Оглавление

Об авторах

Антонио Джулли – директор по программному обеспечению и предприниматель с тягой к созданию и управлению глобальными инновационными технологическими компаниями. Специализируется в области поисковых систем, онлайновых сервисов, машинного обучения, информационного поиска, аналитики и облачных вычислений. Профессиональный опыт приобретал в шести странах Европы и Америки. Антонио работал в должности исполнительного директора, генерального директора, технического директора, вице-президента и руководителя группы в различных отраслях: от издательского бизнеса (Elsevier) до интернет-технологий для конечного пользователя (Ask.com и Tiscali) и НИОКР в сфере высоких технологий (Microsoft и Google).

Выражаю благодарность своему талантливому соавтору, Суджиту Палу, за неизменное стремление помочь, не требуя ничего взамен. Я очень ценю его преданность командной работе, благодаря чему эта книга и смогла стать чем-то стоящим.

Благодарю также Франсуа Шолле и многих людей, внесших вклад в Keras, за то, что они тратили свое время и силы на создание впечатляющего инструментария для глубокого обучения, который прост в использовании и не требует сверхъестественных усилий.

Спасибо также нашим редакторам из издательства Packt, Дивия Пуджари, Черил Дса и Динешу Павару, и рецензентам из Packt и Google за поддержку и ценные предложения. Без вас эта книга не состоялась бы.

Я также благодарен своему начальнику, Брэду, и коллегам Майку и Коррадо из Google, которые подвигли меня написать эту книгу, читали ее черновые варианты и высказывали свое мнение.

Еще я признателен кофейне Same Fusy в Варшаве, где у меня впервые появилась мысль написать эту книгу, когда я наслаждался чашечкой чая, выбранной из весьма обширного меню.

Это место обладает особой магией, и я горячо рекомендую его всем, ищущим, где бы подстегнуть свое воображение (`http://www.samefusy.pl/`*).*

Далее я хочу поблагодарить отдел кадров в Google, пошедший навстречу моему пожеланию отдать все отчисления от продажи этой книги на стипендии представителям этнических меньшинств.

Спасибо моим друзьям, Эрику, Лауре, Франческо, Этторе и Антонелле, которые поддерживали меня, когда я в том нуждался. Дружба – большая ценность, и вы – мои настоящие друзья.

Спасибо моему сыну Лоренцо, который побудил меня устроиться в Google, моему сыну Леонардо за постоянное стремление открывать что-то новое и моей дочери Авроре, благодаря которой я встречаю каждый день с улыбкой. И наконец, спасибо моему отцу Элио и матери Марии за их любовь.

Суджит Пал – руководитель отдела технологических исследований в Elsevier Labs, работает над созданием интеллектуальных систем поиска по содержимому и метаданным. В область его интересов входят информационный поиск, онтологии, обработка естественных языков, машинное обучение и распределенная обработка. В настоящее время занимается классификацией и установлением сходства изображений с применением моделей глубокого обучения. До этого работал в промышленности безрецептурных медицинских препаратов, где участвовал в построении онтологической системы семантического поиска, организации контекстной рекламы и платформ обработки данных. Ведет посвященный технологиям блог *Salmon Run*.

Выражаю благодарность своему соавтору, Антонио Джулли, пригласившему меня принять участие в написании книги. Это редкая возможность, благодаря которой я многому научился. К тому же, если бы не он, меня бы здесь в буквальном смысле не было.

Хочу поблагодарить Рона Дэниэла, директора Elsevier Labs, и Брэдли П. Аллена, главного архитектора в Elsevier, которые познакомили меня с глубоким обучением и заставили поверить в возможности этой технологии.

Благодарю также Франсуа Шолле и многих людей, внесших вклад в Keras, за то, что они тратили свое время и силы на создание впечатляющего инструментария для глубокого обучения, который прост в использовании и не требует сверхъестественных усилий.

Спасибо также нашим редакторам из издательства Packt, Дивия Пуджари, Черил Дса и Динешу Павару, и рецензентам из Packt и Google за поддержку и ценные предложения. Без вас эта книга не состоялась бы.

Я также признателен коллегам и начальникам, с которыми работал на протяжении своей жизни, а особенно тем, кто верил в меня и помогал мне строить свою извилистую профессиональную карьеру.

Наконец, я благодарен своей семье, которая на протяжении нескольких месяцев мирилась с тем, как я разрывался между работой, этой книгой и семьей – именно в таком порядке. Надеюсь, вы согласитесь, что дело того стоило.

О рецензенте

Ник Макклюр в настоящее время работает старшим специалистом по анализу данных в компании PayScale Inc., Сиэтл, штат Вашингтон, США. До этого работал в компании Zillow and Caesars Entertainment. Защитил диссертации по прикладной математике в Университете штата Монтана, колледже Святого Бенедикта и Университете Святого Иоанна. Ник – автор книги «TensorFlow Machine Learning Cookbook», вышедшей в издательстве Packt Publishing.

Его страсть – изучать и делиться знаниями об аналитике, машинном обучении и искусственном интеллекте. Плоды своих размышлений Ник публикует в блоге на сайте `fromdata.org` и в своем аккаунте в Твиттере по адресу `@nfmcclure`.

Предисловие

Книга, которую вы держите в руках, – краткое, но обстоятельное введение в современные нейронные сети, искусственный интеллект и технологии глубокого обучения. Она написана специально для программистов и специалистов по анализу и обработке данных.

Назначение

В книге представлено более 20 работоспособных нейронных сетей, написанных на языке Python с использованием модульной библиотеки Keras, работающей поверх библиотек TensorFlow от Google или Theano от компании Lisa Lab.

Читатель шаг за шагом познакомится с алгоритмами обучения с учителем, начиная с простой линейной регрессии и классического многослойного перцептрона и кончая более сложными глубокими сверточными сетями и порождающими состязательными сетями. В книге также рассматриваются алгоритмы обучения без учителя: автокодировщики и порождающие сети. Подробно объясняется, что такое рекуррентные сети и сети с долгой краткосрочной памятью (long short-term memory, LSTM). Описывается функциональный API библиотеки Keras и обсуждается, как расширить Keras, если встретится задача, для которой в ней нет готового решения. Также рассматриваются более крупные и сложные системы, состоящие из описанных ранее структурных блоков. В заключение дается введение в технологию глубокого обучения с подкреплением и ее применение к построению игр со встроенным искусственным интеллектом.

Если говорить о практических приложениях, то в книгу включен код программ для классификации новостей по заранее заданным категориям, для синтаксического анализа текста, для анализа эмоциональной окраски текста, для синтеза текстов и частеречной разметки. Не оставлена без внимания также обработка изображений: распознавание рукописных цифр, классификация изображений по категориям и распознавание объектов с последующим аннотированием изображений. Из области анализа зву-

ковых сигналов взят пример распознавания слов, произносимых несколькими лицами. Техника обучения с подкреплением применяется для построения глубокой сети Q-обучения, способной автономно играть в игры.

Суть книги составляют эксперименты. Каждая сеть представлена несколькими вариантами, качество которых постепенно улучшается путем изменения входных параметров, формы сети, вида функции потерь и применяемых алгоритмов оптимизации. В ряде случаев приводятся сравнительные результаты обучения на CPU и GPU.

Чем глубокое обучение отличается от машинного обучения и искусственного интеллекта

Искусственный интеллект (**ИИ**) – очень широкая область исследований, посвященная *когнитивным* способностям машин: обучение определенному поведению, упреждающее взаимодействие с окружающей средой, способность к логическому выводу и дедукции, компьютерное зрение, распознавание речи, решение задач, представление знаний, восприятие действительности и многое другое (за подробностями отсылаем к книге S. Russell, P. Norvig «Artificial Intelligence: A Modern Approach», Prentice Hall, 2003). Менее формально под ИИ понимается любая ситуация, в которой машины имитируют *интеллектуальное* поведение, считающееся присущим человеку. Искусственный интеллект заимствует методы исследования из информатики, математики и статистики.

Машинное обучение (**МО**) – отрасль ИИ, посвященная тому, как обучать компьютеры решению конкретных задач без программирования (см. книгу C. M. Bishop «Pattern Recognition and Machine Learning», Springer, 2006). Основная идея МО заключается в том, что можно создавать алгоритмы, способные обучаться на данных и впоследствии давать предсказания. Существует три основных вида МО. В случае обучения с учителем машине предъявляются данные и правильные результаты, а цель состоит в том, чтобы машина обучилась на этих примерах и смогла выдавать осмысленные результаты для данных, которые раньше не видела. В случае обучения без учителя машине предъявляются только сами данные, а она должна выявить структуру без постороннего вмешательства.

В случае обучения с подкреплением машина ведет себя как агент, который взаимодействует с окружающей средой и обучается находить варианты поведения, приносящие вознаграждение.

Глубокое обучение (**ГО**) – подмножество методов МО, в которых применяются искусственные нейронные сети (ИНС), построенные на базе аналогии со структурой нейронов человеческого мозга (см. статью Y. Bengio «Learning Deep Architectures for AI», Found. Trends, vol. 2, 2009). Неформально говоря, слово «глубокий» подразумевает наличие большого числа слоев в ИНС, но его интерпретация со временем менялась. Если еще четыре года назад считалось, что 10 слоев достаточно, чтобы называть сеть *глубокой*, то теперь глубокой обычно называется сеть, содержащая сотни слоев.

Глубокое обучение – это настоящее цунами (см. статью C. D. Manning «Computational Linguistics and Deep Learning» в журнале «Computational Linguistics», vol. 41, 2015) в области машинного обучения в том смысле, что сравнительно небольшое число хитроумных методов с огромным успехом применяется в самых разных областях (обработка изображений, текста, видео и речи, компьютерное зрение), что позволило добиться значительного прогресса по сравнению с результатами, достигнутыми за предшествующие десятки лет. Своими успехами ГО обязано также наличию больших объемов обучающих данных (например, набора ImageNet в области обработки изображений) и относительно дешевых графических процессоров (GPU), позволяющих построить очень эффективную процедуру вычислений. В компаниях Google, Microsoft, Amazon, Apple, Facebook и многих других методы глубокого обучения постоянно используются для анализа больших

массивов данных. Теперь эти знания и навыки вышли за рамки чисто академических исследований и стали достоянием крупных промышленных компаний. Они стали неотъемлемой составной частью современной программной продукции, и владение ими обязательно для программиста. В этой книге не предполагается наличие у читателя специальной математической подготовки. Однако же знакомство с языком Python является необходимым условием.

Краткое содержание книги

В главе 1 «Основы нейронных сетей» излагаются основные сведения о нейронных сетях.

В главе 2 «Установка Keras и описание API» описано, как установить Keras в облаке AWS, Microsoft Azure, Google Cloud или на вашу собственную машину, а также дается краткий обзор различных API библиотеки Keras.

Глава 3 «Глубокое обучение с применением сверточных сетей» знакомит с понятием сверточной сети. Это фундаментальное новшество стало причиной успеха глубокого обучения в применении к различным предметным областям, от видео до речи, выйдя далеко за пределы обработки изображений, где эта идея первоначально зародилась.

Глава 4 «Порождающие состязательные сети и WaveNet» содержит введение в порождающие состязательные сети, используемые для синтеза данных, похожих на порождаемые людьми. Мы представляем глубокую нейронную сеть WaveNet, предназначенную для высококачественной имитации человеческого голоса и звучания музыкальных инструментов.

В главе 5 «Погружения слов» обсуждаются методы глубокого обучения, служащие для выявления связей между словами и группировки похожих слов.

В главе 6 «Рекуррентные нейронные сети» рассматривается класс нейронных сетей, оптимизированных для обработки последовательных данных, в т. ч. текста.

Глава 7 «Дополнительные модели глубокого обучения» содержит краткий обзор функционального API Keras, регрессионных сетей, автокодировщиков и т. д.

В главе 8 «Искусственный интеллект играет в игры» вы узнаете о глубоком обучении с подкреплением и о том, как Keras позволяет

его использовать для построения глубоких сетей, умеющих играть в аркадные игры.

Приложение содержит сводку обсуждаемых в книге тем и информацию о нововведениях в версии Keras 2.0.

Что необходимо для чтения книги

Вам понадобится следующее программное обеспечение:

- TensorFlow версии 1.0.0 или выше;
- Keras версии 2.0.2 или выше;
- Matplotlib версии 1.5.3 или выше;
- Scikit-learn версии 0.18.1 или выше;
- NumPy версии 1.12.1 или выше.

К оборудованию предъявляются следующие требования:

- 32- или 64-разрядная архитектура;
- CPU с таковой частотой не ниже 2 ГГц;
- оперативная память объемом не меньше 4 ГБ;
- не менее 10 ГБ свободного места на диске.

На кого рассчитана эта книга

Если вы – специалист по анализу и обработке данных со знанием машинного обучения или занимаетесь программированием ИИ и знакомы с нейронными сетями, то эта книга станет неплохой отправной точкой для овладения методами глубокого обучения с применением библиотеки Keras. Знание Python – обязательное условие.

Графические выделения

В этой книге тип информации обозначается шрифтом. Ниже приведено несколько примеров с пояснениями.

Фрагменты кода внутри абзаца, имена таблиц базы данных, папок и файлов, URL-адреса, данные, которые вводит пользователь, и адреса в Твиттере выделяются следующим образом: «Кроме того, мы загружаем истинные метки соответственно в `Y_train` и `Y_test` и применяем к ним унитарное кодирование».

Кусок кода выглядит так:

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(12, input_dim=8, kernel_initializer='random_uniform'))
```

Желая привлечь внимание к части кода, мы выделяем ее полужирным шрифтом:

```
# 10 outputs
# final stage is softmax
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()
```

Входная и выходная информация командных утилит выглядит так:

```
pip install quiver_engine
```

Новые термины и **важные фрагменты** выделяются полужирным шрифтом. Например, элементы графического интерфейса в меню или диалоговых окнах выглядят в книге так: «Первоначально наша простая сеть имеет верность **92.22 %**, т. е. примерно 8 из 100 рукописных символов распознаются неправильно».

Таким значком обозначаются предупреждения и важные примечания.

Таким значком обозначаются советы и рекомендации .

Отзывы

Мы всегда рады отзывам читателей. Расскажите нам, что вы думаете об этой книге – что вам понравилось или, быть может, не понравилось. Читательские отзывы важны для нас, так как помогают выпускать книги, из которых вы черпаете максимум полезного для себя.

Чтобы отправить обычный отзыв, просто пошлите письмо на адрес `feedback@packtpub.com`, указав название книги в качестве темы. Если вы являетесь специалистом в некоторой области и хотели

бы стать автором или соавтором книги, познакомьтесь с инструкциями для авторов по адресу www.packtpub.com/authors.

Поддержка клиентов

Счастливым обладателям книг Packt мы можем предложить ряд услуг, которые позволят извлечь из своего приобретения максимум пользы.

Загрузка кода примеров

Вы можете скачать код примеров к этой книге из своей учетной записи на сайте http://www.packtpub.com. Если книга была куплена в другом месте, зайдите на страницу http://www.packtpub.com/support, зарегистрируйтесь, и мы отправим файлы по электронной почте.

Для скачивания файлов с кодом выполните следующие действия:

1. Зарегистрируйтесь или зайдите на наш сайт, указав свой адрес электронной почты и пароль.
2. Наведите мышь на вкладку **SUPPORT** в верхней части страницы.
3. Щелкните по ссылке **Code Downloads & Errata**.
4. Введите имя книги в поле **Search**.
5. Выберите интересующую вас книгу.
6. С помощью выпадающего меню укажите, где вы приобрели книгу.
7. Нажмите **Code Download**.

Загруженный файл можно распаковать, воспользовавшись последними версиями программ:

- WinRAR / 7-Zip для Windows;
- Zipeg / iZip / UnRarX для Mac;
- 7-Zip / PeaZip для Linux.

Код к этой книге имеется также на странице сайта GitHub по адресу https://github.com/PacktPublishing/Deep-Learning-with-Keras. По адресу https://github.com/PacktPublishing/ размещен также код и видео к другим книгам из нашего обширного каталога. Полюбопытствуйте!

Загрузка цветных иллюстраций

Мы также предлагаем PDF-файл, содержащий цветные изображения, встречающиеся в книге. Цвет поможет лучше понять, как изменяются результаты. Этот файл можно скачать по адресу `https://www.packtpub.com/sites/default/files/downloads/DeepLearningwithKeras_ColorImages.pdf`.

Опечатки

Мы проверяли содержимое книги со всем тщанием, но какие-то ошибки все же могли проскользнуть. Если вы найдете в нашей книге ошибку, в тексте или в коде, пожалуйста, сообщите нам о ней. Так вы избавите других читателей от разочарования и поможете нам сделать следующие издания книги лучше. При обнаружении опечатки просьба зайти на страницу `http://www.packtpub.com/support`, выбрать книгу, щелкнуть по ссылке **Errata Submission Form** и ввести информацию об опечатке. Проверив ваше сообщение, мы поместим информацию об опечатке на нашем сайте или добавим ее в список замеченных опечаток в разделе Errata для данной книги.

Список ранее отправленных опечаток можно просмотреть, выбрав название книги на странице `http://www.packtpub.com/books/content/support`. Запрошенная информация появится в разделе **Errata**.

Нарушение авторских прав

Незаконное размещение защищенного авторским правом материала в Интернете – проблема для всех носителей информации. В издательстве Packt мы относимся к защите прав интеллектуальной собственности и лицензированию очень серьезно. Если вы обнаружите незаконные копии наших изданий в любой форме в Интернете, пожалуйста, незамедлительно сообщите нам адрес или название веб-сайта, чтобы мы могли предпринять соответствующие меры.

Просим отправить ссылку на вызывающий подозрение в пиратстве материал по адресу `copyright@packtpub.com`.

Мы будем признательны за помощь в защите прав наших авторов и содействие в наших стараниях предоставлять читателям полезные сведения.

Вопросы

Если вас смущает что-то в этой книге, вы можете связаться с нами по адресу `questions@packtpub.com`, и мы сделаем все возможное для решения проблемы.

Глава 1

Основы нейронных сетей

Искусственные нейронные сети (для краткости *нейросети* или просто *сети*) – это класс моделей машинного обучения, в основе которых лежат исследования центральной нервной системы млекопитающих. Нейросеть состоит из нескольких взаимосвязанных *нейронов*, организованных в *слои*, которые обмениваются между собой сообщениями (как говорят, *возбуждаются*) при выполнении определенных условий. Первые исследования относятся к 1950-м годам, когда было введено понятие перцептрона (см. статью F. Rosenblatt «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain», Psychological Review, vol. 65, pp. 386–408, 1958), двуслойной сети для выполнения простых операций. Затем в конце 1960-х годов был предложен *алгоритм обратного распространения* для эффективного обучения многослойных сетей (см. статьи P. J. Werbos «Backpropagation through Time: What It Does and How to Do It», Proceedings of the IEEE, vol. 78, pp. 1550–1560, 1990 и G. E. Hinton, S. Osindero, Y. W. Teh «A Fast Learning Algorithm for Deep Belief Nets, Neural Computing, vol. 18, pp. 1527–1554, 2006). В некоторых работах утверждается, что эти методы корнями уходят глубже, чем принято считать (см. статью J. Schmidhuber «Deep Learning in Neural Networks: An Overview», by, vol. 61, pp. 85–117, 2015). Нейронные сети были предметом интенсивных научных исследований до 1980-х годов, когда на первый план вышли другие, более простые, подходы. Однако с середины 2000-х годов отмечается возрождение интереса к этой теме в связи с прорывным алгоритмом быстрого обучения, предложенным Дж. Хинтоном (см. S. Leven «The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting», Neural Networks, vol. 9, 1996, и D. E. Rumelhart, G. E. Hinton, R. J. Williams «Learning Representations by Backpropagating Errors», vol. 323, 1986), и появлением (примерно в 2011 году) графических процессоров для массово-параллельных численных расчетов.

Эти достижения проложили дорогу современному *глубокому обучению*, классу нейронных сетей, для которых характерно большое число слоев и которые способны обучать весьма изощренные модели на основе иерархии уровней абстрагирования. Несколько лет назад *глубокой* считалась сеть с 3–5 слоями, теперь же их число возросло до 100–200.

Обучение путем последовательного абстрагирования напоминает модели зрения в мозге человека, эволюционировавшие миллионы лет. Человеческая система зрения действительно состоит из нескольких уровней. Глаз соединен с областью мозга, которая называется **первичной зрительной корой**, или **зрительной корой V1** и занимает задний полюс затылочной доли каждого полушария. Эта область имеется у многих млекопитающих и играет важную роль в распознавании простых образов и обработке информации об изменении ориентации, пространственной частоте и цвете. Согласно некоторым оценкам, первичная зрительная кора содержит примерно 140 миллионов нейронов и 10 миллиардов связей между ними. Кора V1 соединяется с областями V2, V3, V4, V5 и V6, отвечающими за всю последующую обработку изображений и распознавание более сложных объектов: фигур, лиц, животных и многого другого. Такая многослойная организация явилась результатом множества попыток, которые природа предпринимала на протяжении сотен миллионов лет. Согласно оценкам, кора головного мозга человека насчитывает порядка 16 миллиардов нейронов, и примерно 10-25 % из них относятся к зрению (см. статью S. Herculano-Houzel «The Human Brain in Numbers: A Linearly Scaled-up Primate Brain», vol. 3, 2009). Глубокое обучение заимствовало идею многослойной организации у зрительной системы человека: наружные слои искусственных нейронов обучаются базовым свойствам изображений, а более глубокие обрабатывают более сложные концепции.

В этой книге рассмотрено несколько важных аспектов нейронных сетей на примере кода с использованием минималистской и весьма эффективной библиотеки глубокого обучения Keras, написанной на языке Python и работающей поверх библиотеки TensorFlow от Google (см. `https://www.tensorflow.org/`) или библиотеки Theano, разработанной в Монреальском университете (см. `http://deeplearning.net/software/theano/`). Итак, приступим.

В этой главе будут рассмотрены следующие темы:

- перцептрон;
- многослойный перцептрон;
- функции активации;
- градиентный спуск;
- стохастический градиентный спуск;
- алгоритм обратного распространения.

Перцептрон

Перцептрон – это простой алгоритм, который получает входной вектор x, содержащий n значений ($x_1, x_2, \ldots, x_n$), которые часто называются входными признаками, или просто признаками, и выдает на выходе 1 (да) или 0 (нет). Формально говоря, мы определяем функцию:

$$f(x) = \begin{cases} 1, & \text{если } wx + b > 0 \\ 0 & \text{в противном случае} \end{cases}$$

Здесь w – вектор весов, wx – скалярное произведение $\sum_{j=1}^{m} w_j x_j$, а b – смещение. Как мы знаем из геометрии, уравнение $wx + b = 0$ определяет граничную гиперплоскость, положение которой изменяется в зависимости от значений w и b. Если x лежит выше этой гиперплоскости (в двумерном случае – прямой), то ответ положительный, иначе отрицательный. Очень простой алгоритм! С помощью перцептрона нельзя выразить ответ «*может быть*». Он может ответить *да* (1) или *нет* (0), если мы знаем, как определить w и b, а это и есть процесс обучения, который обсуждается в следующих разделах.

Первый пример кода с использованием Keras

Исходным строительным блоком Keras является модель, а простейшая модель называется **последовательной**. В Keras последовательная модель представляет собой линейный конвейер (стек) слоев нейронной сети. В следующем фрагменте определен один слой с 12 нейронами, который ожидает получить 8 входных переменных (признаков):

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(12, input_dim=8, kernel_initializer='random_uniform'))
```

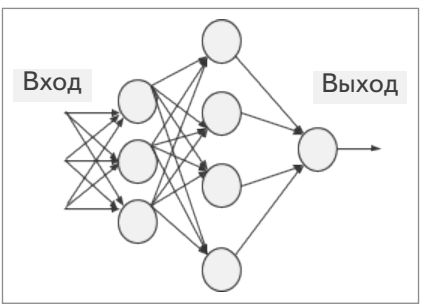
На этапе инициализации каждому нейрону можно назначить вес. Keras предлагает несколько вариантов, перечислим наиболее употребительные:

- `random_uniform`: веса инициализируются равномерно распределенными случайными значениями из диапазона (*–0.05*, *0.05*) Иными словами, любое значение из этого интервала выбирается с одинаковой вероятностью;
- `random_normal`: веса инициализируются нормально распределенными случайными значениями со средним 0 и стандартным отклонением *0.05*. Те, кто не знает, что такое нормальное распределение, могут представлять себе симметричную *колоколообразную* кривую;
- `zero`: все веса инициализируются нулями.

Полное описание параметров имеется на странице `https://keras.io/initializations/`.

Многослойный перцептрон – первый пример нейросети

В этой главе мы определим первый пример сети с несколькими линейными слоями. Исторически перцептроном называлась модель с единственным линейным слоем, поэтому модель с несколькими слоями логично назвать **многослойным перцептроном** (МСП). На рисунке ниже показана нейронная сеть общего вида с одним входным, одним промежуточным и одним выходным слоем.



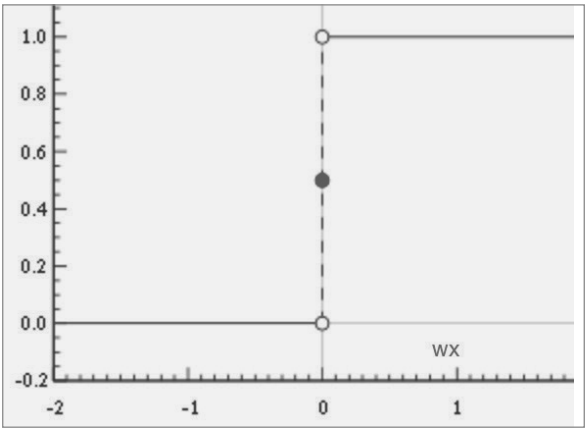
Каждый узел первого слоя получает входной сигнал и возбуждается в соответствии с заранее определенными локальными граничными условиями. Выход первого слоя подается на вход второго, а выход второго – последнему слою, состоящему всего из одного нейрона. Интересно, что такая многослойная организация отдаленно напоминает работу человеческой системы зрения, о чем мы уже говорили.

Эта *сеть* плотная в том смысле, что каждый нейрон одного слоя связан со всеми нейронами предыдущего слоя и всеми нейронами следующего слоя.

Проблемы обучения перцептрона и их решение

Рассмотрим один нейрон; каков оптимальный выбор веса w и смещения b? В идеале мы хотели бы предъявить набор обучающих примеров и поручить компьютеру выбрать вес и смещение, так чтобы ошибка при вычислении результата была минимальна. Переходя к конкретике, предположим, что имеется набор изображений кошек и другой набор изображений, на которых кошек нет. Для простоты будем считать, что каждый нейрон анализирует только один входной пиксель. Мы хотели бы, чтобы в процессе обработки изображений компьютером наш нейрон изменял свой вес и смещение таким образом, чтобы число изображений, ошибочно распознанных как не кошки, со временем уменьшалось. Этот подход кажется интуитивно очевидным, но для него требуется, чтобы малое изменение веса (и/или смещения) приводило к малому изменению результата.

Если имеется большой скачок на выходе, то *прогрессивное* обучение невозможно (разве что пробовать все возможные направления – это называется исчерпывающим поиском – не зная, достигаем ли мы какого-нибудь улучшения). В конце концов, дети ведь учатся постепенно. К сожалению, для перцептрона такое «постепенное» поведение не характерно. Перцептрон выдает значение *0* или *1*, разница между ними велика, и это никак не способствует его обучению, что видно из следующего рисунка:



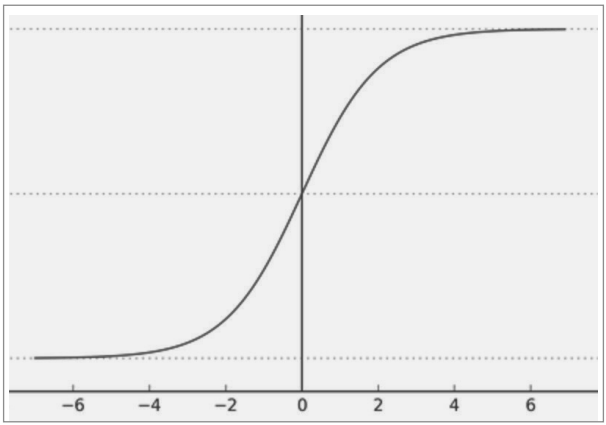
Нам нужно что-то более гладкое – непрерывная, дифференцируемая функция, монотонно возрастающая от 0 до 1.

Сигмоида

Сигмоида определяется следующим образом:

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

Как видно из графика ниже, она непрерывна и изменяется от 0 до 1, когда аргумент пробегает область определения $(-\infty, \infty)$:



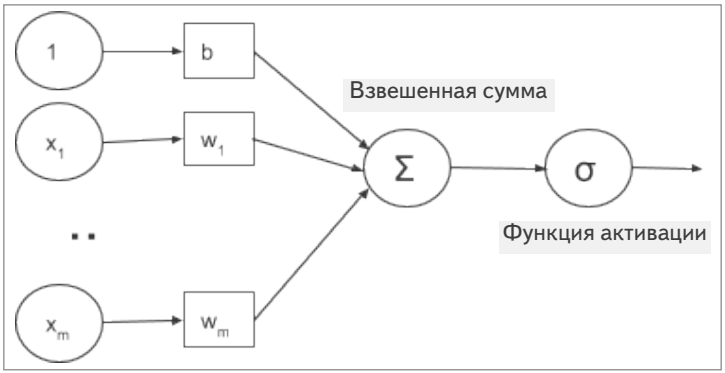
Нейрон может воспользоваться сигмоидой для вычисления нелинейной функции $\sigma(z = wx + b)$. Отметим, что когда величина $z = wx + b$ очень велика и положительна, $e^{-z} \to 0$, так что $\sigma(z) \to 1$, а когда эта величина велика по модулю и отрицательна, то $e^{-z} \to \infty$, так что $\sigma(z) \to 0$. Иными словами, нейрон с сигмоидной функцией активации ведет себя подобно перцептрону, но изменяется плавно и может порождать такие значения, как 0.5539 или 0.123191. В некотором смысле сигмоидный нейрон умеет давать ответ «может быть».

Блок линейной ректификации

Сигмоида – не единственная гладкая функция, применяемая в нейронных сетях. В последнее время стала очень популярна совсем простая функция, называемая блоком линейной ректификации (rectified linear unit, ReLU), поскольку в экспериментах она дает замечательные результаты. ReLU определяется формулой $f(x) = \max(0, x)$, а ее график показан на рисунке ниже. Как видим, она равна нулю для отрицательных значений x и линейно возрастает для положительных.

Функции активации

Сигмоида и ReLU называются *функциями активации*. В разделе «Тестирование различных оптимизаторов в Keras» мы увидим, что непрерывное изменение, характерное для этих функций, крайне важно для разработки алгоритмов обучения, которые адаптируются постепенно, стремясь уменьшить ошибку сети. На рисунке ниже показана схема применения функции активации σ к входному вектору $(x_1, x_2, \ldots, x_m)$, вектору весов $(w_1, w_2, \ldots, w_m)$, смещению b и сумматору Σ:

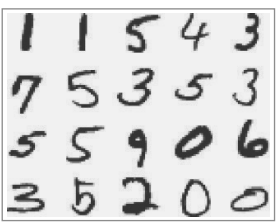


Keras поддерживает несколько функций активации, их полный перечень приведен на странице `https://keras.io/activations/`.

Реальный пример – распознавание рукописных цифр

В этом разделе мы построим сеть, умеющую распознавать рукописные цифры. Для этого будем использовать набор данных MNIST (см. `http://yann.lecun.com/exdb/mnist/`), включающий 60 000 обучающих и 10 000 тестовых примеров. В обучающих примерах человеком проставлены правильные ответы. Например, с изображением рукописной цифры три ассоциирована метка 3.

Когда имеется набор данных, содержащий правильные ответы, говорят об *обучении с учителем*. Обучающие примеры используются для обучения сети. С тестовыми примерами также ассоциированы правильные ответы, но идея в том, чтобы притвориться, будто они неизвестны, дать сети возможность сделать предсказание, а затем, сравнив его с правильным ответом, оценить, насколько хорошо сеть научилась распознавать цифры. Так что тестовые примеры применяются только для проверки сети – что и не удивительно.



Все изображения в наборе MNIST полутоновые, размера 28 × 28 пикселей. На рисунке приведено несколько примеров.

Унитарное кодирование

Во многих приложениях удобно преобразовывать категориальные (нечисловые) признаки в числовые. Например, категориальный признак – цифру, принимающую значение *d* от 0 до 9, – можно представить бинарным вектором длины 10, в котором *d*-ый элемент равен 1, а все остальные – нулю. Такое представление называется *унитарным кодированием* (one-hot encoding) и очень часто применяется в добыче данных, когда алгоритм обучения рассчитан на работу с числами.

Определение простой нейронной сети в Keras

Воспользуемся библиотекой Keras, чтобы определить сеть, распознающую рукописные цифры из набора MNIST. Начнем с очень простой нейросети и постепенно будем ее улучшать.

Keras предоставляет средства для загрузки набора данных и разбиения его на обучающий, `X_train`, и тестовый, `X_test`. Для поддержки вычислений на GPU данные преобразуются к типу `float32` и нормируются на интервал [0, 1]. Кроме того, в `Y_train` и `Y_test` мы загружаем правильные метки и применяем к ним унитарное кодирование. Вот как выглядит код:

```
from __future__ import print_function
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # для воспроизводимости результатов

# сеть и ее обучение
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
NB_CLASSES = 10 # количество результатов = числу цифр
OPTIMIZER = SGD() # СГС-оптимизатор, обсуждается ниже в этой главе
N_HIDDEN = 128
VALIDATION_SPLIT=0.2 # какая часть обучающего набора зарезервирована для контроля

# данные: случайно перетасованы и разбиты на обучающий и тестовый набор
#
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# X_train содержит 60000 изображений размера 28x28 -->
преобразуем в массив 60000 x 784 RESHAPED = 784
```

```
#
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# нормировать
#
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# преобразовать векторы классов в бинарные матрицы классов
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

Во входном слое с каждым пикселем изображения ассоциирован один нейрон, т. е. всего получается 28 × 28 = 784 нейрона.

Обычно значения, ассоциированные с пикселями, нормируются с целью привести их к диапазону [0, 1] (это значит, что яркость каждого пикселя делится на максимально возможную яркость 255). На выходе получается 10 классов, по одному для каждой цифры.

Последний слой состоит из единственного нейрона с функцией активации softmax, являющейся обобщением сигмоиды. Softmax *сплющивает* k-мерный вектор, содержащий произвольные вещественные числа, в k-мерный вектор вещественных чисел из интервала (0, 1). В нашем случае она агрегирует 10 ответов, выданных предыдущим слоем из 10 нейронов:

```
# 10 выходов
# на последнем этапе softmax
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()
```

Определенную таким образом модель необходимо откомпилировать, т. е. привести к виду, допускающему исполнение базовой библиотекой (Theano или TensorFlow). Перед компиляцией необходимо принять несколько решений:

- выбрать *оптимизатор*, т. е. конкретный алгоритм, который будет обновлять веса в процессе обучения модели;
- выбрать целевую функцию, которую оптимизатор использует для навигации по пространству весов (часто целевая

функция называется также *функцией потерь*, а процесс оптимизации – *минимизацией* потери);
- оценить качество обученной модели.

Перечислим несколько распространенных целевых функций (полный перечень целевых функций, поддерживаемых Keras, приведен на странице `https://keras.io/objectives/`):

- **Среднеквадратическая ошибка** (**СКО**): это усредненная сумма разностей квадратов между предсказанными и истинными значениями. Если обозначить Υ вектор n предсказаний, а Y – вектор n наблюдаемых значений, то среднеквадратическая ошибка равна

$$CKO = \frac{1}{n}\sum_{i=1}^{n}(\Upsilon - Y)^2$$

Если предсказание сильно отличается от истинного значения, то возведение в квадрат делает отличие еще более явственным.

- **Бинарная перекрестная энтропия:** если модель предсказывает значение p, тогда как истинное значение равно t, то бинарная перекрестная энтропия равна:

$$-t\ log(p) - (1 - t)\ log(1 - p)$$

Эта целевая функция подходит для предсказания бинарных меток.

- **Категориальная перекрестная энтропия:** это логарифмическая потеря в случае нескольких классов. Если модель предсказывает значения $p_{i,j}$, тогда как истинные значения равны $t_{i,j}$, то категориальная перекрестная энтропия равна:

$$L_i = -\Sigma_j\ t_{i,j}\ log(p_{i,j})$$

Эта целевая функция подходит для предсказания многоклассовых меток. Она же по умолчанию используется совместно с функцией активации softmax.

Ниже перечислено несколько популярных показателей качества (полный список см. на странице `https://keras.io/metrics/`):

- **верность:** отношение числа правильных предсказаний к общему числу меток;
- **точность:** доля правильных ответов модели;
- **полнота:** доля обнаруженных истинных событий.

Показатели качества похожи на целевые функции, различаются они только тем, что показатели используются не для обучения модели, а для оценки ее качества. Компиляция модели в Keras производится просто:

```
model.compile(loss='categorical_crossentropy',
optimizer=OPTIMIZER,
metrics=['accuracy'])
```

Для обучения откомпилированной модели служит функция `fit()`, принимающая в частности следующие параметры:

- `epochs`: число периодов – сколько раз обучающий набор предъявляется модели. На каждой итерации оптимизатор пытается подкорректировать веса, стремясь минимизировать целевую функцию;
- `batch_size`: сколько обучающих примеров должен увидеть оптимизатор, прежде чем он обновит веса.

Обучить модель в Keras очень просто. Допустим, что выполняется `NB_EPOCH` итераций:

```
history = model.fit(X_train, Y_train,
   batch_size=BATCH_SIZE, epochs=NB_EPOCH,
   verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
```

Мы зарезервировали часть обучающего набора для контроля. Идея в том, чтобы отложить часть обучающих данных для контроля качества в процессе обучения. Эта рекомендуемая практика для всех задач машинного обучения, и далее мы будем ее придерживаться.

После того как модель обучена, ее следует проверить на тестовом наборе, который содержит ранее не предъявлявшиеся примеры. Таким образом, мы сможем получить минимальное значение, достигаемое целевой функцией, и наилучшее значение показателя качества.

Подчеркнем, что обучающий и тестовый набор не должны пересекаться. Не имеет никакого смысла оценивать модель на примере, который она видела во время обучения. Смысл обучения в том, чтобы модель могла обобщаться на ранее не встречавшиеся данные, а не в том, чтобы она запоминала то, что уже известно.

```
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

Вы только что определили свою первую нейронную сеть на Keras, можете принимать поздравления. Всего несколько строк кода – и компьютер умеет распознавать рукописные цифры. Теперь выполним этот код и оценим качество.

Прогон простой сети Keras и создание эталона для сравнения

На рисунке ниже показано, что происходит во время прогона программы:

```
code — -bash — 118×71
gulli-macbookpro:code gulli$ python keras_MINST_V1.py
Using TensorFlow backend.
60000 train samples
10000 test samples
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 7850
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
48000/48000 [==============================] - 0s - loss: 1.4102 - acc: 0.6554 - val_loss: 0.9073 - val_acc: 0.8244
Epoch 2/200
48000/48000 [==============================] - 0s - loss: 0.8006 - acc: 0.8279 - val_loss: 0.6625 - val_acc: 0.8567
Epoch 3/200
48000/48000 [==============================] - 0s - loss: 0.6467 - acc: 0.8495 - val_loss: 0.5650 - val_acc: 0.8704
Epoch 4/200
48000/48000 [==============================] - 0s - loss: 0.5728 - acc: 0.8600 - val_loss: 0.5112 - val_acc: 0.8778
Epoch 5/200
48000/48000 [==============================] - 0s - loss: 0.5280 - acc: 0.8677 - val_loss: 0.4767 - val_acc: 0.8822
```

Сначала распечатывается архитектура сети, мы видим типы слоев, форму выхода, количество оптимизируемых параметров и характер связей между слоями. Затем сеть обучается на 48 000 примерах, а 12 000 примеров зарезервировано для контроля. Построенная нейронная сеть тестируется на 10 000 примерах. Как видим, Keras использует для вычислений базовую библиотеку TensorFlow. Пока что не будем вдаваться в детали обучения, но отметим, что программа выполнила 200 итераций и с каждым разом верность улучшалась.

По завершении обучения мы проверяем модель на тестовом наборе и видим, что на обучающем наборе получена верность 92.36%, на контрольном – 92.27%, а на тестовом – 92.22%.

Это значит, что доля неправильно распознанных рукописных символов составляет чуть менее одного на десять. Безусловно, этот результат можно улучшить.

```
Epoch 198/200
48000/48000 [==============================] - 0s - loss: 0.2761 - acc: 0.9230 - val_loss: 0.2762 - val_acc: 0.9224
Epoch 199/200
48000/48000 [==============================] - 0s - loss: 0.2760 - acc: 0.9231 - val_loss: 0.2762 - val_acc: 0.9223
Epoch 200/200
48000/48000 [==============================] - 0s - loss: 0.2758 - acc: 0.9236 - val_loss: 0.2761 - val_acc: 0.9227
 9888/10000 [============================>.] - ETA: 0s
Test score: 0.277792117235
Test accuracy: 0.9222
gulli-macbookpro:code gulli$
```

Улучшение простой сети в Keras посредством добавления скрытых слоев

Мы достигли верности 92.36% на обучающем наборе, 92.27% – на контрольном и 92.22% – на тестовом. Для начала неплохо, но есть возможность добиться большего. Посмотрим, как.

Первое улучшение – включить в сеть дополнительные слои. После входного слоя поместим первый плотный слой с `N_HIDDEN` нейронами и функцией активации `relu`. Этот слой называется *скрытым*, потому что он напрямую не соединен ни с входом, ни с выходом. После первого скрытого слоя добавим еще один, также содержащий `N_HIDDEN` нейронов, а уже за ним будет расположен выходной слой с 10 нейронами, которые возбуждаются, если распознана соответствующая цифра. Вот код, определяющий новую сеть:

```
from __future__ import print_function
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD
from keras.utils import np_utils
np.random.seed(1671) # для воспроизводимости результатов

# сеть и ее обучение
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
NB_CLASSES = 10 # количество результатов = числу цифр
```

```
OPTIMIZER = SGD() # СГС-оптимизатор, обсуждается ниже в этой главе
N_HIDDEN = 128
VALIDATION_SPLIT=0.2 # какая часть обучающего набора зарезервирована для контроля

# данные: случайно перетасованы и разбиты на обучающий и тестовый набор
#
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# X_train содержит 60000 изображений размера 28x28 --> преобразуем в массив
60000 x 784 RESHAPED = 784
#
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# нормировать
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# преобразовать векторы классов в бинарные матрицы классов
Y_train = np_utils.to_categorical(y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(y_test, NB_CLASSES)

# M_HIDDEN скрытых слоев
# 10 выходов
# на последнем этапе softmax
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
  optimizer=OPTIMIZER,
  metrics=['accuracy'])
history = model.fit(X_train, Y_train,
  batch_size=BATCH_SIZE, epochs=NB_EPOCH,
  verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

Выполним этот код и посмотрим, какие результаты дает такая многослойная сеть. Неплохо. Добавив два скрытых слоя, мы достигли верности 94.50% на обучающем наборе, 94.63% – на кон-

трольном и 94.41% – на тестовом, т. е. получили прирост 2.2% на тестовом наборе по сравнению с предыдущей сетью. И при этом число итераций резко уменьшилось – с 200 до 20. Хорошо, но мы хотим большего.

Если хотите, можете посмотреть, что будет, если добавить только один скрытый слой вместо двух или если добавить больше двух слоев. Оставляю это в качестве упражнения. На рисунке ниже показан результат работы последней программы.