Содержание

Предисловие

Добро пожаловать!

Несколько слов введения в эту книгу,
как получить файлы и рисунки,
и благодарности тем, кто помогал мне

Что вы получите от этой книги

Привет!

Если вы интересуетесь глубоким обучением (Deep Learning, DL) и машинным обучением (Mashing Learning, ML), то эта книга для вас.

Моя цель в данной книге – дать вам прочные навыки эффективного практического применения машинного обучения и глубокого обучения.

После прочтения этой книги вы будете уметь:

- разрабатывать и обучать собственные нейронные сети;
- использовать нейронные сети для понимания данных и создания новых данных;
- присваивать описательные категории текстам, изображениям и другим типам данных;
- предсказывать последующие значения последовательности данных;
- исследовать структуру ваших данных;
- обрабатывать ваши данные с максимальной эффективностью;
- использовать языки программирования и библиотеку DL по своему желанию;
- воспринимать новые знания и идеи и применять их на практике;
- получать удовольствие от обсуждения глубокого обучения с другими специалистами.

Мы используем серьезный, но дружелюбный подход, сопровождаемый большим количеством иллюстраций. Мы делаем это без каких-либо кодов и без всякой математики, за исключением умножения.

Если это звучит привлекательно для вас, добро пожаловать!

Для кого эта книга

Эта книга создана для тех, кто хочет использовать машинное обучение и глубокое обучение в своей работе. Это программисты, инженеры, ученые, руководители, музыканты, врачи и все, кто хочет работать с большими объемами данных, извлекая из них полезную информацию или формируя новые данные.

Многие инструменты машинного обучения и особенно глубокого обучения имеются в многочисленных библиотеках со свободным доступом, которые любой при желании может немедленно загрузить.

Но хотя эти инструменты легко доступны и легко устанавливаемы, они все же требуют значительных технических знаний для правильного их применения. Совсем не трудно попросить компьютер выполнить что-то бессмысленное, и он радостно выполнит это, выдав на выходе бессмыслицу.

Такого рода вещи происходят все время. Хотя машинное обучение и глубокое обучение – мощный инструмент, он вовсе не слишком дружелюбен к пользователю.

Выбор правильных алгоритмов и затем применение их надлежащим образом требуют в дополнение последовательных, технически грамотных решений. Когда, как часто бывает, события развиваются не так, как планировалось, необходимы знания, чтобы понять, что происходит внутри системы, с тем чтобы исправить положение дел.

Существует много подходов к освоению этих существенных знаний в зависимости от того, как вы хотите ими овладеть.

Некоторые любят жесткий детальный алгоритмический анализ, сопровождаемый обширной математикой. Если это тот способ, с помощью которого вы хотите овладеть знаниями, то существуют солидные труды, которые предлагают этот стиль представления знаний [Bishop06], [Goodfellow17]. Он требует серьезных усилий, но дает глубокое понимание того, как и почему этот механизм работает. Если вы выберете данный способ, то вам придется проделать значительную работу, чтобы применить теоретические знания на практике.

Другая крайность, когда человек просто хочет знать, как решить некую конкретную задачу. Серьезных книг, которые содержат библиотеки с рецептами машинного обучения, достаточно много [Chollet17], [Müller-Guido16], [Raschka15], [VanderPlus6]. Эти методы проще, чем математический подход, но вы можете ощущать недостаток структурной информации, объясняющей, как это работает. Без этой информации и соответствующей терминологии трудно разобраться, почему что-то, что, вы полагали, должно работать, не работает или почему что-то не работает так хорошо, как вы полагали. Это может вызвать желание почитать литературу, описывающую новые идеи и результаты, потому что обсуждения обычно предполагают общий уровень знаний о предмете, используемой библиотеке и языке.

В этой книге принят промежуточный подход. Я намерен дать вам инструменты для уверенного практического применения глубокого обучения. Я хочу, чтобы вы могли в вашей работе сделать разумный выбор и были способны следовать в общем потоке новейших идей, появляющихся почти каждый день.

Моя цель здесь – осветить фундаментальные основы достаточно глубоко, чтобы у вас была надежная база поддержки в вашей работе. Я хочу, чтобы вы имели достаточную базу не только для понимания материалов этой книги, но и материалов, которые, возможно, понадобятся вам для консультаций и изучения в процессе работы с глубоким обучением.

Это не книга о программировании. Программирование – важный аспект, но оно неизбежно вовлекает во все детали, которые необязательны для нашего основного предмета изучения. Примеры программирования ограничат нас одной библиотекой или одним языком программирования. И хотя детали необходимы для создания законченных систем, они могут отвлечь нас, когда будет необходимо сосредоточить внимание на основополагающих идеях. Вместо того чтобы вдаваться в дискуссии о циклах, индексах и структуре данных, мы будем все это

обсуждать здесь независимо от какого-либо языка или библиотеки. Если вы усвоите главную идею, то прочтение документации для любой библиотеки будет несложным делом.

Мы спустимся на землю в главах 15, 23 и 24, когда будем обсуждать научную библиотеку машинного обучения и библиотеку глубокого обучения Keras[1]. Обе эти библиотеки базируются на языке Python. В этих главах мы погружаемся в детали библиотеки языка программирования Python и имеем в них много примеров с кодами.

Даже если вы не знаете данного языка, эти программы послужат вам примером технологий и программных структур, что поможет справляться с новыми проблемами. Коды в этих главах с программированием доступны в качестве файлов на языке Python. Они могут быть использованы с помощью программного окружения, базирующейся на браузере графической веб-оболочки Jupyter [Jupyter16] или с помощью классического, разработанного для Python окружения, такого как PyCharm [JetBrain17].

Большинство других глав также может быть поддержано выборочными файлами (notebooks) на Python. В них приводится код для каждой компьютерной графики в книге, часто используя методы, обсуждаемые в этой главе. Поскольку в книге отсутствует ориентация на Python и программирование (за исключением упомянутых выше глав), эти записи присутствуют в книге как бы «за сценой» и только слегка комментируются.

Машинное обучение, глубокое обучение и большие данные оказывают неожиданно быстрое и глубокое влияние на общество во всем мире. Что это означает для людей и культуры – сложный и важный предмет. Ряд интересных книг и статей, обсуждающих эту тему, часто приходит к тонкой смеси положительных и отрицательных выводов [Agüera y Arcas17] [Barrat15] Domingos15] [Kaplan16].

Почти без математики

Многие, отнюдь неглупые люди не являются поклонниками сложных математических уравнений, и если это вы, то здесь вы дома.

В этой книге почти нет математики. Если вы справляетесь с умножением, то этого достаточно, поскольку это вся математика, которую мы используем.

Большинство обсуждаемых нами алгоритмов базируется на солидных теоретических источниках и является результатом аккуратного анализа и проработки. Важно знать это обстоятельство, когда вы модифицируете алгоритм для какой-то новой задачи или другой реализации. Но на практике почти все используют хорошо оптимизированные реализации с открытым кодом, написанные экспертами и доступные в бесплатных библиотеках.

Наша цель – понять принципы этих технологий, усвоить, как применять их надлежащим образом и как интерпретировать результаты. Ничто из этого не требует от нас вдаваться в математические подробности.

Если вы любите математику или хотите ознакомиться с теорией, следуйте ссылкам к каждой главе. Многие из этих материалов превосходны, интеллекту-

---

[1] См.: *Джулли А., Пал С.* Библиотека Keras – инструмент глубокого обучения. М.: ДМК Пресс, 2017. – *Прим. перев.*

ально полновесны и описывают детали, которые я сознательно опустил в этой книге. Но если математика не ваш конек, то и нет необходимости в математических деталях.

Много рисунков

Некоторые идеи более отчетливо проявляются в рисунках, чем в словах. И даже если слова выполняют свою работу, рисунки цементируют идеи. Поэтому эта книга обильно проиллюстрирована рисунками.

Все рисунки данной книги доступны для свободного считывания (см. далее).

Загрузки

Вы можете загрузить Jupiter/Python для этой книги, все рисунки и другие файлы, относящиеся к этой книге, совершенно свободно.

Все файлы (notebooks)

Все файлы оболочки Jupiter/Python в данной книге доступны на GitHub.

Файлы для главы 15 (scikit-learn – Python-модуль для машинного обучения) и глав 23 и 24 (Keras) содержат все коды, которые представлены в этих главах.

Другие файлы доступны «за сценой» для просмотра того, как получены рисунки. Они слегка документированы и служат скорее справкой, чем учебным пособием.

Файлы опубликованы по лицензии Массачусетского технологического института (MIT), что, по существу, означает, что вы свободны использовать их для любых целей. Никаких гарантий, что в них отсутствуют дефекты, что они будут надежно работать, что они не приведут к сбоям и т. п., не дается. Чувствуйте себя свободными извлекать коды и адаптировать их так, как вам удобно, хотя в лицензии говорится об использовании только в личных целях (это для файлов с пометкой LICENSE).

https://github.com/blueberrymusic/DeepLearningBookCode-Volume1
https://github.com/blueberrymusic/DeepLearningBookCode-Volume2

Все рисунки

Все рисунки в этой книге доступны на GitHub в формате PNG с высоким разрешением. Вы можете свободно использовать их в классах, обсуждениях, лекциях, докладах, статьях и даже других книгах.

Так же, как код, рисунки опубликованы по лицензии Массачусетского технологического института (MIT), и вы можете использовать их по своему усмотрению, сохраняя уведомление об авторских правах. Используя рисунки, вы не обязаны кредитовать меня как их создателя, но я буду признателен вам, если вы сделаете это.

Наименования файлов соответствуют номерам рисунков в книге, поэтому их нетрудно найти. Когда вы будете искать что-либо визуально, то будет полезно по-

смотреть страницы в уменьшенном формате. Каждая такая страница содержит 20 изображений:

https://github.com/blueberrymusic/DeepLearningBookFigures-Thumbnails

Сами рисунки сгруппированы в двух томах:

https://github.com/blueberrymusic/DeepLearningBookFigures-Volume1
https://github.com/blueberrymusic/DeepLearningBookFigures-Volume2

Источники

Перечень литературы и ресурсов содержит другие файлы, такие как шаблоны для иконок глубокого обучения, которые используются нами в этой книге.

https://github.com/blueberrymusic/DeepLearningBook-Resources

Опечатки

Хотя мной приняты все возможные меры, в книге такого объема трудно избежать ошибок. Если вы обнаружите что-то, что покажется вам неверным, пожалуйста, дайте мне знать по адресу andrew@dlbasics.com.

Два тома

Книга оказалась очень большой, поэтому я сделал ее двухтомником с примерно одинаковым размером.

Поскольку двухтомник является, по существу, одной книгой, второй том начинается там, где заканчивается первый. Если вы в данный момент читаете второй том, то, следовательно, вы или уже прочитали первый, или чувствуете себя достаточно уверенно для понимания содержания второго тома.

Благодарности

Авторы любят говорить, что никто не пишет книги в одиночку. Мы говорим так, потому что так оно и есть.

Я чрезвычайно благодарен Эрику Брауну (Eric Braun), Эрику Хейнесу (Eric Haines), Стиву Друкеру (Steven Drucker) и Тому Рейке (Tom Reike) за последовательную и воодушевляющую помощь в этом проекте, помогавшую мне уверенно чувствовать себя на протяжении всего времени осуществления проекта. Спасибо за ваше дружелюбие и поддержку.

Большое спасибо моим рецензентам за щедрые и глубокие комментарии, значительно улучшившие книгу: Адаму Финкельштейну (Adam Finkelstein), Алексу Колберну (Alex Colburn), Александру Келлеру (Alexander Keller), Алин Рокфорд (Alyn Rockwood), Анджело Песке (Angelo Pesce), Барбаре Монес, (Barbara Mones), Брайану Уайвиллу (Brian Wyvill) , Крейгу Каплану (Craig Kaplan), Дагу Робле, (Doug Roble), Эрику Брауну (Eric Braun), Эрику Хейнесу (Eric Haines), Грегу Тэрку (Greg Turk), Джеффу Халтвисту (Jeff Hultquist), Джессике Ходжинс (Jessica Hodgins),

Кристи Мортон (Kristi Morton), Лезли Истед (Lesley Istead), Луис Авардо (Luis Avarado), Матт Фарр (Matt Pharr), Майку Тика (Mike Tyka), Морган МакГвире (Morgan McGuire), Паулю Бэдсли (Paul Beardsley), Паулю Страуссу (Paul Strauss), Петеру Шэрли (Peter Shirley), Филиппу Слузаллеку (Philipp Slusallek), Сербан Порумбеску (Serban Porumbescu), Стефанусу Ду Тойту (Stefanus Du Toit), Стивен Друкер (Steven Drucker), Венхао Ю (Wenhao Yu) и Закори Эриксон (Zackory Erickson).

Особая благодарность ответственным рецензентам Александру Келлеру (Alexander Keller), Эрику Хейнесу (Eric Haines), Джессике Ходжинс (Jessica Hodgins) и Луис Авардо (Luis Avarado), которые прочитали всю книгу или большую часть рукописи и дали существенные предложения как по представлению содержания, так и по структуре книги.

Благодарю Морган МакГвире (Morgan McGuire) за применение технологии Markdeep, что позволило мне в большей степени сосредоточиться на том, что я должен сказать, чем на том, каким должен быть формат.

Спасибо Тодду Жиманскому (Todd Szymanski) за вдумчивые советы по оформлению и компоновке содержания и титула книги и выявление ошибок в их размещении.

Спасибо первым читателям, которые выявили опечатки и другие проблемы:

Кристиану Форгангу (Christian Forfang), Дэвиду Полу (David Pol), Эрику Хейнесу (Eric Haines), Гопи Меенакшисундараму (Gopi Meenakshisundaram), Косте Смоленскому (Kostya Smolenskiy), Мурисио Вивес, (Mauricio Vives), Майку Вонгу (Mike Wong) и Мринал Мохит (Mrinal Mohit).

Все эти люди улучшили книгу, но окончательные решения были за мной. И проблемы, которые остались, являются моей ответственностью.

Справочные материалы

Этот раздел появляется в каждой главе. Он содержит справки по всем документам, которые относятся к содержанию данной главы. Здесь также могут присутствовать другие полезные материалы: статьи, веб-страницы, документация, блоги и иные источники.

Где только возможно, я предпочитал те источники, которые доступны онлайн, поэтому вы можете немедленно получить их, используя Сеть. Исключение составляют обычно книги, но иногда я привожу важные онлайн-справки, даже если за ними стоят требования оплаты.

[Agüera y Arcas17] *Blaise Agüera y Arcas, Margaret Mitchell, and Alexander Todorov*. Physiognomy's New Clothes. Medium, 2017. https://medium.com/@blaisea/physiognomys-new-clothes-f2d4b59fdd6a.

[Barrat15] *James Barrat*. Our Final Invention: Artificial Intelligence and the End of the Human Era. St. Martin's Griffin, 2015.

[Bishop06] *Christopher M. Bishop*. Pattern Recognition and Machine Learning. Springer-Verlag, 2006. С. 149–152.

[Chollet17] *François Chollet*. Deep Learning with Python. Manning Publications, 2017.

[Domingos15] *Pedro Domingos*. The Master Algorithm. Basic Books, 2015.

[Goodfellow17] *Ian Goodfellow, Yoshua Bengio, Aaron Courville*. Deep Learning. MIT Press, 2017. http://www.deeplearningbook.org/[1].

[JetBrains17] *Jet Brains*. Pycharm Community Edition IDE. 2017. https://www.jetbrains.com/pycharm/.

[Jupyter16] The Jupyter team. 2016. http://jupyter.org/.

[Kaplan16] *Jerry Kaplan*. Artificial Intelligence: What Everyone Needs to Know. Oxford University Press, 2016.

[Müller-Guido16] *Andreas C. Müller and Sarah Guido*. Introduction to Machine Learning with Python. O'Reilly Press, 2016[2].

[Raschka15] *Sebastian Raschka*. Python Machine Learning. Packt Publishing, 2015[3].

[VanderPlas16] *Jake VanderPlas*. Python Data Science Handbook. O'Reilly Media, 2016.

[1] *Гудфеллоу Я., Бенджио И., Курвилль А.* Глубокое обучение. М.: ДМК Пресс, 2017. ISBN: 978-5-97060-618-6.

[2] *Мюллер А., Гвидо С.* Введение в машинное обучение с помощью Python. М.: Вильямс, 2017. ISBN: 978-5-9908910-8-1.

[3] *Рашка С.* Python и машинное обучение. М.: ДМК Пресс, 2017. ISBN: 978-5-97060-409-0.

Глава 20

Глубокое обучение

Мы увидим базовые структуры
сетей глубокого обучения
и обзор многих типов слоев,
из которых они состоят

20.1. Зачем здесь эта глава

В предыдущих главах мы заложили хорошие основы для разработки алгоритмов нейронных сетей. В этой главе используем их для построения сетей путем объединения искусственных нейронов в **слои**. Как мы видели в главе 18, это позволит нам использовать эффективный алгоритм обратного распространения для повышения производительности сети. Сеть состоит из последовательности слоев, и часто ее называют **глубокой сетью**, и когда такая сеть обучает по представленным ей данным, то мы называем это **глубоким обучением**.

Мы обсудим терминологию глубокого обучения и рассмотрим наиболее популярные слои, которые используются в сетях глубокого обучения, а также некоторые примеры сетей и как построить новые сети и интерпретировать их результаты.

В этой главе будет рассказано об остальной части книги, где рассматриваются специализированные формы глубокого обучения для различных задач.

20.2. Обзор глубокого обучения

Нейронные сети строятся как большое количество слоев, и часто их называют **глубокими сетями** (их можно было бы назвать «высокими», «широкими» или «длинными» сетями, но общепринятым стало «глубокие»). Когда мы используем глубокие сети, мы обычно говорим, что осуществляем **глубокое обучение**.

Выражение **глубокое обучение** обычно относится к нейронным сетям, которые состоят из последовательности слоев. Более общее выражение **машинное обучение** обычно относится и к глубокому обучению, и к другим алгоритмам, которые мы рассматривали (например, классификатору в главе 13), которые не основаны на нейронных сетях. Но некоторые авторы трактуют «машинное обучение» и «глубокое обучение» как две разные области, так что «машинное обучение» касается алгоритмов, где не используются нейронные сети. Поэтому одни книги со словами «машинное обучение» в названии включают нейронные сети, а другие – нет. Всегда стоит убедиться, какое из них конкретный автор использует.

Результатом послойной организации нейронов является способность обучающей сети анализировать данные **иерархически**. Первые слои видят сырые данные, и каждый последующий слой способен использовать для обработки больший объем информации от нейронов предыдущего слоя. Для примера: при рассмотрении фотографии первый слой обычно смотрит на индивидуальные пиксели. Следующий слой смотрит на группы пикселей, слой после него – на группы этих групп и т. д. Начальные слои могут заметить, что одни пиксели темнее других, в то время как последующие слои замечают, что скопление пикселей похоже на глаз, а еще значительно позже слой может идентифицировать полосы, которые показывают, что изображение в целом – это тигр.

Рисунок 20.1 показывает пример архитектуры глубокого обучения, использующей три слоя.

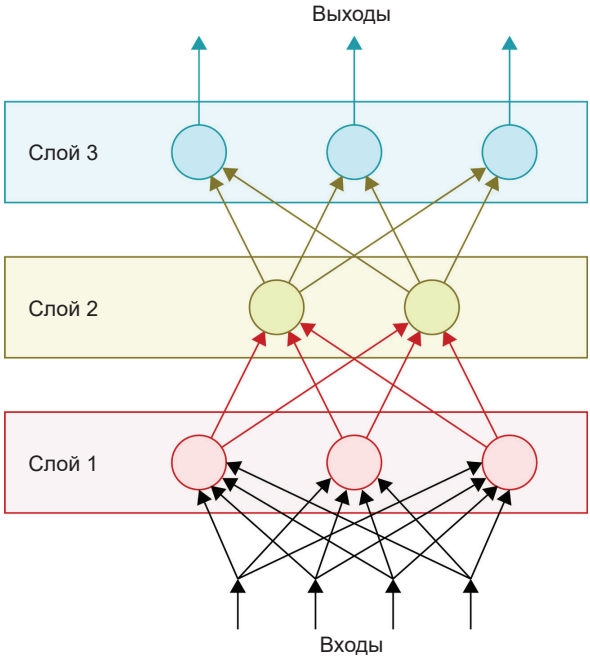

Рис. 20.1 ❖ Сеть глубокого обучения. Мы имеем 4 входа, проходящих через 3 слоя и создающих 3 выхода в конце. Мы говорим, что сеть полносвязная, потому что каждый нейрон в каждом слое получает сигнал от каждого предыдущего слоя

Когда мы рисуем слои вертикально, как на рис. 20.1, входы почти всегда мы рисуем внизу, а выходы, где появляются наши результаты, – почти всегда вверху.

Самый верхний слой (слой 3 на рис. 20.1) называется **выходным слоем**. Хотя с величинами, которые поступают из этого слоя, перед тем как воспользоваться ими, могут проводиться другие операции, например такие как софтмакс, с которым мы познакомились в главе 17, мы обычно рассматриваем этот слой как конечный в нейронной сети, потому что он последний, содержащий нейроны.

Возможно, мы ожидаем, что соответственно вначале будет входной слой, и это было бы естественно, если бы мы дали такое название слою 1 на рис. 20.1. Но

терминология не такова. «Входной слой» существует, но он редко показывается явно. Он скорее относится к памяти, которая хранит входные величины. Мы можем представлять себе входной слой как ряд стрелок внизу рис. 20.1.

Слои 1 и 2 на рис. 20.1 называются **скрытыми слоями**. Если мы представим себе кого-то, смотрящего на сеть извне, сверху или снизу, то он видит только выходной или входной слой. Мы можем представить слои, находящиеся между ними, как скрытые от взгляда, и потому их называют «скрытыми слоями» (их можно видеть со стороны, но мы не будем обращать внимания на это небольшое упущение терминологии).

Иногда мы встречаемся с рисунками сети слева направо, как на рис. 20.2.

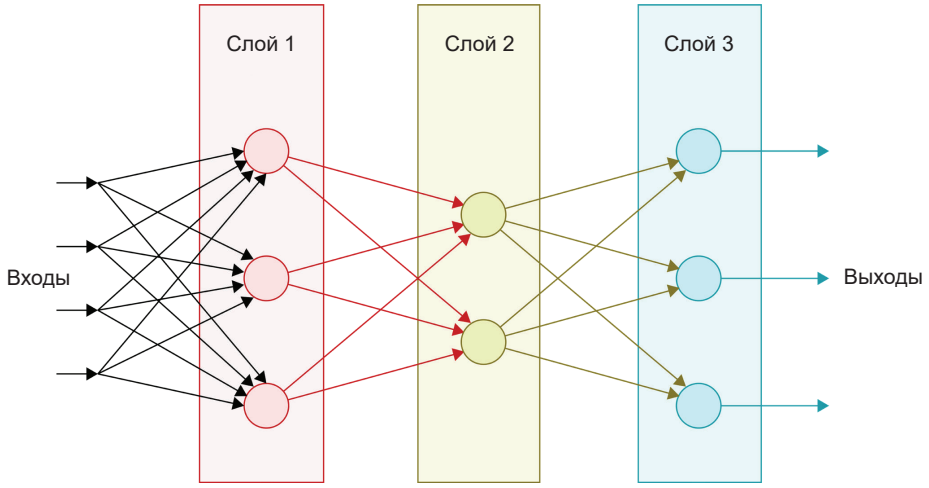

Рис. 20.2 ❖ Та же сеть, что и на рис. 20.1, нарисованная с данными, проходящими слева направо

Даже когда сеть нарисована таким образом, все равно будем использовать терминологию вертикальной ориентации. Авторы могут сказать, что слой 2 выше слоя 1 и ниже слоя 3. Мы можем всегда придерживаться такого подхода, независимо от того, как нарисована диаграмма. Если мы имеем в виду «выше» или «над», это означает, что слой ближе к выходу, а если «ниже» или «под» – значит, ближе к входу.

20.2.1. Тензоры

Хотя фундаментом сетей глубокого обучения является манипуляция числами, концептуально важна организация данных в виде списков чисел. Этот список может быть одноразмерным, как на рис. 20.3 (a), то есть просто числами, записанными одно за другим.

Мы можем называть это **сеткой**, или **матрицей**. Трехразмерный список, как на рис. 20.3 (c), может хранить объемные данные, или выборки, каждая из которых состоит из множества характеристик, измеренных многократно. Мы называем их **объемами**, или **блоками**.

Чтобы упростить обсуждение, мы будем называть список любого размера и размерности тензором. Слово тензор имеет более широкое значение в некоторых об-

ластях математики и физики. Здесь мы используем его просто как группу чисел, организованных как многоразмерный список.

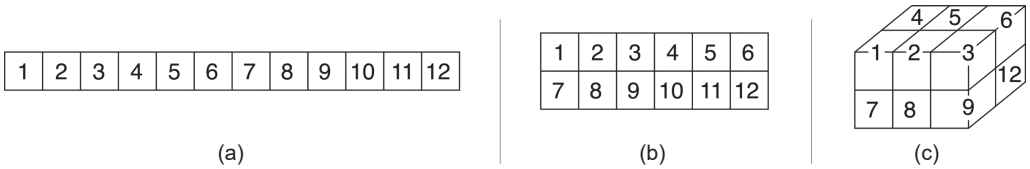

Рис. 20.3 ❖ Три тензора, каждый из 12 элементов: (a) 1D-тензор является списком; (b) 2D-тензор является сеткой; (c) 3D-тензор является кубом. Во всех этих случаях, как и в случае большей размерности, вся структура полностью заполнена. То есть все ряды, столбцы и прочие поверхности имеют одинаковую длину

Поэтому мы часто будем употреблять выражение «входной тензор» (имея в виду входные величины), «выходной тензор», имея в виду выходные величины, а также «тензор» для других тензоров внутри сети, когда он вычисляет новое представление входящих данных.

Мы говорим, что каждый тензор имеет размерность и размер в каждой размерности. Все это вместе говорит о **форме** тензора.

20.3. Вход и выход слоев

Многие сети имеют единственный входной слой и единственный выходной слой. Эти обозначения просто определяют положение слоя во множестве слоев: вход является началом (снизу или слева), а выход – концом (вверху или справа).

Как мы обсуждали раньше, входной слой на является слоем нейронов. Это лишь концептуальный хранитель для входных данных. Входной слой обычно создается и сохраняется для нас автоматически библиотекой глубокого обучения, и мы редко имеем с ним дело напрямую. Тем не менее мы должны помнить о нем хотя бы потому, что иногда мы хотим обработать наш вход, перед тем как остальная сеть получит его, поэтому мы помещаем некоторого рода шаг обработки между входным слоем и первым слоем нейронов в сети. В противоположность этому выходной слой содержит нейроны, и мы создаем его явным образом при построении сети. Его тип и структура полностью зависят от нас. Часто формальное определение для выхода отсутствует. Какой бы слой мы не поместили вверху нашего набора слоев, мы называем его «выходным слоем» для данной архитектуры.

20.3.1. Входной слой

Входной слой обычно не показывается на диаграмме архитектуры глубокого обучения. Он просто является памятью, хранящей входные данные. Заметим, что это не нейроны, поскольку входной слой не обрабатывается. Его можно представлять себе просто как некоторое собрание ячеек памяти, каждая из которых хранит одно число входа. Рисунок 20.4 иллюстрирует идею.

Некоторые авторы используют термин «входной слой», имея в виду первый слой обработки в сети, поэтому надо быть настороже, встречая термин «первый слой», и быть уверенным, в каком смысле используется этот термин.

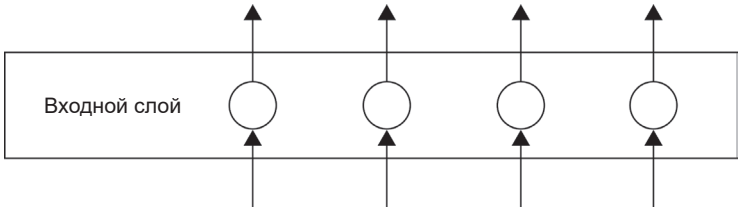

Рис. 20.4 ❖ Входной слой является просто хранилищем, в котором мы временно храним входные данные

20.3.2. Выходной слой

Выходной слой – это то, где результаты сети взаимодействуют с внешним миром.

Когда мы строим нашу архитектуру, то выбираем число нейронов в выходном слое, соответствующее решаемой задаче.

Если это задача регрессии с одним-единственным численным выходом, то в этом слое будет один нейрон, и величина этого нейрона будет нашим предсказанием.

Если мы строим бинарный классификатор, то у нас есть выбор. Величины, близкие к 0, означают вход одного класса, в то время как величины, близкие к единице, означают вход другого класса. В качестве альтернативы мы можем иметь два выходных нейрона, один для каждой категории. Обычно находят, какой нейрон имеет большую величину, и приписывают ему соответствующую категорию входа.

Классификатор с большим количеством классов чаще будет иметь столько выходов, сколько классов. Например, предположим, что мы пытаемся распознать прописные буквы латинского алфавита. Мы тогда должны иметь 26 выходных нейронов, один для каждой буквы, что обеспечит величину для каждой буквы. Мы можем выбрать выход с наибольшей величиной как лучший выбор для этой категории входа. На рис. 20.5 показана идея. Если мы интерпретируем эти выходы как вероятности, то можем пропустить их через софтмакс, как это обсуждалось в главе 17.

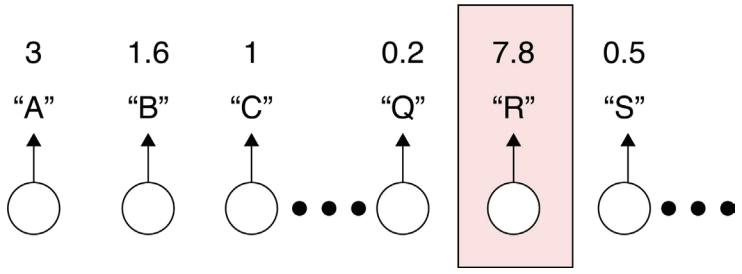

Рис. 20.5 ❖ Если мы классифицируем отдельные буквы, то можем иметь 26 выходов, каждый из которых даст величину для данной буквы. Здесь показано, что буква «R» имеет наибольшую величину

20.4. Обзор слоев глубокого обучения

Большинство библиотек предлагает широкий выбор типов слоев. В этом разделе мы рассмотрим некоторые наиболее употребительные и полезные типы. Поскольку структура каждой библиотеки ориентируется на определенный тип слоев и то,

как они работают, будет наиболее простым в качестве примера сосредоточиться на какой-нибудь одной библиотеке. Мы выбрали в качестве таковой Keras [Keras16], потому что она предлагает хороший выбор и будет детально рассмотрена в главах 23 и 24. Даже для этой библиотеки наш обзор не будет исчерпывающим.

Мы сосредоточимся здесь только на базисной структуре и функции каждого слоя. Большинство слоев имеет опции параметров, которые могут быть использованы для настройки их поведения, если нас не устраивает их работа по умолчанию.

Одной из опций, доступных для большинства обрабатывающих слоев, является функция активации, применяемая на выходе нейронов. Вспомним из главы 17, что функция активации является небольшим нелинейным преобразованием, которое применяется на выходе каждого нейрона перед передачей его дальше. Хотя теоретически могут быть применены различные функции активации в каждом нейроне каждого слоя, это используется редко. На практике мы обычно применяем одну и ту же функцию активации в каждом нейроне данного слоя.

Последующий обзор сознательно сделан кратким. Мы вернемся к некоторым из этих слоев в отдельной главе, посвященной их принципам и использованию. Другие будут рассмотрены более детально при рассмотрении их применения.

20.4.1. Полносвязные слои

Полносвязные слои (также называемые **FC** (Fully–Connected), или **плотными** (dense)) – это набор нейронов, каждый из которых получает вход от каждого нейрона предыдущего слоя. Например, если имеется 4 нейрона в плотном слое и 4 нейрона в предыдущем слое, то каждый нейрон этого слоя будет иметь 4 входа, один от предыдущего слоя, или всего $4 \times 4 = 16$ соединений.

На рис. 20.6 показана диаграмма полносвязного слоя с тремя нейронами, следующего за слоем из 4 нейронов.

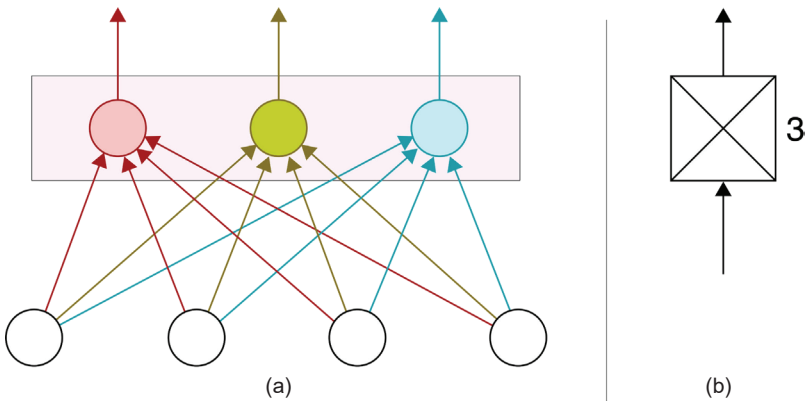

Рис. 20.6 ❖ Полносвязный слой: (a) цветные нейроны образуют полносвязный слой. Каждый нейрон в верхнем слое получает вход от каждого нейрона в предыдущем слое; (b) наш схематический символ для полносвязного слоя

На рис. 20.6 (b) показан схематический символ, который мы будем использовать для плотных слоев. Идея заключается в том, что два нейрона вверху и внизу

символа и линии являются 4 соединениями в слое. Когда это существенно, это также то, где мы определяем функцию активации.

Плотные слои используются во многих местах сетей глубокого обучения. Некоторые сети, называемые **полностью соединенными сетями**, или **многослойными персептронами** (Multy-layers Perceptrons, MLPs), представляют собой просто группу плотных слоев.

20.4.2. Функции активации

Многие слои позволяют нам определить функцию активации, которую следует использовать для всех нейронов в данном слое. Выбор часто определяется списком, включающим многие из функций активации, которые мы видели в главе 17: ReLU, сигмоид и тангенциальная.

Но вместо этого мы можем создать наши нейроны вообще без функций активации, а затем включить «слой» функций активации после них. Нейроны сами по себе не имеют функций активации, но затем их выходы подаются в слой функции активации, где последние применяются. Это имеет тот же результат, как и применение индивидуальных функций активации, но позволяет, если это более удобный способ, разбить данное действие на два шага.

При решении задачи классификации мы часто заканчиваем сеть плотным слоем, который имеет столько выходов, сколько имеется категорий. Затем мы подаем выходы на слой софтмакс, с которым мы познакомились в главе 17. Будет выполнено масштабирование наших выходов, обеспечивающее равенство суммы всех выходов 1. Эти шаги позволят нам интерпретировать выходы софтмакс как вероятности. На рис. 20.7 показана данная идея.

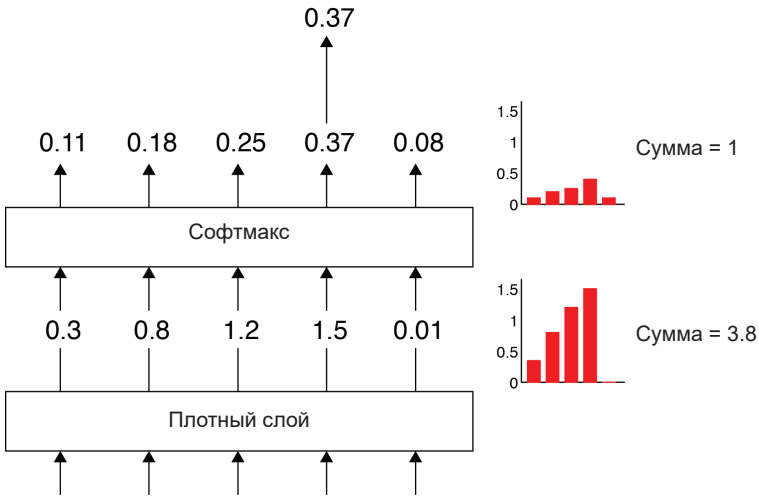

Рис. 20.7 ❖ Операция софтмакс изменяет список чисел так, что они представляют вероятности. Используя математические преобразования, мы получаем результат, при котором каждый выход лежит в пределах от 0 до 1 и сумма всех выходов равна 1. В этом примере мы имеем пять величин, являющихся выходами полносвязного, или плотного, слоя. Их величины лежат между 0.01 и 1.5, давая в сумме примерно 3.8. После преобразования с помощью софтмакс они лежат между 0 и 1 и в сумме дают 1

20.4.3. Выпадающий слой

Избыточное обучение является проблемой для многих нейронных сетей. Как только сеть начинает запоминать обучающие данные и потому начинать избыточно обучаться, мы обычно останавливаем обучение. Любой метод, который мы можем использовать, чтобы отложить начало избыточного обучения, является методом **регуляризации**. Техника регуляризации очень важна, потому что эти методы позволяют нам обучать сети более длительный период до начала избыточного обучения, обеспечивая лучшее обучение.

Одним из таких методов для задержки начала избыточного обучения называется **выпадение**, которое может быть использовано в глубоких сетях путем включения **выпадающих слоев** [Stivastrava14]. Выпадающий слой не содержит нейронов. В отличие от слоя софтмакс, выпадающий слой даже ничего не вычисляет. Вместо этого он просто временно отсоединяет некоторые нейроны от предшествующего слоя. Слой активен только в процессе обучения. Когда мы используем сеть для предсказания, выпадающий слой не работает.

Выпадающий слой руководствуется параметром, определяющим процент нейронов, который следует задействовать. В начале каждой эпохи мы случайным образом выбираем процент нейронов в предыдущем слое и временно отключаем их входы от других нейронов. По существу, эти нейроны изолированы, каждый из них представляет изолированный участок. Поскольку они отсоединены, то не участвуют ни в вычислениях, ни в обновлениях. Когда эпоха закончена и веса обновлены, нейроны и все их соединения восстанавливаются.

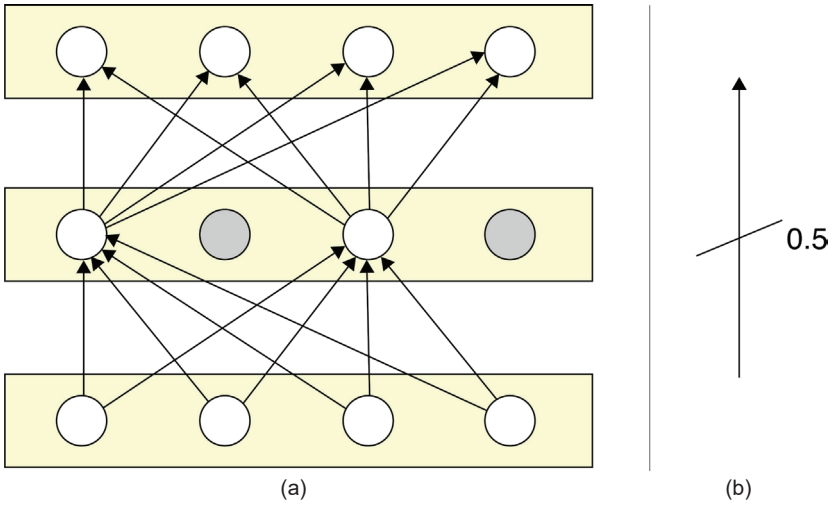

Рис. 20.8 ❖ Как работает выпадение. Здесь мы применяем выпадение к среднему слою: (a) мы здесь явно не отображаем выпадающий слой, а лишь демонстрируем его эффект. Мы установили процент выпадения равным 0.5, то есть 50 % из 4 нейронов в среднем слое выбраны для отключения перед началом эпохи. Они помечены здесь серым цветом. Фактически они не являются частью сети. Когда эпоха закончена, все их входные и выходные связи восстанавливаются; (b) наше схематическое обозначение для выпадающего слоя – диагональная косая черта слева внизу направо вверх. Справа указывается пропорция нейронов, которая выбрана для отсоединения

Например, предположим, что процент выпадения равен 20 % (обычная величина) и предыдущий слой имеет 100 нейронов. Тогда вначале каждой эпохи мы случайным образом выбираем 20 нейронов (поскольку 20 % из 100 равно 20), и они временно отсоединяются в сети. Когда эпоха закончилась, мы восстанавливаем эти нейроны и их связи так, как будто они никогда на были отключены. В начале следующей эпохи мы снова случайным образом выбираем новый набор нейронов и временно удаляем их, повторяя процесс с каждой новой эпохой. На рис. 20.8 эта идея проиллюстрирована графически.

Выпадение – это способ предотвратить превращение любого из наших нейронов в слишком специализированного. Предположим, что один нейрон в системе классификации фотографий становится слишком специализированным для обнаружения, скажем, глаз у кошек. Это полезно для распознавания кошачьих мордочек, но бесполезно для всех других фотографий, которые система могла бы классифицировать. Данный род специализации легко приведет к избыточному обучению. Если разные нейроны в сети хорошо находят только одну или две характеристики в обучающих данных, то они могут хорошо выполнять свои функции, потому что выделяют отличительные детали, которые они обучены различать. Но система в целом будет работать плохо, когда встретится с новыми данными, в которых отсутствуют точные сигналы, для которых эти нейроны специализированы.

Чтобы избежать такого рода специализации, ведущей к избыточному обучению, мы должны прежде всего избежать такой специализации. Этому и служит выпадение.

При случайном удалении нейронов иногда будут выбраны и специализированные нейроны. Это значит, что оставшиеся нейроны будут вынуждены работать и принять на себя ответственность за работу удаленных нейронов. Когда специализированные нейроны будут снова подсоединены, их специализированная работа уже не будет востребована в той же степени, поэтому они станут более свободными для общей работы. Оба этих шага обеспечивают нейронам возможность быть более разносторонними в ответной реакции и потому менее склонными к переобучению.

Другими словами, выпадение помогает избавиться от избыточного обучения путем распределения обучения среди всех нейронов.

Выпадающий слой может следовать за любым слоем с нейронами.

20.4.4. Групповая нормализация

Другим методом регуляризации является метод, называемый **групповая нормализация**, часто также называемый **бачнорм** (batchnorm) [Ioffe15]. Так же, как и выпадение, бачнорм может применяться как слой, который мы включаем в сеть, и этот слой тоже не содержит нейронов. В отличие от выпадения, бачнорм производит некоторые вычисления, хотя для него не существует параметров и он не управляется нами.

Бачнорм используется, чтобы модифицировать величины, которые поступают из вычислительного слоя, такого как полносвязный слой, или одного из слоев, который мы увидим далее. Это выглядит странно, поскольку главная цель наших слоев – обучиться вырабатывать выходные величины, которые ведут к хорошим результатам. Почему мы хотим модифицировать эти выходы?

Оказывается, что некоторые модификации выходов слоя могут создать такие числа лучше согласованными с последующими вычислениями. Например, предположим, что мы делим все выходящие из слоя величины на постоянное число, скажем, 2. Тогда каждая величина, которая передается следующему слою, будет половиной того, что было бы в противном случае. Поскольку *все* величины делятся на 2, он меняет свой *абсолютный* размер, но не меняет свой *относительный* размер. Поэтому величина, в 3 раза большая, чем другие величины, которые в 3 раза больше, так и останется больше в 3 раза.

Если мы подключим математику, это изменение в масштабе не меняет относительного соотношения выходов. Например, если мы осуществляем классификацию, такое деление оставит нейрон с наибольшей величиной в сети нейроном с наибольшей величиной после масштабирования, и мы будем по-прежнему идентифицировать ту же категорию на выходе.

Поэтому какой смысл в таком масштабировании? Оно будет продолжать следовать по сети как слишком большое число. Вспомним, что при нашем обсуждении регуляризации в главе 9 мы видели, что поддержание величин небольшими помогает предотвратить избыточное обучение.

Техника, которую использует бачнорм, чтобы масштабировать величины, подобна той, которую мы видели в главе 12. Там мы видели, что когда данные подготавливаются для машинного обучения, мы часто хотим нормализовать их. То есть мы перемещаем и масштабируем величины так, чтобы их среднее было равно 0, а стандартное отклонение равно 1. Интуиция подсказывает, что нормализация входного слоя – хорошая идея (и она таковая и есть), но также хорошей идеей будет нормализовать данные и внутренних слоев. Бачнорм это и делает, перемещая и масштабируя данные одного слоя так, что он имеет среднее, равное 0, и стандартное отклонение, равное 1, создавая тем самым правильные пропорции и предоставляя следующему слою условия для более простой и эффективной работы. Это объясняет «норм» в «бачнорм». Часть наименования «бач» (batch – группа, пакет) появилась потому, что мы применяем этот шаг после сбора всех групповых значений выходов из слоя, из которого мы следуем. Как обсуждалось в главе 18, на практике эта «группа» почти всегда относится к мини-пакетам, значительно меньшим, чем вся обучающая выборка. Мы применяем эту технику, размещая слой бачнорм после слоя вычислений, но *перед* функцией активации. Таким образом мы создаем вычислительный слой без функции активации, следующей за слоем бачнорм, и затем с функцией активации, как показано на рис. 20.9.

Итак, слой бачнорм собирает вместе все величины, выходящие из слоя, как группа. Затем он нормализует все величины, с тем чтобы они все имели среднее, равное нулю, и стандартную девиацию, равную 1.

Это предотвращает перемещение выходящих величин либо в очень положительную, либо в очень отрицательную область или чрезмерное расширение (либо сжатие). Данное действие помогает удерживать величины ближе к области, где функция активации имеет наибольшую полезную нелинейность.

Короче говоря, это позволяет избавляться от избыточного обучения, давая возможность проводить обучение дольше.

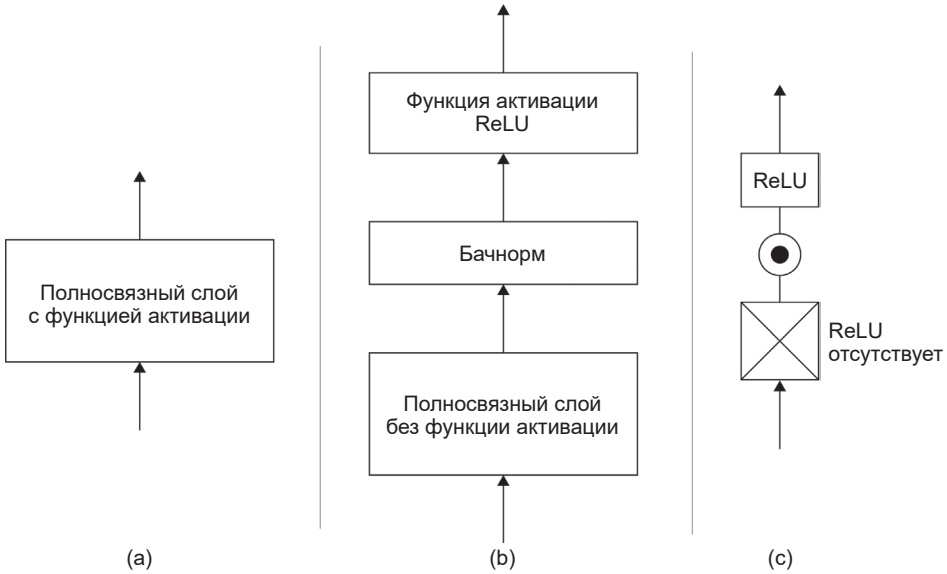

Рис. 20.9 ❖ Как применяется слой групповой нормализации (бачнорм): (a) прежнее изображение, показывающее полносвязный слой с функцией активации ReLU, который мы хотим регуляризировать с помощью бачнорма; (b) изображение «после». Мы помещаем функцию активации в ее собственный слой и подсоединяем к нему бачнорм. Затем мы подсоединяем слой бачнорм, а к нему функцию активации ReLU, которую мы имели прежде; (c) наша схематическая версия сети в (b). Она начинается с полносвязного слоя без функции активации, за которым следует иконка слоя бачнорм, а затем слой функции активации ReLU. Иконка бачнорм показывает, что данные нормализованы, и изображается черным кружком в центре круга большего размера

20.4.5. Свертка

Слой свертки наиболее успешно используется для обработки 2D-изображений. Например, нейронная сеть, основанная на свертке, используется для идентификации лиц на фотографиях. Свертка также полезна для 1D-последовательностей, 3D-объемов и даже для данных больших размерностей. Более подробно мы рассмотрим свертку в главе 21.

Сейчас давайте остановимся на общем смысле, чтобы получить общую картину. Мы рассмотрим свертку на примере изображения 2D. В добавление к этому входному изображению мы также создадим другое маленькое изображение 3 × 3 пикселей и назовем его **фильтром**.

Теперь мы можем двигать этот маленький прямоугольный фильтр по всему входному изображению. Для каждого пикселя на входе мы будем центрировать наше 3 × 3-изображение относительно него и будем умножать друг на друга величину каждого пикселя в 3 × 3-фильтре на величину пикселя входного изображения под ним. Мы будем складывать эти величины, и они станут величиной этого пикселя в выходном изображении.

Если у нас будет два фильтра, то мы будем создавать два выходных изображения, одно для каждого фильтра. Мы можем иметь 3 фильтра или 300 фильтров, и каждый из них будет подвергаться такой же обработке и создавать выходное изображение.

Интуиция подсказывает, что при подобной обработке каждый фильтр «ищет» специфические характеристики в изображении, скажем, полосы тигра или человеческую родинку. Поскольку фильтры перемещаются по всему изображению, они могут найти элементы, которые они ищут на всем изображении. Если мы используем умножение слоев свертки один за другим, они могут работать иерархически с каждым слоем, используя результаты предыдущего слоя, чтобы помочь найти большее изображение.

Рисунок 20.10 показывает пример слоя свертки 2D в действии для изображения 5×4 с двумя различными фильтрами 3×3. Мы по очереди располагаем центр фильтра 3×3 над каждым пикселем изображения 5×4, умножаем каждый пиксель фильтра на величину изображения под ним и складываем все результаты, чтобы получить величину для этого пикселя на выходе фильтра. Мы пока предполагаем, что если какой-нибудь из пикселей в фильтре попадает на край входного изображения, мы просто используем 0 для этой несуществующей величины входного изображения.

Мы рассмотрим данный процесс более детально в главе 21, которая целиком посвящена сетям, основанным на свертке.

Свертка размерностью 2D очень хорошо работает с изображениями, и часто ее используют в сетях для обработки изображений. Многие библиотеки также предлагают свертку с другим числом измерений, 1D и 3D.

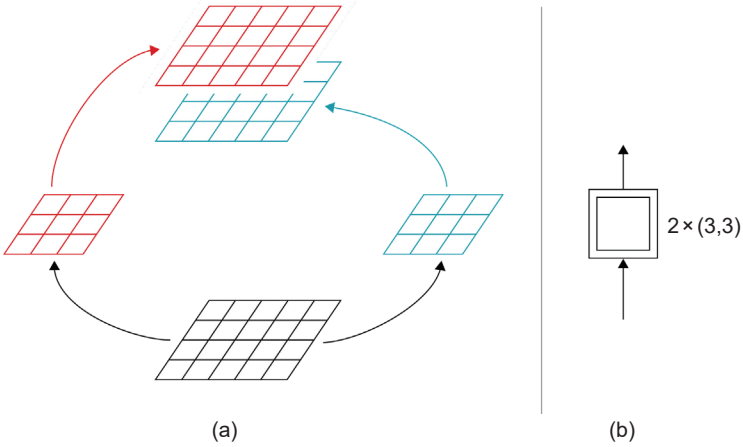

Рис. 20.10 ❖ Слой свертки применяет 1 или больше изображений к входному изображению. Мы имеем здесь начальное изображение 5×4 и фильтр 3×3: (a) мы перемещаем красный 3×3-фильтр над входным изображением, располагая центр фильтра над каждым пикселем на входе. Мы умножаем его величину на величины пикселей под ним. Сложение результатов дает нам величину в красном выходном изображении. Мы осуществляем такой же процесс для синего фильтра, создавая синее выходное изображение; (b) наш символ для слоя свертки: маленький квадрат внутри большего по размерам квадрата, что предполагает, что маленький квадрат – это маленькое изображение, движущееся над большим изображением. Справа мы показываем, как много этих маленьких квадратиков используется, и их размер. Если необходимо, мы показываем также функцию активации для этого слоя

20.4.6. Объединение слоев

Слияние слоев позволяет нам изменить размер данных, проходящих через сеть. Этот процесс часто используется для изображений, когда мы хотим уменьшить размер изображения, чтобы можно было обработать его быстрее.

Давайте предположим, что входное изображение имеет размер 512 × 512. Это четверть миллиона пикселей, слишком много для обработки.

Было бы полезно пропустить через один слой сети изображение 512 × 512, что позволит работать с его деталями. Но затем в следующем слое можно работать с версией изображения 256 × 256 пикселей и затем с версией изображения 128 × 128 и т. д.

Чтобы сделать это, мы можем в буквальном смысле уменьшить размер изображения. Предположим, что мы рассматриваем верхний слева блок пикселей 2 × 2, изымаем из него наибольшую величину и записываем ее вверх слева нового изображения. Теперь мы перемещаем два пикселя вправо и выбираем следующий блок 2 × 2 пикселей и т. д., как показано на рис. 20.11.

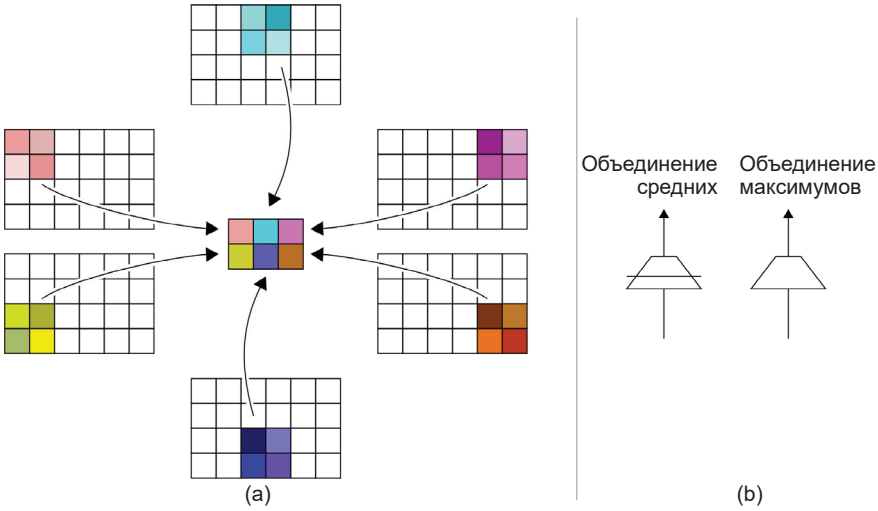

Рис. 20.11 ❖ Объединенный слой: (a) когда мы применяем объединение к изображению размерностью 2D, то собираем маленькие блоки (часто квадратные) и используем некоторую версию их данных (обычно или среднее, или один с наибольшей величиной); (b) наши схематические символы для объединения. Символ предполагает уменьшение длины стороны выхода. Две версии различаются тем, что это либо объединяемые средние, либо объединяемые максимумы

В результате мы получаем изображение со сторонами в половину размера оригинала. Если мы использовали блок 3 × 3, стороны будут иметь размер в одну треть оригинала.

Использование больших размеров блока – часто хороший способ сделать изображение маленьким. Популярным является и использование среднего величин. Большинство библиотек предлагает, по крайней мере, эти два выбора. Как и слои свертки, объединение слоев также часто доступно при разных размерностях, таких как 1D, 2D и 3D.

Объединенные слои в большинстве случаев используются после слоев свертки, мы используем их, чтобы делать все меньше и меньше версию входного изображения, используя рассмотренный выше метод. Но этот метод становится все менее

популярным в связи (как мы увидим в главе 21) с тем, что объединенные слои могут уменьшать размер своих изображений в процессе работы. Разрешение слоям свертки изменять размер изображений часто более эффективно, чем объединение слоев, и может дать нам более быстрый результат и более быстрое обучение.

20.4.7. Рекуррентные слои

В мире много интересных последовательностей. Взять хотя бы цены акций в течение нескольких дней, ноты, образующие песню, измерения, выполняемые оборудованием, кадры, создающие фильм, или слова, написанные или произнесенные на каком-то языке. Естественно спросить обо всех таких последовательностях, являются ли они похожими на другие последовательности (например, эта книга написана тем же автором, что и та, другая книга?), как они могут быть выражены в других терминах (например, перевод строки из слов на другой язык) или как они поведут себя в будущем (например, какова будет цена акций завтра?).

Нейронные сети, созданные из слоев, с которыми мы встречались до сих пор, могут с полным правом быть предметом для такого рода вопросов [vandenOord16]. Но традиционная проблема с подобными сетями в том, что они не имеют **памяти**, что означает, что они бедны с точки зрения использования **контекста**, который важен, когда мы хотим ответить на вопросы, касающиеся последовательности. Например, если мы переводим отрывок речи, то можем рассматривать каждый раз только одно слово. Контекст позволяет нам рассматривать слова, существовавшие до него, и может быть сделан более точный перевод.

Мы можем устранить такую нехватку памяти заменой нашего базового нейрона нейроном с несколько более сложной структурой обработки, называемой **рекуррентный элемент**, или **рекуррентная ячейка**. Мы можем построить сеть глубокого обучения, используя смесь стандартных слоев и слоев, превращенных в рекуррентные ячейки. Если рекуррентные ячейки являются важной частью сети, то ее часто называют **рекуррентная нейронная сеть**, или **RNN** (Recurrent Neural Network).

RRN могут ответить на все вопросы, поставленные выше, и многие другие. Они используются в диапазоне от перевода с языков до автоматической подписи фотографий и даже написания прозы в стиле известного писателя.

Отметим, что рекуррентные ячейки не являются родственными концепции рекурсии, которая звучит очень похоже, но представляет собой совершенно другую идею. Рекурсия включает функции, которые вызывают сами себя, часто с модифицированными аргументами. Рекуррентность относится к повторению, или повторяемости, действий. Рекуррентная ячейка повторяет данную операцию снова и снова, поэтому это не рекурсия. Мы детально рассмотрим рекуррентные сети в главе 22. А сейчас достаточно знать, что они обеспечивают нас гибким способом добавлять память и контекст в наши сети.

На рис. 20.12 (а) показан один из стандартных способов изображения рекуррентной ячейки, рядом со схематическим символом для всех слоев с ней. Мы также показываем символ для рекуррентной ячейки, возвращающей последовательность, которую мы будем рассматривать в главе 22.

Когда мы включаем рекуррентный слой в нашу сеть, то определяем, сколько памяти хотим использовать в нашей ячейке. Мы отложим детальное рассмотрение этого до главы 22, где увидим, что существует широкий выбор того, как встраивать и использовать подобные слои.

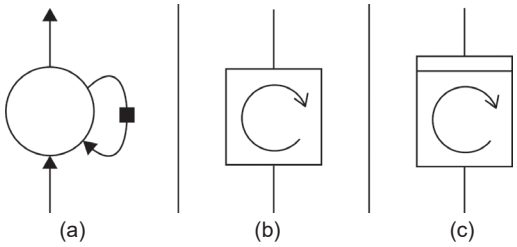

Рис. 20.12 ❖ Рекуррентная ячейка: (a) типичный вариант изображения рекуррентной ячейки. Черный квадратик представляет один шаг памяти; (b) наше схематическое отображение слоя с такой ячейкой; (c) схематическое отображение для рекуррентной ячейки, возвращающей последовательность

20.4.8. Другие применяемые слои

Существует много способов, которые мы могли бы хотеть использовать для преобразования данных в процессе их следования по нашей сети. Чтобы сохранить метафору «стопка слоев», мы можем внедрить любое из этих преобразований в его собственный слой и просто добавить этот слой в стопку при построении нашей архитектуры.

Подобно свертке и бачнорм-слоям, которые мы уже рассмотрели, эти слои не имеют нейронов (или рекуррентных ячеек) и не содержат весов, которые обновляются. Поэтому эти слои не обучаются и не изменяются с течением времени. Они просто выполняют потребительские функции, которые помогают нам формировать и модифицировать наши данные, когда они текут от одного слоя вычислений к следующему. Называть эти потребительские операции «слоями» – может быть, натяжка, но это удобно с точки зрения представления сетей как стопки слоев, что стало уже стандартной конвенцией.

В дополнение к свертке, бачнорму и объединению слоев существует несколько других популярных категорий применяемых слоев. Давайте рассмотрим их.

Нормализующий слой стремится модифицировать перемещающиеся данные, чтобы регуляризировать сеть или сохранить небольшими веса, не допуская начала избыточного обучения. Бачнорм-слой, который мы рассматривали выше, является примером слоя нормализации.

Шумовой слой добавляет случайные величины в каждую часть данных, которая проходит через него. Это может помочь в случаях избыточного обучения. Наряду с применяемыми другими способами он предотвращает излишнюю специализацию нейронов, при которой нейрон всегда отвечает одним и тем же образом на те же входные данные.

Слой преобразования формы позволяет нам изменять форму тензора, следующего через него. Например, мы можем иметь 3D-тензор с формой $10 \times 3 \times 5$, состоящий всего из 150 элементов, и мы хотим снизить размерность до 2D-сетки и представить данные в виде изображения. Мы можем использовать слой преобразования формы, объявив необходимость интерпретировать его как тензор размера 10×15. Вспомним, что вся идея «формы» заключается в интерпретации слоем входных данных, или, проще говоря, в том, как обрабатываются элементы тензоров разной формы.

Обрезающий слой особенно полезен при работе с изображениями. Он просто вырезает прямоугольник из изображения и отбрасывает остальное (эквивалентно мы можем сказать, что он убирает кромку и оставляет внутренности).