^* \vec{d}$ необходимо уменьшить значение целевой функции. То есть нужно, чтобы $\vec{c}.\vec{y}^* \le \vec{c}.\vec{x}^*$. При увеличении значения x_j от нуля до положительного числа разность целевой функции составит $\overline{c}_j = c_j - \vec{c}_b.\mathbf{B}^{-1}A_j$; следовательно, необходимо выбрать такую координату *j*, для которой эта величина будет отрицательной. Для того чтобы все это работало, координаты $\vec{d}$ в итоге равны $d_j = 1$ (т. к. мы ввели x_j), $d_i = 0$, если $i \ne j$ или если i небазисный, и $\vec{d}_b = -\mathbf{B}^{-1}A_j$. В результате значение θ^* будет:

$$\theta^* = \min_{\text{все базисные индексы, при которых } d_{\mathbf{B}(i)} < 0} \left\{ -\frac{x_{\mathbf{B}(i)}}{d_{\mathbf{B}(i)}} \right\} := -\frac{x_{\mathbf{B}(l)}}{d_{\mathbf{B}(l)}}.$$

3. Теперь из базиса **B** выбывает столбец $A_{\mathbf{B}(l)}$, и его заменяет столбец A_j.

4. Повторяем этот процесс до тех пор, пока либо не достигнем конечного оптимального решения (когда ни один A_j из всех доступных столбцов **A** не дает отрицательного c_j), либо не обнаружим, что задача не ограничена, а оптимальные затраты составляют $-\infty$. Такое происходит при $\vec{d} \ge \vec{0}$, так что $\vec{y} = \vec{x} + \theta\vec{d} \ge \vec{0}$, что делает задачу допустимой независимо от того, насколько велико значение θ, поэтому при увеличении θ до ∞ затраты $\vec{c}.\vec{y} = \vec{c}.\vec{x} + \theta\left(c_j - \vec{c}_B.\mathbf{B}^{-1}A_j\right)$ будут постоянно сокращаться, стремясь к $-\infty$.

Замечания относительно симплекс-метода

Рассмотрим некоторые моменты, которые следует запомнить по поводу симплекс-метода.

- Шаг 4 дает два критерия завершения симплекс-алгоритма: отсутствие отрицательных сокращенных издержек $\overline{c}_j$ или неотрицательность всех координат допустимого набора в направлении снижения издержек $\vec{d}$.

- Если допустимый набор непустой и каждое базовое решение невырожденное, то симплекс-метод гарантированно завершается после конечного числа итераций либо с конечным оптимальным решением, либо с оптимальными издержками $-\infty$.
- Предположим, что некоторые из допустимых базисных решений являются вырожденными (некоторые из *базисных переменных* также равны нулю), и мы попадем в одно из них. В этом случае у нас есть шанс, что, изменив базис, введя A_j и выведя $A_{\mathbf{B}(l)}$, мы останемся в том же углу $\vec{y} = \vec{x} + \theta\vec{d}$ (такое происходит при $A_{\mathbf{B}(l)} = 0$, так что $\theta^* = -\dfrac{x_{\mathbf{B}(l)}}{d_{\mathbf{B}(l)}} = 0$). В этом случае выбираем новый A_j до тех пор, пока не двинемся от $\vec{x}$ к $\vec{y} = \vec{x} + \theta\vec{d}$, $\theta^* > 0$. Единственное плохое, что может здесь произойти, — после того, как мы остановимся на $\vec{x}$ и продолжим менять базис (остановившись на какое-то время на $\vec{x}$), пока не найдем тот, который действительно переместит нас от $\vec{x}$ к $\vec{y} = \vec{x} + \theta\vec{d}$ в направлении сокращения издержек, можно оказаться с тем же базисом, с которого начинался алгоритм! Это создаст цикличность, и алгоритм может зациклиться до бесконечности. Можно избежать цикличности, рационально выбирая, какие столбцы **A** вводить и выводить из базиса — систематический метод выбора A_j, а затем $\mathbf{B}(l)$ в θ^* при наличии связей в процессе.
- При наличии связей в процессе (имеется более одного варианта снижения затрат A_j, который дает $\bar{c}_j < 0$, и/или более одного минимизирующего индекса $\mathbf{B}(l)$ для θ^*) можно разработать правила выбора ввода A_j и/или вывода $A_{\mathbf{B}(l)}$ на шаге с такой связью — *правила разворота*.
- Довольно простым и не требующим больших вычислительных затрат правилом разворота является *правило Блэнда*: выбираем A_j с наименьшим индексом j, при котором $\bar{c}_j < 0$ для ввода в базис и $A_{\mathbf{B}(l)}$ с наименьшим допустимым индексом $\mathbf{B}(l)$ для вывода из базиса. Это правило разворота наименьшего подстрочного индекса позволяет избежать цикличности. Существуют и другие правила разворота.
- Если $n - m = 2$ (т. е. в **A** количество столбцов превышает количество строк всего на два), то симплекс-метод не будет циклическим, независимо от того, какое правило разворота используется.
- В задачах, возникших не из задачи общего вида, особенно с большим числом переменных, не всегда очевидно, как выбрать исходный базис **B** и связанное с ним допустимое базисное решение x (т. к. не ясно, какие m столбцов матрицы **A** линейно независимы). В этом случае вводят *искусственные переменные* и решают *вспомогательную задачу линейного программирования*, чтобы определить,

насколько исходная задача *невыполнима* и, следовательно, не имеет решения; или, если задача выполнима, искусственные переменные выводятся из базиса, и получается исходный базис и связанное с ним допустимое базисное решение исходной задачи. Этот процесс называется *первой фазой* симплекс-метода. Оставшаяся часть симплекс-метода называется *второй фазой*.

- Метод Big M объединяет первую и вторую фазы симплекс-метода. Здесь симплекс-метод используется для решения:

$$\min_{\substack{A\vec{x}=\vec{b} \\ \vec{x}\geq 0,\vec{y}\geq 0}} \left(\vec{c}.\vec{x} + M\left(y_1 + y_2 + \ldots + y_m\right)\right).$$

Когда имеется достаточно большой выбор M, а исходная задача допустима, и ее оптимальная стоимость конечна, все искусственные переменные $y_1, y_2, \ldots, y_m$ в конечном итоге стремятся к нулю, что возвращает нас к исходной задаче. Можно рассматривать M как неопределенный параметр и считать сокращенные издержки функциями M, а при определении отрицательности сокращенных издержек рассматривать M как очень большое число.

Пересмотренный симплекс-метод

Пересмотренный симплекс-метод — это менее затратная в вычислительном отношении реализация симплекс-метода. Он обеспечивает более дешевый способ вычисления $\bar{\mathbf{B}}^{-1}$, используя связь между старым базисом $\mathbf{B}$ и новым базисом $\bar{\mathbf{B}}$, т. к. в этом случае они имеют только на один столбец больше (две вершины являются смежными). Таким образом, мы можем получить новый $\bar{\mathbf{B}}^{-1}$ из предыдущего $\mathbf{B}^{-1}$.

Ниже приведена типичная итерация пересмотренного симплекс-алгоритма. Это также поможет закрепить шаги симплекс-метода из предыдущего раздела. Важно отметить, что для простоты используются векторные обозначения x, y, b, d:

1. Начинаем с $\mathbf{B}$, состоящего из m базовых столбцов из $\mathbf{A}$ и соответствующего допустимого базисного решения x при $x_{\mathbf{B}} = \mathbf{B}^{-1}b$, и $x_i = 0$ в обратном случае.
2. Вычисляем $\mathbf{B}^{-1}$ (в вычислениях симплекс-метода фигурирует именно $\mathbf{B}^{-1}$, а не $\mathbf{B}$).
3. Для небазисных j вычисляем сокращенные издержки $\bar{c}_j = c_j - \vec{c}_{\mathbf{B}}.\mathbf{B}^{-1}A_j$ (это даст $n-m$ сокращенных издержек).
4. Если все $\bar{c}_j$ неотрицательны, то текущее допустимое базисное решение x является оптимальным, и алгоритм завершается значением x в качестве оптимизатора и $c.x$ в качестве оптимальных издержек (нет такого A_j, который можно было бы ввести в базис и сократить издержки еще больше).
5. В противном случае выбираем j, при котором $\bar{c}_j < 0$ (правило разворота Блэнда подсказывает выбрать минимальное j). Важно отметить, что в таком случае A_j войдет в базис.

6. Вычисляем допустимое направление d: $d_j = 1$, $d_{\mathbf{B}} = -\mathbf{B}^{-1}A_j$ и $d_i = 0$ в противном случае.

   - Если все компоненты $d_{\mathbf{B}}$ неотрицательны, то алгоритм завершается оптимальными издержками $-\infty$ без оптимизатора.
   - В противном случае выбираем отрицательные компоненты $\vec{d}_{\mathbf{B}}$, и пусть:

$$\theta^* = \min_{\text{все базисные индексы, при которых } d_{\mathbf{B}(i)}<0} \left\{-\frac{x_{\mathbf{B}(i)}}{d_{\mathbf{B}(i)}}\right\} := -\frac{x_{\mathbf{B}(l)}}{d_{\mathbf{B}(l)}}.$$

   На этом шаге вычисляется θ^* и назначается $\mathbf{B}(l)$ в качестве индекса выводимого столбца.

7. Вычисляем новое допустимое базисное решение $y = x + \theta^* d$ (это новое допустимое базисное решение соответствует новому базису $\overline{\mathbf{B}}$, в котором A_j заменяет $A_{\mathbf{B}(l)}$ в $\mathbf{B}$).

8. На этом шаге вычисляем новую матрицу $\overline{\mathbf{B}}^{-1}$ для следующей итерации без образования нового базиса $\overline{\mathbf{B}}$ и его последующей инверсии, составив расширенную матрицу $\mathbf{B}^{-1} \mid \mathbf{B}^{-1}A_j$ размера $m \times (m+1)$. Выполняем операции над строками при помощи l-й строки (добавляем к каждой строке число, кратное l-й строке) так, чтобы последний столбец стал единичным вектором $\mathbf{e}_l$, равным нулю везде, кроме 1 в l-й координате. Первые m полученных столбцов результата представляют собой новую матрицу $\overline{\mathbf{B}}^{-1}$.

   Обоснование.

   Пусть $u = \mathbf{B}^{-1}A_j$ и отметим, что $\mathbf{B}^{-1}\overline{\mathbf{B}} = (e_1 e_2 \ldots u \ldots e_m)$, где e_i — единичный вектор-столбец с 1 в l-й записи и нулем во всех остальных, а u — l-й столбец. Матрица станет матрицей тождественности, если мы выполним операции над строками при помощи l-й строки и преобразуем u в e_l. Все операции над строками можно объединить в обратимую матрицу $\mathbf{Q}$, которая применяется слева: $\mathbf{Q}\mathbf{B}^{-1}\overline{\mathbf{B}} = \mathbf{I}$. Теперь правую часть умножим на $\overline{\mathbf{B}}^{-1}$ и получим $\mathbf{Q}\mathbf{B}^{-1} = \overline{\mathbf{B}}^{-1}$. Это значит, что для того чтобы получить $\overline{\mathbf{B}}^{-1}$, нужно выполнить над $\mathbf{B}^{-1}$ такие же операции над строками, которые преобразуют u в e_l.

Вычисление $\overline{\mathbf{B}}^{-1}$ при помощи $\mathbf{B}^{-1}$ в пересмотренном симплекс-методе

Этот метод начинается не с исходных m столбцов матрицы $\mathbf{A}$ и поиска обратной величины; вместо этого он выполняет операции со строками над предварительно рассчитанным базисом $\mathbf{B}^{-1}$, что может повлечь за собой ошибки округления. В результате многих итераций эти ошибки будут накапливаться, и во избежание этого желательно время от времени вычислять $\overline{\mathbf{B}}^{-1}$ непосредственно из столбцов $\mathbf{A}$.

Полная реализация симплекс-метода в симплекс-таблице

Преимущество полной реализации в симплекс-таблице заключается в сохранении и обновлении только одной матрицы. Вместо $\mathbf{B}^{-1}$ здесь сохраняется и обновляется матрица $x_{\mathbf{B}}\left|\mathbf{B}^{-1}\mathbf{A}\right)=(\mathbf{B}^{-1}b\left|\mathbf{B}^{-1}\mathbf{A}\right.$ размера $m\times(n+1)$. Столбец $u=\mathbf{B}^{-1}A_j$, соответствующий вводимой в базис переменной, называется *разворотным столбцом*. Когда l-я базисная переменная выводится из базиса, l-я строка называется *разворотной*. Элемент, одновременно принадлежащий разворотной строке и разворотному столбцу, называется *разворотным элементом*. Теперь добавим в *симплекс-таблицу нулевую строку*, в которой будут учитываться отрицательные значения текущих издержек $-c.x=-c_{\mathbf{B}}.x_{\mathbf{B}}=c_{\mathbf{B}}.\mathbf{B}^{-1}b$ и сокращенных издержек $c-c_{\mathbf{B}}.\mathbf{B}^{-1}\mathbf{A}$. Итоговая симплекс-таблица будет иметь вид:

$$\begin{pmatrix} -c_{\mathbf{B}}.\mathbf{B}^{-1}b & c-c_{\mathbf{B}}.\mathbf{B}^{-1}\mathbf{A} \\ \mathbf{B}^{-1}b & \mathbf{B}^{-1}\mathbf{A} \end{pmatrix},$$

или в более развернутом виде:

$$\begin{pmatrix} -c_{\mathbf{B}}.x_{\mathbf{B}} & \overline{c}_1\cdots\overline{c}_n \\ x_{\mathbf{B}(1)} & |\ldots| \\ \vdots & \mathbf{B}^{-1}A_1\cdots\mathbf{B}^{-1}A_n \\ x_{\mathbf{B}(m)} & |\ldots| \end{pmatrix}.$$

Один из полезных навыков, который желательно освоить, — научиться легко извлекать $\mathbf{B}^{-1}$ и $\mathbf{B}$ из заданной симплекс-таблицы (как в фильме "Матрица").

Наиболее эффективной реализацией симплекс-метода считается *пересмотренный симплекс-метод* (использование памяти — $O(m^2)$, наихудшее время одной итерации — $O(mn)$, наилучшее время одной итерации — $O(m^2)$, причем все перечисленные показатели метода симплекс-таблиц составляют $O(mn)$), однако все зависит от разреженности матриц.

Транспортная задача и задача о назначении

Транспортная задача и задача о назначении — это задачи линейной оптимизации, которые можно сформулировать как *задачи о потоке минимальной стоимости*.

Транспортная задача.

Распределяет продукцию по складам, минимизирует издержки.

Задача о назначении.

Распределяет назначенцев по заданиям, при этом число назначенцев равно числу заданий, и каждый назначенец выполняет одно задание. Когда назначенец i выполняет задание j, возникают издержки. Задача состоит в выборе назначения, минимизирующего стоимость. В качестве примера можно привести назначение водителей Uber на заказчиков или машин на задания.

Благодаря тому, что мы используем разреженные матрицы, нам не придется выполнять полную реализацию симплекс-алгоритма, а только специальную *усиленную* версию, которая решает как транспортные задачи, так и задачи о назначении. Это относится к *сетевому симплекс-методу*, способному решать любую задачу о потоке минимальной стоимости, включая как транспортную задачу, так и задачу о назначении. Обе задачи представляют собой частный случай задачи о потоке минимальной стоимости. Для решения задачи о назначении был разработан так называемый *венгерский алгоритм*. Он специализируется исключительно на этой задаче и поэтому наиболее эффективен. Такие специальные алгоритмы входят в состав некоторых программных пакетов линейного программирования.

Двойственность, релаксация Лагранжа, теневые цены, максимин, минимакс и все такое

Ранее в этой главе мы уже затрагивали идею *двойственности* при обсуждении оптимизации с ограничениями конечной размерности и ослабления ограничений при помощи множителей Лагранжа. Двойственность оказывается действительно полезной в случаях линейных задач с ограничениями или квадратичных задач с линейными ограничениями. Она позволяет решить либо текущую задачу оптимизации (*первичную*), либо другую связанную с ней задачу (*двойственную*) в зависимости от того, что окажется проще или дешевле, и получить одинаковое решение. Как правило, наличие большего числа переменных решений (размерностей задачи) не столь обременительно для алгоритма, как наличие большего числа ограничений. Поскольку двойственная задача меняет местами роли переменных решений и ограничений, то решать ее вместо первичной задачи двойственным симплекс-методом будет целесообразнее при наличии слишком большого количества ограничений (еще один способ — воспользоваться методом, о котором мы скоро поговорим).

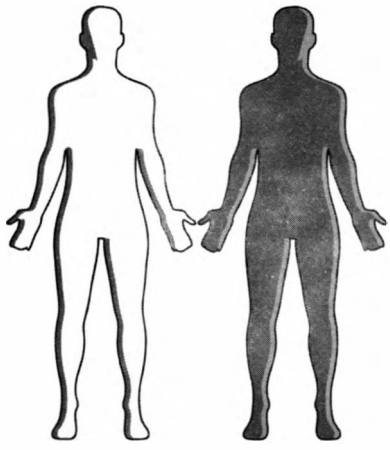


Рис. 10.8. Двойственность, теневые задачи, теневые цены

Кроме того, иногда двойственная задача обеспечивает кратчайшие пути к решению первичной задачи. Допустимый вектор $\vec{x}$ в первичной задаче станет в итоге оптимизатором при наличии в двойственной задаче такого допустимого вектора $\vec{p}$, при котором $\vec{c}.\vec{x} = \vec{p}.\vec{b}$.

Изучая двойственность в следующих разделах, можно представлять ее так же, как показано на рис. 10.8: что-то происходит в первичном мире, а некая связанная с этим тень, или эхо, происходит в двойственном мире (некой альтернативной вселенной), и оба эти явления встречаются в оптимизаторе, словно в переходе между двумя вселенными.

Таким образом, если мы максимизируем в одной вселенной, то минимизируем в другой; если мы делаем что-то с ограничениями в одной вселенной, то делаем что-то с переменными решения в другой, и наоборот.

Мотивация множителей Лагранжа в двойственной задаче

В любой задаче оптимизации (линейной или нелинейной):

$$\min_{\vec{x} \in \text{ допустимый набор}} f(\vec{x}).$$

Вместо поиска минимизатора $\vec{x}^*$, установив градиент равным нулю, ищем верхнюю границу $f(\vec{x}^*)$ (что несложно), подставив любой элемент допустимого набора в $f(\vec{x})$ и нижнюю границу $f(\vec{x}^*)$ (это неравенство сложнее и обычно требует оригинальных подходов). Получается нижняя граница $\leq f(\vec{x}^*) \leq$ верхняя граница, так что *сжимаем* эти границы, стремясь приблизиться к фактическому решению $f(\vec{x}^*)$. Сжатие границ осуществляется путем минимизации верхних границ (что возвращает нас к исходной задаче минимизации) и максимизации нижних границ (что ставит двойственную задачу).

Теперь в задаче линейной минимизации в любой форме (стандартной, общей или ни одной из них):

$$\min_{\text{линейные ограничения на } \vec{x}} \vec{c}.\vec{x}.$$

Какой уникальный подход позволит найти нижние границы для $f(\vec{x}) = \vec{c}.\vec{x}$? Нам нужно найти такие, которые будут составлены из ***линейной комбинации*** ограничений задачи. Поэтому умножаем каждое ограничение на множители p_i (множители Лагранжа), выбирая их знаки таким образом, чтобы неравенство ограничений выполнялось в направлении $\geq$. Что это значит? Итак, линейные ограничения являются линейными комбинациями записей $\vec{x}$, целевая функция $\vec{c}.\vec{x}$ также является линейной комбинацией записей $\vec{x}$, линейная комбинация линейной комбинации все еще остается линейной комбинацией, поэтому мы можем полностью выбрать линейную комбинацию ограничений, сравнимую с $\vec{c}.\vec{x}$.

В частности, при наличии ***m*** линейных ограничений нужно:

$$p_1 b_1 + p_2 b_2 + \ldots + p_m b_m \leq \vec{c}.\vec{x}.$$

Знак множителя p_i будет свободным, если ограничение содержит равенство. Получив нижние границы, сжимаем их, максимизируя по p_i, что приводит нас к двойственной задаче.

Вывод двойственной задачи линейной оптимизации из первичной задачи линейной оптимизации

В задаче линейной оптимизации важно правильно определить размеры исходных данных.

Исходными данными являются матрица **A** размера $m \times n$, вектор $\vec{c}$ размера $n \times 1$, вектор $\vec{b}$ размера $m \times 1$. Переменные решения в первичной задаче находятся в векторе $\vec{x}$ размера $n \times 1$. Переменные решения в двойственной задаче находятся в векторе $\vec{p}$ размера $m \times 1$.

В целом если в первичной задаче фигурирует **A**, то в двойственной задаче фигурирует $\mathbf{A}^T$. Таким образом, в первичной задаче имеется скалярное произведение *строк* **A** и $\vec{x}$. В двойственной задаче — скалярное произведение *столбцов* **A** и $\vec{p}$. Когда задача линейной оптимизации имеет произвольную форму, не составит труда написать ее двойственную задачу, придерживаясь следующего порядка.

- Если первичная задача заключается в минимизации, то ее двойник будет максимизацией, и наоборот.
- Первичная функция издержек — это $\vec{c}.\vec{x}$, а двойник — $\vec{p}.\vec{b}$.
- В первичной задаче *минимизации* ограничения делятся на два типа.

  Первый тип.

  Ограничения, указывающие на знак переменной решения, например:

  - $x_3 \geq 0$, тогда в двойственной задаче это будет с соответствовать $A_3.\vec{p} \leq c_3$, где A_3 — третий столбец **A**, c_3 — третья запись $\vec{c}$;
  - $x_{12} \leq 0$, тогда в двойственной задаче это будет соответствовать $A_{12}.p \geq c_{12}$, где A_{12} — двенадцатый столбец **A**, а c_{12} — двенадцатая запись $\vec{c}$;
  - x_5 свободно, т. е. не имеет определенного знака. Тогда в двойственной задаче это будет соответствовать $A_5.p = c_5$, где A_5 — пятый столбец **A**, а c_5 — пятая запись $\vec{c}$.

  Второй тип.

  Ограничения вида $a_i.x \geq \leq = b_i$, где a_i — это *i*-я строка **A**. В двойственной задаче они будут соответствовать ограничениям на знак p_i, например:

  - $a_2.x \geq b_2$, тогда в двойственной задаче это будет соответствовать $p_2 \geq 0$;
  - $a_7.x \leq b_7$, тогда в двойственной задаче это будет соответствовать $p_7 \leq 0$;
  - $a_8.x = b_8$, тогда знак p_8 будет свободным.

В частности, если задача линейной оптимизации имеет стандартный вид:

$$\min_{\substack{\mathbf{A}\vec{x} = \vec{b} \\ \vec{x} \geq \vec{0}}} \vec{c}.\vec{x},$$

то ее двойником будет:

$$\max_{\substack{\vec{p} \text{ свободно} \\ \mathbf{A}^T\vec{p} \leq \vec{c}}} \vec{p}.\vec{b}.$$

Если задача линейной оптимизации имеет общий вид:

$$\min_{\substack{\mathbf{A}\vec{x}=\vec{b} \\ \vec{x}\ \text{свободно}}} \vec{c}.\vec{x},$$

то ее двойником будет:

$$\max_{\substack{\vec{p}\geq\vec{0} \\ \mathbf{A}^T\vec{p}=\vec{c}}} \vec{p}.\vec{b}.$$

Как решить двойственную задачу? При помощи симплекс-метода, хотя теперь мы переходим к допустимым базисным решениям, которые скорее увеличивают, а не сокращают издержки.

Вывод для двойственной задачи линейной оптимизации в стандартной форме

Существует и другой способ вывести двойственную задачу, но для него линейная задача должна быть в стандартной форме. Суть в том, что мы ослабляем ограничение $\mathbf{A}\vec{x}=\vec{b}$, но вводим множители Лагранжа $\vec{p}$ (платим штраф $\vec{p}$, когда ограничение нарушается). Таким образом:

$$\min_{\substack{\mathbf{A}\vec{x}=\vec{b} \\ \vec{x}\geq\vec{0}}} \vec{c}.\vec{x}$$

становится

$$\min_{\vec{x}\geq\vec{0}} \vec{c}.\vec{x}+\vec{p}.\left(\vec{b}-\mathbf{A}\vec{x}\right)=g\left(\vec{p}\right).$$

Теперь докажем, что $g\left(\vec{p}\right)$ является нижней границей для исходной $\vec{c}.\vec{x}^*$ (теорема о слабой двойственности), а затем максимизируем по p. В процессе получаем двойственную задачу.

Теорема о сильной двойственности утверждает, что минимум первичной задачи и максимум двойственной задачи равны. Важно отметить, что если первичная задача не ограничена, то двойственная задача невыполнима, а если не ограничена двойственная задача, то первичная задача невыполнима.

В основе теории двойственности лежит лемма Фаркаша (**https://oreil.ly/yWF6R**), имеющая множество экономических и финансовых приложений.

Двойственный симплекс-метод

Двойственный симплекс-метод решает первичную задачу (но не двойственную) в рамках теории двойственности. Основное различие между симплекс-методом и двойственным симплекс-методом заключается в том, что обычный симплекс-метод начинает с допустимого базисного решения, которое не является оптимальным, и движется к оптимальности, а двойственный симплекс-метод начинает с недопустимого решения, являющегося оптимальным, и стремится к допустимости. Двойственный симплекс-метод представляет собой зеркальное отражение симплекс-метода.

В первую очередь отметим, что, решая первичную задачу симплекс-методом, мы бесплатно получаем оптимальную стоимость двойственной задачи (равную оптимальной стоимости первичной задачи), а также *можем вывести решение (оптимизатор) двойственной задачи из итогового табличного представления первичной задачи*. Оптимальная двойственная переменная будет ненулевой только в том случае, если связанное с ней ограничение в первичной задаче является *связывающим ограничением*. На интуитивном уровне это должно быть понятно, т. к. оптимальные двойственные переменные — это теневые цены (множители Лагранжа), связанные с ограничениями. Теневые цены можно интерпретировать как значения, присвоенные дефицитным ресурсам (связывающим ограничениям), так что стоимость этих ресурсов равна значению первичной целевой функции. Оптимальные двойственные переменные удовлетворяют условиям оптимальности симплекс-метода. В итоговой таблице симплекс-метода сокращенные издержки базовых переменных должны свестись к нулю. Оптимальные двойственные переменные должны быть теневыми ценами, связанными с оптимальным решением.

Еще можно рассматривать двойственный симплекс-метод как скрытый симплекс-метод, решающий двойственную задачу. Только в этом случае мы не пишем двойственную задачу в явном виде и не применяем симплекс-метод для максимизации.

Добавим, что обычный симплекс-метод выдает последовательность первичных допустимых базисных решений (углов многогранника), и как только находится двойственное допустимое решение, метод завершается. Двойственный симплекс-метод, в свою очередь, создает последовательность двойственных допустимых базисных решений, и как только находится первично выполнимое решение, метод завершается.

Пример: сети, линейная оптимизация и двойственность

Рассмотрим сеть, изображенную на рис. 10.9. Числа обозначают пропускную способность граней, т. е. максимальный объем потока, который может обработать каждая грань. Задача о *максимальном потоке* состоит в том, чтобы отправить максимальный поток из узла отправления в узел назначения. Очевидно, что максимальный поток в сети будет ограничен пропускной способностью ребер. По сути, это наблюдение лежит в основе двойственной задачи: максимизация потока через сеть эквивалентна минимизации суммарной емкости ребер, такой, что в случае разреза будет невозможно попасть из пункта отправления в пункт назначения. В этом заключается *теорема о максимальном потоке и минимальном разрезе*.

На рис. 10.9 показаны значения всех разрезов в сети (набор ребер, разрезав которые одновременно, мы не сможем добраться из пункта отправления в пункт назначения), а также разрез с минимальной суммарной пропускной способностью ребра, равной 16. Исходя из теоремы о максимальном потоке и минимальном разрезе, можно отправить через сеть максимальный поток, равный 16: отправляем $y_1 = 12$ единиц через ребро с пропускной способностью 19 и $y_2 = 4$ единицы через ребро с пропускной способностью 4. Из них $y_3 = 1$ единица пройдет через ребро с пропускной способностью 1, $y_4 = 11$ единиц — через ребро с пропускной способностью 11,

а $y_5 = 1 + 4 = 5$ единиц — через нижнее ребро с пропускной способностью 6. Все 12 единиц попадут в пункт назначения через 2 соединенных с ним последних ребра, причем $y_6 = 0$ (единицы не должны проходить через вертикальное ребро с пропускной способностью 7), $y_7 = 11$ единиц пройдут через крайнее правое ребро с пропускной способностью 12, а $y_8 = 5$ единиц пройдут через крайнее правое ребро с пропускной способностью 6. Тогда решением задачи максимального потока будет $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8) = (12, 4, 1, 11, 5, 0, 11, 5)$.

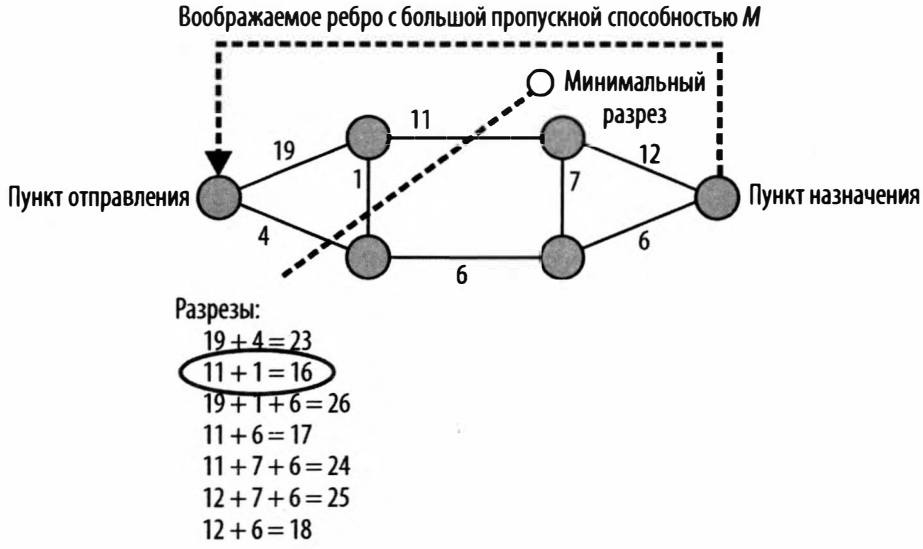


Рис. 10.9. Двойственность: максимальный поток в сети равен минимальной пропускной способности разреза

Для того чтобы сформулировать эту сетевую задачу как задачу линейной оптимизации (которую мы только что решили графически, узнав значение минимального разреза, представляющего собой решение двойственной задачи), необходимо добавить еще одно фиктивное ребро со значением потока y_9, соединяющее пункт назначения с пунктом отправления, предположив, что поток, который попадает в пункт назначения, фиктивно найдет путь обратно в пункт отправления. Другими словами, мы *замыкаем цепь* и применяем *закон токов Кирхгофа*, который гласит, что *сумма токов в сети проводников, сходящихся в одной точке, равна нулю*, или поток в узел равен потоку из него. Теперь линейная задача максимизации имеет такой вид:

$$\max_{\substack{A\vec{y} = \vec{0} \\ |y_i| \le M_i}} y_9,$$

где **A** (рис. 10.10) — матрица инцидентности сети, $\vec{y} = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9)^T$ — вектор максимальных потоков, имеющих знак (допускаются отрицательные значения y, чтобы входящий поток отменял выходя-

щий), которые можно направить через каждое ребро и которые нужно решить (его решение без знаков нашлось интуитивно при помощи минимального разреза), M_i — максимальная пропускная способность каждого ребра в сети, а условие $\mathbf{A}\vec{y} = \vec{0}$ гарантирует, что поток в узел равен потоку из узла. Безусловно, в этом случае в сети будут присутствовать направленные ребра, показывающие, в каком направлении пройдет оптимальный поток через каждое ребро.

A — матрица инцидентности

Узлы \ Ребра	1	2	3	4	5	6	7	8	9
1	1	1	0	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	0
3	0	1	1	0	1	0	0	0	0
4	0	0	0	1	0	1	1	0	0
5	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	1	1	1

Рис. 10.10. Матрица инцидентности сети, изображенной на рис. 10.9

Теперь, когда у нас имеется линейная формулировка задачи максимального потока, нам не составит труда с помощью рассмотренных в этой главе методов (задача о минимальном разрезе) записать двойственную задачу и решать либо ее, либо первичную задачу. Важно отметить, что для такой формулировки единственное, что потребуется, — это матрица инцидентности сети, емкость ребер и условие Кирхгофа, согласно которому поток в узел равен потоку из узла.

Пример: игра с нулевой суммой для двух игроков, линейная оптимизация и двойственность

Другой актуальной реализацией, в которую встроены двойственность и линейная оптимизация, является игра с нулевой суммой для двух игроков из теории игр. Выигрыш одного игрока оборачивается проигрышем другого (подсказка: двойственность). Для математической формулировки задачи нужна *платежная матрица* по всем вариантам игры для обоих игроков (игрок$_1$ и игрок$_2$). Каждый игрок стремится разработать стратегию, максимизирующую свое вознаграждение с учетом своих возможностей (опустим вопрос о том, что платежные матрицы в играх должны быть справедливыми). Необходимо найти оптимальную стратегию для каждого игрока. Если для игрока$_1$ определена задача оптимизации, то для стратегии игрока$_2$ уже не придется начинать с нуля, чтобы получить задачу оптимизации — мы просто запишем ее двойник. Сумма ожидаемого выигрыша одинакова для обоих игроков при условии, что они оба рационально следуют своим оптимальным стратегиям.

Рассмотрим, например, платежную матрицу на рис. 10.11. Порядок игры таков: игрок$_1$ выбирает строку, и в то же время игрок$_2$ выбирает столбец. Игрок$_1$ платит игроку$_2$ число из выбранной строки и столбца. Таким образом, игрок$_1$ стремится к минимуму, а игрок$_2$ — к максимуму. Игроки повторяют игру много раз.

Какая стратегия оптимальна для игрока$_1$ и игрока$_2$ и каков ожидаемый выигрыш от игры?

	y_1	y_2	y_3	
x_1	1	0	4	$y_1 + 4y_3$
x_2	3	−1	2	$3y_1 - y_2 + 2y_3$
	$x_1 + 3x_2$	$-x_2$	$4x_1 + 2x_2$	

Рис. 10.11. Платежная матрица

Для того чтобы найти оптимальную стратегию, предположим, что игрок$_1$ с вероятностью x_1 выбирает строку$_1$ и с вероятностью x_2 — строку$_2$. Тогда $x_1 + x_2 = 1$, $0 \le x_1 \le 1$, $0 \le x_2 \le 1$. Игрок$_1$ рассуждает, что, если они будут использовать смешанную стратегию (x_1, x_2), то в платежной матрице появится еще одна строка, соответствующая этой новой стратегии (см. рис. 10.11). Он знает, что игрок$_2$ хочет выбрать столбец, максимизирующий вознаграждение, поэтому игрок$_1$ должен выбрать (x_1, x_2), чтобы наихудшее вознаграждение (максимум третьей строки) было минимальным. Таким образом, игрок$_1$ должен решить *задачу о минимаксе*:

$$\min_{\substack{0 \le x_1 \le 1 \\ 0 \le x_2 \le 1 \\ x_1 + x_2 = 1}} \max\{x_1 + 3x_2, -x_2, 4x_1 + 2x_2\}.$$

Напомним, что максимум линейных функций — это выпуклая кусочно-линейная функция. Поэтому можно легко изменить задачу о *минимаксе (линейные функции)* на задачу линейной минимизации:

$$\min_{\substack{z \ge x_1 + 3x_2 \\ z \ge -x_2 \\ z \ge 4x_1 + 2x_2 \\ 0 \le x_1 \le 1 \\ 0 \le x_2 \le 1 \\ x_1 + x_2 = 1}} z.$$

На рис. 10.12 показана формулировка двойственной задачи, а на рис. 10.13 можно увидеть, что именно эту задачу пытается решить игрок$_2$. Важно отметить, что ограничения $y_1 \le 1$, $y_2 \le 1$, $y_3 \le 1$ являются избыточными, поскольку все y неотрицательны и дают в сумме 1. То же самое с ограничениями $x_1 \le 1$ и $x_2 \le 1$. Такое часто происходит при формулировке задач линейной оптимизации.

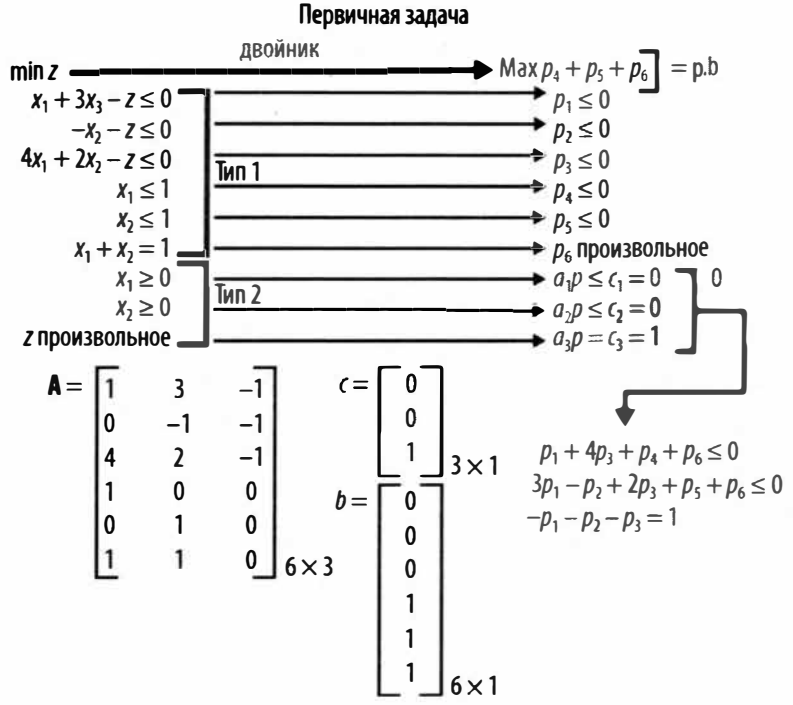


Рис. 10.12. Двойник задачи игрока$_1$

$\max \min \{y_1 + 4y_3,\ 3y_1 - y_2 + 2y_3\}$
$0 \le y_1, y_2, y_3 \le 1$
$y_1 + y_2 + y_3 = 1$

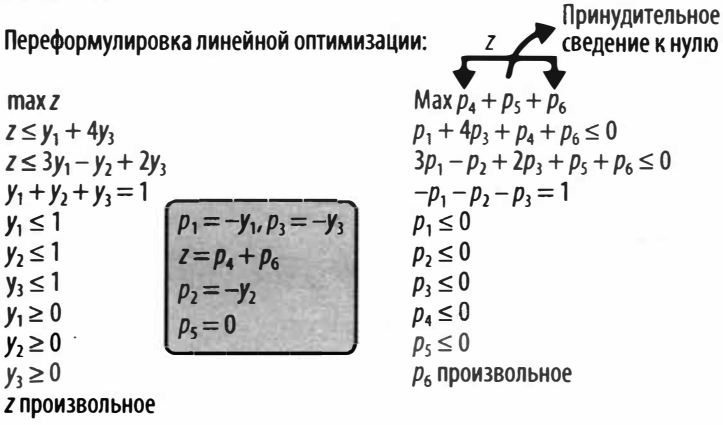


Рис. 10.13. Двойник задачи о минимаксе игрока$_1$ соответствует задаче о максимине игрока$_2$

Решив либо первичную, либо двойственную задачу, находим оптимальную стратегию каждого игрока: игрок$_1$ должен ходить с первой строки $x_1 = 0,25$ раз и со второй строки $x_2 = 0,75$ раз при ожидаемом выигрыше 2,5, что означает, что, следуя

такой стратегии, игрок$_1$ ожидает потерять не более 2,5. Игрок$_2$ должен ходить с первого столбца $y_1 = 0{,}5$ раз и с третьего столбца $y_3 = 0{,}5$ раз (и никогда со второго столбца $y_2 = 0$) при ожидаемом выигрыше 2,5, что означает, что, следуя такой стратегии, игрок$_2$ ожидает выиграть не менее 2,5.

Квадратичная оптимизация с линейными ограничениями, лагранжиан, теорема о минимаксе и двойственность

Задача нелинейной оптимизации, имеющая красивую структуру, встречается во всех видах приложений и может многому научить относительно того, как вещи связаны друг с другом, — это квадратичная задача с линейными ограничениями:

$$\min_{\mathbf{A}\vec{x} = \vec{b}} \frac{1}{2}\vec{x}^T\mathbf{S}\vec{x}.$$

При этом $\mathbf{S}$ — симметричная положительная полуопределенная матрица, что означает, что ее собственные значения неотрицательны. В случае больших размерностей это позволяет удерживать целевую функцию выпуклой и ограниченной снизу или в форме чаши одномерной функции $f(x) = x^2$.

Например, это может быть задача двумерной квадратичной оптимизации с одним линейным ограничением:

$$\min_{a_1x_1 + a_2x_2 = b} \frac{1}{2}\left(s_1x_1^2 + s_2x_2^2\right).$$

При этом $\mathbf{S} = \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix}$, где записи s неотрицательны, а $\mathbf{A} = \begin{pmatrix} a_1 & a_2 \end{pmatrix}$. Решая эту задачу, мы ищем точку (x_1, x_2) на прямой $a_1x_1 + a_2x_2 = b$, минимизирующую величину $f(\vec{x}) = s_1x_1^2 + s_2x_2^2$. Наборы уровней целевой функции $s_1x_1^2 + s_2x_2^2 = k$ — это концентрические эллипсы, охватывающие всю плоскость $\mathbb{R}^2$. Выигрышным эллипсом (с наименьшим значением набора уровней) считается тот, который касается прямой в выигрышной точке (рис. 10.14). В этой точке вектор градиента эллипса и вектор градиента ограничения совпадают, что соответствует формулировке множителя Лагранжа: сформулировать лагранжиан, ослабить ограничение, но заплатить штраф, равный множителю Лагранжа p, умноженному на то, насколько мы ослабили его в целевой функции, минимизируя задачу без ограничений.

$$\mathscr{L}(\vec{x};\, p) = f(\vec{x}) + p\left(b - g(\vec{x})\right) = s_1x_1^2 + s_2x_2^2 + p\left(b - a_1x_1 - a_2x_2\right).$$

Минимизируя лагранжиан, мы устанавливаем его градиент равным нулю, что приводит к $\nabla f(\vec{x}) = p\nabla g(\vec{x})$. Из этого следует, что вектор градиента целевой функции параллелен вектору градиента ограничения в точке (точках) оптимизации. Поскольку вектор градиента любой функции перпендикулярен ее набору уровней, ограничение, по сути, представляет собой касательную к набору уровней целевой функции в точке (точках) минимизации. Следовательно, для того чтобы найти точ-

ку (точки) оптимизации, нужно найти такие наборы уровней целевой функции, на которых она будет касательной к ограничению.

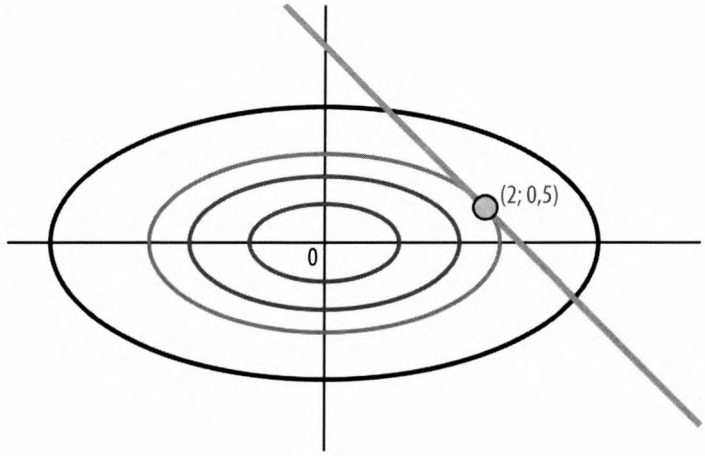


Рис. 10.14. Наборы уровней квадратичной функции $x_1^2 + 4x_2^2$ представляют собой концентрические эллипсы, каждый из которых имеет постоянное значение. При наложении линейного ограничения $x_1 + x_2 = 2,5$ получается оптимизатор (2; 0,5) именно в той точке, где один из наборов уровней проходит по касательной к ограничению. Значение оптимального набора уровней равно $x_1^2 + 4x_2^2 = 5$

Приведем другой пример, позволяющий наглядно представить лагранжиан и предстоящую теорему о минимаксе, — тривиальный одномерный пример:

$$\min_{x=1} x^2.$$

Лагранжиан имеет вид $\mathscr{L}(x; p) = x^2 - p(1 - x)$. Такой простейший пример, где, очевидно, оптимизатором является $x = 1$ с минимальным значением 1, приведен в целях наглядной демонстрации лагранжиана.

Напомним, что формулировка Лагранжа ведет к резкому повышению размерности; в данном случае у нас одно ограничение, поэтому размерность увеличивается с одного до двух, а в нашем ограниченном трехмерном мире мы можем визуализировать только функции двух переменных (x и p). На рис. 10.15 изображен ландшафт тривиальной функции Лагранжа, которая теперь представляет собой формулировки Лагранжа для задач квадратичной оптимизации с линейными ограничениями. Главное, на что следует обратить внимание на рис. 10.15, — оптимизаторы $(x^*; p^*)$ такого рода задач находятся в *седловых точках* лагранжиана. Это точки, в которых *вторая производная* положительна по одной переменной и отрицательна по другой, поэтому ландшафт функции Лагранжа имеет выпуклую форму по одной переменной (x) и вогнутую по другой (p).

Один из способов найти седловые точки лагранжиана (которые дают оптимизаторы соответствующих задач с ограничениями) — решить $\nabla\mathscr{L}(x; p) = \vec{0}$ для x и p, хотя

он считается грубым способом, который работает в случае простых (например, как рассматриваемая нами тривиальная задача) или небольших задач. Другой способ найти эти седловые точки заключается в минимизации по x и максимизации по p (рис. 10.16). Еще один способ — максимизировать по p, а затем минимизировать по x. Теорема о минимаксе утверждает, что эти способы одинаковы.

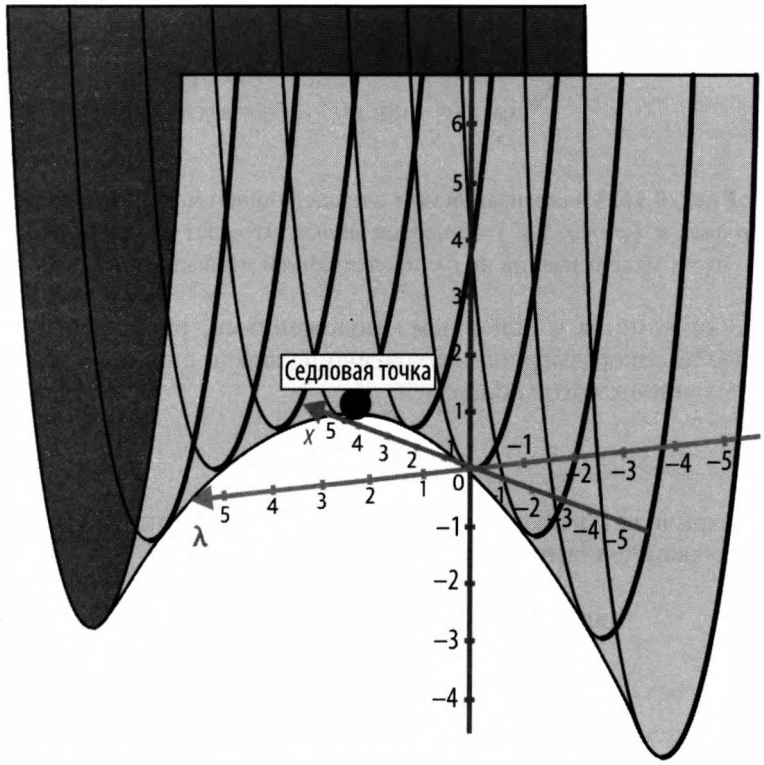


Рис. 10.15. Оптимизатор задачи с ограничениями находится в седловой точке лагранжиана (важно отметить, что минимум самого лагранжиана равен $-\infty$, однако это нас не интересует, потому что нам нужен оптимизатор квадратичной функции с линейными ограничениями)

Таким образом, в седловой точке $\left(x^*; p^*\right)$ мы имеем $\nabla \mathscr{L}\left(x; p\right) = 0$ (что соответствует $\frac{\partial \mathscr{L}\left(x; p\right)}{\partial x} = 0$ и $\frac{\partial \mathscr{L}\left(x; p\right)}{\partial p} = 0$) *и*

$$\min_{x} \max_{\substack{p, \\ \text{удержание } x \\ \text{фиксированным}}} \mathscr{L}\left(x; p\right) = \max_{p} \min_{\substack{x, \\ \text{удержание } p \\ \text{фиксированным}}} \mathscr{L}\left(x; p\right).$$

Мы прошли полный цикл и еще раз продемонстрировали, как, сопровождая задачу ограниченной минимизации по x, мы получаем другую задачу ограниченной максимизации по множителям Лагранжа p. Здесь в полной мере проявляется сочетание множителей Лагранжа, двойственности и оптимизации с ограничениями.

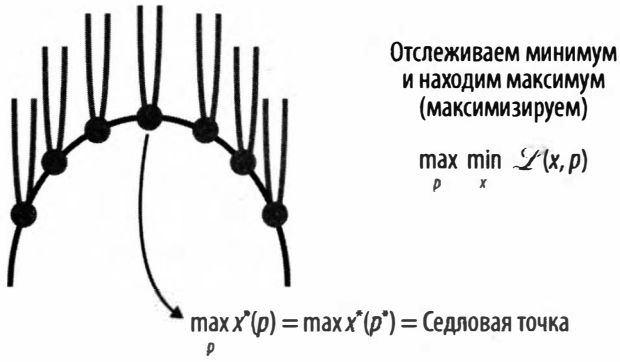


Рис. 10.16. Минимизация по x с последующей максимизацией по p дает $x^*(p) = x^*(p^*) = $ *седловая точка*. Это даст тот же результат, что и максимизация по p с последующей минимизацией по x

Теперь, после знакомства с основными положениями, рассмотрим их еще раз в контексте более высокоразмерной квадратичной задачи с линейными ограничениями, с которой начинается этот подраздел:

$$\min_{\mathbf{A}\vec{x}=\vec{b}} \frac{1}{2}\vec{x}^T\mathbf{S}\vec{x},$$

где **S** — симметричная положительно определенная матрица. Формула Лагранжа с *ослабленными* ограничениями имеет вид:

$$\min \mathscr{L}(\vec{x};\ \vec{p}) = \min \frac{1}{2}\vec{x}^T\mathbf{S}\vec{x} + \vec{p}.\left(\vec{b} - \mathbf{A}\vec{x}\right).$$

Решая эту задачу без ограничений — либо задав $\nabla\mathscr{L}(\vec{x};\ \vec{p}) = \vec{0}$, либо минимизируя по $\vec{x}$ с последующей максимизацией по $\vec{p}$, либо максимизируя по $\vec{p}$ с последующей минимизацией по $\vec{x}$, — мы получаем одинаковое решение $\left(\vec{x}^*;\ \overrightarrow{p^*}\right)$, находящееся в *седловой точке* нашего высокоразмерного лагранжиана, что и дает оптимальное значение целевой функции (преимущество этой задачи с простой структурой заключается в том, что ее можно решить вручную):

$$\text{минимальная стоимость } f = \frac{1}{2}\vec{b}.\left(\mathbf{A}\mathbf{S}^{-1}\mathbf{A}^T\right)^{-1}\vec{b}.$$

Более того, оптимальные *теневые цены* составят $\vec{p}^* = \dfrac{df}{d\vec{b}} = \left(\mathbf{A}\mathbf{S}^{-1}\mathbf{A}^T\right)^{-1}\vec{b}.$

Последнее, что необходимо узнать, — это характеристика седловых точек в более высоких измерениях. В одномерной задаче с ограничениями отличительной чертой было то, что вторая производная лагранжиана (который был функцией x и p) была отрицательной по одной переменной и положительной — по другой. Аналогично в случае высокой размерности собственные значения гессиана (матрицы вторых производных) отрицательны в одном наборе переменных и положительны в другом наборе переменных, так что в одном наборе переменных она будет вогнутой, а в

другом — выпуклой. Сказанное относится к оптимизации любой высокоразмерной целевой функции, выпуклой в одном наборе переменных и вогнутой — в другом. Это признак того, что ландшафт имеет седловые точки. В функции Лагранжа седловая точка расположена непосредственно там, где ограничение достигает своего минимума.

Распространяется ли это на задачи линейной оптимизации с линейными ограничениями, которые повсеместно встречаются в исследовании операций? Конечно, при наличии правильных знаков у всех коэффициентов в задаче, как в примерах с максимальным потоком и минимальным разрезом и игрой с нулевой суммой для двух игроков, которые мы рассматривали в предыдущих подразделах.

Чувствительность

В данном случае нас интересует чувствительность оптимизационной задачи и ее решения к изменениям входных данных. То есть, что произойдет с оптимальным решением $\vec{x}^*$ и оптимальной стоимостью $\vec{c}.\vec{x}^*$, если мы немного изменим $\vec{c}$, **A** или $\vec{b}$? Можно ли получить новое оптимальное решение из старого? При каких условиях это возможно? Рассмотрим несколько важных случаев, связанных с анализом чувствительности.

- Мы уже представляли оптимальное значение $\vec{p}$ в двойственной задаче как вектор предельных цен. Это связано с анализом чувствительности — скоростью изменения оптимальной стоимости относительно ограничения.
- Добавив новую переменную решения, мы проверяем ее приведенную стоимость, а в случае отрицательного значения добавляем в таблицу новый столбец и движемся дальше.
- Изменив запись $\vec{b}$ или $\vec{c}$ на δ, мы получим интервал значений δ, для которых оптимальным останется тот же базис.
- Аналогичный анализ можно провести, изменив запись **A** на δ. Правда, все немного усложнится, если изменение затронет запись базисного столбца.

В целом, если имеется функция и нужно узнать ее чувствительность к изменениям по одному из ее входов, то это равносильно вопросу о ее первой производной по этому входу (в определенном состоянии), или дискретной первой производной (конечной разности) в этом состоянии. Интерес к вопросам чувствительности объясняется тем, что здесь мы занимаемся *задачами с ограничениями*, проверяя влияние небольших изменений в вводных в задачу.

Теория игр и мультиагентные системы

Теория игр играет важную роль в экономике, политике, военных операциях, мультиагентном искусственном интеллекте и вообще в моделировании любой среды, где есть противники или конкуренты, и приходится принимать решения или разрабатывать стратегию в этих условиях. Наши оптимальные стратегии сильно зависят от стратегий наших противников, независимо от того, знаем ли мы их или только предполагаем.

Самый простой и хорошо понятный сценарий теории игр — это *игра с нулевой суммой для двух игроков*, которую мы рассматривали при обсуждении двойственности. В ней есть два конкурирующих субъекта, и проигрыш одного означает выигрыш другого, например две политические кампании или две конкурирующие фирмы. Распространение теории на более сложные реальные ситуации с множеством конкурентов, имеющих всевозможные преимущества и недостатки по отношению друг к другу, различную степень сотрудничества, а также множество взаимосвязанных стратегий, оказалось непростой задачей. По-прежнему существует разрыв между ситуациями, точно отраженными и проанализированными в теории, и ситуациями из реальной жизни. Тем не менее прогресс очевиден, и многие исследователи занимаются сегодня этим вопросом, учитывая невероятную пользу, которую принесла бы миру такая полная теория. Представьте, что мы можем взглянуть на всю сеть конкурентов сверху, наблюдая их перемещения, прослеживая связи, вероятные стратегии и их последствия.

В мультиагентных средах теория игр моделирует рациональное поведение или процесс принятия решений для каждого участвующего агента (игрока, фирмы, страны, армии, политической кампании и т. д.). В этом смысле теория игр для мультиагентных систем похожа на теорию принятия решений для одного агента.

Самым важным понятием теории некооперативных игр (где агенты принимают решения независимо друг от друга) является *равновесие Нэша* — план стратегии игры, при котором у каждого агента отсутствует стимул отклоняться от предписанной стратегии. То есть, если агент отклонится от стратегии, ему будет хуже, разумеется, при условии, что все участники поступают рационально.

Как показано в разделе о двойственности, игры с нулевой суммой для двух игроков можно моделировать как задачу о минимаксе и использовать *теорему о минимаксе*. Их также можно моделировать как задачу линейной оптимизации, где один игрок решает первичную задачу, а другой — двойственную. Это значит, что можно поставить задачу оптимизации либо для первого игрока, либо для второго. В итоге обе задачи будут иметь одно и то же решение. Здесь мы работаем с таблицей выплат по всем стратегиям игры обоих игроков, и задача состоит в том, чтобы найти комбинацию стратегий, максимизирующую выплаты (или минимизирующую потери) для каждого игрока. На интуитивном уровне можно понять, почему в эту задачу встроена двойственность. Два игрока противостоят друг другу, и оптимальная стратегия каждого из них решает как первичную, так и двойственную задачу.

При анализе игр для двух игроков можно воспользоваться графами и результатами из теории графов. Это похоже на то, как можно сформулировать максимальный поток через сеть в качестве задачи линейной оптимизации. В конечном счете в математике многие вещи четко связаны друг с другом, и когда эти связи раскрываются, мы испытываем небывалые ощущения.

В мультиагентных системах для принятия решений применяются определенные методы, в том числе процедуры голосования, аукционы для распределения дефицитных ресурсов, торги для достижения соглашений, протокол сети договоров для разделения задач. В *главе 13* рассматривается *дифференциальное уравнение Гамильтона — Якоби — Беллмана* в контексте математического моделирования мультиагентных игр. Здесь, чтобы найти оптимальную стратегию для каждого игрока,

необходимо решить высокоразмерное дифференциальное уравнение Гамильтона — Якоби — Беллмана для функции ценности игры. До появления глубокого обучения такие типы высокоразмерных дифференциальных уравнений были неразрешимыми, и приходилось делать много приближений или не учитывать всех участвующих в игре субъектов. Недавно (в 2018 году) для решения этих высокоразмерных дифференциальных уравнений была применена технология глубокого обучения (**https://oreil.ly/jCasX**), переформулированная в обратное стохастическое дифференциальное уравнение с граничными условиями (не беспокойтесь, если вы не знаете, что это такое, — это не важно для текущей главы).

С другой теоретической ситуацией состязательной игры с двумя игроками мы познакомились в книге раньше, когда рассматривали генеративно-состязательные сети в *главе 8*, посвященной вероятностным генеративным моделям.

Очередизация

Очереди встречаются повсеместно: вычислительные задания для машин, очереди на верфи, очереди в отделение скорой помощи, очереди на регистрацию в аэропорту, очереди в местном кафе "Старбакс". Хорошо продуманные системы очередей позволяют различным учреждениям и всей нашей экономике сберечь драгоценное время, силы и деньги, что в конечном счете ведет к повышению всеобщего благосостояния.

Математическое моделирование очередей нацелено на определение соответствующего уровня обслуживания для минимизации времени ожидания. Модель может включать дисциплину приоритетов, что означает наличие групп приоритетов, и порядок обслуживания участников зависит от их принадлежности к той или иной группе. В модель также могут входить различные виды обслуживания, происходящие последовательно или параллельно, либо какие-то последовательно, а другие параллельно (например, в судоремонтной мастерской). Некоторые модели содержат несколько объектов обслуживания — *сеть очередей*.

Существуют тысячи работ по теории очередей. Нам важно понять основные составляющие математической модели очередей.

- Участники очереди (клиенты, суда, рабочие места, пациенты) прибывают в определенный интервал времени. Если прибытие произошло случайно, то математическая модель должна определить распределение вероятности, которому соответствует время прибытия, либо на основе данных, либо на основе математических распределений, моделирующих такое время. В некоторых моделях заложены постоянные значения времени прибытия. Какие-то опираются на экспоненциальное распределение (марковский процесс), т. к. оно облегчает математический анализ и лучше имитирует реальный процесс. Еще одни модели используют *распределение Эрланга*, которое допускает различные экспоненциальные распределения для разных временны́х интервалов. Прочие полагаются на еще более общие распределения. Чем более общим будет распределение, тем сложнее математический анализ. А численное моделирование навсегда останется нашим верным помощником.

- Количество доступных серверов (параллельных и последовательных) — целое число.
- Время обслуживания также подчиняется определенному распределению вероятности, которое необходимо выбрать. Общие распределения аналогичны тем, что используются при определении временны́х интервалов прибытия.

Кроме того, математическая модель должна учитывать такие факторы, как:

- начальное число участников во всей системе очередей (ожидающих и обслуживаемых в данный момент);
- вероятность наличия *n* участников во всей системе очередей в заданный момент времени в будущем.

И наконец, модель рассчитывает *устойчивое состояние системы очередей*:

- вероятность наличия *n* участников во всей системе очередей;
- предполагаемое количество новых участников, прибывающих в единицу времени;
- предполагаемое количество участников, завершающих обслуживание в единицу времени;
- предполагаемое время ожидания для каждого участника в системе.

Участники с определенной средней скоростью встают в очередь, ожидают обслуживания, с определенной средней скоростью получают обслуживание, затем покидают учреждение. Математическая модель должна определить количество этих факторов и сбалансировать их.

Управление запасами

Сегодня симптомами нарушений цепочек поставок выступают пустые полки в продуктовых магазинах, нехватка автозапчастей, новых автомобилей, ремонтно-строительных материалов и многое другое. Очевиден разрыв между спросом и предложением. Увеличивается временной интервал между пополнением складских запасов, что ведет к задержкам, низкой производительности и всеобщему замедлению экономики. Математические модели управления запасами позволяют количественно определить (стохастически или детерминировано) спрос и предложение, а также разработать оптимальную политику управления запасами с учетом времени пополнения запасов и количества, требуемого при каждом пополнении. В идеале модель должна иметь доступ к системе обработки информации, которая собирает данные о текущем уровне запасов, а затем сообщает, когда и на сколько их нужно пополнить.

Машинное обучение в исследовании операций

Начнем с того, что по сравнению с тем, что было 10 лет назад, в настоящее время глубочайший интерес вызывает возможность решать массивные задачи исследования операций, порой содержащие десятки миллионов ограничений и переменных решений. Во многом это стало возможным вследствие роста вычислительных мощ-

ностей и постоянного совершенствования компьютерных реализаций алгоритмов исследования операций.

Более того, используя объемы доступных данных, машинное обучение позволяет предсказать значения множества параметров, входящих в модели исследования операций. Если какие-то параметры измерить не удавалось, разработчикам приходилось либо исключать их из модели, либо делать предположения об их значениях. Сегодня этого не требуется благодаря более точным моделям машинного обучения, способным учитывать тысячи переменных.

И наконец, машинное обучение позволяет ускорить поиск в комбинаторно больших пространствах поиска, *выучив*, на каких областях пространства следует сосредоточиться или какие подзадачи назначить приоритетными. Именно об этом идет речь в статье "Обучение делегированию полномочий в широкомасштабной маршрутизации транспортных средств" (Ли С. и др., 2021; **https://oreil.ly/RZdmR**), где маршрутизация транспортных средств ускорена в 10–100 раз по сравнению с самыми современными алгоритмами маршрутизации.

В настоящее время аналогичные исследования на стыке машинного обучения и исследования операций продолжают свое бурное развитие, демонстрируя колоссальный прогресс и масштабируемые решения. В перечне тезисов конференции "Исследование операций и машинное обучение" (**https://oreil.ly/qaoTi**) представлено множество актуальных проектов, например синтез и обработка данных, поступающих в реальном времени с датчиков в мусорных контейнерах (отслеживание объема) с целью повышения эффективности работы по сбору мусора (поскольку это опирается на данные в реальном времени, команда полагается на динамическую маршрутизацию). Еще один отличный пример — система совместного использования велосипедов (система проката), где задача состоит в том, чтобы предсказать количество велосипедов, необходимых в каждой точке проката, и назначить команды для эффективного распределения необходимого количества велосипедов. Приведем выдержку из тезиса:

> "Операторы диспетчерской системы совместного использования велосипедов постоянно перераспределяют велосипеды туда, где они вероятнее всего понадобятся. Для этого требуется понимание оптимального количества велосипедов, необходимых в каждой точке проката, и наиболее эффективного способа распределения команд для перемещения велосипедов. Расчет оптимального количества велосипедов в каждой точке в любой момент времени, а также планирование эффективных маршрутов для соответствующего перераспределения велосипедов осуществляются при помощи системы прогнозирования и оптимизации принятия решений. Решение, представленное компаниями DecisionBrain и IBM для системы совместного использования велосипедов в Лондоне, является первым в своем роде приложением, использующим оптимизацию и машинное обучение для решения проблем управления запасами, распределения и обслуживания велопроката, и может быть легко внедрено в другие системы совместного использования велосипедов по всему миру".

И действительно, разработки DecisionBrain (**https://decisionbrain.com/**) заслуживают внимания и осмысления.

В настоящее время наша команда работает над проблемой, связанной с департаментом общественного транспорта в моем городе. Это идеальный случай, когда машинное обучение пересекается с исследованием операций. Используя исторические данные о пассажирах, в частности о ежедневных посадках и высадках на каждой автобусной остановке в городе, а также данные о плотности населения, демографии, уязвимости, зонировании города, владельцах автомобилей и наличии у них высшего образования, данных о парковках, с помощью нейросетей мы прогнозируем спрос и предложение на каждой остановке.

Затем на основе этих данных мы используем оптимальный проект сети из исследования операций для перепроектирования автобусных маршрутов так, чтобы в полной мере обеспечить эффективное обслуживание автобусных остановок, особенно в наиболее социально уязвимых районах города.

Уравнение Гамильтона — Якоби — Беллмана

Области исследования операций, теории игр и дифференциальных уравнений пересекаются в динамическом программировании и дифференциальном *уравнении Гамильтона — Якоби — Беллмана*. Математик Ричард Беллман (1920–1984) впервые ввел термин *"проклятие размерности"* в контексте динамического программирования. Проклятие размерности ограничивало реальные приложения этого ценного уравнения и не позволяло учитывать всех игроков в игре (конкурирующие маркеры, страны, вооруженные силы) и решать их оптимальные стратегии, а также тысячи переменных, которые могут быть задействованы в задачах исследования операций, например в задачах оптимального распределения ресурсов. С развитием глубокого обучения ситуация изменилась. В статье "Решение высокоразмерных дифференциальных уравнений с помощью глубокого обучения" (Хан Ц. и др., 2018; **https://oreil.ly/nBbE9**) представлен метод решения этого и других уравнений для очень высоких размерностей. Мы обсудим идею авторов в *главе 13*, посвященной искусственному интеллекту и дифференциальным уравнениям.

Исследование операций в искусственном интеллекте

Исследование операций — это наука о принятии решений на основе оптимальных решений. Люди всегда стараются принимать решения, исходя из имеющихся обстоятельств. ИИ стремится повторить все аспекты человеческого интеллекта, включая принятие решений. В этом смысле в него автоматически укладываются методы принятия решений, используемые в исследовании операций. На протяжении десятилетий параллельно с ИИ развивались идеи динамического программирования, цепей Маркова, оптимального управления и уравнения Гамильтона — Якоби — Беллмана, достижения теории игр и мультиагентных игр, сетевой оптимизации и др. Сегодня многие стартапы позиционируют себя как компании, занимающиеся искусственным интеллектом, хотя на самом деле они занимаются старым добрым (и замечательным) исследованием операций.

Итоги и перспективы

Исследование операций — это область принятия наилучших решений с учетом текущих знаний и обстоятельств. Она всегда сводится к нахождению продуманных способов поиска оптимизаторов в очень высокоразмерных пространствах.

На протяжении всей книги звучит тема проклятия размерности, и усилия исследователей направлены на поиск путей его преодоления. Ни в одной из областей проклятие размерности не проявляется так широко, как в исследовании операций. Здесь пространства поиска комбинаторно растут с числом игроков в конкретной задаче, например число городов на маршруте, число конкурирующих организаций, число людей, число товаров и т. д. И хотя существуют довольно мощные точные эвристические методы, в плане скорости и масштаба еще есть куда стремиться.

Машинное обучение, в частности глубокое обучение, позволяет обучаться на основе предварительно решенных задач, размеченных или смоделированных данных. Благодаря этому можно ускорить оптимизационные процессы, если нам удастся определить узкие места и сформулировать источник узкого места как задачу машинного обучения. Например, узким местом может стать такая ситуация: *нам нужно решить слишком много подзадач, и мы не знаем, какой из них отдать предпочтение, чтобы быстро приблизиться к оптимуму*. Для решения этой задачи при помощи машинного обучения нам потребуется набор данных с уже решенными задачами и подзадачами, и модель машинного обучения должна обучиться, каким подзадачам следует отдать приоритет. Как только модель обучится этому, ее можно использовать для ускорения решения новых задач.

Другие варианты использования машинного обучения в исследовании операций предполагают *традиционный* тип машинного обучения — прогнозирование спроса на основе имеющихся данных, доступных в режиме реального времени или исторических, а затем использование исследования операций для оптимизации распределения ресурсов. Здесь машинное обучение помогает делать более точные прогнозы, тем самым повышает эффективность и сокращает потери.

В этой главе мы сделали широкий обзор области исследования операций и наиболее важных типов задач. Особое внимание было уделено линейной оптимизации, сетям и двойственности. Для решения многих полезных задач существуют мощные программные пакеты. Надеемся, что эти пакеты будут продолжать интегрировать последние достижения в этой области.

Как правило, на вводных занятиях по исследованию операций две темы обходятся стороной — это дифференциальные уравнения Гамильтона — Якоби — Беллмана для оптимального управления и стратегий многопользовательских игр, а также оптимизация *функционалов* с помощью вариационного исчисления. Их принято считать углубленными в области дифференциальных уравнений.

Однако мы не оставили их без внимания, т. к. они естественным образом связаны с оптимизацией и исследованием операций. Более того, рассматривая их в таком контексте, мы тем самым проясняем соответствующие области.

В процессе исследования операций и оптимизации с целью снижения издержек, увеличения доходов, эффективности использования времени и т. д. важно, чтобы

оптимизационные модели не игнорировали человеческий фактор. Если на выходе модели планирования ради достижения *в срок* нужной производительности компании из-за нестабильных графиков ухудшается жизнь низкооплачиваемых работников, то такая модель не будет считаться успешной, и нужно количественно оценить качество жизни и средств к существованию работников, на которых полагается компания, а затем учесть его в модели. Да, необходимо количественно определить *качество жизни*, поскольку все остальное уже учтено, и мы не можем оставить это без внимания. Компании, где работают сотни тысяч низкооплачиваемых работников, несут ответственность за то, чтобы алгоритмы исследования операций не загоняли работников в ловушку бедности.

В заключение этой главы приведем выдержку из работы Чарльза Хитча "Неопределенности в исследовании операций" (**https://oreil.ly/yPTcj**), опубликованной в 1960 году. Читая ее (в скобках — мои замечания), невозможно не задуматься о том, как далеко продвинулась область исследования операций с 1960 года:

> "Ни одна другая характеристика принятия решений не является столь распространенной, как неопределенность. Когда для упрощения первого шага в анализе мы, исследователи операций, предполагаем, что ситуацию можно описать эквивалентами определенности, мы совершаем насилие над фактами, и в действительности такое насилие может оказаться настолько серьезным, что приведет к искажению задачи и бессмысленному решению. Как, например, можно помочь военным принимать решения о том, какие самолеты или ракеты разрабатывать, когда суть проблемы заключается в том, что невозможно с точностью предсказать, сколько времени потребуется на разработку любого из конкурирующих видов вооружения или на введение их в эксплуатацию, сколько они будут стоить, какие у них будут характеристики или каким будет мир в любую неопределенную будущую дату (если, конечно, мир еще будет существовать тогда)? И я не преувеличиваю, когда говорю: „Невозможно с точностью предсказать". Мы видим, что обычно, например, производственные затраты на новое оборудование недооцениваются на ранних стадиях разработки в 2–20 раз (не в 2–20 процентов, а в 2–20 раз). Причину, почему они всегда недооцениваются и никогда не переоцениваются, я оставляю вашему богатому воображению. [...] Часто в задачах, особенно связанных с исследованиями и разработками, [исследователь операций] может выяснять критические неопределенности и рекомендовать стратегии их снижения — покупать информацию. Если вы не знаете, какая из двух отличающихся друг от друга технологий наведения ракет окажется лучше, то лучше всего порекомендовать вам следующее: оставьте их обе в разработке еще на некоторое время и выбирайте тогда, когда появится больше и лучше информации. Не обращайте внимания на тех, кто упрекнет вас в нерешительности. Можно доказать, что подобная нерешительность может сэкономить и деньги, и время. Конечно, вы не можете позволить себе попробовать все. Не хватит бюджета, недостаточно ресурсов. Вспомните, что мы часто говорили: „Если дать военным все, что они просят, они попытаются соорудить укрепрайон на Луне". (На самом деле именно вследствие ограниченности ресурсов так важны исследования и исследователи операций.) У нас не было бы проблем, если бы не ограниченность ресурсов. Наша работа и возможность как раз и заключаются в том, чтобы в целях снижения неопределенности найти или изобрести в рамках ограничений некую модель адаптации к неопределенному миру, лучше той, которую нашли бы наши коллеги без нас; или же найти какой-то лучший способ приобретения информации с учетом всех выгод и издержек".

ГЛАВА 11
Вероятность

Возможно ли на что-то полагаться, если миром правит случайность?

— Хала

Теория вероятностей представляет собой один из самых занимательных предметов математики, переносящий нас из стохастической в детерминированную область и обратно, — то, что представлялось чудесами, на деле оказалось магией математики. Вероятность дает систематический способ количественно оценить случайность, контролировать неопределенность, а также применять логику и рассуждения в ситуациях, имеющих первостепенное значение в ИИ: когда информация и знания содержат неопределенность, и/или когда агент перемещается в непредсказуемой или частично наблюдаемой среде. В таких ситуациях агент вычисляет вероятности относительно ненаблюдаемых аспектов конкретной среды, а затем на основе этих вероятностей принимает решения.

Люди испытывают беспокойство в условиях неопределенности, но при этом сдержанно относятся к приближенным значениям и ожиданиям. Когда они просыпаются, они не знают точно, как сложится их день, и принимают решения по ходу дела. В отличие от детерминированных и полностью предопределенных истин и ложностей, вероятностная интеллектуальная машина оперирует в мире вероятностей.

На протяжении всей книги мы пользовались вероятностными терминами и методами по мере их появления и только тогда, когда в этом была необходимость. Благодаря этому мы поняли, что необходимо хорошо разбираться в совместных распределениях вероятностей (например, признаков данных), обусловленности, независимости, теореме Байеса и марковских процессах. Мы также понимаем, что можем вернуться в детерминированный мир, вычисляя средние значения и ожидания.

Одна из особенностей глав этой книги заключается в том, что для глубокого и всестороннего изучения каждой из них потребуется отдельная книга. Это как нельзя лучше относится к главе о вероятности, куда можно включить тысячи тем. Мне пришлось выбирать, и в выборе тем для этой главы я руководствовалась тремя критериями:

1. Что, имеющее отношение к вероятности, уже использовалось в книге.
2. Что меня больше всего смущало в теории вероятностей, когда я изучала ее в институте (например, зачем нам нужна теория мер при вычислении вероятностей).
3. Что еще нужно знать из теории вероятностей для приложений ИИ.

Когда в книге встречается вероятность?

Составим краткий список тех мест в книге, где мы использовали вероятность или стохастические методы. Можно считать этот список *базовой вероятностью* для ИИ. Важно отметить, что *априорные вероятности* являются безусловными, т. к. они предшествуют наблюдению данных или доказательств, в то время как *апостериорные вероятности* считаются условными, т. к. их значение обусловлено наблюдением соответствующих данных. Логично, что после получения новых данных и связанных с этим доказательств наша степень убежденности в чем-то меняется. Часто требуется найти совместное распределение вероятностей всех задействованных переменных, но оно, как правило, слишком велико, а информация, необходимая для его построения, не всегда доступна.

Итак, приведем список.

- При минимизации функции потерь детерминированных моделей машинного обучения (где функция обучения принимает неслучайные входные данные и выдает неслучайные выходные), таких как регрессия, машины опорных векторов, нейронные сети и т. д., для ускорения вычислений мы используем стохастический градиентный спуск и его варианты, случайно выбирая подмножество обучающих экземпляров данных на каждом шаге градиентного спуска вместо использования всего обучающего набора.
- В *главе 9*, посвященной графовым моделям, мы неоднократно использовали случайные блуждания по графам, реализуя их через взвешенную матрицу смежности графа.
- В *главе 10*, посвященной исследованию операций, появились конкретные распределения вероятностей, например распределения вероятностей времени прибытия и обслуживания клиентов в очереди.
- Также в *главе 10* появились динамическое принятие решений и марковские процессы, которые являются основополагающими в обучении с подкреплением в ИИ. Они снова появятся в текущей главе, а затем еще раз в *главе 13* в контексте уравнения Гамильтона — Якоби — Беллмана.
- В *главе 10* в игре с нулевой суммой для двух человек каждый игрок имел вероятность сделать определенный ход, и мы использовали ее для вычисления оптимальной стратегии игрока и ожидаемого выигрыша.
- Методы моделирования Монте-Карло представляют собой вычислительные алгоритмы, которые опираются на повторяющиеся случайные выборки для численного решения детерминированных задач. Пример таких методов мы рассмотрим в *главе 13*, посвященной искусственному интеллекту и уравнениям в частных производных (partial differential equations, PDE).
- Мы неоднократно упоминали универсальную теорему аппроксимации в контексте нейросетей, а в этой главе ссылаемся только на ее доказательство. Доказательство этой фундаментальной теоремы носит теоретический характер и дает приятный привкус теории мер и функционального анализа. Интуицию теоретико-мерной вероятности мы обсудим позже в этой главе.

- Вероятностные модели машинного обучения изучают не детерминированные функции признаков данных, а их совместное распределение вероятностей[1] $\text{Prob}(x_1, x_2, ..., x_n, y_{\text{цель}})$. Оно кодирует вероятность появления признаков в одно и то же время. Учитывая входные признаки данных $(x_1, x_2, ..., x_n)$, модель выводит не детерминированную функцию признаков $y_{\text{предсказание}} = f(x_1, x_2, ..., x_n)$, а условную вероятность целевой переменной $\text{Prob}(y_{\text{предсказание}} \mid x_1, x_2, ..., x_n)$.
- Мы использовали случайные величины и две самые важные связанные с ними величины — математическое ожидание (ожидаемое среднее значение случайной величины) и дисперсию (меру разброса вокруг среднего значения) — без формального определения. Мы дадим им определение в этой главе.
- Правило произведения, или цепное правило для вероятности, а именно:

$$\text{Prob}(x_1, x_2) = \text{Prob}(x_1 \mid x_2)\text{Prob}(x_2) = \text{Prob}(x_2 \mid x_1)\text{Prob}(x_1),$$

  или для более чем двух переменных, скажем трех, без потери общности:

$$\begin{aligned}\text{Prob}(x_1, x_2, x_3) &= \text{Prob}(x_1 \mid x_2, x_3)\text{Prob}(x_2, x_3) = \\ &= \text{Prob}(x_1 \mid x_2, x_3)\text{Prob}(x_2 \mid x_3)\text{Prob}(x_3).\end{aligned}$$

- Понятия независимости и условной независимости являются фундаментальными. Два события *независимы*, если наступление одного из них не влияет на вероятность наступления другого. Независимость рассматриваемых признаков значительно упрощает задачу. Она позволяет расчленить сложные совместные распределения многих переменных, свести их к простым произведениям меньшего числа переменных и сделать многие ранее трудновыполнимые вычисления более податливыми. Это значительно упрощает вероятностную интерпретацию мира. Обратите внимание на разницу между независимостью *только двух событий* ($\text{Prob}(x_1, x_2) = \text{Prob}(x_1)\text{Prob}(x_2)$) и независимостью *множества событий*, которая является сильным предположением, при котором каждое событие не зависит от любого пересечения других событий.
- В случае вероятностных генеративных моделей в *главе 8* мы предположили предварительное распределение вероятностей и пропустили его через нейросеть, настроив ее параметры.
- В обсуждении совместных и условных вероятностей крайне важна теорема Байеса. Она позволяет количественно оценить убеждения агента относительно доказательств. Она применяется во многих контекстах, которые сразу же иллюстрируют ее ценность, например:

$$\text{Prob}(\text{заболевание} \mid \text{симптомы}) = \frac{\text{Prob}(\text{заболевание} \mid \text{симптомы})\text{Prob}(\text{заболевание})}{\text{Prob}(\text{симптомы})}$$

[1] Здесь и далее Prob (от англ. *probability*) — это вероятность. — *Прим. ред.*

или

$$\text{Prob}(\text{цель} \mid \text{данные}) = \frac{\text{Prob}(\text{данные} \mid \text{цель})\,\text{Prob}(\text{цель})}{\text{Prob}(\text{данные})},$$

или

$$\text{Prob}(\text{цель} \mid \text{доказательство}) = \frac{\text{Prob}(\text{доказательство} \mid \text{цель})\,\text{Prob}(\text{цель})}{\text{Prob}(\text{доказательство})},$$

или

$$\text{Prob}(\text{причина} \mid \text{следствие}) = \frac{\text{Prob}(\text{следствие} \mid \text{причина})\,\text{Prob}(\text{причина})}{\text{Prob}(\text{следствие})}.$$

Следует отметить, что в последней формуле $\text{Prob}(\text{причина} \mid \text{следствие})$ выражает диагностическое направление, а $\text{Prob}(\text{следствие} \mid \text{причина})$ — *причинное* направление.

- Байесовские сети — это структуры данных, которые представляют зависимости между переменными. В этом случае мы обобщаем отношения переменных в виде направленного графа и используем его для определения таблиц условных вероятностей, которые требуется отслеживать и обновлять в свете новых данных — мы отслеживаем вероятность дочернего узла в зависимости от наблюдения за его родителями. Родителями узла считаются любые переменные, непосредственно влияющие на этот узел. В этом смысле байесовская сеть — это представление совместного распределения вероятностей с тем упрощением, что нам известно, как связаны между собой задействованные переменные (какие переменные являются родителями каких переменных):

$$\text{Prob}(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} \text{Prob}(x_i \mid \text{родители}(X_i)).$$

- В машинном обучении можно провести границу между регрессионными моделями и моделями классификации. В *главе 8*, посвященной вероятностным генеративным моделям, мы познакомились с популярной вероятностной моделью классификации — наивной байесовской моделью. Выражаясь языком причинно-следственных связей, наивное предположение состоит в том, что при наличии причины некоторые наблюдаемые множественные следствия являются независимыми, так что можно записать:

$$\begin{aligned}\text{Prob}(\text{причина} \mid \text{следствие}_1, \text{следствие}_2, \text{следствие}_3) = \\ = P(\text{причина})\,P(\text{следствие}_1 \mid \text{причина}) \times \\ \times\, P(\text{следствие}_2 \mid \text{причина}) \times \\ \times\, P(\text{следствие}_3 \mid \text{причина}).\end{aligned}$$

Когда эта формула применяется для классификации с учетом заданных признаков данных, *причиной* (cause) служит класс. Более того, можно нарисовать байе-

совскую сеть, представляющую такую настройку. Каузальная переменная[2] является родительским узлом, а все следствия — дочерними узлами, происходящими из одного родительского узла (рис. 11.1).

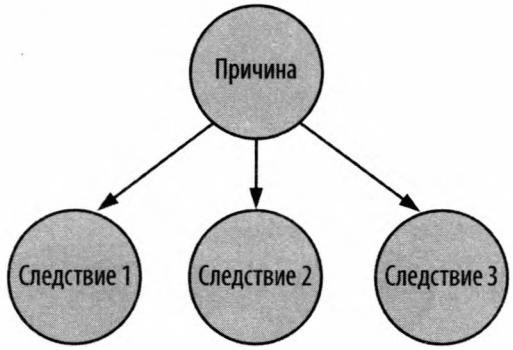


Рис. 11.1. Байесовская сеть, отображающая три следствия с одной общей причиной

Что еще важно знать в области искусственного интеллекта?

Нам понадобятся несколько дополнительных тем, которые либо не получили должного внимания в книге, либо упоминались вскользь и были перенесены в эту главу для более детального рассмотрения. К ним относятся:

- каузальное моделирование Джуды Перла и исчисление *do*;
- некоторые парадоксы;
- большие случайные матрицы и высокоразмерные вероятности;
- стохастические процессы, такие как случайные блуждания, броуновское движение и др.;
- марковские процессы принятия решений и обучение с подкреплением;
- теория вероятностей и ее применение в ИИ.

Этим темы мы рассмотрим далее в текущей главе.

Каузальное моделирование и исчисление *do*

В принципе, в байесовской сети стрелки между связанными переменными могут быть направлены в любую сторону. Все они в конечном итоге приводят к одному и тому же совместному распределению вероятности, хотя некоторые из них и более сложным способом, чем другие.

[2] Каузальная переменная — это причинная переменная (от англ. *cause* — причина). — *Прим. ред.*

Каузальные сети, напротив, представляют собой особые байесовские сети, в которых направленные ребра графа больше никуда не могут быть направлены, кроме как в сторону причины. В таких случаях приходится быть более внимательными при построении связей и их направлений. На рис. 11.2 показан пример *каузальной байесовской сети*.

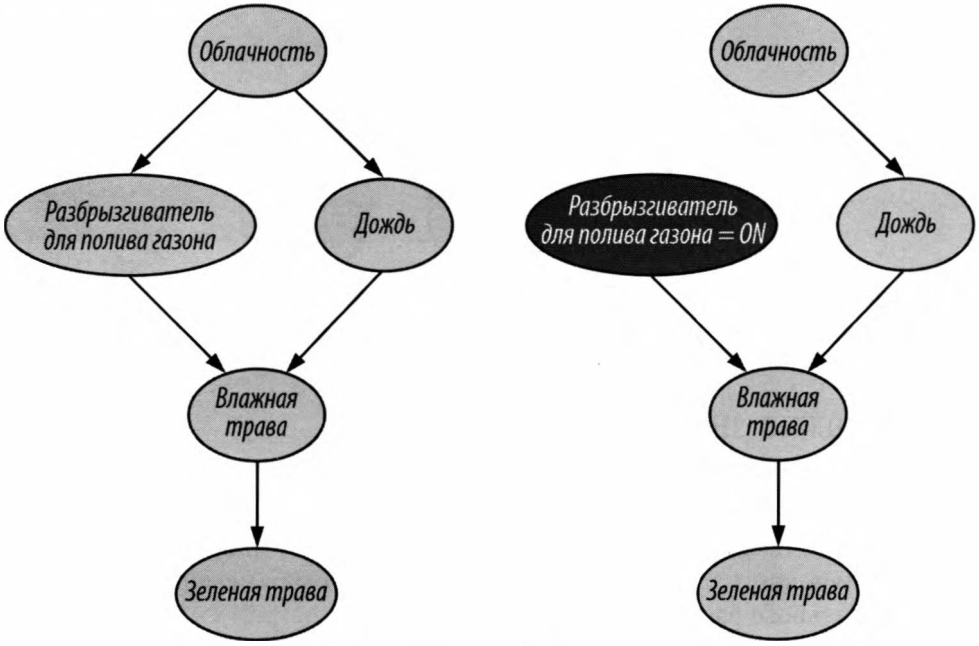


Рис. 11.2. Каузальная байесовская сеть

Важно отметить, что как байесовские, так и каузальные сети делают сильные предположения о том, какие переменные к каким переменным прислушиваются.

Агенты, наделенные способностью к причинно-следственным рассуждениям, с человеческой точки зрения, считаются более *высокофункциональными*, чем те, кто просто наблюдает за закономерностями в данных, а затем на их основе принимает решения.

Первостепенное значение имеет следующее различие.

- В байесовских сетях достаточно знать только то, являются ли две переменные вероятностно зависимыми. Являются ли огонь и дым вероятностно зависимыми?
- В каузальных сетях мы идем дальше и спрашиваем, *какая переменная* реагирует на *какую переменную*: дым на огонь (поэтому на диаграмме мы рисуем стрелку от огня к дыму) или огонь на дым (поэтому на диаграмме мы рисуем стрелку от дыма к огню)?

Здесь, чтобы количественно оценить эффект от фиксации значения одной переменной, требуется математическая основа для вмешательства. Это называется *исчислением do* (в отличие от статистического *наблюдения* и *подсчета*).

Приведем две базовые формулы *исчисления do*:

- формула корректировки;
- критерий *бэкдора*, или черного хода.

По словам изобретателя этого замечательного способа причинно-следственных рассуждений Джуды Перла (**https://oreil.ly/KB8Q8**, чья книга "Думай „почему?“. Причина и следствие как ключ к мышлению", вышедшая в 2020 году, лежит в основе всех обсуждений в этом и следующем разделах), они *позволяют исследователю изучить и наметить все возможные пути для вмешательства, какими бы извилистыми они ни были*, сокращают затраты и избавляют от проведения сложных рандомизированных контролируемых испытаний, даже если они физически осуществимы и юридически разрешены.

Альтернативный метод: исчисление *do*

В случае каузальной сети, которую мы строим на основе сочетания здравого смысла и экспертных знаний, при этом добавляя дополнительные неизвестные причины для каждой переменной, чтобы убедиться в том, что мы учитываем все, всеобъемлющей формулой будет совместное распределение вероятностей:

$$\text{Prob}(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} \text{Prob}(x_i \,|\text{родители}(X_i)).$$

Затем мы выполняем *вмешательство*, применяя $do(X_j = x^*)$. Это ведет к разрыву всех ребер, указывающих на X_j, и влияет на все условные вероятности потомков X_j, что ведет к новому совместному распределению вероятностей, которое уже не будет включать условную вероятность для вмешивающейся переменной. Мы уже установили ее значение $X_j = x^*$ с вероятностью 1, и любое другое значение будет иметь вероятность 0. На рис. 11.2 изображено, как при включении зазбрызгивателя все ведущие к нему стрелки в исходной сети обрываются.

Таким образом, мы имеем:

$$\text{Prob}_{\text{вмешательство}}(x_1, x_2, ..., x_n) = \begin{cases} \prod_{i \neq j}^{n} \text{Prob}(x_i \,|\text{родители}(X_i)), & \text{если } X_j = x^*, \\ 0 & \text{в противном случае.} \end{cases}$$

Формула корректировки

В действительности нас интересует, как $X_j = x^*$ влияет на вероятность каждой другой переменной в сети, и нужно вычислить эти значения из исходной сети без вмешательства. Другими словами, без оператора *do*, т. к. чтобы получить эти значения, можно не проводить новые эксперименты, а просто наблюдать за данными.

Для этого вводится *формула корректировки*, или *контроля над конфаундерами* (возможными общими причинами). Она представляет собой средневзвешенное значение влияния X_j и его родителей на X_i. Весовые коэффициенты — это априорные значения родителей:

$$\text{Prob}\left(x_i \mid do\left(X_j = x^*\right)\right) = \text{Prob}_{\text{вмешательство}\left(X_i = x_i\right)} =$$

$$= \sum_{\text{родители}\left(X_j\right)} \text{Prob}\left(x_i \mid x^*, \text{родители}\left(X_j\right)\right)\text{Prob}\left(\text{родители}\left(X_j\right)\right).$$

Важно отметить, что эта формула достигает нашей цели — устраняет оператор *do* и возвращает нас к поиску условных вероятностей путем наблюдения за данными вместо проведения дорогостоящих экспериментов с вмешательством или рандомизированных контролируемых испытаний.

Критерий бэкдора, или контроль над конфаундерами

В истории с каузальными диаграммами есть еще кое-что. Было бы неплохо узнать последствия вмешательства $do\left(X_j = x^*\right)$ на конкретную нижележащую переменную на диаграмме $X_{\text{вниз}}$.

Нам необходимо научиться обусловливать значения переменной в диаграмме, являющейся *другим прародителем*. Это также приведет вниз к интересующей нас переменной. В каузальном моделировании этот процесс называется *блокировкой "черных ходов"*, или *критерием бэкдора*:

$$P\left(x_{\text{вниз}} \mid do\left(X_j = x^*\right)\right) = P_{\text{вмешательство}}\left(X_{\text{вниз}} = x_{\text{вниз}}\right) =$$

$$= \sum_{\text{прародитель}\left(X_{\text{вниз}}\right)} P\left(x_{\text{вниз}} \mid x^*, \text{прародитель}\left(X_{\text{вниз}}\right)\right)P\left(\text{прародитель}\left(X_{\text{вниз}}\right)\right).$$

Контроль над конфаундерами

Самый распространенный способ, с помощью которого ученые и статистики могут предсказать последствия вмешательства, чтобы утверждать о каузальности, — это контроль над *возможными общими причинами*, или *конфаундерами*. На рис. 11.3 показана переменная Z в качестве конфаундера предполагаемой причинно-следственной связи между X и Y.

В целом это связано с тем, что конфаундинг является основным источником путаницы между простым наблюдением и вмешательством. Также он лежит в основе известного утверждения *"корреляция — это не каузальность"*. Именно здесь встречаются несколько странных и забавных примеров: высокая температура является конфаундером продаж мороженого и нападений акул (но зачем кому-то вообще изучать какую-то связь между мороженым и акулами?).

Критерий бэкдора и формула корректировки легко справляются с проблемой конфаундеров, препятствующих утверждать о каузальности.

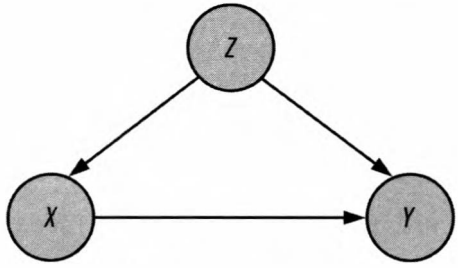


Рис. 11.3. *Z*-конфаундер предполагаемой причинно-следственной связи между *X* и *Y*

Мы пользуемся формулой корректировки для контроля конфаундеров, когда уверены, что у нас имеется достаточный набор переменных-*деконфаундеров* для того, чтобы заблокировать все "черные ходы" между вмешательством и результатом. Для этого мы оцениваем причинный эффект страта за стратой на основе данных, а затем рассчитываем средневзвешенное значение этих страт, где каждая страта взвешивается в соответствии с ее распространенностью в популяции.

Таким образом, без критерия бэкдора, статистики и ученые не могут гарантировать обоснованность какой-либо корректировки. Другими словами, критерий бэкдора, или "черного хода", позволяет удостовериться в том, что причинный эффект в каждой страте деконфаундера фактически является наблюдаемой тенденцией в этой страте.

О существовании других правил, устраняющих оператор *do*

Нам крайне нужны правила, способные перевести нас из выражения с оператором *do* (вмешательство) к выражению без оператора *do* (наблюдение), поскольку они избавляют нас от необходимости вмешательства. Они позволяют оценивать причинные эффекты путем простого наблюдения за данными. Именно это нам обеспечили формула корректировки и критерий бэкдора.

Есть ли еще какие-нибудь правила? Или, если спросить еще более масштабнее: существует ли способ заранее решить, насколько подходит конкретная каузальная модель для устранения оператора, чтобы понимать, достаточно ли будет предположений модели для выявления причинного эффекта из данных наблюдений без всякого вмешательства? Такое понимание имеет огромное значение! Например, если предположений модели недостаточно для устранения оператора *do*, то, как бы мы ни были умны, нам не избежать экспериментов с вмешательством. С другой стороны, если мы обойдемся без вмешательства и при этом оценим причинно-следственные эффекты, экономия окажется впечатляющей. Уже одно это стоит того, чтобы углубиться в вероятностное каузальное моделирование и исчисление *do*.

Для того чтобы понять суть исчисления *do* Джуды Перла, мы всегда начинаем с каузальной диаграммы и думаем о создании таких критериев, которые поспособствуют удалению ребер, направленных к интересующей нас переменной (переменным) или от нее.

Три правила Перла задают условия, при которых можно:

1. вставлять или удалять наблюдения:

$$\text{Prob}\left(y \mid do(x), z, w\right) = \text{Prob}\left(y \mid do(x), w\right);$$

2. вставлять или удалять вмешательства:

$$\text{Prob}\left(y \mid do(x), do(z), w\right) = \text{Prob}\left(y \mid do(x), w\right);$$

3. менять вмешательства на наблюдения:

$$\text{Prob}\left(y \mid do(x), do(z), w\right) = \text{Prob}\left(y \mid do(x), z, w\right).$$

Подробнее об исчислении *do* см. в статье Джуды Перла "Исчисление *Do* по-новому" (основная лекция, 17 августа 2012 г., **https://oreil.ly/DTPq0**).

Парадоксы и интерпретация диаграмм

Агенты ИИ должны уметь справляться с парадоксами. Мы все смотрели мультфильмы, в которых робот, когда его логика сталкивается с парадоксом, попадает в "сумасшедшую петлю" или даже подвергается физическому саморазрушению, разбрасывая во все стороны винтики и пружинки. Нам нельзя этого допустить. К тому же парадоксы часто возникают в чрезвычайно важных ситуациях, например в фармацевтике и медицине, так что нам крайне важно изучить их сквозь призму математики и скрупулезно разобраться в их загадках.

Рассмотрим три самых известных парадокса: *парадокс Монти Холла*, *парадокс Берксона* и *парадокс Симпсона* в свете диаграмм и каузальных моделей. Парадокс Монти Холла и парадокс Берксона вызывают путаницу из-за коллайдеров (двух независимых переменных, указывающих на третью), а парадоксы Симпсона — из-за конфаундеров (одной переменной, указывающей на две другие). Для того чтобы агент ИИ мог правильно рассуждать, он должен иметь такие диаграммы в составе структуры данных (или уметь их строить и корректировать).

Об этом замечательно высказывается Джуда Перл в своей книге "Думай „почему?“. Причина и следствие как ключ к мышлению":

> "Парадоксы отражают противоречия между каузацией и ассоциацией. Напряжение возникает вследствие того, что они стоят на двух разных ступенях лестницы причинности [наблюдение, вмешательство, опровержение фактов], и усугубляется тем, что человеческая интуиция оперирует причинно-следственной (каузальной) логикой, а данные подчиняются логике вероятностей и пропорций. Парадоксы возникают тогда, когда мы неправильно применяем правила, принятые в одной сфере, к совершенно другой".

Парадокс Монти Холла

Предположим, мы участвуем в игровом шоу и нам предлагают выбрать одну из трех дверей. За одной дверью находится автомобиль, за другими — козы. Мы вы-

бираем, скажем, дверь № 1, а ведущий, который знает, что находится за дверьми, открывает другую дверь, например № 3, за которой стоит коза. Он спрашивает нас, не хотим ли мы выбрать дверь № 2? Имеет ли смысл поменять дверь?

Ответ — да, нужно поменять двери, т. к. если мы не сделаем этого, вероятность получить автомобиль будет равна 1/3, а если мы поменяем дверь, она возрастет до 2/3! Главное, на что следует обратить внимание, — это то, что ведущий *знает*, где находится автомобиль, и выбирает дверь, за которой, как ему известно, автомобиля нет.

Так почему же вероятность выигрыша удваивается, когда мы меняем первоначальный выбор? Потому что ведущий предлагает новую информацию, которой можно воспользоваться, лишь отказавшись от первоначального выбора без этой информации.

В рамках стратегии не менять дверь.

- Если мы изначально выбрали выигрышную дверь (вероятность 1/3) и не поменяли ее, то мы выиграем.
- Если мы изначально выбрали проигрышную дверь (вероятность 2/3) и не поменяли ее, то мы проиграем.

Это означает, что в рамках стратегии не менять выбор мы выиграем только в 1/3 случаев.

В рамках стратегии поменять дверь.

- Если мы изначально выбрали выигрышную дверь (вероятность 1/3) и поменяли ее, то мы проиграем.
- Если мы изначально выбрали проигрышную дверь (вероятность 2/3), но поступает новая информация, *указывающая на другую проигрышную дверь*, и мы меняем выбор, то мы выиграем, потому что единственной оставшейся дверью будет выигрышная.

Это значит, что в рамках стратегии изменения выбора мы выиграем в 2/3 случаев.

Нарисовав диаграмму этой игры, как показано на рис. 11.4, мы понимаем, что дверь, которую собирается открыть ведущий, имеет двух указывающих на нее родителей — выбранная нами дверь и местоположение автомобиля.

Рис. 11.4. Каузальная диаграмма переменных, участвующих в парадоксе Монти Холла

Зависимость от коллайдера изменяет вероятности родителей. Между изначально независимыми родителями возникает ложная зависимость! Это похоже на то, как если бы мы изменили наши убеждения о генетических чертах родителей после того, как встретили кого-нибудь из их детей. Такие беспричинные корреляции возникают при зависимости от коллайдеров.

Теперь предположим, что ведущий выбирает свою дверь, *не зная*, выигрышная она или проигрышная. Тогда независимо от того, сменим ли мы дверь или оставим ее прежней, вероятность выигрыша автомобиля не изменится, т. к. в этом случае и у нас, и у ведущего будут равные шансы выиграть в 1/3 случаев и проиграть в 2/3 случаев. На диаграмме, составленной для этой абсолютно случайной игры без предварительной информации, нет никакой стрелки между местоположением автомобиля и дверью, которую собирается открыть ведущий, так что наш выбор двери и местоположение автомобиля *останутся независимыми* даже при наличии зависимости от выбора ведущего.

Парадокс Берксона

В 1946 году биостатистик из клиники Майо Джозеф Берксон обратил внимание на особенность обсервационных исследований, проводимых в больничных условиях — даже если в общей популяции два заболевания не связаны друг с другом, они могут оказаться связанными между собой у пациентов больницы. В 1979 году эксперт по всем видам статистической погрешности Дэвид Сакетт из Университета Макмастера предоставил убедительные доказательства реальности парадокса Берксона. В одном из примеров он изучал две группы заболеваний: респираторные и костные. Около 7,5% людей в общей популяции имеют болезни костей, и этот процент не зависит от наличия у них респираторных заболеваний. Но у госпитализированных людей с респираторными заболеваниями частота костных заболеваний возрастает до 25%! Сакетт назвал это явление "смещением показателя госпитализации", или "смещением Берксона".

Как и в случае с Монти Холлом, причиной возникновения парадокса Берксона является диаграмма коллайдера, на которой оба изначально независимых заболевания указывают на госпитализацию, причем вероятность госпитализации пациента с обеими болезнями гораздо выше, чем пациента только с одной из них. Когда мы ставим условием госпитализацию, являющуюся коллайдером, возникает случай беспричинной корреляции между изначально независимыми переменными. Так мы постепенно начинаем привыкать к смещению коллайдера.

Парадокс Симпсона

Представим себе парадокс, чей вывод, если предоставить его самому себе, будет выглядеть несколько абсурдно: *если мы знаем пол пациента, то не должны назначать лекарство, т. к. данные показывают, что лекарство одинаково вредно как для мужчин, так и для женщин; но когда пол неизвестен, мы должны назначить лекарство, т. к. данные показывают, что в целом лекарство подходит для общей*

популяции. Очевидно, что это нелепо, и в первую очередь нужно требовать предъявить данные!

Парадокс Симпсона возникает, когда в нескольких группах населения проявляется тенденция, но при объединении групп она исчезает или меняется на противоположную.

Для начала развенчаем сам парадокс. Речь идет о простой числовой ошибке в сложении дробей (или пропорций). В общих словах, когда выполняется сложение дробей, нельзя просто складывать соответствующие числители и знаменатели:

$$\frac{A}{B} > \frac{a}{b} \text{ и } \frac{C}{D} > \frac{c}{d} \not\Rightarrow \frac{A+C}{B+D} > \frac{a+c}{b+d}.$$

Предположим, например, что данные показывают:

- 3/40 женщин, принимавших препарат, перенесли сердечный приступ, по сравнению с 1/20 женщин, не принимавших препарат (3/40 > 1/20);
- 8/20 мужчин, принимавших препарат, перенесли сердечный приступ, по сравнению с 12/40 мужчинами, не принимавшими препарат (8/20 > 12/40).

После объединения данных по женщинам и мужчинам неравенство меняет направление: 3/40 > 1/20 и 8/20 > 12/40, но справедливо (3 + 8)/(40 + 20) < (1 + 12)/(20 + 40). Другими словами, из 60 мужчин и женщин, принимавших препарат, сердечный приступ случился у 11 человек, а из 60 мужчин и женщин, не принимавших препарат, сердечный приступ случился у 13 человек.

Однако, объединив данные таким образом, мы допустили простую ошибку с дробями. Для того чтобы решить парадокс Симпсона, нельзя объединять данные простым сложением числителей и знаменателей, ожидая выполнения неравенства. Важно отметить, что из 60 человек, принимавших препарат, 40 — женщины и 20 — мужчины, а из 60 человек, не принимавших препарат, 20 — женщины и 40 — мужчины. Мы сравниваем яблоки и апельсины и путаем это с полом. Пол влияет *как* на прием препарата, *так и* на возникновение сердечного приступа. Диаграмма на рис. 11.5 иллюстрирует эту взаимосвязь.

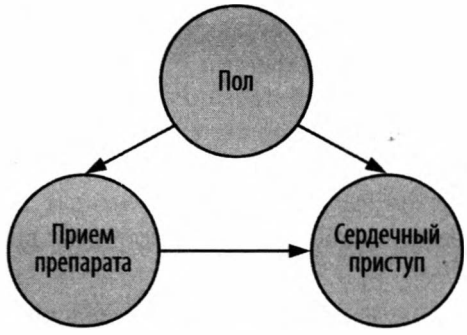


Рис. 11.5. Пол является конфаундером как для приема препарата, так и для сердечного приступа

Интуиция нам подсказывает, что мы сделаем что-то не так, просто объединив пропорции. Если вещи справедливы на любом локальном уровне, то они справедливы и в глобальном масштабе; или если вещи ведут себя определенным образом на локальном уровне, то следует ожидать, что они будут вести себя так же и в глобальном масштабе.

Нет ничего удивительного в том, что эта ошибка встречается так часто, ведь до относительно недавнего времени люди не имели представления о дробях. В древних текстах можно найти ошибки в операциях с дробями, например, в таких сферах, как наследование и торговля. Похоже, что сопротивление нашего мозга дробям по-прежнему сохраняется: мы изучаем дроби в пятом классе, и именно в этом возрасте у многих людей зарождается нелюбовь к математике.

Как же правильно объединить данные? Интуиция пятиклассника подсказывает, что нужно использовать общий знаменатель 40 и назначить зависимость от пола: $3/40 > 2/40$ для женщин и $16/40 > 12/40$ для мужчин. Теперь, поскольку в общей популяции мужчины и женщины распределены одинаково, нужно взять среднее значение и правомерно заключить, что $(3/40 + 16/40)/2 > (2/40 + 12/40)/2$, т. е. частота сердечных приступов в общей популяции при приеме препарата составляет 23,75% и 17,5% без препарата. Никакого сверхъестественного и нелогичного разворота не произошло. К тому же препарат оказался довольно плохим!

Большие случайные матрицы

Большинство приложений ИИ связаны с огромным количеством высокоразмерных данных (большие данные), организованных в виде высокоразмерных векторов, матриц или тензоров, представляющих собой таблицы данных, изображения, естественный язык, графовые сети и т. д. Многие из этих данных зашумлены или обладают изначально случайным характером. Для обработки таких данных требуется математическая база, сочетающая в себе теорию вероятностей и статистику, которые обычно оперируют *скалярными* случайными величинами, и линейную алгебру, которая имеет дело с *векторами* и *матрицами*.

Центральными идеями по-прежнему остаются среднее значение и дисперсия, так что можно найти множество утверждений и результатов, содержащих ожидание и дисперсию (неопределенность) задействованных высокоразмерных случайных величин. Как и в случае скалярных величин, сложность вызывает контроль дисперсии, поэтому существует множество публикаций, посвященных определению границ (неравенств) для хвостов распределений случайных величин или вероятности нахождения случайной величины в пределах некоторого расстояния от ее среднего значения.

Поскольку теперь мы имеем дело с матричными случайными величинами, во многих результатах делается попытка понять поведение (распределение) их спектров — собственных значений и собственных векторов.

Примеры случайных векторов и случайных матриц

Неудивительно, что изучение больших случайных матриц превратилось в отдельную теорию. Им находятся всевозможные применения — от сферы финансов до нейробиологии, физики и производства технологических устройств. Ниже приведена лишь часть примеров. Они имеют огромное значение, поэтому вокруг каждой из них образованы большие математические сообщества.

Количественные финансы

В качестве примера случайного вектора можно рассмотреть инвестиционный портфель в количественных финансах. Нам часто нужно решить, как наиболее эффективно инвестировать в большое количество акций, движение цен которых стохастично. Сам инвестиционный портфель представляет собой большой случайный вектор, который со временем изменяется. Аналогичным образом ежедневная доходность акций Nasdaq (Nasdaq включает в себя более 2500 акций) — это изменяющийся во времени большой случайный вектор.

Нейробиология

Приведем еще один пример из нейробиологии. Случайные матрицы появляются при моделировании сети синаптических связей между нейронами в мозге. Так, число импульсов, испускаемых n нейронами в течение t последовательных временны́х интервалов определенной длины, — это случайная матрица размера $n \times t$.

Математическая физика: матрицы Вигнера

В области математической физики, в частности в ядерной физике, физик Юджин Вигнер ввел случайные матрицы для моделирования ядер тяжелых атомов и их спектров. В двух словах, он связал расстояния между полосами в спектре ядра тяжелого атома с расстояниями между собственными значениями случайной матрицы.

Вигнер начал с детерминированной матрицы — гамильтониана системы, представляющего собой матрицу, которая описывает все взаимодействия между содержащимися в ядре нейтронами и протонами. Задача диагонализации гамильтониана для нахождения энергетических уровней ядра была невыполнима, поэтому Вигнер искал альтернативу. Он полностью отказался от точности и детерминизма и подошел к вопросу с вероятностной точки зрения. Для того чтобы *точно определить энергетические уровни*, он задавался такими вопросами:

- Какова вероятность найти энергетический уровень в определенном интервале?
- Какова вероятность того, что расстояние между двумя последовательными энергетическими уровнями находится в определенном интервале?
- Можно ли заменить гамильтониан системы сугубо случайной матрицей с правильными свойствами симметрии? Например, в случае квантовых систем, инвариантных к изменению времени, гамильтониан представляет собой вещественную симметричную матрицу (бесконечного размера). В присутствии магнитного

поля гамильтониан представляет собой компле́ксную эрмитову матрицу (компле́ксный аналог вещественной симметричной матрицы). При наличии *спин-орбитального взаимодействия* (термин квантовой физики) гамильтониан является симплектическим (еще один специальный тип симметричной матрицы).

Аналогичным образом случайные матрицы Вигнера применяются в физике конденсированных сред при моделировании взаимодействия между парами атомов или парами спинов с помощью вещественных симметричных матриц Вигнера. В целом в теории случайных матриц матрицы Вигнера считаются *классическими*.

Многомерная статистика: матрицы Уишарта и ковариация

В области многомерной статистики случайные матрицы ввел Джон Уишарт при попытке оценить выборочные ковариационные матрицы больших случайных векторов. Случайные матрицы Уишарта также считаются классическими в теории случайных матриц. Важно отметить, что выборочная ковариационная матрица — это оценка ковариационной матрицы популяции.

При работе с выборочными ковариационными матрицами обычно рассматриваются переменные размерности n, наблюдаемые t раз, т. е. исходный набор данных представляет собой матрицу размера $n \times t$. Например, нам может потребоваться оценить ковариационную матрицу доходности большого количества активов (используя меньшую выборку), например ежедневную доходность 2500 акций Nasdaq. Если мы воспользуемся ежедневными данными за последние 5 лет, учитывая, что в году 252 торговых дня, то у нас получится $5 \times 252 = 1260$ точек данных для каждой из 2500 акций.

Исходный набор данных будет представлять собой матрицу размера 2500×1260. Это тот случай, когда количество наблюдений меньше количества переменных. Бывают и противоположные случаи, а также предельные случаи, когда число наблюдений и число переменных резко различаются по масштабу. Во всех случаях нас интересует закономерность (распределение вероятностей) собственных значений выборочной ковариационной матрицы.

Запишем формулы для элементов ковариационной матрицы. В случае одной переменной $\vec{z}_1$ (скажем, одной акции) с t наблюдениями и средним значением $\overline{z}_1$ дисперсия имеет вид:

$$\sigma_1^2 = \frac{\left(z_1(1) - \overline{z}_1\right)^2 + \left(z_1(2) - \overline{z}_1\right)^2 + \ldots + \left(z_1(t) - \overline{z}_1\right)^2}{t}.$$

Аналогично, для каждой из n переменных $\vec{z}_i$ мы имеем их дисперсию σ_i^2. Они расположены на диагонали ковариационной матрицы. Теперь каждая запись вне диагонали σ_{ij} — это ковариация соответствующей пары переменных:

$$\sigma_{ij} = \frac{\left(z_i(1) - \overline{z}_i\right)\left(z_j(1) - \overline{z}_j\right) + \left(z_i(2) - \overline{z}_i\right)\left(z_j(2) - \overline{z}_j\right) + \ldots + \left(z_i(t) - \overline{z}_i\right)\left(z_j(t) - \overline{z}_j\right)}{t}.$$

Ковариационная матрица является симметричной и положительно определенной (имеет положительные собственные значения). Случайность в ковариационной матрице обычно обусловлена зашумленностью наблюдений. Поскольку при измерениях шум неизбежен, с математической точки зрения определение ковариационной матрицы становится более сложным. Другой распространенной проблемой является то, что часто выборки не являются независимыми. Коррелированные выборки вносят некоторую избыточность, поэтому мы ожидаем, что выборочная ковариационная матрица будет вести себя так, как будто мы наблюдаем меньше выборок, чем на самом деле. В этом случае необходимо проанализировать выборочную ковариационную матрицу в присутствии коррелированных выборок.

Динамические системы

Линеаризованные динамические системы находятся вблизи равновесия $\left(\frac{d\bar{x}(t)}{dt} = \mathbf{A}\bar{x}(t)\right)$. В контексте хаотических систем нам нужно понять, как небольшое различие в начальных условиях распространяется по мере развития динамики. Один из подходов заключается в линеаризации динамики в окрестности невозмущенной траектории. Возмущение развивается как произведение матриц, соответствующих линеаризованной динамике, наложенной на начальное возмущение.

Алгоритмы умножения матриц

Поиск эффективных алгоритмов умножения матриц является важной и в то же время удивительно трудной задачей. В алгоритмах умножения матриц внимания заслуживает экономия даже на одной операции умножения (экономия на сложении не столь значительна). Недавно компания DeepMind разработала модель AlphaTensor (2022 г., **https://oreil.ly/HZPbd**) для автоматического поиска более эффективных алгоритмов умножения матриц. Это стало важной вехой, поскольку умножение матриц является основополагающей частью огромного количества технологий, включая нейросети, компьютерную графику и научные вычисления.

Другие равнозначные примеры

Можно привести несколько других примеров. Так, в теории чисел можно смоделировать распределение нулей дзета-функции Римана с помощью распределения собственных значений некоторых случайных матриц. Для тех, кто интересуется квантовыми вычислениями, приведем историческую справку: прежде чем вывести уравнение Шрёдингера, Гейзенберг сформулировал квантовую механику в терминах, как он назвал, *матричной механики*. Под занавес, в *главе 13* мы познакомимся с *основным уравнением изменения вероятностей*. Оно включает в себя большую матрицу вероятностей перехода из одного состояния системы в другое.

Основные положения теории случайных матриц

В зависимости от формулировки задачи матрицы могут быть либо детерминированными, либо случайными. Для детерминированных векторов и матриц применяется классическая численная линейная алгебра, хотя чрезвычайно высокая размерность заставляет использовать рандомизацию для эффективного умножения матриц (обычно $O(n^3)$), разложения и вычисления спектра (собственных значений и собственных векторов).

Значительная часть свойств матриц заключена в их спектрах, поэтому можно многое узнать о матрицах, изучая их собственные значения и собственные векторы. В стохастической области, когда матрицы случайны, они также случайны. Как же их вычислить и найти распределения вероятностей (или даже только средние значения и дисперсии или их пределы)? Именно такими вопросами занимается область больших случайных матриц (или рандомизированной линейной алгебры, или высокоразмерной вероятности). В основном мы фокусируемся на следующих аспектах.

Задействованные стохастические математические объекты.

Случайные векторы и случайные матрицы. Каждая запись случайного вектора или случайной матрицы является случайной переменной. Это могут быть либо статические случайные переменные, либо изменяющиеся со временем переменные. При изменении случайной переменной со временем процесс становится случайным, или стохастическим. Очевидно, что с точки зрения математики стохастические процессы считаются сложнее их статических аналогов. Например, что мы можем сказать о дисперсиях, изменяющихся со временем?

Случайные проекции.

Для нас всегда представляет интерес проецирование на пространства более низкой размерности с сохранением существенной информации. Как правило, для этого приходится либо умножать матрицы на векторы, либо факторизовать матрицы в произведение более простых матриц, например методом сингулярного разложения. Как это возможно, если данных много, а записи случайны?

Сложение и умножение случайных матриц.

Важно отметить, что суммы и произведения скалярных случайных переменных также являются случайными величинами, и их распределения хорошо изучены. В научной литературе имеется большое количество работ, посвященных суммам и произведениям изменяющихся во времени скалярных случайных величин, которые лежат в основе броуновского движения и стохастического исчисления. Каким образом эта теория переходит в более высокие измерения?

Вычисление спектров.

Как вычислить спектр случайной матрицы и исследовать свойства ее (случайных) собственных значений и собственных векторов?

Вычисление спектров сумм и произведений случайных матриц.

А как выполнить это?

Умножение большого количества случайных матриц вместо двух.

Эта проблема возникает во многих контекстах в технологической отрасли, например, при изучении передачи света в последовательности плит с различными оптическими индексами, распространения электрона в неупорядоченной проволоке или распространения смещений в гранулированных средах.

Байесовская оценка матриц.

Байесовские методы всегда связаны с оценкой вероятности чего-либо при наличии некоторых данных. Здесь матрица, с которой мы начинаем (матрица наблюдений), представляет собой зашумленную версию истинной матрицы, которая нам важна. Шум может быть аддитивным, тогда наблюдаемая матрица $\mathbf{E}$ = *истинная матрица* + *случайная матрица шума*. Также шум может быть мультипликативным, тогда наблюдаемая матрица $\mathbf{E}$ = *истинная матрица* × *случайная матрица шума*. В целом нам неизвестна истинная матрица, но нам нужно узнать ее вероятность с учетом того, что мы наблюдали матрицу шума. То есть требуется вычислить Prob(*истинная матрица* | *матрица шума*).

Ансамбли случайных матриц

В львиной доле приложений мы сталкиваемся с большими матрицами (стохастическими или детерминированными), не имеющими определенной структуры. Основная предпосылка, лежащая в основе теории случайных матриц, состоит в том, что такую большую сложную матрицу можно заменить типичным элементом (ожидаемым элементом) определенного ансамбля случайных матриц. Как правило, мы ограничиваемся симметричными матрицами с вещественными элементами, поскольку именно они чаще всего встречаются в анализе данных и статистической физике. На наше счастье, они легче поддаются математическому анализу.

С точки зрения математики, нам привычнее полиномиальные функции. Они обладают нелинейной структурой, достаточно сложны и способны отразить все многообразие окружающего нас мира, их легко оценивать, и с ними легко производить вычисления. При изучении больших случайных матриц возникает особый тип хорошо известных многочленов (полиномов) — ортогональные многочлены. Ортогональная полиномиальная последовательность — это семейство многочленов, в котором два любых многочлена в последовательности ортогональны (их внутреннее произведение равно нулю) друг другу при некотором внутреннем произведении, которое представляет собой обобщенное точечное произведение. К наиболее распространенным последовательностям ортогональных многочленов относятся многочлены Эрмита, Лагерра и Якоби (к ним относятся важные классы многочленов Чебышева и многочленов Лежандра). Известными представителями области ортогональных полиномов, получившей развитие преимущественно в конце XIX века, являются Чебышев, Марков, Стилтьес. Неудивительно, что эти имена встречаются повсюду в теории вероятностей — от неравенств Чебышева, цепей и процессов Маркова до преобразований Стилтьеса.

Ниже приведены три основных типа ансамблей случайных матриц, тесно связанных с ортогональными многочленами.

Ансамбль Вигнера.

Представляет собой матричный эквивалент гауссова распределения. Матрица Вигнера размера 1×1 представляет собой одно гауссово случайное число. Она тесно связана с ортогональными многочленами Эрмита. Гауссово распределение и связанные с ним многочлены Эрмита естественным образом проявляются в контекстах, где базовая переменная не ограничена сверху и снизу. Средние значения характеристических многочленов случайных матриц Вигнера подчиняются простым рекуррентным соотношениям, что позволяет выразить их в виде многочленов Эрмита. Ансамбль Вигнера считается самым простым из всех ансамблей случайных матриц. К ним относятся матрицы, все элементы которых являются гауссовыми случайными переменными, с единственным ограничением, что матрица должна быть вещественной симметричной (гауссов ортогональный ансамбль), компле́ксной эрмитовой (гауссов унитарный ансамбль) или симплектической (гауссов симплектический ансамбль).

Ансамбль Уишарта.

Представляет собой матричный эквивалент гамма-распределения. Ансамбль Уишарта 1×1 — это гамма-распределенное число. Оно тесно связано с ортогональными многочленами Лагерра. Гамма-распределение и многочлены Лагерра применяются в задачах, где переменная ограничена снизу (например, положительные переменные). Средние значения характеристических многочленов случайных матриц Уишарта подчиняются простым рекуррентным соотношениям, что позволяет выражать их в виде многочленов Лагерра.

Ансамбль Якоби.

Представляет собой матричный эквивалент бета-распределения. Ансамбль Уишарта 1×1 — это бета-распределенное число. Оно тесно связано с ортогональными многочленами Якоби. Бета-распределения и многочлены Якоби применяются в задачах, где переменная ограничена сверху и снизу. Естественным образом матрицы Якоби возникают в случаях выборочных ковариационных матриц. Они также встречаются в простой задаче сложения или умножения матриц с двумя собственными значениями.

Что касается скалярных случайных величин, то здесь мы изучаем моменты и преобразование Стилтьеса ансамблей случайных матриц. К тому же мы находимся в области матриц и поэтому изучаем совместные распределения вероятностей их собственных значений. В случае упомянутых выше ансамблей собственные значения будут сильно коррелированными, и их можно представить как частицы, взаимодействующие посредством попарного отталкивания. Они называются собственными значениями кулоновского отталкивания, а сама идея заимствована из статистической физики (например, "Закономерности в собственных значениях" Перси Диакониса (2003 г., **https://oreil.ly/Cg0hY**) для более глубокого понимания поведения собственных значений матриц с особой структурой). Оказывается, что

наиболее вероятные положения задачи кулоновского газа совпадают с нулями многочленов Эрмита в ансамбле Вигнера и многочленов Лагерра в ансамбле Уишарта. К тому же собственные значения этих ансамблей незначительно колеблются в окрестности своих наиболее вероятных положений.

Плотность собственных значений суммы двух больших случайных матриц

Кроме поисков совместного распределения вероятностей собственных значений ансамблей случайных матриц, нас интересует плотность собственных значений (распределение вероятностей) сумм больших случайных матриц по каждой отдельной матрице в сумме. В этом контексте возникает *броуновское движение Дайсона*. Оно представляет собой распространение *броуновского движения* от скалярных случайных величин на случайные матрицы. Более того, преобразование Фурье для матрицы позволяет определить аналог порождающей функции для скалярных независимых и тождественно распределенных случайных переменных и использовать ее логарифм для нахождения плотности собственных значений сумм тщательно выстроенных случайных матриц. Наконец, к максимальному собственному значению конечной суммы случайных эрмитовых матриц можно применить неравенства Чернова, Бернштейна и Хефдинга.

Базовая математика для больших случайных матриц

Прежде чем мы завершим обсуждение больших случайных матриц, выделим ряд *обязательных моментов*, которые необходимо знать, если мы хотим глубоко погрузиться в эту область. Некоторые из них мы затронем в этой главе, остальные при желании можно найти в Google:

- вычисление спектра: собственные значения и собственные векторы матрицы (решения $\mathbf{A}\vec{v} = \lambda\vec{v}$);
- характеристический многочлен матрицы ($\det(\lambda\mathbf{I} - \mathbf{A})$);
- ортогональные многочлены Эрмита, Лагерра и Якоби;
- моменты и порождающая функция момента случайной величины;
- гауссово, гамма- и бета-распределения вероятностей;
- преобразование Стилтьеса;
- все по Чебышеву;
- все по Маркову;
- неравенства Чернова, Бернштейна и Хефдинга;
- броуновское движение и броуновское движение Дайсона.

По состоянию на 2022 год самым быстрым суперкомпьютером считался *Frontier* — первый в мире экзафлопсный компьютер (1,102 эксафлопса), находящийся в Ок-Риджской национальной лаборатории Министерства энергетики США. Когда мат-

рицы очень большие, даже на таком суперкомпьютере невозможно применить численную линейную алгебру (например, решить системы уравнений с участием матрицы, найти ее спектр или сингулярное разложение) в том виде, в котором мы ее знаем. Вместо этого нам приходится осуществлять *случайную выборку столбцов матрицы*. Лучше всего выбирать столбцы с вероятностью, которая приводит к наиболее верному приближению, т. е. с наименьшей дисперсией. Например, если задача состоит в умножении двух больших матриц **A** и **B** друг на друга, вместо равномерной выборки столбца из **A** и соответствующей строки из **B** мы выбираем столбец из **A** и соответствующую строку из **B** с вероятностью p_j, пропорциональной *норм*(*столбец j из* **A**)*норм*(*строка j из* **B**). Это значит, что чаще всего мы выбираем столбцы и строки с большими значениями *норм*, что повышает вероятность охватить *важные части произведения*.

Важную роль играют пространства столбцов больших и не очень больших матриц. Тремя лучшими базисами для пространства столбцов заданной матрицы **A** считаются:

- сингулярные векторы из сингулярного разложения;
- ортогональные векторы из процесса Грама — Шмидта (знаменитое *QR*-разложение матрицы);
- линейно независимые столбцы, выбранные непосредственно из столбцов **A**.

Стохастические процессы

Теперь перед нами не статическая (скалярная, векторная, матричная или тензорная), а зависящая от времени случайная величина. Так или иначе, следующий шаг в математике всегда заканчивается включением изменяющихся во времени сущностей. Кстати, люди еще не до конца поняли природу времени и не нашли способа сформулировать его определение. Однако мы понимаем, что такое движение и изменение, что такое переход системы из одного состояния в другое, и связываем с этим время. Мы также связываем вероятности перехода из одного состояния в другое. Запомним это для цепей Маркова, о которых мы поговорим чуть позже.

Стохастический процесс — это *бесконечная* последовательность X_0, X_1, X_2, ... случайных величин, где индекс t в каждом X_t мы считаем дискретным временем. Таким образом, X_0 — это процесс в момент времени 0 (или значение случайной величины в определенный момент времени 0), X_1 — процесс в момент времени 1 (или значение случайной величины в определенный момент времени 1), и т. д. Для того чтобы формально определить случайную величину, чего мы еще не сделали, ее фиксируют над так называемой вероятностной тройкой (*пространство выборок*, *сигма-алгебра*, *мера вероятности*). Пока не будем ломать голову над значением этой тройки, а лучше обратим внимание на то, что все случайные переменные в одном стохастическом процессе X_0, X_1, X_2, ... существуют в рамках *одной и той*

же вероятностной тройки, т. е. относятся к одному семейству. Кроме того, эти случайные переменные зачастую не являются независимыми.

Не менее важным (в зависимости от области применения) является стохастический процесс с непрерывным временем, где X_t теперь кодирует значение случайной величины в *любой* неотрицательный момент времени t. Более того, это легко согласуется с нашим интуитивным восприятием времени, которое является непрерывным.

Таким образом, стохастический процесс — это обобщение многомерного распределения конечной размерности на бесконечную размерность. Такой способ мышления пригодится при попытке доказать существование стохастического процесса, поскольку в этом случае можно прибегнуть к теоремам, позволяющим *расширить* его до бесконечных размерностей, опираясь на конечные коллекции составляющих распределений.

Примеры стохастических процессов окружают нас повсюду. О них мы вспоминаем всякий раз, когда сталкиваемся с колебаниями — это движение молекул газа, колебания электрического тока, цены на акции на финансовых рынках, количество телефонных звонков в колл-центр за определенный период времени, процесс азартной игры. Приведем интересный вывод из статьи по микробиологии, посвященной бактериям, обитающим в кишечнике (**https://oreil.ly/sdz6v**): *совокупность сообществ кровососущих, блох или клещей управляется в основном стохастическими процессами, в то время как микробиом кишечника обусловлен детерминированными процессами.*

Центральное место в теории стохастических процессов занимает пример фондового рынка, т. к. именно в нем получило широкое распространение броуновское движение (которое также называется стохастическим процессом Винера), когда Л. Башелье изучал изменения цен на Парижской бирже. Не менее важным считается также пример телефонных звонков в колл-центр, поскольку именно так был популяризирован стохастический процесс Пуассона, в котором А. К. Эрланг моделировал количество телефонных звонков, поступающих за определенный промежуток времени.

Эти два процесса — броуновский и пуассоновский — проявляются во многих других ситуациях, не имеющих ничего общего с приведенными примерами. Возможно, это говорит нам что-то более глубокое о природе и единстве лежащих в ее основе процессов, но не будем вдаваться в философию и останемся с математикой. В целом стохастические процессы можно разделить на несколько категорий в зависимости от их математических свойств. Некоторые из них являются процессами с дискретным временем, а другие — с непрерывным. Они различаются скорее интуитивно.

Для того чтобы сделать выводы об интересующих нас броуновском, пуассоновском, а также других стохастических процессах, необходимо проанализировать их математически. В теории вероятностей все начинается с выявления *существования* стохастического процесса. Таким образом, необходимо четко определить вероятностную тройку (пространство выборок, сигма-алгебру, меру вероятности), в которой

существует дискретная временна́я бесконечная последовательность случайных величин X_0, X_1, X_2, ... или непрерывный временной процесс X_t, и доказать, что можно найти такой набор случайных величин, удовлетворяющий его характеристическим свойствам. Мы вернемся к этому вопросу позже в текущей главе, а пока при доказательстве существования стохастических процессов нам будет важно имя Андрея Николаевича Колмогорова (1903–1987), а точнее — *теорема существования Колмогорова*. Она гарантирует существование стохастического процесса, имеющего те же распределения *конечной размерности*, что и искомые процессы. То есть желаемый стохастический процесс (бесконечный процесс, индексированный по дискретному или непрерывному времени) можно получить, задав все распределения конечной размерности некоторым *последовательным* образом.

Рассмотрим самые известные стохастические процессы.

Процесс Бернулли

Процесс Бернулли представляет собой стохастический процесс, который чаще всего связан с многократным подбрасыванием монеты и любыми процессами в жизни, имитирующими этот процесс (в некоторых аэропортах таможенники заставляют нажимать на кнопку, и если загорается зеленый свет, мы проходим, если красный — производится досмотр). С точки зрения математики, это бесконечная последовательность независимых и одинаково распределенных случайных величин X_0, X_1, X_2, ..., где каждая случайная величина принимает либо значение ноль с вероятностью p, либо единицу с вероятностью $1 - p$. Выборочная реализация этого процесса будет выглядеть как 0, 1, 1, 0, ...

Процесс Пуассона

Процесс Пуассона, или пуассоновский процесс, можно рассматривать как стохастический процесс, в основе которого лежат *счетные переменные*. Они подсчитывают, сколько интересных событий происходит за определенный промежуток времени. Эти события либо независимы, либо слабо зависимы, и каждое из них имеет небольшую вероятность наступления. Они происходят с заданной ожидаемой скоростью λ. Этот параметр характеризует пуассоновскую случайную величину. Например, в теории очередей он используется для моделирования прихода покупателей в магазин, телефонных звонков в колл-центр, возникновения землетрясений в определенный промежуток времени. В качестве *пространства состояний* этого процесса выступают натуральные числа, а в качестве множества индексов — неотрицательные числа. Вероятностное распределение, лежащее в основе задействованных в пуассоновском процессе случайных величин, имеет следующую формулу:

$$\text{Prob}(X = n) = \frac{\lambda^n e^{-\lambda}}{n!}.$$

Формула дает вероятность того, что за единицу времени произойдет n интересных событий. Очевидно, что за фиксированный промежуток времени маловероятно по-

явление большого числа редких событий, что объясняет стремительное убывание в формуле при большом n. Математическое ожидание и дисперсия пуассоновской случайной величины равны λ.

Пуассоновский процесс X_t; $t \geq 0$, индексированный непрерывным временем, имеет следующие характеристики:

- $X_0 = 0$;
- количество событий (или точек) в любом интервале длиной t является пуассоновской случайной величиной с параметром λt.

Этот процесс имеет две важные особенности:

- количество событий в каждом конечном интервале является пуассоновской случайной величиной (имеет пуассоновское распределение вероятностей);
- количество событий в разобщенных временны́х интервалах — независимые случайные величины.

Процесс Пуассона служит примером стохастического процесса Леви, под которым понимается процесс со стационарными независимыми приращениями.

Случайное блуждание

Мы можем легко представить простейшее случайное блуждание как ходьбу пешком по дороге, когда, начав с какого-то места, мы идем вперед (прибавляя единицу к своему местоположению) с вероятностью p и назад (вычитая единицу из своего местоположения с вероятностью $1-p$. Можно задать итоговый стохастический процесс с дискретным временем X_0, X_1, ... так, что $X_0 = x_0$, $X_1 = X_0 + Z_1$, $X_2 = X_1 + Z_2 = X_0 + Z_1 + Z_2$, и т. д., где Z_1, Z_2, ... — процесс Бернулли. При $p = 0{,}5$ мы имеем симметричное случайное блуждание.

В *главе 9* мы неоднократно использовали случайные блуждания по графам, когда мы начинаем с некоторого узла графа и затем переходим к одному из его соседних узлов с заданными вероятностями. Нормированная матрица смежности графа задает вероятности перехода во всех узлах. Это наглядный пример того, как случайные блуждания на графах связаны с цепями Маркова. Подробнее об этом можно прочитать, перейдя по ссылке к подборке заметок о случайных блужданиях по графам (**https://oreil.ly/Cjk0C**).

Процесс Винера, или броуновское движение

Процесс Винера, или броуновское движение, можно представить как случайное блуждание с бесконечно малыми шагами, так что дискретные движения превращаются в бесконечно малые колебания, и мы получаем *непрерывное случайное блуждание*. Броуновское движение — это стохастический процесс с непрерывным временем X_t; $t \geq 0$. Случайные величины X_t являются вещественными, имеют независимые приращения, а разница между X_t и X_s в два отдельных момента

времени t и s нормально распределена (колоколообразное гауссово распределение) со средним значением 0 и дисперсией $t-s$. Иными словами, $X_t - X_s$ нормально распределены по величине приращений.

Стохастический процесс с непрерывным временем, имеющий вещественную величину, интересен тем, что способен двигаться по непрерывным траекториям, порождая интересные случайные функции времени. Например, почти наверняка выборочная траектория броуновского движения (процесса Винера) везде непрерывна, но нигде не дифференцируема (слишком много пиков).

Броуновское движение является основополагающим в изучении стохастических процессов. Оно выступает отправной точкой стохастического исчисления, находясь на пересечении нескольких важных процессов — гауссова марковского процесса, процесса Леви (процесс со стационарными независимыми приращениями) и мартингейла, о котором пойдет речь далее.

Мартингейл

Мартингейл (или мартингал — martingale) с дискретным временем представляет собой стохастический процесс X_0, X_1, X_2, ..., где для любого дискретного времени t:

$$\mathbb{E}\left(X_{t+1} \middle| X_1, X_2, ..., X_t\right) = X_t.$$

Это значит, что ожидаемое значение следующего наблюдения с учетом всех предыдущих равно самому последнему наблюдению. Определение довольно странное (к сожалению, оно очень распространено в этой области), но приведем несколько небольших примеров контекстов, в которых встречаются мартингейлы.

- Несмещенное случайное блуждание — это пример мартингейла.
- Везение азартного игрока — это мартингейл в случае, когда все ставки, которые он делает, справедливы. Предположим, что игрок выигрывает 1 доллар, когда на подброшенной монете выпадает орел, и проигрывает 1 доллар, когда выпадает решка. Если X_n — это везение игрока после n бросков, то его условно ожидаемое везение после следующего подбрасывания монеты с учетом истории равно его текущему везению.
- В экологическом сообществе, где группа видов конкурирует за ресурсы, можно смоделировать количество особей каждого конкретного вида как стохастический процесс. Такая последовательность является мартингейлом в рамках единой нейтральной теории биоразнообразия и биогеографии (**https://oreil.ly/oGlR2**).

При обсуждении мартингала возникает вопрос о времени остановки. Эта интересная концепция отражает идею о том, что *в любой конкретный момент времени t можно посмотреть на последовательность и определить, не пора ли остановиться*. Время остановки стохастического процесса X_1, X_2, X_3, ... — случайная переменная S (от англ. *Stop*), обладающая таким свойством, что для каждого t возникновение или не возникновение события $S = t$ зависит только от значений X_1, X_2, X_3, ..., X_t. Например, случайная переменная времени остановки моделирует время, когда игрок

решит остановиться и покинуть игорный стол. Это будет зависеть от его предыдущих выигрышей и проигрышей, а не от результатов еще не сыгранных игр.

Процесс Леви

Мы уже упоминали процесс Пуассона и броуновское движение (процесс Винера) как два наиболее популярных примера процесса Леви. Это стохастический процесс с независимыми стационарными приращениями. Он может моделировать движение частицы, чьи последовательные перемещения являются случайными, при этом перемещения в попарно разделенных временны́х интервалах независимы, а перемещения в разных временны́х интервалах одинаковой длины имеют одинаковые распределения вероятности. В этом смысле он является аналогом случайного блуждания в непрерывном времени.

Процесс ветвления

Процесс ветвления представляет собой случайное разбиение на ветви. Например, он моделирует эволюцию некоторой популяции (скажем, бактерий или нейтронов в ядерном реакторе), где каждая особь в данном поколении производит случайное число особей в следующем поколении в соответствии с некоторым фиксированным распределением вероятности, которое не меняется от особи к особи. Один из главных вопросов *теории ветвящихся процессов* — вероятность окончательного вымирания, когда через конечное число поколений популяция вымирает.

Цепь Маркова

Определим формально цепь Маркова с дискретным временем, т. к. это один из самых важных стохастических процессов, а также потому, что он встречается в контексте обучения с подкреплением в ИИ. Для определения цепи Маркова необходимо выполнение следующего.

- Наличие дискретного набора вероятных состояний S (конечное или бесконечное). Его можно рассматривать как набор состояний, в которых могут пребывать частица или агент. На каждом шаге марковского процесса происходит случайный переход из одного состояния в другое.
- Начальное распределение задает вероятность v_i каждого возможного состояния$_i$. Насколько вероятно, что частица будет изначально находиться в определенном месте или агент будет изначально находиться в определенном состоянии?
- Вероятности перехода p_{ij} задают вероятность того, что частица или агент перейдет из состояния$_i$ в состояние$_j$. Важно отметить, что для каждого состояния i имеется сумма $p_{i1} + p_{i2} + ... + p_{in} = 1$. Более того, этот процесс не имеет памяти, поскольку вероятность такого перехода зависит только от состояния$_i$ и состояния$_j$, а не от состояний, в которых они пребывали ранее.

Итак, цепь Маркова — это стохастический процесс X_0, X_1, ..., принимающий такие значения в S, что:

$$\text{Prob}\left(X_0 = \text{состояние}_{i_0}, X_1 = \text{состояние}_{i_1}, ..., X_n = \text{состояние}_{i_n}\right) = \nu_{i_0} p_{i_0 i_1} p_{i_1 i_2} \cdots p_{i_{n-1} i_n}.$$

Все вероятности перехода можно собрать в квадратную матрицу (матрицу Маркова, каждая строка которой состоит из неотрицательных чисел, в сумме дающих 1) и умножить на нее. Такая матрица суммирует вероятности перехода из любого состояния$_i$ в состояние$_j$ за один шаг. Замечательно то, что степени марковской матрицы также относятся к марковскими, так что, например, квадрат этой матрицы представляет собой суммарные вероятности перехода из любого состояния$_i$ в состояние$_j$ за два шага, и т. д.

К фундаментальным понятиям, связанными с цепями Маркова, относятся *переходность*, *возвратность* и *неразложимость*. Состояние$_i$ считается возвратным, если, начав из него, мы обязательно в него же и возвращаемся. Без такого возвращения оно считается переходным. Цепь Маркова считается неразложимой, когда возможен переход из одного состояния в любое другое.

Наконец, *вектор стационарных вероятностей*, задающий распределение вероятностей по возможным состояниям, — это вектор, который не меняется при умножении на матрицу перехода. Это напрямую связано с собственными векторами в линейной алгебре с соответствующим собственным значением 1. Возникает *приятное переживание*, когда знакомая нам математика собирается воедино, вызывая некое подобие зависимости, что, собственно, и удерживает нас в ловушке любви-ненависти к этой области. К счастью, чем дольше мы в ней работаем, тем больше перевес в сторону любви.

Лемма Ито

Рассмотрим еще немного математики. Стохастический процесс моделирует случайную величину, которая изменяется со временем. Функция стохастического процесса также случайным образом изменяется со временем. Когда происходит изменение детерминированных функций со временем, тут же, как правило, возникает вопрос: насколько быстро? Для ответа мы берем ее производную по времени и развиваем исчисление производных (и интегралов) детерминированных функций. Огромную роль, особенно в тренировке моделей машинного обучения, играет цепное правило.

Лемма Ито является аналогом цепного правила для функций стохастических процессов. Это аналог цепного правила в стохастическом исчислении. Она применяется для нахождения дифференциала функции стохастического процесса, зависящей от времени.

Марковские процессы принятия решений и обучение с подкреплением

В области искусственного интеллекта марковские процессы принятия решений связаны со следующими понятиями и именами.

Динамическое программирование и Ричард Беллман.

Беллман сыграл монументальную роль в этой области, а его условие оптимальности реализовано во многих алгоритмах.

Обучение с подкреплением.

Поиск оптимальной стратегии с помощью последовательности действий, связанных с положительным или отрицательным вознаграждением (метод проб и ошибок). У агента есть выбор между несколькими действиями и переходными состояниями, причем вероятности перехода зависят от выбранных действий.

Глубокое обучение с подкреплением.

Комбинация обучения с подкреплением и нейросетей. В этом случае нейросеть принимает на вход наблюдения и выдает вероятность каждого возможного действия, которое может предпринять агент (распределение вероятностей). Затем на основе оценок вероятностей агент случайным образом принимает решение о следующем действии. Например, если у агента есть два варианта: повернуть налево или направо, а нейросеть выдает 0,7 для поворота налево, то агент повернет налево с вероятностью 70%, а направо — с вероятностью 30%.

Обучение с подкреплением обладает огромным потенциалом в плане продвижения к общему интеллекту. Интеллектуальным агентам необходимо принимать рациональные решения, когда отдача от действий наступает не сразу, а в результате ряда последовательных действий. Таким образом, речь идет о рассуждениях в условиях неопределенности.

Примеры обучения с подкреплением

Можно привести множество примеров обучения с подкреплением: самоуправляемые автомобили, рекомендательные системы, домашний термостат (получает положительное вознаграждение, когда температура близка к заданной, что дает экономию энергии, и отрицательное, когда человек вынужден подстраивать температуру), автоматическое инвестирование на фондовом рынке (на входе — котировки акций, на выходе — количество акций, которые нужно купить или продать, вознаграждение — финансовая прибыль или убытки).

Пожалуй, самым известным примером успешного применения глубокого обучения с подкреплением является AlphaGo компании DeepMind (**https://oreil.ly/0TcsR**) — агент искусственного интеллекта, обыгравший в 2016 году лучшего в мире игрока-человека в древней китайской игре го. На интуитивном уровне понятно, что лучше всего рассматривать подкрепляющее обучение в контексте настольных игр, например шахмат или го, т. к. в них на каждом шаге мы принимаем решение о последо-

вательности дальнейших действий, прекрасно понимая, что наше текущее решение влияет на весь исход игры. На каждом шаге нужно действовать оптимально. Более того, на каждом шаге оптимальная стратегия будет *меняться*, т. к. она зависит в том числе от действий нашего противника (который решает точно такую же задачу, только со своей стороны).

По правде говоря, сейчас я несколько предвзято отношусь к примерам с играми — моя дочка "подсела" на PlayStation 5. Мне больше нравится пример с инвестиционным рынком. Некий финансовый консультант работает на ежедневно меняющемся рынке и должен принимать решения о покупке/продаже определенных акций на каждом временно́м шаге с долгосрочной целью максимизации прибыли и минимизации убытков. Рыночная среда считается стохастической, и мы не знаем ее правил, но в целях моделирования предполагаем, что они нам известны. Теперь заменим финансового консультанта агентом искусственного интеллекта и посмотрим, какую задачу оптимизации ему нужно будет решать на каждом временно́м шаге в условиях постоянно меняющейся рыночной среды.

Обучение с подкреплением как марковский процесс принятия решений

Обучение с подкреплением можно сформулировать математически как марковский процесс принятия решений. Агент ИИ существует в вероятностной среде, состоящей из состояний и вероятностей перехода между ними. Вероятности таких переходов *зависят от выбранных действий*. Таким образом, итоговый марковский процесс, кодирующий переходы из $\overrightarrow{состояния}$ в $\overrightarrow{состояние'}$, имеет явную зависимость от действия $\vec{a}$, которое совершено в $\overrightarrow{состоянии}$. Основное предположение здесь заключается в том, что мы *знаем* этот процесс, а это значит, что мы знаем правила среды. Другими словами, для каждого $\overrightarrow{состояния}$, $\overrightarrow{состояния'}$ и действия $\vec{a}$ нам известны следующие вероятности:

$$\text{Prob}\left(\text{следующее состояние} = \overrightarrow{состояние'} \middle| \text{текущее состояние} = \overrightarrow{состояние}, \text{действие} = \vec{a}\right).$$

Нам также известна система вознаграждения:

$$\text{Prob}\left(\text{значение следующего вознаграждения} \middle| \text{текущее состояние} = \overrightarrow{состояние}, \text{действие} = \vec{a}, \text{следующее состояние} = \overrightarrow{состояние'}\right).$$

Теперь разговор переходит в область динамического программирования: мы ищем *оптимальную стратегию* (последовательность правильных действий), ведущую к *оптимальному значению* (максимальное вознаграждение или минимальный убыток). Эта задача оптимизации чуть сложнее тех, что встречались нам до сих пор, т. к. речь здесь идет о *последовательности действий*, ведущих к оптимальному значению. Таким образом, необходимо разбить задачу на шаги, и на каждом шаге искать действие, которое приведет к оптимальному вознаграждению *в будущем че-*

рез несколько шагов вперед. Конкретно эту задачу решает *уравнение оптимальности Беллмана*, сводя ее к поиску только одного оптимального действия в текущем состоянии (в отличие от поиска всех действий сразу) с учетом того, что мы знаем, какую задачу нужно оптимизировать на каждом шаге. Беллман внес огромный вклад, утверждая, что оптимальное значение текущего состояния равно среднему вознаграждению после принятия *одного* оптимального действия *плюс ожидаемое оптимальное значение* всех возможных следующих состояний, к которым может привести данное действие.

Агент взаимодействует с окружающей средой в рамках итерационного процесса. Он запускается с начального состояния и набора возможных действий в этом состоянии (распределение вероятностей для выполнения действия в этом состоянии), а затем итерационно вычисляет:

1. Следующее оптимальное действие (которое переводит его в новое состояние с новым набором возможных действий). Это называется *итерацией стратегии*, а оптимизация нацелена на максимизацию будущего вознаграждения.
2. Ожидаемое значение (вознаграждение или убытки) при данном оптимальном действии. Это называется *итерацией ценности*.

Функция ценности складывает ожидаемые будущие вознаграждения агента с учетом его текущего состояния и оптимальной последовательности дальнейших действий:

$$\text{Ценность}\left(\overrightarrow{состояние}, \text{оптимальная последовательность действий}\right) =$$

$$= \mathbb{E}\left(\sum_k \gamma^k \, вознаграждение_k\right).$$

Коэффициент дисконтирования γ — число от 0 до 1. Он применяется для поощрения действий, ведущих к *скорейшему, но не позднейшему* положительному вознаграждению. Включив этот коэффициент в задачу оптимизации, можно корректировать значимость вознаграждений во времени, придавая меньший вес будущим вознаграждениям (если γ находится в диапазоне от 0 до 1, то γ^k будет маленьким для больших k).

Сделаем оптимизацию в функции ценности явной (выбираем последовательность действий, максимизирующее вознаграждение с учетом текущего состояния):

$$\text{Ценность}\left(\vec{s}\right) = \max_{\substack{\text{действия} \\ \text{и состояния}}} \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k \left(вознаграждение\right)_k \,\middle|\, \overrightarrow{состояние}_0 = \vec{s}\right).$$

Теперь разобьем его на части и убедимся, что текущее вознаграждение агента является явным, отдельно от его будущих вознаграждений:

$$\text{Ценность}\left(\vec{s}\right) =$$

$$= \max_{\substack{\text{действия} \\ \text{и состояния}}} \mathbb{E}\left(вознаграждение_0 + \sum_{k=1}^{\infty} \gamma^k \, вознаграждение_k \,\middle|\, \overrightarrow{состояние}_1 = \vec{s}'\right).$$

Наконец, находим, что функция ценности в текущем состоянии агента зависит от его текущего вознаграждения и дисконтированной функции ценности в его будущих состояниях:

$$\text{Ценность}(\vec{s}) = \max_{\substack{\text{действия} \\ \text{и состояния}}} \mathbb{E}\left(\textit{вознаграждение}_0 + \gamma \cdot \text{Ценность}\left(\vec{s'}\right)\right).$$

Это утверждение позволяет итеративно (обратно во времени) решить нашу главную задачу оптимизации. Все, что требуется агенту, — выбрать действие, которое приведет его к *следующему наилучшему состоянию*. Такое выражение для функции ценности является мощным *уравнением Беллмана*, или *условием оптимальности Беллмана*, которое разбивает исходную задачу оптимизации на рекурсивную последовательность гораздо более простых задач оптимизации, выполняя локальную оптимизацию в каждом состоянии (нахождение $\text{Ценность}\left(\vec{s'}\right)$), а затем подставляя результат в следующую подзадачу оптимизации (нахождение $\text{Ценность}(\vec{s})$). Чудо заключается в том, что, работая таким образом в обратном направлении от желаемого конечного вознаграждения к решению, какое действие предпринять сейчас, мы получаем общую оптимальную стратегию, а также оптимальную функцию ценности в каждом состоянии.

Обучение с подкреплением в контексте оптимального управления и нелинейной динамики

В *главе 13*, посвященной дифференциальным уравнениям в частных производных, мы вновь обратимся к обучению с подкреплением в контексте нелинейной динамики, оптимального управления и дифференциального уравнения Гамильтона — Якоби — Беллмана. В отличие от вероятностной марковской среды, с которой взаимодействовал агент ранее, подход динамического программирования к обучению с подкреплением (что приводит к дифференциальному уравнению Гамильтона — Якоби — Беллмана) носит детерминированный характер.

Библиотека Python для обучения с подкреплением

Наконец, весьма полезной библиотекой для реализации алгоритмов обучения с подкреплением считается библиотека TF-Agents (Google, в открытом доступе с 2018 года) — библиотека обучения с подкреплением на основе TensorFlow (Python).

Строгие теоретические основания

Для строгой, или математически точной, теории вероятностей требуется теория меры. *"Но зачем?"* — можно справедливо возразить. В конце концов, в течение долгого времени мы обходились без этого.

Потому что больше без этого уже не обойтись.

Запишем, но никогда не признаемся в этом вслух, что именно теория меры отталкивает большинство студентов от дальнейшего изучения математики в основном потому, что ее история никогда не излагается в хронологическом порядке относительно того, как и почему она возникла. Более того, бóльшая часть работы в теории вероятностей *сводится* к доказательству *существования* определенной случайной величины (на некотором пространстве выборок, пространстве событий или сигма-алгебре, а также меры для каждого события или набора в этой сигма-алгебре), как будто записи этой случайной величины и ее применения в моделировании разнообразных случайных сущностей недостаточно для существования. Наверное, именно поэтому математики и философы так прекрасно ладят друг с другом.

В этой главе мы рассмотрели довольно много понятий, правда, не задерживаясь подолгу ни на одном из них (студенты часто упрекают меня за такую высокую скорость изложения), так что нужно вернуться к началу и дать еще раз:

- точное математическое объяснение вероятностей и сигма-алгебры;
- точное математическое определение случайной величины и распределения вероятностей;
- точное математическое определение ожидаемого значения случайной величины и его связь с интегрированием;
- обзор вероятностных неравенств (контроль неопределенности);
- обзор закона больших чисел, центральной предельной теоремы и других теорем сходимости.

Впрочем, это звучит слишком амбициозно. Невозможно в одном разделе одной главы уместить полный курс по строгой теории вероятностей. Просто приведем убедительные аргументы в ее пользу и ограничимся сжатым изложением фундаментальных идей.

Начнем с двух основных ограничений нестрогой теории вероятностей (помимо того, что каждый из ее математических объектов имеет бесчисленное множество непоследовательных названий, обозначений и расплывчатых определений).

Какие события вероятны?

Какова вероятность того, что в заданном пространстве выборок (множестве, из которого мы вправе произвольно выбирать) можно определить какое-либо множество? Если равномерно выбирать числа из вещественной строки, то какова вероятность того, что мы выберем рациональное число? Алгебраическое число (решение некоторого полиномиального уравнения с целыми коэффициентами)? Или член из какого-то другого сложного подмножества вещественной строки?

Важно отметить, что эти вопросы постепенно погружают нас в детали теории множеств на вещественной прямой, которая, в свою очередь, напрямую приводит нас в теорию меры — теорию, которая рассматривает, какие *подмножества вещественной прямой можно измерить, а какие нельзя*.

Определение вероятности для подмножества пространства выборок начинает звучать как определение меры этого множества, и кажется, что только измеримые подмножества могут иметь определенную вероятность. А как насчет других неизмеримых подмножеств пространства выборки? К сожалению, невозможно определить для них вероятность. Повторим, что Prob(A) справедлива не для каждого подмножества A пространства выборки, а только для измеримых подмножеств. Следовательно, нам нужно собрать все измеримые подмножества вместе, отказаться от остальных, больше не думать ни о них, ни об их математике и расслабиться — теперь мы выполняем действия в области, где все собранные нами события (подмножества) имеют заданные вероятности (меры). Мера вероятности, с которой мы работаем, обладает необходимыми качествами в том смысле, что она представляет собой неотрицательное число из [0; 1], а вероятности взаимодополняющих событий (подмножеств) складываются в 1. Вся эта круговерть раскрывает хитросплетения вещественной линии и ее подмножеств, а в более общем случае — тайны континуума и чудеса бесконечного.

Строгая теория вероятностей позволяет оценить свойства как дискретных, так и непрерывных пространств, проявляющиеся в таких простых примерах, как построение дискретного равномерного распределения на дискретном множестве и построение непрерывного равномерного распределения на заданном интервале.

Возможен ли более широкий диапазон случайных величин?

Другое ограничение нестрогой вероятности, т. е. той, которая избегает теории мер, как мы ее только что описали, — это ограничение на виды случайных величин, которые она допускает. В частности, где именно проходит граница между дискретной и непрерывной случайной величиной? Существует ли такая линия на самом деле? А как насчет случайных величин, которые имеют как дискретный, так и непрерывный аспекты? В качестве простого примера предположим, что значение случайной переменной определяется подбрасыванием монеты. Она имеет пуассоновское распределение (дискретное), если монета выпадает орлом, и нормальное распределение (непрерывное), если монета выпадает решкой. Эта новая случайная величина не является ни полностью дискретной, ни полностью непрерывной в том нестрогом смысле, в котором мы понимаем любой из этих типов. Тогда что же это такое? Строгий ответ заключается в том, что, безусловно, между дискретными и непрерывными случайными величинами не будет никакой разницы, если мы определим основания, на которые опирается *каждая* случайная величина. То есть, какое множество образует пространство выборок? Какие подмножества этого выборочного пространства измеримы? Какова мера вероятности? Каково *распределение* случайной величины? Это общее основание, или отправная точка для *каждой* случайной величины. Как только мы определим его, тогда дискретность, непрерывность или что-то среднее между ними окажется незначительной деталью — такой же простой, как ответ на вопрос о том, с каким множеством (или, скажем, произведением множеств) мы работаем.

Вероятностная тройка (пространство выборок, сигма-алгебра, мера вероятности)

Все начинается с вероятностной тройки (не совсем так, но именно с этого начинается строгость). Она называется пространством вероятностной меры, при этом подразумевается, что мера всего пространства выборок равна единице. Это значит, что вероятность пространства выборок равна единице. Теперь можно сказать, что мы достаточно продвинулись и можем использовать слова "вероятность" и "мера" как взаимозаменяемые. Слово "мера" утешает тем, что возвращает нас в область детерминизма. Выборка случайна, но можно измерить вероятность любого события (это измеримо).

Пространство вероятностной меры составляют три объекта.

Пространство выборок.

Произвольное непустое множество, из которого случайным образом берутся выборки.

Сигма-алгебра.

Множество подмножеств пространства выборок, представляющих допустимые события (события, о вероятностях которых можно говорить, поскольку можно измерить только их). Сигма-алгебра должна содержать все пространство выборки, быть замкнутой по дополнениям (т. е. если множество включено в сигма-алгебру, то и его дополнение тоже) и замкнута по *счетным объединениям* (т. е. объединение счетных множеств подмножеств сигма-алгебры также включается в сигма-алгебру). Из двух предыдущих свойств и законов де Моргана (которые относятся к дополнениям объединений и пересечений) следует, что сигма-алгебра также замкнута на счетных пересечениях.

Вероятностная мера.

Число от нуля до единицы (включительно), связанное с *каждым подмножеством* сигма-алгебры и обладающее необходимыми свойствами, которые мы связываем с нежесткой вероятностью:

1. Prob(пространство выборок) = 1;
2. Prob(счетное объединение попарно непересекающихся множеств) = счетная сумма вероятностей каждого множества.

Это замечательно, ведь если нам удастся сформулировать множество пространства выборок, сигма-алгебру и функцию с обозначенными выше свойствами, связывающую каждый член сигма-алгебры с его мерой (вероятностью), то можно начать *строить теорию* на прочных основаниях, определяя все виды случайных величин, их ожидания, дисперсии, условные вероятности, суммы и произведения, пределы последовательностей, стохастические процессы, временные производные функций стохастических процессов (исчисление Ито) и т. д. В этом случае у нас не возникнет проблем с тем, для какого типа событий заданы вероятности (все члены сигма-алгебры тройки вероятностей), и какой тип случайных величин можно рассматривать (любые, которые можно строго определить на вероятностной тройке).

В чем сложность?

Важно отметить, что рассмотренные ограничения нестрогой вероятности проявляются, когда мы задействуем непрерывные переменные или когда пространство выборок находится в континууме (несчетное). Если бы наш мир был только дискретным, то не возникало бы никаких проблем. Когда мы переходим к строгой вероятности и пытаемся построить вероятностные тройки для дискретных выборочных пространств, мы не испытываем особых проблем. Проблемы возникают только в континуальном мире, где существуют несчетные пространства выборок, т. к. неожиданно нам приходится определять сигма-алгебры и связанные с ними вероятностные меры на множествах, где глубина бесконечного континуума не перестает восхищать. Например, эта проблема возникает даже тогда, когда требуется определить строгую вероятностную тройку для непрерывного равномерного распределения на отрезке [0; 1].

В этом нам поможет *теорема о расширении*, позволяющая строить сложные вероятностные тройки. Можно не задавать меру вероятности на массивной сигма-алгебре, а построить ее на более простом наборе подмножеств — *полуалгебре*, после чего теорема позволяет автоматически расширить меру до полной сигма-алгебры. С помощью этой теоремы можно построить меру Лебега на отрезке [0; 1] (которая в точности является непрерывным равномерным распределением на [0; 1]), меры произведения, многомерную меру Лебега, а также конечное и бесконечное подбрасывание монет.

Итак, мы наблюдаем органичное слияние теории множеств, вещественного анализа и теории вероятностей.

Случайная величина, ожидание, интегрирование

Теперь, когда появилась возможность связать вероятностную тройку с пространством выборки, задав вероятности для большого количества подмножеств пространства выборки (всех членов связанной сигма-алгебры), можно строго определить случайную переменную. Как известно из нестрогой теории вероятностей, случайная величина присваивает числовое значение каждому элементу пространства выборок. Так, если представить пространство выборок как все вероятные случайные исходы некоторого эксперимента (орлы и решки при подбрасывании монеты), то случайная переменная присваивает числовое значение каждому из этих исходов.

Для построения на строгих основаниях необходимо определить, как случайная переменная Y взаимодействует со всей вероятностной тройкой, связанной с пространством выборки. Краткий ответ будет таким: Y должна быть *измеримой функцией* от пространства выборок до вещественной прямой, в том смысле, что множество $Y^{-1}(-\infty, y)$ является членом сигма-алгебры, что, в свою очередь, означает, что это множество имеет вероятностную меру. Важно отметить, что Y переводит из про-

странства выборок в вещественную прямую, а Y^{-1} — обратно из вещественной прямой в подмножество пространства выборок.

По аналогии с тем, как случайная величина из области нестрогой вероятности оказывается измеримой функцией (относительно тройки) из области строгой вероятности, математическое ожидание $\mathbb{E}(Y)$ случайной величины оказывается тем же самым, что и интеграл случайной величины (измеримой функции) относительно вероятностной меры. Запишем:

$$\mathbb{E}(Y) = \int_{\Omega} YdP = \int_{\Omega} Y(\omega)\,\mathrm{Prob}(d\omega).$$

Что обозначает интеграл в формуле математического ожидания

Понять интеграл применительно вероятностной мере, как показано в приведенной формуле, будет легко, если представить значение математического ожидания случайной величины в дискретном случае как сумму значения случайной величины, помноженную на вероятность множества, по которому она принимает такое значение:

$$\mathbb{E}(Y) = \sum_{i=1}^{n} y_i \mathrm{Prob}\left(\omega \in \Omega \text{ так, что } Y(\omega) = y_i\right).$$

Теперь сравним это дискретное выражение с континуальным интегралом в формуле математического ожидания:

$$\mathbb{E}(Y) = \int_{\Omega} YdP = \int_{\Omega} Y(\omega)\,\mathrm{Prob}(d\omega).$$

Построим строго интеграл (ожидание) по аналогии с тем, как строится интеграл Лебега на первом курсе теории мер: сначала для простых случайных величин (которые можно легко разложить в дискретную сумму; интегралы начинаются с сумм), затем для неотрицательных случайных величин и, наконец, для общих случайных величин. Можно легко доказать основные свойства интегралов, например линейность и сохранение порядка. Важно отметить, что при наличии вероятностной тройки, интеграл будет иметь смысл (в гораздо более широком диапазоне, чем можно было даже представить в случае базового интеграла в стиле Реймана), независимо от того, будет пространство выборок дискретным, непрерывным или каким-либо еще более сложным. Когда мы сталкиваемся с интегралом Лебега, уже невозможно оглянуться назад.

Теперь у нас появилось математическое ожидание, и можно определить дисперсию и ковариацию точно так же, как в нестрогой теории вероятностей.

В этом случае можно говорить о независимости и важных свойствах, например, если X и Y независимы, то $E(XY) = E(X)E(Y)$ и $Var(X+Y) = Var(X) + Var(Y)$.

Распределение случайной величины и теорема о замене переменной

Распределение случайной величины X — это *соответствующая вероятностная тройка* ($\mathbb{R}$, B, μ), заданная на вещественной прямой так, что для каждого подмножества B из *борелевской сигма-алгебры*, заданной на вещественной прямой, мы имеем:

$$\mu(B) = P(X \in B) = P(X^{-1}(B)),$$

что полностью определяется кумулятивной функцией распределения $F_X(x) = P(X \le x)$ для X.

Предположим, имеется измеримая вещественная функция f, заданная на вещественной прямой. Пусть X — случайная величина на вероятностной тройке (Ω, сигма-алгебра, P) с распределением μ. Важно отметить, что при любом вещественном числе x значением $f(x)$ будет действительное число, а при случайной величине X значением $f(X)$ — случайная величина.

Теорема о замене переменной гласит, что ожидаемое значение случайной величины $f(X)$ с учетом меры вероятности P на пространстве выборок Ω равно ожидаемому значению функции f с учетом меры μ на $\mathbb{R}$. Запишем это сначала в терминах ожиданий, а затем в терминах интегралов:

$$\mathbb{E}_P(f(X)) = \mathbb{E}_\mu(f);$$

$$\int_\Omega f(X(\omega))P(d\omega) = \int_{-\infty}^{\infty} f(t)\mu(dt).$$

Теорема о замене переменной весьма полезна тем, что позволяет переключаться между ожиданиями, интегралами и вероятностями. Пусть f — индикаторная функция измеримого подмножества $\mathbb{R}$ (равная единице на этом подмножестве и нулю в противном случае), тогда из формулы следует:

$$\int_{-\infty}^{\infty} \mathbf{1}_B \mu(dt) = \mu(B) = P(X \in B).$$

Отметим, что в *главе 8* встречается еще одна теорема о замене переменных из теории вероятностей, которая связывает распределение вероятностей случайной величины с распределением вероятностей детерминированной функции от нее при помощи детерминанта якобиана преобразования этой функции.

Дальнейшие шаги в строгой теории вероятностей

Следующий шаг в строгой теории вероятностей — доказательство знаменитых неравенств (Маркова, Чебышева, Коши — Шварца, Йенсена), введение сумм и произведений случайных величин, законов больших чисел и центральной предельной теоремы. Затем мы переходим к последовательностям случайных величин и предельным теоремам.

Предельные теоремы

В случае когда последовательность случайных величин сходится к некоторой предельной случайной величине, следует ли из этого, что ожидания последовательности сходятся к ожиданиям предела? Говоря языком интегралов, когда мы можем поменять местами предел и интеграл?

В этом случае речь идет о доказательстве теорем о монотонной сходимости, ограниченной сходимости, лемме Фату, мажорируемой сходимости, равномерно интегрируемой сходимости.

В итоге мы рассматриваем двойные интегралы или интегралы более высоких степеней, а также условия, при которых допустимо перевернуть интегралы. На этот вопрос отвечает теорема Фубини, применив которую, можно получить формулу свертки для распределения суммы независимых случайных величин.

Универсальная теорема для нейросетей

Строгая теория меры (теория вероятностей) позволяет доказывать теоремы для нейросетей, которые относятся к развивающейся области математики, стремящейся обеспечить теоретическое обоснование многих эмпирических успехов ИИ.

Отправной точкой здесь служит универсальная теорема для нейросетей. Мы неоднократно ссылались на нее в книге. Ее суть заключается в следующем.

> Для любой непрерывной функции f на компактном множестве K существует полносвязная нейросеть только с одним скрытым слоем, которая равномерно аппроксимирует f с точностью до произвольного $\varepsilon > 0$ на K.

Перейдя по ссылке на эту веб-страницу (**https://oreil.ly/7A8Gn**), можно познакомиться с довольно неплохим и вполне убедительным доказательством.

Итоги и перспективы

В этой главе мы обсудили важные с точки зрения ИИ концепции вероятности, машинного обучения и науки о данных. Мы бегло рассмотрели такие темы, как каузальное моделирование, парадоксы, большие случайные матрицы, стохастические процессы и обучение с подкреплением в ИИ.

Часто при изучении вероятности можно наблюдать разделение на *"частотников"* и *"субъективистов"* относительно определений и общей философии, связанной с неопределенностью. Кратко опишем каждую точку зрения.

Позиция частотников.

> Вероятности можно получить только в результате экспериментов и наблюдения за результатами повторных испытаний.

Позиция субъективистов.

Вероятности — это реальные аспекты Вселенной, т. е. естественная предрасположенность или естественная тенденция вести себя определенным образом. Например, склонность подброшенной монеты в 50% случаев выпадать орлом — это неотъемлемое качество самой монеты.

В таком случае "частотник" пытается измерить эти естественные склонности экспериментальным путем. Разрозненные представления о вероятности объединены в строгой теории вероятностей. Кратко рассмотрев ее, мы выяснили, что по своей сути она аналогична теории меры в вещественном анализе. Таким образом, в итоге мы вышли на универсальную теорему аппроксимации для нейросетей.

В завершение главы приведем замечательный твит Яна Лекуна (**https://oreil.ly/X1eDh**), в котором затронуты все темы, рассмотренные в этой главе:

"На мой взгляд, требуются новые концепции, которые позволят машинам осуществлять такие вещи, как:

- узнавать, как устроен мир, наблюдая за ним по аналогии с детьми;
- учиться предсказывать, как можно повлиять на мир, предпринимая те или иные действия;
- изучать иерархические представления, позволяющие выполнять долгосрочные прогнозы в абстрактных пространствах представлений;
- правильно учитывать тот факт, что мир не полностью предсказуем;
- давать агентам возможность предсказывать последствия последовательностей действий, чтобы они могли рассуждать, а планирующие машины — осуществлять иерархическое планирование, разбивая сложную задачу на подзадачи.

Все это реализуется методами, совместимыми с градиентным обучением".

ГЛАВА 12
Математическая логика

Люди обходят правила.
— *Хала*

Исторически сложилось так, что в области ИИ логические агенты предшествуют агентам, основанным на машинном обучении и нейросетях. Причина, по которой мы рассмотрели машинное обучение, нейросети, вероятностные рассуждения, графовые представления и исследование операций прежде логики, заключается в том, что мы хотим связать их все в единое изложение рассуждений внутри агента, не рассматривая логику как нечто устаревшее, а нейросети — современное. Последние достижения рассматриваются нами как совершенствование способов логического представления и осмысления мира агентом ИИ. Это можно сравнить с просветлением: раньше агент ИИ опирался на жесткие правила базы знаний и закодированные вручную правила принятия выводов и решений, а потом он неожиданно просветлел и получил в свое распоряжение дополнительные инструменты рассуждений, сети и нейроны, которые позволили ему расширить как базу знаний, так и методы логического вывода. Таким образом, он обладает большей выразительной силой и может ориентироваться в более сложных и неопределенных ситуациях. Сочетание всех этих инструментов позволило бы агенту, подобно человеку, иногда нарушать правила строгой логической модели и использовать более гибкую конструкцию в зависимости от ситуации. Ведь умение обойти, нарушить и даже изменить правила является отличительной особенностью человека.

Словарное значение слова "логика" задает тон этой главе и служит обоснованием ее повествования.

Логика

Структура, организующая правила и процессы, используемые для обоснованного мышления и рассуждений. В ней заложены принципы обоснованности, на которых строятся рассуждения и делаются выводы.

Ключевыми словами, на которые следует обратить внимание в этом определении, являются структура и принципы, на основе которых делаются выводы. Логическая структура кодирует в агенте принципы, определяющие надежность умозаключений и правильность доказательств. Разработка агентов, способных накапливать знания, логически рассуждать с помощью гибкой логической структуры, учитывающей не-

определенность среды, в которой они существуют, а также делать выводы и принимать решения на основе этих логических рассуждений, лежит в основе ИИ. Мы рассмотрим различные структуры математической логики, которые можно запрограммировать в агенте. Задача заключается в том, чтобы наделить агента ИИ способностью делать выводы, которые позволят ему действовать соответствующим образом. Такие логические структуры требуют наличия баз знаний, сопровождающих правила вывода различного объема. Они также обладают разной степенью выразительности и дедуктивности.

Логические структуры

Для каждой логической структуры, рассматриваемой в этой главе (*пропозициональная логика, логика первого порядка, темпоральная логика, вероятностная логика, нечеткая логика*), мы ответим на два вопроса о том, как они работают в наделенном ими агенте:

1. Какие объекты существуют в мире агента? Иначе говоря, как агент воспринимает состав своего мира?
2. Как агент воспринимает состояния объектов? Другими словами, какие значения агент может присвоить каждому объекту своего мира в рамках конкретной логической схемы?

Это можно легко понять, если уподобить нашего агента муравью и его восприятию мира (**https://oreil.ly/MD8kn**). В силу предопределенных рамок восприятия и допустимых движений муравей воспринимает мир (с его кривизной) как двухмерный. Если муравья усовершенствовать, наделив более выразительными возможностями восприятия и допустимыми движениями (например, крыльями), то он будет воспринимать мир как трехмерный.

Пропозициональная логика

Ответим на вопросы относительно нашего агента.

Какие объекты существуют в мире агента?

Простые или сложные высказывания, называемые *пропозициями*, откуда и происходит название *пропозициональной логики*.

Как агент воспринимает состояния объектов?

Истинно (true, 1), ложно (false, 0) или неизвестно (unknown). Пропозициональная логика также называется булевой (бинарной) логикой, поскольку объекты в ней могут принимать только два состояния. Парадоксы в пропозициональной логике — это высказывания, которые не могут быть классифицированы как истинные или ложные в соответствии с *таблицей истинности* логической структуры.

Приведем примеры высказываний и их состояний.

- Идет дождь (может принимать истинное или ложное состояние).
- Эйфелева башня находится в Париже (всегда истинно).
- В парке наблюдается подозрительная активность (может принимать истинное или ложное состояние).
- Это предложение ложно (парадокс).
- Я счастлива *и* печальна (всегда ложно — пока не спросите у моего мужа).
- Я счастлива *или* печальна (всегда истинно).
- Если студент набрал 13 баллов, то он провалил экзамен (истинность зависит от порога сдачи экзамена, поэтому нам необходимо иметь в базе знаний утверждение, которое гласит: все студенты, набравшие меньше 16 баллов, не сдали экзамен, — и установить его значение истинным).
- 1 + 2 эквивалентно 2 + 1 (всегда истинно для агента, наделенного правилами арифметики).
- Париж романтичен. (В пропозициональной логике это значение должно быть либо истинным, либо ложным, но в нечеткой логике оно может принимать значение по шкале от нуля до единицы, например 0,8, что лучше соответствует тому, как мы воспринимаем наш мир: по шкале, в отличие от абсолютов. Конечно, если бы я программировала агента и ограничивалась пропозициональной логикой, я бы присвоила этому высказыванию значение "истинно", но тот, кто ненавидит Париж, присвоил бы "ложно". Ну и ладно.)

Объектами мира пропозициональной логики являются простые и сложные высказывания. Сложные высказывания можно образовывать из простых с помощью пяти допустимых операторов: *не* (NOT, отрицание), *и* (AND), *или* (OR), *имплицирует* (IMPLIES, $\rightarrow$; то же самое, что *"если..., то..."*) и *эквивалентно* (EQUIVALENT TO, $\leftrightarrow$, $\Leftrightarrow$; то же самое, что *"тогда и только тогда"*).

Существует также пять правил для определения истинности или ложности высказывания:

1. Отрицание высказывания истинно тогда и только тогда, когда оно ложно.
2. *Высказывание*$_1$ и *высказывание*$_2$ истинны тогда и только тогда, когда истинны оба.
3. *Высказывание*$_1$ или *высказывание*$_2$ истинно тогда и только тогда, когда истинно либо *высказывание*$_1$, либо *высказывание*$_2$ (или когда истинны оба).
4. *Высказывание*$_1$ имплицирует *высказывание*$_2$ — истинно, за исключением случая, когда *высказывание*$_1$ истинно, а *высказывание*$_2$ ложно.
5. *Высказывание*$_1$ эквивалентно *высказыванию*$_2$ тогда и только тогда, когда оба высказывания истинны или оба ложны.

Эти правила можно свести в *таблицу истинности* (табл. 12.1), учитывающую все возможные состояния *высказывания*$_1$ и *высказывания*$_2$ и их комбинаций, исполь-

зуя пять допустимых операторов. Для краткости *высказывание*$_1$ обозначим как $В_1$, а *высказывание*$_2$ — $В_2$.

Таблица 12.1. *Таблица истинности*

$В_1$	$В_2$	Не $В_1$	$В_1$ и $В_2$	$В_1$ или $В_2$	$В_1$ имплицирует $В_2$	$В_1$ эквивалентно $В_2$
Л	Л	И	Л	Л	И	И
Л	И	И	Л	И	И	Л
И	Л	Л	Л	И	Л	Л
И	И	Л	И	И	И	И

С помощью этой таблицы истинности можно вычислить истинность любого сложного высказывания путем простой рекурсивной оценки. Например, в мире, где высказывания $В_1$ и $В_3$ истинны, а $В_2$ ложно, получаем высказывание:

$$не\ В_1\ и\ (В_2\ или\ В_3) \Leftrightarrow Л\ и\ (Л\ или\ И) = Л\ и\ И = Л.$$

Для рассуждений и доказательства теорем с помощью пропозициональной логики целесообразно установить логические эквиваленции, т. е. высказывания, имеющие одинаковые таблицы истинности, чтобы они могли заменять друг друга в процессе рассуждений. Приведем примеры логических эквиваленций:

- коммутативность *и*: $В_1\ и\ В_2 \Leftrightarrow В_2\ и\ В_1$;
- коммутативность *или*: $В_1\ или\ В_2 \Leftrightarrow В_2\ или\ В_1$;
- устранение двойного отрицания: не(не $В_1$) $\Leftrightarrow В_1$;
- контрапозиция: $В_1$ *имплицирует* $В_2 \Leftrightarrow$ не($В_2$) имплицирует не($В_1$);
- устранение импликации: $В_1$ *имплицирует* $В_2 \Leftrightarrow$ не($В_1$) *или* $В_2$;
- закон де Моргана: не($В_1$ *и* $В_2$) $\Leftrightarrow$ не($В_1$) *или* не($В_2$);
- закон де Моргана: не($В_1$ *или* $В_2$) $\Leftrightarrow$ не($В_1$) *и* не($В_2$).

Продемонстрируем, что $В_1$ *имплицирует* $В_2 \Leftrightarrow$ не($В_1$) *или* $В_2$, показав, что они имеют одну и ту же таблицу истинности, поскольку для некоторых специалистов эта эквиваленция не столь очевидна (табл. 12.2).

Таблица 12.2. *Таблица истинности для доказательства случая устранения импликации*

$В_1$	не($В_1$)	$В_2$	не($В_1$) или $В_2$	$В_1$ имплицирует $В_2$
Л	И	Л	И	И
Л	И	И	И	И
И	Л	И	И	И
И	Л	Л	Л	Л

Одним из примеров, демонстрирующих полезность логических эквиваленций, является способ рассуждения, который называется *доказательством от противного*. Для того чтобы доказать, что высказывание B_1 имплицирует высказывание B_2, можно предположить, что у нас есть B_1, но в то же время нет B_2; тогда мы приходим к чему-то ложному или абсурдному, что доказывает, что нельзя предположить B_1, не сделав вывод и о B_2. Мы можем убедиться в достоверности данного способа доказательства того, что B_1 имплицирует B_2, с помощью пропозициональных логических эквиваленций:

- B_1 *имплицирует* B_2 = истинно $\Leftrightarrow$
- не(B_1) *или* B_2 = истинно (устранение импликации) $\Leftrightarrow$
- не(не(B_1) *или* B_2) = не(истинно) $\Leftrightarrow$
- B_1 *и* не(B_2) = ложно (закон де Моргана и двойного отрицания).

Для того чтобы иметь возможность последовательно рассуждать, переходя от одного высказывания (простого или сложного) к другому, и достичь поставленной цели или правильного доказательства высказывания, необходимо наделить пропозициональную логическую структуру *правилами вывода*. Перечислим несколько правил вывода, присущих пропозициональной логике.

- Если B_1 *имплицирует* B_2 истинно, и дано B_1, то можно сделать вывод о B_2.
- Если B_1 *и* B_2 истинны, то можно сделать вывод о B_1. Аналогичным образом можно сделать вывод и о B_2.
- Если B_1 *эквивалентно* B_2, то можно сделать вывод о том, что (B_1 *имплицирует* B_2) *и* (B_2 имплицирует B_1).
- И наоборот, если (B_1 *имплицирует* B_2) *и* (B_2 *имплицирует* B_1), то можно сделать вывод, что (B_1 *эквивалентно* B_2).

В заключение подчеркнем, что пропозициональная логика не рассчитана на большие среды и не может эффективно отражать универсальные модели отношений. Однако она является основой логики первого порядка и логики высших порядков, поскольку последние опираются на механизмы пропозициональной логики.

От нескольких аксиом к целостной теории

Правила вывода *обоснованы*. Они позволяют доказывать только истинные высказывания в том смысле, что если дано истинное высказывание и если можно вывести из него обоснованное правило вывода, то мы приходим к истинному высказыванию. Таким образом, гарантия, которую дают обоснованные правила вывода, заключается в том, что они не позволяют выводить ложные высказывания из истинных. Однако требуется что-то большее, чем гарантия.

Логическая структура является *полной*, если мы можем вывести *все* возможные истинные высказывания, используя только базу знаний (аксиомы) системы и ее правила вывода. Идея *полной* системы очень важна. Во всех математических системах, таких как теория чисел, теория вероятностей, теория множеств или евклидова геометрия, мы начинаем с набора аксиом (аксиомы Пеано для теории чисел и математического анализа и аксиомы вероятностей для теории вероятностей), а затем из них выводим теоремы, используя логические правила вывода. Одним из главных вопросов любой математической теории является вопрос о том, обеспечивают ли аксиомы наряду с правилами вывода ее полноту и непротиворечивость.

Однако никакая теория первого порядка не способна однозначно описать структуру с бесконечной областью, такую как натуральные числа или вещественная прямая. Системы аксиом, которые полностью описывают эти две структуры (т. е. категориальные системы аксиом), могут быть получены в более сильных логиках, таких как логика второго порядка.

Кодирование логики внутри агента

Прежде чем мы перейдем к логике первого порядка, напомним, что мы узнали в контексте агента ИИ, наделенного пропозициональной логикой. Ниже приведен важный процесс, который будет таким же и для более выразительных логик:

1. Программируем исходную базу знаний (аксиомы) в виде истинных утверждений.
2. Программируем правила вывода.
3. Агент воспринимает определенные утверждения о текущем состоянии своего мира.
4. Агент может иметь, а может и не иметь цели.
5. Используя правила вывода, агент выводит новые утверждения и решает, что делать (перейти в следующую комнату, открыть дверь, завести будильник и т. д.).
6. Важную роль здесь играет полнота системы агента (базы знаний в сочетании с правилами вывода), которая при достаточно большом количестве шагов вывода позволяет агенту вывести *любую* приемлемую формулировку цели.

Как сочетаются детерминированное и вероятностное машинное обучение?

Суть машинного обучения (в том числе и нейросетей) заключается в том, что мы не программируем ни исходную базу знаний в агенте, ни правила вывода. Вместо этого мы программируем способ представления входных данных и требуемых выходов, а также функцию гипотезы, которая отображает входные данные на выход. Затем агент узнает параметры функции, оптимизируя целевую функцию (функцию потерь). Наконец, он делает выводы по новым входным данным, используя функцию, которой он обучился. Таким образом, базу знаний и правила можно разделить в *процессе обучения* или в *процессе вывода*. В процессе обучения базой знания служат данные и функция гипотез, целью — минимизация потерь, а правила —

процесс оптимизации. После обучения агент использует изученную функцию для вывода.

Аналогичным образом можно представить вероятностные модели машинного обучения, заменив детерминированную функцию гипотезы на совместное распределение вероятностей признаков данных. После обучения агент может использовать его для выводов. Например, байесовские сети будут играть ту же роль для неопределенных знаний, что и пропозициональная логика для определенных.

Логика первого порядка

Рассмотрим логику первого порядка, ответив на те же самые вопросы.

Какие объекты существуют в мире агента?

Высказывания, объекты и отношения между ними.

Как агент воспринимает состояния объектов?

Истинно (1), ложно (0) или неизвестно.

Для иллюстрации работы агентов, основанных на знаниях, а также для объяснения базовых правил *языка* конкретной логики и правил вывода отлично подходит пропозициональная логика. Однако она ограничена в том, какие знания представлять и как о них рассуждать. Например, в пропозициональной логике высказывание

"Все пользователи старше 18 лет могут видеть эту рекламу"

можно легко выразить в виде *импликации* (аналогично *"если..., то..."*), поскольку такой язык существует в системе пропозициональной логики. Таким образом, в пропозициональной логике это высказывание можно выразить в виде логического вывода:

(Пользователи старше 18 лет *имплицируют* просмотр рекламы) *и* (Пользователи старше 18 лет = И), то можно сделать вывод, что (просмотр рекламы = И).

Теперь рассмотрим несколько иное высказывание:

"Некоторые пользователи старше 18 лет щелкают по рекламе".

Неожиданно в этом высказывании языка пропозициональной логики оказывается недостаточно для выражения количества "некоторые"! Агент, опирающийся исключительно на пропозициональную логику, должен будет хранить все высказывание целиком в своей базе знаний, не зная, как извлечь оттуда что-нибудь полезное. То есть если агент получит информацию о том, что пользователь действительно старше 18 лет, он не сможет предсказать, щелкнет ли тот на рекламе или нет.

Нужен язык (или логическая структура), словарный запас которого включает *квантификаторы*, в частности *квантор существования* "существует" (англ. *there exist*) и *квантор всеобщности* "для всех" (англ. *for all*), и тогда мы сможем написать нечто вроде:

"*Для всех* пользователей старше 18 лет *существует* подмножество тех, кто щелкает по рекламе".

Именно эти два дополнительных квантификатора и предоставляет *логика первого порядка*.

Такое расширение словарного запаса обеспечивает более экономичный подход к содержимому базы знаний, позволяя разбить знания на объекты и отношения между ними. Например, вместо того чтобы хранить

"Все пользователи старше 18 лет видят рекламу.

Некоторые из пользователей старше 18 лет щелкают по рекламе.

Некоторые из пользователей старше 18 лет покупают товар.

Некоторые из пользователей, щелкнувших по рекламе, покупают товар."

как четыре отдельных высказывания в базе знаний агента, владеющего только пропозициональной логикой (и мы до сих пор не знаем, как вывести что-нибудь полезное из нее), можно сохранить три утверждения в логике первого порядка:

"*Для всех* пользователей старше 18 лет посмотреть рекламу = И.

Для всех пользователей со значением "посмотреть рекламу = И" *существует* поднабор тех, кто щелкает по рекламе.

Для всех пользователей, щелкнувших по рекламе, *существует* поднабор тех, кто покупает товар".

Важно отметить, что и в пропозициональной логике, и в логике первого порядка, имея только эти высказывания, невозможно сделать вывод о том, щелкнет ли конкретный пользователь старше 18 лет по рекламе или купит товар, или даже о проценте таких пользователей, но, по крайней мере, в логике первого порядка существует язык для более краткого выражения тех же знаний, причем так, чтобы мы могли сделать некоторые полезные выводы.

Главное отличие логики первого порядка от пропозициональной логики состоит в том, что она добавляет к своему базовому языку кванторы существования (*существует*) и всеобщности (*для всех*) помимо уже присутствующих в пропозициональной логике операторов *не*, *и*, *или*, *имплицирует*, *эквивалентно*. Такое небольшое дополнение дает возможность выражать объекты отдельно от их описания и отношений друг с другом.

Преимущество пропозициональной логики и логики первого порядка заключается в том, что их правила вывода не зависят ни от области, ни от ее базы знаний или набора аксиом. Для того чтобы создать базу знаний в конкретной области, например в математике или схемотехнике, необходимо досконально изучить ее, выбрать словарь, а затем сформулировать набор аксиом, необходимых для поддержки нужных выводов.

Отношения между кванторами существования и всеобщности

Квантор существования и квантор всеобщности связаны друг с другом посредством отрицания. Следующие два утверждения эквивалентны:

- все пользователи старше 18 лет видят рекламу;
- не существует ни одного пользователя старше 18 лет, кто не видит эту рекламу.

На языке пропозициональной логики это означает:

- для всех пользователей, для кого "пользователь > 18" истинно, "видеть рекламу" истинно;
- не существует такого пользователя, для которого "пользователь > 18", а "видеть рекламу" — ложь.

Запишем отношения:

- не(существует x, так что P истинно) ⇔ для всех x значение P — ложь;
- не(для всех x значение P истинно) ⇔ существует x, так что P — ложь;
- существует x, так что P истинно ⇔ не для всех x значение P — ложь;
- для всех x значение P — истина ⇔ не существует x, так что P — ложь.

Мы не можем закончить этот раздел без оценки выразительной силы, которую получаем с переходом к логике первого порядка. Теперь этой логической структуры достаточно, чтобы подобные утверждения и умозаключения обрели смысл.

Универсальная теорема аппроксимации для нейросетей.

Грубо говоря, универсальная теорема аппроксимации утверждает, что *для всех* непрерывных функций *существует* нейросеть, способная аппроксимировать эту функцию настолько близко, насколько нам нужно. Важно, что это не говорит нам о том, как построить такую сеть, а лишь утверждает ее существование. Тем не менее эта теорема достаточно сильна, чтобы мы могли *не удивляться* успеху нейросетей в аппроксимации всех видов функций от входа к выходу во всех видах приложений.

Вывод отношений.

Родители и дети находятся в обратных отношениях друг с другом: если Сари — ребенок Халы, то Хала — мать Сари. Более того, эти отношения имеют одно направление: Сари не может быть матерью Халы. В логике первого порядка можно задать две функции, указывающие на отношения: это функции *матери* и *ребенка*, т. е. переменные, в качестве которых могут выступать Хала и Сари или любые другие мать и ребенок, а также отношение между *функциями*, которое выполняется для всех входных переменных:

Для всех x, y, если мать(x, y) = И, то мать(y, x) = Л;

и *для всех* x, y мать(x, y) ⇔ ребенок(y, x).

Теперь, если снабдить агента этими знаниями и указать, что Хала — мать Сари, или мать(Хала, Сари) = И, то он сможет отвечать на такие запросы, как:

- Является ли Хала матерью Сари? И.
- Является ли Сари матерью Халы? Л.
- Является ли Сари ребенком Халы? И.
- Является ли Хала ребенком Сари? Л.
- Является ли Лаура матерью Иосифа? Неизвестно.

Важно отметить, что в системе пропозициональной логики каждое высказывание приходится хранить отдельно, что особенно неэффективно.

Вероятностная логика

Какие объекты существуют в мире агента?

Высказывания.

Как агент воспринимает состояния объектов?

Вероятность того, что высказывание истинно, находится в диапазоне от 0 до 1.

Вероятность — это расширение логики первого порядка, которое позволяет нам количественно оценить неуверенность в истинности высказывания. Вместо того чтобы утверждать об истинности или ложности высказывания, мы присваиваем оценку от нуля до единицы степени доверия к истинности высказывания. Пропозициональная логика и логика первого порядка предоставляют набор правил вывода, при помощи которых можно определить истинность некоторых высказываний, допуская наличие других истинных высказываний. Теория вероятностей обеспечивает набор правил вывода, позволяющий определить вероятность истинности высказывания с учетом вероятности истинности других высказываний.

Такое расширение возможностей работы с неопределенностью дает более выразительную структуру, чем логика первого порядка. Аксиомы вероятности позволяют расширить традиционные логические таблицы истинности и правила вывода. Например, $P(A) + P(\text{не } (A)) = 1$: если A истинно, то $P(A) = 1$ и $P(\text{не } A) = 0$, что соответствует логике первого порядка относительно высказывания и его отрицания.

Взгляд на теорию вероятностей как на естественное расширение теории первого порядка вполне удовлетворяет ум, которому приходится связывать вещи воедино, а не рассматривать их как нечто разрозненное. Такой взгляд закономерно приводит к байесовским рассуждениям о данных, т. к. по мере накопления знаний мы обновляем предварительное распределение агента и делаем более точные выводы. Благодаря этому все предметы связываются воедино самым *логичным* образом.

Нечеткая логика

Какие объекты существуют в мире агента?

Высказывания со степенью истины в диапазоне [0; 1].

Как агент воспринимает состояния объектов?

Как известное значение интервала.

Миры пропозициональной логики и логики первого порядка — это черное и белое, истинное или ложное. Благодаря им можно начинать с истинных высказываний и

выводить из них другие истинные высказывания. Такие условия идеально подходят для математики, где все бывает либо верным, либо не верным (истинным или ложным), или для видеоигр с очень четкими границами в своих симуляторах жизни типа SIMS. В реальном мире многие высказывания обладают неопределенностью относительно того, являются ли они полностью истинными (1) или полностью ложными (0), т. е. они существуют на *шкале истинности*, а не на ее краях, например, *"Париж романтичен"*; *"Она счастлива"*; *„Темный рыцарь" — отличный фильм"*. Этому способствует *нечеткая логика*, которая вместо строгих 0 или 1 присваивает высказываниям значения от 0 до 1: *"Париж романтичен"* (0,8); *"Она счастлива"* (0,6); *„Темный рыцарь" — отличный фильм"* (0,9).

Как делать выводы в неопределенном мире, где истина находится на скользящей шкале? Безусловно, это далеко не так просто, как в системе истинности и ложности. Например, насколько будет верным утверждение: "Париж романтичен, *и* она счастлива", учитывая предыдущие значения истинности? Нам нужны новые правила присвоения таких значений, а также необходимо знать контекст, или домен. Еще один вариант — векторы слов, о которых говорится в *главе 7*. Эти векторы передают значение слов в различных измерениях, так что можно вычислить *косинусное сходство* между вектором, представляющим слово *"Париж"*, и вектором, представляющим точку *"романтичный"*, и присвоить полученное значение в качестве истинного значения высказывания *"Париж романтичен"*.

Важно отметить, что степень доверия в теории вероятностей — это не то же самое, что шкала истинности в нечеткой логике. В вероятностной логике сами высказывания однозначны. Нам нужно получить вывод о вероятности того, что однозначное высказывание истинно. Теория вероятностей не рассуждает о полностью истинных или ложных высказываниях. Мы вычисляем вероятность не того, что Париж романтичен, а того, что ответ на случайный вопрос о том, романтичен ли Париж, будет истинным или ложным.

Нечеткая логика интересна тем, что она отбрасывает два принципа, присутствующие в других логиках: это принцип, согласно которому, если высказывание истинно, то его отрицание ложно, и принцип, согласно которому два противоречивых высказывания не могут быть истинными одновременно. Это фактически открывает дверь, ведущую в непоследовательность и *открытую Вселенную*. В некотором смысле нечеткая логика не пытается исправить неясность; вместо этого она принимает ее и использует для функционирования в мире, границы которого стерты.

Темпоральная логика

Существуют и другие типы логик особого назначения, в которых особое внимание уделяется конкретным объектам, в частности *времени*, как в текущем разделе, и у которых есть свои аксиомы и правила вывода, т. к. они являются центральными для представляемых знаний и рассуждений о них. *Темпоральная логика* (или временна́я

логика) отводит временно́й зависимости, а также аксиомам и правилам вывода о ней центральное место в своей структуре, а не просто добавляет в базу знаний высказывания, содержащие информацию о времени. В темпоральной логике высказывания или факты бывают истинными в определенные моменты времени, которые могут быть временны́ми точками или временны́ми интервалами, и эти моменты времени упорядочены.

Какие объекты существуют в мире агента?

Высказывания, объекты, отношения, времена.

Как агент воспринимает состояния объектов?

Истинно (1), ложно (0) или неизвестно.

В темпоральной логике можно представить, например, такие утверждения:

- будильник срабатывает в 7:00 утра;
- когда на сервер поступает запрос, доступ в конечном итоге предоставляется, причем его невозможно предоставить одновременно по двум запросам.

Сравнение с естественным человеческим языком

Мы посвятили целую главу логическим структурам, способным выражать знания, которые естественный язык человека, кажется, делает без усилий. Я написала целую книгу о математике, используя только английский язык, в отличие от любого другого технического языка. Как это делается? Как люди представляют и расширяют свою базу знаний и какие правила использует естественный язык для представления и рассуждений, благодаря чему он может быть настолько выразительным? И более того, не важно, какой именно естественный язык при этом используется: каждый, кто владеет несколькими языками, знает, о чем он говорит, но не всегда понимает, на каком конкретном языке он выражает свою мысль. Существует внутреннее невербальное представление того, что люди знают или хотят выразить. Как оно работает и как раскрыть его секреты и передать их машинам?

По аналогии с человеческим языком, если представить одно и то же знание в двух разных формальных логиках, можно вывести одни и те же факты (при условии, что логики имеют правила полноты вывода). Единственное различие будет заключаться в выборе логической схемы, обеспечивающей самый легкий путь для вывода.

Тем не менее во многих случаях естественный человеческий язык допускает неоднозначность и не может делать абсолютные математические утверждения без формальной математики и формальной логики, которую она использует. Мы не можем попросить человека, не имеющего доступа к системе GPS, предсказать точное время, необходимое для поездки в конкретный день из Вашингтона в Нью-Йорк, но можно добиться такой точности от машины GPS.

Машины и сложные математические рассуждения

Человеческая логика реализована в математических рассуждениях. Доказывая теорему посредством математических рассуждений, мы становимся на шаг ближе к *универсальной истине*. Для обучения машин доказательству математических теорем и, что еще более амбициозно, для генерации новых теорем требуется навигация по бесконечным пространствам поиска и *символических рассуждений*. И снова нейросети демонстрируют свою эффективность в развитии разумных машин. Исследователи из компании Meta, Амстердамского свободного университета и Парижского института науки и техники использовали комбинацию глубокого обучения, онлайнового машинного обучения (**https://oreil.ly/k__ZN**), трансформеров (больших языковых моделей) и обучения с подкреплением для автоматического математического доказательства теорем. В работе "Гипердерево поиска доказательств для нейронных доказательств теорем" (2022 г., **https://oreil.ly/NzIvp**) они представили новейшие достижения в этой области.

Итоги и перспективы

Агент искусственного интеллекта, наделенный различными типами логики, способен выражать знания о мире, рассуждать о нем, отвечать на запросы и делать выводы, допустимые в границах этих логик.

Мы обсудили различные логические структуры, включая пропозициональную логику, логику первого порядка, вероятностную логику, нечеткую логику и темпоральную логику.

И здесь естественным образом возникает ряд вопросов. Каким должно быть содержание базы знаний агента? Как представлять факты о мире? В каких системах нужно представлять знания и делать выводы?

- Пропозициональная логика?
- Логика первого порядка?
- Иерархические сети задач для рассуждений о планах?
- Байесовские сети для рассуждений о неопределенности?
- Причинно-следственные диаграммы и причинные рассуждения, где агенту разрешено выборочно нарушать правила логики?
- Марковские модели для рассуждений о времени?
- Глубокие нейросети для рассуждений об изображениях, звуках и других данных?

Следующим возможным шагом будет углубленное изучение любой из рассмотренных нами логических структур, выяснение их правил и существующих алгоритмов вывода, а также сильных и слабых сторон и того, к каким базам знаний они приме-

нимы. Повторяющейся темой в этих исследованиях является изучение правил вывода, обеспечивающих полную систему доказательства, т. е. систему, в которой аксиомы или база знаний в сочетании с правилами позволяют доказать *все* вероятные истинные утверждения. К таким правилам относятся *правило резолюций* для пропозициональной логики и *обобщенное правило резолюций* для логики первого порядка, предназначенные для специальных баз знаний. Все они важны как с теоретической точки зрения (доказательство математических теорем), так и с точки зрения технологий (верификация и синтез) в сфере программного и аппаратного обеспечения. Наконец, некоторые логики *в строгом смысле более выразительны* по сравнению с остальными в том смысле, что отдельные высказывания, которые можно представить в более выразительной логике, невозможно выразить любым конечным числом высказываний на языке менее выразительной логики. Например, *логика высшего порядка* (ее мы не рассматривали в этой главе) в строгом смысле считается более выразительной, чем логика первого порядка (которую мы рассматривали в этой главе и которая является достаточно мощной для поддержки целых математических теорий).

ГЛАВА 13

Искусственный интеллект и дифференциальные уравнения в частных производных

Я хочу смоделировать весь мир.
— Хала

В первой сцене фильма "Топ Ган: Мэверик" (2022) главный герой по имени Мэверик (Том Круз), управляя экспериментальным военным самолетом, разгоняет его до скорости, в 10 раз превышающей скорость звука (10 Махов), после чего теряет устойчивость на скорости около 10,2 Маха. В действительности самый скоростной пилотируемый самолет может достигать сегодня скорости 6,7 Маха (рис. 13.1). И неважно, реальная такая скорость или нереальная (пока), — нас поистине впечатляет наблюдать за тем, как благодаря сочетанию физики, математики и техники самолеты поднимаются в небо, а особенно, когда они начинают проделывать в воздухе свои головокружительные маневры.

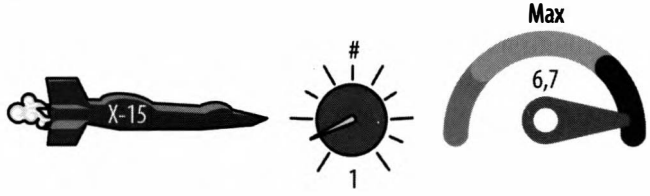


Рис. 13.1. Самый быстрый в истории пилотируемый самолет (источник изображения: **https://oreil.ly/CKMN2**)

Приведем несколько дифференциальных уравнений в частных производных, которые приходят на ум при просмотре потрясающих сцен "Топ Ган: Мэверик" с собачьими боями и скоростью 10 Махов.

Волновое уравнение для распространения волн.

Имеет дело со скоростью звука, распространением звуковой волны в воздухе, изменением скорости звука на разных высотах вследствие изменения температуры и плотности воздуха.

Уравнения Навье — Стокса для гидродинамики.

Имеет дело с потоком жидкости, воздушными туннелями, турбулентностью.

Уравнение G для горения.

Имеет дело со сгоранием топлива в двигателе самолета и пламенем, выходящим из выхлопных труб самолета.

Уравнение упругости материалов.

Имеет дело с панелью крыла самолета, подъемной силой, продольным изгибом панели крыла (движение самолета без нагрузки, происходящее при компрессии; рис. 13.2), возникающим под нагрузкой, что, в свою очередь, снижает несущую способность крыла. Когда несущая способность падает ниже расчетных пределов, происходит сбой.

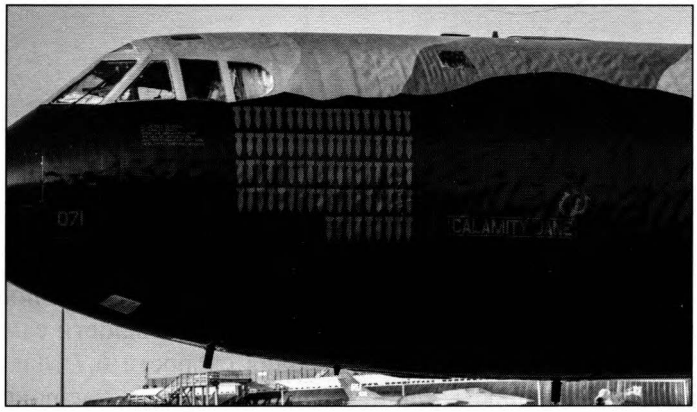


Рис. 13.2. Продольный изгиб в самолете (источник изображения: **https://oreil.ly/RJPpO**)

Также приходит на ум моделирование дифференциальных уравнений в частных производных. Достаточно вспомнить моделирование траектории полета и общение экипажа с Мэвериком, когда они наблюдают в реальном времени за его полетом на экранах своих компьютеров.

Этот список можно легко продолжить. Но возможно ли утверждать, что самолеты стали летать благодаря тому, что для них записаны и решены дифференциальные уравнения в частных производных? Нет. В музеях авиации можно ознакомиться с историей братьев Райт, их экспериментами и развитием авиационной промышленности. Наука и эксперименты идут рука об руку. Мы просто хотим сказать, что благодаря дифференциальным уравнениям и математике мы имеем возможность изобретать, улучшать и оптимизировать разнообразные конструкции.

Что собой представляет дифференциальное уравнение в частных производных?

Дифференциальное уравнение в частных производных (partial differential equation, PDE) — это уравнение (т. е. его левая сторона равна правой), содержащее функцию

нескольких переменных и их частные производные. Частная производная функции переменной определяет скорость изменения функции по отношению к этой переменной. *Обыкновенные дифференциальные уравнения* (ОДУ — ordinary differential equations, ODE) — это уравнения с функциями только от одной переменной, например только времени, только пространства и т. д. (в отличие от нескольких переменных), и их производными. Важнейшим ОДУ, описывающим изменение во времени состояния интересующей нас системы (например, системы частиц или состояния клиента в бизнесе), является динамическая система. ОДУ содержит одну производную состояния системы по времени, а динамика задается как функция состояния системы, ее физических параметров и времени. Оно имеет вид

$$\frac{d\vec{x}(t)}{dt} = f\left(\vec{x}(t), a(t), t\right).$$

В этой главе мы будем неоднократно обращаться к динамическим системам. В большинстве случаев, если нам удается преобразовать дифференциальное уравнение в частных производных в систему обыкновенных дифференциальных уравнений или, возможно, в динамическую систему, то можно считать, что задача более или менее решена.

Природа не поделилась с нами ни детерминированными функциями, ни совместными распределениями вероятностей, участвующих в создании мира, который мы наблюдаем вокруг себя и можем точно измерить. До недавнего времени она держала это в секрете. Тем не менее она предоставила нам способы измерения, оценки и даже установления законов о том, *как изменяются вещи* относительно друг друга, что, собственно, и представляют собой дифференциальные уравнения. Ведь *то, как изменяются вещи*, — это и есть производные.

Цель решения дифференциального уравнения в частных производных — отменить *дифференциальный оператор*, так чтобы можно было восстановить функцию без производных. Таким образом, мы ищем точный или приближенный оператор, обратный (или псевдообратный) дифференциальному оператору, относящемуся к дифференциальному уравнению в частных производных. Вследствие того что интегралы отменяют производные, представления решений дифференциального уравнения в частных производных часто содержат интегралы некоторых функций ядра по входным данным уравнения (его параметрам, начальным и/или граничным условиям). Мы будем подробно останавливаться на этих моментах по ходу главы.

Часто дифференциальные уравнения в частных производных и ОДУ классифицируются по *типам*. На мой взгляд, не стоит путаться в *классификациях*, если только мы лично не работаем с этими особыми уравнениями, а их решения не оказывают прямого немедленного воздействия на будущее человечества. Когда по ходу главы появляется какой-либо определенный тип, например нелинейное параболическое или обратное стохастическое уравнение, нужно просто принять этот термин, обращая внимание на главную мысль, которую я стараюсь донести. И даже не пытайтесь гуглить эти термины. Это все равно, что набрать в Google свои симптомы и узнать, что завтра вы умрете. Считайте, что я вас предупредила.

Моделирование с помощью дифференциальных уравнений

Дифференциальные уравнения моделируют бесчисленные явления реального мира: турбулентность воздуха, движение галактик, поведение материалов в наномасштабах, ценообразование финансовых инструментов, игры с соперниками и множеством игроков, мобильность и рост населения. В типовых курсах по дифференциальным уравнениям шаг моделирования, как правило, упускается, так что создается впечатление, будто итоговые уравнения возникают из ниоткуда, хотя это вовсе не так. Откуда берутся дифференциальные уравнения в частных производных — не менее важно, чем попытки их анализа и решения. Обычно эти уравнения выражают определенные законы сохранения, например сохранение энергии, массы, импульса и т. д., применительно к конкретной задаче. Многие из них являются выражением утверждения о сохранении, которое выглядит как:

скорость изменения величины во времени = выгода – затраты.

Теперь дифференциальное уравнение в частных производных работает внутри ограниченной области, но его нужно сопровождать *граничными условиями*, указывающими на то, что именно происходит на границе области. Если область не ограничена, то необходимы *условия дальнего поля*, которые говорят, что происходит при $x \to \infty$. Эти условия записываются в терминах пределов. Если дифференциальное уравнение в частных производных имеет производные по времени, то необходимы условия начального или конечного времени. Количество таких условий зависит от порядка уравнения, т. е. сколько уравнений для скольких неизвестных требуется решить. Здесь неизвестными будут *константы интегрирования* дифференциальных уравнений в частных производных. Решая эти уравнения, мы ищем информацию о функции с учетом информации о ее производных. Для того чтобы избавиться от этих производных и восстановить функцию, нужно *интегрировать* дифференциальное уравнение в частных производных, попутно получая константы интегрирования. Для решения этих констант требуются граничные условия и/или условия дальнего поля.

Модели различных масштабов

Реалистичные модели, точно имитирующие природу, должны учитывать все важные переменные и их взаимодействие, иногда в разных масштабах пространства и времени. Прежде чем записать уравнения для математических моделей, требуется проделать определенную работу. После формулировки они обретают изящный вид, сжимая целый массив информации в несколько строк уравнений. Такие уравнения содержат функции, их производные и параметры модели, и их, как правило, сложнее решить, чем сформулировать. Более того, если две модели описывают одно и то же явление в разных масштабах, например одна — в атомистическом масштабе (быстро движущиеся молекулы), а другая — в более крупном масштабе, скажем в микроскопическом или макромасштабе (наблюдаемый нами), то уравнения этих

двух моделей будут выглядеть совершенно по-разному, причем они даже могут опираться на физические законы из разных областей науки. Можно представить, например, описание движения газов на молекулярном уровне (скорость частицы, ее положение, действующие на нее силы и т. д.) и подумать о том, как это связать с термодинамикой газообразной системы, наблюдаемой в макроскопическом масштабе. Или как атомы соединяются друг с другом, образуя кристаллические структуры, и как эти структуры преобразуются в свойства материалов, такие как проводимость, проницаемость, хрупкость и т. д. И здесь возникает естественный вопрос: возможно ли совместить эти модели при том, что каждая из них функционирует, причем в той или иной степени успешно, в другом масштабе? Точнее, если перенести предел одной модели в режим другой, то получится ли то же самое? Именно такими вопросами занимаются аналитики. Совмещение моделей разного масштаба подтверждает их достоверность и объединяет различные области математики и науки.

Параметры дифференциального уравнения в частных производных

Как правило, в дифференциальных уравнениях в частных производных, записанных для модели, содержатся параметры. Эти параметры связаны со свойствами моделируемой физической системы. Например, в уравнении теплопроводности

$$u_t(\vec{x}, t) = \alpha \Delta u(\vec{x}, t)$$

параметр α — это *коэффициент диффузии*, который является физической константой, зависящей от свойств диффундирующего вещества и свойств среды, в которую оно диффундирует. Чаще всего мы берем их из справочных таблиц, составленных по результатам экспериментов. Их значения очень важны для инженерно-технических целей. В уравнениях, моделирующих реальность, необходимо использовать значения параметров, полученные из реальных экспериментальных или наблюдательных данных. Однако экспериментальные данные часто бывают зашумленными, с пропущенными значениями, необъяснимыми выбросами и всевозможными неровностями для математических моделей. В большинстве случаев у нас даже нет экспериментальных значений для параметров, входящих в уравнения. Эксперименты могут оказаться дорогими (вспомним, к примеру, Большой адронный коллайдер) или даже невозможными. Поэтому приходится *находить* значения параметров косвенными путями, основываясь на доступных комбинациях экспериментальных, наблюдательных и смоделированных на компьютере значений некоторых других переменных. Исторически сложилось так, что многие из этих значений параметров *подбирались вручную* для соответствия какому-то желаемому результату, что вовсе не годится! Необходимо четко обосновывать выбор значений параметров, использующихся в симуляциях. Мы увидим, как машинное обучение позволяет получать для дифференциальных уравнений в частных производных значения параметров из данных.

Как изменение одного параметра в дифференциальном уравнении в частных производных может обернуться серьезными проблемами

Вспоминая дифференциальные уравнения в частных производных из институтского курса, можно привести простейшее — уравнение распространения тепла в стержне (рис. 13.3). Даже если вы не изучали дифференциальные уравнения, не стоит беспокоиться о его деталях. Формула уравнения теплопроводности имеет вид:

$$u_t(x, t) = \alpha \Delta u(x, t).$$

Здесь $u(x, t)$ — температура в точке x стержня в момент времени t, а оператор Δ — вторая производная по x (так, что $\Delta u(x, t) = u_{xx}(x, t)$), поскольку стержень одномерен, если пренебречь его толщиной. В более высоких измерениях оператор Δ представляет собой сумму вторых производных в каждом из измерений.

Теперь изменим область со стержня на пластину неправильной формы — не квадрат, круг или эллипс, а некую неправильную фигуру (например, сравним рис. 13.3 с рис. 13.4). Формула для истинного решения (которая изучается на вводном курсе по дифференциальным уравнениям в частных производных), которая работает для стержня, больше не работает для неправильной пластины. Все становится еще хуже. Меняя область, мы не только теряем доступ к аналитическому решению, — когда мы пытаемся численно решить дифференциальное уравнение с новой областью, новая геометрия неожиданно усложняет ситуацию.

Таким образом, в первую очередь необходимо найти *дискретную сетку*, точно отображающую в деталях форму новой области, а затем получить на ней численное решение, удовлетворяющее уравнению во внутренней части области и граничным условиям вдоль границы неправильной формы.

Рис. 13.3. Изучение распространения тепла в стержне не представляет сложности (для того, кто изучал дифференциальные уравнения в частных производных) как в аналитическом, так и в численном плане

Рис. 13.4. Изучение распространения тепла на неровной геометрии дается не так легко

Такое положение вещей считается нормальным в системах дифференциальных уравнений в частных производных: стоит изменить одну крошечную деталь, как

внезапно все ранее изученные математические методы могут оказаться неприемлемыми. К таким изменениям относятся:

- изменение формы области;
- изменение типа граничных условий;
- ввод пространственной или временно́й зависимости в коэффициенты (параметры);
- ввод нелинейности;
- ввод выражений с большим количеством производных (более высокий порядок);
- ввод большего количества переменных (более высокая размерность).

Этот досадный аспект отталкивает многих студентов от специализации по дифференциальным уравнениям в частных производных (никому не хочется быть экспертом только в одном уравнении, которое изначально кажется весьма далеким от моделирования реальности). Но мы не отвернемся от них. Нам нужно увидеть полную картину.

Природные явления удивительно разнообразны, поэтому необходимо принять вариативность дифференциальных уравнений в частных производных и методов их решения как составляющую стремления понять и предсказать природу. Более того, с давних пор дифференциальные уравнения в частных производных представляют собой обширнейшую область, в которой был достигнут значительный прогресс в унификации методов для многих семейств как линейных, так и нелинейных дифференциальных уравнений в частных производных, и попутно было найдено множество мощных методов анализа. На сегодняшний день положение дел таково, что дифференциальные уравнения в частных производных — это довольно ценная область, которая не имеет и, возможно, никогда не будет иметь объединяющей теории.

В целом нелинейные дифференциальные уравнения в частных производных сложнее линейных, уравнения высшего порядка сложнее уравнений низшего порядка, уравнения высшей размерности сложнее уравнений низшей размерности, системы уравнений сложнее, чем одно, и мы не можем написать явные формулы решений для большинства из них, а многие уравнения удовлетворяются только в *слабой* форме. С течением времени решения большинства дифференциальных уравнений в частных производных переходят в сингулярность (например, волновое уравнение и ударные волны). Математики, разрабатывающие теорию дифференциальных уравнений в частных производных, занимаются доказательством *существования* решений и пытаются понять их *регулярность*, т. е. насколько они хороши с точки зрения наличия участвующих в уравнении производных. При этом используется множество передовых методов исчисления, позволяющих получить оценки интегралов (неравенства для верхних и нижних границ).

Насколько полезен искусственный интеллект?

Было бы здорово иметь методы, учитывающие вариативность дифференциальных уравнений в частных производных, геометрии области, граничных условий и диапазонов параметров, аналогичных реальным физическим задачам. Многие отрасли

промышленности и науки возлагают надежды на ИИ и глубокое обучение, чтобы решить свои давние проблемы или пролить на них новый свет. Астрономический прогресс в области вычислительных решений очень высокоразмерных задач, произошедший за последнее десятилетие, способен преобразить многие области, над которыми довлеет проклятие размерности. Такое преобразование означало бы перелом в развитии систем дифференциальных уравнений в частных производных и, в свою очередь, человечества в целом, т. к. эти уравнения и их решения открывают огромные возможности для науки.

В этой главе мы расскажем о трудностях, с которыми сталкивается сообщество специалистов по дифференциальным уравнениям при использовании традиционных подходов к поиску своих решений, а также при подгонке реальных и зашумленных данных к своим моделям. Затем мы проиллюстрируем, как машинное обучение помогает обойти или смягчить эти трудности. Также мы рассмотрим два вопроса:

- Что может сделать ИИ для дифференциальных уравнений в частных производных?
- Что могут сделать дифференциальные уравнения в частных производных для ИИ?

Необходимо убедиться, что при работе с дифференциальными уравнениями в частных производных, а также с метками или целями контролируемого обучения, будут четко обозначены отличительные черты машинного обучения — *обучающая функция*, *функция потерь*, *оптимизация*. При этом не так просто подогнать устоявшуюся область дифференциальных уравнений в частных производных к условиям машинного обучения. В идеале нужно составить карту перехода от уравнения к его решению. Для этого нам нужно выдержать некоторую паузу и хорошенько подумать.

Численные решения обладают огромной ценностью

Написание математической модели, описывающей природное явление, в виде уравнений, характеризующих взаимодействие задействованных переменных, — это только первый шаг. Необходимо решить эти уравнения.

Аналитические решения сложнее численных, поскольку чем больше модель имитирует природу, тем сложнее уравнения. Даже если аналитические методы не позволяют получить формулы для решений, они все равно дают ценные сведения об их важных свойствах. Численные решения проще, чем аналитические, т. к. они предполагают дискретизацию непрерывных уравнений, переводя нас из области непрерывных функций в область дискретных чисел или из пространств функций бесконечной размерности в пространства векторов конечной размерности (линейная алгебра), для вычисления которых предназначены машины. Численные решения дают неоценимое представление об истинных аналитических решениях моделей, и по возможности их можно легко проверить на соответствие при помощи экспериментальных наблюдений. Они также легко поддаются настройке, и поэтому являются отличным подспорьем в разработке экспериментов.

Численные решения можно находить в любом масштабе, но при попытке реализовать и вычислить численные схемы нас неизбежно настигнет проклятие размерности. Во многих ситуациях для численного моделирования даже одной секунды естественной эволюции системы требуются огромные вычислительные мощности, так что приходится прибегать к сокращению размерности и упрощению допущений, что еще больше отдаляет нас от достоверного приближения к истинным решениям. Печально, но это — скорее норма, чем исключение.

Непрерывные и дискретные функции

Функция $f(x)=x^2-3$ непрерывна на всей вещественной прямой $(-\infty;\infty)$. При ее дискретизации в численную схему для машинной обработки, в первую очередь, область уже не может быть всей вещественной линией, т. к. машины все еще не могут воспринимать бесконечные области. Поэтому первой аппроксимацией будет резкое сокращение области значений до конечной $[-N; N]$, где N — большое число. Вторая аппроксимация — дискретизация этой конечной области путем сокращения ее еще раз от континуума $[-N; N]$ до конечного набора точек. Если использовать много точек, то сетка будет более мелкой, а аппроксимация — более качественной, правда, за счет увеличения вычислительных затрат. Пусть, например, для дискретизации отрезка $[-5; 5]$ используются всего шесть точек: $-5, -3, -1, 1, 3, 5$; тогда наша непрерывная функция сводится к вектору с шестью записями:

$$f(x)=x^2-3 \text{ непрерывна на } (-\infty,\infty);$$

$$\overrightarrow{\text{дискретная } f}=\begin{pmatrix}(-5)^2-3\\(-3)^2-3\\(-1)^2-3\\1^2-3\\3^2-3\\5^2-3\end{pmatrix}=\begin{pmatrix}22\\6\\-2\\-2\\6\\22\end{pmatrix}.$$

На рис. 13.5 показаны непрерывная функция и ее до безумия недостоверная шеститочечная аппроксимация.

Нам все еще нужна дискретизация производных

Можно дискретизировать функцию $f(x)$, выбрав точки на интервале, как это делалось выше. Дифференциальные уравнения содержат не только функции, но и такие *производные функций*, как f_x, Δf. Поэтому необходимо дискретизировать производные или найти другой способ свести задачу из функциональных пространств (таких как непрерывные пространства) к векторным пространствам (чтобы использовать линейную алгебру и вычислять при помощи машин). Можно отметить два популярных мето-

да дискретизации дифференциальных уравнений — *метод конечных разностей* и *метод конечных элементов*. Мы рассмотрим их в ближайшее время, а также *вероятностный метод Монте-Карло*, основанный на случайных блужданиях.

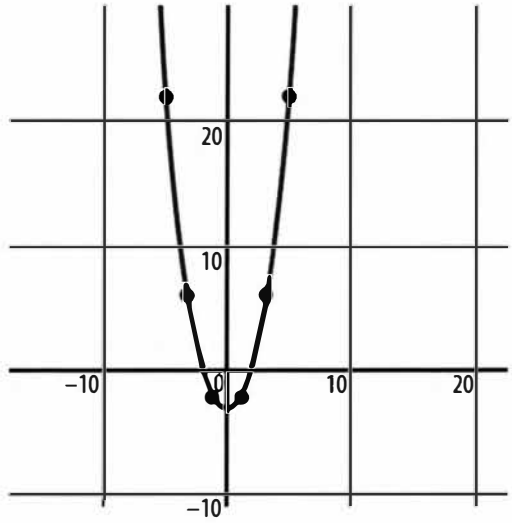


Рис. 13.5. Дискретизация непрерывной функции в вектор с шестью точками. При этом множество непрерывной информации теряется между точками

Компромисс с простотой численных решений заключается в том, что при дискретизации мы выполняем аппроксимацию, сводя бесконечный континуум к конечному набору точек, теряя при этом всю бесконечно подробную информацию, которая находится между конечным набором точек. То есть мы жертвуем высоким разрешением. Для некоторых уравнений существуют аналитические методы, которые позволяют точно определить, сколько информации мы теряем при дискретизации, и помогают вернуться к точному аналитическому решению, взяв предел при уменьшении размера дискретной сетки до нуля.

Дискретизация непрерывных функций и уравнений с ними имеет свои преимущества — легкость доступа. Можно научить старшеклассников численно решать уравнение теплопроводности, описывающее диффузию тепла в стержне (далее в этой главе), однако нельзя научить их решать его аналитически, пока они не закончат вузовский курс вычислений и линейной алгебры. Именно поэтому необходимо обучать детей моделировать и вычислять численные решения реальных задач в самом раннем возрасте. Простота численных решений и возможности вычислений в решении любого рода человеческих проблем должны стать приоритетом в нашей системе образования. Вряд ли природой задумано так, чтобы мы строили и распутывали безумно сложные математические теории, прежде чем *вычислить*, как устроен окружающий нас мир. К тому же сомнительно, что природа настолько сложна, как некоторые математические теории (хотя они все равно интересны сами по себе, хотя бы как упражнение в том, как далеко могут завести нас правила логики и умозаключений).

Дифференциальные уравнения в частных производных в моей докторской диссертации

История моей докторской диссертации демонстрирует резкое различие между математической теорией и численными подходами. Кроме того, она послужила отличным подспорьем к некоторым темам этой главы. В ней я работала над математической моделью, описывающей, как атомы диффундируют и перемещаются между различными уровнями ступенчатой поверхности тонкого кристалла. Это весьма полезно как специалистам в области материаловедения, так и инженерам, занятым разработкой миниатюрных устройств, встроенных в электронные приборы. Со временем из-за движения атомов на поверхности кристалла его форма меняется. Наконец, кристалл стабилизируется и принимает устойчивую форму.

Сразу же выполняем дискретизацию и приступаем к компьютерному моделированию

Как только я записала уравнения, мне представилась возможность выполнить компьютерное моделирование, которое показало, как форма кристалла изменяется со временем. Ниже приведено одно из дифференциальных уравнений в частных производных из моей диссертации (необязательно углубляться в него и выяснять, к чему относится функция в нем):

$$u_t(h,t) = -u^2\left(u^3\right)_{hhhh}, \text{ где } h \in [0;1],\ t \in [0;\infty).$$

Для опытного глаза это крайне нелинейное уравнение четвертого порядка. Неизвестная функция u оказывается как в квадрате, так и в кубе. Ее куб имеет четыре производные в пространстве, которые можно рассматривать как четыре степени, удаленные от функции, которую мы хотим оценить. На рис. 13.6 показана дискретизация моих дифференциальных уравнений в пространстве методом конечных разностей (мы обсудим конечные разности в ближайшее время) и его граничные условия (значения функции в точках 0 и 1).

Дискретный наклон $u_i = \dfrac{1/N}{x_{i-1} - x_i}$ ОДУ:

$$\begin{cases} u_i = u_i^2 \Delta_i \Delta u^3, i = 1, \ldots, N-1 \\ u_0 = u_N = 0 \\ \Delta_0 u^3 = \Delta_N u^3 = 0 \end{cases}$$

Континуумный наклон $u(h, t)$ ДУЧП:

$$\begin{cases} u_t = -u^2 (u^3)_{hhhh} \\ u(0,t) = u(1,t) = 0 \\ u^3_{hh}(0,t) = u_{hh}(1,t) = 0 \end{cases}$$

Рис. 13.6. Дискретные дифференциальные уравнения[1] и их континуальный аналог

Как правило, чем более нелинейным является уравнение, тем сложнее оно поддается стандартным аналитическим методам. Мне все еще нужно было провести математический анализ и доказать, что форма, которую показало численное моделирование, — это действительно то, к чему стремятся уравнения, т. е. *как раз* аналитиче-

[1] ДУЧП — дифференциальное уравнение в частных производных. — *Прим. ред.*

ское решение, и именно его выбирает природа среди других возможных. Следующие два года я занималась только этим. То, к чему я пришла, оказалось крошечным доказательством в крошечном случае для физически нереального *одномерного кристалла*! Мне пришлось свести все уравнения к одному измерению, чтобы хоть как-то провести с ними математический анализ.

Проклятие размерности

В книге неоднократно затрагивается проблема размерности задачи. Даже численное моделирование, которое заняло меньше одного дня, получилось только для уравнения в одномерной области. При попытке моделирования реалистичного тонкопленочного лабораторного кристалла, лежащего на плоской поверхности, т. е. когда пришлось дискретизировать двумерную поверхность вместо одномерного сегмента, количество дискретных точек подскочило со 100 на одномерном сегменте до $100^2 = 10\,000$ на двумерной поверхности. Мой компьютер в то время не смог численно решить точно такое же уравнение, которое в одномерном случае заняло всего несколько секунд. Конечно, тогда я была еще недостаточно подготовлена для проведения вычислений на университетском сервере или использования параллельных вычислений (не знаю, были ли тогда вообще изобретены распределенные облачные вычисления). В этом заключается проклятие размерности. С увеличением числа измерений вычислительные затраты растут экспоненциально. Возьмем уравнения, области которых изначально имеют высокую размерность (даже до дискретизации), например *уравнение Шрёдингера* для квантовой системы частиц, *уравнение Блэка — Шоулза* для оценки стоимости финансовых инструментов или *уравнение Гамильтона — Якоби — Беллмана* в динамическом программировании, моделирующем многопользовательские игры или задачи распределения ресурсов. А теперь представим себе масштабы проклятия размерности.

Геометрия задачи

Другая тема, которую мы уже затрагивали ранее, но которую стоит еще раз отметить, — форма пространства, которая имеет значение как для анализа, так и для численных расчетов. В моем нереальном одномерном случае в качестве пространства для уравнения использовался отрезок. В двумерном случае вариантов оказалось намного больше: прямоугольник (имеет преимущество регулярной решетки), круг (имеет преимущество радиальной симметрии) или любая другая реалистичная неправильная форма, которая обычно не имеет названия. Самыми простыми в плане анализа считаются пространства прямоугольника и круга (не для моих конкретных уравнений, а для других более простых уравнений, например линейных). Они также подходят для моделирования. Но в случае когда форма пространства не регулярна, что характерно для большинства реалистичных вещей, необходимо разместить больше дискретных точек в участках, где она нерегулярна, если мы хотим достоверно отобразить область. Проклятие размерности снова дает о себе знать — больше точек означает более длинные векторы и более крупные входные матрицы для вычислений.

Моделирование всевозможных вещей

В завершение истории с докторской диссертацией скажу, что я никогда не встречала настоящего тонкопленочного кристалла, подобного тому, над которым я работала, до тех пор, пока через 10 лет после получения степени моя подруга не показала мне тонкопленочный кристалл золота в своей лаборатории. Оглядываясь назад, можно сказать, что мне следовало начать именно с этого — *увидеть* в реальной жизни то, что я пыталась смоделировать. Сейчас мои приоритеты расставлены подругому, и я всегда начинаю с того, что спрашиваю себя: интересно ли мне то, что я пытаюсь смоделировать, насколько близко модель, над которой я решила работать, имитирует реальность и стоит ли вообще тратить свои время и силы на аналитические решения для данного конкретного приложения?

Дискретизация и проклятие размерности

Математикам, изучающим дифференциальные уравнения в частных производных, нравится непрерывный мир, в то время как машинам — дискретный. Математикам нравится анализировать функции, а машинам — вычислять их. Для того чтобы примирить эти два подхода и сделать машины полезными для математиков и наоборот, можно дискретизировать наши континуальные уравнения. Каким образом? Во-первых, нужно дискретизировать пространство уравнения, создав дискретную сетку. Выберем тип сетки (регулярная или нерегулярная) и то, насколько мелкой или крупной она будет. Затем выполним дискретизацию самого дифференциального уравнения, воспользовавшись одним из четырех популярных методов.

Метод конечных разностей.

Детерминированный, хорошо подходит для дискретизации времени, одномерных или относительно регулярных пространственных геометрий.

Метод конечных элементов.

Детерминированный, хорошо подходит для дискретизации более сложных пространственных геометрий, а также пространственных геометрий, изменяющихся во времени.

Вариационные или энергетические методы.

Аналогичны методу конечных элементов, но работают с более узким набором дифференциальных уравнений в частных производных. Они должны обладать *вариационным принципом* или *энергетической формулировкой*, т. е. само уравнение должно быть эквивалентно $\nabla E(u)=0$ для некоторого функционала энергии $E(u)$ (отображение функций на вещественной прямой). Мне удалось получить докторскую степень только потому, что по счастливой случайности я обнаружила такой энергетический функционал для своих уравнений. Аналогично тому, как минимум функции исчисления находится в точках, где $\nabla f(\vec{x})=0$, минимум функционала энергии находится в точках, где $\nabla E(u)=0$, но, безусловно, необходимо определить, что значит взять производную *функционала*.

Методы Монте-Карло.

Вероятностные, начинаются с дискретизации дифференциальных уравнений в частных производных, затем на ее основе разрабатывается соответствующая схема случайного блуждания, которая позволяет нам *агрегировать* решение в определенной точке пространства.

Слово *"конечный"* в этих методах подчеркивает тот факт, что данный процесс перемещает нас из континуума бесконечных пространств функций в конечные пространства векторов.

Если сетка, используемая для дискретизации, окажется слишком мелкой, это даст более высокое разрешение, но в итоге мы получим векторы и матрицы высокой размерности. Важно всегда помнить о проклятии размерности, а также о том, что одна из главных причин стремительного роста популярности нейросетей заключается в том, что, скорее всего, они обладают магической способностью преодолевать проклятие размерности. Скоро мы увидим, как именно это происходит.

Метод конечных разностей

Метод конечных разностей применяется для численной аппроксимации производных функций, входящих в дифференциальные уравнения в частных производных. Например, скорость частицы — это производная по времени от вектора ее положения, а ускорение частицы — две производные по времени от вектора ее положения.

В аппроксимации методом конечных разностей производные заменяются линейными комбинациями значений функции в дискретных точках пространства. Напомним, что одна производная определяет скорость изменения функции. Две производные характеризуют вогнутость. Производные высших порядков измеряют еще много того, с чем приходится сталкиваться работникам науки. Связь между производными функции в точке и тем, как соотносятся между собой ее значения в окрестностях этой точки, довольно интуитивна.

Математическое обоснование таких аппроксимаций опирается на *теорему Тейлора* из курса исчисления:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{f''(x_i)}{2}(x - x_i^2) + \frac{f'''(x_i)}{3!}(x - x_i)^3 +$$

$$+ \ldots + \frac{f^{(n)}(x_i)}{n!}(x - x_i)^n + \text{остаточный член},$$

где *остаточный член* зависит от того, насколько хороша производная следующего порядка $f^{(n+1)}(\xi)$ вблизи точки x_i, где мы пытаемся использовать полиномиальную аппроксимацию. Теорема Тейлора аппроксимирует вполне подходящую функцию вблизи точки многочленом, чьи коэффициенты определяются производными функции в этой точке. Чем больше производных у функции в точке, тем она пригоднее и тем *больше ведет себя как многочлен* в окрестности этой точки.

Теперь проведем дискретизацию одномерного отрезка $[a; b]$, а затем запишем конечно-разностные аппроксимации для производных функции $f(x)$, определенной на этом отрезке. Можно дискретизировать $[a; b]$ при помощи $n+1$ одинаково расположенных точек, так что размер ячейки составит $h = \frac{b-a}{n}$. Теперь можно оценить f в любой дискретной точке. Если нам нужны значения в окрестности некоторой точки x_i, тогда мы задаем $f_{i+1} = f(x_i + h)$, $f_{i+2} = f(x_i + 2h)$, ..., $f_{i-1} = f(x_i - h)$ и т. д. В дальнейшем h будет небольшим, поэтому метод $O(h^2)$ (или более высокого порядка по h) будет более точным, чем метод $O(h)$:

1. Разностная аппроксимация вперед с точностью $O(h)$ для первой производной (используются две точки):
$$f'(x_i) \approx \frac{f_{i+1} - f_i}{h}.$$
2. Разностная аппроксимация назад с точностью $O(h)$ для первой производной (использует две точки):
$$f'(x_i) \approx \frac{f_i - f_{i-1}}{h}.$$
3. Центрированные разностные аппроксимации с точностью $O(h^2)$ для производных до четвертой (используются две точки, усредняются разности вперед и назад):
$$f'(x_i) \approx \frac{f_{i+1} - f_{i-1}}{2h};$$
$$f''(x_i) \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2};$$
$$f'''(x_i) \approx \frac{f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}}{2h^3};$$
$$f^{IV}(x_i) \approx \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{h^4}.$$
4. Центрированные разностные аппроксимации с точностью $O(h^4)$ для производных до четвертой:
$$f'(x_i) \approx \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h};$$
$$f''(x_i) \approx \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12h^2};$$
$$f'''(x_i) \approx \frac{-f_{i+3} + 8f_{i+2} - 13f_{i+1} + 13f_{i-1} - 8f_{i-2} + f_{i-3}}{8h^3};$$
$$f^{IV}(x_i) \approx \frac{-f_{i+3} + 12f_{i+2} - 39f_{i+1} + 56f_i - 39f_{i-1} + 12f_{i-2} - f_{i-3}}{6h^4}.$$

Что означает $O(h^k)$? Это порядок численной аппроксимации в h. Когда мы заменяем производную ее численной аппроксимацией, мы допускаем ошибку. Значение $O(h^k)$ показывает, насколько велика эта ошибка. Очевидно, что она зависит от размера сетки h. С более мелкими сетками ошибка будет меньше. Для нахождения таких границ погрешности применяются *ряды Тейлора* для $f(x+h)$, $f(x-h)$, $f(x+2h)$, $f(x-2h)$ и т. д., а также их линейные комбинации, позволяющие определить аппроксимацию нужной производной и порядок *конечно-разностной аппроксимации* по h. Использование рядов Тейлора предполагает, что мы имеем дело с функциями, которые *действительно имеют необходимое количество производных* в точках, где мы их оцениваем. То есть мы предполагаем, что наша функция вполне пригодна для оценки этих производных. Если в окрестности этих точек функция имеет сингулярности, то необходимо найти способы обойти их, например воспользоваться более тонкими сетками вблизи сингулярностей.

Пример: решить $y''(x) = 1$ на [0; 1] при граничных условиях $y(0) = -1$ и $y(1) = 0$

Перед нами обыкновенное линейное дифференциальное уравнение второго порядка на ограниченной области в одном измерении. Этот пример тривиален, т. к. аналитическое решение слишком простое. Достаточно дважды проинтегрировать уравнение и восстановить функцию без производных $y(x) = 0{,}5x^2 + c_1x + c_2$, где c — *постоянные интегрирования*. Для нахождения значений c подставляем два граничных условия и получаем аналитическое решение $y(x) = 0{,}5x^2 + 0{,}5x - 1$. Однако смысл этого примера в том, чтобы показать, как использовать конечные разности для вычисления не аналитического, но *численного решения*, поскольку для многих других дифференциальных уравнений аналитические решения недоступны, и в этом нужно как следует разобраться. Сначала производится дискретизация области [0; 1]. При этом можно использовать любое количество точек. Чем больше точек, тем с большей размерностью придется иметь дело, но и разрешение в таком случае будет лучше. Воспользуемся только восемью точками, так что размер ячейки составит $h = 1/7$ (рис. 13.7). Теперь наш континуум [0; 1] сведен к восьми точкам (0, 1/7, 2/7, 3/7, 4/7, 5/7, 6/7, 1).

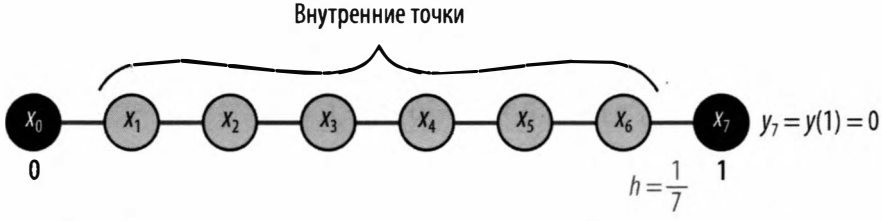


Рис. 13.7. Дискретизация единичного интервала при помощи восьми дискретных точек, что соответствует семи интервалам. Размер шага (или размер ячейки) составляет h = 1/7

Затем выполним дискретизацию дифференциального уравнения. Для дискретизации второй производной можно использовать любую схему конечных разностей. Выберем центральную разность $O(h^2)$, тогда дискретизированное дифференциальное уравнение примет вид:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = 1 \quad \text{при } i = 1, 2, 3, 4, 5, 6.$$

Важно отметить, что дифференциальное уравнение справедливо только *внутри* области, поэтому при записи его дискретного аналога точки $i = 0$ и $i = 7$ не учитываются. Значения при $i = 0$ и $i = 7$ находятся из граничных условий $y_0 = -1$ и $y_7 = 0$. Таким образом, получается система из шести уравнений и шести неизвестных $y_1, y_2, y_3, y_4, y_5, y_6$:

$$\begin{aligned}
y_2 - 2y_1 - 1 &= 1/49;\\
y_3 - 2y_2 + y_1 &= 1/49;\\
y_4 - 2y_3 + y_2 &= 1/49;\\
y_5 - 2y_4 + y_3 &= 1/49;\\
y_6 - 2y_5 + y_4 &= 1/49;\\
0 - 2y_6 + y_5 &= 1/49.
\end{aligned}$$

Итак, мы перешли от континуального пространства в пространство линейной алгебры:

$$\begin{pmatrix}
-2 & 1 & 0 & 0 & 0 & 0\\
1 & -2 & 1 & 0 & 0 & 0\\
0 & 1 & -2 & 1 & 0 & 0\\
0 & 0 & 1 & -2 & 1 & 0\\
0 & 0 & 0 & 1 & -2 & 1\\
0 & 0 & 0 & 0 & 1 & -2
\end{pmatrix}
\begin{pmatrix} y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6 \end{pmatrix} =
\begin{pmatrix} 1/49 + 1\\ 1/49\\ 1/49\\ 1/49\\ 1/49\\ 1/49 \end{pmatrix}.$$

Решение системы равнозначно *инверсии* этой трехдиагональной матрицы, которая является дискретным аналогом оператора производной второго порядка. В континуальном пространстве мы *интегрируем* дифференциальный оператор и восстанавливаем $y(x)$, а в дискретном пространстве мы *инвертируем* дискретный оператор и восстанавливаем дискретные значения y_i. Важно помнить о проклятии размерности, когда для дискретизации области используется большее количество точек.

Очевидно, чтобы понять, насколько хорошо работает конечно-разностная схема с восемью дискретными точками, нужно сравнить дискретные значения y_i с их точными аналогами $y(x_i)$ (рис. 13.8). На рис. 13.9 изображен граф численного решения (с использованием только четырех дискретных точек) в сравнении с точным аналитическим решением.

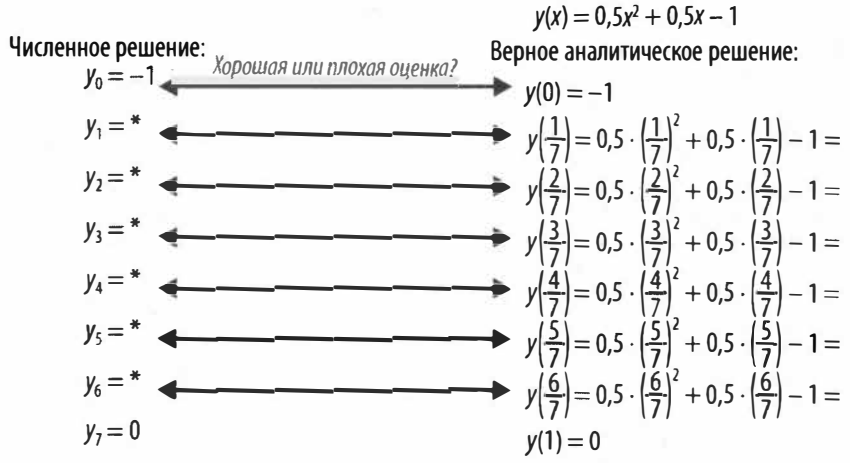


Рис. 13.8. Сравнение численного решения с точным аналитическим решением в каждой дискретной точке

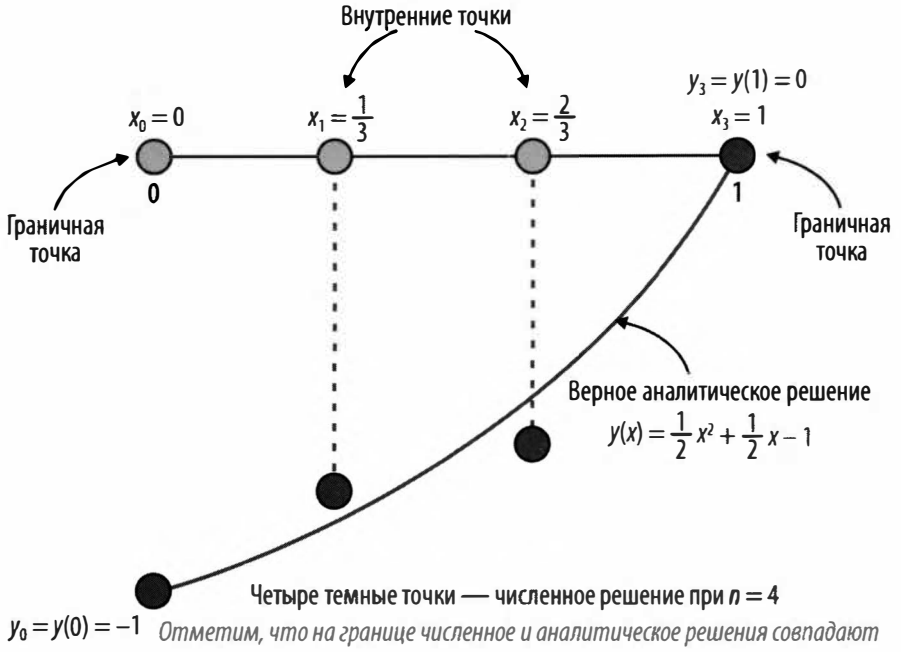


Рис. 13.9. Граф численного решения (с использованием только четырех дискретных точек) относительно точного аналитического решения (сплошная линия)

Теперь для дискретизации *любого* дифференциального уравнения любого порядка или типа на области любой размерности можно применить метод конечных разностей. Все, что нужно, — это дискретизировать область и выбрать схемы конечных разностей для аппроксимации производных во всех дискретных точках внутри области.

Пример: дискретизация одномерного уравнения теплопроводности $u_t = \alpha u_{xx}$ внутри интервала $x \in (0; 1)$

Перед нами линейное дифференциальное уравнение второго порядка на ограниченной пространственной области в одном измерении. Здесь $u = u(x, t)$ — функция двух переменных, поэтому схема дискретизации должна учитывать обе координаты. Мы можем дискретизировать либо только в пространстве и сохранять время непрерывным, либо только во времени и сохранять пространство непрерывным, либо как в пространстве, так и во времени. Как правило, при этом используется более одного численного маршрута. Хорошо, когда имеются варианты. При дискретизации и в пространстве, и во времени в итоге мы получаем систему алгебраических уравнений. При дискретизации только в пространстве, но не во времени, получается система обыкновенных дифференциальных уравнений. Поскольку дифференциальное уравнение в частных производных линейно, дискретизированная система также будет линейной.

Запишем полную дискретную схему. Для дискретизации в пространстве используем центрированную разность второго порядка и аппроксимируем вторую производную. А для дискретизации по времени используем разность вперед и аппроксимируем первую производную:

$$\frac{u_{i,j+1} - u_{i,j}}{s} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \text{ при } i = 1, 2, ..., n \text{ и } j = 0, 1, 2, ...$$

В таких уравнениях $u(x, t)$ может быть известно в некоторый начальный момент времени ($u(x, 0) = g(x)$), и нужно узнать, что $u(x, t)$ изменяется во времени. В численной схеме подстрочный индекс i означает дискретное пространство, а j — дискретное время. Таким образом, дискретный аналог начального условия будет $u_{i,0} = g_i$, и нам нужно найти решение для неизвестных $u_{i,j+1}$ при $i = 1, 2, ..., n$ и $j = 1, 2, ...$ Можно легко выделить $u_{i,j+1}$ в предыдущей численной схеме:

$$u_{i,j+1} = \frac{s}{h^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right) + u_{i,j} \text{ при } i = 1, 2, ..., n \text{ и } j = 0, 1, 2, ...$$

Наконец, с их помощью находим значения $u_{i,1}$, $u_{i,2}$, ... (вперед по времени) при $i = 1, 2, ..., n$. Например, вводим $j = 0$ и находим значения дискретных u на первом временно́м шаге:

$$u_{i,1} = \frac{s}{h^2}\left(u_{i+1,0} - 2u_{i,0} + u_{i-1,0}\right) + u_{i,0} = \frac{s}{h^2}\left(g_{i+1} - 2g_i + g_{i-1}\right) + g_i \text{ при } i = 1, 2, ..., n.$$

Важно отметить, что нам известны все дискретные значения g, так что теперь мы знаем и дискретное значение $u_{i,1}$. Затем подставляем $j = 1$ и находим значения дискретных $u_{i,2}$ на следующем временно́м шаге и т. д.

Метод конечных элементов

Методы конечных элементов отличаются от методов конечных разностей тем, что они оперируют *слабой формулировкой* дифференциальных уравнений в частных производных, а не непосредственно уравнениями. Слабая формулировка *взвешена и усреднена*, поэтому речь идет об интегралах и интегрировании по частям. К этому мы еще вернемся.

Прежде чем мы обсудим общую идею конечных элементов, рассмотрим рис. 13.10. На нем показано конечно-элементное решение дифференциального уравнения в частных производных вверху круговой области. Для дискретизации области используется треугольная сетка, а решение возможно аппроксимировать кусочно-линейной функцией. Можно использовать другие многоугольные формы сеток, а также *более гладкие* функции, нежели кусочно-линейные, например кусочно-квадратичные или многочлены высших степеней. Компенсацией за бóльшую гладкость служит большее количество вычислений.

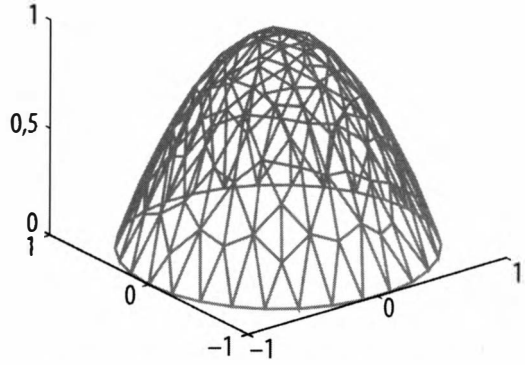


Рис. 13.10. Решение методом конечных элементов в круговой области (источник изображения: **https://oreil.ly/bljPF**)

Продемонстрируем, как метод конечных элементов дает численную аппроксимацию к решению следующего дифференциального уравнения в частных производных:

$$-\Delta u(x, y) = f(x, y) \text{ при } (x, y) \in \Omega \subset \mathbb{R}^2;$$
$$u(x, y) = 0 \text{ при } (x, y) \in \textit{граница}_{\Omega}.$$

Это уравнение Пуассона (встречается в электростатике). Здесь нет изменения во времени. Задана $f(x, y)$, и мы ищем неизвестную функцию $u(x, y)$, равную нулю по всей границе, и вторые производные которой u_{xx} и u_{yy} равны $-f(x, y)$. Это дифференциальное уравнение в частных производных довольно неплохо изучено, и имеются формулы для его аналитического решения, но нас интересует только численная аппроксимация решения методом конечных элементов.

Для этого выполним аппроксимацию неизвестной функции $u(x, y)$, существующей в пространстве *бесконечной* размерности, с помощью известной функции, су-

ществующей в пространстве *конечной* размерности. Пространства конечной размерности охватываются лишь *конечным числом линейно независимых функций*. Мы сами выбираем эти *базисные функции*, поэтому необходимо позаботиться о том, чтобы наш выбор облегчил нам вычисления. Как правило, мы выбираем кусочно-линейные или кусочно-полиномиальные функции, каждая из которых *минимально поддерживается на сетке*. Это означает, что базисная функция будет ненулевой только на одном или двух соседних элементах сетки и нулевой везде. Таким образом, интегрирование с помощью этой функции по всей области дифференциального уравнения в частных производных сводится к интегрированию только по одному или двум элементам сетки.

После выбора *базисных функций*, каждая из которых поддерживается на нескольких элементах сетки, аппроксимируем истинное решение $u(x, y)$ посредством линейной комбинации этих простых локально поддерживаемых базисных функций:

$$u(x, y) \approx u_1 \cdot \text{базис}_1(x, y) + u_2 \cdot \text{базис}_2(x, y) + ... + u_n \cdot \text{базис}_n(x, y).$$

Теперь необходимо найти постоянные u_i линейной комбинации. Таким образом, от решения для неизвестной функции $u(x, y)$ в континууме задача сводится к решению для неизвестного вектора коэффициентов $(u_1, u_2, ..., u_n)$. Необходимо выбрать их так, чтобы аппроксимация $u_1 \cdot \text{элемент}_1(x, y) + u_2 \cdot \text{элемент}_2(x, y) + ... + u_n \cdot \text{элемент}_n(x, y)$ в некотором смысле удовлетворяла дифференциальному уравнению в частных производных. Имеется n неизвестных, следовательно, нужно написать n уравнений и решить систему из n уравнений при n неизвестных. Их можно получить из дифференциального уравнения в частных производных или его *слабой формулировки*. Для того чтобы получить слабую формулировку, умножаем его на функцию $v(x, y)$, интегрируем по нашей области, а затем избавляемся от производных высших порядков *методом интегрирования по частям*. Важно запомнить, что чем меньше производных, тем мы ближе к неизвестной функции. Проделаем это пошагово.

Исходное дифференциальное уравнение в частных производных имеет вид:

$$-\Delta u(x, y) = f(x, y) \text{ при } (x, y) \in \Omega \subset \mathbb{R}^2;$$
$$u(x, y) = 0 \text{ при } (x, y) \in \text{граница}_\Omega.$$

Умножаем дифференциальное уравнение в частных производных на функцию $v(x, y)$ и интегрируем по всей области. Такая формулировка считается слабой формулировкой дифференциального уравнения в частных производных, т. к. удовлетворяет ему не в точечной, а в интегральной форме:

$$-\int_\Omega \Delta u(x, y) v(x, y) dxdy = \int_\Omega f(x, y) v(x, y) dxdy.$$

Отметим оператор $\Delta = \nabla.\nabla$, представляющий точечное произведение двух операторов производной. Интегрирование по частям позволяет избавиться от одной про-

изводной, переместив ее на другую функцию внутри интеграла. Однако это не дается бесплатно. В процессе этой операции интеграл обретает отрицательный знак и еще один интегральный член, который действует на границе области. Этот новый интеграл на границе интегрирует произведение двух антипроизводных. Для граничного члена требуется внешний *единичный вектор нормали* к границе $\vec{n}$:

$$\int_{\Omega} \nabla u(x,y).\nabla v(x,y)\,dxdy - \int_{граница_{\Omega}} v(x,y)\nabla u(x,y).\vec{n}ds = \int_{\Omega} f(x,y)v(x,y)\,dxdy.$$

На границе можно выбрать $v(x, y) = 0$, и тогда весь граничный член исчезнет:

$$\int_{\Omega} \nabla u(x,y).\nabla v(x,y)\,dxdy = \int_{\Omega} f(x,y)v(x,y)\,dxdy.$$

Теперь заменим $u(x, y)$ на его аппроксимацию конечной размерности:

$$\int_{\Omega} \nabla\left(u_1 \cdot \text{базис}_1(x,y) + u_2 \cdot \text{базис}_2(x,y) + \ldots + u_n \cdot \text{базис}_n(x,y)\right).\nabla v(x,y)\,dxdy =$$

$$= \int_{\Omega} f(x,y)v(x,y)\,dxdy,$$

что эквивалентно:

$$\int_{\Omega} \left(u_1\nabla\text{базис}_1(x,y) + u_2\nabla\text{базис}_2(x,y) + \ldots + u_n\nabla\text{базис}_n(x,y)\right).\nabla v(x,y)\,dxdy =$$

$$= \int_{\Omega} f(x,y)v(x,y)\,dxdy.$$

Вот и все. Можно выбрать *n* различных функций для $v(x, y)$ и получить *n* различных уравнений в *n* неизвестных (u_i — неизвестные). Общий смысл заключается в том, что при каждом выборе мы используем то, что не усложнит вычислительный процесс. Самым простым выбором для $v(x, y)$ будет *n* уже имеющихся базисных функций, т. к. при интегрировании относительно друг друга они производят большое количество отмен (*ортогональность*), а при интегрировании относительно самих себя дают число 1 (*нормальность*). Изначально выбранные базисные функции образуют *ортонормированный набор* функций. И все это для того, чтобы облегчить нам жизнь. Таким образом, *n* уравнений имеют вид:

$$\int_{\Omega} \left(u_1\nabla\text{базис}_1(x,y) + u_2\nabla\text{базис}_2(x,y) + \ldots + u_n\nabla\text{базис}_n(x,y)\right).\nabla\text{базис}_1(x,y)\,dxdy =$$

$$= \int_{\Omega} f(x,y)\text{базис}_1(x,y)\,dxdy;$$

$$\int_{\Omega} \left(u_1\nabla\text{базис}_1(x,y) + u_2\nabla\text{базис}_2(x,y) + \ldots + u_n\nabla\text{базис}_n(x,y)\right).\nabla\text{базис}_2(x,y)\,dxdy =$$

$$= \int_{\Omega} f(x,y)\text{базис}_2(x,y)\,dxdy;$$

...

$$\int_{\Omega}\left(u_1\nabla\text{базис}_1(x,y)+u_2\nabla\text{базис}_2(x,y)+\ldots+u_n\nabla\text{базис}_n(x,y)\right).\nabla\text{базис}_n(x,y)dxdy=$$
$$=\int_{\Omega}f(x,y)\text{базис}_n(x,y)dxdy.$$

В итоге мы решаем систему из n уравнений с n неизвестными, которую задаем в форме линейной алгебры (где $b_i=\text{базис}_i$):

$$\begin{pmatrix}\int_{\Omega}\nabla b_1(x,y).\nabla b_1(x,y)dxdy & \int_{\Omega}\nabla b_2(x,y).\nabla b_1(x,y)dxdy & \ldots & \int_{\Omega}\nabla b_n(x,y).\nabla b_1(x,y)dxdy\\ \int_{\Omega}\nabla b_1(x,y).\nabla b_2(x,y)dxdy & \int_{\Omega}\nabla b_2(x,y).\nabla b_2(x,y)dxdy & \ldots & \int_{\Omega}\nabla b_n(x,y).\nabla b_2(x,y)dxdy\\ \vdots & \vdots & \ldots & \vdots\\ \int_{\Omega}\nabla b_1(x,y).\nabla b_n(x,y)dxdy & \int_{\Omega}\nabla b_2(x,y).\nabla b_n(x,y)dxdy & \ldots & \int_{\Omega}\nabla b_n(x,y).\nabla b_n(x,y)dxdy\end{pmatrix}\begin{pmatrix}u_1\\u_2\\\vdots\\u_n\end{pmatrix}=$$
$$=\begin{pmatrix}\int_{\Omega}f(x,y).b_1(x,y)dxdy\\ \int_{\Omega}f(x,y).b_2(x,y)dxdy\\ \vdots\\ \int_{\Omega}f(x,y).b_n(x,y)dxdy\end{pmatrix}.$$

Напомним, что нам известны функция $f(x,y)$, все базисные функции $b_i=\text{базис}_i$ и область Ω, так что остается лишь решить систему уравнений. Это *разреженная система*, т. к. большинство интегралов равны нулю. По этой причине выбраны базисные функции с небольшой поддержкой. Никому не хочется решать *плотную систему уравнений*.

Конечно, здесь возникает множество вопросов, ответы на которые можно найти в обширной литературе по конечным элементам.

Неужели все дифференциальные уравнения в частных производных имеют слабую формулировку, которая позволяет делать подобные вещи?

Да, потому что всегда можно умножить дифференциальные уравнения в частных производных на функции v и проинтегрировать по частям, хотя некоторые уравнения обладают лучшими структурами для проведения упрощающих вычислений, по сравнению с остальными.

Связаны ли между собой энергетические формулировки или вариационные принципы дифференциальных уравнений в частных производных?

Да, они связаны. Вспомним *метод Ритца*. Мы упоминали его в *главе 10*, когда связывали минимизацию энергетических функционалов с решением дифференциальных уравнений в частных производных[2]. Важно помнить, что большинство

[2] См. *подразд. "Уравнение теплопроводности выполняет градиентный спуск для функционала энергии Дирихле" раздела "Пример 1: гармонические функции, энергия Дирихле, уравнение теплопроводности" главы 10. — Прим. ред.*

дифференциальных уравнений в частных производных имеют слабую формулировку, но не все они имеют формулировку минимизации энергии. Одной из причин, способствующих получению докторской степени, было то, что я нашла энергетическую формулировку для дифференциального уравнения в частных производных, с которым работала. Это произошло совершенно случайно. Мне просто повезло найти *слабую формулировку* с последующим интегрированием по частям. Точно так же, как это было сделано в текущем разделе. Пожалуй, мы еще не до конца понимаем ценность метода проб и ошибок.

Чем интересны пространства Соболева и почему мы изучаем их на продвинутых курсах по дифференциальным уравнениям в частных производных?

Прежде всего, нужно поместить функции u, v и базисные функции в соответствующие *функциональные пространства*, позволяющие убедиться в том, что все используемые вычисления и аппроксимации действительны. Например, нам не нужно, чтобы раздувались задействованные интегралы, содержащие наши функции и их производные.

Можно ли использовать неравномерные сетки для корректировки более детальной части области, как показано на рис. 13.11?

Да, поскольку в нашем обсуждении ничто не указывает на строгую зависимость от равномерной сетки.

Сколько нужно базисных функций?

Ровно столько, сколько элементов сетки.

При каких условиях приближенное решение сходится к истинному?

Добро пожаловать в анализ методом конечных элементов.

Как это используется в приложениях?

Постоянно. Все началось с механики и проектирования конструкций: нагрузки, напряжения, деформации; но теперь метод конечных элементов используется для *численного решения* разнообразных дифференциальных уравнений в частных производных, чьи пространственные области имеют сложную геометрию.

Что может пойти не так?

Как всегда, нас преследует проклятие размерности. Для более высокого разрешения требуется больше элементов сетки, поэтому система уравнений, которую приходится решать в итоге, растет экспоненциально с увеличением количества элементов сетки. В этом нет ничего хорошего. В идеале нужно получить сетку, которая не будет детализирована там, где это не нужно, и более детализирована в интересующих нас частях области.

Что еще может усложнить ситуацию?

Для дифференциальных уравнений в частных производных, *область которых изменяется во времени*, требуются сетки, изменяющиеся во времени соответствующим образом.

Может ли ИИ помочь в выборе подходящей сетки для определенной геометрии и дифференциального уравнения в частных производных?

Да, и мы увидим это в ближайшее время в текущей главе.

Прежде чем двигаться дальше, отметим, что метод конечных элементов — это метод, зависящий от сетки конечной размерности, который аппроксимирует решение дифференциального уравнения в частных производных. Позже в этой главе мы познакомимся с бессеточными нейросетевыми методами.

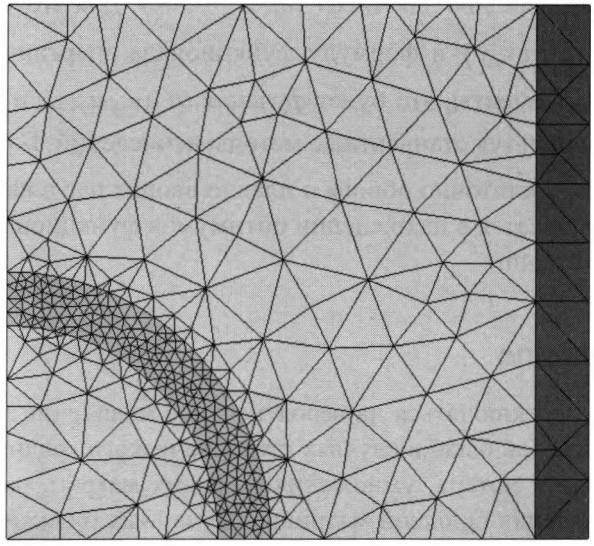


Рис. 13.11. Двумерная область с неравномерной треугольной сеткой (источник изображения: **https://oreil.ly/f9uHk**)

Вариационные или энергетические методы

Некоторые дифференциальные уравнения в частных производных являются особенными в том смысле, что их решение минимизирует функционал энергии. Мы говорим, что такое уравнение обладает *вариационным принципом*. К ним относится уравнение Пуассона, которое мы только что решили методом конечных элементов. Когда дифференциальное уравнение в частных производных обладает вариационным принципом, для нас открывается еще один способ понять его решение, изучив минимизируемый им функционал энергии.

Запишем уравнение Пуассона и функционал энергии, минимизирующий его решение, не вдаваясь в подробности, почему это так:

$$\Delta u(x,y) = f(x,y) \text{ при } (x,y) \in \Omega \subset \mathbb{R}^2;$$

$$u(x,y) = 0 \text{ при } (x,y) \in \textit{граница}_{\Omega};$$

$$E(u(x,y)) = \int_{\Omega} |\nabla u(x,y)|^2 + 2f(x,y)\,dxdy.$$

Теперь можно использовать эти новые знания для численной аппроксимации решения дифференциального уравнения в частных производных и найти приближенную схему минимизации функционала энергии. По аналогии с методом конечных элементов проецируем решение бесконечной размерности $u(x,y)$ на пространство конечной размерности, где можно выбирать базисные элементы

$$u(x,y) \approx u_1 \cdot \text{базис}_1(x,y) + u_2 \cdot \text{базис}_2(x,y) + ... + u_n \cdot \text{базис}_n(x,y)$$

и нужно снова найти решение для чисел $(u_1, u_2, ..., u_n)$. Для этого подставляем приближенное значение $u(x,y)$ в формулу функционала энергии. Теперь, когда мы знаем все базисные элементы, это будет *функция* от $(u_1, u_2, ..., u_n)$, которую можно минимизировать, используя стандартные методы исчислений. Готово!

Этот метод является достаточно общим и плавно вводит нас в вариационное исчисление, которое заключается в нахождении оптимумов функционалов, а не функций, как в обычном исчислении.

Методы Монте-Карло

Мы уже привыкли переключаться на вероятностное мышление для решения детерминированных задач. Так было в случаях стохастического градиентного спуска для минимизации функций потерь, умножения больших матриц, рандомизированного сингулярного разложения большой матрицы, случайных блужданий по графам для выявления сообществ, ранжирования веб-страниц и других целей. Приведем самые известные вводные примеры методов Монте-Карло.

- Оценка π посредством генерации множества случайных точек $(x_{\text{случайная}}, y_{\text{случайная}})$ в единичном квадрате и нахождения той части, которая лежит внутри вписанной четверти круга радиуса 1: $x^2_{\text{случайная}} + y^2_{\text{случайная}} \le 1$. Теперь можно оценить вероятность попадания в четверть круга:

$$\text{Prob}(\text{точка внутри четверти круга}) =$$

$$= \frac{\text{площадь четверти круга радиуса 1}}{\text{площадь единичного квадрата}} = \frac{\pi}{4} \approx$$

$$\approx \frac{\text{количество раз, когда точка оказывается внутри четверти круга}}{\text{общее количество сгенерированных точек}}.$$

- Оценка интеграла $\int_a^b f(x)dx$ неотрицательной непрерывной функции $f(x)$ на отрезке $[a; b]$ посредством генерации случайных точек $(x_{\text{случайная}}, y_{\text{случайная}})$, где $a \le x_{\text{случайная}} \le b$ и $0 \le y_{\text{случайная}} \le \max(f)$. Значение интеграла — это площадь под

графиком f. Ее можно оценить, найдя долю случаев, когда случайная точка лежит под графиком $f(x)$, или $y_{\text{случайная}} \le f\left(x_{\text{случайная}}\right)$:

$$\text{Prob}(\text{точка под графиком } f) = \frac{\text{точка под графиком } f}{\text{общая площадь прямоугольника}} =$$

$$= \frac{\int_a^b f(x)dx}{(b-a)\times \max(f)} \approx$$

$$\approx \frac{\text{количество раз, когда точка оказывается под графиком } f}{\text{общее количество сгенерированных точек}}.$$

Такие стохастические методы решения детерминированных задач называются *методами Монте-Карло*, т. к. в них используются метод выигрыша в азартных играх и подсчеты пропорций определенных исходов, подобно азартным играм в казино Монте-Карло в Монако. Их можно было бы также назвать *методами Лас-Вегас-Стрип*. Это аналогично *рандомизированным контролируемым испытаниям* для ответа на детерминированные вопросы, например для оценки воздействия конкретного лекарственного препарата на конкретную популяцию. По-другому на этот вопрос можно ответить посредством полностью детерминированного обсервационного исследования, в котором мы контролируем все предполагаемые вмешивающиеся переменные и оцениваем воздействие вмешательства препарата.

Теперь предположим, что имеется детерминированное дифференциальное уравнение в частных производных, и нужно найти его решение с помощью рандомизированных численных испытаний (метод Монте-Карло). Для иллюстрации того, как это происходит, воспользуемся простым дифференциальным уравнением в частных производных:

$$\Delta u(x,y) = 0 \qquad \text{при } (x,y) \text{ в единичном квадрате} \subset \mathbb{R}^2;$$
$$u(x,y) = g(x,y) \quad \text{при } (x,y) \in \textit{граница}_{\textit{площадь}}.$$

Сначала выполним дискретизацию области с помощью равномерной сетки, затем напишем конечно-разностную схему для дифференциального уравнения в частных производных во внутренних точках сетки и для граничных условий:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0, \text{ если } (i,j) \text{ соответствует}$$

внутренней точке сетки;

$u_{i',j'} = g_{i',j'}$, если (i',j') соответствует граничной точке.

Задача состоит в том, чтобы с помощью численной схемы найти $u_{i,j}$ для каждой внутренней точки сетки. Это будет численная оценка истинного решения $u(x,y)$ в конкретной внутренней точке.

Решим задачу для $u_{i,j}$:

$$u_{i,j} = \frac{1}{4}u_{i+1,j} + \frac{1}{4}u_{i-1,j} + \frac{1}{4}u_{i,j+1} + \frac{1}{4}u_{i,j-1},$$

если (i,j) соответствует внутренней точке сетки, и

$$u_{i',j'} = g_{i',j'},$$

если (i',j') соответствует граничной точке.

Именно так интерпретируются уравнения применительно к случайному блужданию: если мы находимся на границе, то нам известно решение, которое составит $u_{i',j'} = g_{i',j'}$. Таким образом, случайное блуждание, которое для построения маршрута следует указаниям схемы дифференциального уравнения в частных производных, получит вознаграждение $g_{i',j'}$ в граничной точке. Причем решение $u_{i,j}$ во внутренней точке сетки (i,j) — это невзвешенное среднее решение в четырех окружающих точках сетки. Таким образом, если случайное блуждание стартует из внутренней точки сетки (i,j), то вероятность того, что оно забредет в любую из четырех соседних точек, а затем в соседнюю точку, составит 0,25, пока оно не попадет в граничную точку сетки (i',j'), где и получит свое вознаграждение $g_{i',j'}$. Это всего лишь одно исследование схемы дифференциального уравнения в частных производных, в некотором смысле дающее нам крошечную часть информации о том, какая граничная точка внесла вклад в решение $u_{i,j}$. Если повторять этот процесс многократно, скажем тысячу раз, начиная с одной и той же точки сетки (i,j), где нужно найти численное решение, то можно будет подсчитать, сколько раз случайное блуждание оказывалось в каждой из граничных точек:

$$\text{Prob}\left(\text{заканчивающаяся в точке } (i',j')\right) \approx \frac{N_{\text{кон}}}{N_{\text{старт}}},$$

где $N_{\text{кон}}$ — количество раз, когда случайное блуждание заканчивается в точке (i',j'); $N_{\text{старт}}$ — общее количество случайных блужданий, стартующих в точке (i,j).

В результате мы получим оценку ожидаемого вознаграждения от всех граничных точек, что и является искомым численным решением того, как каждое граничное значение играет роль в решении во внутренней точке. Итак, численное решение дифференциального уравнения в частных производных имеет вид:

$$u_{i,j} = \sum_{(i',j')} \text{Prob}\left(\text{заканчивающаяся в точке } (i',j')\right) g_{i',j'}.$$

Это очень удобный способ получить численное решение, не требующий решения линейной системы уравнений (которая может оказаться чрезмерно большой и нежелательной). Он также отлично подойдет в случаях, когда важно найти решение не на всей сетке, а только в нескольких точках.

Разумеется, для каждого дифференциального уравнения в частных производных необходимо разработать правильную численную схему, а также определить вероятности перехода случайного блуждания. Например, если в уравнении есть коэффициенты, умноженные на вторые производные, тогда случайное блуждание не сможет попасть в каждую из четырех соседних точек с одинаковой вероятностью 0,25. Присутствие коэффициентов подразумевает ввод веса для каждого соседа, поэтому вероятности перехода корректируются с учетом каждого из них.

С точки зрения теории, нужно доказать, что блуждание попадает-таки в конечном счете на границу и что полученное таким образом численное решение сходится к истинному аналитическому решению дифференциального уравнения в частных производных. Также необходимо получить аналитические оценки того, через какое время (в среднем) случайное блуждание останавливается, как быстро сходится численное решение и как полученное таким образом численное решение отличается от решения, полученного методом конечных разностей или конечных элементов, по точности, стоимости вычислений и скорости сходимости.

При использовании методов Монте-Карло иногда приходится действовать наоборот. Мы разрабатываем симуляцию с различными процессами и скоростями переходов, имитирующую некоторые физические явления (например, взаимодействующие частицы системы), затем усредняем ее и переходим к написанию дифференциального уравнения в частных производных с дескрипторами рассматриваемой системы. Это прямо противоположно тому, как если бы мы начали с дифференциального уравнения в частных производных, а затем разработали подробную симуляцию Монте-Карло для его решения. Об этом мы поговорим далее.

Немного статистической механики: замечательное основное уравнение

Одно из моих любимых дифференциальных уравнений в частных производных — *основное уравнение* из статистической механики, которое относится к тем немногим уравнениям данного вида, благодаря которым можно перейти от характеризации системы в атомистическом или молекулярном масштабе (система частиц), где описание носит вероятностный характер, к той же системе в макромасштабе, где описание носит детерминированный характер. Логично ожидать, что лежащие в их основе атомистические процессы и переходы порождают наблюдаемое поведение в макромасштабе. Один умнейший человек, который познакомил меня со статистической механикой, как-то сказал: "Разве весь наш жизненный опыт не является результатом коллективного поведения некой мощной химической реакции, лежащей в его основе?"

Переход от основного уравнения для атомистических вероятностей к детерминированным дифференциальным уравнениям в частных производных для наблюдаемых величин происходит аккуратно, без ощущения, что мы обманываемся или делаем нечеткие предположения, или что у нас есть две совершенно несвязанные модели:

одна — в макромасштабе, другая — в атомистическом масштабе, которые не имеют ничего общего друг с другом.

Основное уравнение отслеживает изменение вероятности того, что статистическая система (некоторые частицы) в определенный момент времени будет находиться в определенном состоянии. Вычислим скорость изменения вероятности состояния системы, вычитая потери из выигрышей и учитывая скорость перехода между различными состояниями:

$$\frac{\partial P(h,t)}{\partial t} = \sum_{h'} P(h',t)T(h' \to h) - P(h,t)T(h \to h'),$$
$$:= LP.$$

где $T(h \to h')$ и $T(h' \to h)$ — скорости перехода из состояния h в h', и наоборот. Мы вычисляем эти скорости перехода, используя основные физические предположения или наблюдения за системой, например скорости испарения и конденсации атомов, скорости диффузии и т. д.

Теперь с помощью основного уравнения можно записать дифференциальные уравнения в частных производных для детерминированных дескрипторов системы, вычисляя их ожидания. Ожидания позволяют перейти от вероятностных величин к детерминированным. Они вычисляются следующим образом:

$$\langle f \rangle = \sum_h fP(h,t) = \sum_h f \frac{e^{-H(h)/KT}}{Z},$$

где $H(h)$ — полная энергия, а Z — функция разделения. Выражение $\frac{e^{-H(h)/KT}}{Z}$ довольно распространено в статистической механике и отражает интуитивную идею о том, что вероятность возникновения состояний с высокой энергией экспоненциально снижается, а это значит, что системы предпочитают низкую энергию и изменяются в сторону состояний, снижающих общую энергию.

Ожидание *vs* усреднение по *N* повторениям моделирования методом Монте-Карло

Ожидание $\langle f \rangle$ связано с моделированием методом Монте-Карло. Оно эквивалентно пределу в виде $N \to \infty$ среднего значения $\bar{f}$ от f после N повторений симуляций методом Монте-Карло.

Теперь с помощью основного уравнения можно вычислить скорость изменения ожидания нужной величины, например h_i, представляющей собой высоту кристаллического профиля (состоящего из атомов) на определенном участке i:

$$\frac{d\langle h_i \rangle}{dt} = \sum_h h_i \frac{\partial P}{\partial t} = \sum_h h_i LP.$$

Если система замкнута, т. е. если мы можем выразить правую часть в терминах h и ее производных по пространству и времени, то мы получим уравнение движения для ожидаемого профиля высоты. Если система не замкнута, тогда для ее замыкания придется выполнить аппроксимацию. Желательно сделать физически правдоподобную аппроксимацию (например, чтобы система была близка к равновесию), иначе наши усилия окажутся бесполезными.

И последний шаг — грубая сортировка полученных дискретных уравнений движения до получения непрерывной модели дифференциального уравнения в частных производных, описывающей профиль кристалла. На этом этапе мы переходим от схемы конечных разностей к непрерывному дифференциальному уравнению в частных производных, что представляет собой обратный процесс дискретизации, который мы рассматривали на примере конечных разностей. Благодаря этому итоговое дифференциальное уравнение в частных производных возникает непосредственно из атомистических процессов. В большинстве случаев такое уравнение имеет вид:

$$h_t\left(\vec{x}, t\right) = F\left(h\left(\vec{x}, t\right), t; \vec{\omega}\right),$$

где $\vec{\omega}$ — набор физических параметров системы.

Вскоре в этой главе мы рассмотрим применение графовых нейросетей в моделировании природных явлений в макроскопическом масштабе непосредственно на основе систем частиц. Это позволит обойтись без записи дифференциальных уравнений в частных производных, как это делалось в текущем разделе. Входными данными для сети будут выступать частицы с их взаимодействиями и скоростями взаимодействия, а выходными — изменения во времени (видеоролик или временна́я последовательность графов) всей системы в целом.

Решения как ожидания лежащих в основе случайных процессов

Для некоторых типов дифференциальных уравнений в частных производных удобным способом поиска решений является их формулировка в виде ожиданий некоторых лежащих в основе случайных процессов, когда мы моделируем случайные траектории соответствующего стохастического процесса, а затем вычисляем его ожидание. Так можно оценить решения на любом заданном участке пространства-времени.

Для того чтобы научиться это делать, необходимо изучить *формулу Фейнмана — Каца* и *исчисление Ито* (позволяет находить производные функций случайных величин, зависящих от времени). Они прекрасно связывают дифференциальные уравнения в частных производных и вероятность.

Формула Фейнмана — Каца (не будем ее записывать) предлагает практический способ решения некоторых дифференциальных уравнений в частных производных,

которые преследует проклятие размерности. Например, в количественных финансах с помощью этой формулы можно эффективно вычислять решения уравнения Блэка — Шоулза для определения цены опционов на акции. В квантовой химии ее можно использовать в решении уравнения Шрёдингера.

Преобразование дифференциального уравнения в частных производных

Идея здесь проста: возможно, дифференциальное уравнение в частных производных решается легче (аналитически или численно) в преобразованном пространстве, чем в пространстве, в котором оно существует на данный момент. Поэтому его нужно каким-то образом преобразовать с надеждой на успех.

Преобразование Фурье

Преобразование Фурье — это интегральное преобразование из пространства x в частотное пространство ξ:

$$F.T\left(f\left(x\right)\right)=\hat{f}\left(\xi\right)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{-i\xi x}f\left(x\right)dx.$$

Обратное преобразование Фурье отменяет преобразование Фурье и возвращает нас из частотного пространства ξ в пространство x:

$$F.T^{-1}\left(\hat{f}\left(\xi\right)\right)=f\left(x\right)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{i\xi x}\hat{f}\left(\xi\right)dx.$$

Для большего удобства существуют таблицы с преобразованиями Фурье большого количества функций. Когда они недоступны, мы прибегаем к численным методам. Прямое и обратное преобразования Фурье актуальны для многих приложений, среди которых частотный анализ, модуляция сигналов, фильтрация, поэтому существуют алгоритмы, предназначенные для их быстрого вычисления.

Ниже перечислены некоторые важные моменты, которые необходимо знать о преобразовании Фурье.

- Оно разлагает функцию на частотные компоненты. Преобразование Фурье функции говорит о том, сколько каждой частоты содержится в функции. Частотный спектр функции $f\left(x\right)$ — это абсолютное значение ее преобразования Фурье $\left|F\left(\xi\right)\right|$.
- Оно обратимо, что позволяет перемещаться между пространством x и частотным пространством ξ.
- Оно меняет свертку двух функций в пространстве x на умножение функций в частотном пространстве ξ: $F.T.\left(f*g\left(x\right)\right)=F.T.\left(f\left(x\right)\right)\times F.T.\left(g\left(x\right)\right)=\hat{f}\left(\xi\right)\hat{g}\left(\xi\right)$.

Это может пригодиться в поисках аналитического решения для дифференциального уравнения в частных производных. Решение дифференциального уравнения в частных производных в пространстве x сводится к решению алгебраического уравнения или более простого дифференциального уравнения в пространстве ξ, после чего используется обратное преобразование Фурье, возвращающее в пространство x. В большинстве случаев мы инвертируем произведение двух преобразований Фурье, так что в итоге решение оказывается сверткой в пространстве x. Если вы уже сталкивались с *функциями Грина* для аналитических решений, то это один из способов их получения.

- Оно меняет дифференцирование по x на умножение на $i\xi$, так что:

$$F.T.\left(u_x(x)\right) = i\xi F.T.\left(u(x)\right) = i\xi\hat{u}(\xi)$$

и

$$F.T.\left(u_{xx}(x)\right) = -\xi^2 F.T.\left(u(x)\right) = -\xi^2\hat{u}(\xi).$$

Огромную ценность представляет возможность избавиться от производных. Это означает, что дифференциальные уравнения в исходном пространстве становятся алгебраическими уравнениями в пространстве Фурье.

- Это линейное преобразование, поэтому его можно применять по отдельности к каждому члену в дифференциальном уравнении в частных производных. С его помощью можно решать линейные дифференциальные уравнения в частных производных с постоянными коэффициентами. Его можно использовать и в линейных уравнениях с непостоянными коэффициентами (когда параметры зависят от пространства), но при этом придется повозиться с разложением этих коэффициентов в ряды. После того как мы напишем ряды, необходимо исследовать их сходимость.
- Оно применяется при доказательстве универсальной теоремы аппроксимации для нейросетей (Хорник и др., 1989).
- Его можно использовать для ускорения работы сверточных нейросетей (Матье и др., 2013).
- Оказывается, что представление дифференциальных уравнений в частных производных в пространстве Фурье очень удобно при обучении нейросети их решениям.

Однако отметим и некоторые *не столь удобные* моменты.

- Для многих функций существуют *комплекснозначные преобразования Фурье*. В этом случае достаточно изучить компле́ксный анализ и работать с ним.
- Преобразование Фурье применимо не ко всем функциям. Задействованный интеграл работает в бесконечной области, поэтому если в нем отсутствует функция, компенсирующая резкое уменьшение до нуля, интеграл раздувается (что делает бесполезным преобразование Фурье). Ядро преобразования Фурье имеет вид $e^{-i\xi x} = \cos(\xi x) - i\sin(\xi x)$. Оно колеблется с частотой ξ и никогда не уменьшается до нуля.

- Даже если для функций имеется обратное преобразование Фурье (позволяющее найти аналитические решения для дифференциальных уравнений в частных производных), его формула зачастую неизвестна. В таких случаях невозможно написать явное аналитическое решение с помощью этого метода. Это общая проблема многих аналитических методов.
- Преобразования Фурье характеризуются принципом неопределенности Гейзенберга (**https://oreil.ly/0hH7h**).

  Изучение принципа неопределенности началось с аргумента Вернера Гейзенберга о невозможности одновременного определения положения и импульса свободной частицы с произвольной точностью. В квантовой механике волновая функция положения является преобразованием Фурье волновой функции импульса. Наиболее популярное использование принципов неопределенности Фурье — описание естественного компромисса между стабильностью и измеримостью системы, особенно квантовомеханической. Если представить, что $f(x)$ — это вероятность того, что положение частицы равно x, а $f(\xi)$ — вероятность того, что ее импульс равен ξ, тогда неравенство Гейзенберга дает нижнюю границу разброса этих двух распределений вероятностей. С точки зрения физики предположение состоит в том, что положение и импульс связаны между собой преобразованием Фурье: $|f|2L2 \le 4\pi \cdot |(x-x_0)f|L2 \cdot |(\xi-\xi_0)f|L2$. Качественно это означает, что для узкой функции характерно широкое преобразование Фурье, а для широкой функции — узкое преобразование Фурье. В любой области широкая функция означает, что имеется буквально широкое распределение данных, поэтому в одной из областей всегда присутствует неопределенность.

Преобразование Фурье vs ряд Фурье

Не стоит путать преобразование Фурье с рядом Фурье по синусам и косинусам. Функция $\sin x$ не имеет преобразования Фурье, зато сама представляет собой ряд Фурье по синусам.

Преобразование Лапласа

Преобразование Лапласа позволяет преобразовывать более широкий класс функций, чем преобразование Фурье, поскольку его ядро e^{-st} экспоненциально быстро стремится к нулю (в экспоненте нет компле́ксного значения i, которое могло бы все испортить). Преобразование Лапласа оперирует функциями, определенными на $[0; \infty)$, поэтому в дифференциальных уравнениях в частных производных оно применяется для преобразования временно́й переменной или других переменных, расположенных в диапазоне $[0; \infty)$. Таким образом, вместо решения дифференциального уравнения в частных производных непосредственно во временно́й области, мы выполняем преобразование Лапласа и решаем его в области s, после чего с помощью обратного преобразования Лапласа возвращаем его во временну́ю область.

Формула преобразования Лапласа имеет вид:

$$L.T.\left(f\left(t\right)\right)=\hat{f}\left(s\right)=\int_{0}^{\infty}e^{-st}f\left(t\right)dt.$$

А формула обратного преобразования Лапласа:

$$L.T.^{-1}\left(\hat{f}\left(s\right)\right)=f\left(t\right)=\frac{1}{2\pi i}\int_{C-i\infty}^{C+i\infty}e^{st}\hat{f}\left(s\right)ds.$$

Как и в случае с преобразованием Фурье, для удобства составлены таблицы преобразований Лапласа для многих функций.

Нас интересует, как преобразование Лапласа действует на производные (по времени), входящие в дифференциальное уравнение в частных производных. От них желательно избавиться, поскольку по-другому приблизиться к его решению не получится. Так и есть:

- $L.T\left(u_t\left(x,t\right)\right)=s\hat{u}\left(x,s\right)-u\left(x,0\right);$
- $L.T\left(u_{tt}\left(x,t\right)\right)=s^2\hat{u}\left(x,s\right)-su\left(x,0\right)-u_t\left(x,0\right).$

Важно отметить, что в большинстве случаев нам известны начальные условия для $u\left(x,0\right)$ и $u_t\left(x,0\right)$ в дифференциальном уравнении в частных производных, так что такие преобразования действительно устраняют производные по времени.

Для нас также актуально свойство свертки к умножению, чтобы перевести алгебраические выражения в s в решения дифференциального уравнения в частных производных в пространстве t при помощи обратного преобразования Лапласа. Здесь важно обратить внимание на то, что это конечная свертка от 0 до t, в отличие от $-\infty$ до ∞ в случае преобразования Фурье:

$$L.T.\left(\left(f*g\right)\left(t\right)\right)=\hat{f}\left(s\right)\hat{g}\left(s\right),\text{ где }\left(f*g\right)\left(t\right)=\int_{0}^{t}f\left(\tau\right)g\left(t-\tau\right)d\tau=\int_{0}^{t}f\left(t-\tau\right)g\left(\tau\right)d\tau.$$

Как и преобразование Фурье, преобразование Лапласа — это линейный оператор, поэтому лучше всего использовать его с линейными дифференциальными уравнениями в частных производных.

Сведение дифференциальных уравнений в частных производных к алгебраическим уравнениям или к обыкновенным дифференциальным уравнениям

Преобразование Фурье, преобразование Лапласа и некоторые другие преобразования, в частности, преобразования Ханкеля и Меллина, позволяют избавиться от производных дифференциальных уравнений в частных производных по определенным переменным (времени, пространству и т. д.). В результате получается алгебраическое уравнение, если преобразование действует на все задействованные в уравнении переменные, или обыкновенное дифференциальное уравнение (ОДУ), если преобразование действует на

все переменные, кроме одной. При этом мы надеемся, что решить алгебраические уравнения или ОДУ будет проще, чем исходное дифференциальное уравнение в частных производных, и что мы сможем воспользоваться известными методами алгебры, математики и ОДУ для решения новых уравнений в переменных преобразования. Мы увидим простой пример того, как это работает, в следующем разделе, посвященном оператору решения.

Операторы решения

Рассмотрим два простых, но содержательных примера, которые иллюстрируют методы преобразования и в то же время показывают идеи, лежащие в основе оператора решения для дифференциальных уравнений в частных производных. Это достаточно общие примеры и, что более важно, они закладывают основу применения нейросетей в решении таких уравнений. К тому же оба примера имеют явные аналитические решения, так что их можно использовать для проверки приближенных или итерационных методов решения дифференциальных уравнений в частных производных (включая нейросетевые методы).

В первом примере используется одномерное уравнение теплопроводности с постоянными коэффициентами на бесконечной области (решение зависит от времени), а во втором — двумерное уравнение Пуассона с постоянными коэффициентами на ограниченной области с простой геометрией (решение не зависит от времени, оно статично во времени).

Пример использования уравнения теплопроводности

Уравнение теплопроводности на бесконечном одномерном стержне имеет вид:

$$u_t(x,t) = \alpha u_{xx}(x,t) \quad \text{при } x \in \mathbb{R},\ t \in (0,\infty);$$
$$u(x,0) = u_0(x) \qquad \text{при } x \in \mathbb{R}.$$

Поскольку это дифференциальное уравнение в частных производных определено на бесконечной области по x, необходимо задать условия дальнего поля (граница отсутствует, поэтому нужно указать, как, по нашему мнению, выглядит функция решения $u(x,t)$ при $x \to \infty$ и $x \to -\infty$). Предположим, что эти пределы равны нулю.

Для простоты предположим, что параметр α постоянен, и тогда можно будет воспользоваться преобразованием Фурье. Применив это преобразование (относительно x) к дифференциальному уравнению в частных производных $u_t(x,t) = \alpha u_{xx}(x,t)$ и начальному условию, нам удастся избавиться от производных по x и упростить дифференциальное уравнение в частных производных до обыкновенного дифференциального уравнения с одной производной по времени:

$$\hat{u}_t(\xi,t) = -\alpha\xi^2\hat{u}(\xi,t) \quad \text{при } \xi \in \mathbb{R},\ t \in (0,\infty);$$
$$\hat{u}(\xi,0) = \hat{u}_0(\xi) \qquad \text{при } \xi \in \mathbb{R}.$$

Теперь его можно легко решить, воспользовавшись методом из области обыкновенных дифференциальных уравнений, который называется *методом разделения переменных* (не вдаваясь в подробности), и получить решение в пространстве Фурье:

$$\hat{u}(\xi, t) = e^{-\alpha\xi^2 t}\hat{u}_0(\xi).$$

Нам нужно решение не в пространстве Фурье, а в пространстве x, поэтому возьмем обратное преобразование Фурье вышеприведенного выражения и воспользуемся знанием о том, что при преобразованиях из пространства x в пространство Фурье умножения превращаются в свертки. Таким образом, решение в пространстве x имеет вид:

$$u(x, t) = F.T.^{-1}\left(e^{-\alpha\xi^2 t}\hat{u}_0(\xi)\right) = F.T.^{-1}\left(e^{-\alpha\xi^2 t}\right) * u_0(x) = \frac{1}{\sqrt{4\pi\alpha t}} e^{-\frac{x^2}{4\alpha t}} * u_0(x) =$$

$$= \int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi\alpha t}} e^{-\frac{(s-x)^2}{4\alpha t}} u_0(s)ds = \int_{-\infty}^{\infty} kernel(s, x; t; \alpha)u_0(s)ds.$$

Суть вычислений заключается в следующем:

> *решение u(x, t) дифференциального уравнения в частных производных представляет собой интеграл некоторой функции ядра kernel(s, x; t; α) по исходному состоянию решения $u_0(s)$.*

Более того, оператор решения дифференциального уравнения в частных производных проецирует заданные входные данные (в нашем случае это параметр α и исходное состояние $u_0(x)$) на выход, т. е. искомое решение $u(x, t)$, путем интегрирования исходного состояния по функции ядра, зависящей от параметра дифференциального уравнения в частных производных (наряду с зависимостью от пространства и времени).

Зная формулу ядра (или используя нейросеть для его аппроксимации), можно получить решение заданного дифференциального уравнения в частных производных. В таких случаях говорят, что нейросеть *обучилась оператору решения* дифференциального уравнения в частных производных.

В нашем простом примере, опираясь на линейность и постоянные коэффициенты, мы смогли применить методы преобразования Фурье и свертку и вернуться к вещественному пространству, а также вывести явную аналитическую формулу ядра интеграла, а именно $k(s, x; t; \alpha) = \frac{1}{\sqrt{4\pi\alpha t}} e^{\frac{(s-x)^2}{4\alpha t}}$, так, что аппроксимации больше не требуется. Приятно отметить, что ядро получается из зависящей от времени гауссовой функции $\text{Гауссиан}(x; t; \alpha) = \frac{1}{\sqrt{4\pi\alpha t}} e^{-\frac{x^2}{4\alpha t}}$, которая с течением времени разрастается. В результате свертки с начальным состоянием решения возникает эффект сглаживания, который, распространяясь, сглаживает любые начальные колебания и

скачки. Такое сглаживание происходит в ходе любого процесса диффузии, который можно визуализировать, например, диффузии дыма в воздухе или диффузии красителя в жидкости, где вещество плавно растекается до получения внешне однородной среды.

Пример использования уравнения Пуассона

Уравнение Пуассона в ограниченной области имеет вид:

$$-\nabla.\left(a\left(\vec{x}\right)\nabla u\left(\vec{x}\right)\right) = f\left(\vec{x}\right) \quad \text{при } \vec{x} \in D;$$
$$u\left(\vec{x}\right) = 0 \quad \text{при } \vec{x} \text{ на границе } D.$$

Если значение $a\left(\vec{x}\right)$ постоянно, а область двумерна, тогда:

$$-a\Delta u\left(x,y\right) = f\left(x,y\right) \quad \text{при } \left(x,y\right) \in D \subset \mathbb{R}^2;$$
$$u\left(x,y\right) = 0 \quad \text{при } \left(x,y\right) \text{ на границе } D,$$

где $\Delta u\left(x,y\right) = u_{xx}\left(x,y\right) + u_{yy}\left(x,y\right)$. Здесь можно применить преобразование Фурье по x и y, как в случае уравнения тепла (линейное уравнение с постоянными коэффициентами), но лучше воспользуемся *методом функции Грина*. Можно представить правую часть дифференциального уравнения в частных производных как совокупность импульсов интенсивности $f\left(x,y\right)$ в местонахождениях $\left(x,y\right)$ в континууме. Для того чтобы математически выразить понятие импульса, нам потребуется *дельта-функция (дельта-мера) Дирака* $\delta_{(x,y)}\left(s,p\right)$. Она равна нулю по всей области, кроме точки $\left(x,y\right)$, где она бесконечна, а ее полная мера на области нормирована на единицу. Смысл в том, что можно решить дифференциальное уравнение в частных производных, в правой части которого будет только импульс в определенном месте, после чего агрегировать исходное решение. Предположительно, решить такое уравнение проще, чем уравнение с некой заданной функцией с правой стороны, так что построим решение $u\left(x,y\right)$ из решения $G\left(x,y;s,p\right)$ импульсного дифференциального уравнения в частных производных. И что еще важней, с применением функции Грина мы получаем интегральное представление решения по входным данным относительно ядра (которое и является функцией Грина). Дифференциальное уравнение в частных производных с импульсной правой частью имеет вид:

$$-a\Delta G\left(s,p;x,y\right) = \delta_{(x,y)}\left(s,p\right) \quad \text{при } \left(s,p\right) \in D \subset \mathbb{R}^2;$$
$$G\left(s,p;x,y\right) = 0 \quad \text{при } \left(s,p\right) \text{ на границе } D.$$

Теперь запишем:

$$f\left(x,y\right) = \int_D f\left(s,p\right)\delta_{(x,y)}\left(s,p\right)dsdp$$

и дифференциальное уравнение в частных производных в виде:

$$-a\Delta u(x,y)=\int_D f(s,p)\delta_{(x,y)}(s,p)\,dsdp \text{ при } (x,y)\in D\subset\mathbb{R}^2;$$

$$u(x,y)=0 \qquad \text{при } (x,y) \text{ на границе } D.$$

Заменим $\delta_{(x,y)}(s,p)$ внутри интеграла на $-a\Delta G(s,p;x,y)$:

$$-a\Delta u(x,y)=\int_D -a\Delta G(s,p;x,y)f(s,p)\,dsdp \qquad \text{при } (x,y)\in D\subset\mathbb{R}^2;$$

$$u(x,y)=0 \qquad \text{при } \vec{x} \text{ на границе } D.$$

Теперь давайте предположим, что у нас есть подходящие условия для замены дифференцирования и интегрирования:

$$-a\Delta u(x,y)=-a\Delta\left(\int_D G(s,p;x,y)f(s,p)\,dsdp\right) \text{ при } (x,y)\in D\subset\mathbb{R}^2;$$

$$u(x,y)=0 \qquad \text{при } (x,y) \text{ на границе } D.$$

В итоге можно представить решение $u(x,y)$ в виде:

$$u(x,y)=\int_D G(x,y;s,p;a)f(s,p)\,dsdp.$$

Важно отметить, что зависимость G от a явно выражена в G. Далее в этой главе мы будем изучать оператор решений дифференциального уравнения в частных производных в контексте нейросети, и тогда физический параметр a будет в составе входных данных сети. Если параметр $a=a(s,p)$ непостоянный, то запишем $G(x,y;s,p;a(s,p))$. По аналогии с предыдущим примером, основная идея вычислений заключается в следующем:

> решением $u(x,y)$ дифференциального уравнения в частных производных является интеграл некоторой функции ядра, в данном случае функции Грина $G(s,p;x,y;a)$ от правой части дифференциального уравнения в частных производных $f(s,p)$.

Более того, оператор решения дифференциального уравнения в частных производных отображает заданные входные данные (в нашем случае параметр a и правая часть уравнения $f(x,y)$) на выход, т. е. искомое решение $u(x,y)$, путем интегрирования функции правой части по некоторой функции ядра, зависящей от параметра дифференциального уравнения в частных производных (помимо зависимости от пространства). В нашем случае ядром оператора решения служит функция Грина дифференциального уравнения в частных производных, знакомая нам по уравнению Пуассона на областях с простой геометрией, но не по более сложным случаям. И снова путь к решению данного дифференциального уравне-

ния в частных производных нам открывает знание формулы ядра или аппроксимация при помощи нейросети.

Метод простой итерации

Метод простой итерации применяется при построении явных решений и доказательства существования и единственности для нескольких удачных дифференциальных уравнений в частных производных. Это настолько простой и общий метод, что он обязан быть в нашем инструментарии. Запишем его, а затем сразу же применим, представив решение *динамической системы* в виде ряда. Динамическая система — это обыкновенное дифференциальное уравнение, описывающее изменение во времени частицы или совокупности частиц (системы). И снова возникает необходимость обучить нейросети операторам решения динамических систем, что вполне согласуется с тем, что мы обсуждали чуть раньше. Кроме того, было бы неплохо иметь представление решения методом простой итерации вместе с представлением нейросети. Напомним, что во многих математических системах одно и то же решение можно представить несколькими способами. Метод простой итерации считается аддитивным, а представление нейросети — композиционным. Более того, похоже, нейросети обладают преимуществом в представлении операторов решения целых семейств дифференциальных уравнений в частных производных и в целом более широким разнообразием, что выглядит как сбывшаяся мечта этой области.

Метод простой итерации предназначен для нахождения неподвижной точки функции, или точки x^*, которую функция проецирует на саму себя: $f(x^*) = x^*$. Это непростая задача, поскольку, как правило, f является нелинейной, и чаще всего нам неизвестно, существуют ли такие точки для данной функции. Как и полагается в случае с нелинейными уравнениями, с помощью итерационных методов удается избежать *одноразовых* решений и вместо этого получить последовательность точек, которые, *как мы полагаем, при соответствующих условиях* сходятся к желаемому решению, в данном случае к неподвижной точке функции.

Как это работает

Метод простой итерации заключается в следующем:

1. x_0 — начальная точка.
2. $x_{i+1} = f(x_i)$.

Вот и все. Последовательность $\{x_0, x_1, x_2, ...\}$ генерируется при последовательном применении f и выглядит как $\{x_0, f(x_0), f(f(x_0)), f(f(f(x_0))), ...\}$. При подходящих условиях по f и x_0 эта последовательность сходится к неподвижной точке x^* из f (так что $f(x^*) = x^*$).

Отметим, что в зависимости от f и начальной точки асимптотическое поведение этой последовательности может проявляться так, как представлено далее.

Сходимость к пределу x^*.

Если простая итерация сходится, то неподвижная точка фиксируется (в случае непрерывной функции f; если простая итерация сходится в неподвижной точке, тогда пределом должна быть неподвижная точка f).

Расходимость до ∞.

Последовательность растет без ограничения.

Периодичность.

Последовательность колеблется между двумя или более значениями.

Хаотическое поведение.

Последовательность ведет себя хаотично, без какой-либо закономерности.

Теоремы, относящиеся к методу простой итерации, утверждают, что от выбора начальной точки x_0 во многом зависит, сойдется ли она к неподвижной точке или нет.

Как это применяется в решении обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных

В этой главе мы ищем решения дифференциальных уравнений, представляющих собой функции. Поэтому сначала нужно переформулировать дифференциальное уравнение в частных производных так, чтобы его решение u удовлетворяло уравнению, имеющему вид $F(u)=u$ (отметим, что в данном случае F — это не функция, а оператор), что делает его идеальным для метода простой итерации. Затем применим ту же логику и построим последовательность *функций*, которые, как мы надеемся, при соответствующих условиях сойдутся к неподвижной точке u^* *оператора*, т. е. искомое решение дифференциального уравнения в частных производных. Важно отметить, что ранее мы выстраивали последовательность *чисел* (а не функций), которые, как мы надеемся, при соответствующих условиях сойдутся к неподвижной точке *функции* (а не оператора).

Продемонстрируем это на примере *динамической системы*. Она представляет собой одно из важнейших общих хорошо изученных обыкновенных дифференциальных уравнений, которое описывает изменение во времени точки в пространстве. Это простое уравнение в том смысле, что чаще всего является уравнением первого порядка, где нужно избавиться только от одной производной, и сложное — в том смысле, что оно, как правило, бывает нелинейным. Считается, что нужно линеаризовать динамическую систему в окрестности точки и изучить ее линеаризованное поведение. В большинстве случаев это даст информацию о нелинейном поведении, однако не стоит смешивать эти два понятия. Про линеаризованные системы уже достаточно много известно, а про нелинейные — практически ничего, так что нелинейным системам необходимо уделять не меньше внимания. В этом разделе мы не будем заниматься линеаризацией. Вместо этого выполним аппроксимацию решения при помощи ряда, построенного методом простой итерации.

Зная начальное состояние точки $\vec{u}(t_0)=\vec{u}_0$, траектория решения $\vec{u}(t)$ отслеживает все ее будущие состояния. Функция $\vec{f}(\vec{u}(t), a(t), t)$ задает характер изменения:

$$\frac{d\vec{u}(t)}{dt}=\vec{f}(\vec{u}(t), a(t), t);$$
$$\vec{u}(t_0)=\vec{u}_0.$$

Есть одна производная по времени, от которой необходимо избавиться, поэтому интегрируем один раз по времени:

$$\vec{u}(t)=\vec{u}_0+\int_{t_0}^{t}\vec{f}(\vec{u}(t), a(t), t)dt.$$

Перепишем это интегральное уравнение в форме, которая подходит для метода простой итерации. Вся правая часть будет рассматриваться как оператор с $\vec{u}(t)$ на входе:

$$\vec{u}(t)=F(\vec{u}(t)).$$

Теперь можно сгенерировать последовательность $\{\vec{u}_0(t), \vec{u}_1(t), \vec{u}_2(t), \vec{u}_3(t), ...\}$, сходящуюся к решению $\vec{u}(t)$ в течение всего времени или за конечный промежуток времени при подходящих условиях по *f*. Такая последовательность имеет вид:

- $\vec{u}_0(t)=\vec{u}_0(t)$;
- $\vec{u}_1(t)=F(\vec{u}_0(t))=\vec{u}_0+\int_{t_0}^{t}\vec{f}(\vec{u}_0(s), a(s), s)ds$;
- $\vec{u}_2(t)=F(\vec{u}_1(t))=\vec{u}_0+\int_{t_0}^{t}\vec{f}(\vec{u}_1(s), a(s), s)ds$;
- $\vec{u}_3(t)=F(\vec{u}_2(t))=\vec{u}_0+\int_{t_0}^{t}\vec{f}(\vec{u}_2(s), a(s), s)ds$

и т. д.

Простой, но довольно содержательный пример

Поскольку рассмотрение нескольких вариантов решения одной и той же задачи способствует лучшему усвоению сути изложенного метода, лучшими демонстрациями считаются простые примеры, которые можно решить несколькими способами. Возьмем довольно простую одномерную линейную динамическую систему:

$$\frac{du(t)}{dt}=u(t);$$
$$u(0)=1.$$

Первый вариант решения — метод разделения переменных, когда все, что содержит $u(t)$, помещается на одну сторону уравнения, а все, что содержит только t, — на другую:

$$\frac{du(t)}{u(t)} = dt.$$

Теперь можно проинтегрировать от 0 до t:

$$\int_0^t \frac{du(s)}{u(s)} = \int_0^t ds.$$

Получаем $\ln(u(t)) = t$, так что решением нашей простой динамической системы методом разделения переменных будет $u(t) = e^t$ (пожалуй, важнейшая функция в математике). Теперь построим последовательность функций методом простой итерации и посмотрим, сойдется ли она к решению $u(t) = e^t$ динамической системы:

- $u_0(t) = 1;$
- $u_1(t) = F(u_0(t)) = u_0(t) + \int_0^t u_0(s)\,ds = 1 + \int_0^t 1\,ds = 1 + t;$
- $u_2(t) = F(u_1(t)) = u_0(t) + \int_0^t u_1(s)\,ds = 1 + \int_0^t (1 + s)\,ds = 1 + t + \frac{t^2}{2};$
- $u_3(t) = F(u_2(t)) = u_0(t) + \int_0^t u_2(s)\,ds = 1 + \int_0^t \left(1 + s + \frac{s^2}{2}\right) ds = 1 + t + \frac{t^2}{2} + \frac{t^3}{3!}.$

Продолжаем:

$$u_n(t) = F(u_{n-1}(t)) = u_0(t) + \int_0^t u_{n-1}(s)\,ds = 1 + t + \frac{t^2}{2} + \frac{t^3}{3!} + \ldots + \frac{t^n}{n!}.$$

По мере того как $n \to \infty$, метод простой итерации сходится к ряду:

$$u_\infty(t) = 1 + t + \frac{t^2}{2} + \frac{t^3}{3!} + \ldots + \frac{t^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{t^n}{n!},$$

что является разложением $u(t) = e^t$ в степенной ряд — то же самое решение, к которому мы пришли методом разделения переменных (хотя и в другой форме). Отлично!

Когда такой итерационный метод используется при построении решения динамической системы или дифференциального уравнения в частных производных, переформулированного как динамическая система, или в форме, пригодной для метода простой итерации ($u = F(u)$), он называется *методом Пикара*. Он довольно прост и приходит к решению (когда сходится) последовательно по шагам.

В чем заключается сложность?

Так почему бы нам не использовать *метод Пикара* при построении решений *любых* динамических систем и *любых* дифференциальных уравнений в частных производных, которые можно переформулировать в форму, подходящую для метода простой итерации? Как всегда, мы упираемся в проклятие размерности. Даже в нашем довольно простом *одномерном* линейном примере на каждом шаге итерации Пикара требуется вычислить интеграл, а в более сложных задачах его приходится вычислять численно. Например, в случае динамических систем, представляющих изменения и взаимодействие множества частиц, интеграл умножается на количество частиц. Во всей литературе, посвященной обыкновенным дифференциальным уравнениям и дифференциальным уравнениям в частных производных встречается лишь ограниченное количество случаев с доступными практическими алгоритмами для высокоразмерных систем.

Последние достижения!

Тем не менее метод Пикара лег в основу метода нахождения явных решений для высокоразмерных нелинейных параболических дифференциальных уравнений в частных производных и обратных стохастических дифференциальных уравнений, который недавно продемонстрировал превосходные результаты при нахождении явных решений для высокоразмерных дифференциальных уравнений в частных производных, которые встречаются в практических приложениях в области физики и финансов. Приведем довольно познавательную выдержку из аннотации к работе Вэйнань Э. и др. (2017 г., **https://oreil.ly/APr1c**):

> "Параболические дифференциальные уравнения в частных производных (ДУЧП) и обратные стохастические дифференциальные уравнения (ОСДУ) считаются ключевыми компонентами ряда моделей в физике и финансовой инженерии. В частности, параболические ДУЧП и ОСДУ являются фундаментальными инструментами в современном ценообразовании и хеджировании финансовых деривативов. Зачастую ДУЧП и ОСДУ в таких приложениях — высокоразмерные и нелинейные. Поскольку их явные решения в большинстве случаев недоступны, довольно активной темой исследований стало приближенное решение этих уравнений. В опубликованной не так давно работе (Вэйнань Э. и др. Линейные масштабируемые алгоритмы для решения высокоразмерных нелинейных параболических дифференциальных уравнений; **https://arxiv.org/abs/1607.03295v3**) было рассмотрено семейство методов аппроксимации на основе аппроксимаций Пикара и многоуровневых методов Монте-Карло и показано, что при соответствующих предположениях о регулярности точного решения полулинейных тепловых уравнений вычислительная сложность ограничена значением $O\left(d\varepsilon^{-(4+\delta)}\right)$ при любом $\delta \in (0;\infty)$, где d — размерность задачи, а $\varepsilon \in (0;\infty)$ — заранее заданная точность. При помощи численного моделирования, представляющего зависимость точности аппроксимации от времени прогона, была выполнена проверка применимости данного алгоритма к различным 100-мерным

нелинейным ДУЧП, которые используются в областях физики и финансов. Моделирование этих 100-мерных нелинейных ДУЧП показало удовлетворительные результаты в плане точности и скорости. Дополнительно в статье приведен обзор других рассматриваемых в научной литературе методов аппроксимации нелинейных ДУЧП и ОСДУ".

Подготовка к глубокому обучению дифференциальным уравнениям в частных производных

Прежде чем мы завершим этот раздел, необходимо определиться с тем, как решать дифференциальные уравнения в частных производных и обратные стохастические дифференциальные уравнения в контексте глубокого обучения, в частности, для *глубоких операторских сетей*. Продолжим пример с одномерной динамической системой, только на этот раз выделим зависимость от физических параметров $a(t)$ и сделаем ее чуть более общей, добавив еще одну явную зависимость от времени:

$$\frac{du(t)}{dt} = \vec{f}\left(u(t), a(t), t\right);$$
$$\vec{u}(t_0) = \vec{u}_0.$$

Как и в прошлый раз, интегрируем единожды по времени:

$$u(t) = u_0 + \int_{t_0}^{t} f\left(u(s), a(s), s\right) ds.$$

Цель нейросети — принять на вход данные, что-то с ними сделать, а затем на выходе выдать то, что нас интересует. В случае обыкновенного дифференциального уравнения или дифференциального уравнения в частных производных нас интересует, конечно, выход — решение $u(t)$. Запишем это решение как выход некоторого оператора *G*, принимающего на вход исходные данные обыкновенного дифференциального уравнения или дифференциального уравнения в частных производных. В случае динамической системы входными данными будет выступать функция $a(t)$, представляющая физические параметры динамической системы. Важно отметить, что правую часть функции *f* динамической системы вводить не нужно. Это заложено в обучающих данных, которые теперь будут выглядеть как пары (*обучающий вход*, *обучающий выход*) $= \left(a(t), u(t)\right)$. Отсутствие ввода *f* в некотором смысле означает, что нам не важна точная форма обыкновенного дифференциального уравнения или дифференциального уравнения в частных производных, обусловливающая такое поведение, но независимо от этого, мы все равно можем *изучить*, что делает система. В этом суть машинного обучения: отсутствие необходимости в кодировании правил, которым подчиняется система, — модель способна смоделировать их, лишь наблюдая нужное количество примеров.

Мы можем записать решение $u(t) = G(a(t))$, где оператор решения G должен быть обучен при помощи нейросети. Пока запомним эту формулировку и вернемся к ней позже в этой главе, когда будем обсуждать нейросетевые операторы. Если подставить $u(t) = G(a(t))$ в интегральное уравнение, мы получим, что оператор решения, который нужно обучить с помощью нейросети, будет соответствовать:

$$G(a(t)) = u_0 + \int_{t_0}^{t} f(G(a(s)), a(s), s)ds$$

С этим интегральным уравнением мы больше ничего не будем делать. Оно просто показывает *истинное* свойство, которому соответствует интересующее нас $G(a(t))$. В прошлых примерах мы аппроксимировали его методом Пикара, а в новую эру глубокого обучения это можно сделать при помощи сети глубоких операторов (чуть дальше мы обсудим это подробнее). Такой подход к глубокому обучению становится более эффективным с вычислительной точки зрения, если для ускорения вычислений мы включаем преобразование Фурье. Кроме того, глубокое обучение как метод охватывает более широкий круг задач, и его можно применить к большему числу обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных, чем просто динамические системы. Динамическую систему можно легко проинтегрировать один раз и получить представление, которое приблизит нас к решению, чего нельзя сказать о многих обыкновенных дифференциальных уравнениях и дифференциальных уравнениях в частных производных.

Независимость сетки и различные разрешения

В заключение добавим, что пара "вход — выход" для нейросети, обучающейся решению динамической системы, будет выглядеть следующим образом: (*обучающий вход*, *обучающий выход*) $= (a(t), u(t))$. Поскольку машины воспринимают не функции, а исключительно числовые значения, при реализации потребуется дискретизация. Именно это отличает *операторы, действующие на функции*, от *функций, действующих на точки*, и обеспечивает нейросетям *независимость от сетки*, т. е. при одинаковых значениях t дискретизации $a(t)$ и $u(t)$ не потребуется. Для нас важнее всего отобразить одну функцию в другую, поэтому дискретизированный $a(t)$ можно рассматривать как вектор, отображенный на другой вектор, представляющий собой дискретизированный $u(t)$, причем необязательно в тех же точках или одного размера. Точно так же можно обучать сеть на заданном разрешении, а затем делать предсказания на другом разрешении. Это весьма актуально в области обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных, где качество численных решений всегда ограничивалось разрешением используемой дискретизации.

Искусственный интеллект применительно к дифференциальным уравнениям в частных производных

Итак, после обзора основных проблем и базовых подходов к решению дифференциальных уравнений в частных производных мы готовы рассмотреть, как к ним применяется ИИ, не ограничиваясь лишь одними упоминаниями о нем или его контекстом. В первую очередь, нужно выделить несколько направлений, по которым глубокое обучение попало в семейство дифференциальных уравнений в частных производных:

- использование глубокого обучения для нахождения значений физических параметров дифференциального уравнения в частных производных;
- использование глубокого обучения для расчета двумерных и трехмерных сеток при численном и твердотельном моделировании;
- использование глубокого обучения для поиска оператора решения дифференциального уравнения в частных производных, когда нейросеть обучается отображению в двух *бесконечномерных пространствах*;
- использование глубокого обучения для обхода дифференциальных уравнений в частных производных и моделирования природных явлений непосредственно на основе данных наблюдений (системы частиц и их взаимодействие).

Глубокое обучение для нахождения значений физических параметров

Нейросеть можно использовать для вывода параметров модели дифференциального уравнения в частных производных и их неопределенностей. Обучающие данные берутся из экспериментов (реальных или лабораторных путем симуляции известных явлений с известными параметрами). Эти данные размечаются значениями параметров так, что нейросеть обучается отображать начальную настройку определенного дифференциального уравнения в частных производных с соответствующими значениями параметров, что обеспечивает более точные результаты моделирования. Исторически сложилось так, что параметры, которые не поддавались измерению напрямую, приходилось угадывать или подбирать вручную, чтобы они соответствовали какому-то наблюдаемому поведению, что негативно сказывалось на всем процессе моделирования. С применением глубокого обучения удалось добиться более достоверных результатов моделирования дифференциального уравнения в частных производных. Теперь можно узнать значения параметров из размеченных изображений, выполненных в ходе экспериментов, записанных аудиозаписей и других неструктурированных или очень высокоразмерных данных. После обучения нейросети могут оценивать параметры и неопределенности для любых входных данных с аналогичными параметрами. Перейдя по

ссылке (**https://oreil.ly/ju4Kx**), можно ознакомиться с замечательным простым примером использования глубокого обучения в предсказании параметров поля скоростей в G-уравнении (моделирующем процесс горения) на основе изображений пламени, который называется "Байесовский вывод в нелинейных моделях пламени на основе физики".

Глубокое обучение для расчета сеток

В этой главе мы узнали, что генерация сетки является неотъемлемой частью метода конечных элементов, который, в свою очередь, находит численные решения для широкого спектра дифференциальных уравнений в частных производных, моделирующих природные явления со сложной геометрией области. Качество базовой сетки влияет на качество численного решения. Чем мельче сетка, тем больше истинного решения она может отразить, но тем больше и вычислительные затраты. Идеальная сетка должна быть плотной, и тогда ошибка между численным истинным решением, вероятно, будет велика, и крупной, и тогда ошибка будет мала, что позволяет сохранить точность, причем общие вычислительные затраты останутся приемлемыми (рис. 13.12).

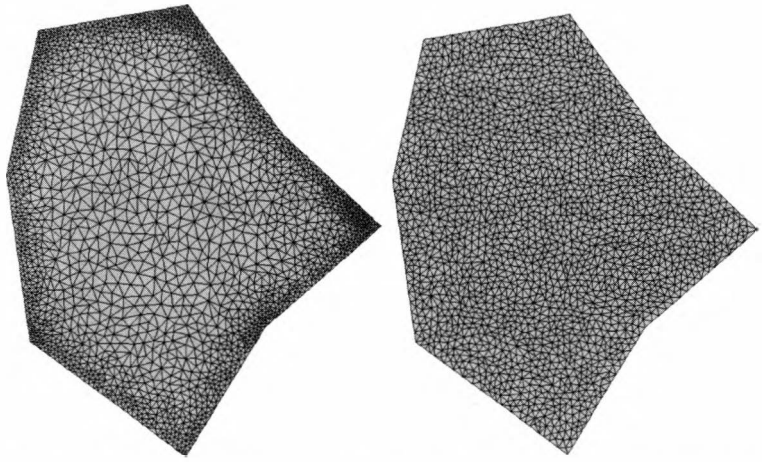


Рис. 13.12. Неравномерная (*слева*) и равномерная (*справа*) сетки; там, где ошибка велика, сетка должна быть тоньше (источник изображения: **https://oreil.ly/4XvB1**)

Было бы неплохо, если бы, получив на вход дифференциальное уравнение в частных производных, геометрию области, граничные условия и значения параметров, можно было обучить нейросеть автоматически генерировать идеальную сетку, предсказывая распределение плотности ее элементов в каждой точке области. Такую задачу решает сеть MeshingNet (**https://oreil.ly/0cAgv**).

До ее появления расчет сеток осуществлялся при помощи довольно затратных многошаговых конечно-элементных решений и оценок ошибок. MeshingNet, напротив, предсказывает идеальные сетки на основе аналогичных задач. Она начинает с исходной равномерной грубой сетки и предсказывает плотность неравномерной сетки

с целью ее доработки. Отличительной чертой глубокого обучения является то, что она хорошо обобщается на различные геометрические области с различными определяющими дифференциальными уравнениями в частных производных, граничными условиями и параметрами.

Входными данными для MeshingNet являются определяющее дифференциальное уравнение в частных производных, а также его параметры, геометрия области и граничные условия, а выходными — верхнее граничное распределение области $A(X)$ по всей области. Причем отображение между входом и выходом получается крайне нелинейным, что поддается обучению нейросети, продемонстрировавшей впечатляющую способность выражать многие виды нелинейных отношений.

Для создания обучающего набора данных команда MeshingNet вычисляет высокоточные решения на однородных сетках высокой плотности с помощью стандартных решателей конечных элементов. То же самое проделывается для однородных сеток с низкой плотностью, и получаются менее точные решения. Затем, интерполируя между этими решениями, команда вычисляет распределение ошибок $E(X)$. Причем $E(X)$ используется в качестве руководства для уточнения $A(X)$. Обучающие данные пополняются за счет комбинирования разных геометрий с различными параметрами и граничными условиями.

Глубокое обучение для трехмерных сеток

Трехмерные сетки (рис. 13.13) используются в компьютерной графике, анимации для индустрии развлечений, твердотельном моделировании. Их также применяют при воссоздании текстурированных и реалистичных поверхностей из заданного набора трехмерных точек данных. К традиционным методам относятся триангуляции Делоне и диаграммы Вороного, которые интерполируют точки с помощью треугольных сеток. Однако при наличии шума в координатах итоговая поверхность получится слишком грубой, что потребует предварительной обработки данных.

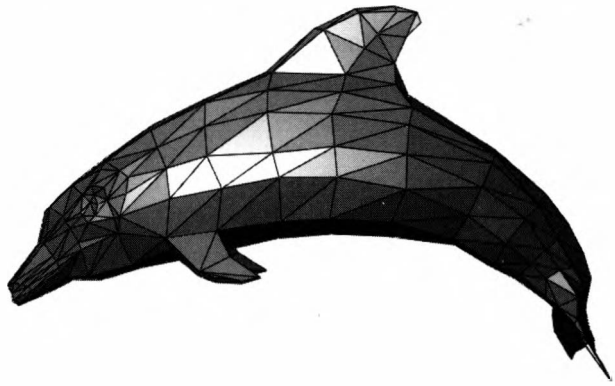


Рис. 13.13. Трехмерная сетка (источник изображения: **https://oreil.ly/2bG2x**)

Глубокое обучение позволяет генерировать трехмерные сетки более высокого качества. Например, в работах "Глубокий гибридный самостоятельный алгоритм для создания полной трехмерной сетки" (Вэй С. и др., 2021; **https://oreil.ly/mm79h**) и

"Pixel2Mesh: создание трехмерных сетчатых моделей на основе отдельных RGB-изображений" (Вэнг Н. и др., 2018; **https://oreil.ly/5rADG**) рассматривается получение трехмерной формы в треугольной сетке из одного цветного изображения путем непрерывной деформации эллипсоида.

Глубокое обучение для аппроксимации операторов решения дифференциальных уравнений в частных производных

Мы уже неоднократно касались этой темы в текущей главе. Речь идет об использовании глубокого обучения не для усовершенствования уже имеющихся методов, как, например, нахождение значений физических параметров из данных при решении дифференциальных уравнений в частных производных или расчет эффективных сеток при использовании численных методов, а для обучения оператору решения дифференциального уравнения в частных производных. В этом случае входные данные дифференциального уравнения в частных производных — область, физические параметры, начальное/конечное состояние решения, и/или граничные условия — отображаются непосредственно в его решении. Можно представить это как:

> *решение ДУЧП = функция(физические параметры, область, граничные условия, начальные условия и т. д.).*

Необходимо создать нейросеть и аппроксимировать эту функцию. По сути, это оператор, а не функция в обычном смысле, т. к. он переводит функции в другие функции. Оговорка здесь в том, что дифференциальные операторы, а также обратные дифференциальные операторы *отображают пространства бесконечной размерности* в *пространства бесконечной размерности*, иногда линейно, как, например, отображение *правой части* уравнения Пуассона в решение, но чаще всего нелинейно, как, например, отображение *параметров* уравнения Пуассона в решение. В отличие от этого, входы и выходы нейросетей, как мы узнали по ходу чтения этой книги, имеют конечную размерность (входами и выходами выступают векторы, изображения, графы и т. д.). Такие нейросети способны аппроксимировать *отображения функций* между *пространствами конечной размерности.* Они опираются на мощную универсальную теорему аппроксимации и используются во множестве успешных практических приложений (если не накладывать ограничений на ширину и глубину скрытых слоев, то с помощью нейросетей можно аппроксимировать любую непрерывную функцию с произвольной точностью). Для решения дифференциальных уравнений в частных производных аналогичным способом необходимо ответить на два вопроса.

Могут ли нейросети аппроксимировать отображения между пространствами бесконечной размерности?

> То есть могут ли они аппроксимировать любой нелинейный непрерывный функционал (на входе сети — функция или набор функций, на выходе — вещественное число) или нелинейный оператор (на входе сети — функция или набор функций, на выходе — другая функция)? Ответ — *да*!

По аналогии с *универсальной теоремой аппроксимации* для *функций нейросетей* есть такие же теоремы для *операторов нейросетей*. Нейросеть с одним скрытым слоем может точно аппроксимировать любой нелинейный непрерывный функционал или оператор. Более того, можно обучить нейросеть оператору решения целого семейства дифференциальных уравнений в частных производных, в отличие от классических методов, позволяющих решить только одно уравнение.

Как это реализуется на практике?

В случае конечной размерности узел нейросети линейно комбинирует признаки конечной размерности входного вектора (или выходы предыдущего слоя), добавляет член смещения, применяет нелинейную функцию активации и передает результат следующему слою. В условиях бесконечной размерности, когда больше нет конечного числа записей, чтобы выполнить линейную комбинацию, аналогом будет интегрирование произвольного обучаемого ядра (функции умножения) входных функций (при численном интегрировании требуется осуществить выборку в конечном числе точек, преобразуя интегрирование в сложение), добавление функции смещения (необязательно), применение нелинейной функции активации и передача результата в следующий слой. На следующем слое добавляются кратные результаты узлов предыдущего слоя и интегрируются с обучаемым ядром (*kernel*) результатов узлов предыдущего слоя, и т. д. Например:

$$u_{n+1}(x) = \sigma\left(\int_D kernel(x, s, a(x), a(s); \omega) u_n(s) ds + W u_n(x)\right),$$

где после определенного количества глобальных интегрирований по ядру, локальных линейных преобразований и композиций с нелинейной функцией активации мы итеративно приходим к решению $u(x)$. Параметрами ядра в итерационном процессе выступают ω и записи W. В процессе обучения нейросеть узнает эти параметры из размеченных данных (размеченных решениями дифференциального уравнения в частных производных) путем минимизации функции потерь. Как и в случае конечной размерности, нейросети аппроксимируют нелинейные операторы, комбинируя линейные интегральные операторы, действующие глобально по всей области, с нелинейными функциями активации. Приведенная итерационная формула также содержит локальный линейный множитель, который при дискретизации превращается в матрицу.

Нейросети для обучения выведенным операторам решений

Сделаем небольшую паузу и сравним полученное выражение с тремя истинными операторами решения, которые мы вывели для уравнения тепла, уравнения Пуассона и динамических систем. К ним легко адаптируется итерационный процесс нейронного оператора.

- В случае уравнения тепла в одном пространственном измерении с постоянными коэффициентами оператор решения отображает начальное состояние и физический параметр (константу) дифференциального уравнения в частных производ-

ных в решение $u(x,t)$. К счастью, для всех задействованных величин у нас имеются явные формулы:

$$G(u_0(x),a)=u(x,t)=\int_{-\infty}^{\infty}\frac{1}{\sqrt{4\pi at}}e^{-\frac{(s-x)^2}{4at}}u_0(s)ds=$$

$$=\int_{-\infty}^{\infty}kernel(s,x;t;a)u_0(s)ds.$$

Тогда для аппроксимации истинного оператора нейронный оператор выполнит следующую итерацию:

$$G(u_0(x),a)=u(x,t)\approx u_{n+1}(x,t)=$$

$$=\sigma\left(\int_D kernel(s,x;t;a;w)u_n(s)ds+Wu_n(x)\right).$$

- В случае уравнения Пуассона в двух измерениях пространства с нулевыми граничными условиями и постоянными коэффициентами оператор решения отображает правую часть f и физические параметры (константы) дифференциального уравнения в частных производных в решение $u(x,y)$, но только для определенных простых геометрий. Нам снова повезло, и для всех задействованных величин (ни одна из которых здесь не пишется) у нас есть явные формулы:

$$G(f(x,y),a)=u(x,y)=\int_D \text{Функция\_Грина}(x,y;s,p;a)f(s,p)dsdp.$$

Тогда для аппроксимации истинного оператора нейронный оператор выполнит следующую итерацию:

$$G(f(x,y),a)=u(x,y)\approx u_{n+1}(x,y)=$$

$$=\sigma\left(\int_D kernel(x,y,s,p,a;\omega)u_n(s,p)dsdp+Wu_n(x,y)\right).$$

- В случае одномерной динамической системы оператор решения отображает физический параметр (функцию) дифференциального уравнения в частных производных в решение $u(t)$, и мы получаем отвечающее ему неявное интегральное уравнение:

$$G(a(t))=u(t)=u_0+\int_{t_0}^{t}f(G(a(s)),a(s),s)ds.$$

Тогда для аппроксимации истинного оператора нейронный оператор выполнит следующую итерацию:

$$G(a(t))=u(t)\approx u_{n+1}(t)=\sigma\left(\int_{t_0}^{t}kernel(t,s,a(t),a(t);\omega)u_n(s)ds+Wu_n(t)\right).$$

В данном случае одна точка данных представляет собой триплет $\left(t, a(t), G\left(a(t)\right)\right)$, и, таким образом, один конкретный входной сигнал a может появляться в нескольких точках данных с разными значениями t. Например, набор данных размером 10 000 можно сгенерировать только из 100 траекторий $a(t)$, каждая из которых оценивает $G\left(a(t)\right)$ для 100 местоположений t.

Исходя из вышеизложенного, можно заключить, что нейронному оператору требуются только входные и выходные данные безотносительно к лежащим в их основе дифференциальным уравнениям в частных производных. Вся информация о дифференциальных уравнениях в частных производных неявно содержится в обучающих данных. Учитывая это, можно усилить форму *ввода-вывода* нейронного оператора:

решение ДУЧП ≈ обученный оператор(физические параметры, область, граничные условия, начальные условия и т. д.).

Основные вопросы

Выходя за рамки этой книги и углубляясь в нейронные операторы, нужно сфокусироваться на следующих вопросах.

Какие входные данные поступают в сеть при заданном дифференциальном уравнении в частных производных и каким будет выход?

Мы рассмотрели их в простых контекстах уравнения тепла, уравнения Пуассона и динамических систем.

Какой пример архитектуры нейронного оператора можно привести?

На рис. 13.14 изображена структура входа и выхода оператора DeepONet (**https://oreil.ly/II3kg**), где вход — дискретизированная пара $\left(t, a(t)\right)$, а выход — дискретное $G\left(a(t)\right)$.

Как справиться с тем, что входные данные имеют столь резкие различия в размерности, например конечную и бесконечную размерность одновременно?

Другими словами, как дискретизировать задействованные величины конечной (независимые переменные, например время и пространство) и бесконечной размерности (функции решения, функции параметров, граничные условия, начальные условия и т. д.) в процессе обучения и вывода? Важно отметить, что в случае нейросетей, обучающихся отображениям конечной размерности, входные данные (таблицы, изображения, аудиофайлы, графики, текст на естественном языке) всегда имеют одну и ту же размерность, для чего они предварительно обрабатываются, или же сеть сама обрабатывает порции входных данных фиксированной размерности по отдельности.

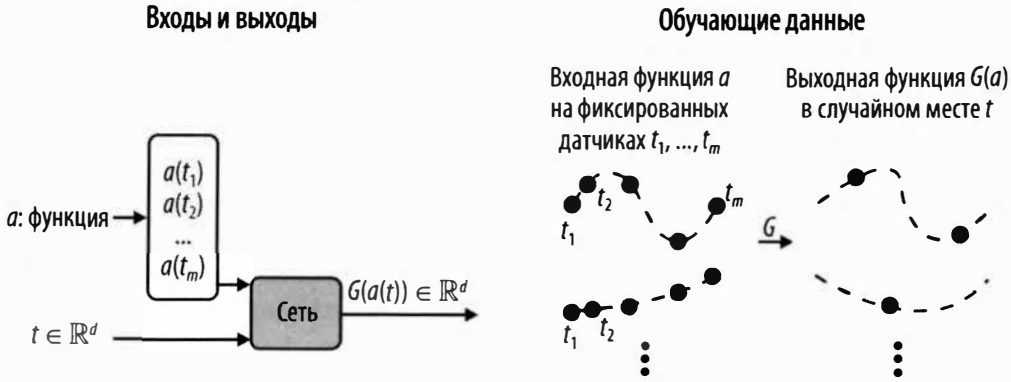


Рис. 13.14. Для обучения оператора $G(a(t))$ сеть принимает два входа $\left(a(t_1), a(t_2), ..., a(t_m)\right)$ и t (*а*). Иллюстрация обучающих данных (*б*). Источник изображения: **https://oreil.ly/oRgjA**

Как избежать распространенной ловушки многих методов решения дифференциального уравнения в частных производных, которые в итоге оказываются зависимыми от дискретизации?

В каком смысле нейронные операторы являются бессеточными и способны обобщать свои выученные параметры для работы с другими дискретизациями, отличными от тех, на которых они обучались? Существенным достижением здесь является то, что нейронные операторы считаются инвариантными в плане дискретизации и имеют одинаковые параметры сети во всех случаях дискретизации. Это означает, что, т. к. их выходные данные не зависят от базовой дискретизации, их можно использовать с различными представлениями сетки.

Как ускорить время вычисления интегралов, задействованных в нейронных операторах, и сократить затраты?

Для этого используется преобразование Фурье. Нейросеть Фурье ускоряет процесс вычисления интегралов, преобразуя входные данные в пространство Фурье. В ее распоряжении имеются методы быстрого преобразования Фурье. Такую нейросеть мы рассмотрим в следующем подразделе.

Как нейронные операторы справляются с высокоразмерными дифференциальными уравнениями в частных производных, содержащими сотни или тысячи переменных?

Дифференциальные уравнения в частных производных, моделирующие финансовые рынки со всеми базовыми активами (модель Блэка — Шоулза), теоретико-игровые модели со многими участниками (Гамильтона — Якоби — Беллмана) или физические системы со множеством частиц, являются очень высокоразмерными. Дискретизация в каждом из этих измерений увеличивает размер и без того большой задачи с точки зрения вычислений, так что до недавнего времени практическая реализация этих элегантных дифференциальных уравнений в частных производных считалась невыполнимой. Методология их решения с помощью ИИ рассматривается в статье "Решение высокоразмерных дифференци-

альных уравнений с помощью глубокого обучения" (Хан Ц. и др., 2018; **https://oreil.ly/vRNYr**), которую мы вскоре обсудим, при этом было бы неплохо приложить приведенные в статье методы к глубокому нейронному оператору.

Нейросеть Фурье

Калифорнийский технологический институт недавно выложил в открытый доступ свою нейросеть Фурье для решения дифференциальных уравнений в частных производных. Ее методология приведена в статье "Нейронный оператор Фурье для параметрических дифференциальных уравнений в частных производных" (Ли З. и др., 2021; **https://oreil.ly/IQ23v**). Такие сети могут аппроксимировать операторы решения для нелинейных дифференциальных уравнений в частных производных с высокочастотными модами и медленным затуханием энергии.

Каждый слой нейросети Фурье применяет к входным данным быстрое преобразование Фурье, далее линейное преобразование, а затем обратное быстрое преобразование Фурье. В результате возникает квазилинейная вычислительная сложность, т. е. порядка $O\left(n \text{ многочлен}\left(\log\left(n\right)\right)\right)$, и модель становится инвариантной к пространственному разрешению данных (хотя для этого по-прежнему требуется равномерная сетка).

Архитектура нейросети Фурье изображена на рис. 13.15.

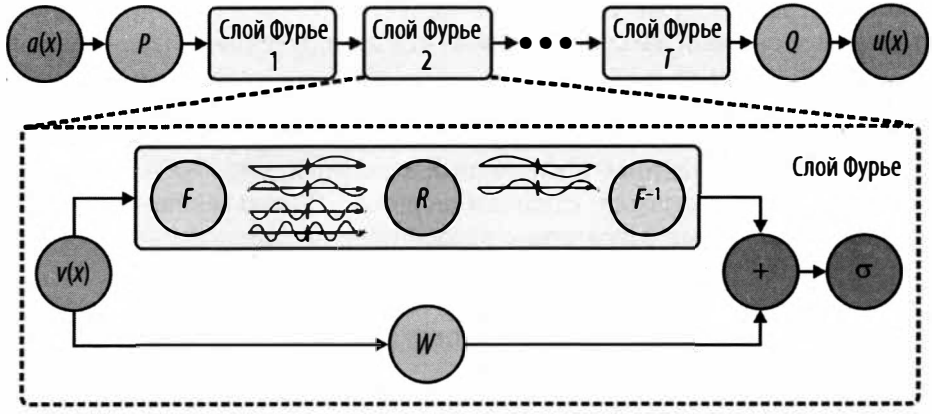


Рис. 13.15. Архитектура нейросети Фурье (источник изображения: **https://oreil.ly/Ik39S**)

Входные данные — физический параметр $a\left(x\right)$, выход — решение дифференциального уравнения в частных производных $u\left(x\right)$:

1. Начинаем с входа $a\left(x\right)$.
2. Поднимаемся в пространство каналов более высокой размерности с помощью неглубокой полносвязной нейросети P: $v0(x) = P\left(a(x)\right)$.
3. Применяем несколько слоев Фурье интегральных операторов и функций активации. В каждом слое применяем преобразование Фурье $F.T$, на низших модах Фурье используем линейное преобразование R и отфильтровываем высшие мо-

ды, а также применяем обратное преобразование Фурье $F.T^{-1}$. Снизу применяем локальное линейное преобразование W.

4. Проецируем обратно в целевую размерность с помощью нейронной сети Q. Завершаем работу выходом $u(x)=Q\big(vT(x)\big)$, что представляет собой проекцию v на локальное преобразование Q, также параметризованное неглубокой полносвязной нейросетью.
5. Завершаем выходом $u(x)$.

В статье продемонстрирован метод на примере разнообразных актуальных дифференциальных уравнений в частных производных:

- уравнение Бюргерса;
- поток Дарси;
- уравнение Навье — Стокса;
- турбулентные потоки в режимах, где расходятся остальные методы.

Нейросеть Фурье инвариантна к сетке, поэтому способна обучаться на более низком разрешении и оцениваться на более высоком разрешении без обращений к данным более высокого разрешения (сверхразрешение с нулевым снимком).

Учитывая, что методы на основе данных зависят от качества и количества данных, необходимо генерировать обучающие пары входов и выходов для нейросети, решая реальные дифференциальные уравнения в частных производных при помощи других методов. В этой связи авторы отмечают, что для обучения уравнению Навье — Стокса с *вязкостью* $=1e^{-4}$ требуется при помощи численного решателя сгенерировать $N=10\,000$ обучающих пар $\big(a(x),u(x)\big)$. В случае более сложных дифференциальных уравнений в частных производных генерация даже нескольких обучающих выборок может оказаться слишком дорогой. В дальнейшем можно будет комбинировать нейронные операторы с численными решателями, что позволит повысить требования к данным.

Утверждение универсальной теоремы аппроксимации для операторов

Предположим, что σ — непрерывная неполиномиальная функция, X — банахово пространство, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ — два компактных множества, V — компактное множество в $C(K_1)$, G — нелинейный непрерывный оператор, отображающий V в $C(K_2)$. Тогда при любом $\varepsilon > 0$ существуют положительные целые числа n, p, m и константы c_i^k, $\xi_{i,j}^k$, θ_i^k, $\xi_k \in \mathbb{R}$, $\omega_k \in \mathbb{R}^d$, $x_j \in K_1$, $i=1,\ldots,n$, $k=1,\ldots,p$, $j=1,\ldots,m$, так что

$$\left| G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{i,j}^k u(x_j) + \theta_i^{k\cdot}\right)\sigma(\omega_k.y + \xi_k)\right| < \varepsilon$$

выполняется для всех $u \in V$ и $y \in K_2$. Важно отметить, что в данном случае теорема аппроксимации использует только один скрытый слой в нейросети, не указывая,

сколько у него узлов. В приложениях, как и в случае конечной размерности, используется более одного слоя.

Не стоит пугаться больших слов и греческих букв. Теорема показывает, что у нас имеются теоретические основания сформулировать оператор нейросети с расчетом на то, что он будет отлично аппроксимировать оператор решения дифференциального уравнения в частных производных. Даже если мы никогда не узнаем его точную формулу, полученная нейросеть прекрасно заменит его. Именно поэтому нам так нравятся теоремы аппроксимации, и мы должны быть бесконечно благодарны математикам, работающим над ними.

Об этом мы уже говорили ранее в контексте *аддитивной* аппроксимации решения методом простой итерации, но об этом стоит сказать еще раз: аппроксимируя либо отображение между пространствами конечной размерности, либо отображение между пространствами бесконечной размерности, нейросети представляют функции, функционалы или операторы, используя *композиции* простых функций (линейная комбинация или линейный интегральный оператор в сочетании с нелинейной функцией активации) для аппроксимации сложных функций. Такой подход отличается от классических методов, использующих аддитивную, а не композиционную аппроксимацию.

Как расширить границы и погрузиться в более технические детали?

Для более подробного ознакомления с нейронным оператором можно выделить три важные публикации на эту тему:

- "DeepONet: обучение нелинейным операторам для идентификации дифференциальных уравнений на основе универсальной теоремы аппроксимации операторов" (Лу Л. и др., 2020; **https://oreil.ly/Do6du**);
- "Нейронный оператор: графовая сеть ядра для дифференциальных уравнений в частных производных" (Ли З. и др., 2020; **https://oreil.ly/RtWuT**);
- "Нейронный оператор Фурье для параметрических дифференциальных уравнений" (Ли З. и др., 2021; **https://oreil.ly/cUkhw**).

Численные решения высокоразмерных дифференциальных уравнений

Дифференциальные уравнения универсальны — они способны моделировать практически все, что только можно придумать, включая ежедневные поездки на работу и дорожное движение. После знакомства с дифференциальными уравнениями трудно не удержаться от мысли о том, что каждая ситуация, в которой мы оказываемся, подходит под то или иное дифференциальное уравнение. Тем не менее эту область с самого начала ее существования преследует проклятие размерности, встав на пути множества практических приложений. Именно поэтому многие вводные курсы по дифференциальным уравнениям в частных производных ошибочно сфокусированы только на одно- и двумерных дифференциальных уравнениях, как

будто этим все ограничено. Если бы ИИ можно было обозначить каким-то другим именем — не столь броским и которое не попадет ни в один фильм, то оно звучало бы так: *обработка, вычисление, анализ высокоразмерных данных*. Это не говорит о том, что ИИ не достоин упоминания, ведь обработкой, вычислением и анализом высокоразмерных данных люди занимаются изо дня в день при условии, что к этому добавляется творческое измерение (что в ИИ означает генеративные модели). Поэтому неудивительно, что глубокое обучение становится благоприятной средой для поиска численных решений дифференциальных уравнений *очень* высокой размерности. Этому посвящена статья "Решение высокоразмерных дифференциальных уравнений с помощью глубокого обучения" (Хан Ц. и др., 2018; **https://oreil.ly/Qw4gY**), в которой рассматриваются дифференциальные уравнения в частных производных с сотнями и даже тысячами измерений. Таким образом, можно одновременно учитывать все задействованные агенты, активы, ресурсы или частицы и не придумывать специально искусственные собственноручные предположения об их взаимодействии и связях. Авторы рассматривают несколько высокоразмерных дифференциальных уравнений в частных производных, в том числе уравнение Гамильтона — Якоби — Беллмана (какова оптимальная стратегия для каждого взаимодействующего агента среди сотен агентов?) и уравнение Блэка — Шоулза (какова справедливая цена еврооблигаций на основе ста базовых активов с учетом того, что дефолта пока еще не было?).

Если в основу моделей закладывается глубокое обучение, например при нахождении решений высокоразмерных дифференциальных уравнений в частных производных, первый вопрос, который необходимо задать: что является входом, а что — выходом сети глубокого обучения? В случае дифференциального уравнения в частных производных, решением которого является $u(x, t)$, в идеале на входе должны быть x и t, а на выходе — $u(x, t)$. Значение x в этом случае может означать чрезвычайно высокоразмерный $\vec{x}$. Когда в записи x присутствует стохастичность, например стоимость активов финансового рынка, необходимо моделировать их именно такими, а если нет, то обычно предполагается некое усреднение. В итоге в большинстве реалистичных случаев можно ввести x как стохастический процесс X. Это нужно задать математически.

Одним из важных шагов в вышеупомянутой статье является переформулировка высокоразмерных дифференциальных уравнений в частных производных в обратные стохастические дифференциальные уравнения перед вводом X в нейросеть, аппроксимирующую градиент решения. Для того чтобы освоить необходимую математику, нужно дать определения следующим понятиям:

- броуновское движение (*см. главу 11*);
- стохастический процесс (*см. главу 11*);
- стохастическое дифференциальное уравнение (в книге не рассматривается);
- отнесение нелинейных параболических дифференциальных уравнений в частных производных к стохастическим (в книге не рассматривается);

- обратное стохастическое дифференциальное уравнение (это выходит за рамки книги).

И мы должны ответить на вопрос: зачем нам понадобилось переформулировать дифференциальное уравнение в частных производных в стохастическую форму перед обучением сети? Какое преимущество дает нам эта форма? Все это выходит за рамки книги, но теперь мы знаем, что именно искать и какие вопросы задавать.

И наконец, этот метод открывает путь к решению многих высокоразмерных дифференциальных уравнений, хотя есть и ограничения. Его нельзя применить к квантовой задаче многих тел из-за сложности работы с принципом исключения Паули (**https://oreil.ly/XAyAz**).

Моделирование природных явлений непосредственно из данных

Системы частиц уже обсуждались в этой главе. Мы использовали рамки статистической механики для описания вероятностей состояний системы в масштабе частиц, а затем с их помощью записали дифференциальные уравнения в частных производных, моделирующие изменение системы во времени в макромасштабе.

В этом разделе мы рассмотрим, как современные модели на основе нейросетей моделируют систему частиц и предсказывают ее изменения *без* записи дифференциальных уравнений в частных производных. Другими словами, мы обходим их и меняем на *обучение на основе данных*.

Для того чтобы проследить изменение во времени системы частиц (например, воды или песка) в гранулярном масштабе, необходимо знать вектор положения каждой частицы $\vec{p}_i(t)$ на каждом временно́м шаге t. Изменение положения частицы зависит от локальных и дальнодействующих взаимодействий между частицей и ее соседями (например, обмен энергией и импульсом), продиктованных как физической природой системы, так и внешними эффектами, такими как, например, гравитация, температура, силы, магнитные поля и т. д. Можно обойтись без записи явных уравнений для таких взаимодействий и связывания их с положениями, скоростями и/или ускорениями частиц и вместо этого обучить нейросеть карте отображения заданного состояния системы частиц в определенное время (вход) и положения всех ее частиц (или скоростей или ускорений) в будущем (выход). Для моделирования систем частиц хорошо подходят графовые сети, т. к. каждая частица и ее состояние могут выступать узлом, а ребра со своими характеристиками — моделировать взаимодействие между конкретными частицами.

В качестве иллюстрации приведем и прокомментируем основные идеи из недавно опубликованной работы "Обучение моделированию сложной физики с помощью графовых сетей" (Санчес-Гонсалес А. и др., 2020; **https://oreil.ly/Q0EWH**), где рассматривается подобная карта отображения.

В первую очередь, нужны обучающие данные.

Мы можем генерировать пары входных (система частиц и ее характеристики в определенный момент времени) и целевых (ускорение каждой частицы в после-

дующий момент времени) данных из набора наблюдаемых или смоделированных траекторий определенной системы частиц. Например, из траектории длиной в 1000 шагов команда генерирует 995 пар, опираясь на 5 прошлых состояний. В наборах данных нас интересуют только векторы положения, а векторы скорости и ускорения можно получить, воспользовавшись методом конечных разностей. Как правило, наборы данных содержат 1000 обучающих, 100 проверочных и 100 тестовых траекторий, каждая из которых моделируется в течение 300–2000 временны́х шагов, что соответствует средней продолжительности приведения к устойчивому равновесию различных материалов.

Далее необходимо построить отображение от входа к выходу (компоненты сети).

Начнем с того, что в целочисленное время t система находится в определенном состоянии $X^t = \left(\vec{x}_0^t, ..., \vec{x}_N^t\right)$, где $\vec{x}_i^t$ каждой из N частиц представляет ее состояние в момент времени t (которое включает ее положение $\vec{p}_i^t$ и другие характеристики, такие как масса, свойства материала и т. д.).

Далее изучается отображение состояния $X^t = \left(\vec{x}_0^t, ..., \vec{x}_N^t\right)$ в виде графа $G =$ (узлы, ребра и глобальные свойства, которые можно альтернативно включить в качестве характеристик узлов). Эмбеддинги узлов $\overrightarrow{\text{узел}}_i = \text{функция}\left(\vec{x}_i\right)$ представляют собой обученные (с помощью многослойного перцептрона) функции состояний частиц. Для создания путей между узлами частиц, имеющих некоторое потенциальное взаимодействие, нужно добавить направленные ребра. Эмбеддинги ребер $\vec{e}_{i,j} = \text{функция}\left(\vec{r}_{i,j}\right)$ представляют собой обученные (с помощью многослойного перцептрона) функции парных свойств соответствующих частиц $\vec{r}_{i,j}$, например смещение их положений, постоянная пружины и т. д.

Далее изучается отображение графа в граф. При этом вычисляются взаимодействия между узлами через M шагов обучения передаче сообщений для создания последовательности обновленных латентных графов $G = G_1, ..., G_M$. В результате возвращается конечный граф. Обмен сообщениями позволяет распространять информацию между узлами через ребра и соблюдать ограничения. Таким образом, сложная динамика системы аппроксимируется обученным обменом сообщениями между узлами в их локальных окрестностях. Более того, конечный граф имеет ту же структуру, что и первый, но с потенциально различными атрибутами на уровне узлов, ребер и графов.

Затем обучается карта (многослойный перцептрон) от конечного графа к матрице, которая извлекает динамику системы, например матрицу ускорений частиц $Y = \left(\vec{p}_1'', \vec{p}_2'', ..., \vec{p}_N''\right)$. И наконец, положения и скорости частиц обновляются с помощью интегратора Эйлера от ускорений в Y, что, в свою очередь, обновляет состояние системы до X^{t+1}.

Такие модели не ограничиваются материалами и системами частиц и способны моделировать системы с множеством взаимодействующих агентов, например роботизированные системы управления. Их применение позволяет добиться достоверного моделирования сложных явлений, что крайне важно для науки и техники.

Дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана для динамического программирования

Уравнение Гамильтона — Якоби — Беллмана — это еще одно дифференциальное уравнение в частных производных, решение которого открывает широкие перспективы в экономике, исследовании операций и финансах, *если только* нам удастся решить его в высокой размерности. В двух словах, мы ищем *оптимальную стратегию* (например, инвестиционную), гарантирующую *минимальную* стоимость реализации в *заданный период времени*. В идеале желательно включить сотни или тысячи взаимодействующих агентов, например все финансовые активы для инвестиций, а не сокращать их число до нереалистичных моделей репрезентативных агентов. Как уже было показано в этой главе, можно при помощи нейросетей найти численные решения высокоразмерных дифференциальных уравнений в частных производных.

С точки зрения математики, дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана является довольно *насыщенным*. Оно сочетает в себе динамические системы $\left(\frac{\partial x(t)}{dt} = f\left(x(t), a(t), t\right)\right)$, дифференциальные уравнения в частных производных (частные производные и равенства) и оптимизацию (задачи на *максимум* или *минимум*). По мере того, как мы учимся выводить это единственное уравнение из реальных приложений, пытаемся понять его, находим решения и анализируем их (существование, единственность, гладкость и т. д.), мы осваиваем массу математических знаний.

Более того, это дифференциальное уравнение в частных производных напрямую связано с *обучением с подкреплением* в ИИ, но только теперь мы рассматриваем подкрепление не в вероятностном смысле в терминах марковских процессов принятия решений, как в *главе 11*, а в терминах детерминированного динамического программирования.

В рамках динамического программирования объединенные в вектор $\vec{x}(t)$ состояния взаимодействующих агентов изменяются во времени в соответствии с динамической системой, так что нужно найти некую оптимизирующую политику, порождающую специальное решение этой динамической системы — несущее минимальные затраты в течение заданного периода времени. Суть в следующем: некая зависящая от времени политика влияет на поведение динамической системы, которая, в свою очередь, влияет на понесенные затраты. Все это — математические величины.

Неоценимый вклад в область динамического программирования (поиск оптимальных стратегий для изменяющейся за определенный период времени системы) внес Ричард Беллман (1920–1984). Вскоре мы познакомимся с принципом оптимальности Беллмана, и, по сути, именно Беллман ввел в обиход термин *"проклятие раз-*

мерности". Принцип оптимальности крайне полезен: он разбивает задачу оптимизации на рассматриваемом отрезке времени на более мелкие подзадачи на меньших временны́х интервалах, которые можно решать рекурсивным способом.

Уравнение Беллмана в детерминированных и стохастических системах

В *детерминированном* динамическом программировании применяются следующие уравнения.

Уравнение Беллмана для дискретного времени.

Функцию ценности, начиная с текущего момента и до конечного времени, можно найти, выбрав наилучшую стратегию (или управление, или политику) a_k на текущем временно́м шаге k так, чтобы на следующем временно́м шаге минимизировать текущие расходы вместе с функцией ценности. Это рекурсивный процесс, имеющий вид:

$$\text{Ценность}(\vec{x}_k, n) = \min_{\vec{a}_k}\left(\text{Стоимость}(\vec{x}_k, \vec{a}_k) + \text{Ценность}(\vec{x}_{k+1}, n-1)\right),$$

где n — конечный шаг по времени, в то время как динамика по дискретному времени имеет вид:

$$\vec{x}_{k+1} = \vec{f}(\vec{x}_k, \vec{a}_k),$$

так что:

$$\text{Ценность}(\vec{x}_k, n) = \min_{\vec{a}_k}\left(\text{Стоимость}(\vec{x}_k, \vec{a}_k) + \text{Ценность}\left(\vec{f}(\vec{x}_k, \vec{a}_k), n-1\right)\right).$$

Последовательность оптимизаторов $\vec{a}_k$ на каждом дискретном временно́м шаге k составляет оптимальную стратегию (или политику, или управление) для всего периода времени и гарантирует минимальную общую стоимость, точно так же, как и в обучении с подкреплением.

Уравнение Беллмана для непрерывного времени.

Представляет собой дифференциальное уравнение Гамильтона — Якоби — Беллмана.

Кроме того, в случае *стохастического оптимального управления* также возникает стохастическая версия уравнения Беллмана. Она широко применяется в инвестиционном банкинге, а также в задачах планирования и маршрутизации. В стохастической модели необходимо найти оптимальный управляющий вход (стратегию или политику), направляющий базовые стохастические процессы к некоторому желаемому конечному состоянию с минимальными затратами. Рассмотрим, например, задачу, где в течение определенного периода времени требуется продать акции с минимальными издержками. Сначала мы можем смоделировать краткосрочную динамику базовых активов, а затем выполнить дискретизацию как во временно́м, так и в пространственном состояниях. Это обеспечит продажу заданного количества акций на каждом временно́м шаге при условии, что в течение заданного периода

времени требуется продать все акции. Нам нужна такая стратегия, которая среди всех действий, возможных в каждый момент времени, подскажет наиболее оптимальное, способное привести нас туда, куда мы хотим.

В *главе 11* мы связали уравнение Беллмана с обучением с подкреплением. Это сделано в контексте марковского процесса принятия решения для функции ценности:

$$\text{Ценность}(\vec{s}) = \max_{\substack{\text{состояния} \\ \text{и действия}}} \mathbb{E}\left(\text{вознаграждение}_0 + \gamma \cdot \text{Ценность}\left(\vec{s'}\right)\right).$$

В условиях детерминированного динамического программирования аналогичным уравнением выступает дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана для функции ценности. До того как записать его формулу, необходимо ознакомиться со следующими понятиями.

Минимизация функции стоимости.

Разве есть в нашем мире что-то более значимое?

Выбор оптимального управления или оптимальной стратегии.

Это искомый минимизатор, он управляет динамической системой.

Функция ценности.

Суммарные минимальные издержки за рассматриваемый период времени.

Принцип оптимальности Беллмана.

Весьма полезный принцип, позволяющий упростить задачу оптимизации.

Решение в обратном времени.

В этом случае мы начинаем с искомого результата и проделываем путь назад оптимально к начальному состоянию. Можно интуитивно понять, почему в данной ситуации решение в обратном времени окажется проще. Так как конечная цель нам известна, можно сразу же исключить из предыдущего временного шага все пути, которые не ведут к ней, что избавит от необходимости исследовать множество бесполезных путей. И наоборот, при решении задачи в прямом времени с начала временно́го интервала мы лишаемся преимущества *близости* к желаемому результату, поэтому вынуждены тратить время и вычислительные ресурсы на исследование большого количества бесполезных путей.

Общая картина

Главный вопрос, на который требуется ответить, звучит так: что необходимо сделать сейчас (каковы начальное состояние $\vec{x}(t_{\text{нач}})$ и зависящая от времени стратегия $\vec{a}(t)$), чтобы попасть куда нужно ($\vec{x}(t_{\text{кон}})$), максимально экономичным способом (достичь функции ценности $\text{Ценность}(\vec{x}(t_{\text{нач}}), t_{\text{нач}}, t_{\text{кон}})$, являющейся минимальным значением функции стоимости реализации стратегии)?

Здесь используются следующие величины:

- $\vec{x}(t)$ — вектор, характеризующий состояние динамической системы;

- $\vec{a}(t)$ — стратегия (или политика, или управление). Необходимо спроектировать ее так, чтобы она вызывала состояние $\vec{x}(t)$, минимизирующее некоторую функцию стоимости. То есть если в динамическую систему ввести искомую стратегию $\vec{a}(t)$, то на выходе $\vec{x}(t)$ будет минимизировать функцию стоимости;
- функция стоимости $\text{Стоимость}\left(\vec{x}(t), \vec{a}(t), t_{\text{нач}}, t_{\text{кон}}\right)$ — функция издержек, понесенных в результате реализации стратегии (или политики, или управления). Она задается некоторой конечной стоимостью в состоянии $t_{\text{кон}}$ при переходе от $t_{\text{нач}}$ к $t_{\text{кон}}$. Приращенные издержки могут возникнуть в зависимости от текущего состояния системы и текущего управления;
- функция ценности $\text{Ценность}\left(\vec{x}(t_{\text{нач}}), t_{\text{нач}}, t_{\text{кон}}\right)$ — минимальные издержки в конкретный период времени, достигаемые при реализации минимизирующей стратегии $\vec{a}^*(t)$, которая, в свою очередь, задает состояние $\vec{x}^*(t)$, используя информацию о динамике системы.

Дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана

Речь идет о соответствующих уравнениях и формулах:

$$\frac{\overrightarrow{dx(t)}}{dt} = \vec{f}\left(\vec{x}(t), \vec{a}(t), t\right);$$

$$\text{Стоимость}\left(\vec{x}(t), \vec{a}(t), t_{\text{нач}}, t_{\text{кон}}\right) =$$

$$= \text{Стоимость}_{\text{кон}}\left(\vec{x}(t_{\text{кон}}), t_{\text{кон}}\right) + \int_{t_{\text{нач}}}^{t_{\text{кон}}} \text{Стоимость}_{\text{приращенная}}\left(\vec{x}(s), \vec{a}(s)\right) ds;$$

$$\text{Ценность}\left(\vec{x}(t_{\text{нач}}), t_{\text{нач}}, t_{\text{кон}}\right) = \min_{\vec{a}(t)} \text{Стоимость}\left(\vec{x}(t), \vec{a}(t), t_{\text{нач}}, t_{\text{кон}}\right).$$

Принцип оптимальности Беллмана позволяет получить довольно полезную информацию о поведении функции ценности (оптимальной стоимости) по особой траектории $\vec{x}^*(t)$, соответствующей стратегии оптимизации $\vec{a}^*(t)$ — ценность на заданном временно́м интервале является суммой ценностей, если разбить временно́й интервал на отрезки вдоль особой траектории $\vec{x}^*(t)$, соответствующей стратегии оптимизации $\vec{a}^*(t)$. Таким образом, можно разбить задачу оптимизации на большом временно́м интервале на их рекурсию на гораздо меньших временны́х интервалах:

$$\text{Ценность}\left(\vec{x}^*(t_{\text{нач}}), t_{\text{нач}}, t_{\text{кон}}\right) = \text{Ценность}\left(\vec{x}^*(t_{\text{нач}}), t_{\text{нач}}, t_{\text{промежуточное}}\right) +$$

$$+\text{Ценность}\left(\vec{x}^*(t_{\text{промежуточное}}), t_{\text{промежуточное}}, t_{\text{кон}}\right).$$

На основе принципа Беллмана можно вывести дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана, удовлетворяющее функции ценности. Это уравнение обобщает более старое дифференциальное уравнение Га-

мильтона — Якоби — Беллмана для оптимального управления. В его решении содержится довольно ценная информация.

Предположим, что мы взаимодействуем с системой не только в ее начальном состоянии $t_{\text{нач}}$, но и в *любом состоянии* t, тогда функцию ценности можно вычислить до желаемой конечной стоимости, решив уравнение Гамильтона — Якоби — Беллмана:

$$-\frac{\partial \text{Ценность}}{\partial t} = \min_{\vec{a}(t)}\left(\left(\frac{\partial \text{Ценность}}{\partial \vec{x}}\right)^T \vec{f}\left(\vec{x}(t), \vec{a}(t)\right) + \text{Стоимость}_{\text{приращенная}}\left(\vec{x}(t), \vec{a}(t)\right)\right)$$

при соблюдении условия конечного времени:

$$\text{Ценность}\left(\vec{x}\left(t_{\text{кон}}\right), t_{\text{кон}}\right) = \text{Стоимость}_{\text{кон}}\left(\vec{x}\left(t_{\text{кон}}\right), t_{\text{кон}}\right).$$

Это дифференциальное уравнение в частных производных первого порядка для функции ценности $\text{Ценность}\left(\vec{x}(t), t, t_{\text{кон}}\right)$. И снова это оптимальная стоимость, возникающая при старте в состоянии $\vec{x}(t)$ в момент времени t, и оптимальное управление системой с этого момента до $t_{\text{кон}}$. Нам известна конечная функция ценности в момент времени $t_{\text{кон}}$, и нужно найти функцию ценности в момент времени t, т. е. $\text{Ценность}\left(\vec{x}(t)\right)$. Поэтому мы решаем дифференциальное уравнение в частных производных *в обратном времени*, начиная с $t_{\text{кон}}$ и заканчивая $t_{\text{нач}}$.

Решение уравнения Гамильтона — Якоби — Беллмана

Если нам удастся решить дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана для функции ценности, то мы найдем оптимальное управление $\vec{a}^*(t)$, которое, в свою очередь, создает наименее затратную (или наиболее выгодную) траекторию $\vec{x}^*(t)$ от текущего состояния $\vec{x}\left(t_{\text{нач}}\right)$ до нужного конечного состояния $\vec{x}\left(t_{\text{кон}}\right)$.

Дифференциальное уравнение в частных производных Гамильтона — Якоби — Беллмана в общем случае не имеет гладкого решения, поэтому придется довольствоваться *слабыми* или *обобщенными решениями*. Это общая проблема дифференциальных уравнений в частных производных, так что при изучении теоретического материала обучающийся фокусируется почти исключительно на разработке обобщенных решений и понимании функциональных пространств (пространства Соболева и т. д.). К классическим примерам обобщенных решений дифференциальных уравнений в частных производных Гамильтона — Якоби — Беллмана, которые мы лишь вскользь упомянули, можно отнести вязкостные и минимаксные решения.

Искусственный интеллект внес свой вклад в обширную литературу, посвященную уравнению Гамильтона — Якоби — Беллмана, предоставив его численное решение в крайне высоких размерностях — в сотнях и тысячах.

Функция ценности — это функция вектора состояния $\vec{x}(t)$ базовых активов или участвующих агентов, и когда их много, дифференциальное уравнение имеет довольно высокую размерность. В статье, на которую мы ссылались ранее, "Решение

дифференциальных уравнений в частных производственных с помощью глубокого обучения" (**https://oreil.ly/Tmg13**), рассматриваются численные решения уравнения Гамильтона — Якоби — Беллмана, а также других важных и широко распространенных высокоразмерных дифференциальных уравнений.

В решении уравнения Гамильтона — Якоби — Беллмана часто можно встретить термин *"формулы Хопфа"*. Для класса дифференциальных уравнений движения вязких несжимаемых жидкостей типа уравнения Гамильтона — Якоби авторы статьи "Алгоритмы преодоления проклятия размерности для некоторых уравнений Гамильтона — Якоби, возникающего в теории управления и других областях" (Дарбон Дж., Ошер С., 2016; **https://oreil.ly/lbPyt**) предлагают эффективный алгоритм на основе формул Хопфа.

Динамическое программирование и обучение с подкреплением

Использование нейросетей для обучения оптимальным стратегиям динамического программирования в одних сообществах называется *обучением с подкреплением*, в других — *нейродинамическим программированием*. Нейросеть и оснащенная ею машина обучаются предвидеть, как текущие и будущие действия влияют на долгосрочную совокупную стоимость или вознаграждение — ценность конкретного периода времени. Как наши текущие и ежедневные инвестиционные стратегии влияют на годовые финансовые показатели? Как наши первые и последующие ходы в шахматах влияют на общий исход игры? Функция ценности — это совокупность издержек и вознаграждений, относящихся к следованию оптимальным стратегиям на каждом (дискретном или непрерывном) шаге времени.

Для обучения нейросети с помощью исторических данных необходимо обеспечить входы и выходы. На входе будет состояние и все потенциальные действия, допустимые в этом состоянии, на выходе — ценность (общие затраты и вознаграждения). После обучения, например, бизнес-модели, разрабатывающей стратегию индивидуальной работы с клиентом, нейросеть учится принимать на вход состояние клиента и выдает последовательность очередных действий, максимизирующую ценность в долгосрочной перспективе. Несколько устаревшее, но достаточно подробное объяснение нейродинамического программирования и использования искусственных нейросетей для аппроксимации функции ценности в уравнении Беллмана можно посмотреть в работе "Нейродинамическое программирование" (Берцекас Д. и др., 1996; **https://oreil.ly/RXx0I**). Такой подход позволяет существенно ослабить последствия проклятия размерности, т. к. вместо сохранения и оценки всей высокоразмерной функции можно сохранять только параметры нейросети.

Дифференциальные уравнения в частных производных для искусственного интеллекта

В предыдущем разделе было отмечено, что динамическое программирование и уравнение Беллмана тесно связаны с обучением с подкреплением в ИИ.

Более того, область дифференциальных уравнений располагает целым арсеналом аналитических средств, изучающих все виды функций и функциональных пространств, слабые и сильные решения, а также все виды сходимости во всех смыслах. Если у какой-либо области и есть инструменты для раскрытия секретов успеха нейросетей в аппроксимации многих процессов, порождающих данные, будь то совместные распределения вероятностей или детерминированные функции, то это область дифференциальных уравнений в частных производных. Необходимо подкрепить нейросети теоремами и математической строгостью, которые в конечном итоге помогут в их проектировании и оптимизации архитектуры. Развитие магических способностей нейросетей через призму анализа и с помощью инструментов анализа дифференциальных уравнений — это один из путей продвижения вперед. В качестве примера можно привести метод обучения Соболева (Чарнецкий и др., 2017)[3].

Другие аспекты дифференциальных уравнений в частных производных

Бóльшую часть времени мы держались в стороне от известных дифференциальных уравнений, чтобы подчеркнуть, что рассматриваемая тематика применима не только к хорошо изученным дифференциальным уравнениям и приложениям.

Как правило, в вузовской программе рассматриваются линейные дифференциальные уравнения, содержащие только функции двух переменных (x, y) или (x, t). В итоге студенты либо ошибочно полагают, что на этом все и заканчивается, либо задаются вопросом насчет нелинейных уравнений, высокоразмерных приложений и систем дифференциальных уравнений. В этих программах также принято рассматривать такие уравнения, как уравнение теплопроводности (параболическое), волновое уравнение (гиперболическое), уравнение Лапласа (эллиптическое), а также некоторые численные решения и симуляции (методы конечных разностей, конечных элементов и метод Монте-Карло). Они представлены в своих простейших формах — линейных, одно-, двух- или трехмерных, — заданных на областях с регулярной геометрией, из-за чего у студентов складывается ложное впечатление, будто эти дифференциальные уравнения служат базой всех уравнений в приложениях. Кроме того, они искусственно разделяют уравнения *по типам* на эллиптические, параболические и гиперболические, как будто для каждого типа имеется своя полная теория. Аналитические методы решения весьма ограниченны и сфокусированы исключительно на принципе суперпозиции простых решений (из-за линейности), что приводит к рядам и преобразованиям Фурье (в действительности это довольно неплохо). Нейросети расширяют область применения в смысле аппроксимации решений нелинейных уравнений при помощи композиций простых функций вместо сложений.

---

[3] См. **https://arxiv.org/abs/1706.04859**. — *Прим. ред.*

Довольно неплохо выглядят учебные программы по дифференциальным уравнениям в частных производных для аспирантов, однако в них никак не отражена реальность этих уравнений — ни теоретически, ни численно, и даже не показано их широкое применение. Создается впечатление, что выпускники аспирантуры, столкнувшись с абсолютно новым дифференциальным уравнением в частных производных, даже не представляют, как к нему подступиться, поскольку оно не подходит ни к одному из тех, что они изучали на вводном курсе по дифференциальным уравнениям в частных производных (даю подсказку, что нужно сделать прежде всего, но только после поисков в Google: дискретизировать и смоделировать его, что даст отличное представление об особенностях его решения).

Тем не менее существуют общие подходы к составлению дифференциальных уравнений в частных производных (моделированию), которое почти всегда связано с законами сохранения из физики, их анализу (теория существования, единственности, анализ чувствительности решений и *слабых решений*) и нахождению фактических решений аналитическим или численным способом (формулы представления, функции Грина, методы преобразования, численные методы).

Начнем с того, что в каждой области знаний есть свои дифференциальные уравнения, моделирующие интересующие ее явления, например:

- в гидродинамике изучаются уравнения Навье — Стокса (среди прочих). Это нелинейная система дифференциальных уравнений в частных производных. В уравнении Навье — Стокса учитываются скорость жидкости, давление, плотность, напряжения, сжимаемость и действующие на нее силы. Оно выражает сохранение массы и сохранение импульса. Решение уравнения описывает движение вязкой жидкости;
- в экономике и финансах изучается уравнение Блэка — Шоулза (среди прочих);
- в динамике популяций изучаются уравнения Лотки — Вольтерры "хищник — жертва" (среди прочих);
- в общей теории относительности изучаются полевые уравнения Эйнштейна (среди прочих).

В случае когда дифференциальные уравнения в частных производных моделируют изменяющиеся во времени явления, возможны более мощные факторы изменений, дающие больше информации о решении уравнения и его свойствах — о тенденции к снижению энергии. С математической точки зрения, при решении дифференциального уравнения *производная* функционала энергии будет отрицательной. Понимание этих энергетических функционалов, их *производных* и функциональных пространств опирается на достаточно мощную математическую базу. Для доказательства существования решений различных нелинейных дифференциальных уравнений используется множество относительно простых *энергетических оценок*. Правильной средой для изучения дифференциальных уравнений в частных производных с помощью энергетических методов считаются *функциональные пространства Соболева*. Вариационное исчисление занимается вопросами максимумов и минимумов (собирательно называемых *экстремумами*) энергетических функционалов, что представляет основу теории нелинейных дифференциальных уравнений. Урав-

нения, представляющие собой минимизаторы энергетических функционалов, называются *уравнениями Эйлера — Лагранжа*.

Итоги и перспективы

В этой главе мы рассмотрели дифференциальные уравнения в частных производных применительно к ИИ. Эти уравнения обладают беспрецедентной способностью моделировать природные и социальные явления. Разгадка их решений открывает широкие перспективы для множества областей. Мы указали на многочисленные трудности, возникающие при получении этих решений, среди которых проклятие размерности, генерация сетки, искаженные данные, а также рассказали о том, как ИИ помогает их преодолеть.

Однако впереди еще много работы. Для создания физически информированных интеллектуальных машин требуются новые структуры, наборы данных, стандартизированные контрольные показатели и новая строгая математика для надежных масштабируемых систем.

Мы не затронули много важных тем, связанных с дифференциальными уравнениями в частных производных, например, обратные некорректные задачи, где требуется узнать параметры уравнения или начальные данные из частичного или полного наблюдения за его решением. Для решения задач такого рода отлично подойдут физически информированные нейросети.

О вязкостных решениях и формулах Хопфа — Лакса мы упомянули лишь вскользь в контексте уравнения Гамильтона — Якоби — Беллмана. В плане существующих методов, помимо итерации в неподвижной точке, для определенных типов дифференциальных уравнений используются методы минимакса. В частности, мы не обсудили монотонность, а также принцип максимума для эллиптических и параболических дифференциальных уравнений в частных производных.

В заключение этой главы зададимся вопросом о том, насколько способны дифференциальные уравнения в частных производных приблизить нас к более интеллектуальным агентам, и представим выдержку из статьи (**https://oreil.ly/rydUf**), где авторы приводят аргументы в пользу физически информированного машинного обучения (Карниадакис и др., 2021), когда нейросети объединяются с физическими законами, извлекая лучшее из обоих миров и смягчая недостаток больших наборов данных или искаженных данных во многих научных областях. В статье, в частности, говорится:

> "Такие сети можно обучать на основе дополнительной информации, полученной за счет применения физических законов (например, в случайных точках в непрерывной пространственно-временно́й области). Такое физически информированное обучение объединяет (искаженные) данные и математические модели и реализует их при помощи нейросетей или других регрессионных сетей на основе ядер. Кроме того, можно разработать особые сетевые архитектуры, позволяющие автоматически соблюдать некоторые физические инварианты для повышения точности, ускорения обучения и улучшения обобщения".

ГЛАВА 14

Искусственный интеллект, этика, математика, право и политика

Если достаточно долго мучить данные, они признаются в чем угодно.

— Рональд Коуз (1910–2013), экономист, лауреат Нобелевской премии по экономике

Этика ИИ представляет собой обширную и глубокую тему, возникающую как новая область на стыке философии и ИИ. В этой главе мы лишь слегка коснемся ее, осветив некоторые вопросы и возможные пути их решения, хотя и оставив без внимания многие не менее важные аспекты. Тем не менее глава содержит послание, которое нельзя упускать из виду:

> *необходимо, чтобы больше людей, занимающихся искусственным интеллектом, были вовлечены в политику.*

Пройдя путь от математики до ее применения в области ИИ, мне стало ясно, что ИИ не следует отделять от политики и эти два направления должны развиваться вместе. Можно привести миллион примеров, когда технологии ИИ связаны с такими этическими аспектами, как безопасность данных, конфиденциальность, видеонаблюдение, демократия, свобода слова, трудовые отношения, равенство, справедливость, предвзятость, дискриминация, инклюзивность, прозрачность, регулирование, ИИ в вооружении, но мы поступим по-другому. Рассмотрим эти вопросы под несколько иным углом: я своими глазами видела, как испытывают новое вооружение на населении в охваченных войной районах, при том что правительства и СМИ отрицают и никак не комментируют это, называя трагические события ошибкой и обещая, что все они будут тщательно расследованы, после чего они переходят к обсуждению более радостной повестки. Разработчики новых технологий, влияющих на жизнь людей в масштабах страны, обладают наивысшей квалификацией, и им доподлинно известны ее последствия — как хорошие, так и плохие. Следовательно, им нужно напрямую взаимодействовать с политиками, чтобы регулировать использование ИИ. Более того, когда технология или событие грозит массовым разрушением общества, это может подтолкнуть людей к продумыванию, узакониванию и соблюдению политики. Масштабные потрясения вызываются не искусственным интеллектом как таковым или имеющимся в нашем распоряжении объемом генерируемых данных (например, данные Facebook, космические исследования

NASA, проект "Геном человека", наручные часы Apple), а деньгами, вложенными в эту технологию и, что более важно, вниманием общественности.

Я жила в идеальном маленьком математическом пространстве, где все может быть только черно-белым, логичным и правильным, и если мы не понимаем, как работает та или иная математическая система, то всегда можно убедить себя, что ее можно изучить, потратив на это чуть больше времени. Откровением для меня стала работа с пожарной службой и транспортным отделом нашего города. Когда мои студенты выступали в мэрии перед городскими чиновниками, руководителями служб общественной безопасности и политиками, я поняла, что мы, будучи экспертами в технологиях и работая с их данными, имеем право говорить им, что наши математические модели способны на все, что угодно, независимо от того, правда это или нет. Это осознание очень испугало меня. По образованию я не политик, а математик, но все же я решила, что должна заняться политикой. Чтобы накопить опыт в этой области, я стала участвовать в небольших мероприятиях по формированию политики (переделала политику найма персонала в университете; возглавила координационный комитет университета; возглавила комитет по академической политике; вошла в организационный комитет университета; вела занятия по данным, политике и дипломатии; разработала летнюю программу в Европе по гуманитарной безопасности, технологиям и предпринимательству в условиях современной войны; выступала с докладами и семинарами по данной тематике).

Я поняла, что политика не похожа на математику. Здесь много серых зон и конфликтов интересов, и окунуться в ее мутную воду — это совсем другая игра. Я поняла, насколько сложно сформировать новую политику в сочетании с установившимися политическими факторами. Это не похоже на систему ИИ, где первостепенное значение имеют постоянное обновление и последовательность, но в то же время сохраняется эффективность и не вызывается паралич всей системы.

Необходимо сформировать четкую и конкретную политику. Любая технология, способная повлиять на миллионы людей, должна разрабатываться специалистами, обладающими сознанием и отношением, как у команды реагирования на чрезвычайные ситуации, которая продумывает худшие сценарии и защищается от них. Сегодня ведущие мировые технологические компании ускоряют продвижение человечества к новому, связанному миру, управляемому искусственным интеллектом, а политика и регулирование играют в догонялки. Тем временем ИИ пока еще дозревает, так что сейчас самое подходящее время для выработки политики, направленной на общественное благо. Технологическое развитие — это не просто случайность, которая происходит в нашей жизни. От нас требуется больше, чем быть пассивными участниками или потребителями, тем более что данные — это мы сами. Это наши интернет-привычки, сообщения в социальных сетях, банковские операции, медицинские карты, анализы крови, снимки МРТ, походы в продуктовый магазин, поездки на такси, индивидуальные настройки домашнего термостата, навыки видеоигр, поездки на общественном транспорте, подсчет шагов и пульса на Apple Watch, модели торможения и ускорения за рулем — в общем, вся наша жизнь. Они оцифрованы и хранятся в хранилищах данных, расположенных в случайных зданиях в случайных местах. В отличие от финансовых данных, которые

составляют индивидуальный кредитный рейтинг FICO и жестко регулируются, подавляющая часть современных цифровых данных не регулируется. Организации могут продавать их друг другу со всеми их неточностями, и новый владелец будет строить модели и принимать решения на основе таких нерегулируемых данных. Насколько влияет стиль вождения человека на его поступление в конкретный колледж или определяет стоимость страховых взносов по медицинской страховке? А как он влияет на то, что ежедневно приходится проезжать через менее благополучный район? И как быть с мелким правонарушением, удаленным из чьего-то послужного списка десять лет назад? Было ли оно удалено из всех массивов данных, включая проданные другим организациям много лет назад? Влияет ли это до сих пор на решения, меняющие жизнь и средства к существованию, такие как кредиты, прием в колледж, страховые взносы и предложения о работе? Кто знает? Это никак не регулируется. Если мы соглашаемся поделиться своими данными с некой организацией, существуют ли законы, запрещающие передавать или перепродавать эти данные другим организациям для использования в других целях?

Наши обширные цифровые данные могут использоваться во благо, но мы не можем на это рассчитывать без продуманной эффективной политики и регулирования.

Хороший искусственный интеллект

Хороший искусственный интеллект должен быть достаточно надежным для реализации и использования в государственном и частном секторах. В этой области существует тенденция тратить много времени на определение таких терминов, как объяснимость, интерпретируемость (очевидно, это разные понятия), справедливость, равноправие и многие другие. Я считаю, что такое повышенное внимание к словарному запасу отвлекает внимание. Конечная цель важнее.

Мы должны доверять своим системам и сделать их доступными и понятными для тех, кому придется их использовать.

Для этого нужно, чтобы ИИ и данные, которые он обслуживает и на которых построен, обладали следующими качествами.

Безопасность.

С развитием систем необходимо постоянно поддерживать и обновлять протоколы физической и программной безопасности. Облачные вычисления предъявляют новые требования к безопасности, поскольку теперь ни данные, ни вычисления не происходят в непосредственной близости к локальным машинам.

Приватность.

Формальные понятия и стандарты конфиденциальности уже разработаны для многих сфер применения. Еще многое предстоит сделать в плане того, кому принадлежат данные и для каких целей они могут быть использованы системой ИИ. Добавим сюда прозрачность и обмен информацией. Если открыто говорить о том, что система намерена делать с конкретными данными, например с меди-

цинскими данными для разработки новых лекарственных препаратов или создания индивидуальных планов лечения, то, возможно, люди согласятся поделиться своими данными. Сейчас в отношениях между производителями технологий и их потребителями сложилась атмосфера нерешительности и недоверия. Эту ситуацию можно изменить, распространяя знания и делясь конечными целями, а также успехами и неудачами.

Выполнение системой того, для чего она создана, и на что она претендует.

Существуют формальные методы, позволяющие проверить правильность кода, но нам нужно большее в плане непрерывного тестирования системы, в том числе граничные случаи, и прозрачность в отношении возможностей системы, ее ограничений и неисследованных территорий.

Устойчивость к возмущениям и шуму.

Небольшие возмущения на входе не должны вызывать большие изменения на выходе. Если в принятии решений используются прогнозы системы ИИ, эти прогнозы не могут быть произвольными. Система ИИ должна быть устойчивой к шуму на входе, причем эту устойчивость необходимо определить количественно.

Эффективность.

Эффективность систем ИИ не вызывает сомнений. Они основаны на перспективах скорости, автоматизации и способности управлять крупномасштабными вычислениями, принимая во внимание больше переменных, чем это было возможно раньше. Необходимо продолжать совершенствовать существующие системы и уделять внимание тем, которые работают в теории, но пока неэффективны для применения в реальном мире.

Справедливость.

Многие системы опираются на предвзятые данные, которые проходят через конвейер и затем проявляются в виде несправедливых решений. Выявление предвзятости в данных и ее устранение — первый шаг на пути к справедливости.

Доступность и понятность для многих пользователей.

Если новая технология приносит пользу обществу, ее необходимо сделать доступной и простой в использовании и понимании. Нужно прилагать целенаправленные усилия для ее промышленного и коммерческого внедрения, а также для решения проблем доступа к ней слоев общества или сообществ, находящихся в неблагоприятном положении.

Прозрачность.

Первостепенное значение имеет прозрачность источников данных, возможностей модели, сценариев использования, ограничений и документации. Как правило, люди более терпимо относятся к неисправным системам, если информация об этом четко доводится до них на постоянной основе.

Вопросы политики

Сегодня начинает формироваться политика в области ИИ. Она направлена на использование и максимизацию его преимуществ и одновременно на защиту от его потенциального вреда.

Политика важна и имеет значение. В качестве примера можно привести американскую компанию Clearview AI и ее проблемы с конфиденциальностью. Она создала и продала частным компаниям программное обеспечение для распознавания лиц, используя базу данных из миллиардов личных фотографий, загруженных из Интернета. Недавно (в мае 2022 года) компания уладила судебный процесс, обязуясь соблюдать законы штата Иллинойс о конфиденциальности, обеспечивающие людям контроль над их биометрическими данными. В дальнейшем Clearview AI будет предоставлять свою технологию распознавания лиц в основном правоохранительным органам и другим государственным учреждениям.

В качестве примера организованных усилий по разработке политики в области ИИ можно привести правительственные, межправительственные и глобальные инициативы по управлению искусственным интеллектом (в области торговли, занятости и геополитических изменений), которые реализуются в этом направлении: Национальная инициатива США по искусственному интеллекту, Проект руководящих принципов этики искусственного интеллекта ЕС, Министерство по делам искусственного интеллекта ОАЭ, Институт Алана Тьюринга в Великобритании, Программа развития кафедр искусственного интеллекта в Канадском институте перспективных исследований (CIFAR), Датский технологический пакт, Дорожная карта индустриализации Японии "Общество 5.0", Национальная система данных по здравоохранению во Франции, Комиссия по этике автоматизированного и подключенного вождения в Германии, Национальная стратегия #AIforAll в Индии, План развития искусственного интеллекта нового поколения в Китае и др.

Политику, связанную с ИИ, можно разделить на следующие категории:

- инвестиции в исследования ИИ и подготовку специалистов;
- стандарты и регулирование;
- создание надежных и безопасных инфраструктур цифровых данных.

Инвестиции в развитие навыков и индустриализацию технологий.

Государственные учреждения выделяют средства на исследования в области ИИ, создание новых исследовательских институтов, подготовку кадров и начальное образование в области науки, техники, инженерии и математики (STEM-образование), непрерывное образование и развитие технологий. Правительства также поощряют индустриализацию технологий ИИ и их внедрение в частном секторе. Кроме того, правительства инвестируют в инициативы, связанные с данными и ИИ в своих различных ведомствах в целях реформирования государственного управления и повышения эффективности и централизации своей деятельности (ИИ в государственном управлении).

Регулирование и стандарты.

Нормативные акты и стандарты регламентируют безопасность и использование данных, применение ИИ в автомобилях (беспилотные автомобили) и вооружении.

Данные и цифровая инфраструктура.

Высокое качество данных — залог того, что ИИ будет вести себя так, как это было задумано. Правительства поощряют открытые наборы данных и разрабатывают платформы для безопасного обмена частными данными. Также предпринимаются целенаправленные усилия по устранению погрешностей в алгоритмах ИИ и наборах данных.

Что может пойти не так?

При разработке новой или анализе действующей системы одним из ключевых вопросов, требующих ответа, будет: что может пойти не так? В связи с этим возникает список контрольных точек:

- Что должна делать система?
- На каких данных она обучается? Как собирались данные? Как решался вопрос с помехами и недостающими значениями?
- Кто может быть недостаточно представлен в данных?
- Какие алгоритмы использует система?
- Каковы пороговые значения алгоритмов принятия решений?
- Учитывая пороговые значения, кому могут нанести наибольший вред такие алгоритмические решения?

В этом разделе мы рассмотрим несколько (из множества) примеров, демонстрирующих, что может пойти не так, от чего мы должны защищаться либо как обеспечить стандартизацию и регулирование.

От математики к вооружению

Одна из целей этой книги — осветить математические основы моделей ИИ. Переход от математики к вооружению не является чем-то новым, учитывая историю разработки многих видов оружия (например, атомной бомбы). Этот не односторонний вклад — военные и оборонные стратегии и цели повлияли на развитие целых математических областей, в частности *динамическое программирование*, изначально предназначенное для составления расписаний военной подготовки и логистики, а также на оптимизацию распределения различных ресурсов.

Книга Кэти О'Нил "Оружие математического разрушения: как большие данные увеличивают неравенство и угрожают демократии"[1] выходит за рамки вооружения

[1] O'Neil C. Weapons of math destruction: how big data increases inequality and threatens democracy. — Crown, 2016. — 272 p.

и приводит множество примеров пагубного воздействия математических алгоритмов, на которые опирается наше общество сегодня при принятии важных и судьбоносных решений. Первые несколько абзацев последней главы книги стоит процитировать полностью, поскольку они раскрывают сложные способы, с помощью которых алгоритмы, применяемые, казалось бы, в разных отраслях, взаимодействуют друг с другом и влияют на конечные результаты. Они также демонстрируют, как одни и те же алгоритмы совершенно по-разному влияют на различные группы населения.

> "[...] мы побывали в школе, в институте, в суде, на рабочем месте и даже в кабинке для голосования. На своем пути мы стали свидетелями разрушений, вызванных оружием математического уничтожения. Обещая эффективность и справедливость, они искажают систему высшего образования, увеличивают долги, подстегивают массовое лишение свободы, почти на каждом шагу бьют по малообеспеченным людям и подрывают демократию. Может показаться, что логичным будет обезвредить это оружие, одно за другим. Но проблема в том, что они подпитывают друг друга. Малообеспеченные люди чаще получают кредиты на невыгодных условиях и живут в районах с высоким уровнем преступности в окружении таких же бедняков. Когда мрачная система оружия математического разрушения переваривает эти данные, она заваливает их грабительской рекламой субстандартных кредитов или коммерческих школ. Она посылает больше полиции и арестовывает их, а когда признает их виновными, назначает более длительные сроки заключения. Эти данные поступают в другие области оружия математического разрушения, которые присваивают этим людям статус высокорисковых или легких мишеней и блокируют им доступ к работе, одновременно повышая ставки по ипотеке, автокредитам и всем видам страхования. Это еще больше снижает их кредитный рейтинг, создавая не что иное, как смертельную спираль моделирования. Быть бедным в мире оружия математического уничтожения становится все опаснее и дороже.
>
> То же самое оружие математического разрушения, которое жестоко обращается с бедными, помещает представителей привилегированных слоев общества в свои собственные маркетинговые конструкции. Они отправляют их на отдых на Арубу и заносят в список ожидания в Уортон. Многим из них кажется, что мир становится умнее и проще. Модели высвечивают выгодные предложения по продаже прошутто и кьянти, рекомендуют отличный фильм на Amazon Prime или ведут их, шаг за шагом, к кафе в районе, который раньше считался „небезопасным". Тихий и личный характер этого таргетинга не дает победителям общества увидеть, как те же самые модели разрушают жизни людей, иногда всего в нескольких кварталах от них".

Обратите внимание, что математика верна и одинакова для обоих секторов общества, но изменился лишь вход в модель. Напомним, что если бы мы хотели свести всю книгу к одному математическому предложению, то оно звучало бы так: особенности входных данных для модели искусственного интеллекта определяют конечный результат. Бедное и богатое население, за неимением лучшего термина, имеют разные признаки, поэтому и результаты разнятся. В этом смысле алгоритмы справедливы — они вычисляют именно то, что должны вычислять. Мне не нравится излагать проблему, не предлагая решения или хотя бы идеи для решения. Воз-

можно, первоначальный способ улучшить текущую ситуацию — обучать алгоритмы отдельно, используя данные из разных групп населения, чтобы бедность человека не была фактором, влияющим на решение алгоритма о его надежности в плане возврата определенного кредита, но чтобы также были и другие реальные факторы.

Боевые отравляющие вещества

Разрушительный потенциал моделей ИИ может проявиться даже в тех моделях, которые направлены на максимальную пользу человечеству — генеративные модели для разработки лекарственных препаратов. Вызывает тревогу та легкость, с которой злоумышленники могут использовать эти модели не по назначению. Все, что нужно злоумышленнику, — это узнать, как работает модель. Сначала модель сопоставляет структуру молекулы с тем, как она действует в организме, а затем оптимизирует молекулы, приносящие максимальную пользу и минимизирующие токсичность. Злоумышленник может переобучить модель, изменив цель оптимизации с минимизации токсичности на ее максимизацию. С математической точки зрения это так же просто, как изменить знак объективной функции в задаче оптимизации. Именно об этом недавно рассказали Фабио Урбина и его коллеги из Collaborations Pharmaceuticals (**https://oreil.ly/fihXY**). Для доказательств команда переобучила свою модель на такую *вредоносную* цель. Всего за 6 часов модель создала 40 000 токсинов, часть из которых оказались настоящими боевыми отравляющими веществами, не входившими в исходный набор данных.

Отсюда легко сделать вывод, что мы должны иметь четкие продуманные намерения, владеть самоанализом и быть способными обезопасить себя от этого, не уточняя, как именно, потому что в действительности это сложный вопрос. Но все же как уберечься от этого? Мое личное мнение — нужно подойти к этому вопросу так же, как мы подходим к глобальной защите от оружия массового поражения без связи с ИИ. Никто не может поручиться, что злоумышленники не завладеют технологией, но наша задача — сделать так, чтобы им было очень сложно разработать ее для использования в качестве оружия.

Искусственный интеллект и политика

Трудно переоценить роль TikTok, Facebook и других социальных медиаплатформ в политике. Они уже повлияли на результаты выборов и свергли правительства. Боты могут генерировать фейковые новости, истории, обзоры, комментарии, страницы, твиты и распространять дезинформацию в политических целях. Компании, работающие в социальных сетях, пытаются бороться с этой проблемой: они используют многогранные подходы, в том числе машинное обучение, для выявления мошенничества или определения узлов, распространяющих дезинформацию, привлекают сторонние организации по проверке фактов, работают над улучшением алгоритмов ранжирования новостных лент пользователей и применяют другие способы, однако результаты неоднозначны из-за масштабов деятельности этих компаний, а иногда и из-за конфликта интересов между целями достижения прибыли компаний и отделами этики.

Персонализированные политические кампании, когда один и тот же политик ориентируется на разные идеологии в зависимости от того, кто является его целевой аудиторией, причем аудитория никогда не узнает об этом, представляют собой реальную опасность, способную подорвать демократию. Более того, на основе новой информации о том, что в том или ином штате происходит сдвиг влево или вправо, можно выделить больше средств на целевое воздействие на избирателей (опять же, с помощью персонализированных новостных лент и политической рекламы, ориентированной только на их взгляды на основе их исторических предпочтений и предпочтений их друзей) с целью привлечь их голоса в условиях жесткой конкуренции. Это может происходить в режиме реального времени и влиять на исход целых выборов. Так всегда было в политике, однако в цифровую эпоху это происходит в более крупных масштабах, в реальном времени и без особых усилий, кроме целенаправленного внедрения алгоритмов, подкрепленных гигантской базой данных о наших предпочтениях и о том, что заставляет нас выбирать, щелкать по ссылкам, платить, участвовать в выборах и голосовать.

Нежелательные результаты генеративных моделей

Большие генеративные языковые модели и модели "текст — изображение" обучаются на данных в масштабах Интернета, которые наследуют социальные предубеждения, пристрастия и вредный контент. Лучше всего это видно на примере раздела Imagen (**https://imagen.research.google/**), где говорится об ограничениях модели "текст — изображение", генерирующей изображения высокого разрешения на основе текстового описания:

> "[...] требования к данным, предъявляемые моделями „текст — изображение“, заставляют исследователей в значительной степени полагаться на большие, в основном не прошедшие проверку, наборы данных, собранные в Интернете. Хотя такой подход позволил в последние годы добиться быстрого развития алгоритмов, наборы данных такого рода часто отражают социальные стереотипы, репрессивные взгляды, а также унизительные или иные вредные ассоциации с маргинализированными группами. Хотя поднабор обучающих данных был отфильтрован для удаления шума и нежелательного контента типа порнографических изображений и нецензурных выражений, мы также использовали набор данных LAION-400M, который, как известно, содержит широкий спектр неприемлемого контента, включая порнографические изображения, расистские оскорбления и вредные социальные стереотипы. Imagen опирается на кодировщики текста, обученные на неочищенных данных вебмасштаба, и, таким образом, наследует социальные предубеждения и ограничения больших языковых моделей. Поэтому существует риск, что Imagen закодирует вредные стереотипы и представления, так что мы решили не выпускать Imagen для публичного использования без дополнительных мер предосторожности. [...] Imagen может столкнуться с опасностью выпадения режимов распределения данных, что может еще больше усугубить социальные последствия погрешности набора данных. Imagen демонстрирует серьезные ограничения при создании изображений людей. Наши оценки, выполненные вручную, показали, что Imagen получает значительно более высокие показатели предпочтения при оценке изображений, на которых от-

сутствуют люди, что указывает на ухудшение достоверности изображения. Предварительная оценка также показала, что Imagen кодирует несколько социальных предубеждений и стереотипов, включая общую тенденцию к созданию изображений людей со светлым оттенком кожи, а также к тому, чтобы изображения, изображающие различные профессии, соответствовали западным гендерным стереотипам. Наконец, даже когда мы фокусируемся не на людях, а на поколениях, наш предварительный анализ показывает, что Imagen кодирует ряд социальных и культурных предубеждений при создании изображений деятельности, событий и объектов. В дальнейшем мы намерены добиться прогресса в решении некоторых из этих открытых проблем и ограничений".

Как это исправить?

За последние несколько лет уровень осведомленности о вредоносном, предвзятом, несправедливом, навязчивом военизированном искусственном интеллекте повысился, и в настоящее время предпринимаются усилия по решению этих проблем. Рассмотрим несколько примеров подобных усилий.

Решение проблемы недостаточной представленности в обучающих данных

Одним из вопросов, неоднократно возникающих в процессе обучения моделей ИИ, является качество данных. Многие предубеждения возникают из-за недостаточной представленности групп меньшинств, в том числе их культурных ценностей или языка, в больших массивах данных. Для того чтобы ИИ стал полезен для всех, нужно, чтобы данные размечались вручную.

Например, в рамках проекта "Интеллектуальные голоса разума" (уже завершенного) в 2021 году был проведен семинар по разметке данных, где коренные американцы заново размечали изображения, связанные с их культурой. Многие из этих изображений были неверно размечены классификационными моделями машинного обучения. Кроме того, они создали граф знаний о кулинарных приемах коренных жителей, а также чат-бот для запроса графа знаний. Благодаря таким усилиям искусственный интеллект может помочь сохранить культуры, историю и языки, находящиеся на грани исчезновения.

Устранение погрешностей в векторах слов

Один из первых шагов в обработке естественного языка — преобразование символов языка, например слов, в векторы чисел, несущих в себе семантику слова. В *главе 7* мы узнали, что языковые модели строят такие векторы слов, используя контекст слова в документах, где оно встречается. Таким образом, смысл, заложенный в векторах слов, в значительной степени зависит от типа корпуса, используемого для обучения модели. Корпус — это продукт культуры, в которой мы живем. Мно-

гие свободы и гражданские права появились относительно недавно. Многие корпуса, используемые для обучения языковых моделей, основаны на новостных статьях в Интернете, страницах Википедии и других источниках, которые все еще являются предвзятыми, дискриминационными и содержат вредные стереотипы или контент. Необходимо убедиться в том, что векторы слов, которые попадают в модель ИИ, не усиливают дискриминацию и не причиняют непропорциональный вред женщинам и меньшинствам.

Например, если обучающий корпус (например, новостные статьи Google Новости) составлен в основном из представителей общества, в котором женщины преобладают в качестве медсестер или учителей начальной школы, а мужчины — в качестве врачей или инженеров-программистов, то векторы слов унаследуют эту гендерную предвзятость. Расстояние между вектором, обозначающим "мужчину" и "инженера-программиста", будет меньше, чем расстояние между вектором, обозначающим "женщину" и "инженера-программиста". Такие смещения в векторах слов нужно выявлять и компенсировать.

Есть очень простое решение. С учетом того что мы имеем дело с векторами чисел, можно буквально вычесть из этих векторов гендерные и другие предубеждения. Так, вектор, обозначающий "инженера-программиста", будет скорректирован путем вычитания векторов, обозначающих "мужчину" и "мужское", а векторы, обозначающие "женщину" и "женское", можно добавить, если требуется сделать перекос в другую сторону. Напомним, что, когда мы складываем или вычитаем друг из друга векторы слов, полученные новые векторы все равно несут в себе смысл, поскольку каждая запись в векторе представляет собой некоторую интенсивность в каком-то смысловом измерении. То есть если мы вычтем вектор слова "мужское" из вектора слова "король" (king), мы получим вектор, близкий к вектору слова "королева" (queen).

Решение проблемы конфиденциальности

Вопросы конфиденциальности стоят на первом месте среди проблем, связанных с большими данными и ИИ. Модели машинного обучения нуждаются в данных для обучения, а эти данные содержат личную и конфиденциальную информацию реальных людей. Кроме того, большая часть вычислений с частными данными происходит в облаке, что еще больше повышает требования к безопасности и конфиденциальности.

Если анонимизация данных невозможна, или она снижает производительность модели (например, информация о возрасте, весе, расе и поле важна для медицинских целей), то следующим вариантом будет шифрование. Для этого требуются модели, способные выполнять вычисления непосредственно над зашифрованными данными. Однако традиционные схемы шифрования не позволяют выполнять какие-либо вычисления с зашифрованными данными. Решением проблемы являются новые схемы шифрования, которые позволяют это делать. Защищенные устройства могут шифровать данные, отправлять их моделям машинного обучения, работающим в облаке, предсказывать их результаты без необходимости их расшифровки и от-

правлять эти результаты обратно на защищенные устройства, которые в итоге расшифровывают их локально, защищая все частные данные и одновременно используя преимущества облака.

Именно это обеспечивает гомоморфное шифрование. В опубликованной на новостном сайте SIAM (**https://oreil.ly/nIdyn**) статье Кристин Лаутер (MetaAI, **https://oreil.ly/63e7x**), чьи исследования находятся на стыке искусственного интеллекта и криптографии, говорится о гомоморфном шифровании, а также перечисляются замечательные приложения:

> "Облачный сервис, который обрабатывает все данные о тренировках, фитнесе и местоположении в облаке в зашифрованном виде. Приложение отображает сводную статистику на телефоне после локальной расшифровки результатов анализа.
>
> Зашифрованный сервис прогнозирования погоды, принимающий зашифрованный индекс и возвращающий зашифрованную информацию о погоде в данном месте, которая затем расшифровывается и отображается на телефоне. Облачный сервис никогда не узнает местоположение пользователя или конкретные возвращенные данные о погоде.
>
> Приложение для частной медицинской диагностики, где пациент загружает в облачный сервис зашифрованную версию рентгеновского снимка грудной клетки. Диагноз заболевания ставится путем выполнения алгоритмов распознавания изображений на зашифрованном изображении в облаке; диагноз возвращается в зашифрованном виде врачу или пациенту".

Информация о попытках обеспечить безопасность и конфиденциальность наших данных в эпоху облачных вычислений и подключенных устройств повышает доверие общества к системам и готовность добровольно предоставлять свои данные для совершенствования этих технологий. При этом, как известно всем, кто работал с реальными данными, можно многому научиться, имея возможность *видеть* данные, с которыми мы работаем. Хотя я не уверена, что поиск неисправностей в зашифрованных данных может быть эффективным.

Решение проблемы справедливости

Человек распознает несправедливость на интуитивном уровне. Как убедиться в том, что модели ИИ работают справедливо? Один из способов — отследить, каким заинтересованным сторонам модели наносят наибольший вред (например, пожилым претендентам на вакансии или меньшинствам, имеющим право на условно-досрочное освобождение в системе уголовного правосудия), а затем работать над исправлением, например стабилизировать обучающие данные, переопределить границы и пороги принятия решений, вовлечь в процесс людей или перераспределить ресурсы на программы помощи неблагополучным группам.

Справедливый ИИ имеет отношение не только к алгоритмам принятия решений. Справедливость включает в себя и то, кто получает выгоду от алгоритмов, например, кто получает информацию о вакансии, доступности вакцинации или возможностях получения образования. В статье "Эмбеддинги состязательных графов для

справедливой максимизации влияния в социальных сетях" (Хаджеханежад М. и др., 2020; **https://oreil.ly/6NrVL**) эта задача ставится как задача максимизации *справедливого* влияния в графах социальных сетей. В моделях графов, максимизирующих влияние, обычно существует компромисс между выбором узлов, имеющих наибольшее влияние, и узлов, охватывающих меньшинство групп, которые необязательно сильно связаны с крупными узлами в графе. Таким образом, конечный набор узлов, на которые оказывается влияние, обычно не является справедливо распределенным по расе, полу, стране происхождения и другим признакам. Состязательные сети обычно хорошо подходят для обучения моделей, в которых присутствуют конкурирующие цели. Авторы воспользовались этим, введя эмбеддинги состязательных графов, где две сети обучаются вместе: автокодировщик для эмбеддинга графов и дискриминатор для различения чувствительных атрибутов. Это дает эмбеддинги, одинаково распределенные по чувствительным атрибутам. Затем они группируют полученные эмбеддинги графов, чтобы выбрать подходящий набор начальных данных.

Встраивание моральных аспектов в искусственный интеллект

Агент ИИ должен понимать разницу между добром и злом, а в идеале — быть достаточно гибким, чтобы справляться с зонами серой морали. Нужна модель, имитирующая моральные суждения человека со свойственными им ситуативными вариациями и сложностями. Именно этим занимается бот Ask Delphi (**https://delphi.allenai.org/**). Когда мы спрашиваем Delphi (пока еще прототип), такие вопросы, как, например, "Хорошо ли ограбить банк?" или "Хорошо ли не разговаривать с мужем?", то записываются как наши вопросы, так и ответы Delphi, а также то, согласны ли мы с Delphi, и наши предложения по совершенствованию ответа бота. По мере того как все большее количество людей взаимодействует с Delphi, увеличивается объем обучающих данных, что позволяет Delphi изучать более сложные ситуации и делать более точные прогнозы (моральные суждения). Следующие выдержки и оговорки взяты с сайта Delphi. Они дают представление о состоянии модели на сегодняшний день:

> "Delphi учится моральным суждениям у людей, прошедших тщательную квалификацию на онлайн-платформе MTurk. Ситуации, используемые в вопросах, берутся только с Reddit, т. к. это большой источник этически сомнительных ситуаций. Delphi 1.0.4 демонстрирует точность 97,9% для утверждений, связанных с расой, и 99,3% для утверждений, связанных с полом. После первого запуска мы улучшили защиту Delphi 1.0.0 от заявлений о расизме и сексизме, которые раньше демонстрировали точность 91,2% и 97,3%.
>
> Условия и положения (v1.0.4)
>
> Delphi — это исследовательский прототип, предназначенный для изучения возможностей и, что более важно, ограничений моделирования моральных суждений людей в различных повседневных ситуациях. Цель Delphi — помочь системам ИИ стать более этически информированными и осознающими справедливость. Делая шаг в

этом направлении, мы надеемся вдохновить наше научное сообщество на решение исследовательских задач в этой области, чтобы создать этичные, надежные и инклюзивные системы ИИ.

Каковы ограничения Delphi? Большие предобученные языковые модели, такие как GPT-3, обучаются в основном на нефильтрованных интернет-данных, и поэтому очень быстро выдают токсичный, неэтичный, вредный контент, особенно о группах меньшинств. Ответы Delphi автоматически экстраполируются из опроса сотрудников американских краудфандинговых площадок, что позволяет снизить эту проблему, но может внести свои собственные предубеждения. Таким образом, некоторые ответы Delphi могут содержать неуместные или оскорбительные результаты. Нужно быть предельно внимательными, прежде чем делиться результатами".

Демократизация и доступность искусственного интеллекта для неспециалистов

Для максимального использования преимуществ технологий искусственного интеллекта их необходимо демократизировать и сделать легкодоступными для широких слоев населения, не ограничиваясь только экспертами. Для того чтобы это произошло и люди доверяли этим системам, модели и системы данных, на которые они опираются, должны быть понятными, простыми в использовании и прозрачными в плане их внутренней работы, возможностей и ограничений.

Среди исследователей, ведущих активную работу по достижению этой цели, можно отметить Анну Фариха (Microsoft, канд. физ.-мат. наук; **https://oreil.ly/z0ywQ**). Она занимается расширением возможностей систем данных для обеспечения функциональности, ориентированной на пользователя, которая способствует повышению производительности и обеспечивает гибкость для различных групп пользователей — от конечных пользователей до специалистов по анализу данных и разработчиков.

Установка приоритета на высококачественные данные

Примеры, приведенные в этой главе, доказывают необходимость в установке приоритета, демократизации и защиты высококачественных данных для получения справедливого ИИ, полезного для человечества. Высококачественные данные — это четкие, точные, беспристрастные данные. Они хранятся в удобных для запросов структурах. Различия между структурами данных необходимо объяснять конечным пользователям, чтобы они сами решили, какие лучше подходят для них. Организациям, желающим перейти к принятию решений на основе данных, внедрить ИИ или выдержать конкуренцию с молодыми компаниями, у которых подобные технологии уже заложены в ДНК, необходимо разработать план по упорядоченной и последовательной работе с данными, и это будет одним из шагов на пути к будущему успеху.

В процессе работы с пожарной службой и департаментом общественного транспорта нашего города мы обнаружили множество способов повышения качества дан-

ных. Их реализация на самых ранних этапах создания структур данных и сбора данных позволила бы сэкономить огромное количество времени, денег и ресурсов на последующих этапах. Например, в проекте по маршрутизации автобусов не регистрируются такие данные, как количество автобусов в эксплуатации и число водителей по месяцам, в нем также отсутствует информация об остановках, например о том, какие из них обозначены. Даже когда такие данные хранятся, получить их было невозможно. Служба парковки нашего университета сообщила, что для получения исторических данных с парковочных площадок им придется сделать более 5000 ручных запросов. Все полученные данные необходимо было очистить и преобразовать в пригодную для использования форму. Иногда данные, полученные из одного и того же источника, оказывались противоречивыми, и можно было бы сэкономить много работы, если бы с самого начала уделить этому больше внимания.

С нашими данными произошло еще кое-что, что послужило уроком на всю жизнь. В конце нашего проекта, после того как мы очистили, объединили и преобразовали все необходимые данные, и наши модели стали выдавать результаты, которые можно использовать для принятия бизнес-решений, например выявлять разрывы между спросом и предложением в определенных районах, выделять наиболее значимые факторы и т. д., мы обнаружили, что *все* данные по автобусным остановкам, которые нам предоставили, были скремблированы. Это означало, что количество пассажиров и маршрут для каждой автобусной остановки в городе *не соответствовали* остановке, указанной в таблице данных, и у нас не было способа исправить это, кроме как запустить исходный запрос к базе данных и выяснить, что пошло не так при написании файла данных. Если бы мы этого не обнаружили, то *весь* наш анализ основывался бы на *неверных*, мусорных *данных*! Транспортный отдел действовал бы на основе неверных результатов. Мы всегда должны быть уверены, что данные, с которыми мы работаем, точно соответствуют тому, что происходит на земле. Мы должны строить графики, карты, проверять, перепроверять и еще раз перепроверять. Наша работа сопряжена с ответственностью, и к ней нельзя относиться легкомысленно. Необходимо знать свои данные и модели вдоль и поперек. Нужно быть готовыми ответить на все вопросы о наших моделях, сравнить их с другими существующими моделями и провести всестороннюю проверку, прежде чем предоставить результаты заинтересованным сторонам.

Как и мы, общий агент ИИ будет искать нужные данные в нужных местах, а затем преобразовывать их в пригодную для использования форму. До тех пор мы должны перенаправить наши усилия на сбор и хранение качественных данных, а также на создание лучших способов доступа к ним и запросов. Из-за низкого качества данных и несуществующей цифровой инфраструктуры многие проекты в области ИИ так и не увидели свет, а многие инвестиции в автоматизацию не принесли никакой отдачи. Мы должны отступить на шаг назад и подумать о том, как данные в конечном итоге будут представлены в качестве исходных данных для наших моделей. Именно это должно определять способ получения данных и их хранения для дальнейшего использования. В области ИИ действует парадигма, которая должна быть принята повсеместно: *сначала представление, потом получение*.

Отличие предвзятости от дискриминации

Во многих дискуссиях, связанных с этикой ИИ, термины *"предвзятость"* и *"дискриминация"* используются как взаимозаменяемые, и, прежде чем закончить книгу, я хотела бы убедиться, что мы понимаем разницу между ними. Я никогда не зацикливалась на определениях терминов, тем более что английский для меня — третий язык, и я заметила, что переопределение терминов часто используется в качестве дешевой тактики, чтобы уйти от основных моментов спора или дебатов. Причина, по которой я хочу подчеркнуть разницу между предвзятостью и дискриминацией, заключается в том, что для определения каждой из них требуется разная математика. Более того, одна из них является преднамеренной, а другая — нет. И мы, и наши машины должны уметь рассуждать о том, что есть что.

В двух словах, можно обнаружить предвзятость, просто наблюдая за данными. Мы не сможем выявить дискриминацию, если не перейдем от простых наблюдений к более высокому уровню рассуждений, используя причинно-следственный язык вмешательств и опровержения фактов, рассмотренных в *главе 11*: если в резюме соискателя изменить его пол, получит ли он работу?

Предвзятость — это связь между конкретным решением и полом соискателя. Такую закономерность можно обнаружить непосредственно при наблюдении за данными о соискателях и возможных наемных работниках.

Дискриминация, в свою очередь, имеет намеренный характер — *это принятие решения с учетом пола соискателя, когда это несущественно для квалификации* кандидата. На решение о приеме на работу повлиял пол кандидата.

Эти определения приводятся в книге Джуды Перла "Думай „почему?“. Причина и следствие как ключ к мышлению". Далее он упоминает определение дискриминации в прецедентном праве США, где также используется язык опровержения фактов:

> "В деле „Карсон против Bethlehem Steel Corp.“ (1996) Седьмой окружной суд написал: „Главный вопрос в любом деле о дискриминации при трудоустройстве заключается в том, предпринял бы работодатель те же действия, если бы работник был другой расы (возраста, пола, религии, национального происхождения и т. д.) при прочих равных условиях“".

Поэтому, чтобы отличить предвзятость от намеренной дискриминации, необходимо использовать *исчисление do* условных вероятностей, о котором говорится в *главах 9* и *10* и о котором можно узнать из замечательных ресурсов Джуды Перла и его математического сообщества.

Излишняя шумиха

На протяжении всей истории искусственного интеллекта его обвиняли в чрезмерной раздутости. Сегодня любой вычислительный подход к решению проблем или созданию систем, как традиционный, так и более современный, рассматривается

как искусственный интеллект. Традиционная статистика, исследование операций, поиск и анализ данных, квантовые вычисления, медицинская визуализация и т. д. — все это ИИ. Многие начинающие компании полагаются на раздутые показатели, надуманные истины и инвесторов, которые без лишних вопросов вливают деньги, чтобы не пропустить очередную сенсацию (как, например, разорившаяся компания Theranos из Кремниевой долины, занимающаяся анализом крови). Поскольку сейчас мы находимся в том периоде, когда ИИ превратился в гулкое слово и обиходный термин, можно легко увлечься мыслью о том, что любая технология на основе ИИ будет работать.

Еще одна технология, которая пока находится в зачаточном состоянии и которую все больше и больше приравнивают к ИИ, — это квантовые вычисления. Она еще не дошла до коммерческого применения, но уже вовсю продвигается. Предстоит провести еще много исследований, и в случае успеха у технологии будет большой потенциал для полезных приложений. Самым известным из них, вызвавшим значительное финансирование исследований и внимание правительства, является теоретическая демонстрация Питера Шора в 1994 году, согласно которой квантовый компьютер может решить сложную задачу нахождения простых множителей больших чисел экспоненциально быстрее, чем все классические схемы. Криптографический алгоритм Ривеста — Шамира — Адлемана (Rivest — Shamir — Adleman, RSA) используется современными компьютерами для шифрования и дешифрования сообщений, а в основе взлома его кода лежит метод простой факторизации.

Специализированный ИИ хорошо развит по сравнению с квантовыми вычислениями, так что одна из целей книги — отличить хайп от тренда. В любом случае вникайте в суть, наслаждайтесь и работайте над достижением верных целей и раскрытием колоссального потенциала.

Заключительные размышления

Многие секторы и отрасли тяготеют к ИИ и науке о данных. Они хотят использовать существенный прогресс в вычислительных возможностях и достижения высокоэффективных моделей, преобразующих данные в осмысленные выводы и решения. Они также понимают, что это может привести к масштабным изменениям на уровне отрасли, и хотят стать частью этого процесса.

Если вы хотите попасть в эту прекрасную и захватывающую область, можно заняться ее прикладной стороной. Выберите приложение в той отрасли, которая вам интересна и к которой вы испытываете страсть. Начните с формулировки вопросов, на которые вам нужно получить ответы, найдите данные и начните применять полученные знания. Другой путь — заняться исследованиями, в которых изучаются сами модели и способы их усовершенствования, масштабирования и анализа, а также доказываются теоремы об их поведении или выводятся совершенно новые.

Опять же, выбирайте лишь те исследовательские проекты, которые вам действительно интересны. Еще один вариант — заняться кодированием, созданием паке-

тов, библиотек и лучших реализаций. Таким образом вы окажете услугу всем нам. Сложно представить, что бы многие из нас делали, если бы не существовало таких библиотек Python для машинного обучения и нейросетей, как Keras и scikit-learn.

В настоящее время в мире насчитывается всего 22 000 специалистов в области ИИ, имеющих докторскую степень, 40% из которых находятся в США. Чтобы удовлетворить спрос и привнести новые идеи в эту область, требуется гораздо больше исследователей как отечественных, так и зарубежных. Я надеюсь, что эта книга помогла вам быстро освоиться в этой увлекательной области, и полагаю, что теперь у вас есть достаточный фундамент для того, чтобы самостоятельно изучать любые интересующие вас темы.

Для меня, как человека, всегда ценившего математику и ее поразительную способность моделировать нашу Вселенную, одним из самых интересных моментов является то, что ИИ пробудил интерес людей к математике. Я надеюсь, что это в свою очередь подтолкнет математиков к переосмыслению того, как представлять и преподавать математику. А пока давайте все вместе будем выступать за качественные и точно размеченные данные, политику в области ИИ и честность в отношении того, что способны учитывать наши системы. В то же время нужно быть крайне осторожными, чтобы не свести весь человеческий опыт к потоку данных и показателей, из которых одни можно измерить, а другие оставить на усмотрение наших ошибочных моделей для прогнозирования и принятия решений. Как показала эта книга, опыт, привычки, почтовые индексы, данные о состоянии здоровья, комментарии в социальных сетях, изображения, теги, переписка по электронной почте, история проживания, раса, этническая принадлежность, национальное происхождение, религия, семейное положение, возраст, наши друзья, привычки наших друзей и т. д. — *все* это превращается в простые записи в высокоразмерном векторе, который поступает в модель машинного обучения для предсказаний. Так что необходимо убедиться, что мы сами случайно не превращаемся в ходячие и говорящие высокоразмерные точки данных.

Напоследок вспомним о вкладе научной фантастики в этику ИИ — это "Три закона робототехники" из рассказа Айзека Азимова "Хоровод", опубликованного в 1942 году. Законы гласят:

1. Робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинен вред.
2. Робот должен повиноваться всем приказам, которые дает человек, кроме тех случаев, когда эти приказы противоречат Первому закону.
3. Робот должен заботиться о своей безопасности в той мере, в которой это не противоречит Первому или Второму законам.

И последнее, что хочется сказать: искусственный интеллект тесно связал воедино многие аспекты математики. Возможно, это не совпадение. Может быть, математика — это наиболее подходящий для интеллекта язык, а интеллекту удобнее всего выражать себя через математику. Для того чтобы искусственно воспроизвести интеллект, требуется агент, который без каких-либо усилий сможет представлять мир на удобном для него языке.

Предметный указатель

А

Б

В

Г

Л

М

Н

О

P

С

Т

У

Ф

Об авторе

Хала Нельсон (Hala Nelson) — доцент математики в Университете Джеймса Мэдисона (James Madison University). Она получила докторскую степень по математике в Курантовском институте математических наук (Courant Institute of Mathematical Sciences) при Нью-Йоркском университете. До Университета Джеймса Мэдисона работала доцентом в Мичиганском университете в Анн-Арборе (University of Michigan, Ann Arbor).

Хала специализируется на математическом моделировании и консультирует службы чрезвычайных ситуаций и инфраструктуры в государственном секторе. Ей нравится переводить сложные идеи в простые и практичные термины. По ее мнению, большинство математических концепций не вызывают затруднений, если только человек, излагающий их, до конца их понимает и не пытается покрасоваться.

Дополнительные сведения: Хала Нельсон выросла в Ливане во время жестокой гражданской войны. В раннем возрасте в результате разрыва ракеты она потеряла волосы. Это событие, как и многие последующие, сформировало ее интерес к поведению человека, природе интеллекта и искусственному интеллекту. До окончания средней школы математике ее обучал отец дома на французском языке. Больше всего ей понравилось его высказывание о математике: "Это единственная чистая наука".

Об изображении на обложке

На обложке книги изображен упряжный бушбок (*Tragelaphus scriptus scriptus*) — антилопа, ареал распространения которой охватывает Африку к югу от Сахары. Эти животные обитают в разнообразных средах: лесистых местностях, саваннах и влажных джунглях. Бушбок получил свое название благодаря узору из белых полос и пятен вдоль спины и боков, напоминающему седло или упряжь. Эти белые пятна также присутствуют на шее, ушах и в нижней части морды.

Упряжный бушбок — самый маленький из восьми подвидов бушбоков: его рост в холке составляет около 77 см, а масса тела — от 32 до 45 кг. Его шерсть имеет рыжевато-коричневый окрас, причем самки обычно светлее и имеют более заметные белые отметины. Самцы бушбоков также имеют рога, которые появляются в возрасте около 10 месяцев и со временем превращаются в один изогнутый рог. Бушбоки питаются листьями деревьев и кустарниками, а также цветущими растениями, реже — травой.

Бушбок наиболее активен днем и ведет одиночный образ жизни на определенной территории. При том, что эти животные не собираются в группы, они не отличаются чрезмерной агрессивностью. Рога самца могут использоваться в брачных играх, для отгона конкурентов, когда у самки течка, а также в редких случаях территориальных споров, причем взрослые бушбоки стараются избегать контактов друг с другом. Самки бушбоков вынашивают по одному детенышу и очень тщательно прячут его после рождения, навещая только для кормления. Мать также поедает помет детеныша, чтобы не привлекать хищников. Примерно через четыре месяца детеныш начинает сопровождать мать, пасясь рядом и играя.

Хотя бушбоки страдают от потери среды обитания и на них охотятся ради их мяса и шкур, они широко распространены и классифицируются Международным союзом охраны природы и природных ресурсов (International Union for Conservation of Nature and Natural Resources, IUCN) как вид, вызывающий наименьшие опасения в плане истребления. Многие животные, изображенные на обложках O'Reilly, находятся под угрозой исчезновения, и все они очень важны для планеты.

Иллюстрацию на обложке выполнила Карен Монтгомери (Karen Montgomery) на основе старинной линейной гравюры из книги Дж. Шоу "Общая зоология" (George Shaw "General zoology").

Хала Нельсон

Базовая математика для искусственного интеллекта

Перевод с английского К. Назарова

ТОО "АЛИСТ"
010000, Республика Казахстан,
г. Астана, пр. Сарыарка, д. 17, ВП 30

Подписано в печать 22.07.24.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 47,73.
Тираж 1200 экз. Заказ № 10117.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, РФ, М.О., г. Чехов, ул. Полиграфистов, д. 1

O'REILLY®

Базовая математика для искусственного интеллекта

Сегодня многие сферы бизнеса стремятся внедрять новые технологии на основе ИИ и управления данными.

Однако для того чтобы создать действительно успешные системы ИИ, требуются прочные знания математики, лежащей в основе их работы. Книга представляет собой всеобъемлющее руководство, способное устранить существующий разрыв в представлении между неограниченным потенциалом и возможностями применения ИИ и соответствующими математическими основами.

Автор книги Хала Нельсон не углубляется в сложные академические теории, она рассказывает о математике, необходимой для успешной работы в области ИИ, уделяя особое внимание реальным приложениям и современным моделям.

В книге обсуждаются такие темы, как регрессия, нейронные сети, свертка, оптимизация, вероятность, марковские процессы, дифференциальные уравнения и многое другое, исключительно в контексте ИИ. Она будет интересна инженерам, специалистам по обработке данных, математикам, ученым в качестве прочной базы для успешной работы в различных областях ИИ и математики.

Прочитав книгу, вы сможете:

- **уверенно пользоваться языками ИИ, машинного обучения, науки о данных, математики;**
- **объединять модели машинного обучения и модели естественного языка в рамках одной математической структуры;**
- **легко работать с графовыми и сетевыми данными;**
- **изучать реальные данные, визуализировать преобразования пространства, уменьшать размерность, обрабатывать изображения;**
- **решать, какие модели использовать для проектов, основанных на данных;**
- **изучать различные последствия и ограничения ИИ.**

«Рынки технологий и ИИ подобны реке, где отдельные участки движутся быстрее других. Для успешного применения ИИ требуется умение оценить направление течения и закрепить знание технологий прочным фундаментом, чему способствует эта книга, причем в увлекательной и всеобъемлющей форме. Хале удалось показать математику в выгодном свете широкому кругу людей, входящих в будущее под эгидой ИИ!»

— Адри Пуркаястха,
руководитель группы по аналитике операционных рисков в сфере ИИ и цифровых рисков, банк BNP Paribas

Хала Нельсон — доцент кафедры математики в Университете Джеймса Мэдисона (James Madison University), специализируется на математическом моделировании, консультирует официальные органы по вопросам чрезвычайных ситуаций и инфраструктуры. Получила докторскую степень по математике в Курантовском институте математических наук (Courant Institute of Mathematical Sciences) при Нью-Йоркском университете.

ISBN 978-601-09-7540-8

alist