ДЖОН Д. КЕЛЛЕХЕР

ГЛУБОКОЕ ОБУЧЕНИЕ

Самый краткий и понятный курс

- Доступно для неспециалистов
- Вся базовая информаця в одной книге
- Искусственный интеллект, нейросети, машинное обучение и многое другое

БИБЛИОТЕКА MIT

БОМБОРА
ИЗДАТЕЛЬСТВО

БИБЛИОТЕКА
MIT

JOHN D. KELLEHER

DEEP LEARNING

ДЖОН Д. КЕЛЛЕХЕР

ГЛУБОКОЕ ОБУЧЕНИЕ

Самый краткий и понятный курс

- **Доступно для неспециалистов**
- **Вся базовая информаця в одной книге**
- **Искусственный интеллект, нейросети, машинное обучение и многое другое**

БОМБОРА
ИЗДАТЕЛЬСТВО
Москва 2022

УДК 004.8
ББК 32.813
К34

DEEP LEARNING
John D. Kelleher

Келлехер, Джон Д.

К34 Глубокое обучение. Самый краткий и понятный курс / Джон Д. Келлехер ; [перевод с английского М. А. Райтман]. — Москва : Эксмо, 2022. — 160 с. — (Библиотека MIT).

В этой книге простым и доступным для неспециалистов языком раскрываются такие сложные темы, как искусственный интеллект, нейросети, машинное обучение, глубокое обучение. Автор рассказывает о предпосылках глубокого обучения, его истории и базовых основах, а также проводит экскурс в будущее этой технологии, раскрывая перед читателями ее потенциал.

УДК 004.8
ББК 32.813

Научно-популярное издание

БИБЛИОТЕКА MIT

Келлехер Джон Д.

ГЛУБОКОЕ ОБУЧЕНИЕ

САМЫЙ КРАТКИЙ И ПОНЯТНЫЙ КУРС

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В Обручев*
Ответственный редактор *Е Истомина*
Литературный редактор *Н Калинина*
Младший редактор *Д. Данилова*
Художественный редактор *К Доброслов*
Компьютерная верстка *Э. Брегис*
Корректор *Л Макарова*

Страна происхождения Российская Федерация
Шығарылған елі Ресей Федерациясы

ISBN 978-5-04-116355-6

ООО «Издательство «Эксмо»
123308, Россия, город Москва, улица Зорге дом 1, строение 1, этаж 20, каб 2013
Тел 8 (495) 411 68-86
Home page www.eksmo.ru E-mail info@eksmo.ru
Өндіруші «ЭКСМО» АҚБ Баспасы
123308 Ресей қала Мәскеу, Зорге көшесі, 1 үй 1 ғимарат 20 қабат, офис 2013 ж
Тел 8 (495) 411-68-86
Home page www.eksmo.ru E-mail info@eksmo.ru
Тауар белгісі «Эксмо»
Интернет-магазин www.book24.ru
Интернет-магазин www.book24.kz
Интернет-дүкен www.book24.kz
Импортер в Республику Казахстан ТОО «РДЦ Алматы»
Қазақстан Республикасындағы импорттаушы «РДЦ Алматы» ЖШС
Дистрибьютор и представитель по приему претензий на продукцию в Республике Казахстан ТОО «РДЦ-Алматы»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының өкілі «РДЦ Алматы» ЖШС,
Алматы қ., Домбровский көш. 3-а» литер Б офис 1
Тел 8 (727) 251-59 90/91/92, E mail RDC Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген
Сертификация туралы ақпарат сайтта www.eksmo.ru/certification
Сведения о подтверждении соответствия издания согласно законодательству РФ о техническом регулировании можно получить на сайте Издательства «Эксмо» www.eksmo.ru/certification
Өндірген мемлекет Ресей Сертификация қарастырылмаған

Дата изготовления / Подписано в печать 17 05 2022
Формат 70x100$^{1}/_{16}$.
Печать офсетная Усл печ л 12,96
Тираж 2000 экз. Заказ 4106.

Отпечатано с готовых файлов заказчика
в АО «Первая Образцовая типография»,
филиал «УЛЬЯНОВСКИЙ ДОМ ПЕЧАТИ»
432980, Россия, г Ульяновск, ул Гончарова, 14

ISBN 978-5-04-116355-6

СОДЕРЖАНИЕ

Глубокое обучение открывает дорогу инновациям и изменениям во всех сферах современной жизни. Большинство прорывов в области искусственного интеллекта, о которых вы узнаете из новостей, основаны на глубоком обучении. Очевидно, что понимание этого предмета необходимо и предпринимателю, заинтересованному в повышении эффективности своей организации, и политику, обеспокоенному этикой и приватностью в мире больших данных, и исследователю, работающему со сложной информацией, и просто любопытному обывателю, который хочет иметь представление о потенциале искусственного интеллекта и его влиянии на жизнь обычных людей.

Цель этой книги — дать широкому кругу читателей возможность разобраться в том, что такое глубокое обучение, откуда оно взялось, как работает, на что способно (а на что нет) и какова вероятная траектория его развития в ближайшие десять лет. Глубокое обучение представляет собой набор алгоритмов и моделей, а это значит, что для его понимания необходимо ориентироваться в том, как эти алгоритмы и модели обрабатывают данные. Поэтому данная книга не носит чисто описательный характер, в ней, помимо прочего, объясняется работа алгоритмов. Я попытался представить этот технический материал в доступной форме. Как показывает мой опыт преподавания, технические темы лучше всего усваиваются путем пошагового объяснения основополагающих концепций. Я старался как можно меньше обращаться к математике, но там, где без математических уравнений было не обойтись, стремился преподнести их настолько ясно и просто, насколько мог. Свои объяснения я дополнил примерами и иллюстрациями.

Замечательное свойство глубокого обучения состоит не в сложности его математических концепций, а в том, как с помощью таких простых вычислений ему удается выполнять настолько разнообразный спектр захватывающих и впечатляющих задач. Не стоит удивляться этой простоте. На самом деле модель глубокого обучения — это всего лишь большое (очень большое) количество умножений и сложений с добавлением нелинейных отношений (которые я тоже объясню). Несмотря на это, такие модели способны, к примеру, побеждать чемпионов мира по игре в го, держать пальму первенства в компьютерном зрении и машинном переводе, водить автомобили и т. д. Эта книга служит только введением в глубокое обучение, но я надеюсь, что она достаточно глубокая и для того, чтобы позже, лучше овладев данным материалом, вы могли еще не раз к ней вернуться.

БЛАГОДАРНОСТИ

Появление этой книги стало возможным благодаря жертвам, на которые пошли моя жена Афра и другие члены моей семьи, включая моих родителей, Джона и Бетти Келлехер. Я также получил огромную поддержку от своих друзей, особенно от Алана МакДоннела, Ионелы Лунгу, Саймона Добника, Лоррейн Бирн, Ноела Фитцпатрика и Джозефа ван Генабита.

Я бы также хотел упомянуть о помощи, оказанной работниками издательства MIT Press и целым рядом людей, которые прочитали отдельные разделы этой книги и поделились своими впечатлениями. MIT Press пригласило трех анонимных рецензентов, которые прочитали и прокомментировали мой черновик; я благодарен им за потраченное время и их полезные отзывы. С черновыми вариантами отдельных глав ознакомилось немало людей, и я хотел бы воспользоваться этой возможностью, чтобы публично отметить их вклад; поэтому выражаю признательность Майку Диллинджеру, Магдалене Кацмайор, Элизабет Келлехер, Джону Бернарду Келлехеру, Афре Керр, Филипу Клубичке и Абхиджиту Махалункару. Содержание этой книги опирается на множество бесед о глубоком обучении, которые я имел со своими коллегами и студентами, особенно с Робертом Россом и Джанкарло Салтоном.

Я посвящаю эту книгу моей сестре Элизабет (Лиз) Келлехер в знак признания ее любви и поддержки, а также за ее терпение по отношению к брату, который без конца что-то объясняет.

ВВЕДЕНИЕ В ГЛУБОКОЕ ОБУЧЕНИЕ

Глубокое обучение — это подраздел искусственного интеллекта, посвященный моделированию крупных нейронных сетей, которые способны принимать верные *решения на основе данных*. Глубокое обучение особенно хорошо себя проявляет в задачах со сложными данными и огромными массивами информации. На сегодняшний день оно применяется в большинстве интернет-компаний и высококачественных потребительских технологий. Например, Facebook использует его, помимо прочего, для анализа текста в онлайн-переписках. Google, Baidu и Microsoft выполняют с его помощью поиск по изображениям и машинный перевод. Системы глубокого обучения присутствуют во всех современных смартфонах; например, они стали стандартной технологией для распознавания речи, а также для обнаружения лиц в цифровых камерах. В сфере здравоохранения они применяются для обработки медицинских изображений (рентгеновских снимков, результатов компьютерной магнитно-резонансной томографии) и для диагностики состояния здоровья. Глубокое обучение также является ключевым элементом беспилотных автомобилей, отвечая за локализацию и сопоставление, планирование движения, рулевое управление и восприятие окружающей обстановки, а также за отслеживание состояния водителя.

Возможно, самым известным примером глубокого обучения является AlphaGo* от DeepMind. Го — это настольная игра, похожая на шахматы. AlphaGo стала первой компьютерной программой, сумевшей победить профессионального игрока. В марте 2016 года она обыграла ведущего корейского профессионала Ли Седоля в матче, за которым следило больше двухсот миллионов человек. А в 2017 году она выиграла у китайца Кэ Цзе, находившегося на вершине мирового рейтинга.

---

* https://deepmind.com/research/alphago/. — *Здесь и далее — прим. авт.*

В 2016 году успех AlphaGo стал большим сюрпризом. На тот момент большинство людей считали, что компьютеры смогут на равных соперничать с игроками высшего уровня лишь спустя много лет. Уже давно было известно, что запрограммировать компьютер для игры в го намного сложнее, чем для игры в шахматы. В го намного больше потенциальных комбинаций. Это обусловлено тем, что го имеет бóльшую доску и более простые правила. На самом деле возможных комбинаций в го больше, чем атомов во вселенной. Такое громадное пространство поиска и высокая степень ветвления (количество состояний доски, которых можно достичь за один ход) делает го невероятно сложной игрой — как для людей, так и для компьютеров.

Чтобы проиллюстрировать разницу в сложности между го и шахматами с точки зрения компьютерных программ, можно сравнить историю поединков между людьми и компьютерами в этих двух играх. В 1967 году шахматная программа MacHack-6, разработанная в MIT, успешно соперничала с живыми игроками и имела рейтинг Эло* куда выше начального уровня. В мае 1997 года компьютер DeepBlue сумел обыграть чемпиона мира Гари Каспарова. Для сравнения: первая полноценная программа для игры в го была написана лишь в 1968 году, а в 1997-м сильные игроки по-прежнему легко обыгрывали компьютеры.

Такое отставание отражает разницу в вычислительной сложности между этими двумя играми. Тем не менее второй пример демонстрирует, насколько революционным оказалось влияние глубокого обучения на конкурентоспособность компьютеров при игре в го. Шахматным программам потребовалось 30 лет, чтобы достичь уровня чемпиона мира. Но благодаря глубокому обучению программы для игры в го всего за семь лет преодолели разрыв между опытным любителем и лучшим профессиональным игроком. Повышение эффективности оказалось совершенно экстраординарным, и оно свидетельствует также о прогрессе, который глубокое обучение принесло во многие отрасли.

AlphaGo использует глубокое обучение для оценки состояний доски и для выбора следующего хода. Этот факт помогает понять, почему глубокое обучение приносит значительную пользу во множестве разных предметных областей. Принятие решений играет ключевую роль в нашей жизни. Решения можно принимать «интуитивно», доверившись своему «чутью». Но, наверное, большинство людей согласятся с тем, что решения лучше всего основывать

---

* Рейтинговая система Эло — это метод определения уровня игроков в играх с нулевой суммой, таких как шахматы. Она названа в честь ее создателя, Арпада Эло.

на соответствующей информации. Глубокое обучение позволяет делать это за счет определения и извлечения закономерностей из огромных наборов данных, устанавливая правильные связи между цепочкой сложных входных значений и удачными решениями.

Искусственный интеллект, машинное обучение и глубокое обучение

Глубокое обучение стало результатом исследований в области машинного обучения и искусственного интеллекта. Отношения между этими тремя направлениями проиллюстрированы на рис. 1.1.

Искусственный интеллект как раздел компьютерных наук возник в ходе семинара в Дартмутском колледже летом 1956 года. Там были представлены исследования в разных областях, включая доказательство математических теорем, анализ естественного языка, планирование в играх, компьютерные программы, способные обучаться на примерах, и нейронные сети. Современная область машинного обучения основана на последних двух направлениях.

Рис. 1.1. Отношения между искусственным интеллектом, машинным обучением и глубоким обучением

Машинное обучение подразумевает разработку и оценивание алгоритмов, которые позволяют компьютеру извлекать функции из набора данных (то есть учиться на ряде примеров). Чтобы понять, как это работает, необходимо сначала разобраться с тремя терминами: набор данных, алгоритм и функция.

Глубокое обучение позволяет принимать решения на основе данных за счет определения и извлечения закономерностей из огромных наборов информации, устанавливая правильные связи между цепочкой сложных входных значений и удачными решениями.

В простейшем виде набор данных представляет собой таблицу, каждая строка которой содержит описание одного примера из соответствующей предметной области*, признаки которой разделены по столбцам. Например, в табл. 1.1 показан демонстрационный набор данных из сферы займов. В нем содержатся подробности о заявках от четырех потенциальных заемщиков. Если не считать поля ID, которое нужно лишь для удобства, каждый случай характеризуется тремя значениями: годовым доходом претендента, его текущей задолженностью и его кредитоспособностью.

Таблица 1.1. Набор данных о потенциальных заемщиках и их кредитоспособности

ID	Годовой доход	Текущая задолженность	Кредитоспособность
1	$150	–$100	100
2	$250	–$300	–50
3	$450	–$250	400
4	$200	–$350	–300

Алгоритм — это процесс (рецепт, программа), которому может следовать компьютер. В контексте машинного обучения алгоритм определяет процедуру анализа набора данных и выделяет повторяющиеся в этом наборе закономерности. Например, он может найти закономерность между годовым доходом и текущей задолженностью, с одной стороны, и кредитоспособностью — с другой. В математике подобного рода отношения называют функциями.

Функция — это детерминистическое отношение между входными значениями и одним или несколькими выходными. Детерминированность этого отношения означает, что любой конкретный ввод всегда генерирует один и тот же вывод. Например, сложение является детерминистическим отношением: 2+2 всегда равно 4. Как вы позже увидите, функции можно создавать не только для элементарной арифметики, но и для более сложных предметных областей. Например, мы можем определить функцию, которая принимает в качестве ввода доход и задолженность человека и выдает на выходе его кредитоспособность. Понятие функции играет очень важную роль в глубоком

* Под предметной областью мы имеем в виду проблему или задачу, которую пытаемся решить с помощью машинного обучения. Это может быть, например, фильтрация спама, предсказание цен на недвижимость или автоматическая классификация рентгеновских снимков.

Функция — это детерминистическое отношение между входными значениями и одним или несколькими выходными.

обучении, поэтому стоит еще раз повторить ее определение: функция — это просто отношение между вводом и выводом. По существу, целью машинного обучения является получение функций из данных. Функция может быть представлена множеством разных способов: как простая арифметическая операция (например, сложение и вычитание принимают ввод и возвращают одно выходное значение), последовательность правил *if-then-else* или нечто куда более сложное.

Функцию можно представить с помощью нейронной сети. Глубокое обучение является подразделом машинного обучения, посвященным глубокому моделированию нейронных сетей. Закономерности, которые алгоритмы глубокого обучения извлекают из наборов данных, являются функциями, представленными в виде нейронных сетей. Структура нейронной сети проиллюстрирована на рис. 1.2. Прямоугольники слева обозначают ячейки памяти, в которых находится ввод нейронной сети. Каждый кружок на этой диаграмме обозначает нейрон, а каждый нейрон реализует функцию: он принимает на вход ряд значений и связывает их со своим выводом. Стрелки показывают, как вывод одного нейрона подается на вход другому. В этой сети информация передается слева направо. Например, слева ей на вход можно подавать доход и задолженность, а показатель кредитоспособности будет возвращаться из крайнего правого нейрона.

Для выведения функций нейронная сеть использует стратегию «разделяй и властвуй»: каждый ее нейрон формирует простую функцию, которая в совокупности с результатами других нейронов формирует общую (более сложную) функцию. То, как нейронная сеть обрабатывает информацию, будет описано в главе 3.

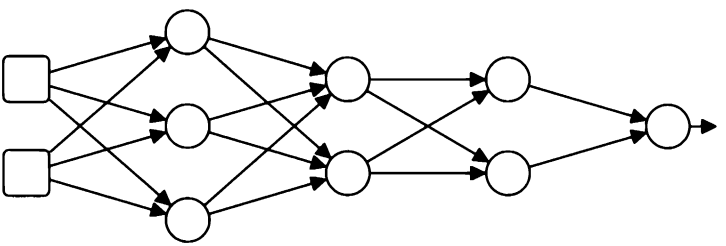


Рис. 1.2. Схематическое изображение нейронной сети

Что такое машинное обучение?

Алгоритм машинного обучения — это поисковый процесс, предназначенный для выбора из ряда потенциальных вариантов функции, которая лучше всего объясняет отношения между признаками набора данных. Чтобы получить

представление о том, как происходит извлечение функции из набора данных (то есть обучение), взгляните на следующий список входных и выходных значений для неизвестной функции. Определите на основе этих примеров, какая арифметическая операция (сложение, вычитание, умножение или деление) лучше всего объясняет отношение, которое устанавливается неизвестной функцией между вводом и выводом:

функция (ввод) = вывод
функция (5, 5) = 25
функция (2, 6) = 12
функция (4, 4) = 16
функция (2, 2) = 04

Большинство людей скажут, что лучше всего здесь подходит умножение, так как оно наиболее точно соответствует наблюдаемому отношению между вводом и выводом:

$5 \times 5 = 25$
$2 \times 6 = 12$
$4 \times 7 = 28$
$2 \times 2 = 04$

В этом конкретном случае выбор лучшей функции выглядит относительно просто, и человек может сделать его без помощи компьютера. Но с увеличением количества входных значений (иногда до сотен и тысяч) и разнообразия потенциальных функций, среди которых нужно выбирать, эта задача существенно усложняется. Именно в этом контексте мощь машинного обучения становится необходимой при поиске лучшей функции для сопоставления закономерностей в наборе данных.

Процесс машинного обучения состоит из двух этапов: собственно обучения и формирования логического вывода. На первом этапе алгоритм обрабатывает набор данных и выбирает функцию, которая лучше всего соответствует выявленным закономерностям. Затем эта функция кодируется в виде компьютерной программы (последовательности правил if-then-else или параметров определенного уравнения). Это так называемая модель, а анализ данных с целью извлечения функции часто называют обучением модели. В сущности, модель — это функция, закодированная в виде компьютерной программы. Однако в машинном обучении понятия модели и функции

настолько близки, что отличиями между ними зачастую пренебрегают и используют их как синонимы.

В глубоком обучении функции, извлекаемые из наборов данных в ходе их анализа, представлены в виде моделей нейронных сетей. Иными словами, модель нейронной сети кодирует функцию в виде компьютерной программы. Обычно в начале обучения нейронная сеть инициализируется с использованием случайных параметров (о параметрах нейронной сети мы поговорим позже; пока можете считать их значениями, которые влияют на поведение кодируемой функции). Такая сеть очень плохо сопоставляет различные входные значения и заданный ввод в предоставленном наборе данных. В процессе обучения она анализирует доступные примеры и в каждом случае сравнивает свой собственный вывод с результатом, указанным в наборе данных, корректируя свои параметры так, чтобы получить более близкое соответствие. Найдя функцию, которая выдает достаточно точное решение (с точки зрения того, насколько близок ее вывод к правильным результатам, указанным в наборе данных) для нашей проблемы, алгоритм машинного обучения завершает работу и возвращает итоговую модель. На этом этап обучения заканчивается.

С этого момента модель остается неизменной. Второй этап машинного обучения — формирование логических выводов. Это необходимо в том случае, когда модель применяется к новым примерам, правильные результаты для которых нам неизвестны, и мы хотим, чтобы она их для нас сгенерировала. Большая часть работы в машинном обучении заключается в формировании точной модели (то есть извлечении из данных подходящей функции). Это вызвано тем, что широкомасштабное применение обученной модели в промышленной среде для дальнейшего получения логических выводов из новых примеров требует навыков и методик, которыми большинство специалистов по анализу данных попросту не владеют. В компьютерной индустрии все шире распространяется понимание того, что для широкомасштабного применения систем искусственного интеллекта требуется особая квалификация, и это выражается в растущем интересе к области под названием DevOps. Данный термин выражает необходимость в сотрудничестве между разработчиками и командами эксплуатации/администрирования (последние отвечают за применение масштабирования готовых систем в промышленных условиях, а также за обеспечение их стабильной работы). Для описания обязанностей по применению обученных моделей также используют термины MLOps (machine learning operations — обслуживание машинного обучения) и AIOps (artificial intelligence operations — обслуживание искусственного интеллекта). Вопросы, относящиеся к процессу использования систем, выходят за рамки

этой книги, поэтому мы сосредоточимся на том, что такое глубокое обучение, для чего его применяют, как оно развивалось на протяжении своей истории и каким образом можно генерировать точные модели глубокого обучения.

Насущный вопрос в этом контексте звучит так: какую пользу приносит извлечение функций из данных? Дело в том, что извлеченную функцию можно применить к незнакомому набору данных, и значения, которые она при этом выдаст, помогут выработать правильные решения для этих новых задач (то есть ее можно использовать для получения логических выводов). Как вы помните, функция — это всего лишь детерминистическое отношение между вводом и выводом. Однако за простотой этого определения кроется разнообразие, присущее функциям. Рассмотрим следующие примеры.

- Фильтрация спама — это функция, которая принимает на вход электронное письмо и выдает значение, определяющее, является ли это письмо спамом.
- Распознавание лиц — это функция, которая принимает на вход изображение и помечает пиксели, относящиеся к лицу.
- Предсказание генов — это функция, которая принимает на вход последовательность ДНК в геноме и выделяет участки ДНК, которые являются кодирующими.
- Распознавание речи — это функция, которая принимает на вход аудиозапись и выдает текстовую транскрипцию того, что на ней слышно.
- Машинный перевод — это функция, которая принимает на вход предложение на одном языке и выдает его эквивалент на другом.

Машинное обучение стало настолько важным в последние годы именно благодаря тому, что огромное количество решений для множества задач в различных предметных областях можно представить в виде функций.

С чем связана сложность машинного обучения?

Существует ряд факторов, которые делают задачу машинного обучения сложной, даже несмотря на использование компьютеров. Во-первых, в большинстве наборов данных есть так называемый шум*, поэтому поиск функции,

* Понятие «информационный шум» означает поврежденные или неправильные данные. Он может быть вызван неисправными датчиками, ошибками при вводе данных и т. д.

которая в точности соответствует данным, не всегда оказывается лучшей стратегией, поскольку подразумевает обучение этому шуму. Во-вторых, часто бывает так, что количество возможных функций превышает количество примеров в наборе данных. Это означает, что задача, стоящая перед машинным обучением, сформулирована неправильно: предоставленной информации недостаточно для нахождения *единого* лучшего решения; вместо этого данным будут соответствовать несколько потенциальных решений. Чтобы это проиллюстрировать, рассмотрим такую задачу, как выбор арифметической операции (сложения, вычитания, умножения или деления): она лучше всего соответствует отношениям между вводом и выводом неизвестной функции. Вот пример отношений для этой задачи выбора функции:

функция (ввод) = вывод
функция (1,1) = 1
функция (2,1) = 2
функция (3,1) = 3

Умножение и деление подходят для этих примеров лучше, чем сложение и вычитание. Тем не менее на основе этого набора данных нельзя точно определить, какая именно операция играет роль неизвестной функции — умножение или деление, так как обе они соответствуют всем примерам. Следовательно, задача поставлена неправильно: ее условие не позволяет выбрать один лучший ответ.

Для решения таких задач можно попробовать собрать больше данных (больше примеров) в надежде на то, что это поможет отличить подходящую функцию от всех остальных альтернатив. Однако зачастую такую стратегию невозможно применить, поскольку дополнительные данные либо недоступны, либо требуют слишком больших усилий для сбора. Чтобы преодолеть этот недостаток, присущий задачам машинного обучения, алгоритмы пытаются дополнить имеющиеся данные рядом предположений о том, какими характеристиками должна обладать лучшая функция (или модель), и таким образом корректируют свой выбор. Эти предположения называются индуктивным сдвигом алгоритма, поскольку в логике процесс формирования общего правила на основе конкретных примеров известен как индуктивное умозаключение. Например, если все лебеди, которые вам когда-либо встречались, были белыми, вы можете сделать из этого логический вывод о том, что *все лебеди белые*. Эта концепция важна в данном контексте, потому что алгоритм машинного обучения формирует (или извлекает) общее правило (функцию) из набора конкретных примеров (набора данных). Следовательно, предположения, влияющие на этот алгоритм,

фактически корректируют процесс индуктивного умозаключения, и именно поэтому их называют индуктивным сдвигом алгоритма.

Итак, алгоритм машинного обучения использует для выбора лучшей функции два источника информации: набор данных и предположения (индуктивный сдвиг), вследствие которых он склонен отдавать предпочтение одним функциям перед другими, независимо от закономерностей, обнаруженных в наборе. Индуктивный сдвиг наделяет алгоритм машинного обучения собственным «взглядом» на предоставленные ему данные. Но, как и в реальном мире, не существует таких взглядов (сдвигов), которые были бы правильными во всех ситуациях (наборах данных). Вот почему алгоритмов машинного обучения так много: каждый из них кодирует свой собственный индуктивный сдвиг. Их допущения могут различаться по силе. Чем они сильнее, тем меньше свободы получает алгоритм при выборе функции, которая лучше всего объясняет выявленные закономерности. Набор данных и индуктивный сдвиг в каком-то смысле уравновешивают друг друга: алгоритмы с большим индуктивным сдвигом обращают меньше внимания на предоставленную информацию. Например, если алгоритм отдает предпочтение очень простым функциям, не имеет значения, насколько сложны закономерности в наборе данных, поскольку его индуктивный сдвиг очень большой.

В главе 2 мы объясним, как уравнение прямой можно использовать в качестве шаблона для определения функции. Это очень простое математическое уравнение. Алгоритмы машинного обучения, которые используют его в качестве шаблона для функций, соответствующих определенному набору данных, делают предположение о том, что генерируемая ими модель должна кодировать простое линейное отношение между вводом и выводом. Это предположение является примером индуктивного сдвига. На самом деле этот сдвиг довольно большой, так как несмотря на то, насколько сложные (или нелинейные) закономерности найдены в наборе данных, алгоритм попытается втиснуть в них линейную модель.

Алгоритмы с большим сдвигом могут столкнуться с одной из двух проблем. С одной стороны, если сдвиг слишком сильный, алгоритм проигнорирует важную информацию, и полученная функция не будет учитывать нюансы реальных закономерностей набора данных. Иными словами, функция окажется слишком сложной для этой предметной области, и вывод, который она генерирует, не будет точным. Такой исход называют недообучением. С другой стороны, если сдвиг слишком маленький (или щадящий), алгоритм получает излишнюю свободу в поиске функции, которая должна максимально соответствовать данным. В этом случае функция, скорее всего, получится слишком сложной для

данной предметной области и, что еще хуже, будет захватывать шум, закравшийся в учебный набор данных. Это ограничит способность функции делать обобщения о новых данных (которые не входили в учебный набор). Такой исход называют переобучением. Поиск алгоритма машинного обучения, способного находить подходящий баланс между данными и индуктивным сдвигом в конкретной предметной области, является ключом к подбору функции, которая не страдает от недообучения или переобучения и умеет делать удачные обобщения (сохраняя точность как при формировании логических выводов, так и при обработке новых примеров, не входивших в учебный набор данных).

Тем не менее в предметной области, сложность которой оправдывает использование машинного обучения, невозможно заранее определить правильные допущения для получения из данных подходящей модели. Следовательно, специалисты по анализу данных должны доверять своей интуиции (то есть прибегать к обоснованным догадкам) и подбирать оптимальные алгоритмы машинного обучения в заданной предметной области методом проб и ошибок.

Нейронные сети имеют относительно слабый индуктивный сдвиг. Вследствие этого в ходе глубокого обучения модели нейронных сетей чаще переобучаются, чем недообучаются. Основное внимание они уделяют данным, поэтому их лучше всего использовать для очень больших массивов информации. Чем больше набор данных, тем больше информации он предоставляет и тем логичней уделять ему повышенное внимание. Действительно, одним из важнейших факторов, способствовавших становлению глубокого обучения в последнее десятилетие, является возникновение концепции больших данных (англ. Big Data). Громадные массивы информации, ставшие доступными благодаря популяризации социальных платформ в Интернете, а также повсеместное внедрение датчиков, создали новые сценарии применения моделей нейронных сетей в ряде предметных областей. Составить представление о масштабах больших данных, которые используются в исследовании систем глубокого обучения, помогут такие факты: программное обеспечение для распознавания лиц от Facebook, DeepFace, обучалось на четырех миллионах изображений, принадлежащих более чем четырем тысячам людей (Taigman и др., 2014).

Ключевые элементы машинного обучения

В приведенном выше примере выбора арифметической операции, которая лучше всего объясняет отношения между вводом и выводом в наборе данных, наглядно показаны три ключевых элемента машинного обучения.

1. Данные (набор готовых примеров).
2. Набор функций, из которых алгоритм пытается выбрать наиболее подходящую для имеющихся данных.
3. Некий показатель пригодности, с помощью которого можно оценить, насколько хорошо каждый из вариантов подходит для имеющихся данных.

В успешном проекте машинного обучения корректными должны быть все три элемента; ниже мы опишем каждый из них более подробно.

Мы уже познакомились с понятием набора данных, представленного двумерной таблицей (или матрицей $n \times m$)*, в которой каждая строка содержит информацию об одном примере, а каждый столбец описывает один из признаков предметной области. Например, в табл. 1.2 проиллюстрировано, как в виде набора данных можно представить ввод и вывод первой задачи этой главы с поиском неизвестной арифметической функции. Этот набор состоит из четырех примеров (или образцов), каждый из которых представлен двумя входными признаками и одним выходным (целевым). Проектирование и подбор признаков для описания примеров является чрезвычайно важным этапом в любом проекте машинного обучения.

Как это часто бывает в компьютерных технологиях и машинном обучении, при подборе признаков приходится идти на компромисс. Если снабдить набор данных минимальным количеством признаков, можно потерять важную информацию, в результате чего функция, которую вернет алгоритм машинного обучения, будет плохо работать. С другой стороны, если собрать все признаки, имеющие место в предметной области, это может привести к включению бесполезной или лишней информации, что тоже, скорее всего, обернется получением плохо работающей функции. Одна из причин этого такова: чем больше мы добавляем лишних или бесполезных признаков, тем выше вероятность того, что закономерности, извлекаемые алгоритмом машинного обучения, будут основаны на ложных отношениях между этими признаками. В таких случаях алгоритм неспособен отличить настоящие закономерности в данных от ложных, которые возникают лишь из-за отдельной группы примеров, включенных в набор.

---

* В некоторых случаях требуются более сложные представления наборов данных. Например, временные ряды могут потребовать трехмерного представления, состоящего из последовательности двухмерных матриц, описывающих состояние системы в определенные моменты времени и размещенных в хронологическом порядке. Высокоразмерные матрицы называются тензорами.

Таблица 1.2. Простой табличный набор данных

Ввод 1	Ввод 2	Цель
5	5	25
2	6	12
4	4	16
2	2	04

Чтобы найти подходящее сочетание признаков, которые следует включить в набор данных, необходимо консультироваться со специалистами в соответствующей предметной области, использовать статистический анализ распределения отдельных признаков и отношений между их парами, а также использовать метод проб и ошибок в процессе создания моделей и проверки их производительности при добавлении или удалении конкретных признаков. Этот процесс проектирования набора данных требует больших усилий и зачастую занимает значительную часть времени и ресурсов, отведенных для проекта машинного обучения. Тем не менее без этого не обойтись, если вы хотите, чтобы ваш проект был успешным. Действительно, именно определение признаков, которые будут информативными для конкретной задачи, часто делает проект по-настоящему ценным и полезным.

Второй элемент в проекте машинного обучения — это группа функций-кандидатов, среди которых алгоритм ищет потенциальное объяснение закономерностей в данных. В рассмотренной ранее задаче с неизвестной арифметической операцией группа потенциальных вариантов была строго ограничена и состояла из четырех функций: *сложения, вычитания, умножения* и *деления*. В более общих случаях группа функций определяется посредством индуктивного сдвига в алгоритме машинного обучения и представления функции (модели), которое мы используем. Например, модель нейронной сети является очень гибким представлением функции.

Третий и последний элемент машинного обучения — это показатель пригодности. Речь идет о функции, которая принимает на вход результат работы функции-кандидата, сгенерированный в ходе ее применения к данным, и тем или иным способом сравнивает его с самими данными. Результатом этого сравнения будет значение, показывающее, насколько потенциальная функция подходит для имеющихся примеров. Чтобы оценить пригодность этой арифметической операции, можно посчитать, для скольких примеров функция-кандидат возвращает значение, которое в точности соответствует тому, которое указано в данных. Умножение даст четыре из четырех, сложение даст

один из четырех, а деление и вычитание дадут ноль. В машинном обучении применяется множество функций приспособленности, и выбор той, которая подходит для вашего проекта, будет играть ключевую роль в его успехе. Разработке новых функций приспособленности в машинном обучении посвящена обширная область исследований. Машинное обучение можно разделить на три категории в зависимости от того, как представлен набор данных и как определены функции-кандидаты и функция приспособленности: обучение с учителем, обучение без учителя и обучение с подкреплением.

Обучение с учителем, без учителя и с подкреплением

Обучение с учителем является самым распространенным видом машинного обучения. Для каждого образца в наборе данных указывается ожидаемое выходное (или целевое) значение. Например, если мы используем набор данных из табл. 1.1 для подбора функции, которая сопоставляет годовой доход и задолженность с кредитоспособностью, последняя будет целевым признаком, который предоставляется вместе с данными. Для применения машинного обучения с учителем в наборе данных должен содержаться целевой признак для каждого образца. Иногда сбор этих целевых признаков может быть довольно сложным и затратным. В некоторых случаях для назначения каждому образцу подходящих целевых значений приходится нанимать специалистов. Однако наличие этой информации помогает в процессе обучения. Алгоритм сравнивает результат работы функции с целевым выводом, указанным в наборе данных, и использует полученную разницу (или погрешность) для оценки приспособленности функции-кандидата, и это помогает в поиске наиболее подходящей функции. Именно эта обратная связь между метками в наборе данных и алгоритмом служит «учителем» при этом типе машинного обучения. И именно этот тип использовался в нашем примере с выбором арифметических операций для объяснения поведения неизвестной функции.

Машинное обучение без учителя обычно применяется для кластеризации данных. Такой тип анализа информации подходит, к примеру, для сегментации клиентов, когда компания хочет разделить своих клиентов на конкретные группы, чтобы создавать для них разные рекламные кампании или продукцию с разным дизайном. В машинном обучении без учителя набор данных не содержит целевых значений. Следовательно, алгоритм не может оценить приспособленность потенциальной функции на основе одних только данных. Вместо этого он пытается определить функции, которые группируют похожие

образцы в кластеры таким образом, чтобы у образцов из одного кластера было больше общего, чем у образцов из разных кластеров. Следует отметить, что изначально определение кластеров либо отсутствует, либо является очень обобщенным. Например, специалист по анализу данных, руководствуясь интуицией, может указать алгоритму, сколько всего должно быть кластеров, не уточняя при этом, каким должен быть их размер и какого рода образцы должны находиться в каждом из них. В машинном обучении без учителя алгоритм часто выполняет начальную кластеризацию наугад и затем постепенно корректирует кластеры (перемещая образцы из одного в другой), улучшая их приспособленность. Функция приспособленности, которая используется в этом процессе, обычно отдает предпочтение кандидатам, которые достигают высокой степени сходства внутри отдельных кластеров и, как следствие, значительных различий между самими кластерам.

Обучение с подкреплением лучше всего подходит для задач в режиме реального времени, таких как управление роботами или участие в играх. В этих сценариях агент должен сформировать набор правил, как следует себя вести в заданном окружении, чтобы получить подкрепление. Задача агента — сопоставить текущие наблюдения об окружающей среде и свое собственное внутреннее состояние (свою память) с действием, которое он должен выполнить. Например, *куда должен двигаться робот, вперед или назад, или что должна сделать компьютерная программа: переместить пешку или взять ферзя противника*. Выводом из этого правила (функции) будет следующее действие, которое агент должен выполнить в соответствии с текущей ситуацией. В такого рода сценариях сложно создать набор готовых данных, поэтому обучение с подкреплением зачастую происходит *в реальном времени*: агент помещается в среду, где он экспериментирует с различными правилами (которые вначале могут быть произвольными), постепенно обновляя их в ответ на подкрепление, которое эта среда ему дает. Если подкрепление положительное, в правиле закрепляется связь актуальных наблюдений и состояния с выполненным действием; если же подкрепление отрицательное, эта связь ослабляется. В отличие от машинного обучения с учителем и без учителя этот процесс выполняется «на лету», поэтому этапы формирования отношений и логических выводов постоянно чередуются. Агент делает вывод о том, какое следующее действие он должен выполнить, и использует обратную связь со своим окружением, чтобы понять, как обновить свои правила. Отличительной чертой обучения с закреплением является то, что целевой вывод сформированной функции (действия агента) отделен от механизма подкрепления. Оно может зависеть от нескольких действий, поэтому непосредственно после отдельно взятого действия подкрепления может вообще не быть. Например,

в игре в шахматы подкрепление может быть +1, если агент выигрывает партию, и –1, если проигрывает. Однако эта обратная связь становится доступной только после заключительного хода игры. Поэтому один из вызовов обучения с подкреплением заключается в проектировании таких обучающих механизмов, которые способны правильно распределять подкрепление по цепочке совершенных действий, чтобы правило можно было соответствующим образом обновлять. Компания DeepMind Technologies, принадлежащая Google, породила большой интерес к этой области, продемонстрировав, как обучение с подкреплением можно применять в моделях глубокого обучения для выработки правил использования элементов управления в семи разных видеоиграх для Atari. На вход система принимала значения пикселей, взятые прямо с экрана, а правила управления определяли, какие кнопки джойстика агент должен использовать в каждый момент игры. Компьютерные игры очень хорошо подходят для обучения с закреплением, так как для формирования успешных правил агент может предпринять тысячи и тысячи попыток, и при этом не требуется создания и маркировки огромных наборов данных с разными ситуациями и нажатиями джойстика. Система DeepMind научилась играть в эти игры настолько хорошо, что в шести из них ей удалось превзойти все остальные компьютерные системы, а в трех ее результаты были лучше, чем у специалистов.

Глубокое обучение можно применять ко всем трем видам машинного обучения: с учителем, без учителя и с закреплением. Хотя первый используется чаще всего, поэтому глубокое обучение в этой книге рассматривается в основном в его контексте. Тем не менее большинство проблем и принципов глубокого обучения, представленных здесь, применимы также к обучению с учителем и с закреплением.

В чем кроется успех глубокого обучения?

В любом процессе, основанном на данных, главным фактором успеха является понимание того, что и как следует измерять. Вот почему в машинном обучении так важен процесс подбора и проектирования признаков. Как уже упоминалось выше, для этого могут потребоваться знания в предметной области, статистический анализ данных и ряд экспериментов по формированию моделей с разными сочетаниями признаков. Следовательно, проектирование и подготовка набора данных может занять существенную долю времени и ресурсов, отведенных на проект; в некоторых случаях на это может уходить до 80% от общего бюджета (Келлехер и Тирни, 2018). Проектирование признаков — это одна

из тех областей, в которых глубокое обучение может иметь существенное преимущество перед машинным. В традиционном машинном обучении это проектирование часто требует значительных усилий со стороны человека. Глубокое обучение подходит к этому с другой стороны, пытаясь автоматически извлечь из данных признаки, которые будут наиболее полезными для текущей задачи.

Давайте рассмотрим один из примеров проектирования признаков. Индекс массы человеческого тела (англ. body mass index, или BMI) — это отношение веса человека (в килограммах) к его росту (в метрах, возведенных в квадрат). В сфере здравоохранения BMI используется для определения людей с недостаточной, нормальной, избыточной или чрезмерно избыточной массой тела (ожирением). Такая категоризация помогает предсказать вероятность возникновения у человека заболеваний, связанных с весом, таких как диабет. BMI здесь применяется потому, что с его помощью врачи могут делить пациентов на категории, имеющие отношение к соответствующим недугам. Обычно чем человек выше, тем он тяжелее. Однако на большинство заболеваний, связанных с массой тела (таких как диабет), влияет не рост, а то, насколько пациент тяжелее других людей аналогичной комплекции. BMI является полезным показателем, который можно использовать для классификации людей по весу, поскольку учитывает зависимость массы тела от роста. Это пример признака, полученного (вычисленного) из элементарных свойств — в данном случае веса и роста. BMI также иллюстрирует то, как производные признаки, основанные на элементарных, могут оказаться полезнее для принятия решений. Идею BMI еще в восемнадцатом веке предложил Адольф Кетле.

Как уже упоминалось ранее, в ходе работы над проектом машинного обучения много времени и усилий уходит на определение или проектирование (производных) признаков, которые помогают решить соответствующую задачу. Преимущество глубокого обучения заключается в том, что оно способно автоматически извлечь полезные производные признаки прямо из данных (о том, как это делается, мы поговорим в следующих главах). Действительно, при наличии достаточно крупных наборов данных глубокое обучение демонстрирует такую высокую эффективность в извлечении признаков, что его модели теперь превосходят по точности модели многих других методов машинного обучения, в которых признаки определяются вручную. Вот почему глубокое обучение так эффективно в тех предметных областях, где образцы описываются с помощью очень большого количества признаков. Такие наборы данных формально называют высокоразмерными. Например, высокоразмерной можно назвать коллекцию фотографий, если у каждого пикселя на фото есть свой признак. В таких сложных предметных областях очень трудно подбирать признаки вручную:

представьте, каких усилий это потребовало бы в случае с распознаванием лиц или машинным переводом. Поэтому в таких непростых условиях вполне логично прибегать к автоматическому извлечению признаков из больших наборов данных. У глубокого обучения есть еще одна похожая особенность: оно позволяет создавать сложные нелинейные связи между вводом и выводом; концепция нелинейных связей будет рассмотрена в главе 3, а в главе 6 мы покажем, как эти связи извлекаются из данных.

Подведение итогов, и что вас ждет впереди

В этой главе основное внимание уделялось месту глубокого обучения в контексте более широкой области машинного обучения. Поэтому именно машинному обучению посвящена большая часть материала. В частности, вы познакомились с понятием функции как детерминистического отношения между вводом и выводом и узнали, что задача машинного обучения состоит в поиске функции, которая сопоставляет входные признаки с выходными, которые можно наблюдать в образцах набора данных.

В этом контексте глубокое обучение было представлено в качестве подраздела машинного обучения, задача которого состоит в проектировании и оценивании учебных алгоритмов и архитектур моделей для современных нейронных сетей. Одной из особенностей глубокого обучения является его подход к формированию признаков. В большинстве проектов машинного обучения это в основном делается вручную, что может потребовать глубокого понимания предметной области и большого количества времени/расходов. Для сравнения: модели глубокого обучения способны формировать полезные признаки на основе низкоуровневых элементарных данных и устанавливать сложные нелинейные связи между вводом и выводом. Эта способность зависит от наличия крупных наборов данных; но если такие наборы имеются, глубокое обучение часто превосходит другие методы машинного обучения. К тому же во многих случаях это позволяет генерировать высокоточные модели для сложных предметных областей, будь то машинный перевод, распознавание речи или обработка изображений и видео. Глубокое обучение в каком-то смысле раскрыло настоящий потенциал больших данных. Ярче всего это проявляется в сфере потребительской электроники. Однако тот факт, что глубокое обучение может использоваться для анализа громадных массивов информации, имеет последствия и для неприкосновенности частной жизни и гражданских свобод (Келлехер и Тирни, 2018). Вот почему

так важно понимать, что это такое, как оно работает и для чего его можно/нельзя применять. Впереди вас ожидает следующее:

- В главе 2 вы познакомитесь с некоторыми основополагающими вопросами глубокого обучения: что такое модель, как установить ее параметры на основе данных и каким образом из сочетания простых моделей можно создавать более сложные.
- В главе 3 мы объясним, что такое нейронные сети, как они работают и что мы понимаем под глубокой нейронной сетью.
- В главе 4 будет представлена история развития глубокого обучения. Мы сосредоточим внимание на основных концептуальных и технологических прорывах, которые поспособствовали прогрессу в области машинного обучения. В частности, вы узнаете, чем объясняется взрыв популярности глубокого обучения в последние годы.
- В главе 5 описывается текущее положение дел в этой области. Мы представим две наиболее популярные архитектуры глубокого обучения на сегодняшний день: сверточные и рекуррентные нейронные сети. Первые идеально подходят для обработки изображений и видео, а вторые незаменимы при работе с последовательными данными, такими как речь, текст или временные ряды. Понимание различий и сходства этих двух архитектур поможет вам подстраивать глубокие нейронные сети под характеристики определенных типов данных и позволит по достоинству оценить разнообразие подходов к проектированию сетей.
- В главе 6 объясняется, почему модели глубоких нейронных сетей обучаются с помощью алгоритмов градиентного спуска и обратного распространения. Понимание этих двух алгоритмов позволит вам как следует разобраться в текущем состоянии сферы искусственного интеллекта. Например, это поможет понять, почему при наличии достаточного количества данных вы можете обучить компьютер выполнять конкретную задачу в рамках строго определенной предметной области на уровне, недоступном человеку, и почему исследователи все еще испытывают трудности, работая с более универсальными формами интеллекта.
- В главе 7 мы попробуем заглянуть в будущее глубокого обучения. Мы проанализируем основные тенденции в этой сфере и то, какой, по нашему мнению, будет их роль в ближайшие годы. В этой главе мы также обсудим некоторые трудности, с которыми сталкивается данная область — в частности, понимание и интерпретация принципов работы глубоких нейронных сетей.

КОНЦЕПТУАЛЬНЫЕ ОСНОВЫ

В этой главе вы познакомитесь с некоторыми фундаментальными концепциями, лежащими в основе глубокого обучения. Мы попытаемся отделить описание основ этих концепций от формальной терминологии, применяемой в глубоком обучении, которая будет рассмотрена в последующих главах.

Сеть глубокого обучения — это математическая модель, имеющая (отдаленное) сходство со структурой головного мозга. Чтобы разобраться в этой области на концептуальном уровне, полезно иметь наглядное представление о том, что такое математическая модель, как можно устанавливать ее параметры, как эти модели можно сочетать (компоновать) и каким образом с помощью геометрии можно проиллюстрировать механизм обработки информации в модели.

Что такое математическая модель?

В простейшем виде математическая модель — это уравнение, которое описывает, как одна или несколько входных переменных относятся к выходной. В этом смысле она представляет собой то же самое, что и функция, — связь между вводом и выводом.

При любом рассмотрении моделей необходимо помнить утверждение Джорджа Бокса: *все модели ошибочные, но некоторые из них полезные!* Полезная модель должна иметь какое-то отношение к реальному миру. Самым наглядным примером такого отношения является смысл, вкладываемый в переменную. Возьмите такое значение, как 78 000; само по себе оно бессмысленно, так как у него нет никакой связи с реальными концепциями, но, если записать *годовой доход=$78 000*, мы будем знать, что это число описывает определенный аспект реальности. Наделив переменную модель каким-то смыслом, мы можем представить ее как описание процесса, в ходе которого разные

аспекты реального мира взаимодействуют между собой и порождают новые явления. Затем эти события описываются выводом модели.

В качестве элементарного примера модели можно привести уравнение прямой:

$$y = mx + c$$

В этом уравнении y является выходной переменной, x — входной, а m и c — это два параметра модели, с помощью которых можно регулировать отношение между вводом и выводом, определяемое моделью.

Представим гипотезу: годовой доход влияет на то, насколько человек счастлив. Давайте опишем отношение между этими двумя переменными*. Используя уравнение прямой, можно определить модель, которая описывает это отношение следующим образом:

$$\text{счастье} = m \times \text{доход} + c$$

У этой модели есть смысл, так как ее переменные (не путать с параметрами) связаны с концепциями реального мира. Чтобы сделать ее полной, необходимо задать значения ее параметров: m и c. На рис. 2.1 проиллюстрировано, как изменение этих значений влияет на отношение между *доходом* и *счастьем*, определенное моделью.

Один из важных аспектов этой диаграммы состоит в том, что, независимо от значений, присвоенных параметрам, отношение между входной и выходной переменными, определяемое моделью, можно изобразить в виде прямой линии. Это неудивительно, поскольку при определении модели за основу было взято уравнение прямой. И именно поэтому подобные модели называют линейными. Еще одним интересным свойством этой диаграммы является то, как изменение параметров модели влияет на отношение между доходом и счастьем.

Сплошная линия, уходящая вверх под большим углом, имеет параметры ($c = 1$, $m = 0{,}08$). Это модель мира, в котором люди с нулевым доходом имеют счастье на уровне 1. Увеличение дохода делает людей существенно счастливей.

---

* Оказывается, зависимость между годовым доходом и счастьем до какой-то поры остается линейной. Но по достижении определенного уровня деньги перестают делать людей счастливей. Как показало исследование Кахнемана и Дитона (2010), в Америке общий порог, после которого доход уже не улучшает эмоциональное состояние, составляет около $75 000.

Штриховая линия с параметрами (c = 1, m = 0,06) представляет модель, в которой люди с нулевым доходом имеют счастье уровня 1, и увеличение дохода тоже делает людей счастливей, но не настолько, как в случае с первой моделью. Наконец, пунктирная линия с параметрами (c = 4, m = 0,02) описывает модель мира, в котором никого нельзя назвать особенно несчастным: даже люди с нулевым доходом имеют уровень счастья 4 из 10. И хотя дополнительный доход влияет на счастье, это влияние довольно умеренное. В третьей модели предполагается, что доход относительно слабо связан со счастьем

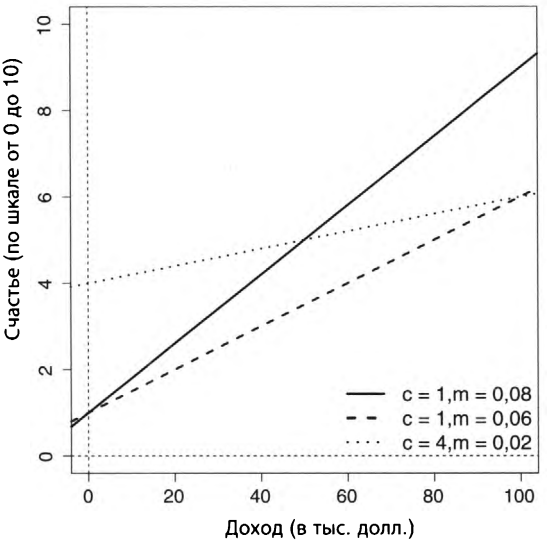


Рис. 2.1. Три разные линейные модели, описывающие влияние дохода на счастье

В целом отличия между тремя прямыми, изображенными на рис. 2.1, показывают, как изменение параметров влияет на линейную модель. Изменение *c* позволяет двигать прямую вверх и вниз. Это особенно заметно, если сосредоточиться только на оси *y*: вы можете видеть, что прямая, определенная моделью, всегда пересекает (отсекает) вертикальную ось в месте, значение которого равно *c*. Вот почему параметр c линейной модели называют отсечением. Отсечение можно представить в виде величины выходной переменной при нулевом вводе. Изменение параметра *m* влияет на угол (наклон) прямой. Этим определяется, насколько сильно изменения дохода сказываются на счастье. Можно считать, что значение наклона— это мера того, насколько важен доход для счастья. Если доход очень важен (то есть если даже небольшие его изменения существенно влияют на счастье), параметр наклона нашей модели будет иметь большое значение. Можно также сказать, что параметр

наклона линейной модели описывает важность, или вес, входной переменной при определении величины вывода.

Линейные модели с несколькими входными переменными

Уравнение прямой можно брать за основу математических моделей, у которых больше одной входной переменной. Например, представьте такую ситуацию: вы получили должность кредитного инспектора в финансовом учреждении, и один из аспектов вашей работы состоит в определении того, одобрять или отклонять то или иное заявление на получение кредита. Пообщавшись со специалистами в этой предметной области, вы выработали гипотезу, что при моделировании кредитоспособности человека следует учитывать как его годовой доход, так и его текущую задолженность. Если предположить, что между этими двумя переменными и кредитоспособностью человека существует линейная зависимость, соответствующую модель можно записать так:

$$\text{кредитоспособность} = (\text{доход} \times \text{вес дохода}) + (\text{задолженность} \times \text{вес задолженности}) + \text{отсечение}$$

Обратите внимание, что вместо параметра m в этой модели используются отдельные веса для каждой входной переменной; каждый вес определяет, насколько важен соответствующий ввод для определения вывода. Если использовать математическую запись, эта модель будет выглядеть так:

$$y = (\text{input}_1 \times \text{weight}_1) + (\text{input}_2 \times \text{weight}_2) + c,$$

где y представляет итоговую *кредитоспособность*, input_1 — это переменная *дохода*, input_2 — это переменная *задолженности*, а c — отсечение. Использование отдельного веса для каждого ввода модели позволяет как угодно расширять количество переменных в уравнении прямой. Все модели, определенные таким образом, остаются линейными в рамках тех измерений, которые определяются количеством входных значений и выводом. Это означает, что линейная модель с двумя вводами и одним выводом определяет не прямую, а, скорее, плоскость — именно так выглядит двухмерная прямая, расширенная до трехмерного пространства.

Запись математической модели с множеством входных значений может быть довольно хлопотным занятием, поэтому математики любят записывать

все максимально лаконично. Например, уравнение, приведенное выше, может иметь следующую краткую форму:

$$y = \sum_{i=1}^{n} (input_i \times weight_i) + c$$

Эта запись говорит о том, что для вычисления выходной переменной *y* нужно сначала пройтись по всем *n* значениям, поданным на вход, умножить каждое из них на соответствующий вес и затем сложить результаты этих умножений с добавлением отсечения в конце. Символ $\sum$ обозначает сложение результатов умножения, а *i* говорит о том, что мы умножаем каждый ввод на вес с тем же индексом. Эту запись можно сделать еще более компактной, если считать отсечение одним из весов. Для этого можно ввести переменную $input_0$, которая всегда равна 1, и сделать отсечение ее весом — то есть $weight_0$. Это позволит записать нашу модель следующим образом:

$$y = \sum_{i=0}^{n} (input_i \times weight_i)$$

Обратите внимание, что индекс начинается с 0, а не с 1, так как теперь мы используем дополнительный ввод, $input_0 = 1$, а отсечение помечено как $weight_0$.

Линейные модели можно записывать разными способами, но их основная особенность в том, что вывод в них вычисляется как *сумма n входных значений, умноженных на соответствующие им веса*. Следовательно, это семейство моделей определяет операцию под названием *взвешенная сумма*, поскольку мы взвешиваем каждый ввод и суммируем результаты. Хотя вычислить взвешенную сумму довольно легко, но во многих ситуациях она оказывается крайне полезной; это основная операция, которая применяется в любом нейроне любой нейронной сети.

Задание параметров линейной модели

Вернемся к нашему рабочему примеру с созданием модели для вычисления кредитоспособности людей, подавших заявления на получение банковского кредита. Для простоты изложения проигнорируем параметр отсечения, так как в принципе он не отличается от других параметров (весов для входных значений). Итак, избавившись от отсечения, мы получим следующую линейную модель (или взвешенную сумму) того, как доход и задолженность человека относятся к его кредитоспособности:

Умножение входных значений на веса с последующим сложением называется *взвешенной суммой*.

$$\text{кредитоспособность} = (\text{доход} \times \text{вес дохода}) + (\text{задолженность} \times \text{вес задолженности})$$

Чтобы сделать эту модель полной, необходимо задать ее параметры, то есть нужно указать значение веса для каждого ввода. Подобрать соответствующие значения могут помочь специалисты в предметной области.

Например, если предположить, что повышение дохода сильнее влияет на кредитоспособность, чем аналогичное увеличение задолженности, первый вес должен быть больше второго. Это предположение воплощено в следующей модели; в частности, здесь указано, что доход в три раза важнее задолженности при определении кредитоспособности человека:

$$\text{кредитоспособность} = (\text{доход} \times 3) + (\text{задолженность} \times 1)$$

Недостаток этого подхода в том, что мнения специалистов в предметной области расходятся. Например, вам может показаться, что наделение дохода весом, втрое большим, чем вес задолженности, является нереалистичным. В этом случае вы можете, например, отрегулировать модель с использованием одинаковых весов для обоих входящих значений, предполагая, что доход и задолженность в равной степени влияют на определение кредитоспособности. Чтобы избежать разногласий между специалистами, параметры можно устанавливать на основе данных. И в этом помогает машинное обучение. Нейронная сеть берет набор данных и выдает параметры (или веса) модели.

Извлечение параметров модели из данных

Далее в этой книге мы опишем стандартный алгоритм для извлечения весов линейной модели, который называется градиентным спуском. Но, чтобы вы понимали, о чем идет речь, давайте сделаем небольшой обзор. Начнем с набора данных, состоящего из образцов, для которых у нас есть как ввод (доход и задолженность), так и вывод (кредитоспособность). Такой набор для нашего примера* проиллюстрирован в табл. 2.1.

---

* Это тот же набор данных, который приводится в табл. 1.1 в главе 1; он приводится еще раз для удобства.

Нейронная сеть берет набор данных и выдает параметры (или веса) модели.

Процесс извлечения весов начинается с угадывания их начальных значений. Велика вероятность того, что эти догадки сделают нашу модель крайне неточной. Но это нормально, поскольку мы будем использовать набор данных для постепенного обновления весов, делая модель все лучше и лучше с точки зрения того, насколько хорошо она соответствует имеющейся информации. В этом примере в качестве начальной модели будет использоваться та, которую мы описывали выше:

$$\text{кредитоспособность} = (\text{доход} \times 3) + (\text{задолженность} \times 1)$$

Таблица 2.1. Набор данных с заявлениями о выдаче кредита и известными рейтингами кредитоспособности заявителей

ID	Годовой доход	Текущая задолженность	Кредитоспособность
1	$150	−$100	100
2	$250	−$300	−50
3	$450	−$250	400
4	$200	−$350	−300

Процесс улучшения весов модели в целом заключается в выборе образца из набора данных и введении его значений в модель. Это позволяет вычислить примерный вывод для этого образца. Дальше можно определить погрешность модели, отняв примерный вывод от правильного, который указан для образца в наборе данных. С помощью этой погрешности можно улучшить приспособленность модели к данным. Для этого нужно обновить ее веса, используя следующую стратегию (или правило обучения).

- Если погрешность равна 0, веса модели менять не нужно.
- Если погрешность положительная, вывод модели для этого образца был недостаточно большим, поэтому мы должны его увеличить за счет увеличения весов для всех входных переменнных с положительными значениями и уменьшения весов для всех входных переменных с отрицательными значениями.
- Если погрешность отрицательная, вывод модели для этого образца был слишком большим, поэтому мы должны его уменьшить за счет уменьшения весов для всех входных переменнных с положительными значениями и увеличения весов для всех входных переменных с отрицательными значениями.

Чтобы проиллюстрировать процесс обновления весов, возьмем первый образец из табл. 2.1 (доход = 150, задолженность = –100, а кредитоспособность = 100). Проверим, насколько точной окажется наша начальная модель, и обновим веса в соответствии с итоговой погрешностью.

$$\text{кредитоспособность} = (\text{доход} \times 3) + (\text{задолженность} \times 1) = (150 \times 3) + (-100 \times 1) = 350$$

Когда входные значения для нашего образца вводятся в модель, примерное значение кредитоспособности получается равным 350. Это больше, чем соответствующий показатель, указанный для этого образца в наборе данных (100). В итоге погрешность модели отрицательная (100–350 = –250), поэтому, исходя из правила обучения, описанного выше, вывод модели нужно уменьшить за счет уменьшения весов для положительных переменных и увеличения — для отрицательных. В данном случае величина дохода положительная, а задолженности — отрицательная. Если уменьшить вес дохода на 1 и увеличить вес задолженности на 1, получится следующая модель:

$$\text{кредитоспособность} = (\text{доход} \times 2) + (\text{задолженность} \times 2)$$

Чтобы понять, улучшило ли это изменение весов нашу модель, можно проверить, является ли генерируемый ею примерный вывод более точным, чем у старой модели. Это проиллюстрировано ниже с использованием того же образца:

$$\text{кредитоспособность} = (\text{доход} \times 2) + (\text{задолженность} \times 2) = (150 \times 2) + (-100 \times 2) = 100$$

На этот раз примерная кредитоспособность, сгенерированная моделью, совпадает со значением, указанным в наборе данных. Это говорит о том, что обновленная модель лучше приспособлена к имеющейся информации по сравнению с оригинальной. Более того, она генерирует правильный вывод для всех образцов в наборе данных.

В этом примере достаточно было лишь одного обновления, чтобы извлечь комбинацию весов, которая делает поведение модели согласованным со всеми образцами в наборе данных. Но обычно, чтобы получить хорошую модель, приходится много раз анализировать разные образцы и корректировать веса. Кроме того, чтобы упростить этот пример, мы исходили из того, что для обновления весов их нужно увеличивать или уменьшать на 1. На самом же

деле вычисление величины изменения каждого веса в машинном обучении выглядит сложнее. Но если не принимать во внимания эти нюансы, в основе глубокого обучения лежит описанный здесь процесс обновления весов (параметров) модели для улучшения ее приспособленности к набору данных.

Сочетание моделей

Итак, мы уже знаем, как построить линейную модель для вычисления приблизительной кредитоспособности заявителя и изменить ее параметры, чтобы она лучше соответствовала набору данных. Однако работа кредитного инспектора не ограничивается одним лишь вычислением кредитоспособности: он должен также решить, следует ли выдавать заявителю кредит. Иными словами, нам требуется правило, которое принимает на вход рейтинг кредитоспособности и выдает решение по заявлению на получение кредита. Например, можно использовать такое правило: *кредит получает человек с кредитоспособностью выше 200*. Это тоже модель: она связывает входную переменную, в данном случае *кредитоспособность*, с выходной — *решением по кредиту*.

Таким способом мы можем принять решение по заявлению. Сначала мы используем модель кредитоспособности, чтобы преобразовать профиль заявителя (состоящий из годового дохода и задолженности) в рейтинг кредитоспособности. Затем мы пропускаем этот рейтинг через модель принятия решений, чтобы одобрить или отклонить заявление. Этот процесс можно кратко записать в псевдоматематическом виде:

```
решение по кредиту = правило о принятии решений
                     (кредитоспособность = (доход × 2) + (задолженность × 2))
```

Если использовать такую запись, весь процесс принятия решения по заявлению о выдаче кредита для первого образца из табл. 2.1 выглядит следующим образом:

```
решение по кредиту = правило о принятии решений
                     (кредитоспособность = (доход × 2) + (задолженность × 2)) =
                     правило о принятии решений (кредитоспособность =
                     (150 × 2) + (–100 × 2)) =  правило о принятии решений
                     (кредитоспособность = 100) = отклонить
```

Теперь мы получили возможность использовать модель (состоящую из двух более простых моделей — правила о принятии решений и взвешенной суммы) для описания того, как нужно принимать решение по кредиту. Более того, если использовать данные предыдущих заявителей для задания параметров (то есть весов), наша модель будет зависеть от того, как мы обработали предыдущие заявления. Это полезно, поскольку мы можем использовать данную модель для рассмотрения новых заявлений в соответствии с тем, как выносились предыдущие решения. Когда кто-то подает новое заявление, мы просто используем нашу модель для его обработки и принятия решения. Именно возможность применять модель к новым образцам делает математическое моделирование настолько успешным.

Используя вывод одной модели в качестве ввода другой, мы создаем третью, комбинированную, модель. Такой подход к построению сложных моделей из более простых лежит в основе сетей глубокого обучения. Как вы увидите, нейронные сети состоят из большого количества мелких элементов под названием *нейроны*. Каждый из этих нейронов представляет собой отдельную простую модель, которая соотносит набор входных значений с выходным. Общая модель, реализуемая сетью, формируется путем подачи вывода одной группы нейронов в качестве ввода другой, вывод которой затем подается на вход третьей и т. д., пока не будет сгенерирован итоговый вывод модели. Основная идея состоит в том, что подключение вывода одних нейронов к входу следующих позволяет последним учиться решать другую часть общей задачи, используя промежуточные решения, выданные предыдущими нейронами. Это похоже на то, как правило о принятии решений генерирует окончательный ответ по заявлению о выдаче кредита, отталкиваясь от вычислений, произведенных моделью кредитоспособности. Мы еще вернемся к теме компоновки моделей в следующих главах.

Пространства ввода, пространства весов и пространства активации

Математические модели можно записывать в виде уравнений, но часто бывает полезно взглянуть на них с точки зрения геометрии. Например, графики, показанные на рис. 2.1, помогли нам понять, как изменения параметров линейной модели влияют на отношения между ее переменными. Существует целый ряд геометрических пространств, которые следует различать и понимать в контексте нейронных сетей. Это пространство ввода, пространство весов

и пространство активации нейрона. Для иллюстрации этих трех видов пространств можно воспользоваться моделью принятия решений по заявлениям о выдаче кредита из предыдущего раздела.

Начнем с описания концепции пространства ввода. У нашей модели принятия решений по кредитам есть два ввода: годовой доход и текущая задолженность заявителя. Эти входные значения для наших четырех образцов перечислены в табл. 2.1. Пространство ввода этой модели можно изобразить в виде графика, если относиться к каждой входной переменной как к оси в системе координат. Пространством ввода оно называется, поскольку каждая его точка определяет возможное сочетание входных значений модели. Например, на графике в левой верхней части рис. 2.2 показаны позиции всех четырех заявлений о выдаче кредита в рамках пространства ввода модели.

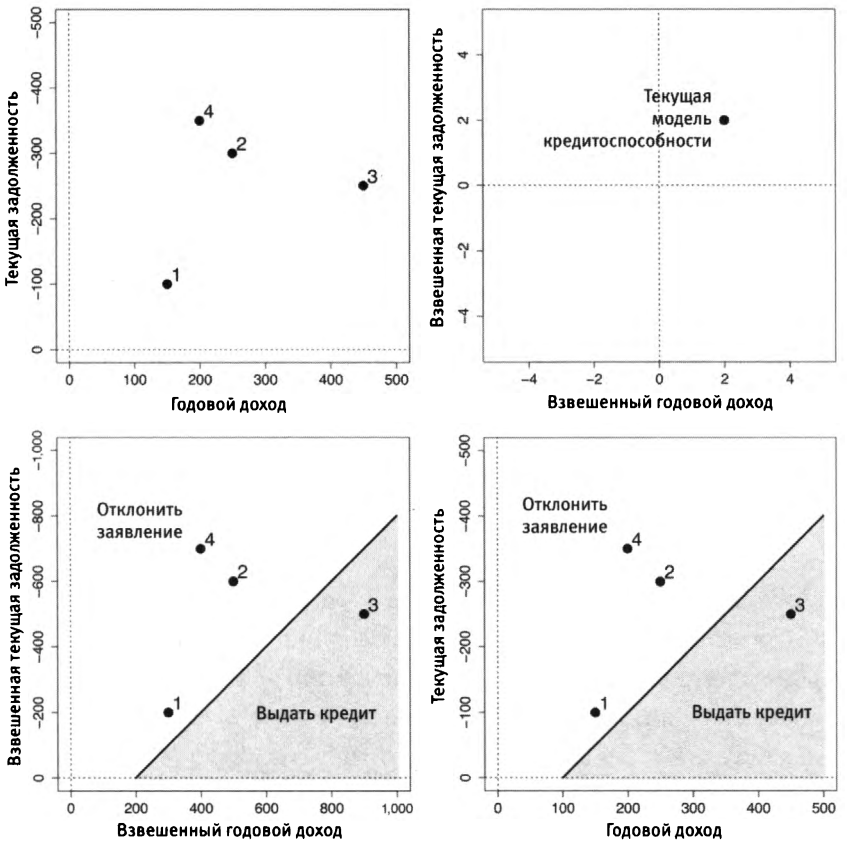


Рис. 2.2. Существуют четыре разных пространства координат, относящихся к обработке модели принятия решений по кредиту: слева вверху показано пространство ввода; справа вверху — пространство весов; слева внизу видно пространство активации (или решений); справа внизу — пространство ввода в сочетании с границей принятия решений

Пространство весов отражает множество возможных сочетаний весов, которые могут использоваться в модели. Чтобы его изобразить, мы можем определить систему координат, в которой каждому весу соответствует отдельная ось. У модели принятия решений по кредитам есть всего два веса: один для годового дохода и другой для текущей задолженности. Следовательно, пространство весов этой модели будет двухмерным. График в правой верхней части рис. 2.2 иллюстрирует часть пространства весов этой модели. На изображении выделено местоположение сочетания весов, использованное в этой модели (2,2). Каждая точка в этой системе координат описывает возможный набор весов модели и, следовательно, относится к определенной взвешенной сумме. Таким образом, перемещение по пространству весов эквивалентно изменению модели, поскольку при этом обновляется связь между ее вводом и выводом.

Линейная модель привязывает набор входных значений к точке в новом пространстве путем вычисления на их основе взвешенной суммы: каждый ввод умножается на вес, а результаты умножений суммируются. В нашей модели принятия решений по кредитам именно в этом пространстве применяется такое правило. Таким образом, его можно было бы назвать пространством принятия решений, но по причинам, которые станут понятными после описания структуры нейрона в следующей главе, его называют пространством активации. Оси пространства активации модели соответствуют ее взвешенному вводу. Следовательно, каждая точка этого пространства определяет набор взвешенных входных значений. Применение правила принятия решений (такого как наше правило *кредит получает человек с кредитоспособностью выше 200*) к каждой точке этого пространства активации с последующей записью результатов принятого решения в каждом случае позволяет нанести на график границу решений модели в этом пространстве. Граница решений отделяет точки, превышающие пороговое значение, от остальных точек. Пространство активации нашей модели принятия решений по кредитам проиллюстрировано на графике в левой нижней части рис. 2.2. Вы можете видеть положения четырех заявлений на получение кредита, перечисленных в табл. 2.1, спроецированные на это пространство активации. Черная прямая, проходящая по диагонали, обозначает границу решений, с учетом которой третье заявление принимается, а остальные отклоняются. При желании эту границу можно спроецировать обратно на исходное пространство ввода, если записывать для каждой координаты в этом пространстве информацию о том, по какую сторону от границы решений она оказывается в результате использования функции взвешенной суммы. На графике в правой

нижней части рис. 2.2 показана граница решений, проведенная в исходном пространстве ввода (обратите внимание на то, что у осей другие значения) и сгенерированная с использованием этого процесса. Мы еще вернемся к концепции пространства весов и границы решений в следующей главе при описании того, как корректирование параметров нейрона изменяет сочетания входных значений, обеспечивающих высокую степень его активации.

Итоги

Основная идея, изложенная в этой главе, состоит в том, что линейная математическая модель, выраженная как уравнение или нанесенная на график в виде прямой, описывает отношение между набором входных значений и выводом. Следует помнить, что не все математические модели являются линейными, и примеры этого встречаются в данной книге. Однако основополагающее вычисление взвешенной суммы входных значений действительно определяет линейную модель. Еще одна важная мысль, с которой вы познакомились в этой главе, — у линейной модели (взвешенной суммы) есть набор параметров (то есть весов), которые используются во взвешенной сумме. Корректируя эти параметры, мы можем менять отношение между вводом и выводом, которое описывает модель. При желании мы можем устанавливать эти веса вручную, используя наш опыт в предметной области, но для этого можно применять и машинное обучение, приспосабливая модель к закономерностям, найденным в наборе данных. Последняя ключевая идея этой главы заключается в том, что сложные модели можно создавать, объединяя простые. При этом вывод одной или нескольких моделей используется в качестве ввода другой модели. Мы применяли эту методику при определении нашей составной модели для принятия решений по кредитам. Как вы узнаете из следующей главы, эта модель по своей структуре очень похожа на нейроны в нейронной сети. Нейрон тоже вычисляет взвешенную сумму своих входных значений и затем передает результаты вычислений второй модели, которая решает, следует его активировать или нет.

В этой главе мы попытались познакомить вас с основными концепциями, прежде чем переходить к терминологии машинного и глубокого обучения. Чтобы продемонстрировать, как эти концепции и термины соотносятся между собой, отметим, что наша модель принятия решений аналогична нейрону с двумя вводами, который использует пороговую функцию активации. Два финансовых индикатора (годовой доход и текущая задолженность)

эквивалентны входным значениям, которые принимает нейрон. Набор индикаторов, описывающих отдельный образец, иногда называют вектором ввода или вектором признаков; в этом случае роль образца играет один заявитель, описанный с использованием двух признаков: годового дохода и текущей задолженности. К тому же, как и модель принятия решений по кредитам, нейрон назначает вес каждому вводу, умножая их попарно и затем слагая результаты этих умножений для получения общей величины входных значений. Наконец, аналогично тому, как мы применяли пороговое значение к рейтингу кредитоспособности, чтобы принять решение о выдаче кредита, нейрон применяет функцию (известную как функция активации) для преобразования общей величины ввода. На ранних этапах развития нейронных сетей роль функций активации играли пороговые функции, которые работали точно так же, как пороговый рейтинг, использованный в данном примере с выдачей кредита. В более современных сетях применяются разные виды функции активации (такие как логистическое уравнение, tanh или ReLU). Мы познакомимся с ними в следующей главе.

НЕЙРОННЫЕ СЕТИ КАК СОСТАВНЫЕ ЭЛЕМЕНТЫ ГЛУБОКОГО ОБУЧЕНИЯ

Термин *глубокое обучение* описывает семейство моделей *нейронных сетей* с многочисленными слоями простых программ для обработки информации, известных как нейроны и объединенных в сеть. В этой главе мы попытаемся дать четкое и комплексное введение в принципы работы этих нейронов и объяснить, как их взаимные связи формируют искусственные нейронные сети. В последующих главах мы покажем, как эти сети обучаются с использованием данных.

Нейронная сеть — это вычислительная модель, созданная по аналогии со структурой человеческого мозга. Человеческий мозг состоит из огромного количества нервных клеток под названием нейроны. По некоторым оценкам их число в головном мозге достигает ста миллиардов (Геркулано-Хаусел, 2009). Нейроны имеют простую трехуровневую структуру: тело, набор отростков (дендритов) и один большой отросток под названием *аксон*. Эта структура и то, как нейроны соединены между собой в головном мозге, показано на рис. 3.1. Дендриты и аксоны исходят из тела клетки, причем дендриты одних нейронов соединяются с аксонами других. Дендриты играют роль каналов ввода, принимая сигналы, отправляемые другими нейронами по своим аксонам. Аксон выступает как выходной канал, и другие нейроны, дендриты которых к нему подключены, принимают его сигналы в качестве ввода.

Нейроны работают очень похожим образом. Если входящие сигналы достаточно сильны, они посылают по своему аксону электрический импульс под названием *потенциал действия*, который передается другим подключенным к нему нейронам. Таким образом, нейрон играет роль реле, которое принимает набор входных сигналов и либо выдает потенциал действия, либо нет.

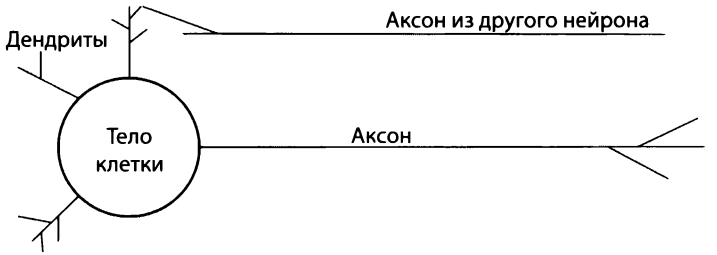


Рис. 3.1. Структура нейрона в головном мозге

С точки зрения биологии это весьма упрощенное объяснение. Однако оно охватывает основные аспекты, необходимые для понимания аналогии между структурой человеческого мозга и вычислительными моделями, известными как нейронные сети. Эти аспекты перечислены ниже.

- Головной мозг состоит из большого количества простых взаимосвязанных элементов под названием нейроны.
- Работа головного мозга в некотором смысле заключается в обработке информации, закодированной в виде сильных и слабых электрических сигналов (или потенциалов активации), которые распространяются по сети нейронов.
- Каждый нейрон принимает набор сигналов от своих соседей и преобразует их в вывод с сильным или слабым сигналом.

Эти характеристики свойственны любой вычислительной модели, которая используется в нейронных сетях.

Искусственные нейронные сети

Искусственная нейронная сеть состоит из множества простых элементов обработки информации, которые называются нейронами. Способность нейронной сети моделировать сложные отношения является результатом не сложных математических моделей, а, скорее, взаимодействий между большим количеством простых нейронов.

Структура нейронной сети проиллюстрирована на рис. 3.2. Принято считать, что нейроны в сети организованы в виде слоев. В сети, изображенной ниже, таких слоев насчитывается пять: один входной, три скрытых и один

выходной. Скрытым называют слой, который не является ни входным, ни выходным. Сети глубокого обучения — это нейронные сети с множеством скрытых слоев. Чтобы сеть считалась глубокой, число скрытых слоев должно быть не меньше двух. Однако в большинстве случаев их намного больше. Следует отметить, что глубина сети равна количеству скрытых слоев плюс выходной слой.

На рис. 3.2 прямоугольники во входном слое обозначают ячейки памяти, с помощью которых изображают ввод сети. Эти ячейки можно считать нейронами восприятия. Они не занимаются обработкой информации, а просто выводят то значение, которое хранится в ячейке памяти.

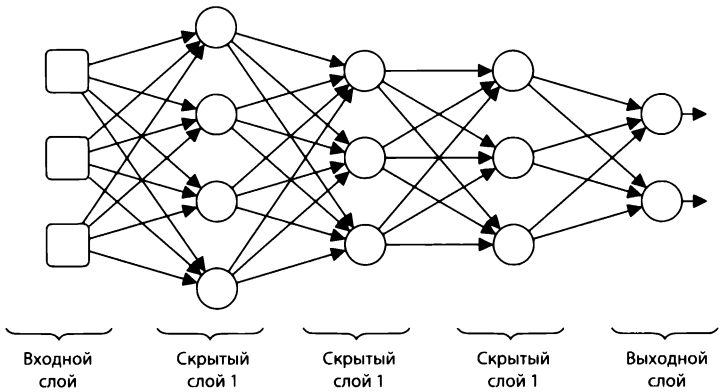


Рис. 3.2. Схематическая топология простой нейронной сети

Кружками на диаграмме обозначаются нейроны, обрабатывающие информацию. Каждый из этих нейронов принимает на вход набор числовых значений и привязывает их к единому выводу. Любой ввод обрабатывающего нейрона является либо выводом нейрона восприятия, либо выводом другого обрабатывающего нейрона.

Стрелки на рис. 3.2 иллюстрируют прохождение информации по сети, от вывода одного нейрона до ввода другого. Каждая связь в сети соединяет два нейрона и является направленной; это означает, что информация, передающаяся по этому соединению, двигается лишь в одном направлении. У каждой такой связи есть свой *вес*; это просто число, но оно играет важную роль. Вес соединения влияет на то, как нейрон обрабатывает поступающую по нему информацию. Обучение искусственной нейронной сети фактически сводится к поиску лучшего (или оптимального) сочетания весов.

Сети глубокого обучения — это нейронные сети с множеством скрытых слоев.

Как искусственный нейрон обрабатывает информацию

Обработка информации внутри нейрона (то есть соотношение входных значений с выводом) очень похожа на модель принятия решений по кредитам, которую мы разработали в главе 2. Как вы, наверное, помните, эта модель сначала вычисляла взвешенную сумму для входных признаков (дохода и задолженности). Веса, использовавшиеся во взвешенной сумме, корректировались с учетом набора данных таким образом, чтобы результат вычисления дохода и задолженности заявителя в качестве ввода являлся как можно более точной оценкой рейтинга его кредитоспособности. Второй этап обработки в модели принятия решений по кредитам заключался в применении к результатам вычисления взвешенной суммы (приблизительному рейтингу кредитоспособности) правила принятия решений. Это правило имело вид функции, которая преобразовывала рейтинг кредитоспособности в решение о том, выдавать заявителю кредит или нет.

Нейрон также осуществляет двухэтапный процесс для сопоставления входных значений с выводом. На первом этапе происходит вычисление взвешенной суммы значений, поданных на вход нейрону. Затем результат пропускается через вторую функцию, которая связывает его с итоговым выходным значением. При проектировании нейрона на втором этапе можно использовать много разных видов функций: это может быть как простое правило о принятии решений из нашего примера с выдачей кредитов, так и что-то более сложное. Обычно вывод нейрона называют значением активации, поэтому вторая функция, связывающая его с взвешенной суммой, называется функцией активации.

То, как эти этапы обработки выражены в структуре искусственного нейрона, проиллюстрировано на рис. 3.3. Символ Σ означает вычисление взвешенной суммы, а символ φ обозначает функцию активации, которая принимает эту сумму и генерирует вывод нейрона.

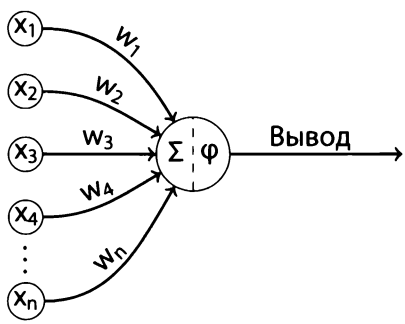


Рис. 3.3. Структура искусственного нейрона

Нейрон, представленный на рис. 3.3, принимает n входных значений $[x_1, \ldots, x_n]$ по n входным соединениям, у каждого из которых есть свой вес $[w_1, \ldots, w_n]$. Для вычисления взвешенной суммы входные значения умножаются на веса и слагаются в итоговую величину. Математически эту операцию можно записать так:

$$z = (x_1 \times w_1) + (x_2 \times w_2) + \ldots + (x_n \times w_n)$$

Эту формулу можно сделать более компактной:

Формула

Например, если нейрон с весами $[w_1 = -3, w_2 = 1]$ получил ввод вида $[x_1 = 3, x_2 = 9]$, взвешенная сумма будет следующей:

$$z = (3 \times -3) + (9 \times 1) = 0$$

Второй этап обработки внутри нейрона состоит в применении к взвешенной сумме (значению z) *функции активации*. На рис. 3.4 показаны графики ряда потенциальных функций активации, ввод которых (z) находится в диапазоне [–1, … +1] или [–10, … +10] в зависимости от того, какой из этих интервалов лучше всего иллюстрирует график функции. В верхней части рис. 3.4 представлена пороговая функция активации. Примером таковой может служить правило о принятии решений, которое мы использовали в случае с выдачей кредитов; в роли порога в этом правиле выступал любой рейтинг кредитоспособности выше 200. Пороговая активация была популярной на ранних этапах исследования нейронных сетей. В центре рис. 3.4 показаны графики функций активации на основе *логистического уравнения* и *гиперболического тангенса* (tanh). Нейроны, которые их использовали, до недавних пор пользовались популярностью в многослойных сетях. Внизу рис. 3.4 показан график выпрямляющей (или шарнирной, или положительной линейной) функции активации. Она широко применяется в современных нейронных сетях; в 2011 году выпрямляющая функция была представлена как улучшенное средство для обучения глубоких сетей (Глорот и др., 2011). Как вы узнаете, знакомясь с историей глубокого обучения в главе 4, одна из тенденций в исследовании нейронных сетей заключается в переходе с пороговой функции активации на логистическое уравнение и tanh, а затем на выпрямляющую функцию.

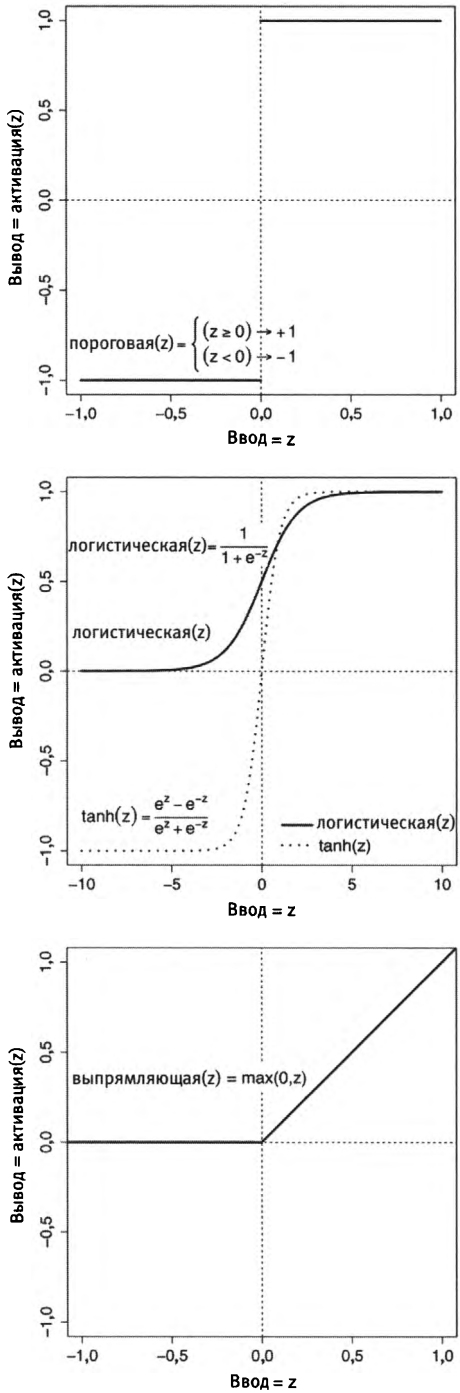


Рис. 3.4. Сверху вниз: пороговая функция, логистическое уравнение и tanh, выпрямляющая линейная функция

Вернемся к нашему примеру. Результат вычисления взвешенной суммы был таков: z = 0. На рис. 3.4 (сплошная кривая на среднем рисунке) изображен график логистической функции активации. Если предположить, что нейрон использует именно эту функцию, данный график показывает, как взвешенная сумма соотносится с выходной активацией: logistic (0) = 0,5. Вычисление выходной активации этого нейрона можно подытожить так:

$$\text{Вывод} = \text{функция\_активации}\left(z = \sum_{i=1}^{n} x_i \times w_i\right) =$$
$$\text{logistic}(z = (3 \times -3) + (9 \times 1)) =$$
$$\text{logistic}(z = 0) = 0{,}5$$

Обратите внимание: процесс обработки информации в этом нейроне почти идентичен тому, который использовался в модели принятия решений по кредитам из предыдущей главы. Основное отличие в том, что пороговое правило о принятии решений, которое привязывало показатель взвешенной суммы к итоговому выводу (одобрению или отклонению заявления), было заменено логистической функцией, превращающей показатель взвешенной суммы в значение между 0 и 1. В зависимости от положения этого нейрона в сети его выходная активация (в данном случае $y = 0{,}5$) станет либо вводом одного или нескольких нейронов в следующем слое, либо частью общего вывода сети. Если нейрон находится в выходном слое, интерпретация его выходного значения будет зависеть от задачи, которую он помогает моделировать. Если он размещен в одном из скрытых слоев сети, его вывод, возможно, удастся интерпретировать лишь в том смысле, что он представляет некий производный признак (похожий на индекс BMI, рассмотренный в главе 1), который сеть посчитала полезным для генерации общих результатов. Мы еще вернемся к проблеме интерпретации вывода отдельных нейронов сети в главе 7.

Ключевая мысль этого раздела заключается в том, что нейрон, элементарный составной элемент нейронной сети и глубокого обучения, определяется двумя последовательными операциями: вычислением взвешенной суммы и передачей результата в функцию активации.

Рисунок 3.4 демонстрирует, что ни tanh, ни логистическое уравнение не являются линейными функциями. На самом деле они имеют характерные графики в форме буквы «S» (а не прямой). Не все функции активации выглядят так (например, пороговая и выпрямляющая функции имеют другую форму), но все они устанавливают нелинейную связь вывода с взвешенной суммой. Появление в нейронах нелинейных отношений привело к тому, что сегодня мы используем функции активации.

Почему необходима функция активации?

Чтобы понять, зачем в нейроне нужна нелинейная связь, необходимо учитывать, что вся работа нейронной сети, в сущности, сводится к определению связи между вводом и выводом — например, между текущей позицией в игре го и ее оценкой или между рентгеновским снимком и диагнозом пациента. Нейроны являются элементарными составляющими нейронных сетей и, как следствие, связей, которые эти сети определяют. Общее отношение между вводом и выводом сети состоит из аналогичных отношений каждого ее нейрона. Из этого следует, что если ограничить все нейроны линейными связями (то есть вычислением взвешенной суммы), то же ограничение будет свойственно и всей сети в целом. Однако многие отношения в окружающем мире, которые мы хотели бы сгенерировать, являются нелинейными, и, если попытаться втиснуть их в рамки линейной модели, результат получится очень неточным. Применение линейной модели для моделирования нелинейного отношения было бы примером недообучения, которое мы обсуждали в главе 1; оно происходит, когда модель, используемая для кодирования закономерностей в наборе данных, слишком простая и, как результат, неточная.

Два значения имеют линейное отношение, если изменение одного из них приводит к пропорциональному изменению другого. Например, если у работника фиксированная почасовая оплата, которая не меняется на выходных или при выполнении сверхурочной работы, это означает, что между количеством проработанных часов и заработной платой существует линейное отношение. График зависимости рабочих часов от заработанных денег будет иметь вид прямой линии: чем стремительней она поднимается вверх, тем выше почасовая оплата. Но мы можем сделать систему оплаты для нашего потенциального работника немного сложнее — например, увеличив почасовую ставку при сверхурочной работе или на выходных. В этом случае отношение между количеством проработанных часов и оплатой перестанет быть линейным. Нейронные сети, особенно сети глубокого обучения, обычно используются для моделирования отношений куда более сложных, чем в этом примере с зарплатой. Чтобы достичь результата высокой степени точности, сеть должна быть способна усваивать и представлять сложные нелинейные связи. И для их реализации на этапе обработки данных внутри нейронов должен быть предусмотрен нелинейный этап (функция активации).

В принципе, любая нелинейная функция активации позволяет нейронной сети сформировать нелинейное отношение между вводом и выводом.

Но, как вы увидите позже, большинство функций, изображенных на рис. 3.4, обладают замечательными математическими свойствами, которые помогают в обучении нейронных сетей, и именно поэтому они настолько популярны в этой области.

То, что придание поведению нейронов некой нелинейности позволяет сети формировать нелинейные отношения между вводом и выводом, еще раз подтверждает: общее поведение сети определяется совокупностью работ, выполняемых отдельными ее нейронами. Нейронные сети решают задачи по принципу «разделяй и властвуй»: каждый нейрон занимается отдельным аспектом, и общая задача решается путем объединения этих аспектов в единое решение. Важная особенность нейронных сетей, которая делает их такими мощными, состоит в том, что во время обучения, когда устанавливаются веса для внутренних соединений, сеть учится разбивать большую задачу на мелкие части, делегируя решение этих подзадач отдельным нейронам с последующим объединением результатов.

Нейроны сети могут использовать разные функции активации. Но в целом каждый слой сети состоит из нейронов одного типа (с одинаковыми функциями активации). Нейроны также иногда называют модулями, разделяя их в соответствии с тем, какие функции активации они используют: нейроны с пороговой функцией активации называются пороговыми модулями, модули с логистической функцией активации называются логистическими, а нейроны, в которых применяется выпрямляющая функция активации, объединяются в так называемые выпрямляющие модули (англ. rectified linear units, или ReLU). Например, у сети может быть слой модулей ReLU, соединенный со слоем логистических модулей. Выбор функции активации для тех или иных нейронов сети осуществляет специалист по анализу данных, который проектирует сеть. Для принятия решения специалист может провести ряд экспериментов, чтобы проверить, какая функция активации показывает лучшую производительность для заданного набора данных. Однако часто выбор делается исходя из того, какая функция активации является наиболее популярной на текущий момент. Например, сейчас в нейронных сетях больше всего распространены модули типа ReLU, но ситуация может поменяться, если кто-то разработает новую функцию активации и продемонстрирует ее преимущества. Как будет отмечено в конце этой главы, элементы нейронных сетей, которые задаются вручную специалистом по анализу данных еще до процесса обучения, называются гиперпараметрами.

Термин «гиперпараметр» описывает участки модели, определяемые вручную, чтобы их можно было отличить от параметров модели, которые

Нейронные сети решают задачи по принципу «разделяй и властвуй»: каждый нейрон занимается отдельным аспектом, и общая задача решается путем объединения этих аспектов в единое решение.

устанавливаются автоматически алгоритмом машинного обучения. Параметры нейронной сети — это веса, которые используются нейронами при вычислении взвешенной суммы. Как уже упоминалось в главах 1 и 2, стандартный процесс обучения для определения этих параметров (весов сети) начинается с присваивания им произвольных значений; затем по ходу этого процесса веса постепенно регулируются, повышая точность модели в отношении набора данных. В главе 6 описываются два алгоритма, которые чаще всего применяют в нейронных сетях: градиентный спуск и обратное распространение. Далее мы сосредоточимся на том, как изменение параметров нейрона влияет на его поведение в ответ на принимаемый ввод.

Как изменение параметров нейрона влияет на его поведение?

Параметры нейронной сети — это веса, с помощью которых нейрон вычисляет взвешенную сумму. Те же вычисления применяются и в линейной модели, но в нейроне отношения между весами и конечным выводом более сложные, так как, прежде чем сгенерировать итоговый вывод, взвешенная сумма проходит через функцию активации. Чтобы понять, как нейрон принимает решение по заданному вводу, необходимо рассмотреть отношения между весами нейрона, вводом, который он принимает, и выводом, который он генерирует в ответ.

Самой наглядной связью между весами и конечным выводом для заданного ввода обладают нейроны с пороговой функцией активации. Нейрон, использующий функцию этого типа, является эквивалентом модели принятия решений по кредитам, с помощью которой мы классифицировали рейтинги кредитоспособности, сгенерированные путем вычисления взвешенной суммы (чтобы одобрить или отклонить соответствующие заявления). В конце главы 2 вы познакомились с концепцией пространств ввода, весов и активации (см. рис. 2.2). Пространство ввода для модели принятия решений по кредитам с двумя входными переменными можно представить в виде плоскости: одна переменная (годовой доход) формирует ось x, а другая (текущая задолженность) — ось y. Каждая точка на этом графике обозначает потенциальное сочетание входных переменных модели, а набор точек на плоскости ввода определяет множество возможных входных значений, которые модель способна обработать. Веса, применяемые в этой модели, можно представить в виде прямой, которая разделяет пространство ввода на две области: первая область содержит все входные значения, которые приводят к выдаче кредита,

а значения во второй соответствуют заявлениям, которые в итоге отклоняются. В этом примере изменение весов, применяемых в модели принятия решений, влияет на то, какие заявления принимаются, а какие отклоняются. Это вполне логично, поскольку при этом меняется степень важности, придаваемая нами величине доходов и величине задолженности заявителя, при принятии решения о выделении ему кредита.

В обобщенном виде описанную выше модель принятия решений по кредитам можно представить в виде нейрона нейронной сети. Эквивалентом этой модели можно считать нейрон с двумя входными значениями и пороговой функцией активации. Он также имеет аналогичное пространство ввода. На рис. 3.5 показаны три графика пространства ввода для нейрона с двумя входами и пороговой функцией активации, который активируется, только если взвешенная сумма больше нуля. Графики отличаются тем, что нейрон определяет разные границы решений (она представлена в виде черной прямой).

При создании каждого графика на рис. 3.5 веса нейрона задавались вручную. Затем для каждой точки в пространстве ввода записывался итоговый результат (сильная или слабая активация), когда координаты точки использовались в качестве ввода нейрона. Входные точки, в которых нейрон показывал сильную активацию, выделены серым цветом, а остальные — белым. Единственное отличие между нейронами, на основе которых создавались эти графики, заключалось в весах, использованных при вычислении взвешенной суммы входных значений. Стрелка на каждом графике обозначает вектор веса, применяемый нейроном. В данном примере этот вектор описывает направление от начальной позиции к точке и расстояние между ними*. Далее станет ясно, что интерпретация набора весов, используемых нейроном, в качестве вектора (стрелки, ведущей от начальной позиции к координатам весов) в его пространстве ввода помогает продемонстрировать, как изменение весов влияет на границу решений.

---

* Начальная позиция (начало координат) — это точка в системе координат, в которой пересекаются оси. В двухмерном пространстве это место пересечения осей *x* и *y* — то есть точка с координатами x=0, y=0.

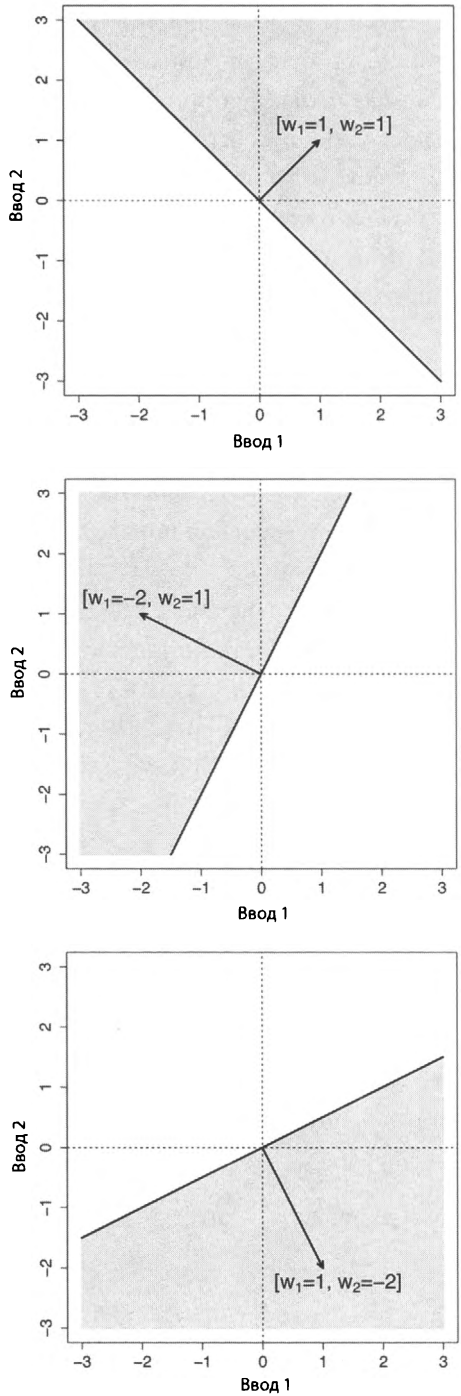


Рис. 3.5. Границы решений для нейрона с двумя входами. Сверху вниз: векторы весов $[w_1=1, w_2=1]$, $[w_1=-2, w_2=1]$ и $[w_1=1, w_2=-2]$

Во всех графиках использовались разные веса. Это отражено в разных направлениях стрелки (вектора веса). В частности, изменение веса приводит к вращению вектора вокруг начальной позиции. Обратите внимание: в каждом графике граница решений зависит от того, куда указывает вектор, всегда оставаясь перпендикулярной (то есть располагаясь под прямым углом) по отношению к нему. Поэтому изменение весов вращает не только сам вектор, но и границу решений нейрона. Это вращение влияет на множество входных значений, которые приводят к сильной активации нейрона (серая область).

Чтобы понять, почему граница решений всегда перпендикулярна вектору весов, необходимо обратиться к линейной алгебре. Как вы помните, каждая точка в пространстве ввода обозначает потенциальную комбинацию входных значений нейрона. Теперь представьте, что каждая из этих комбинаций определяет направление стрелки из начальной позиции в соответствующую точку в пространстве ввода. У каждой точки будет своя отдельная стрелка. И каждая из этих стрелок очень похожа на вектор весов, однако в отличие от него она указывает на координаты входных значений, а не на координаты весов. Если представить набор входных значений в виде вектора, вычисление взвешенной суммы можно выполнить путем умножения вектора весов на вектор ввода. В линейной алгебре умножение векторов называют скалярным произведением. Вам достаточно знать только то, что результат скалярного произведения зависит от угла между двумя умножаемыми векторами. Если угол между ними меньше 90°, результат будет положительным, а если нет — отрицательным. Таким образом, мы получим положительное значение для всех векторов ввода, которые находятся под острым углом к векторам весов; для всех остальных векторов значение будет отрицательным. Наш нейрон демонстрирует сильную активацию, только если ему на вход поступают положительные значения. Следовательно, граница решений находится под прямым углом к вектору весов, так как все входные значения, векторы которых имеют острый угол по отношению к вектору весов, приведут к передаче функции активации положительного ввода, в результате чего нейрон активируется с высоким выводом; для всех остальных входных значений вывод активации нейрона будет низким.

Вернемся к графикам на рис. 3.5. В каждом случае границы решений находятся под разными углами, но все они проходят через точку пространства, из которой выходит вектор (то есть через начальную позицию). Это показывает, что изменение весов нейрона ведет к вращению границы решений, но не производит ее параллельного переноса. При параллельном переносе границы решений она сместилась бы вперед или назад вдоль вектора весов

и перестала пересекать его в начале координат. Правило, согласно которому все границы решений должны проходить через начало координат, ограничивает возможности нейрона в плане выявления различий между двумя входными образцами. Для преодоления этого ограничения обычно расширяют вычисление взвешенной суммы за счет дополнительного элемента под названием *сдвиг* (или смещение). Это не то же самое, что индуктивный сдвиг, о котором шла речь в главе 1. Этот элемент больше похож на параметр отсечения в уравнении прямой, который смещает график вверх и вниз по оси *y*. Он нужен для того, что переместить границу решений относительно начальной позиции.

Сдвиг — это просто дополнительное значение, которое вводится в вычисление взвешенной суммы. Для этого взвешенная сумма, прежде чем попасть в функцию активации, сдвигается. Вот уравнение, которое описывает этапы обработки ввода в нейроне (сдвиг обозначен символом b):

$$\text{Вывод} = \text{activation\_function}\left(\underbrace{x = \left(\sum_{i=1}^{n} x_i \times w_i\right)}_{\text{взвешенная сумма}} + \underbrace{b}_{\text{сдвиг}}\right)$$

На рис. 3.6 показано, как значение сдвига влияет на границу решений нейрона. Если сдвиг отрицательный, граница решений отдаляется от начальной позиции в том направлении, куда указывает вектор весов (как в верхнем и среднем графиках на рис. 3.6); если сдвиг положительный, граница решений перемещается в противоположном направлении (нижний график на рис. 3.6). В обоих случаях граница решений остается перпендикулярной по отношению к вектору весов. Величина сдвига также влияет на то, как далеко граница решений переместится от начальной позиции: чем больше сдвиг, тем дальше она окажется (сравните верхний график на рис. 3.6 со средним и нижним).

Значение сдвига лучше не устанавливать вручную, а позволить нейрону сформировать его автоматически. Чтобы это сделать, сдвиг проще всего считать дополнительным весом, который корректируется в процессе обучения нейрона, как и все остальные веса. Для этого лишь нужно добавить в каждый вектор ввода дополнительное входное значение, равное 1. Для удобства его можно обозначить как ввод 0 ($x_0 = 1$), а сам сдвиг будет выглядеть как вес 0 (w_0)*. Структура искусственного нейрона с внедренным сдвигом w_0 показана на рис. 3.7.

---

* В главе 2 мы использовали тот же подход для внедрения параметра отсечения линейной модели в ее веса.

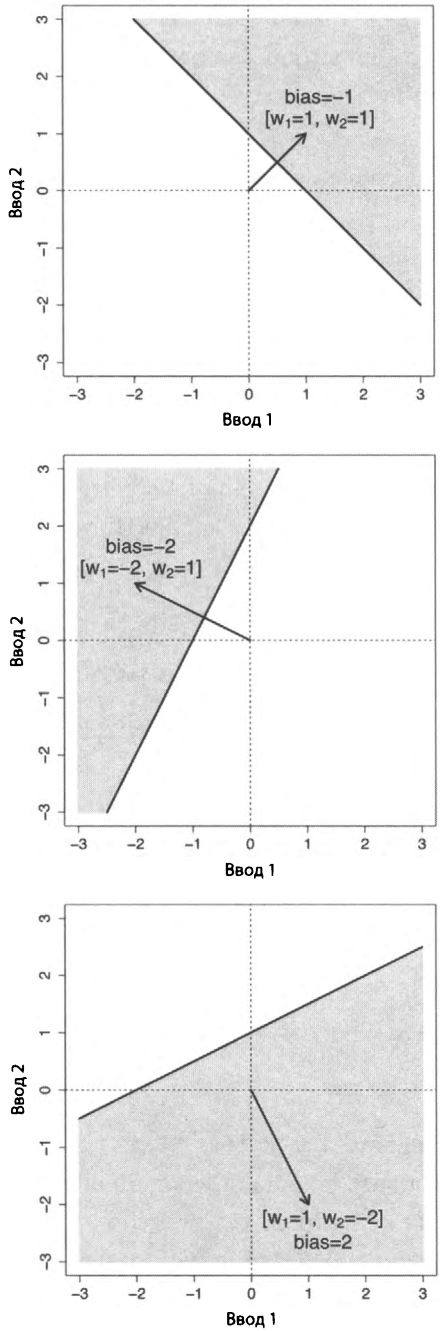


Рис. 3.6. Графики границы решений для нейрона с двумя входами, которые иллюстрируют, как на нее влияет сдвиг. Сверху вниз: вектор весов $[w_1=1, w_2=1]$ со сдвигом –1, вектор весов $[w_1=-2, w_2=1]$ со сдвигом –2 и вектор весов $[w_1=1, w_2=-2]$ со сдвигом 2

После внедрения сдвига в набор весов нейрона уравнение, определяющее связь между вводом и выходной активацией, можно упростить следующим образом (по крайней мере, с точки зрения того, как оно записывается):

$$\text{Вывод} = \text{функция\_активации}\left(z = \sum_{i=0}^{n} x_i \times w_i\right)$$

Обратите внимание на то, что в этом уравнении индекс *i* находится в диапазоне от 0 до n, так как теперь он включает фиксированное входное значение $x_0 = 1$ и сдвиг w_0; в предыдущей версии этого уравнения индекс начинался с 1. Это означает, что нейрон способен сформировать сдвиг, подбирая подходящий вес w_0. Для этого используется тот же процесс, что и для остальных весов: в начале обучения сдвиг каждого нейрона в сети инициализируется с помощью произвольного значения и затем корректируется в зависимости от того, какую производительность показывает сеть для заданного набора данных.

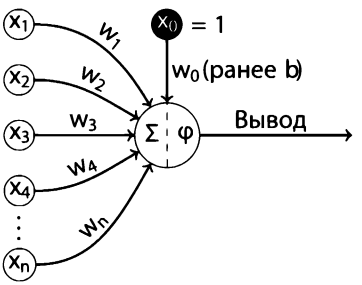


Рис. 3.7. Искусственный нейрон со сдвигом, внедренным как w_0

Ускорение обучения нейронных сетей с помощью графических адаптеров

Сдвиг в набор весов внедряется не только для удобства записи уравнений: это позволяет использовать специализированное оборудование для ускорения процесса обучения нейронных сетей. Благодаря тому, что сдвиг можно считать еще одним весом, можно представить взвешенную сумму входных значений (включая сдвиг) в виде произведения двух векторов. Как уже отмечалось ранее (при объяснении того, почему граница решений перпендикулярна вектору весов), набор входных значений можно считать вектором. И поскольку существенная доля вычислений в нейронных сетях приходится на умножение векторов и матриц, появляется возможность использовать

специализированное аппаратное обеспечение для ускорения этих операций. Примерами устройств, специально созданных для скоростного умножения матриц, являются графические адаптеры (англ. graphics processing units, или GPU).

В стандартной сети с прямой связью все нейроны одного слоя принимают вывод (активацию) всех нейронов предыдущего слоя. Это означает, что все нейроны, принадлежащие одному слою, получают один и тот же ввод. Так мы можем посчитать взвешенную сумму для всего слоя, используя лишь одну операцию умножения вектора на матрицу. Это намного быстрее, чем вычислять отдельную взвешенную сумму для каждого нейрона в слое. Чтобы этого добиться, мы помещаем вывод каждого нейрона из предыдущего слоя в вектор и храним все веса соединений между двумя слоями нейрона в матрице. Затем мы умножаем вектор на матрицу, в результате чего получается вектор со взвешенными суммами для всех нейронов.

На рис. 3.8 показано, как вычисление взвешенных сумм для всех нейронов в отдельно взятом слое сети можно выполнить за счет единственной операции умножения матриц. Данная диаграмма состоит из двух отдельных частей: слева показаны соединения между нейронами двух слоев сети, а справа — матричная операция, которая вычисляет взвешенные суммы нейронов во втором слое. Чтобы было легче проследить, как соотносятся две эти диаграммы, слева мы выделили соединение с нейроном E, а справа — взвешенную сумму в том же нейроне.

Обратите внимание на правую часть диаграммы. Вектор размером 1×3 (1 строка, 3 столбца) слева внизу хранит результаты для нейронов первого слоя; стоит отметить, что это вывод, полученный из функции активации ϕ (какой именно, здесь не уточняется — это может быть пороговая или логистическая функция, tanh или ReLU). Матрица размером 3× 4 (3 строки и 4 столбца) в правой верхней части хранит веса соединений между двумя слоями нейронов. В каждом ее столбце находятся веса соединений, подключенных к входам нейронов второго слоя сети. В первом столбце хранятся веса нейрона D, во втором — веса нейрона E и т.д.*. Умножая вектор 1×3 с результатами слоя 1 на матрицу весов 3×4, мы получаем вектор размером 1×4, в котором находятся взвешенные суммы четырех нейронов в слое 2: z_D — это взвешенная сумма входных значений нейрона D, z_E — это взвешенная сумма входных значений нейрона E и т. д.

---

* Чтобы подчеркнуть это вертикальное расположение, веса были проиндексированы в порядке «столбец — строка», а не «строка — столбец».

Чтобы сгенерировать вектор 1×4 со взвешенными суммами нейронов из слоя 2, мы последовательно умножаем вектор активации на каждый столбец матрицы. Сначала первый (крайний слева) элемент вектора умножается на первый (самый верхний) элемент столбца, затем второй элемент вектора умножается на второй сверху элемент столбца и т. д., пока каждый элемент вектора не будет умножен на соответствующий элемент в столбце. В конце полученные результаты суммируются и сохраняются в векторе вывода. На рис. 3.8 проиллюстрирован процесс умножения вектора активации на второй столбец в матрице весов (который содержит веса для нейрона E) с последующим сохранением суммы этих умножений в векторе вывода в виде значения z_E.

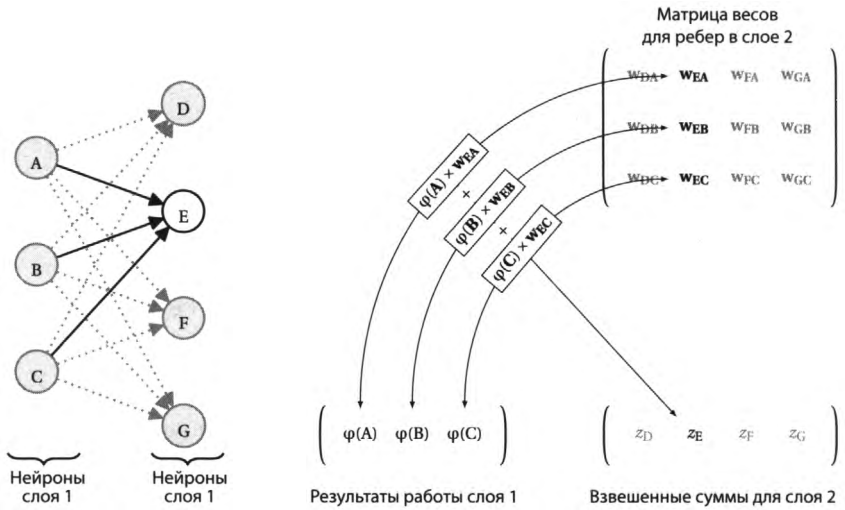


Рис. 3.8. Графическая иллюстрация топологических соединений отдельно взятого нейрона E и соответствующего вектора на примере умножения, в результате которого вычисляются взвешенные суммы входных значений этого нейрона и его соседей по слою*

Фактически вычисления, выполняемые всей нейронной сетью, можно представить в виде цепочки скалярных произведений с поэлементным применением функции активации к результатам каждого умножения. На рис. 3.9 видно, что нейронную сеть можно изобразить как в виде графа (слева), так и в виде последовательности матричных операций (справа). Во втором случае символ × обозначает стандартное скалярное произведение (описанное выше), а применение функции активации к каждому элементу вектора,

* Подробней о размере и увеличении сетей см.: Гудфеллоу и др., 2016. Стр. 23.

созданному путем предыдущего умножения матриц, записывается как →ϕ→. Вывод поэлементного применения функции активации представляет собой еще один вектор, который содержит вывод нейронов в определенном слое сети. Чтобы сделать соответствие между двумя представлениями более наглядным, в обоих случаях входные значения обозначены как I_1 и I_2, вывод трех скрытых модулей — как A_1, A_2 и A_3, а итоговый вывод сети — как y.

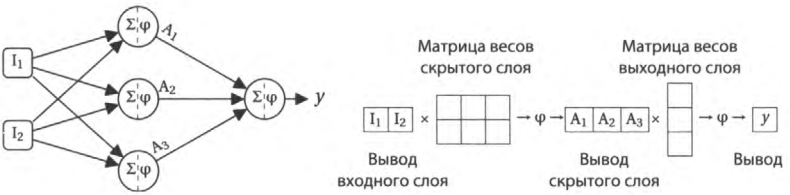


Рис. 3.9. Представление одной и той же нейронной сети: графическое (слева) и в виде последовательности матричных операций (справа)

Следует добавить, что матричное представление дает возможность легко оценить глубину сети: она равна количеству слоев, с которыми связаны матрицы весов (или количеству матриц весов, которые требуются сети, что одно и то же). Вот почему при определении глубины сети не учитывается входной слой: для него не предусмотрена матрица весов.

Как уже упоминалось выше, тот факт, что большую часть вычислений в нейронной сети можно представить в виде последовательности матричных операций, имеет важные последствия для вычислений при глубоком обучении. Нейронная сеть может содержать больше миллиона нейронов, и существует тенденция удваивания размеров этих сетей каждые два-три года. Более того, обучение глубоких сетей состоит в их последовательном применении к образцам, взятым из очень больших наборов данных, с последующим обновлением параметров (то есть весов) для улучшения производительности. Следовательно, этот процесс может потребовать очень большого количества итераций, а каждая из них — состоять из миллионов вычислительных шагов. Вот почему ускорение вычислений, которого, к примеру, можно достичь за счет выполнения скалярных произведений с помощью GPU, сыграло настолько важную роль в развитии глубокого обучения.

Отношения между GPU и глубоким обучением пошли на пользу и самим графическим адаптерам. Повышенный спрос на GPU, вызванный интересом к глубокому обучению, существенно повлиял на производителей этих устройств. Компании начали исследовать новые направления. Традиционно они в основном уделяли внимание рынку компьютерных игр, так как

изначальной причиной появления графических чипов было улучшение отрисовки, что, естественно, применялось в игровой индустрии. Но в последние годы эти производители начали ориентировать свою продукцию на задачи, связанные с глубоким обучением и искусственным интеллектом. Более того, они заботятся о том, чтобы их устройства поддерживали самые популярные программные фреймворки для глубокого обучения.

Итоги

Основная мысль этой главы заключается в том, что сети глубокого обучения состоят из большого числа простых вычислительных модулей, которые вместе обучаются и формируют сложные связи на основе больших наборов данных. Эти простые модули, нейроны, выполняют двухэтапный процесс: сначала вычисляется взвешенная сумма входных значений, а полученный результат затем пропускается через нелинейную функцию, известную как функция активации. То, что взвешенную сумму для целого слоя нейронов можно эффективно вычислить с помощью всего одной операции умножения матриц, означает: нейронные сети можно считать последовательностью матричных операций. Это позволило использовать оборудование, оптимизированное для быстрого выполнения скалярного произведения (например, графические адаптеры), чтобы ускорить процесс обучения, что, в свою очередь, поспособствовало увеличению размеров нейронных сетей.

Модульная структура, свойственная нейронным сетям, позволяет разобраться в принципах их работы на самом фундаментальном уровне. Комплексное описание этого уровня вычислений было основной целью данной главы. С другой стороны, модульная природа нейронных сетей порождает массу вопросов, связанных с тем, как их лучше всего составлять для решения той или иной задачи. Например:

- Какие функции активации должны использовать нейроны сети?
- Из скольких слоев должна состоять сеть?
- Сколько нейронов должно быть в каждом слое?
- Каким образом нейроны должны соединяться друг с другом?

К сожалению, на многие из этих вопросов невозможно ответить в сугубо теоретической плоскости. Концепции, которых касаются эти вопросы, в сфере машинного обучения называют гиперпараметрами (что не следует

путать с параметрами модели). Параметры нейронной сети, веса на ребрах (соединениях) формируются в процессе обучения сети с использованием больших наборов данных. Для сравнения: гиперпараметры модели (в данном контексте это параметры архитектуры нейронной сети) и/или алгоритма обучения невозможно сформировать на основе одних лишь данных; их должен задать человек, создающий модель. Для этого применяются эвристические правила, интуиция или метод проб и ошибок. Зачастую существенная часть усилий, направленных на создание сети глубокого обучения, связана с проведением экспериментов в попытках ответить на вопросы о гиперпараметрах, и этот процесс называется оптимизацией гиперпараметров. В следующей главе мы рассмотрим историю развития глубокого обучения, и трудности, связанные со многими из этих вопросов, будут в центре нашего внимания. В последующих главах вы увидите, как различные подходы к решению этих проблем позволяют создавать сети с самыми разными характеристиками, и каждая из этих сетей подходит для решения определенного рода задач. Например, рекуррентные нейронные сети лучше всего справляются с обработкой последовательных данных и временных рядов, тогда как сверточные изначально создавались для работы с изображениями. Однако обе эти разновидности сетей состоят из одних и тех же вычислительных модулей, искусственных нейронов; отличия в их поведении и возможностях связаны с тем, как эти нейроны структурированы и скомпонованы.

КРАТКАЯ ИСТОРИЯ ГЛУБОКОГО ОБУЧЕНИЯ

Историю глубокого обучения можно разделить на три основных этапа эмоционального подъема и новаторства, а между ними — периоды разочарования. Это проиллюстрировано на рис. 4.1, на котором отмечены этапы самых существенных исследований: модули с пороговой логикой (с начала 1940-х до середины 1960-х), коннекционизм (с ранних 1980-х до середины 1990-х) и глубокое обучение (с середины 2000-х и до наших дней). Можно также заметить некоторые ключевые особенности, которые появлялись в нейронных сетях в каждый из трех периодов. Эти изменения стали важнейшими этапами развития глубокого обучения: переход с двоичных значений на непрерывные, переход с пороговых функций активации на логистические уравнения и tanh, а затем на ReLU, а также постепенное «углубление» сетей — от одного слоя ко многим, а затем к глубоким сетям. Наконец, в верхней части рис. 4.1 показаны некоторые из важнейших концептуальных прорывов, обучающих алгоритмов и архитектур моделей, которые поспособствовали развитию глубокого обучения.

Рисунок 4.1 фактически является визуализацией структуры этой главы. Представленные здесь концепции, как правило, будут рассматриваться в хронологическом порядке. Два серых прямоугольника обозначают две важные архитектуры сетей глубокого обучения: сверточные нейронные сети (англ. convolutional neural networks, или CNN) и рекуррентные нейронные сети (англ. recurrent neural networks, или RNN). В данной главе мы опишем историю развития этих двух сетевых архитектур, а в главе 5 дадим подробное описание того, как они работают.

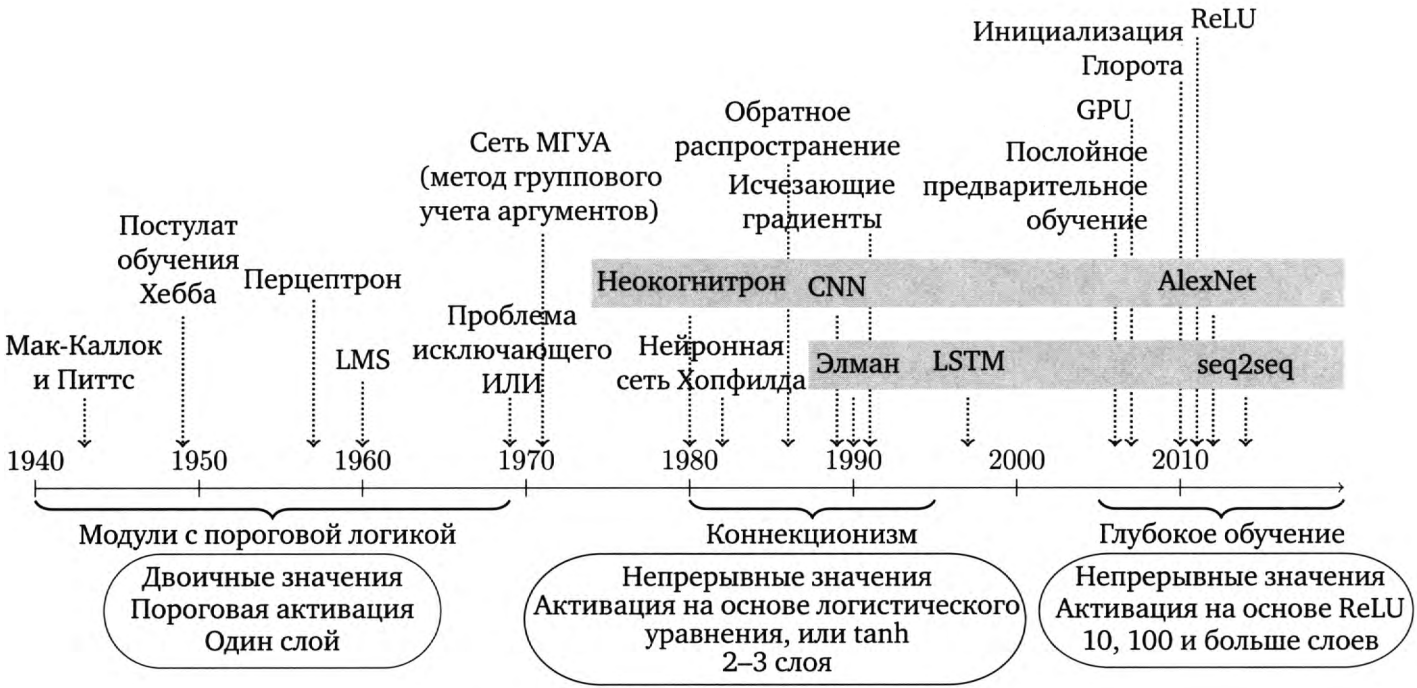

Инициализация
ReLU
Глорота
GPU
Обратное
распространение
Послойное
предварительное
обучение
Сеть МГУА
(метод группового
учета аргументов)
Исчезающие
градиенты
Постулат
обучения
Хебба
Перцептрон
Неокогнитрон
CNN
AlexNet
Проблема
исключающего
ИЛИ
Мак-Каллок
и Питтс
LMS
Нейронная
сеть Хопфилда
Элман
LSTM
seq2seq
1940
1950
1960
1970
1980
1990
2000
2010
Модули с пороговой логикой
Коннекционизм
Глубокое обучение
Двоичные значения
Пороговая активация
Один слой
Непрерывные значения
Активация на основе логистического
уравнения, или tanh
2–3 слоя
Непрерывные значения
Активация на основе ReLU
10, 100 и больше слоев

Рис. 4.1.

Ранние исследования: модули с пороговой логикой

Иногда в научной литературе по глубокому обучению ранние исследования в области нейронных сетей относят к кибернетике — науке, предметом изучения которой является разработка вычислительных моделей процессов управления и обучения, используемых биологическими организмами. Однако на рис. 4.1 используется терминология из книги Нилсон (1965), согласно которой эта работа заключалась в исследовании модулей с пороговой логикой, так как данный термин точно отражает основные характеристики систем, разработанных в этот период. Большинство моделей, разработанных в 1940-х, 50-х и 60-х годах принимали на вход булевы значения (true/false, представленные как +1/-1 или 1/0) и генерировали булев вывод. Они также использовали пороговые функции активации (см. главу 3), а их сети состояли из одного слоя; иными словами, они были ограничены единственной матрицей с корректируемыми весами. В этих ранних исследованиях основное внимание часто уделялось тому, способны ли вычислительные модели на основе искусственных нейронов формировать логические отношения вроде конъюнкции и дизъюнкции.

В 1943 году Уолтер Мак-Каллок и Уолтер Питтс опубликовали важную вычислительную модель биологических нейронов в своей работе под названием «A Logical Calculus of the Ideas I manent in Nervous Activity» (Мак-Каллок и Питтс, 1943). В данном исследовании подчеркивалась двоичная природа неврологической активности в головном мозге и математически описывалась эта активность применительно к логике высказываний. В модели Мак-Каллока и Питтса все входные и выходные значения нейронов были равны либо 0, либо 1. Более того, каждый ввод был либо возбуждающим (с весом +1), либо понижающим (с весом –1). Основным принципом этой модели стало сложение входных значений с последующим применением к ним пороговой функции. В ходе сложения повышающий ввод прибавлял 1, а понижающий вычитал 1. Если полученный результат превышал заданное пороговое значение, вывод нейрона равнялся 1; в противном случае возвращался 0. В данной научной работе Мак-Каллок и Питтс продемонстрировали, как с помощью этой простой модели можно представить логические операции (такие как конъюнкция, дизъюнкция и отрицание). Эта модель включала большинство элементов искусственного нейрона, описанных в главе 3. Однако в ней нейрон оставался неизменным; иными словами, веса и порог задавались вручную.

В 1949 году Дональд О. Хебб издал книгу под названием «The Organization of Behavior», в которой с помощью теории нейропсихологии (которая

объединяла исследования по психологии и физиологии головного мозга) объяснялись основы человеческого поведения. Основополагающей предпосылкой этой теории было то, что поведение является результатом действий и взаимодействий нейронов. С точки зрения исследования нейронных сетей важнейшей идеей этой книги был постулат (позже получивший название *постулат обучения Хебба*), который объяснял существование долгосрочных воспоминаний у животных изменениями в соединениях между нейронами:

> Когда сигнал аксона клетки А достаточно близок к тому, чтобы возбудить клетку В и неоднократно или постоянно принимает участие в ее активации, в одной или обеих клетках происходит некий процесс роста, или метаболическое изменение, повышающее эффективность клетки А в активации клетки В (Хебб 1949, стр. 62).

Этот постулат был важным, так как утверждал, что в соединениях между нейронами хранилась информация (то есть веса сети) и что в ходе изменения этих соединений с помощью повторяющихся последовательностей активации происходило обучение (иными словами, сеть, в которой меняются веса, способна обучаться).

Правило Розенблатта об обучении перцептрона

В течение нескольких лет после выхода книги Хебба рядом исследователей были предложены вычислительные модели поведения нейрона, сочетавшие в себе булевы модули пороговой активации Мак-Каллока и Питтса с механизмом обучения на основе коррекции весов, применяемых к входным значениям. Самой известной из них была модель перцептрона Фрэнка Розенблатта (1958). На концептуальном уровне модель перцептрона можно представить в виде нейронной сети, состоящей из единственного искусственного нейрона с модулем пороговой активации. Важно отметить, что у такой сети есть всего один слой весов. Впервые перцептрон был реализован в системе IBM 704 (и это, вероятно, была первая реализация нейронной сети как таковой). Однако Розенблатт всегда считал, что перцептрон должен стать физическим устройством, и позже эта идея была воплощена в специализированном оборудовании под названием «перцептрон Mark 1». Mark 1 принимал ввод с фотокамеры, генерировавшей 400-пиксельное изображение; это делалось с помощью 400 фотоэлементов, которые, в свою очередь, были соединены

с нейронами. Веса соединений были реализованы с помощью регулируемых электрических резисторов, известных как потенциометры; регуляция весов осуществлялась посредством электромоторов, подключенных к потенциометрам.

Розенблатт предложил процедуру обучения с коррекцией ошибок для обновления весов перцептрона, чтобы он мог научиться различать два класса входных значений: для одного должен генерироваться вывод $y = +1$, а для другого — $y = -1$ (Розенблатт, 1960). Процедура обучения требует наличия набора входных закономерностей, закодированных в булевом виде, для каждой из которых предоставлен ожидаемый вывод. В начале обучения веса перцептрона инициализируются с помощью целевых значений. Затем циклически перебираются учебные образцы, в ходе чего веса сети обновляются в зависимости от разницы между сгенерированным выводом перцептрона и целевым выводом, указанным в наборе данных. Учебные образцы можно передавать сети в любом порядке и по нескольку раз. Полный проход сети по всем образцам называется итерацией; обучение завершается, когда за одну итерацию перцептрон правильно классифицирует все образцы.

Для обновления каждого веса в перцептроне после обработки образца Розенблатт сформулировал правило обучения (известное как правило обучения перцептрона). В нем используется тот же принцип, что и в стратегии с тремя условиями для оптимизации весов в модели принятия решений по кредитам, представленной в главе 2.

- Если вывод модели для заданного образца совпадает с соответствующим выводом, указанным в наборе данных, веса не обновляются.
- Если вывод модели для текущего образца недостаточно большой, он увеличивается путем увеличения весов положительных входных значений и уменьшения весов отрицательных входных значений для данного образца.
- Если вывод модели для текущего образца слишком большой, он уменьшается путем уменьшения весов положительных входных значений и увеличения весов отрицательных входных значений для данного образца.

Процесс обновления веса i (w_i) в соответствии с правилом обучения Розенблатта, записанный в виде уравнения, выглядит так:

$$w_i^{t+1} = w_i^t + \left(\eta \times \left(y^t - \hat{y}^t\right) \times x_i^t\right)$$

В этом правиле w_i^{t+1} — это значение веса i после обновления весов сети в ответ на обработку образца t; w_i^t — это значение веса i, которое используется в ходе обработки образца t; η — это положительная константа, заданная вручную (известная как коэффициент скорости обучения; подробней о ней чуть ниже); y^t — целевой вывод для образца t, заданный в учебном наборе данных; y^t — вывод, сгенерированный перцептроном для образца t, и x_i^t — элемент ввода t, взвешенный с помощью w_i^t в ходе обработки образца.

Этот процесс может показаться сложным, но на самом деле правило обучения перцептрона является всего лишь математическим представлением стратегии обновления весов с тремя условиями, описанной выше. Основное внимание следует обратить на вычисление разницы между ожидаемым вводом и тем, что перцептрон в действительности предсказал: $y^t - y^t$. Результат вычитания говорит нам о том, какое из трех условий мы имеем. Чтобы понять, как работает эта операция, следует помнить, что в модели перцептрона есть только два желаемых вывода: $y = +1$ и $y = -1$. Первое условие выполняется, когда $y^t - y^t = 0$; в этом случае вывод перцептрона правильный, и веса не меняются.

Второе условие обновления весов выполняется в случае, когда вывод перцептрона слишком большой. Это возможно только тогда, когда правильный вывод для образца t равен $y^t = -1$, то есть если $y^t - y^t < 0$. В этой ситуации, когда вывод перцептрона для образца t равен $y^t = +1$, погрешность будет отрицательной ($y^t - y^t = -2$) и вес w_i увеличивается на $+(\eta \times -2 \times x_i^t)$. Если предположить, что в нашем примере η равно 0,5, обновление весов упрощается до $-x_i^t$. Иными словами, если вывод перцептрона слишком большой, правило обновления весов вычитает из каждого веса соответствующее входное значение. Это уменьшит веса для положительных входных значений и увеличит их для отрицательного ввода в данном примере (вычитание отрицательного числа — то же самое, что прибавление положительного).

Третье условие обновление весов — когда вывод перцептрона недостаточно большой. Оно прямо противоположно второму условию и может возникнуть, только если равен $y^t = +1$ — то есть когда $y^t - y^t > 0$. В данном случае ($y^t - y^t = 2$), поэтому вес увеличивается на $+(\eta \times 2 \times x_i^t)$. Если же предположить, что η равно 0,5, обновление весов упрощается до $+x_i^t$. Это говорит о том, что данное правило обновляет веса, прибавляя к ним входные значения. В результате, если ввод отрицательный, вес уменьшается, а если положительный — увеличивается.

В предыдущих абзацах неоднократно упоминается коэффициент скорости обучения η. Он нужен для того, чтобы управлять величиной изменений,

которые вносятся в веса. Это пример гиперпараметра, который задается вручную еще до обучения модели. У него есть один недостаток.

- Если коэффициент скорости обучения слишком маленький, процессу обучения может понадобиться очень много времени для того, чтобы было достигнуто подходящее сочетание весов.
- Если коэффициент скорости обучения слишком большой, веса могут быть так сильно разбросаны по пространству весов, что обучение может вообще не дать результата.

В качестве коэффициента скорости обучения можно выбрать либо относительно небольшое положительное число (например, 0, 01), либо более крупное значение (такое как 1,0). Во втором случае его следует симметричным образом уменьшать по ходу обучения:

$$\eta^{t+1} = \eta^1 \times \frac{1}{t}$$

Для наглядности приведем пример. Представьте, что вы пытаетесь решить головоломку, в которой нужно загнать шарик в отверстие. Вы можете контролировать направление и скорость шарика за счет наклона поверхности, по которой он катится. Если наклонить поверхность слишком сильно, шарик начнет двигаться с большой скоростью и, скорее всего, пройдет мимо отверстия. В этом случае вам опять придется менять наклон поверхности, и если это изменение окажется слишком резким, вы будете повторять процесс снова и снова. С другой стороны, если наклонить поверхность чуть-чуть, шарик может вообще остаться на месте или катиться очень медленно, и весь процесс затянется. Такая задача во многом похожа на поиск лучшего сочетания весов для сети. Представьте, что каждая точка на поверхности, по которой перекатывается шарик, — это потенциальный набор весов. Положение шарика в любой момент времени определяет текущую комбинацию весов в сети. Положение отверстия — это оптимальное сочетание весов для задачи, решению которой мы обучаем нашу сеть. Приведение сети к оптимальному сочетанию весов аналогично закатыванию шарика в отверстие. Коэффициент скорости обучения позволяет определить, насколько быстро мы двигаемся по поверхности в ходе поиска подходящей комбинации. Если сделать его слишком большим, перемещение по поверхности будет быстрым: на каждой итерации веса смогут существенно меняться, то есть, используя

нашу аналогию, шарик будет катиться очень быстро и может миновать отверстие. Точно так же наш процесс поиска может оказаться настолько стремительным, что пропустит оптимальный набор весов. С другой стороны, если коэффициент скорости обучения небольшой, перемещения по поверхности будут очень медленными: на каждой итерации допускаются лишь незначительные обновления; иными словами, шарик будет катиться очень неспешно. Это уменьшает вероятность того, что мы пропустим оптимальное решение, но для его получения может понадобиться слишком много времени. Стратегия, согласно которой коэффициент скорости обучения сначала большой, но затем симметрично уменьшается, подобна тому, как если бы мы резко наклонили поверхность, чтобы придать шарику движение, но затем уменьшили наклон, чтобы лучше контролировать перемещение шарика по мере его приближения к отверстию.

Розенблатт доказал, что алгоритм обучения перцептрона в конечном итоге сойдется на весах, которые позволят ему правильно классифицировать все образцы (если эти веса существуют). Это открытие известно как теорема о сходимости перцептрона (Розенблатт, 1962). Однако сложность обучения заключается в том, что может потребоваться большое количество итераций, прежде чем алгоритм сойдется. Более того, во многих задачах неизвестно заранее, существует ли подходящая комбинация весов; следовательно, если обучение продолжается на протяжении долгого времени, нельзя с уверенностью сказать, завершится оно когда-либо или нет.

Алгоритм минимального среднеквадратичного отклонения

Примерно в то же время, когда Розенблатт работал над перцептроном, Бернард Уидроу и Маркиан Хофф разрабатывали очень похожую модель под названием ADALINE (сокращенно от *adaptive linear neuron* — адаптивный линейный нейрон), а также правило обучения, известное как алгоритм минимального среднеквадратичного отклонения (англ. least mean square, или LMS; Уидроу и Хофф, 1960). Сеть ADALINE состоит из одного нейрона, который сильно напоминает перцептрон; единственное отличие состоит в том, что эта сеть не использует пороговую функцию. На самом деле ее вывод представляет собой взвешенную сумму входных значений. Вот почему этот нейрон называют линейным: взвешенная сумма — это линейная функция (она определяет прямую). Таким образом, ADALINE реализует линейное отношение между вводом и выводом. Правило LMS почти идентично правилу обучения

перцептрона, только вместо вывода для заданного образца y^t используется взвешенная сумма входных значений:

$$w_i^{t+1} = w_i^t + \left(\eta \times \left(y^t - \left(\sum_{i=0}^{n} w_i^t \times x_i^t \right) \right) \times x_i^t \right)$$

Логика обновления по правилу LMS та же, что и в правиле обучения перцептрона. Если вывод слишком большой, веса, применяемые к положительному вводу, делают вывод больше, поэтому при следующей подаче на вход их нужно уменьшить, а веса, применяемые к отрицательному вводу, нужно увеличить. Следуя той же логике, если вывод слишком маленький, веса, применяемые к положительному вводу, нужно увеличить, а те, которые применяются к отрицательному вводу, — уменьшить.

Один из важных аспектов исследований Уидроу и Хоффа заключался в демонстрации того, что с помощью правила LMS сеть можно научить предсказывать любое число, а не только +1 или –1. Это правило обучения было названо алгоритмом минимального среднеквадратичного отклонения, так как использование LMS для постепенной коррекции весов нейрона эквивалентно минимизации среднеквадратичного отклонения для учебного набора. В наши дни LMS иногда называют правилом обучения Уидроу — Хоффа, в честь его создателей; но более широко оно известно как дельта-правило, так как для вычисления величины, на которую изменяются веса, оно применяет разницу (дельту) между желаемым и фактическим выводом. Иными словами, если следовать правилу LMS, вес нужно корректировать пропорционально разнице между выводом сети ADALINE и ожидаемым результатом. Если нейрон отклоняется слишком сильно, величина коррекции весов будет большой, а если нет, то маленькой.

В настоящее время перцептрон считается важным прорывом в развитии машинного обучения, так как он стал первой нейронной сетью, которая когда-либо была реализована. Тем не менее большинство современных алгоритмов для обучения нейронных сетей больше похожи на LMS. Алгоритм LMS пытается минимизировать среднеквадратичное отклонение сети. Как будет отмечено в главе 6, этот процесс постепенного уменьшения погрешности формально включает в себя градиентный спуск на поверхность отклонения; и в наши дни почти все нейронные сети обучаются с использованием какой-то разновидности градиентного спуска.

Проблема исключающего ИЛИ

Розенблатт, Уидроу, Хофф и другие успешно продемонстрировали, что модели нейронных сетей способны автоматически учиться различать наборы закономерностей. Это привлекло большое внимание к исследованиям в области искусственного интеллекта и нейронных сетей. Однако уже в 1969 году Марвин Мински и Сеймур Пейперт издали книгу под названием «Perceptrons», которая, как оказалось впоследствии, одним махом разрушила воодушевление и оптимизм, свойственные ранним исследованиям (Мински и Пейперт, 1969). Стоит отметить, что в 1960-х годах вокруг нейронных сетей было много необоснованной шумихи, и исследователям так и не удалось удовлетворить завышенные ожидания публики. Тем не менее книга Мински и Пейперта сформировала крайне отрицательный взгляд на нейронные сети среди власть имущих, и после ее выхода финансирование исследований в этой области было урезано.

В книге Мински и Пейперта в основном рассматривались однослойные перцептроны. Как вы помните, однослойный перцептрон — то же самое, что один нейрон, который использует пороговую функцию активации, поэтому он способен реализовывать лишь линейную границу решений (в виде прямой)*. Это означает, что однослойный перцептрон может различать два класса входных значений, только если мы можем провести такую прямую через пространство ввода, по одну сторону которой находятся все образцы первого класса, а по другую — второго. Мински и Пейперт выделили это ограничение как недостаток подобных моделей.

Чтобы понять это критическое отношение к однослойным перцептронам, необходимо сначала разобраться с понятием линейно разделимой функции. Чтобы ее проиллюстрировать, проведем сравнение между функциями логического И и ИЛИ. Функция И принимает два входных значения, каждое из которых может быть либо истинным (TRUE), либо ложным (FALSE), и возвращает TRUE, только если оба значения равны TRUE. На графике, представленном в левой части рис. 4.2, изображено пространство ввода этой функции, где каждая из четырех возможных комбинаций приводит либо к истинному выводу (обозначенному пустым кружком), либо к ложному (обозначенному черными кружками). Этот график демонстрирует, что мы можем провести прямую между входными значениями, для которых функция И возвращает

* На рис. 3.6 и 3.7 показана линейная граница решений нейрона (прямая линия), который использует пороговую функцию активации.

TRUE, (T, T), и значениями, для которых возвращается FALSE, {(F, F), (F, T), (T, F)}. Функция ИЛИ работает похожим образом, только она возвращает TRUE, если хотя бы одно из входных значений равно TRUE. На среднем графике, показанном на рис. 4.2, видно, что мы можем провести прямую, разделяющую ввод, который функция ИЛИ классифицирует как TRUE, {(F, T), (T, F), (T, T)}, и ввод, который классифицируется как FALSE, (F, F). Именно потому, что можно провести в пространстве ввода этой функции одну прямую линию, которая делит все входные значения на две категории, функции И и ИЛИ являются линейно разделимыми.

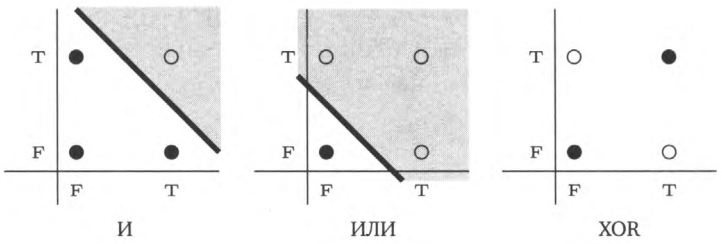


Рис. 4.2. Примеры линейно разделимых функций. На каждом графике черные кружки обозначают ввод, для которого функция возвращает FALSE, а входные значения, для которых возвращается TRUE, помечены пустыми кружками (T означает TRUE, а F — FALSE)

Функция «исключающее ИЛИ» (известная также как XOR), имеет похожую структуру; однако она возвращает TRUE, только если одно из входных значений равно TRUE (но не оба сразу). График в правой части рис. 4.2 показывает пространство ввода функции XOR с четырьмя возможными комбинациями ввода, для каждой из которых возвращается либо TRUE (помечены пустыми кружками), либо FALSE (черные кружки). Как видите, мы не можем провести прямую между двумя классами входных значений функции XOR, истинными и ложными. Из-за отсутствия прямой линии, которой можно было бы разделить две категории ввода, функцию XOR называют нелинейно разделимой. Это вовсе не делает ее уникальной или редкой — таких функций довольно много.

Основным недостатком однослойных перцептронов Мински и Пейперт считали то, что такие однослойные модели неспособны формировать нелинейно разделимые функции наподобие XOR. Причиной этого ограничения является тот факт, что перцептрон имеет линейную границу решений, и его единственный слой не способен научиться различать входные значения, принадлежащие к разным категориям нелинейно разделимой функции.

На момент выхода книги Мински и Пейперта уже было известно о возможности создания нейронных сетей, которые определяют нелинейные границы решений и, следовательно, способны формировать нелинейно разделимые функции (такие как XOR). Ключевой задачей при создании сетей с такими сложными (нелинейными) границами решений было увеличение количества слоев нейронов. Например, на рис. 4.3 показана двухслойная сеть, реализующая функцию XOR. Логические значения TRUE и FALSE представлены в ней в виде чисел: FALSE обозначено как 0, а TRUE — как 1. Активация модулей (когда вывод равен +1) происходит в случае, если взвешенная сумма входных значений ≥ 1; иначе возвращается 0. Обратите внимание на то, что модули в скрытом слое реализуют логические функции И и ИЛИ. Их можно считать промежуточными этапами в решении задачи исключающего ИЛИ. Модуль в выходном слое реализует XOR, компонуя вывод этих скрытых слоев. Иными словами, модуль в выходном слое возвращает TRUE, только если узел И пассивный (вывод = 0), а узел ИЛИ активный (вывод = 1). Однако в то время не было ясного представления о том, как обучать сети с несколькими слоями. К тому же в конце своей книги Мински и Пейперт отметили, что, по их мнению, исследования в области расширения нейронных сетей до нескольких слоев были «бесплодными» (Мински и Пейперт, 1969, разд. 13.2, стр. 23).

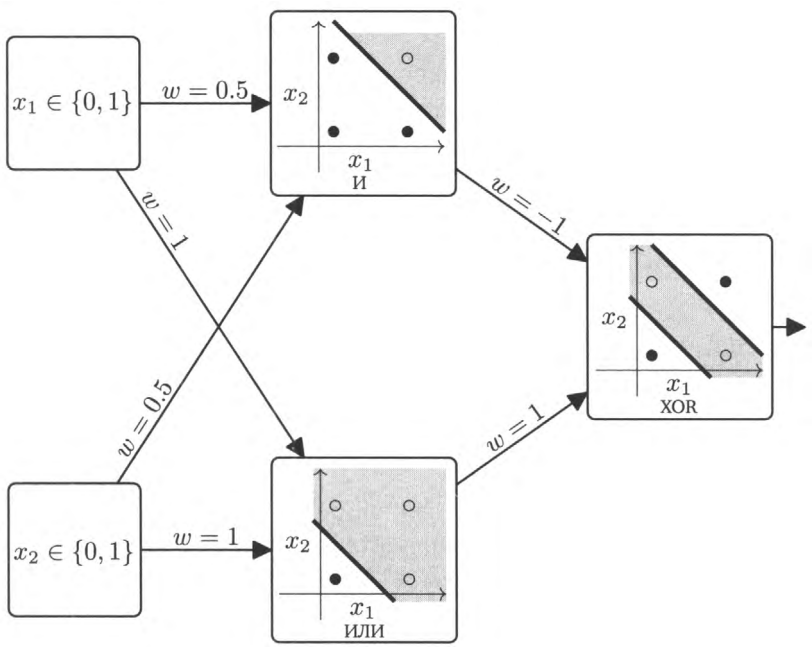


Рис. 4.3. Сеть, реализующая функцию XOR. Все вычислительные модули используют пороговую функцию активации с порогом ≥ 1

По иронии судьбы одновременно с выходом книги Мински и Пейперта украинский исследователь Алексей Ивахненко предложил метод группового учета аргументов (МГУА), а в 1971 году опубликовал научную работу, в которой описывалось, как с помощью этого метода можно обучать нейронные сети с восемью слоями (Ивахненко, 1971). В наши дни сеть МГУА считается первым опубликованным примером глубокой нейронной сети, обучаемой на данных (Шмидхубер, 2015). Однако на протяжении многих лет достижение Ивахненко в основном игнорировалось сообществом исследователей нейронных сетей. В результате в современном глубоком обучении метод МГУА используется очень редко: за прошедшие годы сообщество приняло в качестве стандарта другие обучающие алгоритмы, такие как обратное распространение (описанное ниже). Таким образом, проигнорированное открытие Ивахненко и убедительная критика Мински и Пейперта ознаменовали конец первого периода серьезного интереса к исследованию нейронных сетей.

Тем не менее наследие, оставленное исследователями той поры, определило направление развития этой области вплоть до наших дней. Была обозначена принципиальная внутренняя структура искусственного нейрона: взвешенная сумма входных значений, передаваемая в функцию активации. Возникла идея хранения информации в виде весов сети. Более того, были предложены обучающие алгоритмы на основе итеративной коррекции весов, а также правила обучения, такие как LMS. В частности, подход с изменением весов нейронной сети пропорционально разнице между фактическим и желаемым выводом, примененный в LMS, используется в большинстве современных обучающих алгоритмов. Наконец, были распознаны ограничения, свойственные однослойным сетям, и появилось понимание того, что для их обхода сеть можно расширить за счет дополнительных слоев нейронов. Однако на тот момент было неясно, как обучать такие сети. Чтобы обновлять веса, мы должны знать, как они сказываются на погрешности итогового вывода. Например, согласно правилу LMS, причиной слишком большого вывода нейрона считаются веса, применяемые к положительным входным значениям. Следовательно, если их уменьшить, уменьшится и вывод, а вместе с ним и погрешность. Но в конце 1960-х годов вопрос о том, как моделировать отношение между весами ввода нейронов в скрытых слоях и общим отклонением сети, все еще оставалось без ответа, а не понимая, как веса влияют на это отклонение, их невозможно оптимизировать в скрытых слоях. Проблему сопоставления величины отклонения с элементами сети иногда называют задачей присваивания коэффициентов доверия или задачей определения ответственности за конечный результат.

Коннекционизм: многослойные перцептроны

В 1980-х годах началось переосмысление критики конца 1960-х: ее сочли излишне суровой. В частности, интерес к этой области науки возобновился благодаря двум прорывам: сетям Хопфилда и алгоритму обратного распространения.

В 1982 году Джон Хопфилд опубликовал научную работу с описанием сети, которая может себя вести как ассоциативная память (Хопфилд, 1982). Во время обучения ассоциативная память усваивает набор входных закономерностей. Как следствие, если сети предоставить искаженную версию ввода, который ей показывали раньше, она способна сгенерировать полноценную правильную закономерность. Ассоциативная память находит целый ряд применений, включая дополнение образов и коррекцию ошибок. В табл. 4.1* проиллюстрированы примеры решения этих двух видов задач с использованием ассоциативной памяти, обученной хранить информацию о днях рождения. В сети Хопфилда воспоминания (или входные закономерности) закодированы в виде двоичных строк, и если эти двоичные закономерности имеют значительные отличия, данная сеть способна сохранить до 0,138N строк, где N — количество нейронов в сети. Таким образом, чтобы сохранить 10 отдельных закономерностей, нужна сеть Хопфилда с 73 нейронами, а для 14 закономерностей понадобится 100 нейронов.

Таблица 4.1. Демонстрация применения ассоциативной памяти для дополнения образов и коррекции ошибок

Учебные закономерности		**Дополнение образов**	
`John**12May`	`Liz***?????`	→	`Liz***25Feb`
`Kerry*03Jan`	`??????10Mar`	→	`Des***10Mar`
`Liz***25Feb`		**Коррекция ошибок**	
`Des***10Mar`	`Kerry*01Apr`	→	`Kerry*03Jan`
`Josef*13Dec`	`Jxsuf*13Dec`	→	`Josef*13Dec`

* Эта иллюстрация использования ассоциативной памяти для дополнения образов и коррекции ошибок основана на примере из главы 42 в работе: Маккей, 2003.

Обратное распространение и исчезающие градиенты

В 1986 году группа исследователей в области параллельной распределенной обработки (англ. parallel distributed processing, или PDP) издала двухтомник с обзором текущего положения дел в области нейронных сетей (Румельхарт и др., 1986b, 1986c). Эти книги стали невероятно популярными, а в главе 8 первого тома описывался алгоритм обратного распространения (Румельхарт и др., 1986a). Идея этого алгоритма приходила в голову разным людям*, но именно эта глава, написанная Румельхартом, Хинтоном и Уильямсом и изданная группой PDP, популяризовала его практическое применение. Алгоритм обратного распространения решает задачу присваивания коэффициентов доверия, поэтому его можно использовать для обучения нейронных сетей со скрытыми слоями. Это, наверное, самый важный алгоритм в глубоком обучении. Однако для его полного и четкого объяснения необходимо сначала рассмотреть концепцию градиента ошибок и затем алгоритм градиентного спуска. В связи с этим мы отложим подробное знакомство с обратным распространением до главы 6, которая начинается с объяснения этих ключевых понятий. Тем не менее общую структуру этого алгоритма можно описать довольно быстро. Вначале алгоритм обратного распространения назначает произвольные веса каждому соединению в сети. Затем он итеративно подает на вход сети учебные образцы и обновляет ее веса до тех пор, пока она не начнет демонстрировать нужное поведение. Основная часть этого процесса состоит из двух этапов. На первом этапе (известном как прямая передача) сеть получает ввод, и результатам активации нейронов позволено перемещаться вперед, пока не будет сгенерирован итоговый вывод. Второй этап (известный как обратная передача) начинается в выходном слое и проходит по сети, пока не достигнет ее начала. В начале этого обратного движения для каждого нейрона в выходном слое вычисляется отклонение, которое затем используется для обновления весов этих нейронов. Дальше отклонение каждого выходного нейрона передается (обратно) нейрону из соседнего скрытого слоя пропорционально весам соединений между ними. После этого обмена информацией (или определения ответственности) общая ответственность скрытого нейрона суммируется и используется для обновления его весов. Это обратное

---

* Например, кандидатскую диссертацию Пола Вербоса (1974) принято считать первым опубликованным описанием обратного распространения ошибок в процессе обучения искусственной нейронной сети (Вербос, 1974).

распространение повторяется для всех нейронов, ответственность которых еще не была определена, пока не обновятся все веса в сети.

Ключевой инновацией, которая сделала возможной работу алгоритма обратного распространения, стало изменение в функции активации нейронов. Нейронные сети, которые разрабатывались на ранних этапах исследований в этой области, использовали пороговые функции активации. Алгоритм обратного распространения этого не делает, поскольку для обратной передачи информации функция активации нейронов должна быть дифференцируемой. Пороговую функцию нельзя дифференцировать, так как порог создает разрыв в ее выводе. Иными словами, наклон порога является бесконечно крутым, поэтому в его точке невозможно вычислить градиент функции. Это привело к использованию в многослойных нейронных сетях дифференцируемых функций активации, таких как tanh и логистическое уравнение.

Однако обучению глубоких сетей с помощью алгоритма обратного распространения присуще одно важнейшее ограничение. В 1980-х годах исследователи обнаружили, что обратное распространение хорошо себя показывало в относительно «мелких» сетях (с одним-двумя скрытыми слоями), но с увеличением глубины происходило одно из двух: либо процесс обучения затягивался слишком сильно, либо сети и вовсе не удавалось сойтись на хорошем сочетании весов. В 1991 году Сепп Хохрайтер (совместно с Юргеном Шмидхубером) описал причину возникновения этой проблемы в своей дипломной работе (Хохрайтер, 1991). Она была связана с тем, каким образом распространяются отклонения. По существу алгоритм обратного распространения является реализацией дифференцирования сложной функции. Это подразумевает умножение членов, поэтому отклонение, передающееся от одного нейрона к другому в обратном направлении, может умножаться на числовые члены, значения которых меньше 1. Это происходит многократно по ходу того, как отклонение распространяется обратно по сети. В результате его сигнал становится все меньше и меньше. Часто это затухание оказывается экспоненциальным относительно расстояния от выходного слоя. Таким образом, на протяжении каждой итерации веса, находящиеся ближе ко входу глубокой сети, корректируются совсем незначительно (или вообще не меняются). Иными словами, начальные слои либо обучаются крайне медленно, либо совсем не отходят от своих начальных (произвольных) значений. При этом они играют главную роль в успехе сети, так как именно их нейроны учатся распознавать признаки, которые в последующих слоях становятся основными составными элементами представлений, которые определяют итоговый вывод сети. По техническим причинам, которые мы объясним в главе 6, сигнал

отклонения, который проходит по сети в обратном направлении, фактически является градиентом отклонения сети, поэтому его стремительное затухание называют проблемой исчезающего градиента.

Коннекционизм: локальное и распределенное представления

Несмотря на проблему исчезающего градиента, алгоритм обратного распространения сделал возможным обучение нейронных сетей с более сложными (глубокими) архитектурами. Это соответствует принципу коннекционизма, согласно которому в результате взаимодействия большого количества простых вычислительных элементов может возникнуть разумное поведение. Еще одним аспектом коннекционизма была идея распределенного представления. Методы представления, применяемые в нейронных сетях, можно разделить на локальные и распределенные. В локальном представлении каждый нейрон связан с отдельной концепцией, а в распределенном каждая концепция представлена сочетанием активаций в наборе нейронов. Следовательно, во втором случае каждая концепция зависит от вывода нескольких нейронов, а вывод каждого нейрона влияет на представление нескольких концепций.

Чтобы проиллюстрировать отличия между этими двумя подходами, рассмотрим ситуацию, в которой (по какой-то неизвестной причине) для представления отсутствия или наличия разных кулинарных блюд используется сочетание активаций нескольких нейронов. Более того, у каждого блюда есть два свойства: страна происхождения рецепта и вкус. Доступны три страны: *Италия*, *Мексика* и *Франция*, а блюдо может быть *сладким, кислым* или *горьким*. Поэтому в целом мы имеем девять возможных типов еды: *итальянская+сладкая*, *итальянская+кислая*, *итальянская+горькая*, *мексиканская+сладкая* и т. д. При использовании локального представления потребовалось бы девять нейронов, по одному для каждого блюда. Однако существует ряд методов, которые позволяют представить эту предметную область распределенным образом. Например, мы можем назначить каждой комбинации двоичное число. В этом случае нам хватит всего четырех нейронов: сигнал 0000 обозначает *итальянскую+сладкую* еду, 0001 обозначает *итальянскую+кислую*, 0010 — *итальянскую+горькую* и т. д., вплоть до числа 1000, представляющего сочетание *французская+горькая*. Это очень компактное представление. Но обратите внимание на то, что вывод отдельного нейрона сам по себе не поддается интерпретации: первый справа нейрон будет активным (***1) для сочетаний *итальянская+кислая*, *мексиканская+сладкая*,

В распределенном представлении каждая концепция зависит от вывода нескольких нейронов, а вывод каждого нейрона влияет на представление нескольких концепций.

мексиканская+горькая и *французская+кислая*, и если не знать вывода остальных нейронов, невозможно понять, какую страну или какой вкус обозначает этот сигнал. Однако в глубокой сети отсутствие семантической интерпретируемости вывода скрытых модулей не является проблемой — главное, чтобы нейроны в выходном слое сети могли объединить эти представления в правильный итоговый вывод. Эту кулинарную предметную область можно представить и другим, более понятным образом. Для обозначения стран и вкусов достаточно использовать по три нейрона. В таком случае сочетание активаций 100100 будет представлять *итальянскую+сладкую* еду, 001100 — *французскую+сладкую*, а 001001 — *французскую+горькую*. В этом представлении вывод каждого нейрона можно интерпретировать по отдельности; но, чтобы получить полное обозначение еды (страна+вкус), требуется определенное распределение активаций по набору нейронов. Заметьте, что оба эти представления оказались более компактными, чем локальное. Это позволяет существенно уменьшить количество весов, которые нужны сети, что, в свою очередь, сокращает время ее обучения.

Концепция распределенного представления играет очень важную роль в глубоком обучении. Существует вполне справедливый аргумент в пользу того, что глубокое обучение правильней называть обучением представлениям; он состоит в том, что нейроны в скрытых слоях сети усваивают распределенные представления ввода, являющиеся полезными промежуточными представлениями в отношениях между вводом и выводом, которые сеть пытается установить. С этой точки зрения задачей выходного слоя является формирование таких промежуточных представлений, которые могли бы сгенерировать желаемый вывод. Давайте еще раз обратимся к сети на рис. 4.3, которая реализует функцию XOR. Скрытые модули этой сети усваивают промежуточное представление ввода, которое можно считать сочетанием функций И и ИЛИ; затем выходной слой объединяет это промежуточное представление, чтобы сгенерировать нужный вывод. Можно считать, что в глубокой сети каждый следующий слой усваивает представление, которое является абстракцией по отношению к выводу предыдущего слоя. Именно это последовательное абстрагирование, посредством усваивания промежуточных представлений, позволяет глубоким сетям обучаться таким сложным отношениям между вводом и выводом.

Сетевые архитектуры: сверточные и рекуррентные нейронные сети

Нейроны можно соединять разными способами. На примерах сетей, с которыми вы уже познакомились в этой книге, можно составить представление об относительно простых соединениях: нейроны группируются в слои, при этом каждый нейрон одного слоя соединяется со всеми нейронами следующего. Это так называемые сети с прямой связью. Все их соединения проходят прямо от ввода к выводу, без каких-либо замыканий (циклов). Более того, все примеры сетей, представленные до сих пор, можно считать полносвязными, так как каждый нейрон соединяется со всеми нейронами следующего слоя. В некоторых ситуациях возможно (а иногда и полезно) проектировать и обучать те сети, которые не являются полносвязными, и те, у которых нет прямой связи. Оптимизацию сетевой архитектуры, если проводить ее как следует, можно свести к внедрению в сеть информации о свойствах задачи, которую эта сеть учится моделировать.

В качестве чрезвычайно успешного примера внедрения знаний предметной области в сеть для оптимизации ее архитектуры можно привести сверточные нейронные сети (англ. convolutional neural networks, или CNN), предназначенные для распознавания объектов на изображениях. В 1960-х годах Хьюбел и Визель провели ряд экспериментов со зрительной корой головного мозга кошек (Хьюбел и Визель, 1962, 1965). Они вставляли электроды в кошачий мозг (под наркозом) и изучали реакцию мозговых клеток на различные визуальные раздражители. В качестве раздражителей использовались, например, яркие пятна или линии света, возникающие в определенном месте поля зрения или перемещающиеся по некоторой его области. Оказалось, что на разные раздражители в разных местах поля зрения реагировали разные клетки; то есть каждая клетка зрительной коры была рассчитана на определенный тип визуального раздражителя и связана с определенной областью поля зрения. Эта область называется рецептивным полем клетки. Еще одним результатом этих экспериментов стало разделение клеток на два типа — простые и сложные. Для простых клеток местоположение раздражителя играет ключевую роль: малейший его сдвиг вызывает существенное ослабление реакции клетки. А вот ответ сложных клеток на раздражитель не зависит от того, в каком месте поля зрения он возникает. Хьюбел и Визель (1965) высказали предположение о том, что сложные клетки ведут себя так, будто им поступают проекции от большого количества простых клеток, каждая из которых отвечает на тот же зрительный раздражитель, но с учетом разного местоположения относительно своего рецептивного поля. Эта иерархия, в которой простые клетки передают сигналы

сложным, приводит к тому, что раздражитель, занимающий большую область поля зрения, проходит через простые клетки и в итоге фокусируется в одной сложной клетке. Этот эффект воронки проиллюстрирован на рис. 4.4. Здесь показан слой простых клеток, каждая из которых отслеживает рецептивное поле в отдельной области зрения. Сложная клетка, рецептивное поле которой охватывает этот слой, активируется в случае активации любой простой клетки в этом поле. Таким образом, сложная клетка способна реагировать на зрительный раздражитель, возникающий в любом участке поля зрения.

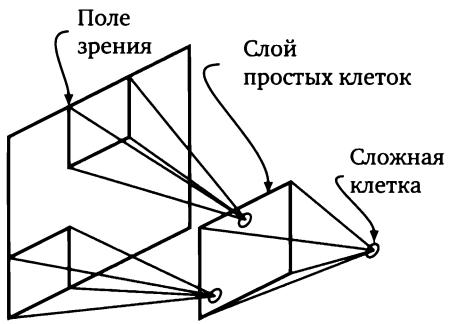


Рис. 4.4. Эффект воронки рецептивных полей, являющийся результатом иерархии простых и сложных клеток

В конце 1970-х и начале 1980-х Кунихико Фукусима, вдохновившись анализом зрительной коры в работах Хьюбела и Визеля, разработал архитектуру нейронных сетей для распознавания визуальных образов, получившую название неокогнитрон (Фукусима, 1980). Структура неокогнитрона основывалась на наблюдении, что сеть распознавания образов должна уметь определить наличие визуального признака независимо от того, в каком месте изображения он находится. Говоря более формально, сеть должна уметь выполнять пространственно-независимое обнаружение визуальных признаков. Например, сеть для распознавания лиц должна распознавать форму глаза независимо от того, в каком участке изображения он размещен, — аналогично тому, как иерархическая модель Хьюбела и Визеля могла определять наличие визуального признака, в каком бы участке поля зрения тот ни возникал.

Фукусима понял, что поведение простых клеток в иерархии Хьюбела и Визеля можно воспроизвести в нейронной сети, используя слой нейронов с одинаковыми наборами весов; при этом каждый нейрон должен получать входной сигнал из небольшой статической области (рецептивного поля), размещенной в определенном участке поля ввода. Чтобы понять, какое отношение нейроны с одинаковыми весами имеют к пространственно-независимому обнаружению

признаков, представьте себе нейрон, принимающий на вход набор пиксельных значений, собранных на каком-то участке изображения. Веса, которые он применяет к этим пиксельным значениям, определяют функцию обнаружения визуальных признаков; эта функция возвращает true (сильная активация), если конкретный визуальный признак (сочетание пикселей) присутствует в его вводе, и false — если нет. Следовательно, если все нейроны используют одни и те же веса, каждый из них будет реализовывать одну и ту же функцию обнаружения. И если их совокупное рецептивное поле охватывает все изображение, то как минимум один из них сумеет распознать визуальный признак и активироваться независимо от того, где этот признак возник.

Фукусима также подметил, что эффект воронки Хьюбела и Визеля (сходящейся в сложных клетках) можно имитировать, если небольшие фиксированные группы нейронов будут передавать свой ввод следующим слоям сети. Таким образом, входные значения каждого нейрона в последнем слое будут охватывать все поле ввода, что позволит сети определить наличие визуального признака в любом месте поля зрения.

Некоторые веса в неокогнитроне устанавливались вручную, а другие генерировались с помощью процесса обучения без учителя. В ходе этого процесса при получении на вход каждого образца сеть выбирала из набора слоев, которые в прошлом демонстрировали для этого ввода большой вывод, отдельный слой с одинаковыми весами. Веса нейронов в этом слое обновлялись так, чтобы усилить их реакцию на это входное сочетание; веса нейронов из других слоев оставались неизменными. В 1989 году Ян Лекун разработал архитектуру сверточных нейронных сетей (англ. convolutional neural network, или CNN), предназначенную специально для обработки изображений (Лекун, 1989). Архитектура CNN имела много особенностей, присущих неокогнитрону, однако Лекун продемонстрировал, как сети этого типа можно обучать с помощью обратного распространения. Сети CNN прекрасно показали себя в обработке изображений и решении других задач. Самым знаменитым представителем этого семейства стала сеть AlexNet, которая в 2012 году победила в состязании ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) (Крижевски и др., 2012). Суть ILSVRC состоит в определении объектов на фотографии. Успех AlexNet вызвал большой энтузиазм по отношению к CNN, и с тех пор ряд других архитектур этого типа выигрывали данное состязание. CNN являются одной из самых популярных разновидностей нейронных сетей, и в главе 5 мы поговорим о них более подробно.

В качестве еще одного примера архитектуры, оптимизированной под характеристики определенной предметной области, можно выделить рекуррентные

нейронные сети (англ. recurrent neural networks, или RNN). RNN предназначены для обработки последовательных данных, таких как естественный язык. Они принимают входные значения по очереди и содержат всего один скрытый слой. Однако вывод каждого скрытого нейрона не только передается выходным нейронам, но и временно хранится в буфере, а затем подается на вход всем остальным скрытым нейронам при следующем вводе. В результате на каждой итерации каждый нейрон в скрытом слое принимает не только текущий ввод, но и вывод, сгенерированный скрытым слоем в ответ на предыдущие входные значения. Чтобы лучше понять, как это работает, вы можете обратиться к рис. 5.2, на котором изображена структура RNN и поток информации, проходящий через сеть. Этот рекуррентный цикл, в ходе которого вывод скрытого слоя для одного ввода подается на вход этому же слою при следующем вводе, снабжает RNN памятью, благодаря которой сеть способна обрабатывать каждый ввод в контексте входных данных, которые она уже обрабатывала*. RNN называют глубокими сетями, так как глубину этой эволюционирующей памяти можно считать эквивалентной длине входной последовательности.

Одним из ранних примеров RNN, получивших широкую известность, является сеть Элмана. В 1990 году Джеффри Элман опубликовал научную работу с описанием сети RNN, обученной предсказывать окончания простых высказываний из двух-трех слов (Элман, 1990). Эта модель обучалась на синтезированном наборе простых предложений, который был сгенерирован с использованием искусственной грамматики. Грамматика была основана на лексиконе из двадцати трех слов, каждое из которых принадлежало к одной лексической категории (например, *мужчина*=NOUN-HUM [сущ. человек], *женщина*=NOUN-HUM, *кушает*=VERB-EAT [глаг. кушать], *печенье*=NOUN-FOOD [сущ. еда] и т. д.). С помощью этого лексикона грамматика определяла пятнадцать шаблонов для генерации предложений (например, из NOUN-HUM+VERB-EAT+NOUN-FOOD получилось бы такое предложение: *мужчина кушает печенье*). После обучения модель была способна генерировать подходящие окончания предложений, такие как *женщина*+*кушает*+*?* = *печенье*. Более того, после запуска сеть могла формировать более длинные строки, состоящие из нескольких предложений. Для этого она использовала контекст,

---

* Сетевая архитектура Хопфилда, представленная в начале этого раздела, также предусматривает рекуррентные соединения (циклы обратной связи между нейронами). Но она устроена таким способом, который не позволяет обрабатывать последовательности. В связи с этим она не считается полноценной рекуррентной архитектурой.

сгенерированный ею в качестве ввода для следующего слова. Это можно проиллюстрировать на следующем примере с тремя предложениями:

девочка кушает хлеб собака двигает мышь мышь двигает книга

Эта задача по генерированию предложений применялась в очень простой предметной области, однако способность RNN формировать правдоподобные словосочетания была воспринята как свидетельство того, что нейронные сети могут моделировать лингвистические формы без знания четких грамматических правил. Впоследствии работа Элмана оказала огромное влияние на психолингвистику и психологию. Следующая цитата принадлежит Полу Черчлэнду (1996) и показывает, насколько важными некоторые считали исследования Элмана:

> Конечно, результатом работы этой сети является использование лишь малой части того обширного лексикона, которым обладает любой нормальный англоговорящий человек. Тем не менее результат есть результат, и рекуррентная сеть, по всей видимости, способна его выдавать. Поразительная демонстрация Элмана вряд ли разрешит разногласия между двумя разными подходами к грамматике — сетевым и основанным на правилах. На это потребуется какое-то время. Однако теперь стороны конфликта могут разговаривать на равных. И я не скрываю, какая из них мне видится фаворитом (Черчлэнд, 1996, стр. 143)*.

Сети RNN хорошо справляются с последовательными данными, но проблема исчезающего градиента оказывает сильное влияние на их работу. В 1997 году Сепп Хохрайтер и Юрген Шмидхубер (исследователи, которые в 1991 году представили объяснение проблемы исчезающих градиентов в RNN) предложили в качестве ее решения модули долгой краткосрочной памяти (англ. long short-term memory, или LSTM; Хохрайтер и Шмидхубер, 1997). Название этих модулей связано с отличием между тем, как нейронная сеть кодирует долгосрочную память (в виде понятий, которые усваиваются на протяжении какого-то периода времени) в ходе обучения, и краткосрочной памятью (под которой понимают немедленную реакцию системы на раздражители). В нейронной сети долгосрочная память кодируется путем коррекции весов, и после обучения эти веса больше не меняются. Для кодирования краткосрочной памяти используются сигналы, проходящие по сети,

---

* Впервые я встретил эту цитату Черчлэнда в Маркус, 2003 (стр. 25).

и значения этих сигналов быстро затухают. Модули LSTM спроектированы таким образом, чтобы краткосрочная память (сигналы активации) сети могла распространяться на протяжении длительных отрезков времени (или длинных входных последовательностей). Внутренняя структура LSTM относительно сложная, и мы опишем ее в главе 5. Но то, что эти модули могут распространять сигналы на протяжении длительного времени, позволяет им обрабатывать последовательности с удаленными зависимостями (взаимодействиями между элементами последовательности, разделенными двумя и больше позициями). Такова, например, зависимость между подлежащим и глаголом в английском предложении «the **dog/dogs** in that house **is/are** aggressive» (собака/собаки в том доме агрессивная/агрессивные). Благодаря этому LSTM-сети подходят для обработки естественного языка, и на протяжении многих лет они являются стандартной структурой многих лингвистических моделей, включая машинный перевод. Например, структура машинного перевода seq2seq (sequence-to-sequence — перевод отдельных предложений), представленная в 2014 году, объединяет две LSTM-сети в последовательность (Суцкевер и др., 2014). Первая сеть, кодировщик, последовательно обрабатывает входные значения и генерирует их распределенное представление. Ее называют кодировщиком, так как она кодирует последовательность слов в виде распределенного представления. Вторая сеть, декодер, принимает на вход распределенное представление ввода и учится генерировать выходную последовательность, элемент за элементом, используя цикл обратной связи, который возвращает последний сгенерированный элемент в качестве ввода для следующей итерации. На сегодня архитектура seq2seq служит основой для большинства современных систем машинного перевода, и подробней мы поговорим о ней в главе 5.

К концу 1990-х годов большинство концептуальных предпосылок для возникновения глубокого обучения уже существовали, в том числе алгоритмы для обучения многослойных сетей и сетевые архитектуры, которые до сих пор не утратили своей высокой популярности (CNN и RNN). Однако разработке глубоких сетей все еще препятствовала проблема исчезающих градиентов. К тому же надежды по поводу нейронных сетей, которые подогрел всплеск энтузиазма в 1990-х, так и не оправдались, что было плохим знаком с коммерческой точки зрения. В то же время прорывы в других сферах машинного обучения, таких как разработка метода опорных векторов (англ. support vector machines, или SVM), отвлекли внимание исследователей от нейронных сетей: на тот момент SVM демонстрировали аналогичную точность результатов, но при этом их было легче обучать. Совокупность этих факторов привела к спаду в исследовании нейронных сетей, который продолжался вплоть до появления глубокого обучения.

Эпоха глубокого обучения

Первое известное упоминание термина «глубокое обучение» приписывают Рине Дехтер (1986), хотя в ее научной работе он не имел отношения к нейронным сетям; в связи с нейронными сетями этот термин был впервые использован в работе Джоанны Айзенберг и др. (2000)*. В середине 2000-х годов интерес к нейронным сетям опять начал расти, и примерно в это же время мир услышал о методе глубокого обучения, который описывал глубокие нейронные сети. Это название подчеркивает, что у данных сетей намного больше слоев, чем у существовавших ранее.

Одним из первых успехов этой новой эпохи в исследовании нейронных сетей стал эксперимент Джефри Хинтона и его коллег, который продемонстрировал, что глубокую нейронную сеть можно обучать с помощью процесса, известного как жадное послойное предобучение. Этот процесс начинается с обучения одного слоя нейронов, который принимает на вход еще не обработанные данные. Данный отдельный слой нейронов можно обучать разными способами, но часто для этого используют автокодировщик. Автокодировщик — это нейронная сеть с тремя слоями: входным, скрытым (кодирующим) и выходным (декодирующим). Сеть учится воссоздавать ввод, который поступает в выходной слой; иными словами, она пытается вывести те же значения, которые получила на входе. Одной из важнейших особенностей этих сетей является то, что они не могут просто скопировать ввод в вывод. Например, в скрытом слое автокодировщика может быть меньше нейронов, чем во входном и выходном слоях. И поскольку он пытается воссоздать ввод в выходном слое, тот факт, что информация должна пройти через узкое место (скрытый слой), заставляет его усваивать входные данные, закодированные скрытым слоем и содержащие лишь самые важные признаки ввода, но игнорировать избыточную или ненужную информацию**.

---

* С критикой научной работы «Deep Learning Conspiracy» (Nature 521, стр. 436), впервые опубликованной Юргеном Шмидхубером в июне 2015 года, можно ознакомиться по адресу: http://people.idsia.ch/~juergen/deep-learning-conspiracy.html.

** Существует ряд способов ограничить автокодировщики так, чтобы не дать сети сформировать неинформативное отношение между вводом и выводом; например, во входные закономерности можно внедрить информационный шум и научить сеть фильтровать его при воссоздании данных. Мы также можем ограничить модули в скрытом (некодирующем) слое двоичными значениями. Например, в своей работе, посвященной предобучению, Хинтон и его коллеги изначально использовали ограниченные машины Больцмана, в которых кодирующий слой состоит из двоичных модулей.

Послойное предобучение с использованием автокодировщиков

В ходе послойного предобучения исходный автокодировщик усваивает кодирование необработанных данных, поступающих в сеть. Как только кодирование усвоено, модули в скрытом кодирующем слое становятся фиксированными, а выходной (декодирующий) слой отбрасывается. Затем обучается второй автокодировщик, но на этот раз его задача состоит в воссоздании представления данных, сгенерированного в результате прохождения этих данных через кодирующий слой исходного автокодировщика. В итоге этот второй автокодировщик размещается непосредственно над кодирующим слоем первого. Этот процесс накладывания кодирующих слоев один поверх другого считается «жадным», поскольку каждый кодирующий слой оптимизируется независимо от последующих слоев. Иными словами, вместо того чтобы пытаться решить общую задачу, стоящую перед сетью, каждый автокодировщик занимается поиском лучшего решения своей локальной задачи (формирование подходящего метода кодирования данных, которые он должен воссоздать).

Как только обучено достаточное количество* слоев, можно начинать оптимизацию. На этом этапе заключительный слой учится предсказывать целевой вывод сети. В отличие от предобучения предыдущих слоев, целевой вывод заключительного слоя разнится от вектора ввода и указан в учебном наборе данных. В простейшем случае оптимизация подразумевает неизменность предобученных слоев (то есть в ходе оптимизации веса в этих слоях не меняются); однако на данном этапе можно обучать и всю сеть целиком. Во втором случае послойное предобучение можно рассматривать как поиск подходящих исходных весов для начальных слоев сети. Кроме того, в качестве итоговой модели, которая генерирует прогнозы, необязательно использовать нейронную сеть. Вполне возможно взять представления данных, сгенерированные послойным предобучением, и подать их на вход алгоритму машинного обучения совершенно другого типа — например, методу опорных векторов или методу ближайшего соседа. Этот пример очень наглядно демонстрирует, как нейронные сети усваивают подходящие представления данных до того, как приступать к решению итоговой задачи (генерации прогнозов). Строго говоря, термин «предобучение» относится только к послойному обучению автокодировщиков; но его часто используют и для обозначения этапа оптимизации модели.

---

* Количество слоев, участвующих в предобучении, — это гиперпараметр, который специалист по анализу данных подбирает интуитивно, используя метод проб и ошибок.

Этапы послойного предобучения показаны на рис. 4.5. Слева проиллюстрировано обучение исходного автокодировщика, в процессе которого кодирующий слой (черные кружки) с тремя модулями пытается сформировать подходящее представление для воссоздания вектора ввода длиной 4. Посредине можно видеть процесс обучения второго автокодировщика, размещенного поверх кодирующего слоя первого. В этом автокодировщике скрытый слой из двух модулей пытается усвоить кодирование вектора ввода длиной 3 (который, в свою очередь, кодирует вектор длиной 4). Серым фоном на каждой диаграмме выделены компоненты сети, которые «замораживаются», то есть остаются неизменными на этом этапе обучения. Справа показан тот этап оптимизации, когда последний выходной слой учится предсказывать ключевой признак модели. В данном случае на этапе оптимизации используются замороженные предобученные слои сети.

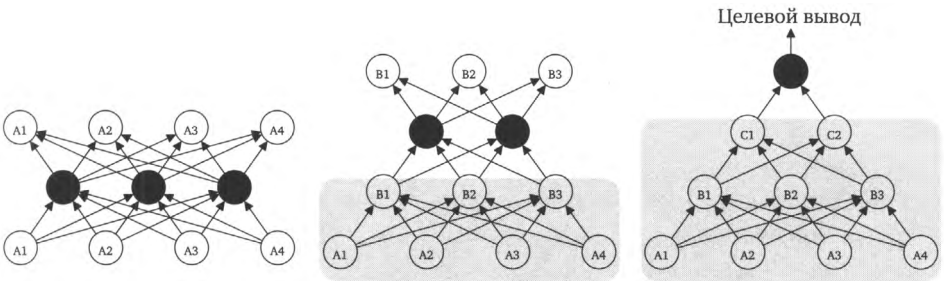


Рис. 4.5. Этапы обучения и оптимизации в жадном послойном предобучении. Черными кружками обозначены нейроны, обучение которых является основной задачей каждого этапа. Серым фоном выделены компоненты сети, которые замораживаются на каждом этапе обучения

Послойное предобучение сыграло важную роль в развитии машинного обучения. Это был первый метод обучения глубоких сетей, получивший широкое распространение*. Но сегодня большинство глубоких сетей обучаются без его использования. В середине 2000-х годов исследователи начали склоняться к мысли, что проблема исчезающих градиентов — это не строгое фундаментальное ограничение, а техническая преграда, которую можно преодолеть. Она не приводит к полному исчезновению градиентов отклонения; некоторые из них по-прежнему возвращаются к начальным слоям сети — просто они очень слабые. На сегодняшний день известен целый ряд факторов, которые играют важную роль в успешном обучении глубоких сетей.

---

* Еще в 1971 году Алексей Ивахненко продемонстрировал, что его метод МГУА способен обучать глубокие сети (до восьми слоев), но в сообществе исследователей это достижение было в основном проигнорировано.

Инициализация весов и функции активации ReLU

Одним из важных аспектов успешного обучения глубокой сети является то, как инициализируются ее веса. Принципы, определяющие влияние этого процесса на обучение, до сих пор неясны. Тем не менее существуют процедуры инициализации весов, которые, как показывает практика, помогают обучать глубокие сети. Одной из таких процедур является инициализация Глорота*. Она основана на ряде допущений, но эмпирический успех ее применения говорит сам за себя. Чтобы вам было легче понять, как она работает, приведем следующий факт: обычно существует связь между величиной значений в множестве и их дисперсией. Как правило, чем больше значения, тем больше их разброс. Таким образом, если дисперсия набора градиентов, проходящих через слой в одной точке сети, похожа на дисперсию набора градиентов, проходящих через другой слой сети, величина этих градиентов, скорее всего, будет аналогичной. Более того, дисперсия градиентов может быть связана с разбросом весов в том же слое, поэтому, чтобы обеспечить проход градиентов по сети, можно сделать так, чтобы разброс весов в каждом слое сети был примерно одинаковым. Инициализация Глорота позволяет назначить весам сети такие начальные значения, чтобы у всех слоев сети был похожий разброс как активаций, проходящих по направлению к выводу, так и градиентов, применяемых во время обратного распространения. Для этого инициализация Глорота вводит эвристическое правило, которое определяет подбор весов для сети с помощью следующего равномерного распределения: w — вес соединения между слоями j, который проходит инициализацию, и j+i, U[–a, a] — равномерное распределение на отрезке (–a, a); n_j — количество нейронов в слое j, а w~U указывает на то, что значение w выбирается из распределения U**.

$$w \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

* Инициализацию Глорота также называют инициализацией Хавьера. Хавьер Глорот является одним из авторов научной работы, в которой впервые была представлена эта процедура инициализации: «Understanding the Difficulty of Training Deep Feedforward Neural Networks» (Proceedings of the 13th International Conference on Artificial Intelligence and Statistics [AISTATS], 2010, стр. 249–256). Его соавтором был Джошуа Бенжио.

** Инициализацию Глорота можно также представить как выбор весов из нормального распределения с нулевым математическим ожиданием и среднеквадратическим отклонением, равным корню 2, поделенному на nj + nj+1. Но оба эти определения имеют одну и ту же цель: обеспечить похожее распределение активаций и градиентов в разных слоях сети.

В середине 2000-х годов исследователи начали склоняться к мысли, что проблема исчезающих градиентов — это не строгое фундаментальное ограничение, а техническая преграда, которую можно преодолеть.

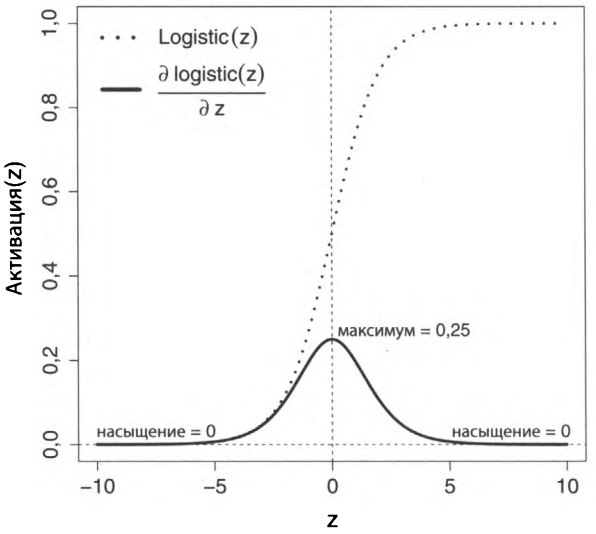


Рис. 4.6. Графики логистической функции и ее производной

Еще один фактор, от которого зависит успех обучения глубокой сети, связан с выбором функции активации для нейронов. Обратное распространение градиента ошибок через нейроны подразумевает умножение этого градиента на значение производной от функции активации в точке вывода нейрона, записанной во время прямого прохода. Производные tanh и логистических функций активации обладают рядом свойств, которые могут усугубить проблему исчезающего градиента, если их использовать на этапе умножения. На рис. 4.6 показан график логистической функции и ее производной. Максимум последней равен 0,25. Следовательно, после умножения градиента ошибок на значение производной логистической функции в точке активации нейрона максимальное значение градиента уменьшится в четыре раза. Еще одна проблема использования логистической функции состоит в том, что большие ее участки являются *насыщенными* (то есть имеют значения, очень близкие к 0 или 1), и скорость ее изменения на этих отрезках почти нулевая; поэтому производная функции стремится к 0. Это отрицательно сказывается на обратном распространении градиентов ошибок, поскольку эти градиенты будут практически обнуляться при прохождении через любой нейрон, который активировался в рамках одного из этих насыщенных участков. В 2011 году было продемонстрировано, что переход на функцию выпрямленной линейной активации, $g(x) = \max(0, x)$, улучшает обучение глубоких нейронных сетей с прямой связью (Глорот и др., 2011). Нейроны, использующие такие функции, называют выпрямленными линейными модулями, или выпрямителями (англ.

rectified linear units, или ReLU). Одно из преимуществ ReLU состоит в том, что функция активации остается линейной на отрезке с положительными значениями, где ее производная равна 1. Это означает, что градиенты могут легко проходить через ReLU с положительными активациями. Однако недостатком этого подхода является то, что градиенты функции на отрезке с отрицательными значениями равны нулю, поэтому ReLU на нем не обучается. Это нежелательное свойство, но его нельзя считать фатальным, так как при обратном распространении градиенты по-прежнему могут проходить через слой ReLU, если он имеет положительный вывод. Более того, существует ряд простых вариантов ReLU, которые пропускают градиент и на отрицательном участке; например, широкое применение получила функция ReLU «с утечкой» (Маас и др., 2013). На сегодня ReLU (и их разновидности) являются самым распространенным видом нейронов в исследованиях глубокого обучения.

Порочный круг наоборот: улучшение алгоритмов, ускорение оборудования, увеличение объема данных

Усовершенствованные методы инициализации весов и новые функции активации положительно сказались на развитии глубокого обучения, однако в последние годы двумя самыми важными факторами успеха в этой области стали ускорение компьютеров и огромное увеличение наборов данных. С точки зрения вычислений существенный прорыв произошел в конце 2000-х годов, когда сообщество глубокого обучения начало использовать графические адаптеры (англ. graphical processing units, или GPUs). Нейронную сеть можно представить в виде последовательного умножения матриц, перемежающихся с применением нелинейных функций активации, а GPU как раз оптимизированы для очень быстрых скалярных произведений. В результате графические адаптеры оказались идеальным аппаратным обеспечением для ускорения обучения глубоких сетей, и их использование существенно помогло развитию этой области. В 2004 году Ох и Джанг сообщили о двадцатикратном увеличении производительности при реализации нейронной сети на основе GPU (Ох и Джанг, 2004), а в следующем году были опубликованы еще две научных работы, которые демонстрировали потенциал графических адаптеров в ускорении обучения нейронных сетей: Штайнкраус и др. (2005) использовали GPU для обучения двухслойной нейронной сети, а Челлапилла и др. (2006) обучали с их помощью CNN. Однако в то время с применением графических адаптеров для обучения сетей возникали существенные

проблемы (обучающий алгоритм необходимо было реализовывать в виде последовательности графических операций), поэтому вначале исследователи в области нейронных сетей не спешили переходить на GPU. В 2007 году эти проблемы были в значительной степени разрешены, когда компания NVIDIA (производитель графических адаптеров) выпустила для своих GPU интерфейс программирования в стиле C под названием CUDA (compute unified device architecture — унифицированная архитектура вычислительных устройств)*. Она была специально создана для того, чтобы облегчить применение графических адаптеров для вычислительных задач общего характера. А уже в следующем году использование GPU для ускорения обучения нейронных сетей стало стандартным решением.

Рис. 4.7. Замкнутый круг, благодаря которому движется вперед развитие глубокого обучения. Эта иллюстрация основана на рис. 1.2 в работе Рейген и др., 2017

Однако даже с такими мощными компьютерными процессорами глубокое обучение не было бы возможным, если бы одновременно не стали доступными огромные наборы данных. Развитие Интернета и социальных сетей, распространение смартфонов и «умных» датчиков привели к тому, что за последнее десятилетие объемы захватываемых данных стали расти невероятно высокими темпами. Это значительно упростило организациям сбор крупных объемов данных. Такой рост был чрезвычайно важен для глубокого обучения, поскольку модели нейронных сетей очень хорошо масштабируются с увеличением количества информации (у них, скорее, могут возникать трудности

* https://developer.nvidia.com/cuda-zone.

с маленькими наборами данных). Все это также подтолкнуло организации к исследованию того, как эти данные можно использовать в новых сферах применения и инновациях. Это, в свою очередь, породило потребность в новых (более сложных) вычислительных моделях. А сочетание больших данных и сложных алгоритмов определяет необходимость создания более быстрого аппаратного обеспечения, способного справиться с дополнительными вычислительными нагрузками. Этот замкнутый круг между большими данными, прорывами в области алгоритмов (например, улучшенная инициализация весов, ReLU и т. д.) и совершенствованием оборудования, являющийся движущей силой революции глубокого обучения, проиллюстрирован на рис. 4.7.

Итоги

В истории глубокого обучения просматривается ряд тенденций. В какой-то момент произошел переход от простого двоичного ввода к более сложным непрерывным значениям. И усложнение входных данных будет продолжаться, так как модели глубокого обучения приносят больше всего пользы в высокоразмерных предметных областях, таких как обработка изображений и естественного языка. Изображения часто состоят из тысяч пикселей, а для работы с языком требуется способность представлять и обрабатывать сотни тысяч разных слов. Вот почему некоторые из самых известных примеров применения глубоких сетей относятся к этим предметным областям: программное обеспечение для распознавания лиц от Facebook, система нейронного машинного перевода от Google и т. д. Но вместе с тем растет число новых областей, в которых собираются большие и сложные наборы цифровых данных. Одной из таких сфер, где глубокое обучение может сыграть существенную роль в ближайшие годы, является здравоохранение, а в качестве примера отрасли с активным применением датчиков можно привести беспилотные автомобили.

Как это ни удивительно, в основе этих мощных моделей лежат простые модули обработки информации: нейроны. Идея коннекционизма, согласно которой сложное поведение может возникать в результате взаимодействия между большим количеством простых вычислительных элементов, остается актуальной по сей день. Это неожиданное поведение возникает в результате того, что последовательные слои сети осваивают иерархическое абстрагирование все более сложных признаков. А это, в свою очередь, достигается благодаря тому, что каждый нейрон учится выполнять простое преобразование

принимаемого им ввода. Затем сеть компонует эти последовательности мелких преобразований, чтобы сформировать сложное (весьма) нелинейное отношение между вводом и выводом. Итоговый вывод модели генерируется выходным слоем нейронов с учетом усвоенного представления, сгенерированного посредством иерархического абстрагирования. Вот почему глубина является таким важным аспектом нейронных сетей: чем глубже сеть, тем лучше модель способна усваивать сложные нелинейные связи. Во многих областях отношение между входными данными и желаемым выводом сводится именно к таким сложным нелинейным связям, почему глубокое обучение и превосходит другие методы машинного обучения.

Важным архитектурным аспектом создания нейронной сети является выбор функции активации, которая будет использоваться в ее нейронах. Именно эта функция привносит элемент нелинейности в сеть, что делает ее незаменимой в ситуациях, когда необходимо сформировать нелинейное отношение между вводом и выводом. Развитие сетей и функций активации идет рука об руку. Появление новых функций в этой области диктовалось потребностью в создании лучших условий для распространения градиента ошибок. Основным толчком к переходу от пороговых функций активации к tanh и логистическим уравнениям была потребность в дифференцируемых функциях, позволявших выполнять обратное распространение; точно так же недавний переход на ReLU был мотивирован улучшенным прохождением градиентов ошибок через сеть. Исследования в этой области продолжаются, и в ближайшие годы можно ожидать появления и внедрения новых функций активации.

Еще одним важным архитектурным решением в создании нейронной сети является определение ее структуры: например, как должны соединяться ее нейроны? В следующей главе мы обсудим два совершенно разных ответа на этот вопрос: применение сверточных и рекуррентных нейронных сетей.

СВЕРТОЧНЫЕ И РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

Адаптация структуры сети к определенным характеристикам данных из предметной области задачи может сократить время ее обучения и улучшить ее точность. Это можно осуществить разными способами: разбить соединения между нейронами в смежных слоях на подмножества (вместо того, чтобы делать слои полносвязными), сделать так, чтобы нейроны использовали общие веса, или добавить в сеть обратные связи. Такой процесс можно считать «встраиванием» в сеть знаний предметной области. Можно также сказать, что мы помогаем сети обучаться, ограничивая количество возможных функций, которые она способна сформировать, что упрощает поиск подходящего решения. То, как следует применять структуру сети к предметной области, не всегда очевидно, но в некоторых областях, в которых данные выглядят очень предсказуемо (например, последовательные данные, такие как текст, или решеточные данные вроде изображений), существуют широко известные сетевые архитектуры, доказавшие свою успешность. В этой главе вы познакомитесь с двумя наиболее популярными архитектурами глубокого обучения — сверточными и рекуррентными нейронными сетями.

Сверточные нейронные сети

Сверточные нейронные сети (англ. convolutional neural networks, или CNN) были созданы для распознавания образов и изначально применялись для обработки цифр, записанных от руки (Фукусима, 1980; Лекун, 1989). Основная задача этой архитектуры состояла в создании сети, в который начальный слой извлекает локальные визуальные признаки, а последующие слои объединяют их в признаки более высокого уровня. Локальный визуальный признак охватывает небольшой набор смежных пикселей в изображении. Например, при распознавании лиц нейроны в начальных слоях CNN учатся активироваться

в ответ на простые локальные признаки (такие как прямые линии под определенным углом или сегменты кривых), а нейроны, размещенные глубже, объединяют эту низкоуровневую информацию в признаки, представляющие части тела (например, глаз или нос); в заключительных слоях сети вывод с частями тела снова объединяется для поиска на изображении целых лиц.

При использовании этого подхода фундаментальная задача распознавания образов сводится к функциям обнаружения признаков, то есть способным определять наличие или отсутствие на изображении локальных визуальных элементов. Эти функции находятся в самом сердце нейронных сетей и формируются путем постепенного отбора подходящего сочетания весов для соединений сети. Именно так CNN учатся обнаруживать локальные визуальные признаки. Но вместе с тем сеть должна обладать такой архитектурой, которая позволила бы ей делать это независимо от того, в каком месте изображения эти признаки находятся. Иными словами, функции обнаружения признаков должны уметь работать пространственно-независимым образом. Например, система распознавания лиц должна уметь распознать форму глаза как в центре изображения, так и в его правом верхнем углу. Эта задача была основным архитектурным принципом, заложенным в CNN. Вот что писал об этом Лекун в 1989 году:

> Полезным является наличие набора элементов, способных обнаруживать конкретный признак в любом месте плоскости ввода. Поскольку точное местоположение признака не имеет отношения к классификации, мы можем позволить себе потерять в этом процессе некоторую позиционную информацию.

Для достижения этой пространственной независимости при обнаружении признаков нейроны в сетях CNN используют общие веса. В контексте распознавания образов функцию, которую реализует нейрон, можно считать механизмом обнаружения визуальных признаков. Например, нейроны в первом скрытом слое сети получают на вход набор пиксельных значений и в случае присутствия в них определенной закономерности (локального визуального элемента) сильно активируются. То, что функция, реализованная нейроном, определяется его весами, означает: два нейрона с одинаковым сочетанием весов реализуют одну и ту же функцию (механизм обнаружения признаков). В главе 4 мы познакомились с концепцией рецептивного поля, описывающей область, из которой нейрон получает свой ввод. Когда два нейрона с одинаковыми весами имеют разные рецептивные поля (то есть каждый из них анализирует свой участок ввода), то вместе они ведут себя подобно механизму

обнаружения признаков, который активируется, если признак присутствует в любом из этих рецептивных полей. Следовательно, чтобы спроектировать сеть с поддержкой пространственной независимости, мы можем создать набор нейронов с общими весами, организованных по следующему принципу:

1) каждый нейрон анализирует свою отдельную часть изображения;
2) совокупность рецептивных полей этих нейронов охватывает все изображение целиком.

Чтобы объяснить, как CNN ищет на изображении локальные признаки, часто используют аналогию с просмотром фотографии в темной комнате с помощью фонарика, излучающего узкую полосу света. В отдельно взятый момент времени ваш фонарик освещает какой-то локальный участок фотографии. Этот участок эквивалентен рецептивному полю одного нейрона, поэтому наведение фонарика — то же самое, что применение функции обнаружения признаков к локальной области изображения. Однако, если вы хотите исследовать всю фотографию, вам нужно более систематично подойти к наведению фонарика. Например, вы можете начать с левого верхнего угла и двигаться вправо, анализируя каждый новый участок, который становится видимым, пока не дойдете до правого края. Затем фонарик можно вернуть в левую часть фотографии, но чуть ниже той позиции, с которой вы начинали, и снова пройтись вправо. Этот процесс можно повторять, пока вы не достигнете правого нижнего угла фотографии. Процесс последовательного поиска по изображению с применением в каждом его локальном (освещенном) участке одной и той же функции является основным принципом сверточных сетей. В CNN этот поиск реализован с помощью набора нейронов с общими весами, совокупность рецептивных полей которых охватывает все изображение.

На рис. 5.1 проиллюстрированы разные этапы обработки, которые часто можно встретить в CNN. Матрица размером 6×6 в левой части представляет изображение, которое подается на вход сети. Справа от ввода находится матрица размером 4×4, представляющая слой нейронов, которые вместе просчитывают изображение в поисках конкретного локального признака. Каждый из них соединен со своим рецептивным полем (участком) изображения размером 3×3, и все они применяют к своему вводу одну и ту же матрицу:

$$\begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix}$$

Рецептивное поле нейрона [0,0] (слева вверху) в этом слое выделено серым прямоугольником размером 3×3, охватывающим верхнюю левую часть изображения. Штриховые стрелки, исходящие из каждой ячейки этой серой области, представляют ввод нейрона [0,0]. Рецептивное поле соседнего нейрона [0,1] представлено прямоугольником размером 3×3 с черной толстой границей. Обратите внимание на то, что рецептивные поля этих двух нейронов пересекаются. Величина пересечения определяется гиперпараметром, который называют длиной шага. В данном случае длина шага равна 1; это означает, что рецептивное поле нейрона относительно ввода смещается настолько, насколько меняется позиция в слое. При увеличении этого гиперпараметра пересечение между рецептивными полями уменьшается.

Рецептивные поля этих двух нейронов ([0,0] и [0,1]) являются матрицами пиксельных значений, и их веса тоже имеют вид матриц. В области компьютерного зрения матрицы весов, применяемые к вводу, называют ядром (или матрицей свертки); операция последовательной передачи ядра по изображению и в рамках каждой локальной области со взвешиванием каждого ввода и прибавлением результата к его локальным соседям называется сверткой. Следует иметь в виду, что эта операция не включает в себя нелинейную функцию активации (она применяется на более позднем этапе обработки). Ядро определяет функцию обнаружения признаков, которую реализуют все нейроны свертки. Свертывание ядра в рамках изображения эквивалентно перемещению средств обнаружения локальных признаков с записью всех позиций изображения, в которых находится нужный визуальный элемент. Результатом этого процесса является карта всех участков изображения с подходящими признаками, поэтому вывод свертки иногда называют картой признаков. Как уже отмечалось выше, операция свертки не включает в себя нелинейную функцию активации (она лишь слагает веса входных значений). Поэтому в карту признаков обычно вводят элемент нелинейности. Зачастую для этого к каждой позиции карты применяют выпрямленную линейную функцию такого вида: выпрямитель (z) = max(0, z). Перемещение выпрямленной линейной функции активации по карте признаков всего лишь заменяет все отрицательные значения на 0. На рис. 5.1 процесс обновления карты путем применения к каждому ее элементу выпрямленной линейной функции активации представлен слоем под названием «Нелинейность».

В цитате Яна Лекуна, приведенной в начале этой главы, упоминается о том, что конкретное местоположение признака на изображении может не иметь никакого отношения к задаче обработки. Учитывая это, сети CNN часто игнорируют информацию о местоположении, чтобы сделать свою

функцию классификации изображений более универсальной. Обычно это достигается за счет уменьшения размерности обновленной карты признаков (субдискретизации) с помощью слоя пулинга. Пулинг в какой-то степени похож на операцию свертки, описанную выше: он тоже многократно применяет одну и ту же функцию к разным участкам пространства ввода. В этом контексте пространством ввода часто выступает карта признаков, элементы которой были обновлены с помощью выпрямленной линейной функции. Более того, у каждой операции пулинга есть рецептивное поле в пространстве ввода (хотя в данном случае они могут и не пересекаться). Существует целый ряд разных функций пулинга; самая распространенная из них, функция максимума, возвращает максимальное входное значение. Вычисление среднего входного значения тоже является функцией пулинга.

Последовательность операций, состоящая из применения свертки, введения нелинейности в карту признаков и субдискретизации, является относительно стандартной для большинства сетей CNN. Обычно считается, что совокупность этих трех операций составляет сверточный слой сети; это можно видеть на рис. 5.1.

То, что свертка выполняет поиск по всему изображению, означает: местоположение искомого визуального признака (сочетания пикселей, определенного общим ядром), каким бы оно ни было, записывается на карту признаков (а также в последующем выводе из слоя пулинга, если таковой используется). Благодаря этому CNN поддерживает пространственно-независимое обнаружение визуальных элементов. Однако у этой возможности есть ограничение: сеть способна определять только элементы какого-то одного типа. Для большей универсальности CNN обучают сразу несколько слоев свертки (фильтров), каждый из которых формирует отдельную матрицу ядра (функцию обнаружения признаков). Стоит отметить, что слой свертки на рис. 5.1 иллюстрирует отдельный фильтр. Выводы нескольких фильтров можно интегрировать разными способами. Например, вы можете взять карты признаков, сгенерированные разными фильтрами, и объединить их в одну карту с множественной фильтрацией. Затем следующий слой свертки примет эту карту на вход. Еще можно воспользоваться плотным слоем нейронов; на рис. 5.1 это проиллюстрировано с помощью заключительного слоя. Этот плотный слой ведет себя точно так же, как стандартные слои в полносвязной прямонаправленной сети. Каждый нейрон в плотном слое соединяется со всеми элементами, которые выводятся каждым из фильтров, и каждый нейрон формирует набор весов, уникальный для этого нейрона, и применяет их к входным значениям. Это означает, что каждый нейрон в плотном слое может научиться по-своему интегрировать информацию из разных фильтров.

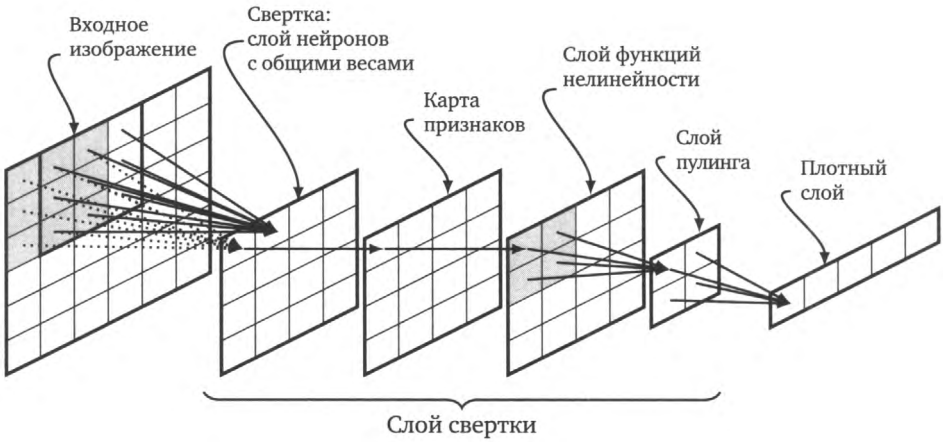


Рис. 5.1. Разные этапы обработки в сверточном слое. Обратите внимание: на этой диаграмме карта признаков обозначает структуры данных; остальные этапы представляют операции с данными

У сверточной сети AlexNet, выигравшей в 2012 году соревнование ImageNet LargeScale Visual Recognition Challenge (ILSVRC), было пять слоев свертки, за которыми шли три плотных слоя. Первый из сверточных слоев содержал девяносто шесть разных ядер (или фильтров), а также поддерживал нелинейность ReLU и пулинг. Второй сверточный слой состоял из 256 ядер и тоже включал в себя нелинейность ReLU и пулинг. Третий, четвертый и пятый слои пропускали этапы нелинейности и пулинга и содержали 384, 384 и 256 ядер соответственно. Вслед за пятым слоем свертки шли три плотных слоя с 4096 нейронами в каждом. В целом сеть AlexNet насчитывала шестьдесят миллионов весов и 650 000 нейронов. Какой бы большой ни казалась цифра в 60 миллионов, на самом деле количество необходимых весов здесь было существенно уменьшено за счет того, что разные нейроны часто совместно использовали одни и те же веса. Пониженное количество необходимых весов является одним из преимуществ сетей CNN. В 2015 году в Microsoft Research разработали сверточную сеть под названием ResNet, которая выиграла состязание ILSVRC 2015 (Хе и др., 2016). Она расширяла стандартную архитектуру сетей CNN за счет использования пропускающих соединений (англ. skip-connections). Пропускающее соединение берет вывод одного слоя и сразу подает его на вход другому слою, который может находиться намного глубже. Это позволило обучать очень глубокие сети. Так, модель ResNet, разработанная в Microsoft Research, обладала 152 слоями.

Рекуррентные нейронные сети

Рекуррентные нейронные сети (англ. recurrent neural networks, или RNN) оптимизированы для обработки последовательных данных. Каждый элемент последовательности обрабатывается по очереди. Сеть RNN содержит всего один скрытый слой, но у нее также есть буфер памяти, который хранит вывод этого скрытого слоя; на каждой итерации скрытый слой принимает не только текущий, но и предыдущий вывод, который берется из буфера. Благодаря рекуррентному потоку информации сеть обрабатывает каждый ввод в контексте, сгенерированном при обработке предыдущего ввода, который, в свою очередь, обрабатывался в контексте предшествовавшей ему итерации. Таким образом, информация, проходящая через рекуррентный цикл, кодирует контекстные данные из всех (потенциально) предыдущих входных значений в последовательности. Это позволяет сети «помнить» о том, с чем она уже сталкивалась ранее, что помогает ей в принятии решения и текущем вводе. Глубина RNN обусловлена тем, что вектор памяти направлен вперед и эволюционирует на каждой итерации; в итоге рекуррентная сеть считается настолько глубокой, насколько длинной является ее последовательность.

Архитектура RNN и то, как поток информации проходит через нее во время обработки последовательности, проиллюстрировано на рис. 5.2. На каждом интервале сеть принимает на вход вектор с двумя элементами. В левой части диаграммы (интервал=1,0) показано прохождение информации по сети при получении первого элемента последовательности. Вектор ввода передается следующим трем нейронам в скрытом слое. В то же время эти нейроны получают информацию, хранящуюся в буфере памяти. Но так как это начальный ввод, в буфере памяти будут находиться только инициализационные значения по умолчанию. Каждый нейрон в скрытом слое обработает ввод и сгенерирует какой-то вывод. В средней части рис. 5.2 (интервал=1,5) показано, как этот вывод проходит по сети: активация каждого нейрона передается выходному слою, который обрабатывает ее, чтобы сгенерировать вывод сети, а также сохраняется в буфере памяти (перезаписывая то, что там находилось прежде). Элементы буфера просто хранят ту информацию, которая в них записана, никак ее не изменяя. Благодаря этому у ребер, соединяющих скрытые модули с буфером, нет никаких весов. Хотя на всех остальных ребрах сети, включая те, которые находятся между модулями буфера и нейронами скрытого слоя, веса присутствуют. На интервале 2 сеть берет из последовательности следующий ввод и передает его скрытому слою нейронов вместе с информацией, хранящейся в буфере. На этот раз буфер содержит вывод, сгенерированный скрытыми нейронами в ответ на первый ввод.

На рис. 5.3 показана рекуррентная сеть, развернутая по времени и обрабатывающая последовательность входных элементов [X_1, X_2,..., X_t]. Каждый прямоугольник на этой диаграмме обозначает слой нейронов. Прямоугольник, помеченный как h_0, представляет состояние буфера памяти в момент инициализации сети; а прямоугольники с метками [h_1,..., h_t] и [Y1,..., Yt] — это скрытый и, соответственно, выходной слои сети на каждом интервале. Каждая стрелка на этой диаграмме представляет набор соединений между двумя слоями. Например, вертикальная стрелка, ведущая от X_1 к h_1, обозначает соединения между входным и скрытым слоем на интервале 1. Точно так же горизонтальные стрелки, соединяющие скрытые слои, отражают процесс сохранения вывода скрытого состояния в буфер памяти на определенном интервале (не показано) и передачу этого вывода скрытому слою на следующем шаге с использованием соединений между буфером памяти и скрытым состоянием. На каждом интервале ввод из последовательности передается в сеть и подается на вход скрытому слою, а тот генерирует вектор активаций, который проходит в выходной слой и попадает в следующий интервал по горизонтальным стрелкам, соединяющим скрытые состояния.

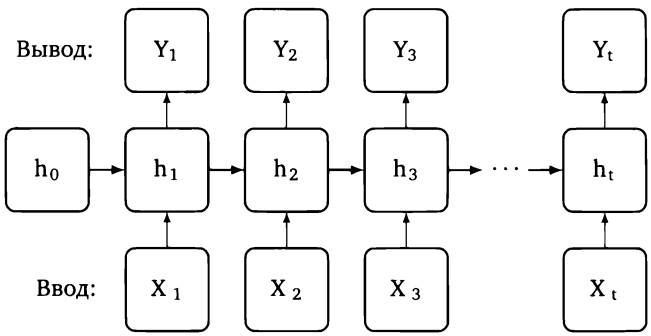


Рис. 5.3. Сеть RNN, развернутая по времени и обрабатывающая последовательность входных элементов [X_1, X_2,..., X_t]

Несмотря на способность обрабатывать последовательности входных данных, сети RNN испытывают трудности с проблемой исчезающих градиентов. Причина такова: чтобы научить рекуррентную сеть обрабатывать такой ввод, ошибку необходимо возвращать обратно по всей длине последовательности. Например, в случае с сетью на рис. 5.3 погрешность вывода Y_t должна вернуться в самое начало, чтобы ее можно было использовать для обновления весов на соединениях от h_0 к X_1 и к h_1. Это влечет за собой обратное распространение погрешности через все скрытые слои, что, в свою очередь, подразумевает неоднократное умножение этой погрешности на веса соединений,

Глубина RNN обусловлена тем, что вектор памяти направлен вперед и эволюционирует на каждой итерации; в итоге рекуррентная сеть считается настолько глубокой, насколько длинной является ее последовательность.

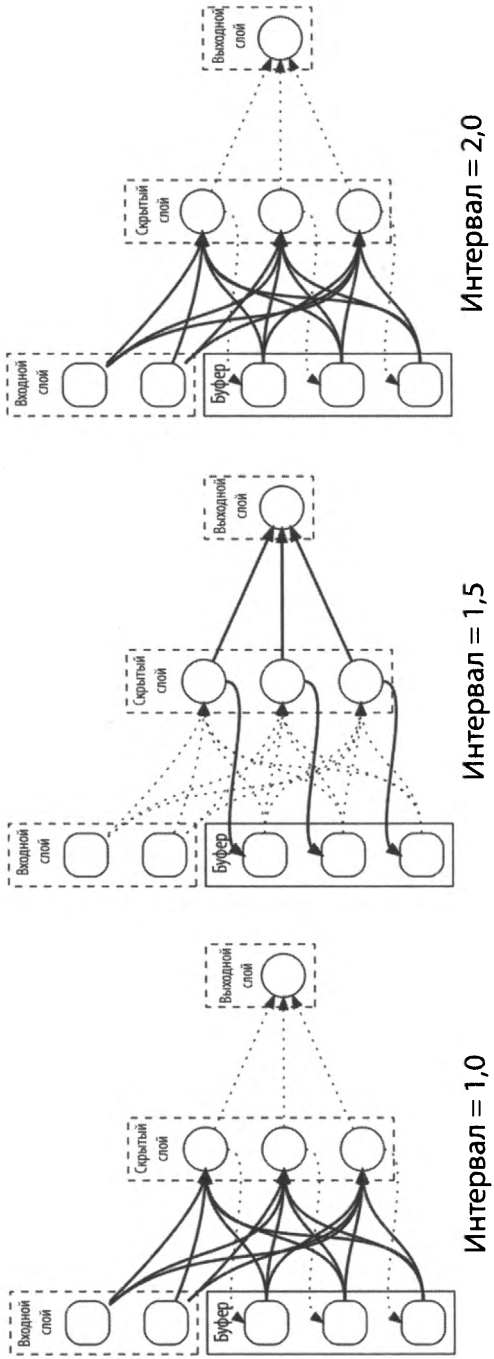


Рис. 5.2. Прохождение информации по сети RNN по мере того, как она обрабатывает последовательность входных элементов. Жирные стрелки — это активные пути распространения информации в каждый момент времени; пунктирные стрелки обозначают соединения, которые неактивны на соответствующем интервале

передающих вывод одного скрытого слоя в другой. Проблема этого процесса состоит в том, что на всех соединениях между скрытыми слоями используются одинаковые веса: каждая горизонтальная стрелка представляет один и тот же набор соединений между буфером памяти и скрытым слоем, и их веса не меняются со временем (то есть они остаются фиксированными на всех интервалах во время обработки заданной входной последовательности). Следовательно, в ходе обратного распространения ошибки через k интервалов градиент ошибок умножается на один и тот же набор весов k раз. Это то же самое, что умножить каждый градиент ошибок на вес в степени k. Если этот вес меньше 1, при возведении в степень он будет экспоненциально уменьшаться; экспоненциальное уменьшение (относительно длины последовательности) будет происходить и с градиентом, пока он не исчезнет.

Сети с долгой краткосрочной памятью (англ. long short-term memory, или LSTM) спроектированы таким образом, чтобы смягчать эффект исчезающих градиентов. Для этого они переносят многократное умножение, происходящее во время обратного распространения в RNN, на один и тот же вектор весов. В основе LSTM-модуля* лежит компонент под названием блок. Это то место, в котором сохраняется и распространяется в обратном направлении вывод (краткосрочная память). По существу, блок зачастую хранит вектор выводов. Прохождение выводов через блок на каждом интервале контролируется тремя компонентами, которые называются вентилями: вентиль забывания, а также входной и выходной вентили. Вентиль забывания определяет, какой вывод блока следует отбрасывать на каждом отдельном интервале; входной вентиль управляет обновлением вывода в блоке, происходящим в ответ на новый ввод; а выходной вентиль выбирает вывод блока, который будет использоваться при генерации конечного результата для текущего ввода. Каждый вентиль состоит из слоев стандартных нейронов, и каждый нейрон в слое соответствует одному выводу в положении блока.

На рис. 5.4 изображена внутренняя структура LSTM-блока. Каждая стрелка обозначает вектор выводов. Блок проходит по верхней части диаграммы слева (c_{t-1}) направо (c_t). Вывод блока может находиться в диапазоне от –1 до +1. Проходя обработку для одного входного значения, вектор ввода x_t сначала объединяется с вектором скрытого состояния, который был передан вперед из предыдущего интервала h_{t-1}. Результат проходит через вентили,

---

* Объяснение работы LSTM-модулей, приведенное здесь, основано на замечательной статье Кристофера Олаха, в которой приводится четкое и подробное описание LSTM. Она доступна по адресу: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

слева направо: вентиль забывания принимает совокупность ввода и скрытого состояния и пропускает этот вектор через слой нейронов с сигмоидной (логистической)* функцией активации. Результат применения сигмоиды в нейронах слоя забывания выражается в том, что вывод этого слоя имеет вид вектора значений в диапазоне от 0 до 1. Затем состояние блока умножается на этот вектор забывания. В результате выходные значения в состоянии блока, умноженные на элементы вектора забывания, равные 0, отбрасываются, а те, что умножены на 1, запоминаются. Умножение состояния блока на вывод сигмоидного слоя фактически фильтрует этот блок.

Затем входной вентиль решает, какую информацию следует добавить к состоянию блока. Обработка на этом этапе выполняется компонентами, помеченными как «Ввод» в средней части рис. 5.4. Вначале вентиль выбирает в состоянии блока элементы, которые нужно обновить, а затем принимает решение о том, какую информацию следует включить в это обновление. Выбор элементов реализуется с использованием механизма фильтрации, похожего на тот, который мы уже встречали в вентиле забывания: объединенный ввод x_t пропускается вместе со скрытым состоянием h_{t-1} через слой сигмоидных модулей, чтобы сгенерировать вектор элементов той же ширины, что и сам блок. Каждый элемент вектора будет находиться в диапазоне от 0 до 1; решение о том, обновлять или не обновлять элемент блока принимается в зависимости от того, к какой границе диапазона приближено значение — 1 или 0. Одновременно с генерацией вектора фильтрации объединенный ввод и скрытое состояние проходят через слой tanh-модулей (то есть нейронов, которые используют tanh в качестве функции активации). И снова каждому выводу в LSTM-блоке соответствует отдельный tanh-модуль. Этот вектор представляет информацию, которая может быть добавлена к состоянию блока. Tanh-модули используются для генерации этого вектора обновлений в связи с тем, что их выходные значения находятся в диапазоне от –1 до +1. Благодаря этому в результате обновления** выходные значения в элементах блока могут как увеличиваться, так и уменьшаться. После генерации этих двух векторов вычисляется итоговый вектор обновлений; для этого вектор вывода слоя tanh

---

* Сигмоида на самом деле является частным случаем логистической функции, и в контексте этого обсуждения разницей между ними можно пренебречь.

** Если бы мы использовали, к примеру, сигмоидные модули с диапазоном вывода от 0 до 1, во время каждого обновления активации могли были только увеличиваться или оставаться неизменными, и в итоге состояние блока было бы насыщено максимальными значениями.

умножается на вектор фильтрации, полученный из слоя сигмоиды. Результат добавляется в блок путем сложения векторов.

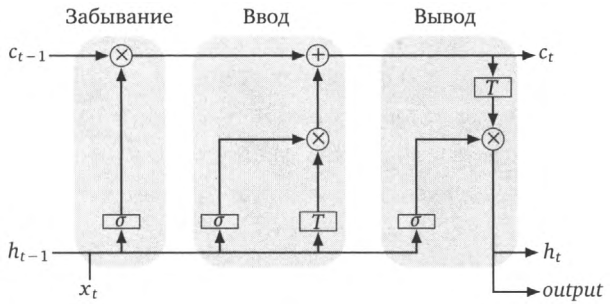


Рис. 5.4. Схема внутреннего устройства LSTM-модуля: σ обозначает слой нейронов с сигмоидной функцией активации, T — слой нейронов с активацией в виде функции tanh. Символ × представляет умножение векторов, а + — их сложение. Эта диаграмма основана на рисунке Кристофера Олаха, доступном по адресу: **http://colah.github.io/posts/2015-08-Understanding-LSTMs/**

Заключительный этап обработки в LSTM состоит в выборе элементов блока, которые должны возвращаться в ответ на текущий ввод. За это отвечают компоненты, помеченные как «Вывод» (в правой части рис. 5.4). Чтобы сгенерировать потенциальный выходной вектор, блок проходит через слой tanh. В то же время объединенный ввод и распространенный вектор скрытого состояния проходят через слой сигмоидных модулей, чтобы создать еще один вектор фильтрации. Затем вычисляется настоящий выходной вектор; для этого его потенциальная версия умножается на этот фильтр. Результат передается в выходной слой и вместе с тем попадает в следующий интервал с новым скрытым состоянием h_t.

Благодаря тому, что LSTM-модуль состоит из нескольких слоев нейронов, его можно считать отдельной сетью. Однако сеть RNN возможно спроектировать так, чтобы LSTM-модуль выступал как ее скрытый слой. В такой конфигурации LSTM принимает ввод на каждом интервале и генерирует вывод для каждого ввода. Рекуррентные сети, в которых используются LSTM-модули, часто называют LSTM-сетями.

LSTM-сети идеально подходят для обработки естественного языка (англ. natural language processing, или NLP). Ключевая проблема применения нейронных сетей в этой области состоит в том, что слова необходимо преобразовывать в числовые векторы. Одним из самых популярных методов подобного преобразования являются модели word2vec, которые Томас Миколов создал вместе со своими коллегами из Google (Миколов и др., 2013). Эти модели основаны на идее, что слова, встречающиеся в похожих контекстах, имеют

похожее значение. Под контекстом здесь имеются в виду окружающие слова. Например, слова *Лондон* и *Париж* имеют похожую семантику, так как часто встречаются в связке с одними и теми же словами, такими как *столица, город, Европа, праздник, аэропорт* и т. д. Модели word2vec представляют собой нейронные сети, реализующие этот принцип семантического подобия путем инициализации всех слов произвольными векторами и последующего их итеративного обновления с использованием взаимных вхождений в рамках лингвистического корпуса, в результате чего словам с похожей семантикой в итоге присваиваются похожие векторы.

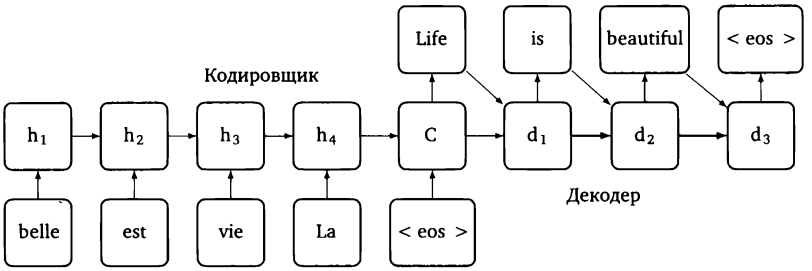


Рис. 5.5. Схема архитектуры seq2seq (или кодировщика-декодера)

Одной из областей NLP, на которую существенно повлияло глубокое обучение, является машинный перевод. На рис. 5.5 показана общая схема архитектуры seq2seq (или кодировщика-декодера) для нейронного машинного перевода (Суцкевер и др., 2014). Эта архитектура состоит из двух LSTM-сетей, соединенных вместе. Первая обрабатывает входящее предложение последовательно, слово за словом. В данном примере исходным языком является французский. Слова подаются в систему в обратном порядке, потому что, как показывает практика, это помогает осуществить лучший перевод. Символ eos обозначает конец предложения (англ. end of sentence). При вводе каждого слова кодировщик обновляет скрытое состояние и передает его вперед, в следующий интервал. Скрытое состояние, сгенерированное кодировщиком в ответ на символ eos, принимается за векторное представление входного предложения. Этот вектор передается в качестве начального ввода в LSTM-декодер. Декодер обучен пословно выводить переведенное предложение, и каждое сгенерированное слово опять становится вводом системы на следующем интервале. Декодер в каком-то смысле «галлюцинирует» перевод, используя в процессе его генерации собственный вывод. Это продолжается, пока он не выведет символ eos.

Идея использования числового вектора для представления (межязыкового) смысла предложений пользуется большой популярностью. Более того, она была расширена для межмодальных/мультимодальных представлений. Например, интереснейшей тенденцией последних лет стала разработка систем автоматической маркировки изображений. Эти системы могут принять на вход изображение и сгенерировать его описание на естественном языке. Их базовая структура очень похожа на архитектуру нейронного машинного перевода, показанную на рис. 5.5. Основное отличие в том, что вместо кодирующей LSTM-сети используется сверточная сеть; она обрабатывает изображение и генерирует его векторное представление, которое затем передается LSTM-кодировщику (Сюй и др., 2015). Это еще один пример того, как способность усваивать сложные представления информации делает глубокое обучение чрезвычайно мощным. В данном случае усваиваются межмодальные представления, которые позволяют передавать содержание изображения в письменном виде. Совместное применение архитектур CNN и RNN становится все популярней, позволяя объединять преимущества обеих систем и работать с очень сложными данными.

Независимо от используемой нами сетевой архитектуры, для создания точной модели необходимо подобрать подходящие веса. Веса нейрона определяют преобразование, через которое проходит его ввод. Поэтому именно веса сети являются фундаментальными составными элементами представления, которое эта сеть усваивает. На сегодняшний день стандартным методом подбора этих весов служит алгоритм обратного распространения, получивший известность еще в 1980-х годах. В следующей главе содержится подробное введение в этот алгоритм.

ФУНКЦИИ ОБУЧЕНИЯ

Модель нейронной сети, независимо от ее глубины и сложности, реализует функцию, соотносящую ввод с выводом. Эта функция определяется весами, которые использует сеть. Поэтому обучение сети (формирование функции, которую сеть должна реализовать) на готовых данных подразумевает поиск сочетания весов, позволяющего сети лучше всего смоделировать закономерности в этих данных. Для извлечения из данных закономерностей чаще всего используют алгоритм градиентного спуска. Отметим его сходство с правилом обучения перцептрона и алгоритмом LMS, описанным в главе 4: он определяет правило обновления весов, которые используются в функции, с учетом того, какое отклонение эта функция демонстрирует. Сам по себе алгоритм градиентного спуска можно использовать для обучения отдельного выходного нейрона. Однако с его помощью нельзя обучить целую глубокую сеть с множеством скрытых слоев. Это ограничение объясняется проблемой присваивания коэффициентов доверия: каким образом следует распределять ответственность за отклонение сети между разными ее нейронами (включая скрытые)? Поэтому при обучении глубоких нейронных сетей алгоритм градиентного спуска работает в связке с алгоритмом обратного распространения.

Процесс обучения глубокой нейронной сети можно охарактеризовать так: произвольная инициализация весов сети с последующим их обновлением на каждой итерации в ответ на погрешности, которые допускаются при работе с набором данных. Это повторяется до тех пор, пока сеть не начнет работать так, как от нее ожидают. В этом контексте алгоритм обратного распространения берет на себя решение проблемы присваивания коэффициентов доверия, а алгоритм градиентного спуска определяет правило обучения, которое обновляет веса в сети.

В этой главе данной книги больше всего математики. Но по большому счету об этих двух алгоритмах вам достаточно знать лишь то, что их можно использовать для обучения глубоких сетей. Поэтому, если вам некогда

углубляться в подробности этой главы, можете ее только пролистать. Но если вы хотите как следует разобраться в этих двух алгоритмах, я советую вам ознакомиться с этим материалом. Эти алгоритмы являются основой глубокого обучения, и, разобравшись, как они работают, вы быстрее всего поймете потенциал и ограничения данной области. Я попытался представить материал этой главы в доступном виде, поэтому, если вы ищете относительно плавное, но тем не менее комплексное введение в эти алгоритмы, вы, как мне кажется, сможете найти его здесь. Данная глава начинается с описания алгоритма градиентного спуска и затем демонстрирует, как можно его использовать для обучения нейронных сетей в сочетании с алгоритмом обратного распространения.

Градиентный спуск

Простейшим из типов функций является линейное отношение между одним вводом и одним выводом. В табл. 6.1 представлен набор данных с одним входным признаком и одним выходным значением. На рис. 6.1 вы можете видеть диаграмму рассеяния этих данных вдоль прямой, которая лучше всего для них подходит. Эту прямую можно использовать в качестве функции для связывания входного значения и вывода, который должен получиться в результате. Например, если x = 0,9, эта линейная функция вернет y = 0,6746. Отклонение (или погрешность) при использовании этой прямой для моделирования данных обозначено пунктирными линиями, соединяющими прямую с каждой исходной величиной.

Таблица 6.1. Демонстрационный набор данных с одним входным признаком, x, и одним выходным (целевым) признаком, y

X	Y
0.72	0.54
0.45	0.56
0.23	0.38
0.76	0.57
0.14	0.17

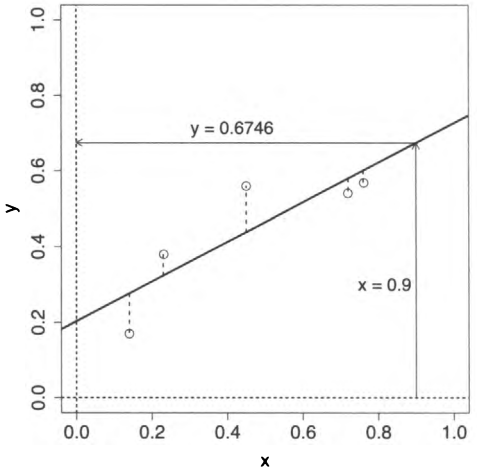


Рис. 6.1. Диаграмма рассеяния данных с прямой, которая лучше всего описывает эти данные. Отклонения для каждого образца представлены в виде вертикальных пунктирных линий. Здесь также показана связь между вводом x = 0,9 и выводом y = 0,6746, которую определяет эта прямая

В главе 2 мы показывали, как линейную функцию можно представить с помощью следующего уравнения прямой:

$$y = mx + c$$

Здесь m — наклон прямой, а c — отсечение по оси y, определяющее, в какой точке прямая пересекает ось ординат. Если взять прямую на рис. 6.1, c = 0,203 и m = 0,524; вот почему функция возвращает y = 0,6746, когда x = 0,9, как показано ниже:

$$0{,}6746 = (0{,}524 \times 0{,}9) + 0{,}203$$

Наклон m и отсечение по вертикали c — это параметры модели, и их можно корректировать, чтобы подогнать модель под данные.

Уравнение прямой имеет непосредственное отношение к операции взвешенной суммы, которая используется в нейроне. В этом можно убедиться, если записать параметры модели в виде весов ($c \rightarrow w_0$, $m \rightarrow w_1$):

$$y = (w_0 \times 1) + (w_1 \times x)$$

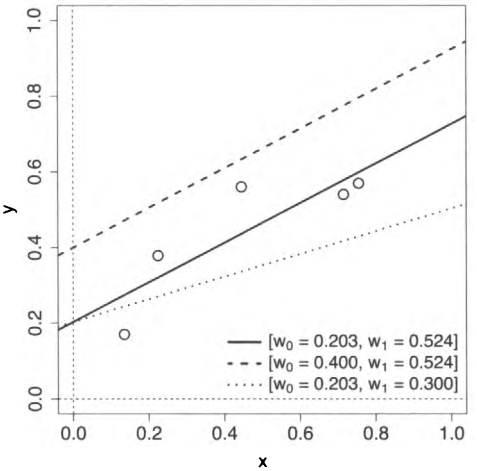


Рис. 6.2. График, иллюстрирующий, как прямая меняется в зависимости от отсечения (w_0) и наклона (w_1)

Путем изменения любого из этих весов (или параметров модели) можно создавать разные прямые (разные линейные модели данных). На рис. 6.2 показано, как изменения наклона и отсечения влияют на график прямой: сверху видно, что происходит при уменьшении наклона. Изменение отсечения w_0 по оси y перемещает прямую по вертикали, а модификация наклона w_1 приводит к вращению прямой вокруг точки (x = 0, y = отсечение).

Каждая из этих новых прямых определяет новую функцию x от y, и у каждой функции будет свое отклонение относительно данных. На рис. 6.2 можно видеть, что сплошная прямая [w_0 = 0,203, w_1 = 0,524] лучше соответствует данным, чем две другие, так как она проходит ближе к точкам в нашем наборе. Иными словами, отклонение этой прямой от каждой точки данных в среднем меньше, чем у двух других прямых. Общее отклонение модели для заданного набора данных можно посчитать, сложив вместе все погрешности, которые она допускает для каждого образца. Для этого обычно используют формулу для вычисления так называемой «суммы квадратов ошибок», или среднеквадратической ошибки (англ. sum of squared errors, или SSE):

$$\mathrm{SSE} = \frac{1}{2}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2$$

Эта формула показывает, как производится сложение отклонений модели для набора данных, содержащего n образцов. Оно вычисляет отклонение модели для каждого из n образцов, вычитая предполагаемое целевое значение, возвращенное моделью, из правильного целевого значения для этого образца,

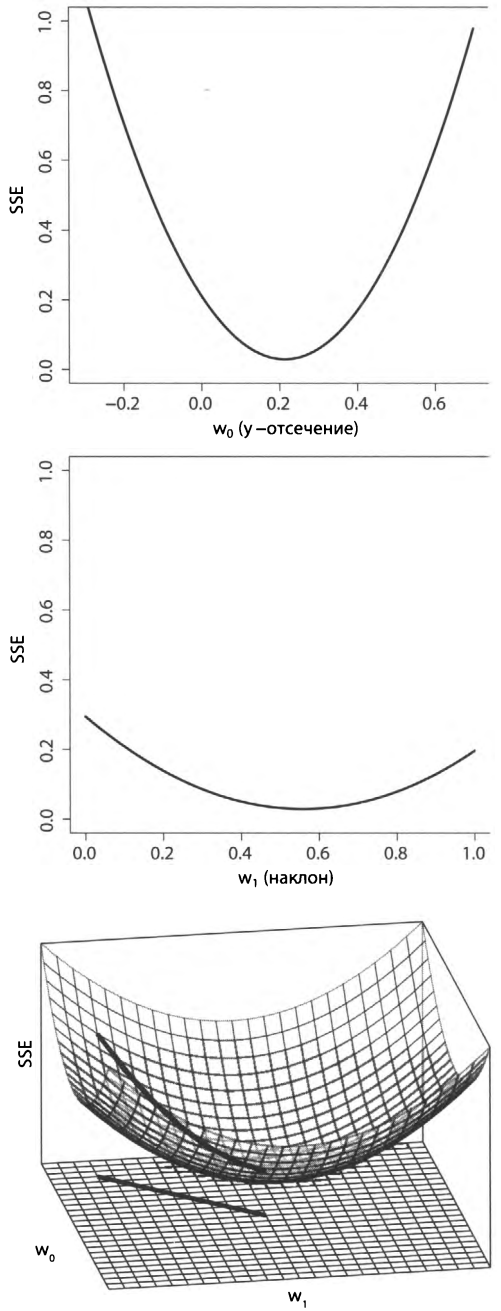


Рис. 6.3. Графики изменения отклонения (SSE) линейной модели в зависимости от ее параметров. Вверху: форма SSE линейной модели с фиксированным наклоном $w_1 = 0{,}524$ и w_0 в диапазоне от 0,3 до 1. Посредине: форма SSE линейной модели с фиксированным отсечением по вертикали $w_0 = 0{,}203$ и w_1 в диапазоне от 0 до 1. Внизу: поверхность ошибок линейной модели в ситуации, когда меняются и w_0, и w_1

указанного в наборе данных. В этом уравнении y_j — это правильное выходное значение для признака, указанное в наборе данных для соответствующего образца j, а $\hat{y}_j$ — это приблизительное целевое значение, возвращенное моделью для того же образца. Затем каждое из этих отклонений возводится в квадрат, и результаты слагаются. Возведение в квадрат гарантирует, что все операнды положительные, поэтому сложение отклонений для образцов, в которых функция недооценивает целевое значение, не нивелирует отклонения для образцов, в которых значение переоценивается. А вот умножение суммы отклонений на ½ не имеет особого отношения к текущему обсуждению, но пригодится позже. Чем меньше SSE функции, тем лучше она моделирует данные. Следовательно, сумму квадратов ошибок можно использовать в качестве меры приспособленности потенциальной функции (в нашем случае это модель прямой) для оценки того, насколько хорошо она соответствует данным.

На рис. 6.3 показано, как отклонение линейной модели зависит от изменения ее параметров. На этих графиках изображена SSE линейной модели для набора данных из табл. 6.1, в котором каждый ввод соответствует ровно одному выводу. У каждого параметра есть одно оптимальное значение, и чем дальше он от него отходит (в любую сторону), тем больше отклонение модели. В результате график изменения погрешности модели в зависимости от каждого параметра принимает вогнутую (чашеобразную) форму. Это особенно хорошо видно на верхнем и среднем графиках на рис. 6.3, где SSE модели сводится к минимуму, когда $w_0 = 0{,}203$ (нижняя точка кривой на верхнем графике) и $w_1 = 0{,}524$ (нижняя точка кривой на среднем графике).

Если нарисовать график зависимости отклонения модели от обоих параметров, получится трехмерная вогнутая поверхность, известная как поверхность ошибок. Чтобы ее проиллюстрировать, в нижней части рис. 6.3 используется чашеобразная сетка. Для создания этой поверхности сначала определяется пространство весов, представленное плоской сеткой внизу графика. Каждая его координата представляет собой сочетание отсечения (значения w_0) и наклона (значения w_1), и поэтому определяет отдельную прямую. Следовательно, перемещение по этому плоскому пространству весов эквивалентно переходу между разными моделями. На втором этапе создания поверхности ошибок каждой прямой назначается высота (то есть координата) в пространстве весов. Это значение описывает SSE модели в заданной координате, или, проще говоря, высота поверхности ошибок относительно плоскости пространства весов — это SSE соответствующей линейной модели, когда ее используют для моделирования набора данных. Координаты пространства весов, соответствующие нижней точке поверхности ошибок, определяют

линейную модель с наименьшей SSE для заданного набора данных (то есть линейную модель, которая лучше всего подходит для этих данных).

Форма поверхности ошибок на графике, изображенном на рис. 6.3, говорит о том, что у этого набора данных есть всего одна оптимальная линейная модель, так как на дне чаши находится одна координата с наименьшей высотой (ошибкой). Удаляясь от этой модели (путем изменения ее весов), мы обязательно получаем большую SSE. Это эквивалентно перемещению к новым координатам в пространстве весов, которые связаны с большей высотой на поверхности ошибок. Вогнутая, или чашеобразная, поверхность ошибок чрезвычайно полезна при формировании линейной функции для моделирования набора данных, поскольку процесс обучения можно свести к поиску нижней точки на этой поверхности. Стандартный алгоритм, который используется в этом поиске, называется градиентным спуском.

Вначале алгоритм градиентного спуска инициализирует модель с помощью набора весов, подобранного случайным образом. Затем вычисляется SSE этой случайно инициализированной модели. Вместе эти веса и SSE определяют исходную точку поиска на поверхности ошибок. Модель, инициализированная случайным образом, с очень высокой долей вероятности покажет плохие результаты, поэтому поиск, скорее всего, начнется на возвышенном участке поверхности ошибок. Однако это плохое начало не является препятствием, так как в процессе поиска можно будет найти лучшее сочетание весов, следуя градиенту, спускающемуся по поверхности ошибок, пока не будет достигнута нижняя точка этой поверхности (место, отдаление от которого в любом направлении приводит к увеличению SSE). Вот почему этот алгоритм называют градиентным спуском: градиент, по которому он спускается, — это градиент модели на поверхности ошибок относительно данных.

Важно здесь то, что переход от исходной позиции к дну происходит не за одно обновление, а постепенно. На каждой итерации текущие веса обновляются таким образом, чтобы перейти в ближайшую точку пространства весов с меньшей SSE. На достижение дна поверхности ошибок может уйти много итераций. Представьте себе туриста, который очутился на склоне холма, окутанного густым туманом. Его машина припаркована на дне долины, но из-за тумана он может видеть лишь на расстоянии нескольких метров в любом направлении. Если предположить, что долина имеет гладкую вогнутую форму, турист может найти путь к своей машине, даже несмотря на туман. Для этого он может шаг за шагом спускаться с холма, следуя локальному градиенту в своей текущей позиции. Отдельное прохождение поиска методом градиентного спуска проиллюстрировано на нижнем графике

рис. 6.3. Черная кривая на поверхности ошибок обозначает путь, который поиск проходит вниз по поверхности, а черная прямая в пространстве весов демонстрирует соответствующие обновления весов, которые происходят в ходе перемещений ко дну. Формально этот метод является одним из алгоритмов оптимизации, так как его цель состоит в поиске оптимального сочетания весов.

Важнейшим элементом алгоритма градиентного спуска является правило, которое определяет, как нужно обновлять веса на каждой итерации. Чтобы понять, как оно описывается, нужно сначала отметить, что поверхность ошибок состоит из нескольких градиентов. В нашем простом примере она сформирована на основе двух кривых. Одна кривая определяется зависимостью SSE от w_0 (см. верхний график на рис. 6.3). Другая определяется зависимостью SSE от w_1 (см. средний график на рис. 6.3). Обратите внимание на то, как изменяется градиент вдоль этих кривых: например, кривая ошибок w_0 имеет крутые градиенты на левом и правом концах графика, но ближе к центру изменения они становятся более плавными. Кроме того, градиенты двух разных прямых могут существенно отличаться; в этом конкретном примере кривая ошибок w_0 в целом имеет куда более крутой градиент, чем w_1.

Тот факт, что поверхность ошибок состоит из разных кривых, каждая из которых обладает собственным градиентом, важен, так как алгоритм градиентного спуска продвигается вниз путем независимого обновления отдельных весов, стремясь уменьшить высоту на кривой ошибок, связанных с соответствующим весом. Иными словами, в ходе одной итерации алгоритма градиентного спуска w_0 обновляется для продвижения вниз по кривой ошибок w_0, а обновление w_1 направлено на то, чтобы спуститься по кривой ошибок w_1. Более того, величина, на которую изменяется каждый вес в ходе одной итерации, зависит от крутизны градиента его кривой ошибок, и этот градиент варьируется от одной итерации к другой по мере спуска. Например, w_0 относительно существенно обновляется на итерациях, в которых процесс поиска находится высоко по любую сторону от w_0; но ближе к нижней точке кривой ошибок w_0 обновления уменьшаются.

Кривая ошибок, связанная с каждым весом, определяется зависимостью SSE от величины обновления этого веса. Математический анализ и, в частности, дифференцирование — это раздел математики, который имеет дело со скоростью изменений. Например, для получения производной функции $y = f(x)$ нужно вычислить, насколько быстро y (вывод) меняется в зависимости от x (ввода). Более того, если функция принимает несколько входных значений $[y = f(x_1,..., x_n)]$, мы можем посчитать скорость изменения ее вывода, y, в зависимости от изменения каждого из них, x_i; для этого в каждой входной точке берется частная производная. Частная производная функция

по отношению к отдельному входному значению вычисляется в два этапа: сначала предполагается, что все остальные входные значения остаются неизменными (то есть скорость их изменения равна 0, что исключает их из уравнения), а затем берется производная от того, что осталось. Наконец, скорость изменения функции для заданного ввода называют также градиентом функции в том месте кривой (которая определяется функцией), где этот ввод находится. Следовательно, частная производная SSE по весу определяет то, как меняется вывод SSE в зависимости от веса — то есть градиент кривой ошибок этого веса. Это именно то, что нужно для определения правила обновления весов методом градиентного спуска: частная производная SSE по весу определяет, как вычисляется градиент кривой ошибок веса, а этот градиент, в свою очередь, говорит о том, насколько нужно обновить вес, чтобы уменьшить отклонение (вывод SSE).

Частная производная функция по отдельной переменной подразумевает, что все остальные переменные остаются неизменными. В результате функция имеет разные частные производные по каждой переменной, так как при их вычислении фиксируются разные члены. Поэтому для каждого веса существует своя частная производная SSE, хотя все они имеют похожую форму. Вот почему каждый вес в алгоритме градиентного спуска обновляется по отдельности: правило обновления веса зависит от частной производной SEE по этому весу, и, поскольку у каждого веса своя производная, правила обновления у них тоже разные. И опять, несмотря на эти отличия, все производные имеют одинаковую форму, поэтому правила обновления весов будут иметь аналогичный вид. Это упрощает определение алгоритма градиентного спуска. Еще одна особенность, которая делает этот процесс более простым, состоит в том, что SSE вычисляется относительно набора данных с n образцами. Это важно, поскольку переменными в SSE выступают только веса; целевой вывод y и ввод x определяются в наборе данных для каждого образца, поэтому их можно считать константами. Благодаря этому при вычислении частной производной SEE по весу многие члены уравнения, которые не включают в себя веса, можно отбросить, так как мы считаем, что они не меняются.

Отношение между выводом SSE и каждым весом становится более прозрачным, если в определении SSE вместо члена $\hat{y}_j$, обозначающего результат, предсказанный моделью, подставить структуру модели, генерирующей это предсказание. Например, если взять модель с одним вводом x_1 и фиктивным входным значением $x_0 = 1$, эта переписанная версия SSE будет выглядеть так:

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \left(w_0 \times x_{j,0} + w_1 \times x_{j,1}\right)\right)^2$$

В этом уравнении входные переменные имеют двойной индекс: первая его часть, j, обозначает образец (или строку в наборе данных), а второй определяет признак (или столбец в наборе данных) ввода. Например, $x_{j,1}$ относится к признаку 1 в образце j. Это определение SSE можно обобщить до модели с *m* входными значениями:

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \left(\sum_{i=0}^{m} w_i \times x_{j,i}\right)\right)^2$$

Вычисление частной производной SSE по определенному весу подразумевает дифференцирование сложной функции и применение ряда стандартных элементов таблицы производных. Результатом этих операций является следующее уравнение (для простоты мы снова обозначили вывод модели как y_i):

$$\frac{\partial\, SSE}{\partial\, w_i} = \sum_{j=1}^{n}\left(\underbrace{(y_j - \hat{y}_j)}_{\substack{\text{отклонение}\\ \text{вывода}\\ \text{взвешенной}\\ \text{суммы}}} \times \underbrace{-x_{j,i}}_{\substack{\text{скорость}\\ \text{изменения}\\ \text{взвешенной}\\ \text{суммы}\\ \text{в зависимости от } w_i}}\right)$$

Эта частная производная определяет, как вычислить градиент ошибок для веса w_i в наборе данных, в котором $x_{j,i}$ — это ввод, относящийся к весу w_i при обработке каждого образца в наборе. В ходе этого процесса умножаются два члена: отклонение вывода и скорость изменения вывода (то есть взвешенная сумма) в зависимости от веса. На эту формулу можно посмотреть следующим образом: если при изменении веса существенно меняется вывод взвешенной суммы, градиент ошибок по отношению к весу будет большим (резким), так как изменение веса сильно влияет на отклонение. Однако этот градиент является восходящим, а мы хотим двигаться вниз по кривой ошибок. Поэтому в правиле обновления весов методом градиентного спуска (представленном ниже) знак «–» перед вводом $x_{j,i}$ опускается. Если пометить итерацию алгоритма (отдельное прохождение по n образцам набора данных) символом t, это правило можно определить так:

$$w_i^{t+1} = w_i^t + \left(\underbrace{\eta \times \sum_{j=1}^{n}\left(\left(y_j^t - \hat{y}_j^t\right) \times x_{j,i}^t\right)}_{\text{error gradient for } w_i}\right)$$

У этого правила обновления весов есть ряд интересных особенностей. Во-первых, оно определяет, как вес w_i должен обновляться после t-го прохождения по набору данных. Это обновление прямо пропорционально градиенту кривой ошибок для этого веса на этой итерации (то есть сложению, которое на самом деле определяет частную производную SSE по этому весу). Во-вторых, с помощью этого правила можно обновлять веса для функций с несколькими входными значениями. Это означает, что алгоритм градиентного спуска подходит для прохождения вниз по поверхностям ошибок с более чем двумя координатами весов. Такие поверхности невозможно изобразить, поскольку у них больше трех измерений, но основные принципы спуска по поверхности ошибок с помощью градиента можно обобщить в виде алгоритма обучения с несколькими вводами. В-третьих, несмотря на то, что правило обновления весов имеет похожую структуру для каждого веса, оно определяет разные изменения на каждой итерации, так как зависит от входных значений образцов набора данных, к которым применяются веса. В-четвертых, операция сложения говорит о том, что на каждой итерации алгоритма градиентного спуска текущую модель нужно применять ко всем n образцам в наборе данных. Это одна из причин, почему обучение глубокой сети требует так много вычислительных ресурсов. Обычно крупные наборы данных разбиваются на части, и каждая итерация обучения основывается на отдельной части, а не на всем наборе. И последнее, на что следует обратить внимание: не считая операции сложения, это правило идентично алгоритму LMS (известному также как дельта-правило Уидроу — Хоффа), с которым мы познакомились в главе 4. Оно реализует ту же логику: если вывод модели слишком большой, веса, относящиеся к положительному вводу, нужно уменьшить; если вывод модели слишком маленький, эти веса следует увеличить. Более того, гиперпараметр скорости обучения имеет то же назначение, что и в правиле LMS, — масштабирование величины обновления весов для того, чтобы алгоритм не пропустил (перешагнул) лучшее сочетание весов. Если использовать это правило, алгоритм градиентного спуска можно кратко описать следующим образом:

1. Сформировать модель на основе начального набора весов.
2. Повторять, пока производительность модели не станет достаточно хорошей.
   - Применить текущую модель к образцам в наборе данных.
   - Откорректировать веса с помощью правила обновления весов.
3. Вернуть итоговую модель.

Одним из следствий независимого обновления весов и того факта, что величина обновления прямо пропорциональна локальному градиенту на соответствующей кривой ошибок, является то, что путь градиентного спуска к самой низкой точке на поверхности ошибок может оказаться извилистым. Дело в том, что градиенты каждой составной кривой ошибок могут не совпадать во всех точках поверхности ошибок (у одного веса градиент может быть более крутым, чем у другого). В результате во время определенной итерации один вес может обновиться сильнее, чем другой, и из-за этого спуск ко дну может не быть прямым. Это явление проиллюстрировано на рис. 6.4. Вы можете видеть несколько представлений (вид сверху) участка линий уровня функции (изолиний) на поверхности ошибок. Эта вогнутая поверхность достаточно длинная и узкая; чем дальше от краев, тем более пологим становится спуск. Расстояние между изолиниями отражает степень наклона. Вследствие этого поиск сначала идет поперек вогнутой поверхности и только потом поворачивает по направлению к ее центру. На диаграмме слева показана первая итерация алгоритма градиентного спуска. Начальная точка находится в месте встречи трех стрелок. Длина пунктирной и точечной стрелок представляет локальные градиенты кривых ошибок — w_0 и, соответственно, w_1. Пунктирная стрелка длиннее; это показывает, что локальный градиент кривой ошибок w_0 является более крутым, чем градиент кривой ошибок w_1. На каждой итерации каждый отдельный вес обновляется прямо пропорционально градиенту соответствующей кривой ошибок; таким образом, на первой итерации обновление для w_0 больше, чем для w_1, поэтому в целом поиск проходит большей частью поперек поверхности, а не вдоль нее. Жирная черная стрелка обозначает общее перемещение по исходному пространству весов в результате обновления этих весов в ходе первой итерации. Точно так же на средней диаграмме показаны градиенты ошибок и общее обновление весов на следующей итерации градиентного спуска. Справа вы можете видеть весь путь, который проходит процесс поиска от начальной точки к глобальному минимуму (самой низкой точке на поверхности ошибок).

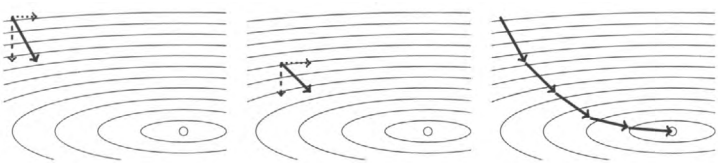


Рис. 6.4. Вид сверху на изолинии поверхности ошибок, иллюстрирующий путь градиентного спуска по этой поверхности. Жирные стрелки обозначают общее перемещение вектора весов в ходе одной итерации алгоритма градиентного спуска. Длина точечной и пунктирной стрелок представляет локальные градиенты кривых ошибок w_0 и w_1, соответственно, на этой же итерации. Справа показан общий путь, проделанный к глобальному минимуму на поверхности ошибок

Правило обновления весов довольно легко применить к обучению отдельного нейрона. В этом случае w_0 — это сдвиг нейрона, а остальные веса относятся к другим его входным значениям. Вычисление частной производной SSE зависит от структуры функции, которая генерирует $\hat{y}$. Чем сложнее эта функция, тем сложнее становится частная производная. Тот факт, что функция, которую определяет нейрон, включает в себя как взвешенную сумму, так и функцию активации, означает: частная производная SSE по весу нейрона будет сложнее частной производной, приведенной выше. Наличие функции активации добавляет в нее еще один член — частную производную функцию активации по выводу функции взвешенной суммы. Это вызвано тем, что данный вывод подается на вход функции активации. Последняя не принимает вес напрямую. Изменения веса влияют на ее вывод опосредованно, через обновление взвешенной суммы. Логистическое уравнение так широко и долго применялось в качестве функции активации в нейронных сетях именно потому, что оно имеет очень простую производную по своим входным значениям. Правило обновления весов методом градиентного спуска для нейрона с логистической функцией выглядит так:

$$w_i^{t+1} = w_i^t + \underbrace{\left(\eta \times \sum_{j=1}^{n} \underbrace{\left(y_j^t - \hat{y}_j^t\right) \times \underbrace{\left(\hat{y}_j^t \times \left(1 - \hat{y}_j^t\right)\right)}_{\substack{\text{производная} \\ \text{логистической функции} \\ \text{по взвешенной сумме}}} \times x_{j,i}^t}_{\text{градиент ошибок для } w_i} \right)}$$

То, что правило обновления весов включает производную функцию активации, означает: оно будет меняться вместе с этой функцией. Однако это изменение будет заключаться всего лишь в обновлении ее производной; общая структура правила останется неизменной.

Это расширенное правило обновления весов показывает, что алгоритм градиентного спуска можно использовать для обучения отдельного нейрона. При этом с его помощью нельзя обучать многослойные нейронные сети, так как определение градиента ошибок для отдельно взятого веса зависит от погрешности функции (член $y_j - \hat{y}_j$). И хотя мы можем вычислить отклонение вывода нейрона в выходном слое сети, сравнив напрямую этот вывод с ожидаемым результатом, это невозможно сделать для нейрона в скрытом слое.

Поэтому мы не можем определить градиенты ошибок для каждого веса. Для решения этой проблемы можно воспользоваться алгоритмом обратного распространения.

Обучение нейронной сети с использованием обратного распространения

У термина «обратное распространение» есть два разных значения. Первое и главное состоит в том, что это алгоритм, с помощью которого можно вычислить чувствительность (градиент, или скорость изменения) отклонения сети к изменениям весов каждого ее нейрона. Получив градиент ошибок для веса, мы можем отрегулировать этот вес, чтобы уменьшить общее отклонение сети. Для этого используется правило обновления весов, похожее на то, которое основано на градиентном спуске. В этом смысле алгоритм обратного распространения является решением задачи присваивания коэффициентов доверия, с которой мы познакомились в главе 4. В то же время он выступает полноценным алгоритмом для обучения нейронной сети. Это второе значение вбирает в себя первое, но также включает правило обучения, определяющее, как следует использовать градиенты ошибок для обновления весов внутри сети. Следовательно, согласно этому второму значению, данный алгоритм состоит из двух этапов: сначала он решает задачу присваивания коэффициентов доверия, а затем использует градиенты ошибок, вычисленные в ходе решения этой задачи, для обновления весов в сети. Эти два значения следует различать, поскольку существует целый ряд разных правил обучения, которые позволяют обновлять веса после назначения коэффициентов доверия. Самым распространенным правилом обучения, которое применяется при обратном распространении, является алгоритм градиентного спуска, представленный ранее. Здесь мы уделяем основное внимание первому значению, согласно которому данный алгоритм решает задачу присваивания коэффициентов доверия.

Обратное распространение: двухэтапный алгоритм

Алгоритм обратного распространения начинается с инициализации всех весов сети с помощью произвольных значений. Стоит отметить, что даже сеть, инициализированная случайным образом, способна генерировать вывод в ответ на поступающий ввод, хотя отклонение этого вывода, скорее всего, будет большим. После инициализации можно приступить к обучению сети, постепенно обновляя ее веса так, чтобы уменьшить отклонение; последнее

вычисляется как разница между выводом, который сеть генерирует в ответ на входную закономерность, и ожидаемым выводом, определенным в учебном наборе данных. Ключевым аспектом этого процесса является решение задачи присваивания коэффициентов доверия — то есть вычисление градиентов ошибок для каждого веса в сети. Алгоритм обратного распространения делает это в два этапа. В ходе первого, известного как прямой проход, на вход сети подается закономерность, и активации нейронов двигаются вперед, пока не будет сгенерирован вывод. Это проиллюстрировано на рис. 6.5. Здесь вы можете видеть взвешенную сумму входных значений, вычисляемую для каждого нейрона (например, z_1 представляет взвешенную сумму вводов для нейрона 1), и получаемый при этом вывод (или активации; например, a_1 обозначает активацию нейрона 1). Присутствие значений z_i и a_i на этой диаграмме подчеркивает тот факт, что в ходе прямого прохождения оба эти значения (для каждого нейрона) сохраняются в памяти. Это делается для того, чтобы позже использовать их в обратном прохождении алгоритма. С помощью значения z_i вычисляется величина изменения весов для входных соединений нейрона. А значение a_i позволяет посчитать величину изменения весов для выходных соединений. Подробности применения этих значений в ходе обратного прохождения будут описаны ниже.

Второй этап, известный как обратное прохождение, начинается с вычисления градиента ошибок для каждого нейрона в выходном слое. Эти градиенты представляют, насколько сильно отклоняется сеть в ответ на изменения во взвешенной сумме нейрона, и в качестве их сокращенного обозначения часто используют символ δ (произносится как «дельта») с индексом, обозначающим нейрон. Например, δ_k — это градиент ошибок сети в ответ на незначительные изменения во взвешенной сумме нейрона k. Необходимо понимать, что в алгоритме обратного распространения вычисляются два разных градиента ошибок:

- Первый — это значение δ для каждого нейрона, представляющее собой скорость изменения погрешности сети в ответ на обновления взвешенной суммы нейрона. У каждого нейрона своя дельта. Именно эти градиенты ошибок алгоритм распространяет в обратном направлении.
- Второй — это градиент ошибок сети с учетом изменения ее весов. У каждого веса в сети есть один такой градиент. Эти градиенты используются для обновления весов. Но, прежде чем их вычислять, необходимо сначала получить δ каждого нейрона (с помощью обратного распространения).

Обратите внимание на то, что у каждого нейрона есть только одна дельта, но при этом у него может быть много разных весов. Таким образом, значение δ можно применять для вычисления сразу нескольких градиентов ошибок.

После получения дельт для выходных нейронов вычисляются значения δ для нейронов в последнем скрытом слое. Для этого часть дельты каждого выходного нейрона присваивается каждому скрытому нейрону, который с ним соединен. Это коэффициенты доверия, которые зависят от весов на соединениях между выходным и скрытым нейронами, а также от активации скрытого нейрона во время прямого прохождения (вот почему они записываются в память). После назначения коэффициентов доверия вычисляется дельта каждого нейрона в последнем скрытом слое; для этого слагаются части дельт, присвоенные нейрону всеми выходными нейронами, с которыми он соединен. Затем тот же процесс повторяется для передачи градиента ошибок из последнего скрытого слоя в предпоследний и дальше, вплоть до входного слоя. Именно из-за этого обратного распространения дельт по сети данный алгоритм и получил свое название. В конце обратного прохода для каждого нейрона сети вычисляется δ (то есть задача присваивания коэффициентов доверия решена), и затем эти значения используются для обновления весов сети (например, с помощью алгоритма градиентного спуска, описанного ранее). Обратный проход алгоритма проиллюстрирован на рис. 6.6. Как видите, чем дальше процесс обратного распространения уходит от выходного слоя, тем меньше становятся дельты. Это следствие проблемы исчезающего градиента (см. главу 4), которая замедляет темпы обучения начальных слоев сети.

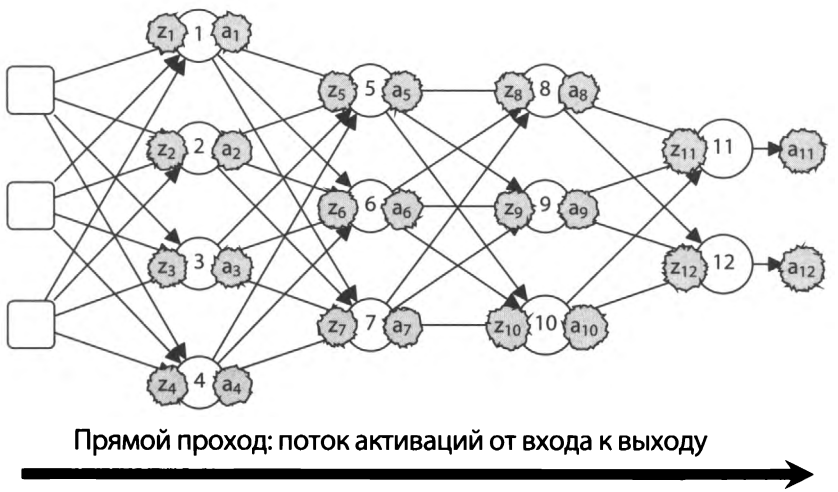


Рис. 6.5. Прямой проход алгоритма обратного распространения

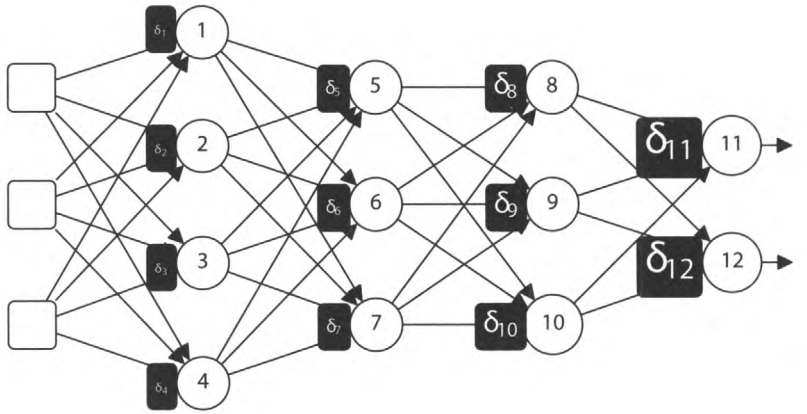


Рис. 6.6. Обратный проход алгоритма обратного распространения

Подведем итоги. Каждая итерация алгоритма обратного распространения состоит из следующих основных шагов:

- Сеть принимает ввод и позволяет активациям нейронов проходить прямо, пока не будет сгенерирован вывод. При этом записывается как взвешенная сумма, так и активация каждого нейрона.
- Для каждого нейрона в выходном слое вычисляется градиент ошибок (δ или дельта).
- Градиенты ошибок передаются в обратном направлении, чтобы получить дельту каждого нейрона в сети.
- Значения δ и алгоритм обновления весов (например, алгоритм градиентного спуска) используются, чтобы вычислить градиенты ошибок для весов сети. Затем происходит обновление весов с помощью этих градиентов.

Алгоритм продолжает повторять эти шаги до тех пор, пока отклонение сети не достигнет приемлемого уровня (или пока сеть не сойдется).

Обратное распространение: передача дельт в обратном направлении

Член δ в уравнении нейрона представляет градиент ошибок сети относительно изменения взвешенной суммы входных значений, вычисляемой этим нейроном. Для более детализированного представления в верхней части рис. 6.7 показаны вычислительные этапы внутри нейрона k; z_k обозначает величину

соответствующей взвешенной суммы. На этой диаграмме нейрон принимает ввод (или активации) от трех других нейронов (h, i, j), и z_k — это взвешенная сумма их активаций. Затем, чтобы вычислить вывод нейрона, a_k, значение z_k пропускается через нелинейную функцию активации, ϕ, такую как логистическое уравнение. Таким образом, δ в нейроне k — это скорость изменения отклонения сети в ответ на небольшие колебания значения z_k. В математическом смысле это частная производная отклонения сети относительно z_k:

$$\delta_k = \frac{\partial \text{Error}}{\partial z_k}$$

В каком бы месте сети ни находился нейрон (в выходном или скрытом слое), его дельта является произведением двух значений:

- скорости изменения отклонения сети в ответ на колебания активации (вывода) нейрона: $\partial E / \partial a_k$;
- скорости изменения активации нейрона в ответ на обновление взвешенной суммы входных значений нейрона: $\partial a_k / \partial z_k$;

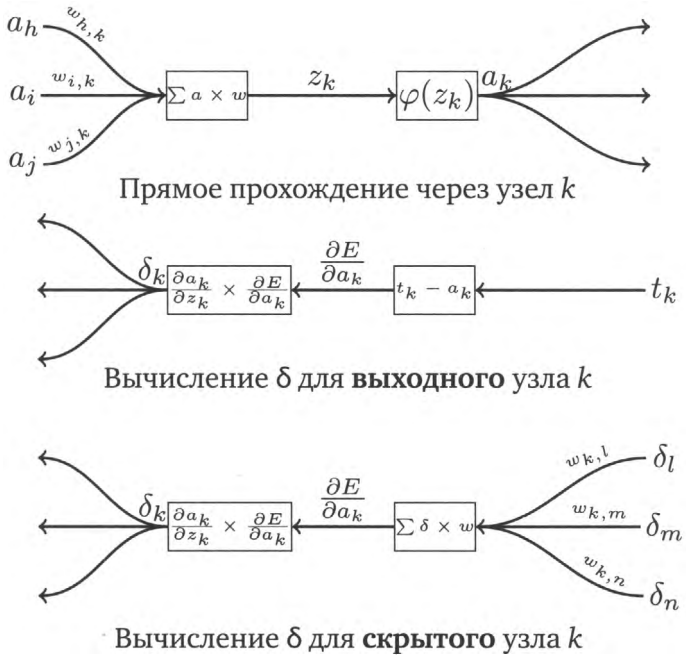


Рис. 6.7. Вверху: прямое прохождение активаций через взвешенную сумму и функцию активации нейрона. Посредине: вычисление значения δ для выходного нейрона (t_k —ожидаемая активация нейрона, a_k — фактическая). Внизу: Вычисление значения δ для скрытого нейрона. Эта диаграмма отчасти создана под влиянием рис. 5.2 и 5.3 из работы: Рид и Маркс II, 1999

$$\delta_k = \frac{\partial E}{\partial a_k} \times \frac{\partial a_k}{\partial z_k}$$

В средней части рис. 6.7 показано, как это произведение вычисляется для нейронов в выходном слое сети. Сначала нужно посчитать скорость изменения отклонения сети по отношению к выводу нейрона ($\partial E / \partial a_k$). Чем больше разница между фактической и ожидаемой активациями нейрона (a_k и t_k), тем быстрее будет меняться отклонение при обновлении вывода нейрона. Поэтому скорость изменения отклонения сети с учетом колебаний активации выходного нейрона k можно посчитать, вычитая фактический вывод нейрона (a_k) из ожидаемого (t_k):

$$\frac{\partial E}{\partial a_k} = t_k - a_k$$

Это значение привязывает отклонение сети к выводу нейрона. Однако дельта нейрона (δ) — это скорость изменения отклонения в ответ на ввод функции активации (z_k), а не на ее вывод (a_k). Следовательно, чтобы получить δ нейрона, значение $\partial E / \partial a_k$ должно пройти обратно через функцию активации. Для этого $\partial E / \partial a_k$ следует умножить на скорость изменения функции активации по отношению к ее входному значению, z_k. На рис. 6.7 скорость изменения функции активации относительно ее ввода обозначена как $\partial a_k / \partial z_k$. Для вычисления этого отношения значение z_k (сохраненное в результате прямого прохождения по сети) подставляется в уравнение производной функции активации относительно z_k. Например, производная логистического уравнения по отношению к его вводу выглядит так:

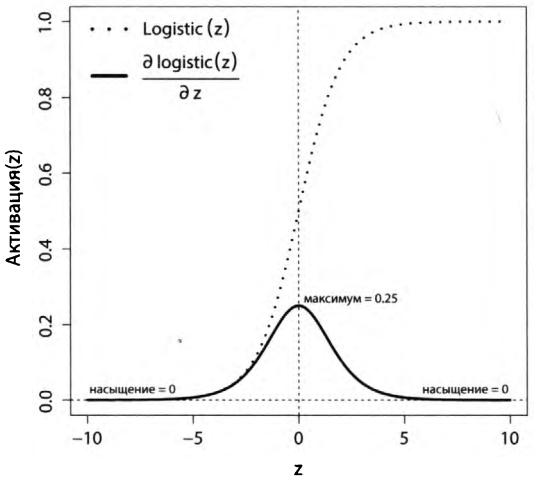
$$\frac{\partial \text{logistic}(z)}{\partial z} = \text{logistic}(z) \times (1 - \text{logistic}(z))$$

На рис. 6.8* изображен график этой функции. Как видите, в результате подстановки z_k в это уравнение получается значение в диапазоне между 0 и 0,25. Например, если $z_k = 0$, $\partial a_k / \partial z_k = 0{,}25$. Вот почему результат взвешенной суммы для каждого нейрона (z_k) сохраняется во время прямого прохождения алгоритма.

Поскольку для получения дельты нейрона нужно выполнить произведение, которое включает в себя производную его функции активации, необходимо, чтобы эта производная поддавалась вычислению. Мы не можем взять производную от пороговой функции активации, поскольку в точке порога

* Эта диаграмма также приводится в главе 4. Она повторяется для удобства.

присутствует разрыв. В итоге алгоритм обратного распространения не подходит для сетей, нейроны которых активируются с помощью пороговых функций. Это одна из причин, почему вместо этих функций в нейронных сетях начали использовать логистическое уравнение и tanh — обе эти функции имеют очень простые производные, поэтому отлично подходят для обратного распространения.



В нижней части рис. 6.7 проиллюстрирован процесс вычисления δ для нейрона в скрытом слое. Здесь применяется произведение тех же значений, что и в выходных нейронах. Отличие заключается в том, что $\partial E / \partial a_k$ сложнее вычислить для скрытых модулей. Мы не можем привязать вывод скрытого нейрона непосредственно к отклонению сети, так как эти две величины связаны лишь косвенно. Эта связь выражается в изменении активации последующих нейронов, которые принимают на вход вывод скрытых нейронов, и величина этих изменений зависит от веса, который применяется к выводу каждым последующим нейроном. К тому же это косвенное влияние на отклонение сети зависит от того, как на данное отклонение воздействуют последующие нейроны (то есть их значения δ). Следовательно, чувствительность отклонения сети к выводу скрытого нейрона можно вычислить как взвешенную сумму значений δ тех нейронов, которые находятся сразу за ним:

$$\frac{\partial E}{\partial a_k} = \sum_{i=1}^{N} w_{k,i} \times \delta_i$$

Таким образом, прежде чем мы сможем получить $\partial E / \partial a_k$ для нейрона k, необходимо вычислить отклонения (значения δ) всех последующих нейронов,

которым тот передает свой вывод. Но это не составляет труда, поскольку мы движемся в обратном направлении, поэтому значения δ последующих нейронов будут получены еще до достижения нейрона k.

Для скрытых нейронов вычисляется другой член произведения δ, $\partial a_k / \partial z_k$. Это делается точно так же, как и в случае с выходными нейронами: значение z_k (взвешенная сумма входных значений нейрона, сохраненная в ходе прямого прохождения по сети) подставляется в производную функции активации нейрона относительно z_k.

Обратное распространение: обновление весов

Основной принцип обновления весов, который используется в алгоритме обратного распространения, состоит в том, что каждый вес в сети должен изменяться прямо пропорционально чувствительности общего отклонения сети к изменениям этого веса. Очевидно, что если изменение веса не влияет на общее отклонение сети, эти две величины являются независимыми, поэтому вес не сказывается на величине отклонения. Чтобы определить зависимость отклонения сети от изменения отдельных весов, следует измерять скорость его изменения в ответ на обновления веса.

Общее отклонение сети — это функция с несколькими входными значениями, в число которых входит как ввод сети, так и все ее веса. Поэтому скорость изменения этой величины в ответ на обновление заданного веса вычисляется с помощью частной производной отклонения по этому весу. В алгоритме обратного распространения для вычисления этой частной производной используется дифференцирование. Например, если взять вес $w_{j,k}$ на соединении между нейронами j и k, частная производная отклонения сети будет представлять собой произведение двух значений:

1) скорости изменения взвешенной суммы вводов нейрона k в зависимости от обновления веса $\partial z_k / \partial w_{j,k}$;
2) скорости изменения отклонения сети в ответ на изменение взвешенной суммы вводов, вычисленной нейроном k (этот второй множитель обозначается как δ_k).

На рис. 6.9 показано, как произведение этих двух значений привязывает вес к отклонению вывода сети. У последних двух нейронов (k и l) единый путь активации. Нейрон k принимает одно входное значение, a_j, а его вывод является единственным вводом нейрона l. Вывод нейрона l составляет общий вывод сети. В этом участке сети есть два веса, $w_{j,k}$ и $w_{k,l}$.

Фундаментальный принцип обновления весов, который используется в алгоритме обратного распространения, состоит в том, что каждый вес в сети должен быть изменен прямо пропорционально чувствительности общего отклонения сети к изменениям этого веса.

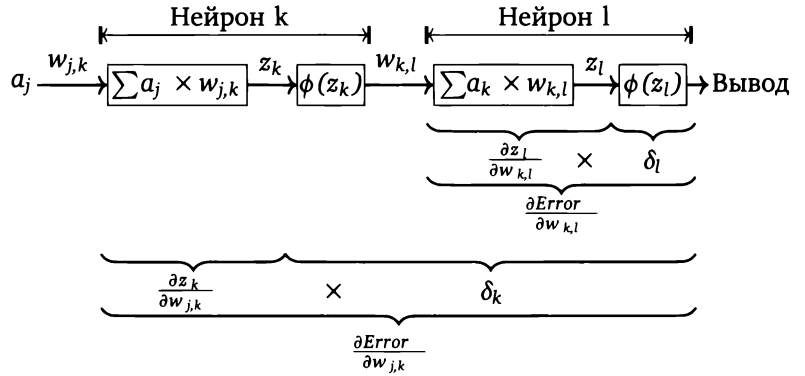


Рис. 6.9. Иллюстрация того, как произведение производных связывает веса сети и ее отклонение

Вычисления, представленные на рис. 6.9, могут показаться сложными, так как состоят из нескольких разных элементов. Но, как вы вскоре увидите, каждый из них в отдельности вычисляется довольно легко; самое сложное здесь — не запутаться в порядке проведения вычислений.

Давайте сосредоточимся на $w_{k,l}$. Этот вес применяется к вводу выходного нейрона. Между ним и выводом (отклонением) сети находятся два этапа: взвешенная сумма, вычисляемая в нейроне l, и нелинейная функция, применяемая к этой сумме функцией активации нейрона l. Если начать с вывода и двигаться в обратном направлении, δ_l вычисляется с помощью уравнения, показанного на рис. 6.7: разница между целевой и фактической активациями нейрона умножается на частную производную функции активации этого нейрона по ее вводу (взвешенной сумме z_k), $\partial a_l / \partial z_l$. Если предположить, что в качестве функции активации нейрон l использует логистическое уравнение, для получения $\partial a_l / \partial z_l$ мы подставляем значение z_l (сохраненное во время прямого прохождения алгоритма) в производную логистической функции:

$$\frac{\partial a_l}{\partial z_l} = \frac{\partial \text{logistic}(z_l)}{\partial z_l} = \text{logistic}(z_l) \times (1 - \text{logistic}(z_l))$$

Таким образом, если исходить из того, что нейрон l использует логистическую функцию, формула δ_l выглядит так:

$$\delta_l = \text{logistic}(z_l) \times (1 - \text{logistic}(z_l)) \times (t_l - a_l)$$

Дельта δ_l привязывает отклонение сети к функции активации (взвешенной сумме z_l). Однако нам хотелось бы вновь привязать отклонение сети к весу $w_{k,l}$. Для этого δ_l нужно умножить на частную производную функции

взвешенной суммы по $w_{k,l}$: $\partial z_k / \partial w_{j,k}$. Эта частная производная описывает, как изменение $w_{k,l}$ влияет на вывод функции взвешенной суммы z_l. Поскольку это линейная функция от весов и активаций, следовательно, в частной производной по отдельному весу все члены, не зависящие от этого веса, обнуляются (то есть считаются константами). В результате остается всего лишь один ввод, связанный с этим весом, — в данном случае a_k.

$$\frac{\partial z_l}{\partial w_{k,l}} = a_k$$

Вот почему активации каждого нейрона сети сохраняются во время прямого прохождения. Вместе эти два выражения, $\partial z_k / \partial w_{j,k}$ и δ_l, привязывают $w_{k,l}$ к отклонению сети: сначала вес связывается с z_l, а затем z_l с функцией активации нейрона и, следовательно, с отклонением. Таким образом, градиент ошибок сети по отношению к изменениям веса $w_{k,l}$ вычисляется так:

$$\frac{\partial Error}{\partial w_{k,l}} = \frac{\partial z_l}{\partial w_{k,l}} \times \delta_l = a_k \times \delta_l$$

На рис. 6.9 показан еще один вес, $w_{k,l}$, но он находится на более глубоком уровне сети, поэтому от итогового вывода (и отклонения) его отделяет больше вычислительных шагов. На этапе обратного распространения (нижняя часть рис. 6.7) значение δ нейрона k вычисляется с помощью следующего произведения:

$$\delta_k = \frac{\partial a_k}{\partial z_k} \times (w_{k,l} \times \delta_l)$$

Если предположить, что в качестве функции активации нейрон k использует логистическое уравнение, отношение $\partial a_k / \partial z_k$ вычисляется аналогично $\partial a_l / \partial z_l$: значение z_k подставляется в уравнение производной логистической функции. Поэтому в развернутом виде вычисление δ_k будет выглядеть так:

$$\delta_k = \text{logistic}(z_k) \times (1 - \text{logistic}(z_k)) \times (w_{k,l} \times \delta_l)$$

Но, чтобы привязать $w_{j,k}$ к отклонению сети, δ_k следует умножить на частную производную функции взвешенной суммы по весу: $\partial z_k / \partial w_{j,k}$. Как уже описывалось выше, эта производная сокращает ввод, связанный с $w_{j,k}$ (то есть a_j), а градиент отклонения сети по отношению к скрытому весу $w_{j,k}$ вычисляется путем умножения a_j на δ_k. Следовательно, произведение этих членов ($\partial z_k / \partial w_{j,k}$ и δ_k) создает цепочку, соединяющую вес $w_{j,k}$ и отклонение сети. В итоге, если

нейрон использует логистическую функцию активации, произведение членов для $w_{j,k}$ будет таким:

$$\frac{\partial \text{Error}}{\partial w_{j,k}} = \frac{\partial z_k}{\partial w_{j,k}} \times \delta_k = a_j \times \delta_k$$

Здесь были рассмотрены вычисления в контексте очень простой сети с единственным маршрутом соединений. Однако они применимы и к более сложным сетям, поскольку вычисление дельт δ для скрытых модулей уже учитывает, что из нейрона может исходить множество соединений. Получив градиент отклонения сети по отношению к весу ($\partial \text{Error} / w_{j,k} = \delta_k \times a_j$), мы можем откорректировать этот вес для уменьшения веса сети, используя правило обновления весов методом градиентного спуска. Это правило показано ниже. В нем используются обозначения из обратного прохождения; вес находится на соединении между нейронами j и k во время итерации алгоритма t:

$$w_{j,k}^{t+1} = w_{j,k}^{t} + \left(\eta \times \delta_k \times a_j\right)$$

Наконец, необходимо отметить, что важной проблемой при обучении нейронных сетей с помощью обратного распространения и градиентного спуска является существенное усложнение поверхности ошибок по сравнению с линейными моделями. Как видно на рис. 6.3, поверхность ошибок линейной модели имеет вид гладкой вогнутой чаши с одним глобальным минимумом (одно оптимальное сочетание весов). Однако поверхность ошибок нейронной сети больше напоминает горный хребет с несколькими впадинами и пиками. Это вызвано тем, что каждый нейрон сети при связывании ввода и вывода использует нелинейную функцию, поэтому функция, реализуемая нейронной сетью, тоже является нелинейной. Эта нелинейность повышает выразительную силу сети с точки зрения того, насколько сложные функции она способна усваивать. Но ценой этого является более сложная поверхность ошибок, поэтому сочетание весов, составляющее ее минимум, не всегда можно найти с помощью алгоритма градиентного спуска: мы можем попросту застрять в локальном минимуме. Но, к счастью, обратное распространение и градиентный спуск часто находят сочетания весов, которые позволяют определять полезные модели, хотя для этого может потребоваться многократное выполнение процесса обучения, чтобы сеть исследовала разные участки поверхности ошибок.

ГЛАВА 7

БУДУЩЕЕ ГЛУБОКОГО ОБУЧЕНИЯ

27 марта 2019 года Джошуа Бенжио, Джефри Хинтон и Ян Лекун получили премию Тьюринга. Этой наградой был отмечен их вклад в становление глубокого обучения в качестве ключевой технологии в ходе современной революции в сфере искусственного интеллекта. Премию Тьюринга часто называют «Нобелевской премией в области информационных технологий», и ее размер составляет 1 миллион долларов. Трое этих исследователей, работая на протяжении десятилетий то совместно, то по отдельности, а иногда и с другими своими коллегами, достигли многого в области глубокого обучения, начиная с популяризации обратного распространения в 1980-х годах и заканчивая разработкой сверточных нейронных сетей, векторного представления слов, механизмов внимания и генеративно-состязательных сетей (это далеко не полный список исследований). В сообщении о премии отмечались недавние потрясающие прорывы в компьютерном видении, робототехнике, распознавании речи и обработке естественного языка, к которым привело глубокое обучение, а также огромное влияние, которое эти технологии оказывают на общество. Миллиарды людей ежедневно используют искусственный интеллект на основе глубоких сетей с помощью мобильных приложений. Также подчеркивалось, что глубокое обучение дало ученым новые мощные инструменты, способствовавшие научным прорывам в самых разных областях, таких как медицина и астрономия. Вручение премии этим исследователям демонстрирует, насколько важным стало глубокое обучение для современной науки и общества. Существенное влияние, которое оно оказывает на технологии, будет только расти в ближайшие десятилетия по мере появления все больших наборов данных, разработки новых алгоритмов и усовершенствования аппаратного обеспечения. Эти направления развиваются, и то, как сообщество, сформировавшееся вокруг глубокого обучения, ими воспользуется, определит темпы развития и инноваций в данной области на протяжении следующих лет.

Большие данные способствуют алгоритмическим инновациям

В главе 1 мы познакомились с разными типами машинного обучения: с учителем, без учителя и с подкреплением. Большая часть этой книги посвящена обучению с учителем в основном потому, что это самый популярный метод. Однако его недостаток в том, что такое обучение может быть довольно затратным и требовать много времени на маркирование набора данных с помощью подходящих целевых меток. И поскольку данных становится все больше, расходы на их маркирование препятствуют разработке новых приложений. Набор данных ImageNet наглядно показывает, каков масштаб работ для назначения меток в проектах глубокого обучения. Этот набор был выпущен в 2010 году и лег в основу соревнования по широкомасштабному распознаванию образов ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), победителями в нем стали AlexNet CNN (в 2012 году) и система ResNet (в 2015 году). Как уже говорилось в главе 4, победа AlexNet породила всплеск энтузиазма вокруг моделей глубокого обучения. Но она была достигнута только благодаря набору данных ImageNet*. Этот набор содержит более четырнадцати миллионов образов, которые были помечены вручную, чтобы указать, какие объекты присутствуют на каждом из них, и более чем в миллионе образов объекты были заключены в ограничивающие параллелепипеды. Такое широкомасштабное маркирование данных требует значительных усилий и денежных вложений; оно стало возможным благодаря платформам краудсорсинга. Возможность создания таких промаркированных наборов данных существует далеко не всегда.

Одним из следствий этого стал растущий интерес к обучению без учителя. Модели автокодировщиков, которые использовались в предобучении Хинтона (см. главу 4), стали первым шагом к внедрению этой методики в нейронные сети, и в последние годы возникли самые разные типы моделей. Еще один шаг к решению этой проблемы — обучение генеративных моделей. Генеративные модели пытаются усваивать распределение данных (то есть моделировать процесс, который эти данные сгенерировал). Подобно автокодировщикам, генеративные модели часто применяются при формировании удобного представления данных перед началом обучения с учителем. Генеративно-состязательные сети (англ. generative adversarial networks, или GAN) — это метод обучения генеративных моделей, привлекающий большое внимание в последние годы (Гудфеллоу и др., 2014). В состав GAN входят две

* http://www.image-net.org/

По мере увеличения наборов данных расходы на их маркирование препятствуют разработке новых приложений.

нейронные сети, генеративная и дискриминационная модели, а также выборка реальных данных. Обучение моделей проходит в форме состязания. Задача дискриминационной модели — научиться отличать реальные данные, взятые из набора, от фиктивных, синтезированных с помощью генератора. Этот генератор, в свою очередь, учится синтезировать данные, которые могут обмануть дискриминационную модель. Генеративные модели, обучаемые с помощью GAN, способны синтезировать изображения, имитирующие определенный художественный стиль (Элгаммал и др., 2017), а также медицинские снимки с выделением патологий (Фрид-Адар и др., 2018). Синтез медицинских снимков с разметкой патологических участков дает возможность автоматически генерировать огромные промаркированные наборы данных, которые можно использовать в обучении с учителем. Более проблемной областью применения сетей GAN является генерация так называемых *глубоких подделок* (англ. deep fakes) — видеороликов, на которых человек совершает нечто, чего никогда не делал; для этого его лицо накладывается на видео с кем-то другим. Глубокие подделки очень сложно распознать. Их неоднократно использовали в ущерб репутации общественных деятелей и для распространения фальшивых новостей.

Еще одно решение проблемы с маркировкой данных состоит в перепрофилировании моделей, обученных на похожих задачах, вместо того чтобы создавать новые модели в каждом отдельном случае. Перепрофилирование обучения (англ. transfer learning) — это использование уже усвоенной информации (или представлений) для ускорения обучения какой-то другой задаче. Чтобы это работало, задачи должны быть из родственных предметных областей. Обработка изображений — это пример предметной области, в которой перепрофилирование обучения часто используется для повышения производительности моделей при выполнении целого ряда разных задач. Это вызвано тем, что низкоуровневые визуальные признаки, такие как грани, являются относительно стабильными и могут использоваться практически в любых визуальных категориях. Более того, тот факт, что модели CNN формируют иерархию визуальных признаков, а их начальные слои учатся распознавать в вводе эти низкоуровневые элементы, позволяет перепрофилировать эти слои предобученных сетей CNN для разных проектов по обработке изображений. Представьте, к примеру, проект, для которого нужна модель классификации образов, способная распознавать объекты из специализированных категорий, но для них нет образцов в общедоступных наборах данных вроде ImageNet. Вместо обучения новых моделей CNN с нуля довольно стандартным в наши дни является следующий процесс: загрузить ведущую на текущий момент

модель (такую как Microsoft ResNet), обученную на ImageNet, заменить ее глубокие слои и обучить эту новую гибридную модель на относительно небольшом наборе данных, промаркированном с использованием подходящих для проекта категорий. Замена глубоких слоев обусловлена тем, что в них содержатся функции, которые объединяют низкоуровневые признаки в категории, относящиеся к изначальной задаче модели. Благодаря тому, что начальные слои уже обучены распознавать низкоуровневые визуальные элементы, обучение ускоряется, а объем данных, необходимых для нового проекта, уменьшается.

Возросший интерес к обучению без учителя, генеративным моделям и перепрофилированию обучения можно считать откликом на проблему маркирования постоянно растущих наборов данных.

Появление новых моделей

С каждым годом количество новых моделей глубокого обучения растет. В качестве свежего примера можно привести капсульные сети (Хинтон и др., 2018; Сабур и др., 2017). Они были созданы, чтобы обойти некоторые ограничения, свойственные сетям CNN. Одна из проблем сверточных сетей состоит в том, что они игнорируют точные пространственные отношения между высокоуровневыми компонентами в структуре объекта; иногда ее называют проблемой Пикассо. На практике это означает, что сеть CNN, обученная идентифицировать лица, может научиться распознавать глаза, нос и рот, но не их расположение относительно друг друга. Следовательно, ее можно обмануть, подсунув изображение, на котором части лица находятся в неправильных местах. Эта проблема возникает из-за того, что слои пулинга в CNN отбрасывают позиционную информацию.

В основе капсульных сетей лежит мысль о том, что человеческий мозг учится распознавать типы объектов вне зависимости от их расположения в поле зрения. В сущности, каждый тип объектов принадлежит к классу с рядом параметров конкретизации. Этот класс кодирует такую информацию, как соотношения между разными частями объекта. Параметры конкретизации определяют то, как абстрактное описание типа объекта можно привязать к его конкретному экземпляру, находящемуся в поле зрения (например, его позу, масштаб и т. д.).

Капсула — это набор нейронов, которые учатся определять наличие объекта определенного типа или его части в заданном участке изображения. Ее

выводом является вектор активности, который представляет параметры конкретизации объекта, если тот присутствует в нужном месте. Капсулы встраиваются в сверточные слои. При этом капсульные сети заменяют пулинг (который зачастую определяет интерфейс между сверточными слоями) процессом под названием «динамическая маршрутизация». Суть этого процесса в том, что каждая капсула одного слоя сети учится предсказывать, какой капсуле в следующем слое лучше всего направить свой вектор вывода.

На момент написания этих строк капсульные сети демонстрируют непревзойденную производительность при распознавании написанных от руки цифр из набора данных MNIST, на котором обучались оригинальные сверточные сети. Однако по современным меркам это набор относительно небольшой, и капсульные сети еще не масштабировались для работы с более крупными объемами информации. Это частично вызвано тем, что процесс маршрутизации замедляет их обучение. Но, если эту проблему удастся преодолеть, мы можем получить новую важную разновидность моделей, существенно приближающую способ анализа образов в нейронных сетях к тому, который использует человек.

В последнее время большой интерес вызывает также модель «трансформер» (Васвани и др., 2017). Она демонстрирует рост тенденции в области глубокого обучения, состоящей в разработке моделей со сложными внутренними механизмами внимания, которые позволяют им динамически сосредотачиваться на подмножествах входных значений при генерации вывода. Модель «трансформер» демонстрирует беспрецедентные результаты в машинном переводе между некоторыми языковыми парами, и в будущем эта архитектура может заменить модель «кодировщик-декодер», описанную в главе 5. На ее основе была создана модель BERT (Bidirectional Encoder Representations from Transformers — двунаправленные представления кодировщика от Трансформеров; Девлин и др., 2018). Это крайне интересная разработка, так как в ее основе лежит идея перепрофилирования обучения (которая обсуждалась выше в контексте проблемы с маркированием данных). Для создания модели обработки естественных языков с помощью BERT обычно используют готовую модель для заданного языка, предобученную на крупном наборе данных без меток (отсутствие меток делает создание этого набора относительно дешевым). Затем на ее основе можно создавать модели для конкретных задач (таких как анализ эмоций в тексте или поиск ответов на вопросы), оптимизируя ее, используя обучение с учителем и относительно небольшой промаркированный набор данных. Успех BERT продемонстрировал, что этот метод является подходящим и эффективным для создания передовых систем обработки естественного языка.

Новые виды оборудования

Современное глубокое обучение работает на графических адаптерах (англ. graphics processing units, или GPU) — специальном оборудовании, оптимизированном для быстрого выполнения скалярных произведений. В конце 2000-х потребительские GPU начали использовать для ускорения обучения нейронных сетей, что послужило ключевым фактором многих прорывов, которые дали толчок развитию глубокого обучения. В последние десять лет производители аппаратного обеспечения оценили важность этого рынка и начали выпускать устройства, спроектированные специально для глубоких сетей и поддерживающие соответствующие библиотеки, такие как TensorFlow и PyTorch. По мере увеличения наборов данных и самих сетей спрос на более быстрое оборудование только растет. Но в то же время все большую обеспокоенность вызывает расход энергии, связанный с глубоким обучением, поэтому начинаются разработки оборудования с пониженным энергопотреблением.

В конце 1980-х годов благодаря работам Карвера Мида* появились нейроморфные компьютеры. Нейроморфный компьютерный чип состоит из схемы сверхвысокого уровня интеграции (very-large-scale integrated, или VLSI), способной соединять миллионы маломощных модулей, известных как импульсные нейроны. По сравнению с искусственными нейронами в стандартных системах глубокого обучения эти нейроны больше напоминают биологические. В частности, импульсный нейрон не активируется в ответ на сочетание входных сигналов, которые передаются ему в определенный момент времени. Вместо этого он сохраняет внутреннее состояние (или потенциал активации), которое меняется со временем по мере получения активационных импульсов. Этот потенциал повышается при получении новых активаций и снижается со временем, если на вход ничего не поступает. Нейрон активируется, когда его потенциал превышает определенный порог. Ввиду временно́го снижения потенциала активации импульсный нейрон генерирует вывод только в случае получения необходимого количества входных сигналов на протяжении ограниченного отрезка времени (схема пульсации). Одно из преимуществ такой временно́й обработки в том, что импульсный нейрон не активируется на каждой итерации, поэтому снижается количество потребляемой электроэнергии.

В сравнении с традиционными центральными процессорами нейроморфные чипы обладают следующими особенностями:

---

* https://en.wikipedia.org/wiki/Carver_Mead

- *Составные элементы*. Традиционные процессоры состоят из транзисторных логических вентилей (таких как И, ИЛИ, И-НЕ и т. д.), тогда как нейроморфные чипы основаны на импульсных нейронах.
- *Аналоговый аспект нейроморфных чипов*. В традиционных цифровых компьютерах информация передается в виде высоких/низких электрических разрядов синхронно с тактовой частотой; в нейроморфном чипе информация передается в виде комбинаций высоких/низких сигналов, которые варьируются со временем.
- *Архитектура*. Традиционные компьютеры основаны на архитектуре фон Неймана, централизованной по своей природе: вся информация в них проходит через центральный процессор. Нейроморфные чипы спроектированы с расчетом на масштабное распараллеливание информационного потока между импульсными нейронами. Последние взаимодействуют между собой напрямую, а не через центральную вычислительную шину.
- *Распределение информации по времени*. Информационные сигналы, проходящие по нейроморфному чипу, используют распределенные представления, похожие на те, которые мы обсуждали в главе 4, но с одним отличием: помимо прочего, они распределены еще и по времени. Распределенные представления более устойчивы к потере информации по сравнению с локальными, и это очень полезно при передаче данных между сотнями тысяч или миллионами компонентов, некоторые из которых вполне могут отказать.

На сегодняшний день существует ряд крупных исследовательских проектов, посвященных нейроморфным вычислениям. Например, в 2013 году Еврокомиссия выделила один миллиард евро на финансирование десятилетнего проекта по изучению человеческого мозга (Human Brain Project*). В нем принимают непосредственное участие более пятисот ученых и больше ста исследовательских центров по всей Европе. Одной из его главных задач является разработка нейроморфных вычислительных платформ, способных имитировать работу полноценного человеческого мозга. Также был разработан ряд коммерческих нейроморфных чипов. В 2014 году компания IBM выпустила чип TrueNorth, состоящий более чем из миллиона нейронов, соединенных вместе с помощью 286 миллионов синапсов. Его энергопотребление примерно в 10 000 раз меньше, чем у традиционного микропроцессора.

---

* https://www.humanbrainproject.eu/

В 2018 году подразделение Intel Labs анонсировало выход нейроморфного чипа Loihi (произносится как *лоу-и-хи*), содержащего 131 072 нейрона, которые соединены 130 000 000 синапсами. Нейроморфные вычисления могут привести к революции в глубоком обучении; однако перед ними по-прежнему стоит ряд вызовов, среди которых можно выделить разработку алгоритмов и шаблонов проектирования ПО для программирования параллельного оборудования в таких огромных масштабах.

Наконец, если заглянуть чуть дальше в будущее, еще одним направлением развития аппаратного обеспечения, которое может привести к радикальным изменениям в глубоком обучении, являются квантовые вычисления. Квантовые чипы уже существуют; например, компания Intel создала экспериментальный 49-кубитный квантовый чип под кодовым именем Tangle Lake. Кубит — это квантовый эквивалент двоичной цифры (бита) в традиционных вычислениях. Он может хранить более одного бита информации; но, по последним оценкам, чтобы квантовые компьютеры стали пригодными для коммерческого использования, они должны состоять как минимум из одного миллиона кубитов. Ожидается, что на достижение таких масштабов уйдет около семи лет.

Трудности интерпретируемости

Суть машинного и глубокого обучения на фундаментальном уровне заключается в принятии решений на основе данных. Несмотря на существование множества мощных алгоритмов и методик, которые могут соперничать с людьми (а иногда и превосходить их) в целом спектре задач, во многих ситуациях одного лишь решения недостаточно. Зачастую решение необходимо обосновать, особенно если оно затрагивает живого человека — как, например, при составлении медицинского диагноза или оценивании кредитоспособности. Это отражено в той части законодательства, которая касается защиты личных данных и принятия алгоритмических решений, влияющих на жизни людей. Например, в пункте 71* декларативной части Общего регламента по защите данных (англ. General Data Protection Regulations, или GDPR) отмечается, что лица, которых касается решение, принятое в результате

---

* Декларативная часть закона не имеет обязательной юридической силы; она предназначена для разъяснения правовых документов.

автоматического процесса, имеют право на разъяснение того, каким образом это решение было принято.

Разные модели машинного обучения предоставляют разные уровни интерпретируемости того, как они пришли к тому или иному решению. Однако глубокие нейронные сети, наверное, хуже всего поддаются интерпретации. С одной стороны, модель глубокого обучения несложна: она состоит из простых вычислительных модулей (нейронов), которые соединены между собой в сеть. Но масштаб этих сетей (а именно, количество нейронов и соединений между ними), распределенная природа их представлений и последовательное преобразование входных данных по мере того, как информация движется по сети, чрезвычайно затрудняет интерпретацию, понимание и, следовательно, объяснение того, как на основе ввода принимается решение.

На сегодня юридический статус права на объяснение в рамках GDPR не совсем ясен, и понимание того, какие именно последствия оно имеет для машинного и глубокого обучения, еще предстоит выработать в ходе судебных разбирательств. Тем не менее этот пример подчеркивает потребность общества в более четком представлении о том, как модели глубокого обучения работают с данными. Способность интерпретировать и понимать внутренние механизмы моделей глубокого обучения важна еще и с технической точки зрения. Например, знание того, как модель использует данные, помогает определить, свойственна ли ей предвзятость в принятии решений, и найти случаи, в которых она дает самые плохие результаты. Исследователи глубокого обучения и искусственного интеллекта в целом уже занимаются этой проблемой. В настоящий момент существует ряд проектов и конференций, посвященных таким темам, как объяснимый искусственный интеллект и человеческая интерпретируемость в машинном обучении.

Крис Олах вместе со своими коллегами выделил следующие основные методики, которые на сегодняшний день применяются для анализа внутренних механизмов моделей глубокого обучения: визуализация признаков, атрибуция и уменьшение размерности (Олах и др., 2018). Чтобы понять, как сеть обрабатывает информацию, можно проанализировать, какой ввод вызывает то или иное поведение — например, активацию нейрона. Это позволяет определить, что именно нейрон научился распознавать в этом вводе. Цель визуализации признаков состоит в генерации и наглядном представлении входных значений, которые вызывают определенную активность в сети. Оказывается, для генерации такого ввода можно использовать методы оптимизации, например, обратное распространение. Вначале генерируется случайный ввод, который постепенно обновляется, пока не будет достигнуто определенное

поведение. Затем мы можем визуализировать полученные входные значения, чтобы лучше понять, что именно сеть в них распознает, когда реагирует определенным образом. Атрибуция пытается объяснить отношение между нейронами — например, как вывод нейрона в одном из слоев влияет на общий вывод сети. Это можно сделать путем генерации салиентности (тепловой карты) нейронов, которая показывает, какой вес сеть присваивает выводу нейрона в ходе принятия определенного решения. Наконец, значительная часть активности внутри глубокой сети связана с обработкой высокоразмерных векторов. Визуализация данных позволяет использовать развитую зрительную кору нашего головного мозга для интерпретации данных и отношений внутри них. Однако визуализировать информацию, количество измерений в которой превышает три, крайне сложно. Следовательно, методы, способные систематически уменьшать размерность высокоразмерных данных и визуализировать результаты, являются чрезвычайно полезными для интерпретации информационных потоков в глубокой сети. t-SNE* — это широко известная методика, которая визуализирует высокоразмерные данные, проецируя каждую их точку на двух- или трехмерную карту (ван дер Маатен и Хинтон, 2008). Исследования в области интерпретации глубоких сетей все еще находятся на начальном этапе, но в ближайшие годы имеют шанс занять более важное место в сообществе глубокого обучения. Тому есть как общественные, так и технические причины.

Заключительные мысли

Глубокое обучение идеально подходит для решения задач с крупными наборами высокоразмерных данных. Благодаря этому оно имеет все шансы сыграть важную роль в решении важнейших научных проблем нашего времени. В последние два десятилетия прорывы в области биологического секвенирования сделали возможной высокоточную генерацию цепочек ДНК. Эти генетические данные могут стать фундаментом для персонализированных медицинских технологий следующего поколения. В то же время международные исследовательские проекты, такие как Большой адронный коллайдер и сеть орбитальных телескопов, ежедневно генерируют огромные объемы данных. Их анализ может помочь нам понять физические свойства вселенной

* Лоуренс ван дер Маатен и Хинтон, «Visualizing Data using t-SNE», Journal of Machine Learning Research 9 (2008): 2579–2605.

в мельчайшем и крупнейшем масштабах. В связи с этим невероятным потоком информации ученые все чаще обращаются к машинному и глубокому обучению.

Но на более обыденном уровне глубокое обучение уже оказывает непосредственное влияние на наши жизни. Вполне возможно, что в последние несколько лет вы, сами того не подозревая, уже начали регулярно использовать глубокие сети. Они лежат в основе поисковых систем, инструментов машинного перевода, механизмов распознавания лиц в вашей фотокамере или на веб-сайте социальной сети, речевого интерфейса вашего смарт-устройства и т. д. Существует потенциальная опасность того, что след данных и метаданных, который вы оставляете во время навигации по Интернету, тоже обрабатывается и анализируется с помощью моделей глубокого обучения. Вот почему так важно понимать, что такое глубокое обучение, как оно работает, на что оно способно и какие ограничения ему присущи на текущий момент.

Функция активации
Функция, которая принимает на вход взвешенную сумму входных значений нейрона и применяет к ней нелинейное отношение. Наличие функций активации в нейронах позволяет сети устанавливать нелинейные отношения. Примерами часто используемых функций активации являются логистические уравнения, tanh и ReLU.

Искусственный интеллект
Область исследований, посвященная разработке вычислительных систем, способных выполнять задачи и действия, которые, как принято считать, требуют участия человеческого интеллекта.

Обратное распространение
Алгоритм, который используется для обучения нейронных сетей со скрытыми слоями нейронов. В ходе обучения веса сети постепенно обновляются, чтобы уменьшить отклонение ее вывода. Для обновления весов на соединениях, ведущих к определенному нейрону, необходимо сначала вычислить приблизительное влияние его вывода на общее отклонение сети. В качестве метода подсчета этих приблизительных значений для каждого нейрона сети используется алгоритм обратного распространения. После этого можно обновить веса каждого нейрона, используя алгоритм оптимизации, такой как градиентный спуск. Обратное распространение состоит из двух этапов: прямого и обратного прохождения. На первом этапе сети передается образец, и затем в ее выходном слое вычисляется итоговое отклонение; для этого ее вывод сравнивается с ожидаемым результатом, указанным в наборе данных. Во время обратного прохождения отклонение сети проходит через все нейроны, от выходного слоя к входному, и каждый из них берет на себя часть «вины» за допущенную погрешность в размере, прямо пропорциональном тому, насколько изменения вывода этого нейрона повлияли на отклонение. Этот процесс называется обратным распространением ошибок — отсюда и название алгоритма.

Сверточная нейронная сеть
Сверточной называют нейронную сеть, у которой есть хотя бы один сверточный слой. Этот слой состоит из нейронов, имеющих общий набор весов,

и совокупность их рецептивных полей охватывает весь ввод. Сочетание выводов таких нейронов называется картой признаков. Во многих сверточных нейронных сетях карты признаков проходят сначала через слой активации ReLU, а затем через слой пулинга.

Набор данных
Ряд образцов, каждый из которых представлен в виде набора признаков. В простейшем случае набор данных имеет вид матрицы $n \times m$, где n — количество образцов (строки), а m — количество признаков (столбцы).

Глубокое обучение
Подраздел машинного обучения, посвященный проектированию и оцениванию обучающих алгоритмов и архитектур для современных моделей нейронных сетей. Глубокой считается нейронная сеть с более чем двумя слоями скрытых модулей (или нейронов).

Сеть с прямой связью
Нейронная сеть, в которой все соединения ведут к нейронам в следующем слое. Иными словами, ни один нейрон не передает свой вывод нейрону из предшествующего слоя.

Функция
Детерминистическое отношение между набором входных значений и одним или несколькими выходными. В контексте машинного обучения термин «функция» часто является синонимом модели.

Градиентный спуск
Алгоритм оптимизации для поиска функции с минимальным отклонением при моделировании закономерностей в наборе данных. В контексте обучения нейронной сети градиентный спуск используется для подбора сочетания весов, которое минимизирует погрешность вывода нейрона. Речь идет о градиенте ошибок нейрона, который возникает в ходе обновления весов. Этот алгоритм часто используется в сочетании с обратным распространением для обучения нейронных сетей со скрытыми слоями нейронов.

Графический адаптер (Graphical Processing Unit, или GPU)
Специальное аппаратное обеспечение, оптимизированное для быстрого умножения матриц (скалярного произведения). Изначально разрабатывалось

для ускорения отрисовки графики, но оказалось полезным и для повышения производительности нейронных сетей.

Длинная краткосрочная память (Long Short-Term Memory, или LSTM)
Сеть, предназначенная для решения проблемы исчезающих градиентов в рекуррентных нейронных сетях. Сеть состоит из блока, в котором поток активаций последовательно проходит через интервалы и набор вентилей, которые регулируют этот поток. Вентили реализованы с помощью слоев функций активации на основе сигмоид и tanh. Стандартная архитектура LSTM предусматривает три таких вентиля: вентиль забывания, вентиль обновления и вентиль вывода.

Машинное обучение (Machine Learning, или ML)
Область компьютерных наук, посвященная разработке и оцениванию алгоритмов, позволяющих компьютерам учиться на опыте. В целом понятие опыта представляет набор данных с событиями, происходившими ранее, а обучение состоит в поиске и извлечении из этого набора полезных закономерностей. Алгоритм машинного обучения принимает на вход набор данных и возвращает модель, которая кодирует извлеченные (или усвоенные) закономерности.

Алгоритм машинного обучения
Процесс анализа набора данных и извлечения модели (то есть функции, реализованной в виде компьютерной программы), соответствующей закономерностям в этом наборе.

Модель
В машинном обучении модель — это компьютерная программа, которая кодирует закономерности, извлеченные обучающим алгоритмом из набора данных. Существует множество разных видов моделей, однако в глубоком обучении они состоят из нескольких слоев со скрытыми нейронами. Модель создается (или обучается) путем применения к набору данных обучающего алгоритма. Обученную модель можно использовать для анализа новых экземпляров; иногда процесс анализа новых экземпляров с помощью обученной модели называют логическим выводом. В контексте машинного обучения термины «модель» и «функция» зачастую выступают синонимами: модель — это функция, реализованная в виде компьютерной программы.

Нейроморфные вычисления
Нейроморфные чипы состоят из очень большого количества импульсных нейронов, соединенных между собой методом масштабного распараллеливания.

Нейронная сеть
Модель машинного обучения, реализованная в виде сети простых модулей обработки информации под названием «нейроны». Изменяя соединения между нейронами, можно создавать разнообразные виды нейронных сетей. Популярными примерами являются сети с прямой связью, а также сверточные и рекуррентные сети.

Нейрон
В контексте машинного обучения (а не в биологическом смысле) нейрон — это простой алгоритм обработки информации, который принимает на вход ряд числовых значений и привязывает их к высокому или низкому уровню активации. Для этого каждое входное значение обычно умножают на вес, затем слагают полученные результаты и пропускают эту взвешенную сумму через функцию активации.

Переобучение
Переобучение на наборе данных возникает, если модель, возвращаемая обучающим алгоритмом, оказывается настолько сложной, что способна моделировать даже мелкие вариации, являющиеся информационным шумом.

Рекуррентная нейронная сеть
Рекуррентная нейронная сеть состоит из одного слоя скрытых нейронов, вывод которых подается обратно в этот же слой при поступлении следующего ввода. Обратная связь (рекуррентность) создает в сети память, которая позволяет обрабатывать каждый ввод в контексте того, что уже было обработано. Рекуррентные нейронные сети идеально подходят для обработки последовательных данных и временных рядов.

Обучение с подкреплением
Цель этого вида обучения состоит в том, чтобы позволить агенту выработать правила поведения в заданном окружении. Правила — это функция, которая привязывает текущие наблюдения агента к его окружению, а его собственное внутреннее состояние к действию. Этот метод обычно применяется для управления в реальном времени — например, в робототехнике и играх.

Модуль ReLU
Нейрон, который использует для активации выпрямленную линейную функцию.

Обучение с учителем
Вид машинного обучения, цель которого — формирование функции, привязывающей набор входных атрибутов образца к точной оценке отсутствующего значения целевого атрибута в том же образце.

Целевой признак
В машинном обучении с учителем целевым называют признак, значение которого учится выводить модель.

Недообучение
Недообучение на наборе данных возникает, если модель, возвращаемая обучающим алгоритмом, оказывается слишком простой для того, чтобы охватить все аспекты отношения между вводом и выводом в заданной предметной области.

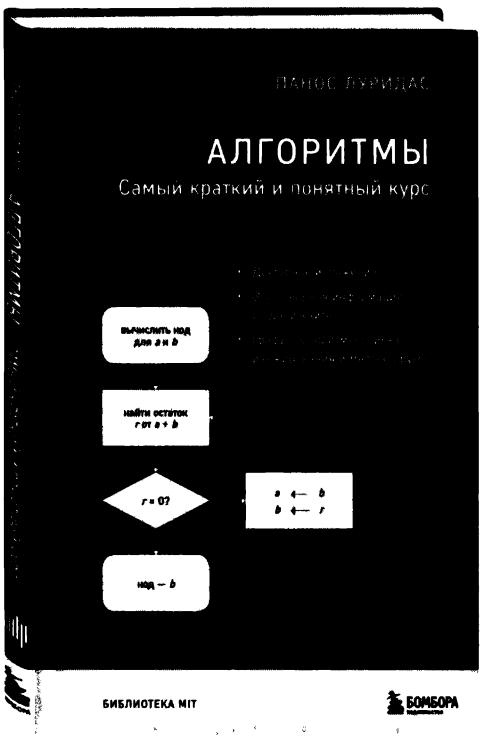
Обучение без учителя
Вид машинного обучения, цель которого состоит в том, чтобы определить в данных закономерности, такие как скопление похожих образцов. В отличие от обучения с учителем здесь нет целевого образца.

Исчезающий градиент
Исчезающий градиент показывает, что с увеличением количества слоев в сети процесс обучения замедляется. Эта проблема возникает из-за того, что при использовании обратного распространения и градиентного спуска в процессе обучения обновление весов на соединениях, ведущих к нейрону, зависит от градиента (чувствительности) отклонения сети по отношению к выводу этого нейрона. В ходе обратного прохождения градиентов ошибок через нейрон выполняется ряд операций умножения, где множитель часто меньше единицы. В итоге во время этого процесса градиент, как правило, становится все меньше и меньше (то есть исчезает). Прямым следствием является то, что в начальных слоях сети веса меняются незначительно, и это затягивает процесс обучения соответствующих нейронов.

БИБЛИОТЕКА MIT

БОМБОР
ИЗДАТЕЛЬСТВО

ПРОСТО О СЛОЖНО



Лучшие издания Массачусетского Технологического Института

bombora.ru
bomborabooks bombora
2022-070

ДЖОН Д. КЕЛЛЕХЕР

компьютерные науки

ГЛУБОКОЕ ОБУЧЕНИЕ

Самый краткий и понятный курс

Познакомьтесь поближе с технологией **глубокого обучения**, лежащей в основе таких разработок в области **искусственного интеллекта**, как компьютерное зрение, самоуправляемые автомобили, распознавание речи и многих других.

Для чтения этой книги вам не потребуются углубленные знания в математике и программировании – ее автор **Джон Д. Келлехер** более двадцати лет преподает машинное обучение в Университете Дублина и мастерски умеет объяснять сложные темы простым и понятным языком.

ISBN 978-5-04-116355-6

9 785041 163556

БОМБОРА
ИЗДАТЕЛЬСТВО

БОМБОРА – лидер на рынке полезных и вдохновляющих книг.
Мы любим книги и создаем их, чтобы вы могли творить, открывать мир, пробовать новое, расти. Быть счастливыми. Быть на волне.

bombora.ru bomborabooks bombora