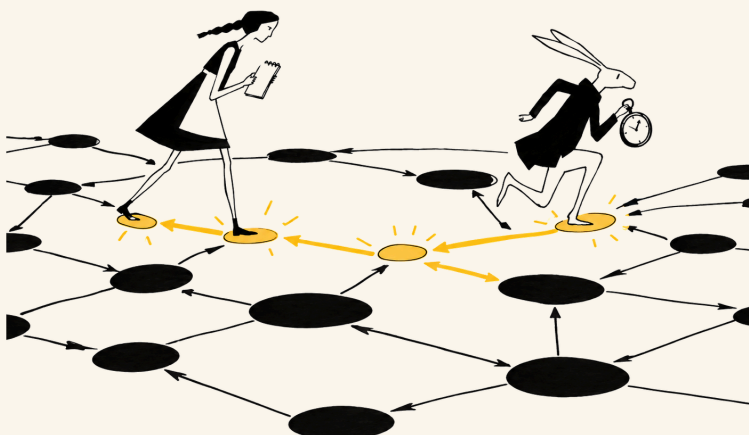


Приключения Алисы в Стране дифференцируемых чудес



Simone Scardapane · 2024

Иллюстрированные аудиолекции

Lecturized

Приключения Алисы в Стране дифференцируемых чудес

Иллюстрированные аудиолекции

Simone Scardapane · 2024

Пересказано в иллюстрированных лекциях Lecturized

lecturized.com

По оригиналу: Alice's Adventures in a Differentiable Wonderland, Volume I: A Tour of
the Land

Содержание

Несколько простых деталей, сделанных очень большими

Обучение как поиск среди ручек настройки

Старые идеи в ожидании данных

Коробки чисел и их формы

Чем эта книга является и чем не является

Благодарности, лицензия и нарезка срезов

Стрелки, длины и скалярное произведение

Таблицы чисел, которые действуют

Считаем работу

Стопки таблиц и сокращённая запись буквами

Наклон в точке

Наклоны во всех направлениях

Где земля падает быстрее всего

Спуск с горы и то, где вы оказываетесь

Шар, который помнит

От бумаги к коду

Что делает кучу примеров хорошей

Обучение без меток и повторное использование выученного

Оценка предсказания

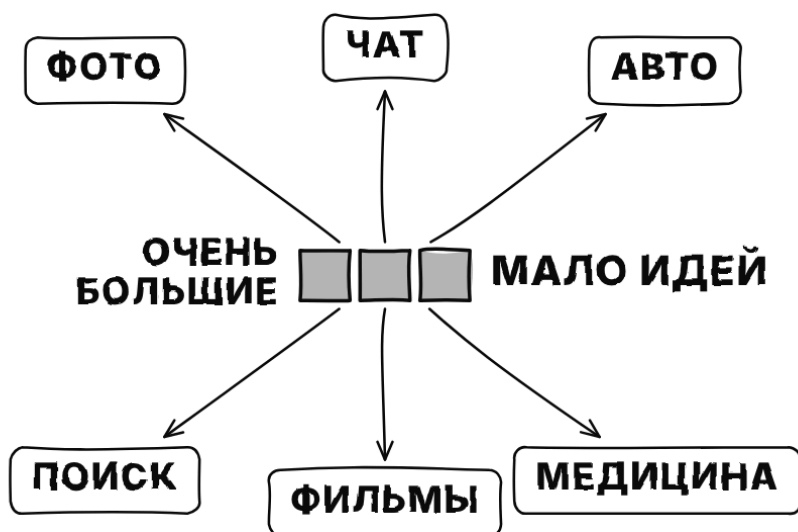
Запоминание — не обучение

Самое вероятное объяснение и много объяснений сразу

Штраф от априорного и простейший блок

Несколько простых деталей, сделанных очень большими

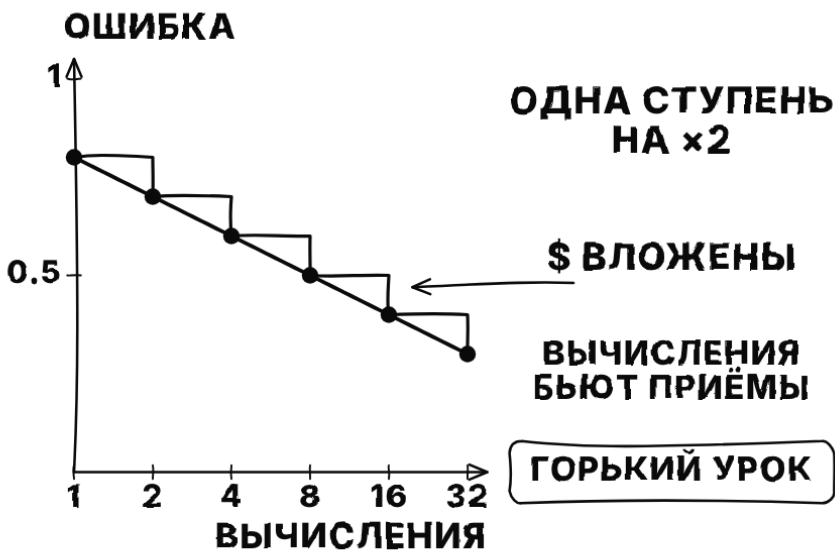
Откройте приложение с фотографиями на телефоне и введите в строку поиска слово «собака». Через секунду оно покажет вам всех собак, которых вы когда-либо фотографировали, хотя вы не подписали ни одного снимка. Попросите чат-ассистента объяснить рецепт или исправить предложение, и он ответит складными абзацами. И то и другое делают нейронные сети, как и многое из того, чего вы никогда не видите: помощь при рулении в автомобилях, ранжирование результатов поиска, фильмы, которые предлагает стриминговый сервис, даже проектирование новых молекул для лекарств. Удивительная новость, с которой начинается книга, в том, что всё это держится на очень небольшом наборе направляющих идей и строительных блоков.



Мало блоков, много применений

За последние годы изменились не столько идеи, сколько их размер. Исследователи заметили закономерность, которую назвали законами масштабирования нейросетей: почти для любой задачи, если одновременно увеличить три вещи — объём данных, вычислительную мощность и размер модели, — точность растёт, и растёт предсказуемо. Если сказать наоборот: вычислительная мощность, нужная для заданной точности, год за годом падает в постоянное число раз. Представьте лестницу, которую видно заранее: каждый раз, удваивая бюджет, вы примерно знаете, на сколько ступеней поднимётесь. Именно эта предсказуемость убедила компании вкладывать в искусственный интеллект огромные суммы. Исследователь Ричард Саттон дал этому уроку имя — горький урок: простые общие инструменты в

сочетании со всё большей вычислительной мощностью снова и снова побеждали хитрые ручные приёмы.



Лестница законов масштабирования

Примите законы масштабирования как данность, и у вас в руках окажется почти волшебный инструмент. Одним предложением: нейронную сеть настраивают так, чтобы она подражала закономерности, скрытой в большой коллекции примеров; на языке книги, она приближает распределение вероятностей по данным, взятым из него. Это легко могло бы пойти не так. Современные сети настолько велики, что могли бы просто запомнить каждый показанный им пример, как ученик, который заучивает ответы прошлогоднего экзамена наизусть и теряется, когда вопросы меняются. Сеть, которая так поступала бы, была

бы всего лишь справочной таблицей. Но обученные сети делают нечто лучшее: они обобщают, то есть справляются со случаями, которых никогда не было в их обучающих данных. А когда коллекции примеров становятся огромными, граница между знакомыми и незнакомыми случаями размывается, и самые большие модели на удивление мало опираются на зубрёжку.

ЗАПОМНИТЬ

ВХОД	ОТВЕТ
2	4
3	6
5	10
4	?

ТАБЛИЦА

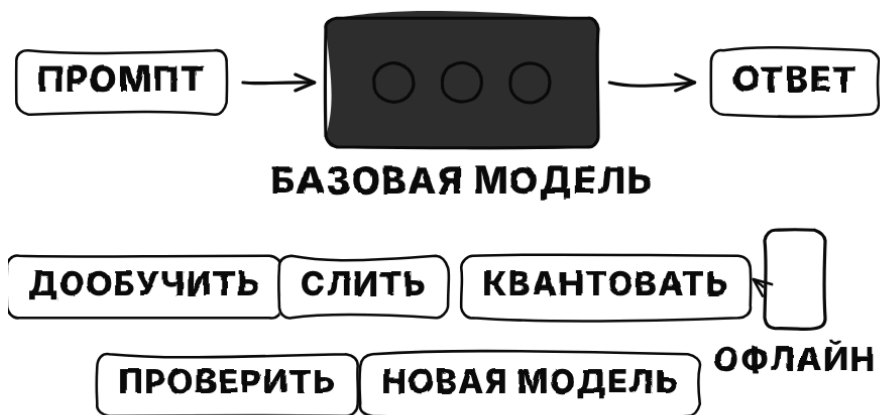
ЗАКОНОМЕРНОСТЬ



Запомнить или обобщить

Такие очень большие модели, обученные один раз и затем используемые для множества разных задач, иногда называют фундаментальными моделями. Вместе с живым сообществом открытого кода они изменили то, как люди пользуются сетями. Многие задачи больше не требуют никакого программирования: вы просто даёте модели промпт, то есть инструкции текстом или

картинками, почти так же, как набираете запрос чат-ассистенту. Сама модель остаётся чёрным ящиком — закрытой машиной, внутрь которой вы никогда не заглядываете. Книга сравнивает это с обычным программным обеспечением: вместо того чтобы писать каждую библиотеку самому на языке вроде C++, вы полагаетесь на программы, написанные другими, часто не видя их кода. Сравнение удачное, потому что лишь у немногих команд в мире есть вычислительная мощь и умение строить по-настоящему большие модели, такие как семейство языковых моделей Llama, точно так же как лишь немногие компании способны строить большие деловые программы, на которых работают фирмы.



Чёрный ящик, который можно приспособить

Но сравнение указывает и в другую сторону. Открытый код позволяет любому приспособить программу или написать новую с нуля, и так же обычный компьютер и немного изобретательности открывают с этими моделями много дверей. Вы можете дообучить модель, потренировав её ещё немного на собственных примерах; слить две модели в одну; квантовать модель, храня её числа грубее, чтобы она работала на маленьком маломощном устройстве; проверить, насколько она устойчива к каверзным входам; или вообще придумать новые виды моделей. Голосовой помощник, который отвечает вам на телефоне без интернета, часто и есть такая сжатая, квантованная модель. Для всего этого нужно заглянуть под капот: понять, как эти модели принимают и преобразуют данные, а также узнать приёмы и причуды, которые практики усвоили опытом и отладкой. Книга представляет себя как вход в этот мир, и её автор надеется, что читатель, любопытный от природы, как Алиса, получит от путешествия удовольствие.

Опыт

Откройте приложение с фотографиями на телефоне и поищите предмет, который вы никогда не подписывали, например «велосипед», «пляж» или «торт на день рождения». Заметьте, что оно находит большинство из них, и заметьте также редкие странные ошибки: лампу, принятую за торт, или собаку, принятую за плюшевого медведя. Эти ошибки — проблеск модели, которая выучила закономерность по примерам, а не список правил, и именно такую машину эта книга разбирает на части.

Запомним

Нейронные сети сегодня стоят за чат-ассистентами, фотоприложениями телефонов, автомобилями, поиском, рекомендациями и наукой, и всё же они построены из очень небольшого набора принципов. Законы масштабирования говорят, что больше данных, больше вычислительной мощности и более крупные модели вместе дают предсказуемый прирост точности, и это вызвало огромные инвестиции. Сеть настраивают так, чтобы она подражала закономерности за своими примерами; она могла бы их запомнить, но большие обученные сети обобщают на случаи, которых никогда не видели. Фундаментальные модели часто используют как чёрные ящики через промпты, но чтобы приспособлять, сжимать или изобретать модели, нужно понимать, что происходит у них внутри. Именно это понимание внутреннего устройства книга и собирается дать.

Вопросы

1. Что, по законам масштабирования нейросетей, происходит, когда данные, вычислительная мощность и размер модели растут одновременно?
 - а. Точность растёт, и на величину, которую можно предсказать заранее
 - б. Точность сначала растёт, а потом всегда падает, потому что модели запоминают

- c. Точность остаётся прежней, но модель работает быстрее
 - d. Точность растёт только для языковых задач, но не для картинок
2. Почему сеть, запомнившая все свои обучающие примеры, была бы малополезна?
- a. Ей понадобилось бы гораздо больше вычислительной мощности для ответа
 - b. Она была бы справочной таблицей, беспомощной на любом случае, которого не видела
 - c. Ей больше нельзя было бы давать текстовые промпты
 - d. Через некоторое время она забыла бы примеры
3. Чему в сравнении книги с программным обеспечением соответствует промпт к фундаментальной модели?
- a. Написанию каждой библиотеки самому на C++
 - b. Созданию собственного корпоративного делового софта
 - c. Использованию чужой программы без просмотра её исходного кода
 - d. Чтению исходного кода библиотеки с открытым кодом
4. Какая из этих задач требует заглянуть под капот модели, а не только давать ей промпты?
- a. Попросить чат-ассистента исправить предложение
 - b. Поискать в своих фотографиях слово «собака»
 - c. Набрать вопрос в поисковой системе
 - d. Квантовать модель, чтобы она работала на маленьком устройстве

Ответы

1. **a)** Законы масштабирования описывают предсказуемый прирост точности почти для любой задачи, когда растут все три вещи; именно эта предсказуемость и привела к инвестициям.
2. **b)** Запоминание превращает сеть в справочную таблицу, как ученика, знающего наизусть прошлогодние ответы; обученные сети полезны тем, что обобщают на новые случаи.
3. **c)** Промпт использует модель как чёрный ящик, точно так же как человек полагается на программы, написанные другими, не имея доступа к их коду.
4. **d)** Дообучение, слияние, квантование или проверка модели требуют понимания того, как она обрабатывает данные внутри, а именно это книга и предлагает.

Задачи

1. Знакомый говорит, что модель, обученная на миллионах фотографий, должна просто их помнить. Опираясь на сказанное в этой части, что бы вы ответили?

Подсказка: Подумайте, что сделала бы модель, которая только помнит, с фотографией, которой никогда не видела.

Ответ: Модель, которая только помнит, была бы справочной таблицей и проваливалась бы на новых фотографиях, а обученные сети распознают предметы на снимках, которых им никогда не показывали. Они обобщают, и самые большие модели на удивление мало опираются на чистое запоминание.

2. Объясните своими словами, почему законы масштабирования убедили компании тратить так много на искусственный интеллект.

Подсказка: Что значит, что прирост предсказуем ещё до того, как вы за него заплатили?

Ответ: Законы масштабирования говорят, что больше данных, вычислительной мощности и размера модели дают предсказуемый рост точности. Поэтому компания может заранее оценить, что ей купит большой бюджет, и крупная инвестиция выглядит надёжной ставкой, а не азартной игрой.

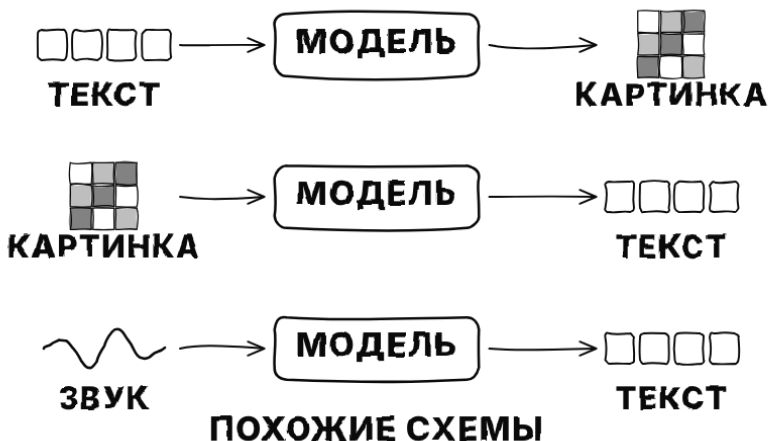
3. Назовите две вещи, которые вы могли бы сделать с готовой моделью на обычном компьютере, и объясните, почему каждая требует понимания устройства модели.

Подсказка: Посмотрите на список возможностей, который эта часть привела после сравнения с открытым кодом.

Ответ: Например, дообучить её на собственных примерах, то есть разумно изменить её внутренние числа, и квантовать её для работы на маленьком устройстве, то есть знать, какие числа можно хранить грубее, не сломав её. И то и другое действует на внутренность модели, а не только на её ответы.

Обучение как поиск среди ручек настройки

Представьте, что вам нужна программа, отличающая фотографии кошек от фотографий собак. Можно было бы попробовать написать правила: острые уши, усы, определённая форма носа. Через час вы бы застряли, потому что всегда найдётся кошка, нарушающая правила, и собака, которая им соответствует. Книга предполагает, что вы уже знаете другой путь, тот, который работает: собрать тысячи фотографий, каждая с подписью «кошка» или «собака», и дать системе подстраиваться, пока её ответы не совпадут с подписями. Это обучение с учителем, ядро того, что называют машинным обучением: собрать примеры нужного поведения, а затем обучить, то есть попросту оптимизировать, систему, пока она не станет подражать этому поведению.



Что входит, что выходит

Идея звучит почти слишком просто, и всё же она достаёт удивительно далеко. Создание картинок по описанию становится задачей собрать громадный набор изображений, каждое со своей подписью. Подражание английскому языку, задача, стоящая за чат-ассистентом, становится задачей собрать огромное количество текста и научиться предсказывать каждое предложение по предыдущим. Чтение рентгеновского снимка становится задачей держать большой архив снимков вместе с решением, которое врач принял по каждому. Рисунок в книге сортирует такие задачи по тому, что входит и что выходит. Генерация изображения берёт текст, ряд символов, и возвращает изображение — упорядоченную сетку цветных точек, называемых пикселями. Описание картинки делает обратное, от

изображения к тексту, а ответ на устный вопрос идёт от звука, ещё одного упорядоченного ряда, на этот раз чисел, к тексту. Поразительно, замечает книга, что модели для всех этих задач устроены очень похоже.

По сути, говорит книга, обучение — это поиск. Вы начинаете с программы, в которой очень много настраиваемых чисел, как у микшерного пульта с миллионами ручек, и эти ручки называются параметрами. Затем вы крутите ручки, пока программа не начнёт работать достаточно хорошо. Трудность в числе ручек: миллионы в маленькой модели, миллиарды и даже триллионы в самых больших — слишком много, чтобы перебирать настройки наугад. Фотоприложение на вашем телефоне было настроено именно так: его ручки крутили снова и снова на огромных кучах подписанных картинок.



ОШИБКА

11.5

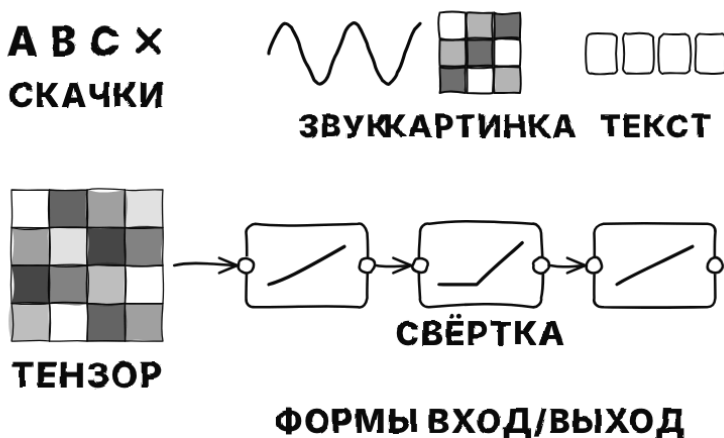
$$\text{rate} = \frac{0.5}{0.1} = 5$$

АВТОДИФФ ГРАДИЕНТНЫЙ СПУСК

Крутим ручки вниз по склону

Дифференцируемые модели делают поиск возможным, разрешая только один вид строительных блоков: математические функции, которые можно дифференцировать, то есть для каждой мы можем выяснить, как меняется её выход, когда вход меняется чуть-чуть. Для функции со многими входами набор этих скоростей изменения называется градиентом. Знать градиент значит знать сразу для каждой ручки, что произойдёт с работой программы, если эту ручку слегка подвинуть. Вот маленькая картинка. Допустим, ошибка программы равна двенадцати, и вы узнаете, что сдвиг одной конкретной ручки вверх на одну десятую снизил бы ошибку на полединицы: скорость — пять единиц ошибки на единицу ручки, так что вы знаете, что её нужно повернуть вверх, и примерно на сколько. С такой

скоростью для каждой ручки вращение ручек превращается в автоматическую процедуру, и две процедуры, на которых строится книга, называются автоматическим дифференцированием, которое вычисляет скорости, и градиентным спуском, который их использует. Первая часть книги, названная «Компас и стрелка», главы со второй по шестую, и выстраивает именно этот механизм.



Блоки, подходящие как вилки

Взгляд на сеть как на цепочку дифференцируемых блоков ставит два вопроса, и второй рисунок книги их иллюстрирует. Первый: какие виды данных могут входить и выходить? Второй: какие виды блоков мы можем использовать? Дифференцируемость — сильное требование. Буквы и целые числа перескакивают от

одного значения к другому без ничего посередине, а величину, которая скачет, нельзя подвинуть чуть-чуть, так что такие данные нельзя обрабатывать напрямую. Зато дифференцируемые модели легко обрабатывают данные, записанные как большие массивы чисел, которые книга называет тензорами, например изображение — сетку значений яркости. Эти массивы преобразуются цепочкой простых шагов, одних прямолинейных, других изогнутых, которые книга называет линейными и нелинейными преобразованиями. На рисунке каждый блок описан тем, что он принимает и что выдаёт, и эти описания решают, какие блоки можно соединять с какими, почти как формы вилок и розеток.

Вторая часть книги посвящена одному типичному блоку, свёртке, в главах с седьмой по девятую. Свёртка подходит любым данным, которые приходят упорядоченным рядом элементов: звуку, изображениям, тексту и видео. По пути книга вводит способы строить глубокие модели, где «глубокие» означает «составленные из многих шагов подряд», и несколько идей, лежащих в основе современных больших языковых моделей: разбиение текста на кусочки, называемые токенами, порождение последовательности по одному кусочку за раз, каждый по предыдущим, и запрет модели подглядывать в то, что идёт дальше. Третья часть, названная «Вниз по кроличьей норе», обращается к другим видам данных: множествам, которыми занимаются слои внимания и модели-трансформеры в главах десятой и одиннадцатой; графам, сетям связанных точек, в главе двенадцатой; и рекуррентным слоям для последовательностей во времени в главе тринадцатой.

За пределами печатных страниц книгу сопровождает сайт, где автор планирует собирать дополнительные главы на темы, не привязанные к одному виду данных: порождение новых данных, вычисления, включающие и выключающие части модели, перенос выученного с одной задачи на другую и объяснение того, чему модель научилась. Эти главы больше тянутся к текущим исследованиям, и читать их можно в любом порядке. Есть и лабораторные занятия в виде блокнотов, охватывающие большую часть книги, а также продвинутые темы, такие как контрастное обучение и слияние моделей — объединение двух обученных моделей в одну.

Запомним

Книга исходит из обучения с учителем: собрать примеры нужного поведения и обучать систему, пока она не станет им подражать, — идея, одинаково покрывающая генерацию изображений, чат-ассистентов и диагностику по рентгену. Задачи можно сортировать по тому, что входит и что выходит, и модели для них устроены поразительно похоже. Обучение — это поиск среди огромного числа настраиваемых чисел, параметров, и оно становится практичным, если строить модели только из дифференцируемых блоков. Градиент говорит сразу для каждого параметра, как меняется качество работы при его сдвиге, что и делает возможными автоматическое дифференцирование и градиентный спуск. Три части книги покрывают этот механизм, затем свёртки, затем внимание, трансформеры, графы и рекуррентные слои, а дополнительные главы и лабораторные блокноты лежат в интернете.

Вопросы

1. Почему книга называет обучение поиском?
 - а. Потому что модель ищет каждый ответ в базе примеров
 - б. Потому что обучение крутит огромное число настраиваемых чисел, пока программа не заработает хорошо

- c. Потому что программист ищет в интернете готовые правила
 - d. Потому что модель перебирает все возможные положения своих ручек по одному
2. Почему дифференцируемая модель не может работать напрямую с буквами или целыми числами?
- a. Потому что они занимают слишком много памяти
 - b. Потому что их вообще нельзя поместить в тензор
 - c. Потому что они скачут от одного значения к другому, и их нельзя подвинуть чуть-чуть
 - d. Потому что компьютеры хранят их как текст, а не как числа
3. Ошибка в двенадцать единиц упала бы на полединицы, если одну ручку повернуть вверх на одну десятую. О чём это говорит?
- a. Скорость — пять единиц ошибки на единицу ручки, так что поворот вверх помогает
 - b. Скорость — полединицы, так что ручка не имеет значения
 - c. Ошибка дойдёт до нуля через двадцать четыре сдвига
 - d. Ручку нужно повернуть вниз, чтобы снизить ошибку
4. Какому виду данных, по книге, подходит свёртка?
- a. Данным без всякого порядка, например множеству
 - b. Данным, устроенным как сеть связанных точек
 - c. Только фотографиям

d. Данным, которые приходят упорядоченным рядом элементов, таким как звук, изображения, текст и видео

Ответы

1. b) Обучение начинается с программы с очень многими параметрами и подстраивает их, пока качество не станет достаточным; при миллионах ручек перебор наугад безнадежен, поэтому и нужны градиенты.
2. c) Дифференцируемому блоку нужно знать, как меняется его выход при малом изменении входа, а у скачущей величины малых изменений нет.
3. a) Полединцы изменения на сдвиг в одну десятую — это скорость пять, а поскольку ошибка падает, ручку нужно повернуть вверх; градиент даёт такую скорость сразу для каждой ручки.
4. d) Свёртки применимы всякий раз, когда данные можно записать как упорядоченную последовательность элементов; множества и графы оставлены слоям внимания и графовым моделям в третьей части.

Разобранные примеры

Читаем скорость изменения Ошибка программы равна двенадцати. Сдвиг одной ручки вверх на одну десятую снизил бы ошибку на полединцы. Какова скорость изменения ошибки по этой ручке, и в какую сторону её поворачивать?

Изменение ручки — одна десятая, а изменение ошибки — падение на полединцы.

Скорость — это изменение ошибки, делённое на изменение ручки: половина, делённая на одну десятую, равна пяти.

Поскольку ошибка падает, когда ручка идёт вверх, скорость говорит, что ошибка снижается на пять единиц на единицу ручки в этом направлении.

Значит, ручку нужно повернуть вверх, и маленький поворот на одну десятую стоит около полединицы ошибки.

Ответ: Скорость пять единиц ошибки на единицу ручки; повернуть ручку вверх.

Задачи

1. Опишите в терминах входа и выхода, как это делает книга, задачу «превратить фотографию в подпись» и задачу «ответить на устный вопрос».

Подсказка: Для каждой скажите, какой вид данных входит и какой выходит.

Ответ: Подпись к картинке берёт изображение, упорядоченную сетку пикселей, и возвращает текст, ряд символов. Ответ на устный вопрос берёт звук, упорядоченный ряд чисел, и возвращает текст. Книга замечает, что модели для таких задач устроены очень похоже.

2. Объясните своими словами, почему чат-ассистента можно построить обучением с учителем.

Подсказка: Какие примеры нужного поведения можно собрать для языка?

Ответ: Подражание языку можно превратить в предсказание каждого предложения по предыдущим, а огромные объёмы текста дают бесконечные примеры именно этого. Обучение системы хорошо делать такие предсказания — это обучение с учителем на тексте.

3. У модели миллиард параметров. Почему безнадёжно улучшать её случайным перебором настроек, и что вместо этого даёт

градиент?

Подсказка: Подумайте, сколько комбинаций у миллиарда ручек и что говорит один градиент.

Ответ: Число возможных настроек астрономически велико, так что случайные попытки почти никогда не попадут на хорошую.

Градиент говорит сразу для каждого параметра, в какую сторону и насколько сильно малый сдвиг меняет качество работы, поэтому ручки можно автоматически крутить в полезном направлении.

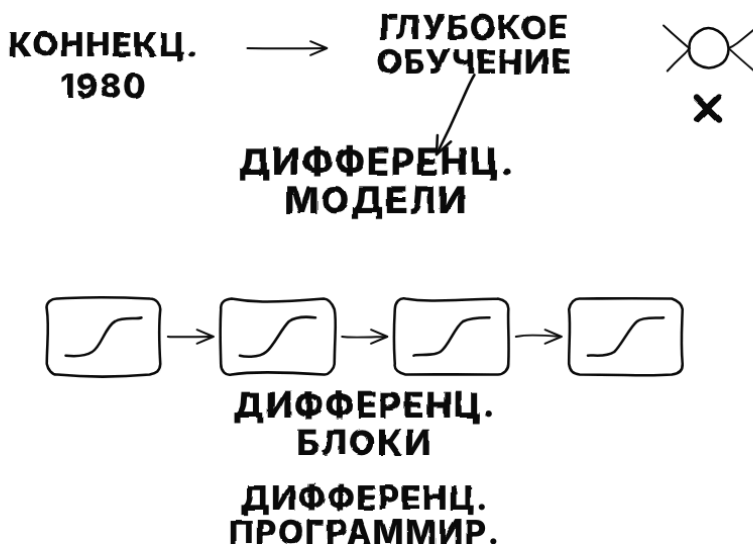
Старые идеи в ожидании данных

Представьте газетную страницу 1958 года. Психолог Фрэнк Розенблатт попал в заголовки благодаря машине под названием перцептрон, одному из первых работающих прототипов нейронной сети, и репортёры уже обещают машины, которые будут ходить, говорить и думать. Книга показывает снимок того момента с ироничной подписью: шумиха вокруг искусственного интеллекта, только на дворе 1958 год. Если это звучит знакомо, то потому, что такие же заголовки сегодня встречают каждого нового чат-ассистента. Эта сцена напоминает, что идеи, стоящие за сегодняшними сетями, гораздо старше шумихи вокруг них.

Само название «нейронная сеть» — наследие тех ранних попыток двадцатого века подражать биологическим нейронам, нервным клеткам мозга. Словарь прижился: чтобы не расходиться с программами, которыми люди реально пользуются, книга тоже будет говорить о нейронах, слоях и активациях. Интерес к таким сетям приходил волнами. Между 2012 и 2017 годами сети усложнялись как никогда прежде благодаря крупным открытым соревнованиям, прежде всего конкурсу ImageNet, где программы состязались в распознавании предметов на более чем миллионе фотографий; книга возвращается к нему в девятой главе. Вторая большая волна началась в 2017 году с трансформера, темы десятой главы. Как свёрточные модели несколькими годами раньше захватили компьютерное зрение, так трансформеры за очень короткое время захватили обработку человеческого языка. Затем последовал прогресс для видео, графов и звука, приведший к сегодняшнему увлечению большими языковыми

моделями, моделями, смешивающими несколько видов данных, и моделями, порождающими новые картинки, текст и звук.

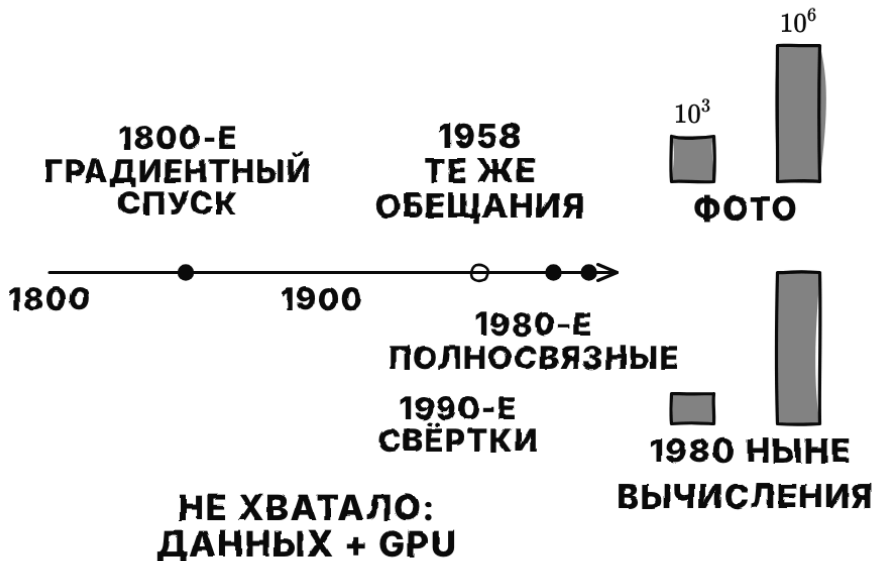
Вместе с волнами менялись и слова. В 1980-е говорили о коннекционизме; позже «глубокое обучение» стало означать современные сети в противоположность меньшим и более мелким моделям прошлого. Но все эти термины остаются расплывчатыми, потому что современные искусственные сети почти совсем не похожи на биологические нейроны. По-настоящему их определяет, утверждает книга, то, что они построены из дифференцируемых блоков — деталей, чей выход меняется плавно, когда вход слегка сдвигают. Поэтому книга, где только может, предпочитает название «дифференцируемые модели». С этой точки зрения нейронные сети принадлежат более широкой области, называемой дифференцируемым программированием, — молодой дисциплине, которая смешивает информатику и оптимизацию, чтобы изучать компьютерные программы, которые можно дифференцировать. Источниками этого взгляда книга называет короткий манифест Яна Лекуна, опубликованный в 2018 году, и второй, Колина Раффела, 2021 года, о построении моделей так, как строится программное обеспечение с открытым кодом.



Имена меняются, блоки остаются

Путешествие по стране дифференцируемых моделей — это и путешествие по истории, и даты стоит услышать. Подгонка простой модели прямой линии градиентным спуском, тема четвёртой главы, была известна по крайней мере с девятнадцатого века. Полносвязные сети того рода, что книга использует позже, восходят к 1980-м. Свёрточные модели уже применялись в конце 1990-х. Так что главные детали существовали десятки лет назад, а не хватало другого: достаточного количества данных и вычислительной мощности, чтобы показать, как хорошо эти модели работают, когда у них вдоволь и того, и другого, и вдоволь параметров. Вот как почувствовать разрыв: телефон у вас в кармане сегодня делает за секунду больше арифметики, чем компьютеры 1980-х за много

часов, а коллекции фотографий, на которых обучают его приложения, содержат миллионы снимков там, где у ранних исследователей были несколько тысяч.



Старые детали, новое топливо

Опыт

Поищите в интернете газетные сообщения 1958 года о перцептроне Розенблатта, а затем поищите заголовки о недавнем чат-ассистенте. Положите их рядом и сравните обещания: машины, которые учатся, которые понимают, которые скоро будут думать. Вы увидите, как точно повторяется язык восторга, тогда как внизу по-настоящему изменились в основном объём данных и вычислительная мощность.

Книга завершает эту короткую историю скромным обещанием. Исследования движутся так быстро, что ни одна книга не может глубоко охватить все темы, но надежда в том, чтобы дать читателю достаточно основы, чтобы он сам находил дорогу в новейших научных статьях.

Запомним

Нейронные сети стары: название идёт от попыток двадцатого века подражать клеткам мозга, а в 1958 году перцептрон уже вызвал волну шумихи. Интерес приходил волнами: скачок размера сетей между 2012 и 2017 годами, подталкиваемый конкурсами вроде ImageNet, и второй скачок с трансформерами с 2017 года, приведший к сегодняшним языковым и генеративным моделям. Современные сети едва напоминают мозг; их настоящая определяющая черта — то, что они построены из дифференцируемых блоков, поэтому книга называет их дифференцируемыми моделями, частью дифференцируемого программирования. Градиентный спуск восходит к девятнадцатому веку, полносвязные сети — к 1980-м, свёрточные модели — к 1990-м; десятилетиями не хватало лишь данных и вычислительной мощности.

Вопросы

1. Почему книга предпочитает название «дифференцируемые модели» названию «нейронные сети»?
 - а. Потому что современные сети почти не похожи на клетки мозга, а их настоящая определяющая черта — то, что они построены из дифференцируемых блоков
 - б. Потому что слово «нейронный» защищено торговой маркой

- c. Потому что дифференцируемые модели — более новое изобретение, чем нейронные сети
 - d. Потому что нейронные сети нельзя применять к языку
- 2. Полносвязные сети появились в 1980-х, а свёрточные модели — в 1990-х. Почему их успех пришёл так много позже?
 - a. Потому что идеи пришлось изобретать заново с нуля
 - b. Потому что достаточно данных и вычислительной мощности, чтобы показать их силу, появилось лишь гораздо позже
 - c. Потому что программисты отказывались ими пользоваться до 2017 года
 - d. Потому что они работают только с фотографиями
- 3. Что произошло с обработкой языка после появления трансформеров в 2017 году?
 - a. Она постепенно отказалась от нейронных сетей
 - b. Она вернулась к правилам, написанным вручную
 - c. Трансформеры захватили её за очень короткое время, как свёрточные модели — зрение
 - d. Она ограничилась одним лишь переводом
- 4. Что снимок перцептрона 1958 года в книге говорит о сегодняшнем восторге вокруг чат-ассистентов?
 - a. Что перцептрон втайне был чат-ассистентом
 - b. Что с 1958 года ничего не улучшилось
 - c. Что шумиха — новое явление эпохи интернета

d. Что волны шумихи вокруг нейронных сетей не новы; идеи старше восторга

Ответы

1. a) Старые названия остаются расплывчатыми, поскольку искусственные сети едва напоминают биологические нейроны; общее у всех современных сетей то, что они построены из дифференцируемых блоков.
2. b) Главные детали существовали десятки лет назад; не хватало данных, вычислительной мощности и параметров, чтобы раскрыть, как хорошо они работают.
3. c) Книга проводит параллель: свёрточные модели захватили компьютерное зрение, а несколько лет спустя трансформеры очень быстро захватили обработку естественного языка.
4. d) Подпись называет это шумихой вокруг ИИ, только в 1958 году, — напоминание, что такие волны уже бывали и что идеи старше сегодняшнего внимания.

Задачи

1. Расставьте в порядке появления, как говорит книга: трансформеры, градиентный спуск для линейных моделей, свёрточные модели, полносвязные сети.

Подсказка: Одно восходит к девятнадцатому веку; одно — из 2017 года.

Ответ: Градиентный спуск для линейных моделей (девятнадцатый век), полносвязные сети (1980-е), свёрточные модели (конец 1990-х), трансформеры (2017).

2. Своими словами: что изменилось между 1990-ми и 2010-ми, из-за чего старые конструкции сетей вдруг стали успешными?

Подсказка: Книга называет два ингредиента плюс размер моделей.

Ответ: Объём доступных данных и вычислительная мощность выросли колоссально, что позволило обучать модели с множеством параметров и показать, как хорошо старые конструкции работают, когда им всего хватает.

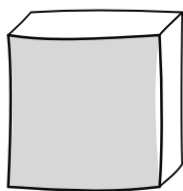
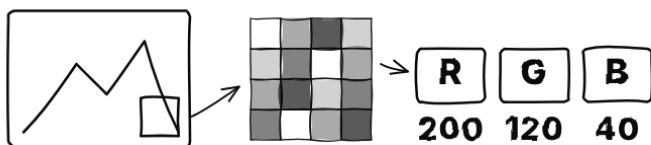
3. Объясните, что такое дифференцируемое программирование, используя идею из предыдущей части о том, что градиент говорит, как величина откликается на малые сдвиги.

Подсказка: Что должен допускать каждый блок такой программы?

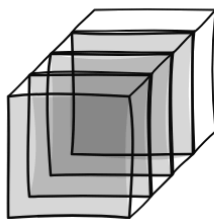
Ответ: Это изучение компьютерных программ, построенных так, что каждый шаг можно дифференцировать, то есть можно вычислить, как выход программы откликается на малые сдвиги её входов и параметров. Нейронные сети — один пример, поэтому книга рассматривает их как дифференцируемые модели.

Коробки чисел и их формы

Возьмите одну фотографию со своего телефона и спросите, что она на самом деле такое для компьютера. Увеличьте достаточно сильно, и она распадётся на крошечные квадратики, у каждого три числа: сколько красного, сколько зелёного и сколько синего. Фотография в 224 квадрата на 224 квадрата — это, значит, блок чисел высотой 224, шириной 224 и глубиной 3. Когда фотоприложение учится на ваших снимках, оно берёт сразу стопку таких блоков, скажем тридцать два, и эта стопка — коробка чисел с четырьмя направлениями: какая фотография, какая строка, какой столбец, какой цвет. Книга называет каждую такую коробку тензором, и это основная вещь, с которой работают дифференцируемые модели.



$224 \times 224 \times 3$



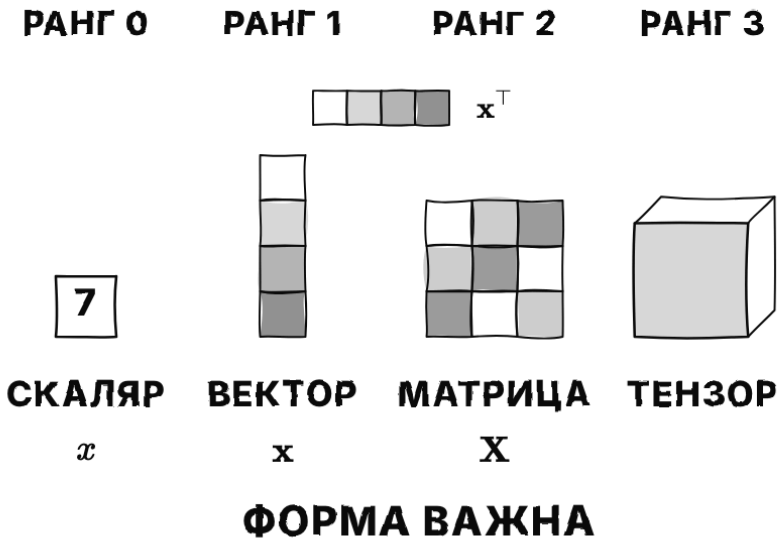
$\times 32$

ФОТО·РЯД·СТОЛБЕЦ·ЦВЕТ
ТЕНЗОР

Фотография — коробка чисел

Тензор в этой книге — просто массив чисел, разложенных в некотором количестве направлений, и это количество направлений называется рангом тензора. Книга признаёт, что математики называют тензором нечто более тонкое, род оператора, и извиняется за заимствование, но заимствованное значение настолько распространено, что она его сохраняет. Одно число, ранг ноль, называется скаляром, и книга пишет его обычной маленькой буквой. Столбец чисел, ранг один, — это вектор, записанный маленькой жирной буквой; когда те же числа выложены в строку, книга отмечает это маленькой приподнятой Т, называемой транспонированием, хотя опускает знак, когда смысл ясен. Рисунок в книге изображает вектор строкой только для удобства; в тексте каждый вектор — столбец.

Прямоугольник чисел, ранг два, — это матрица, записанная заглавной жирной буквой. Для рангов выше двух особых букв нет, а завитые, каллиграфические заглавные оставлены для множеств и распределений вероятностей.



Ранг: ноль, один, два, три

Книга делает одно замечание, которое будет важно повсюду: понять операцию обычно значит понять форму каждого участвующего тензора. Поэтому она пользуется коротким способом записи формы: имя тензора, волнистый знак и список его размеров в скобках. Стопка фотографий из примера выше записалась бы как тензор, чья форма — размер батча, высота, ширина и три. Некоторые размеры — фиксированные числа, как три цвета, а другие оставлены буквами, как размер батча, потому

что они могут меняться от одного применения к другому. Посчитаем, что означает такая форма: тридцать две фотографии, каждая 224 на 224 квадратика с 3 цветами, — это 224 умножить на 224, то есть 50 176 квадратиков, умножить на 3, то есть 150 528 чисел на фотографию, умножить на 32, то есть 4 816 896 чисел во всей стопке. Это почти пять миллионов чисел для одного маленького батча, вот почему формы заслуживают внимания.

$$X \sim (b, 224, 224, 3)$$

МЕНЯЕТСЯ ФИКСИРОВАНО

$$224 \times 224 = 50,176$$

$$\times 3 = 150,528$$

$$\times 32 = 4,816,896$$

≈ 5 МИЛЛИОНОВ

$$x \sim \mathcal{N}(0, 1)$$



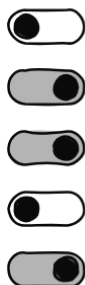
Пишем и считаем форму

Тот же волнистый знак изредка используется и для того, чтобы сказать, что число взято случайно из распределения вероятностей, например число, взятое из знакомой колоколообразной кривой с центром в нуле. Контекст всегда подсказывает, какое значение имеется в виду. Книга пользуется

записью формы вместо более привычного математического способа говорить, что вектор — это список из определённого числа вещественных чисел; так короче, и глаз остаётся на форме.

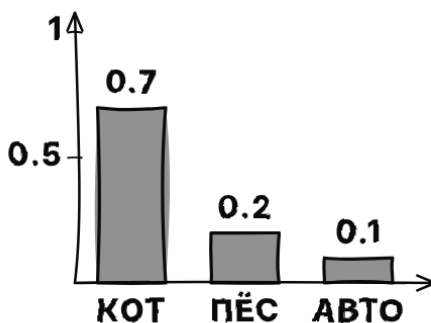
Ещё две пометки говорят, какие значения может содержать тензор. Одна отмечает тензор, элементы которого только нули и единицы, называемый бинарным, как ряд переключателей, каждый из которых либо выключен, либо включён. Другая отмечает вектор, лежащий в симплексе: его элементы никогда не отрицательны и в сумме дают ровно единицу. Именно это выдаёт чат-ассистент, когда угадывает следующее слово: шансы вроде семи десятых для «кошки», двух десятых для «собаки» и одной десятой для «машины», которые в сумме дают единицу. Для тензора более высокого ранга, например таблицы с одной строкой на пример, книга применяет правило вдоль последнего направления, так что каждая строка сама по себе даёт в сумме единицу.

БИНАРНЫЙ



ВЫКЛ / ВКЛ

СИМПЛЕКС



$$0.7 + 0.2 + 0.1 = 1$$

☐	☐	☐	= 1
☐	☐	☐	= 1

Бинарный и симплекс

Наконец, книга ставит на полях три маленькие картинки, чтобы направлять читателя.

Запомним

Тензор здесь означает массив чисел в некотором числе направлений, ранге: у скаляра ранг ноль, у вектора ранг один, у матрицы ранг два. Скаляры — маленькие буквы, векторы — маленькие жирные буквы (столбцы, с приподнятой T для формы строки), матрицы — заглавные жирные буквы; заглавные оставлены для множеств и распределений. Понять операцию в основном значит знать форму каждого тензора, записанную волнистым знаком и списком размеров, одних фиксированных, других оставленных буквами; батч из тридцати двух цветных фотографий 224×224 содержит почти пять миллионов чисел. Дополнительные пометки отмечают тензоры только из нулей и единиц и векторы в симплексе, чьи элементы не отрицательны и в сумме дают единицу.

Вопросы

1. Батч фотографий хранится с четырьмя направлениями: какая фотография, какая строка, какой столбец, какой цвет. Каков его ранг?
 - a. Три, потому что цвет не считается
 - b. Четыре, по одному на каждое направление
 - c. Один, потому что это один батч
 - d. Два, потому что фотография плоская

2. Какой из этих списков чисел лежит в симплексе?
- a. Семь десятых, две десятых и одна десятая
 - b. Половина, половина и половина
 - c. Один, ноль и минус один... нет, минус один не допускается: ноль, один и один
 - d. Шесть десятых, три десятых и две десятых
3. Почему книга оставляет некоторые размеры в форме буквами, например размер батча, а другие пишет числами?
- a. Потому что буквы занимают меньше памяти
 - b. Потому что размеры, записанные буквами, всегда больше
 - c. Потому что эти размеры могут меняться от одного применения к другому, а другие, как три цвета, фиксированы
 - d. Потому что буквы отмечают размеры, которые ещё никому не известны
4. В таблице с одной строкой на пример и пометкой «симплекс» что должно давать в сумме единицу?
- a. Все числа всей таблицы вместе
 - b. Каждый столбец сам по себе
 - c. Только первая строка
 - d. Каждая строка сама по себе, вдоль последнего направления

Ответы

1. b) Ранг — это число направлений массива; фотография, строка, столбец и цвет дают четыре.

2. а) Вектор лежит в симплексе, когда ни один элемент не отрицателен и элементы в сумме дают ровно единицу; семь десятых, две десятых и одна десятая дают в сумме единицу, как шансы следующего слова у чат-ассистента.

3. с) Фиксированные размеры, такие как три цвета, пишутся числами; размеры, меняющиеся от применения к применению, например сколько фотографий в батче, оставляют буквами.

4. d) Для тензоров более высокого ранга книга применяет правило вдоль последнего направления, так что каждая строка отдельно должна давать в сумме единицу.

Разобранные примеры

Считаем числа в форме Батч содержит 8 цветных фотографий, каждая 64 квадрата в высоту и 64 в ширину, по 3 значения цвета на квадратик. Сколько чисел содержит тензор и каков его ранг?

Направления — фотография, строка, столбец и цвет, значит, ранг равен четырём.

В одной фотографии 64 умножить на 64 квадрата, то есть 4096 квадратов.

У каждого квадрата 3 значения цвета, так что одна фотография содержит 4096 умножить на 3, то есть 12 288 чисел.

Восемь фотографий содержат 12 288 умножить на 8, то есть 98 304 числа.

Ответ: 98 304 числа в тензоре ранга четыре.

Задачи

1. Назовите ранг каждого: температура на улице прямо сейчас; дневные температуры одной недели; таблица дневных

температур для каждой недели года.

Подсказка: Посчитайте, сколько направлений нужно, чтобы найти одно число.

Ответ: Одно число — ранг ноль, скаляр. Семь дневных значений в столбце — ранг один, вектор. Таблица с одной строкой на неделю и одним столбцом на день — ранг два, матрица.

2. Модель выдаёт шансы пять десятых, три десятых и три десятых для трёх возможных слов. Лежит ли этот вектор в симплексе? Объясните.

Подсказка: Проверьте два условия: нет отрицательных элементов, и сумма.

Ответ: Нет. Ни один элемент не отрицателен, но в сумме они дают одиннадцать десятых, а не единицу, так что вектор не в симплексе.

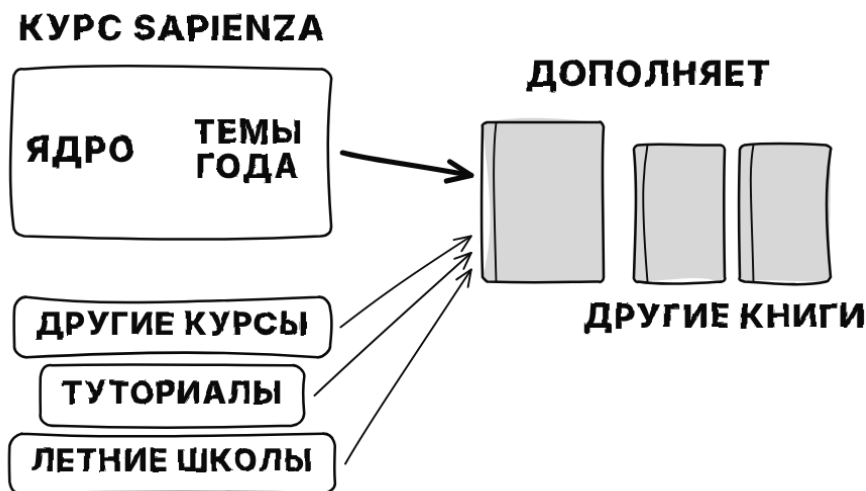
3. Одна цветная фотография 100 на 200 квадратиков хранится как тензор. Сколько чисел она содержит?

Подсказка: Перемножьте размеры всех направлений, включая цвета.

Ответ: 100 умножить на 200 — это 20 000 квадратиков, а с 3 значениями цвета у каждого это 60 000 чисел в тензоре ранга три.

Чем эта книга является и чем не является

Перед долгой прогулкой хороший проводник говорит вам три вещи: откуда пришла тропа, что вы увидите и чего не увидите. Книга делает именно это, прежде чем начать. Она выросла из курса, который автор много лет читает в магистратуре по науке о данных в Римском университете Sapienza, под названием «Нейронные сети для приложений науки о данных». Основные главы — сердце этого курса; остальные главы — темы, которые из года в год появляются и исчезают. Куски добавились и из другого преподавания: курса о нейронных сетях для инженеров-компьютерщиков, введения в машинное обучение для инженеров связи, tutorиалов, аспирантских курсов и летних школ.

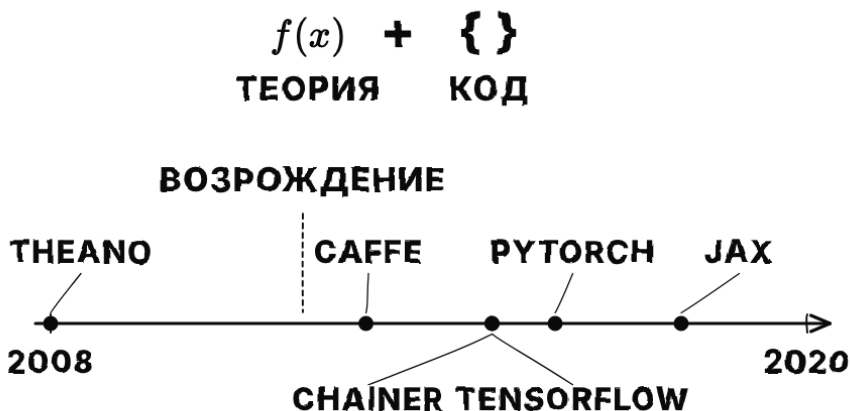


Откуда взялась книга

Уже есть несколько отличных недавних книг о современных глубоких сетях, и книга говорит об этом открыто. В начале она покрывает во многом ту же почву, что и они; позже способ объяснения и некоторые дополнительные разделы расходятся с ними, следуя собственным исследовательским интересам автора. Цель — предложить дополнительную, взаимодополняющую точку зрения, а не заменить остальные.

Выбор книгой названия «дифференцируемые программы» несёт сообщение: чтобы понять их, нужны и теория, и код. То, как модель спроектирована, и то, как она запрограммирована, постоянно формируют друг друга, и автоматическое дифференцирование, встреченное ранее автоматическое

вычисление скоростей изменения, — лучший пример. В самом деле, возрождение нейронных сетей примерно с 2012 года во многом обязано программному обеспечению: библиотекам вроде Theano, Caffe и Chainer, а затем сегодняшним TensorFlow, PyTorch и JAX, которые сделали построение и обучение сетей доступным каждому. Фотоприложение на вашем телефоне и чат-ассистент, которым вы пользуетесь, почти наверняка построены инструментами из этого семейства. Где только может, книга связывает свои идеи с этими инструментами, особенно PyTorch и JAX, но она не руководство по программированию, и для изучения самих библиотек она отсылает к их собственной документации.



Софт вернул сети

Затем идёт честный список того, чем книга не является. Она не пытается быть полной. Она выбирает понятия, распространённые сегодня и достаточно общие, чтобы оставаться полезными какое-то время, но будущее никто не предвидит, и некоторые части могут оказаться неполными или устаревшими к тому времени, когда вы их читаете. Для каждой идеи она показывает несколько вариантов из научных статей, например два разных способа нормализовать числа внутри сети, но существуют сотни других, и за остальным книга приглашает заглянуть в большой онлайн-каталог статей и их кода.

Наконец, книга — о базовых компонентах дифференцируемых моделей, а не о запуске их в большом масштабе. Чтобы большая модель действительно заработала, нужны инженерное умение и некоторая интуиция, и об оборудовании книга говорит мало. Для интуиции, говорит она, нет ничего лучше опыта плюс резких по тону блогов практиков; она рекомендует известный рецепт обучения сетей Андрея Карпатого и его серию видео, где сети строятся с нуля. Думайте об этом как об учёбе готовить: книга учит, что делает каждый ингредиент, а сноровка готовить на сотню гостей приходит с практикой на оживлённой кухне.

Запомним

Книга выросла из давнего магистерского курса в Римском университете Sapienza, расширенного материалом других курсов, tutorиалов и летних школ. Сначала она пересекается с другими хорошими книгами, затем идёт своим путём, стремясь быть взаимодополняющей точкой зрения. Понять дифференцируемые программы значит соединить теорию и код; возвращение нейронных сетей с 2012 года во многом обязано библиотекам вроде TensorFlow, PyTorch и JAX, на которые книга ссылается, не будучи руководством по программированию. Она не полна, может устареть, показывает лишь несколько из множества вариантов каждой идеи и оставляет крупномасштабную инженерию, оборудование и практическую интуицию опыту и текстам практиков.

Вопросы

1. Почему книга говорит, что для понимания дифференцируемых программ нужны и теория, и код?
 - а. Потому что проектирование модели и её программирование постоянно формируют друг друга, как показывает автоматическое дифференцирование
 - б. Потому что каждая глава — упражнение по программированию

- c. Потому что одной теории слишком мало, чтобы заполнить книгу
 - d. Потому что библиотеками нельзя пользоваться, не прочитав книгу
2. Чему эта часть в значительной мере приписывает возрождение нейронных сетей примерно с 2012 года?
- a. Новой теории мозга
 - b. Окончанию конкурса ImageNet
 - c. Мощным программным библиотекам, таким как TensorFlow, PyTorch и JAX
 - d. Запрету правил, написанных вручную
3. Что из этого книга, по её словам, не пытается делать?
- a. Связывать свои идеи с PyTorch и JAX
 - b. Объяснять базовые компоненты дифференцируемых моделей
 - c. Предлагать точку зрения, дополняющую другие книги
 - d. Охватить каждый вариант каждой идеи и инженерии очень больших моделей
4. Куда книга отправляет читателя за практической интуицией об обучении сетей?
- a. Только к своим лабораторным блокнотам
 - b. К опыту и к резким по тону блогам практиков, таким как рецепт Карпатого
 - c. К документации TensorFlow
 - d. К истории перцептрона

Ответы

1. **a)** То, как модели проектируют и как их реализуют, всё время влияет друг на друга, и автоматическое дифференцирование — самый ясный пример.
2. **c)** Возрождение во многом обязано программным библиотекам, от Theano, Caffe и Chainer до сегодняшних TensorFlow, PyTorch и JAX.
3. **d)** Она показывает лишь несколько вариантов каждой идеи, признаёт, что может быть неполной или устаревшей, и оставляет крупномасштабную инженерию, оборудование и интуицию опыту.
4. **b)** Для интуиции нет ничего лучше опыта плюс блогов практиков; книга рекомендует рецепт обучения Андрея Карпатого и его серию видео.

Задачи

1. Объясните своими словами разницу между знанием того, что делают компоненты модели, и умением обучить очень большую работающую модель.

Подсказка: Вспомните сравнение с готовкой в этой части.

Ответ: Знать компоненты — как знать, что делает каждый ингредиент; этому книга и учит. Обучить большую работающую модель — как готовить на сотню гостей: нужны инженерное умение, знание оборудования и интуиция, которые приходят главным образом с практикой.

2. Почему части книги могут устареть к тому времени, когда вы её читаете, и что она делает, чтобы это ограничить?

Подсказка: Посмотрите, как она выбирает понятия.

Ответ: Исследования движутся быстро, и никто не может предвидеть, какие методы сохранятся. Книга снижает риск, выбирая

понятия, которые и распространены сегодня, и достаточно общие, чтобы какое-то время оставаться полезными.

3. Книга не руководство по программированию, и всё же она часто упоминает PyTorch и JAX. Почему оба утверждения совместимы?

Подсказка: Какую роль библиотеки играют в объяснениях?

Ответ: Книга использует библиотеки, чтобы связать идеи с тем, как они реально реализованы, ведь теория и код формируют друг друга. Она не учит библиотекам с нуля; для этого она отсылает к их собственной документации.

Благодарности, лицензия и нарезка срезов

В каждой книге есть страница, которую мало кто читает, и в этой на ней есть кое-что, что стоит услышать. Цветная подсветка её уравнений сделана программным пакетом, написанным S. T. John. Цветные картинки с Алисой и маленькие чёрно-белые символы на полях лицензированы у стокового фотобанка Shutterstock, а картинки с Алисой внутри рисунков воспроизводят оригинальные иллюстрации Джона Тенниела, полученные через Wikimedia. Автор благодарит Роберто Альму за отзыв на ранний черновик и за поддержку публикации, а Коррадо Дзокколо, Эмануэле Родолу, Марцина Слабого, Константина Бурлаченко и Диего Сандовалья — за множество исправлений, как и всех, кто присылал отзывы по электронной почте.

CC BY-SA



АВТОРЫ

**ДЕЛИТЬСЯ ✓
АДАПТИРОВАТЬ ✓
КОММЕРЧЕСКИ ✓
УКАЗАТЬ АВТОРА
ТА ЖЕ ЛИЦЕНЗИЯ**

ЭТИ ЛЕКЦИИ

CC BY-SA

Лицензия, разрешающая адаптацию

Затем идёт лицензия, и она касается вас напрямую. Книга выпущена под лицензией Creative Commons под названием Attribution-ShareAlike. Её собственными словами, она позволяет любому «распространять, перерабатывать, адаптировать материал и строить на его основе на любом носителе и в любом формате» при условии указания автора; коммерческое использование разрешено; а всё, что построено на этой основе, должно выпускаться на тех же условиях. Лекции, которые вы слушаете, — именно такая адаптация, поэтому они тоже несут эту лицензию.

На этом книга открывает вторую главу — о математических основах. Это сжатый обзор, а не первый курс: он предполагает,

что вы уже встречали эти темы, и в основном закрепляет обозначения и показывает, как детали стыкуются, с оглядкой на то, как они реализуются на практике. В главе три ступени, каждая строится на предыдущей: линейная алгебра, затем градиенты для объектов со многими измерениями, затем как оптимизировать функцию с помощью этих градиентов. Теория вероятностей, с упором на принцип, называемый максимум правдоподобия, разбирается отдельно в приложении. Книга бодро предупреждает, что глава полна определений, и просит немного терпения.

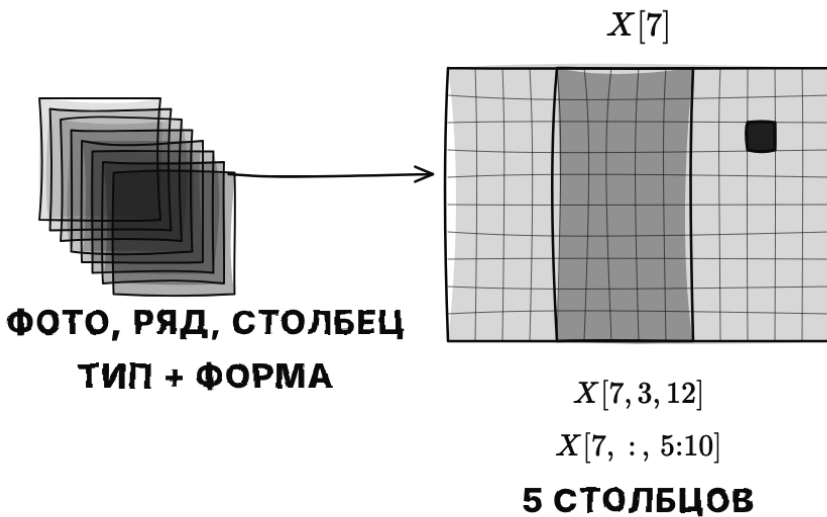
МНОГО ОПРЕДЕЛЕНИЙ



Вторая глава в три ступени

Первая ступень, линейная алгебра, снова начинается с тензора, коробки чисел из предыдущей части, теперь определённого

формально: массив в некотором числе направлений, все элементы которого одного типа. Его форма записывается волнистым знаком и списком размеров, по одному размеру на направление. Тензоры центральны в глубоком обучении по практической причине: одна и та же простая операция, применённая сразу к миллионам чисел, — именно то, что хорошо делают графические чипы, поэтому фотоприложение вашего телефона и серверы за чат-ассистентом работают на таких чипах, называемых GPU, или на ещё более специализированных. Тензор описывается двумя вещами: типом его элементов и его формой. Чаще всего элементы — числа с десятичной точкой, хранимые в формате, который книга обсуждает позже, но тензоры могут содержать и целые числа, например метки классов, или строки текста.



Нарезаем срезы из тензора

Часто нужна лишь часть тензора, срез, и книга следует соглашениям популярной библиотеки NumPy с одним отличием: для удобства чтения она считает позиции с единицы, а не с нуля. Представьте тензор с батчем чёрно-белых картинок: какая картинка, какая строка, какой столбец. Если зафиксировать только первую позицию, скажем картинку номер семь, получится срез — одна целая картинка, прямоугольник строк и столбцов. Если зафиксировать все три позиции — картинка семь, строка три, столбец двенадцать, — получится одно число, яркость одной точки. Для более сложных срезов книга разделяет позиции запятыми и использует двоеточие, означающее всё вдоль направления, и диапазон от одной позиции до другой, не включая её. Возьмём один: картинка семь, все строки и столбцы с

пятого до десятого, не включая десятый. Это оставляет каждую строку и десять минус пять, то есть пять, столбцов, так что срез — полоса из всех строк на пять столбцов. Когда индексация загромождила бы уравнение, книга рисует вокруг выражения светло-серые квадратные скобки с позициями, написанными мелко справа внизу, чтобы глаз отделял то, что индексируется, от того, как.

Запомним

Указаны авторы раскраски уравнений, стоковых картинок и иллюстраций Тенниела и многие читатели, приславшие исправления, а книга выпущена под лицензией Creative Commons Attribution-ShareAlike, разрешающей адаптацию и коммерческое использование при указании автора и той же лицензии, как это делают и эти лекции. Вторая глава — компактный обзор линейной алгебры, градиентов и оптимизации, с вероятностью в приложении. Тензор — массив в некотором числе направлений с элементами одного типа; он описывается этим типом и своей формой и подходит массово-параллельным чипам, используемым для глубокого обучения. Срезы следуют соглашениям NumPy, но считают с единицы: фиксация некоторых позиций даёт меньший тензор, фиксация всех — одно число, а диапазон вроде «с пяти до десяти» оставляет пять позиций.

Вопросы

1. Что из этого разрешено лицензией книги?
 - a. Продавать адаптированную версию, если автор указан и адаптация сохраняет ту же лицензию
 - b. Продавать адаптированную версию под собственной более строгой лицензией
 - c. Адаптировать книгу, не указывая автора
 - d. Ничего, кроме чтения для себя
2. Тензор хранит чёрно-белые картинки по картинке, строке и столбцу. Что вы получите, зафиксировав только номер картинки?
 - a. Одно значение яркости
 - b. Одну целую картинку, прямоугольник строк и столбцов
 - c. Один столбец по всем картинкам
 - d. Тензор с теми же тремя направлениями
3. Почему тензоры так центральны в глубоком обучении, согласно этой части?
 - a. Потому что это единственный способ хранить текст
 - b. Потому что они делают сети похожими на мозг
 - c. Потому что одна операция сразу над миллионами чисел подходит GPU и похожим чипам
 - d. Потому что в них никогда не бывает десятичных чисел
4. Срез берёт картинку семь, все строки и столбцы с пятого до десятого, не включая десятый. Сколько столбцов он сохраняет?
 - a. Шесть
 - b. Десять

c. Четыре

d. Пять

Ответы

1. a) Attribution-ShareAlike разрешает распространение, переработку и коммерческое использование при условии, что автор указан и адаптированный материал несёт ту же лицензию.
2. b) Фиксация одной позиции убирает это направление и оставляет срез из двух других: одну полную картинку.
3. c) Тензоры подходят массово-параллельному оборудованию, такому как графические чипы, которые применяют одну операцию к очень многим числам одновременно.
4. d) Диапазон до конца, не включая его, сохраняет десять минус пять, то есть пять, позиций.

Разобранные примеры

Размер среза Тензор хранит 50 чёрно-белых картинок, каждая 28 строк на 28 столбцов. Вы берёте картинку три, строки с 1 до 15, не включая 15, и все столбцы. Сколько чисел в срезе и какова его форма?

Фиксация картинки три убирает первое направление, так что у среза остаются строки и столбцы.

Строки с 1 до 15, не включая 15, сохраняют 15 минус 1, то есть 14 строк.

Все столбцы — это все 28 столбцов.

Срез — 14 строк на 28 столбцов, то есть 14 умножить на 28, это 392 числа.

Ответ: Прямоугольник 14 строк на 28 столбцов, 392 числа.

Задачи

1. В том же тензоре из 50 картинок 28 на 28 сколько чисел вы получите, зафиксировав картинку, строку и столбец?

Подсказка: Что остаётся, когда зафиксировано каждое направление?

Ответ: Одно число: яркость одной точки. Фиксация каждой позиции не оставляет направлений, так что результат — скаляр.

2. Вы берёте все 50 картинок, только строку 10 и столбцы с 3 до 13, не включая 13. Какова форма среза и сколько чисел он содержит?

Подсказка: Фиксация строки убирает это направление; столбцы считайте как конец минус начало.

Ответ: Направление строк исчезает, остаются 50 картинок на 13 минус 3, то есть 10 столбцов. Срез 50 на 10, он содержит 500 чисел.

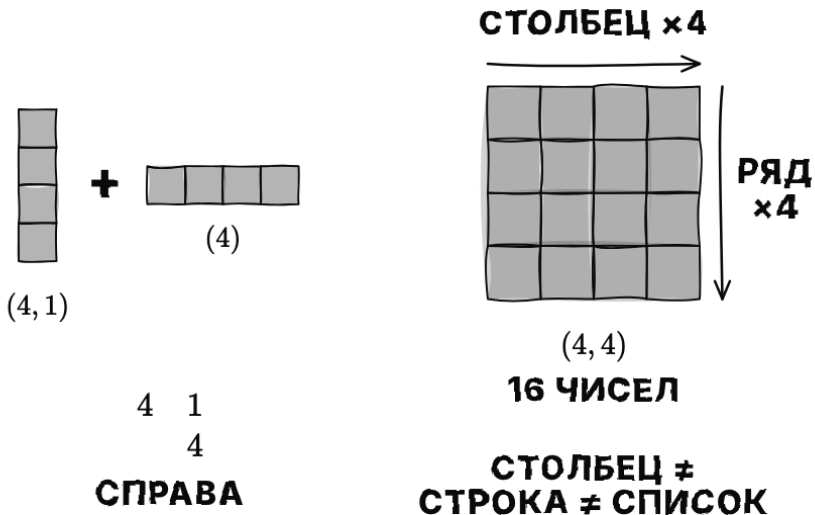
3. Кто-то адаптирует эту книгу в платный видеокурс. Что, согласно лицензии, он обязан сделать, две вещи?

Подсказка: Посмотрите на два условия, приложенные к разрешению.

Ответ: Он должен указать создателя книги и выпустить свой адаптированный материал на тех же условиях Attribution-ShareAlike. Само коммерческое использование разрешено.

Стрелки, длины и скалярное произведение

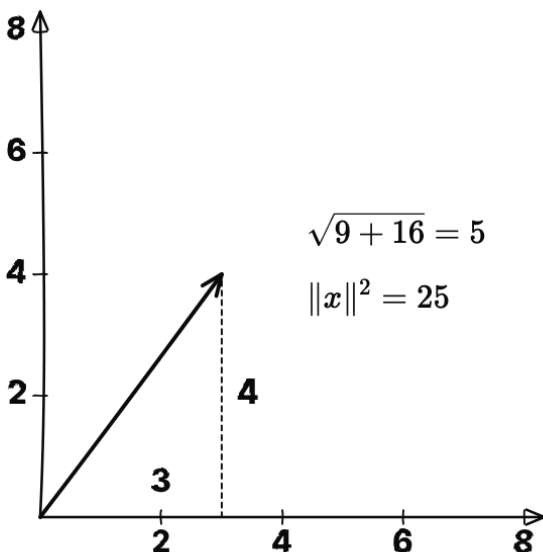
Вот маленькая ловушка, в которую программисты попадают каждый день. Возьмите четыре числа, поставленные столбиком, и четыре числа, лежащие простым списком, и попросите библиотеку вроде PyTorch их сложить. Вы ожидаете четыре суммы, по одной на каждую позицию. А получаете таблицу четыре строки на четыре столбца, шестнадцать чисел, без всякого сообщения об ошибке. Книга показывает ровно это в коротком окошке с кодом, и у виновника есть имя: бродкастинг.



Бродкастинг: четыре плюс четыре даёт шестнадцать

Бродкастинг — правило, заимствованное из библиотеки NumPy, по которому библиотека выравнивает формы двух тензоров справа и тихо повторяет один из них везде, где это заставляет формы совпасть. Столбец из четырёх для библиотеки — тензор с двумя направлениями, четыре строки на один столбец; у простого списка из четырёх направление только одно. Выровненные справа, одинокий столбец растягивается на четыре столбца, а список копируется вниз на четыре строки, и каждая пара складывается. Учебники математики аккуратно разделяют векторы-столбцы и векторы-строки, и книга следует этой привычке, но в коде столбец, строка и простой список — три разные формы, и смешивание их — вот откуда такие сюрпризы. Совет книги — просто держать это в голове, потому что почти каждый фреймворк ведёт себя так же.

Большинство моделей в книге построены из очень простых ингредиентов: сумм, умножений и нескольких изогнутых функций, таких как экспонента, синусы и косинусы и квадратные корни. У векторов, тензоров с одним направлением, своя арифметика, которую математики называют векторным пространством. Любые два вектора одной формы можно скомбинировать: каждый умножить на число и сложить позиция за позицией, получив третий вектор. Например, возьмите вектор два, один и прибавьте удвоенный вектор один, три: позиция один даёт два плюс два, то есть четыре, а позиция два даёт один плюс шесть, то есть семь, так что результат — четыре, семь.



Векторы как стрелки

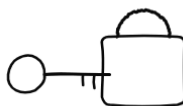
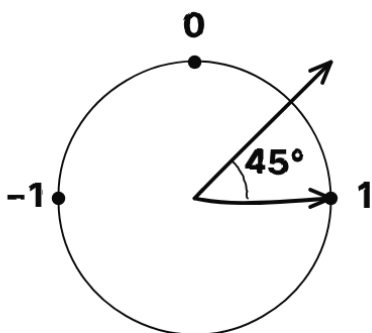
Полезно видеть вектор как стрелку от начала координат к точке. Сложить две стрелки тогда значит приставить их конец к началу, что достраивает параллелограмм, а длина стрелки, её расстояние от начала координат, называется нормой: возведите каждый элемент в квадрат, сложите квадраты и извлеките квадратный корень. Для стрелки три, четыре квадраты — девять и шестнадцать, в сумме двадцать пять, а квадратный корень — пять. Квадрат нормы, просто сумма квадратов элементов, здесь двадцать пять, оказывается особенно удобным.

Самая важная операция над векторами — скалярное произведение: перемножьте два вектора позиция за позицией и сложите результаты. Ответ — всегда одно число. Собственный

пример книги умножает вектор одна десятая, ноль, минус три десятых на вектор минус четыре, пять сотых, одна десятая. Три произведения — минус четыре десятых, ноль и минус три сотых, а их сумма — минус сорок три сотых.

$$(0.1)(-4) + (0)(0.05) + (-0.3)(0.1) \\ = -0.4 + 0 - 0.03 = -0.43$$

$$a \cdot b = \|a\| \|b\| \cos \theta$$



ПОИСК ШАБЛОНА

$$\cos \theta = \frac{1}{1.41} \approx 0.71$$

Скалярное произведение и угол

У скалярного произведения красивый геометрический смысл: оно равно длине первой стрелки, умноженной на длину второй, умноженной на косинус угла между ними. Поэтому если обе стрелки имеют длину один, скалярное произведение — просто косинус их угла, и тогда оно называется косинусным сходством. Оно равно единице, когда стрелки смотрят в одну сторону, минус единице, когда они смотрят в противоположные стороны, и нулю, когда они стоят под прямым углом, что называется

ортогональностью. Возьмите стрелки один, ноль и один, один: их скалярное произведение — один, их длины — один и около одной целой сорока одной сотой, так что косинус — около семи десятых, а это косинус сорока пяти градусов, ровно угла между ними.

Переверните это. Если одна стрелка длины один закреплена, какая другая стрелка длины один даёт наибольшее скалярное произведение? Ответ — сама эта стрелка: косинус наибольший, единица, когда угол равен нулю. Вот почему скалярное произведение в книге повсюду. Позже первый вектор будет входом, а второй — набором настраиваемых параметров, и их скалярное произведение наибольшее, когда вход резонирует с параметрами, подходя к ним как ключ к замку. Книга называет это сопоставлением с шаблоном. Обычный способ, которым функции поиска находят похожие предметы, например фотографии, похожие на выбранную вами, — превратить каждый предмет в вектор и сравнивать их именно этим косинусным сходством.

Два последних наблюдения сэкономят усилия позже. Сумма всех элементов вектора — его скалярное произведение с вектором, целиком состоящим из единиц: для два, пять, три умножение на один, один, один и сложение даёт десять. А квадрат расстояния между двумя векторами можно записать одними скалярными произведениями: скалярное произведение первого с самим собой плюс второго с самим собой минус удвоенное скалярное произведение этих двух. Проверьте на стрелках три, четыре и ноль, четыре: двадцать пять плюс шестнадцать минус дважды шестнадцать даёт девять, и действительно, две точки отстоят друг от друга на три, а три в квадрате — девять. Когда второй вектор

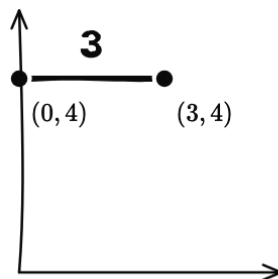
целиком из нулей, правило говорит, что квадрат длины вектора — его скалярное произведение с самим собой.

$$(2, 5, 3) \cdot (1, 1, 1) = 10$$

$$\|a - b\|^2 = a \cdot a + b \cdot b - 2a \cdot b$$

$$25 + 16 - 2 \cdot 16 = 9 = 3^2$$

$$b = 0, \quad \|a\|^2 = a \cdot a$$



Расстояние из скалярных произведений

Опыт

На листе бумаги в клетку нарисуйте стрелку от начала координат к точке три, четыре и измерьте её линейкой: она длиной в пять клеток. Затем нарисуйте стрелки один, ноль и один, один и измерьте угол между ними транспортиром: сорок пять градусов. Наконец, умножьте три, четыре на себя позиция за позицией и сложите: двадцать пять, то есть пять в квадрате, квадрат измеренной вами длины.

Запомним

Библиотеки применяют бродкастинг: выравнивают формы справа и повторяют тензоры до совпадения, так что сложение столбца из четырёх с простым списком из четырёх молча даёт таблицу четыре на четыре. Векторы можно умножать на число и складывать позиция за позицией; как стрелки, сложение достраивает параллелограмм, а норма — это длина, квадратный корень из суммы квадратов.

Скалярное произведение перемножает два вектора позиция за позицией и складывает, давая одно число; оно равно произведению двух длин на косинус угла, так что для единичных векторов это косинусное сходство, от единицы до минус единицы, где ноль означает ортогональность. Для закреплённого единичного вектора скалярное произведение наибольшее с самим этим вектором — сопоставление с шаблоном, на которое будут опираться входы и параметры. Сумма элементов — скалярное произведение с вектором из единиц, а квадраты расстояний можно записать одними скалярными произведениями.

Вопросы

1. Почему сложение столбца из четырёх чисел с простым списком из четырёх чисел даёт шестнадцать чисел в библиотеке вроде PyTorch?

- a. Потому что библиотека выравнивает формы справа и повторяет каждый тензор до совпадения, получая таблицу четыре на четыре
 - b. Потому что библиотека умножает вместо сложения
 - c. Потому что столбец всегда содержит шестнадцать чисел
 - d. Потому что библиотека сообщает об ошибке и заполняет пробел нулями
2. Две стрелки длины один имеют скалярное произведение ноль. О чём это говорит?
- a. Они смотрят в одну сторону
 - b. Они смотрят в противоположные стороны
 - c. Они ортогональны, стоят под прямым углом
 - d. Одна из них целиком из нулей
3. Одна единичная стрелка закреплена; какая единичная стрелка даёт с ней наибольшее скалярное произведение?
- a. Любая стрелка под прямым углом к ней
 - b. Стрелка, смотрящая в противоположную сторону
 - c. Стрелка из всех единиц
 - d. Сама эта стрелка
4. Как сумму всех элементов вектора записать в виде скалярного произведения?
- a. Как его скалярное произведение с самим собой
 - b. Как его скалярное произведение с вектором, целиком состоящим из единиц
 - c. Как его скалярное произведение с вектором из нулей

d. Как квадратный корень из его скалярного произведения с самим собой

Ответы

1. a) Бродкастинг выравнивает формы справа и молча повторяет тензоры; столбец из четырёх строк на один столбец и список из четырёх складываются в четыре строки на четыре столбца.
2. c) Для единичных векторов скалярное произведение — косинус угла, а нулевой косинус означает прямой угол.
3. d) Косинус наибольший, единица, когда угол равен нулю, так что лучшее совпадение — та же стрелка; это сопоставление с шаблоном, которым будут пользоваться входы и параметры.
4. b) Умножение каждого элемента на единицу и сложение просто складывает элементы: для два, пять, три это даёт десять.

Разобранные примеры

Скалярное произведение из книги Найдите скалярное произведение вектора одна десятая, ноль, минус три десятых и вектора минус четыре, пять сотых, одна десятая.

Перемножьте первые элементы: одна десятая на минус четыре — минус четыре десятых.

Перемножьте вторые элементы: ноль на пять сотых — ноль.

Перемножьте третьи элементы: минус три десятых на одну десятую — минус три сотых.

Сложите три произведения: минус четыре десятых плюс ноль минус три сотых — минус сорок три сотых.

Ответ: Минус сорок три сотых, одно число.

Задачи

1. Какова длина, норма, стрелки шесть, восемь?

Подсказка: Возведите каждый элемент в квадрат, сложите квадраты, затем извлеките квадратный корень.

Ответ: Десять: шесть в квадрате — тридцать шесть, восемь в квадрате — шестьдесят четыре, в сумме сто, а квадратный корень из ста — десять.

2. С помощью правила расстояний через скалярные произведения найдите, как далеко друг от друга точки один, два и четыре, шесть.

Подсказка: Вычислите скалярное произведение каждого вектора с самим собой и скалярное произведение двух векторов, затем объедините: первое плюс второе минус удвоенное смешанное.

Ответ: Первый с самим собой даёт один плюс четыре, пять; второй с самим собой — шестнадцать плюс тридцать шесть, пятьдесят два; смешанное произведение — четыре плюс двенадцать, шестнадцать. Пять плюс пятьдесят два минус тридцать два — двадцать пять, квадрат расстояния, так что точки отстоят на пять.

3. Каково косинусное сходство стрелок один, ноль и ноль, один, и какой угол оно означает?

Подсказка: Обе стрелки уже имеют длину один.

Ответ: Их скалярное произведение — один на ноль плюс ноль на один, то есть ноль. Поскольку обе имеют длину один, косинусное сходство равно нулю: они ортогональны, под углом девяносто градусов.

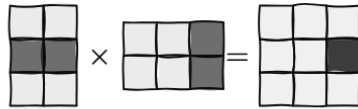
Таблицы чисел, которые действуют

Запишите маленькую таблицу: три квартиры на продажу, и для каждой два числа — площадь и число комнат. У вас три строки и два столбца, прямоугольник чисел, который математики называют матрицей. Прочитайте её строка за строкой, и проявится полезная вещь: каждая строка — одна квартира, маленький вектор сам по себе, а таблица — просто эти три вектора, положенные друг на друга. Книга называет такую стопку батчем векторов данных, и с этого момента модели будут записываться, и на бумаге, и в коде, так, чтобы работать сразу с целыми батчами. Когда фотоприложение на вашем телефоне учится, оно никогда не смотрит на картинки по одной; оно берёт батч, стопку картинок, и обрабатывает всю стопку за один раз.

$$(a \times b)(b \times c) = (a \times c)$$

80	3
55	2
120	4

БАТЧ ИЗ 3 ВЕКТОРОВ



РЯД І КОЛ. J c_{ij}

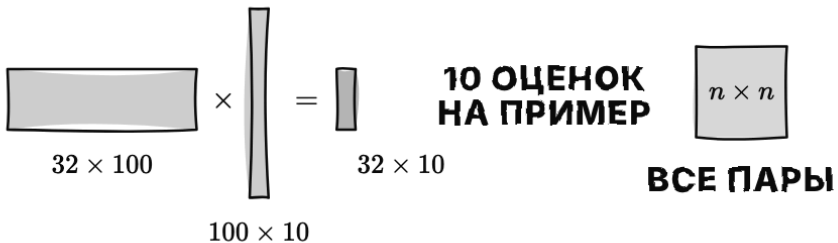
$$c_{ij} = \text{ряд}_i \cdot \text{кол}_j$$

Таблица — стопка векторов

Ключевая операция над матрицами — умножение, и у него есть правило о формах. Можно умножить таблицу из a строк и b столбцов на таблицу из b строк и c столбцов, и результат имеет a строк и c столбцов: два внутренних размера должны совпасть, и они исчезают. Каждый элемент результата — скалярное произведение: элемент в строке i и столбце j — это скалярное произведение строки i первой таблицы со столбцом j второй. Сделаем одно вручную. Возьмите первую таблицу со строками один, два и три, четыре, и вторую со строками пять, шесть и семь, восемь. Верхний левый элемент — строка один, два на столбец пять, семь: пять плюс четырнадцать, девятнадцать. Верхний правый элемент — один, два на шесть, восемь: шесть плюс шестнадцать, двадцать два. Нижняя строка даёт три на пять плюс

четыре на семь, сорок три, и три на шесть плюс четыре на восемь, пятьдесят. Итак, произведение — девятнадцать, двадцать два сверху и сорок три, пятьдесят снизу.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$



Умножаем вручную, затем оптом

Когда второй множитель — одиночный вектор, произведение — это матрица на вектор, что даёт вектор. Теперь взгляд через батчи окупается. Если каждая строка таблицы — один вектор данных и вы умножаете таблицу на перевёрнутую таблицу некоторых весов, вы вычисляете скалярное произведение каждого вектора данных с весами, всё одной-единственной операцией. Батч из тридцати двух примеров по сто чисел в каждом, умноженный на веса формы сто на десять, даёт тридцать два на десять: десять оценок для каждого из тридцати двух

примеров, сразу. А таблица, умноженная на свою же перевёрнутую версию, что даёт квадратную таблицу из n строк и n столбцов, содержит скалярное произведение каждой пары строк, так что все пары примеров сравниваются за один раз.

Второй вид произведения гораздо проще: произведение Адамара, которое перемножает две таблицы одной формы позиция за позицией. Ему не хватает богатых свойств настоящего матричного умножения, но сети постоянно используют его, чтобы маскировать, то есть обнулять некоторые элементы, или масштабировать элементы. Умножьте таблицу один, два над три, четыре позиция за позицией на таблицу один, ноль над ноль, один, и внедиагональные элементы исчезнут: один, ноль над ноль, четыре. Некоторые недавние семейства моделей сильно опираются на такие мультипликативные взаимодействия, и книга к ним вернётся.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \odot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$$

МАСКА

ПОЭЛЕМЕНТНО

$$e^0 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

МАТРИЧНАЯ EXP

$$\exp(0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

TORCH.LINALG

СУММА РЯДА → КВАРТИРА

СУММА СТОЛБЦА → ИТОГИ

Позиция за позицией и две экспоненты

Книга также предупреждает о ловушке обозначений. Запись экспоненты от матрицы обычно означает применение экспоненты к каждому элементу отдельно, и это работает для тензоров любой формы. Но в линейной алгебре есть и настоящая матричная экспонента, определённая только для квадратных матриц как бесконечная сумма матрицы, умноженной на себя всё большее число раз, где каждое слагаемое делится на растущий факториал. Эти две вещи очень разные. Для таблицы, полной нулей, поэлементная экспонента даёт таблицу, полную единиц, тогда как настоящая матричная экспонента даёт единицы только на диагонали и нули в остальных местах. Поэтому библиотеки вроде PyTorch держат по-настоящему матричные операции, такие как матричная экспонента, обратные матрицы и

определители, в отдельном модуле линейной алгебры, как показывает окошко с кодом в книге.

Наконец, редукция, такая как сумма или среднее, проходит вдоль целого направления таблицы, если не заданы границы. В коде это метод тензора с аргументом, называющим направление: суммирование таблицы квартир вдоль столбцов даёт для каждой квартиры площадь плюс комнаты; суммирование вдоль строк даёт итоги по всем трём квартирам.

Почему матричное умножение определено этим странным способом «строка на столбец», а не позиция за позицией? Книга отвечает во вставке. Функция на векторах линейна, когда сначала масштабировать и сложить входы даёт тот же результат, что масштабировать и сложить выходы. Каждую такую функцию можно записать матрицей, и применить функцию к вектору — это ровно умножить матрицу на вектор. Умножение двух матриц тогда соответствует выполнению одной функции после другой, что называется композицией. Например, если одна матрица удваивает каждый элемент, а другая меняет два элемента местами, их произведение — одна матрица, которая удваивает, а затем меняет местами: из три, пять получается десять, шесть, как ни считай.



Матрицы составляют функции

Запомним

Матрица — таблица из строк и столбцов, которую лучше всего видеть как стопку векторов данных, батч, и модели записываются так, чтобы работать сразу с целыми батчами. Матрицы перемножаются, когда внутренние размеры совпадают; каждый элемент результата — скалярное произведение строки первой со столбцом второй, так что одно произведение вычисляет много скалярных произведений сразу. Произведение Адамара перемножает позиция за позицией и используется для маскирования и масштабирования; экспонента, записанная на матрице, означает поэлементную, в отличие от настоящей матричной экспоненты квадратных матриц. Редукции, такие как суммы, проходят вдоль целого направления, выбираемого в коде аргументом оси. Матричное умножение определено так потому, что матрицы — линейные функции: умножение на вектор применяет функцию, а умножение двух матриц составляет композицию двух функций.

Вопросы

1. Таблицу из 32 строк и 100 столбцов умножают на таблицу из 100 строк и 10 столбцов. Какова форма результата?
 - a. 100 строк на 100 столбцов
 - b. 32 строки на 10 столбцов
 - c. 10 строк на 32 столбца

- d. Таблицы нельзя перемножить
2. Что содержит каждый элемент матричного произведения?
- a. Скалярное произведение строки первой таблицы со столбцом второй
 - b. Произведение двух элементов в одной и той же позиции
 - c. Сумму всех элементов обеих таблиц
 - d. Наибольший элемент строки
3. Для таблицы, полной нулей, чем различаются поэлементная экспонента и настоящая матричная экспонента?
- a. Обе дают таблицу нулей
 - b. Обе дают таблицу единиц
 - c. Поэлементная даёт единицы только на диагонали
 - d. Поэлементная даёт все единицы; матричная экспонента даёт единицы на диагонали и нули в остальных местах
4. Почему матричное умножение определено как «строка на столбец», а не позиция за позицией?
- a. Потому что так быстрее на графических чипах
 - b. Потому что попозиционных произведений не существует
 - c. Потому что матрицы представляют линейные функции, и это произведение соответствует применению одной функции после другой
 - d. Потому что оно всегда даёт квадратный результат

Ответы

1. b) Внутренние размеры, 100 и 100, совпадают и исчезают; результат сохраняет внешние размеры, 32 на 10, десять оценок для каждого из тридцати двух примеров.

2. a) Элемент в строке i и столбце j — это скалярное произведение строки i первой таблицы со столбцом j второй.

3. d) Экспонента нуля для каждого элемента — единица, тогда как настоящая матричная экспонента нулевой матрицы оставляет единицы только на диагонали; эти две операции очень разные.

4. c) Каждая линейная функция — матрица; умножение на вектор её применяет, а умножение двух матриц составляет композицию двух функций.

Разобранные примеры

Перемножаем две маленькие таблицы Умножьте таблицу со строками два, ноль и один, три на таблицу со строками один, четыре и два, один.

Верхний левый: строка два, ноль со столбцом один, два даёт два на один плюс ноль на два, то есть два.

Верхний правый: строка два, ноль со столбцом четыре, один даёт восемь плюс ноль, то есть восемь.

Нижний левый: строка один, три со столбцом один, два даёт один плюс шесть, то есть семь.

Нижний правый: строка один, три со столбцом четыре, один даёт четыре плюс три, то есть семь.

Ответ: У произведения строки два, восемь и семь, семь.

Задачи

1. Можно ли таблицу 3 строки на 2 столбца умножить на другую таблицу 3 строки на 2 столбца? А на её собственную

перевёрнутую версию?

Подсказка: Проверьте, совпадают ли внутренние размеры.

Ответ: Нет: внутренние размеры были бы 2 и 3, а они различаются. Да с перевёрнутой версией, которая 2 на 3: внутренние размеры оба равны 2, и результат — 3 на 3, скалярные произведения каждой пары строк.

2. Умножьте таблицу пять, шесть над семь, восемь позиция за позицией, произведением Адамара, на маску ноль, один над один, ноль.

Подсказка: Каждый элемент умножается только на элемент в той же позиции.

Ответ: Ноль, шесть над семь, ноль: маска сохраняет внедиагональные элементы и обнуляет диагональные.

3. Одна матрица удваивает оба элемента вектора; другая меняет два его элемента местами. Примените «удвоить, затем поменять» к вектору три, пять и проверьте, что «поменять, затем удвоить» здесь даёт тот же результат.

Подсказка: Выполняйте шаги по очереди.

Ответ: Удвоение даёт шесть, десять, а обмен даёт десять, шесть. Сначала обмен даёт пять, три, а удвоение даёт десять, шесть. Обе композиции дают десять, шесть, как дала бы и их единая объединённая матрица.

Считаем работу

Перемножьте две таблицы вручную: одну из двух строк и трёх столбцов и одну из трёх строк и четырёх столбцов. У результата две строки и четыре столбца, восемь элементов, и каждый элемент — скалярное произведение трёх пар, так что вы делаете восемь умножить на три, двадцать четыре умножения. Теперь удвойте каждый размер: четыре строки на шесть, умножить на шесть на восемь. У результата тридцать два элемента по шесть умножений в каждом, всего сто девяносто два, в восемь раз больше, чем прежде, хотя каждый размер лишь удвоился. То, как растёт работа с ростом размеров, информатики называют сложностью, и книга вводит её на примере матричного умножения.

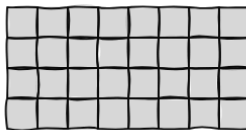
$$(2 \times 3)(3 \times 4)$$



$$8 \times 3 = 24$$

× 8

$$(4 \times 6)(6 \times 8)$$



$$32 \times 6 = 192$$

$$(a \times b)(b \times c)$$

A·C СКАЛЯРНЫХ, ДЛИНА В

время $\sim a b c$

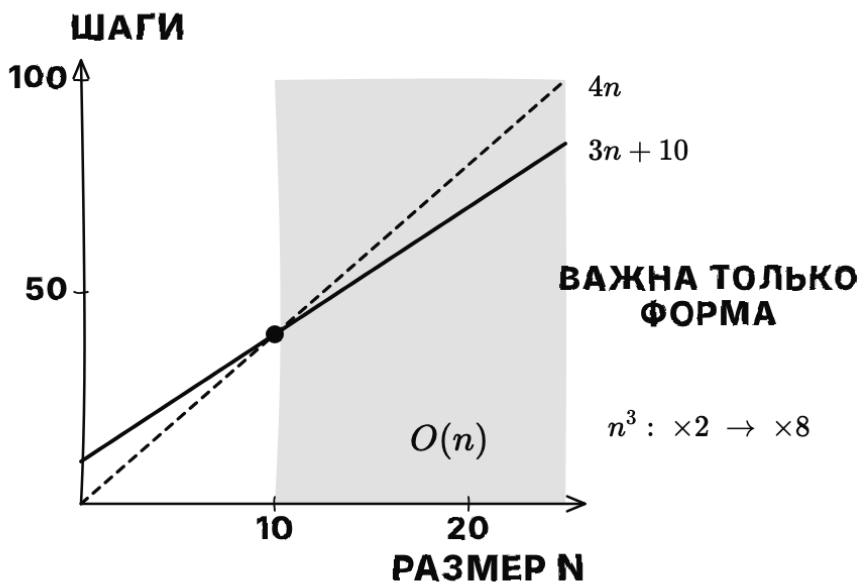
память $\sim a c$

Удвоим размеры — работа ввосемьро

Если применять определение напрямую, таблица из a строк и b столбцов на таблицу из b строк и c столбцов требует a умножить на c скалярных произведений, каждое длины b . Число операций называется временной сложностью. Память, которая нужна вычислению, для простой программы «шаг за шагом» — это просто размер таблицы результата, и это называется пространственной сложностью.

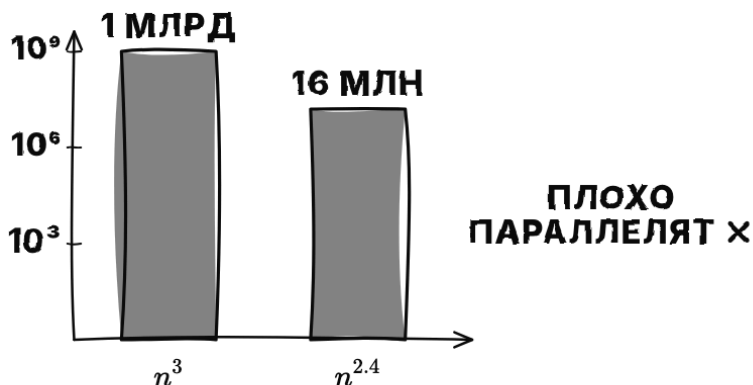
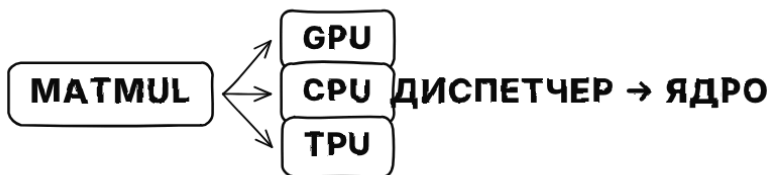
Чтобы говорить о росте, не теряясь в деталях конкретных машин, информатика использует обозначение O большое, где O происходит от немецкого слова *Ordnung*, порядок, как в «порядке приближения». Сказать, что работа программы имеет порядок g , означает вот что: начиная с некоторого размера работа никогда

не превышает некоторого фиксированного кратного g . Например, программа, которой нужно три размера плюс десять шагов, имеет порядок размера, потому что как только размер достигает десяти, три размера плюс десять никогда не больше четырёх размеров. За этой точкой константы и малые дополнительные слагаемые перестают иметь значение, и считается только общая форма роста; это называется асимптотическим анализом. На этом языке наивное матричное умножение имеет порядок a умножить на b умножить на c и растёт пропорционально каждому из трёх размеров. Для двух квадратных таблиц из n строк и n столбцов это n на n на n , и мы говорим, что умножение кубическое: удвойте n , и работа вырастет восьмикратно, ровно как при ручном подсчёте.



О большое: ограничено с некоторого размера

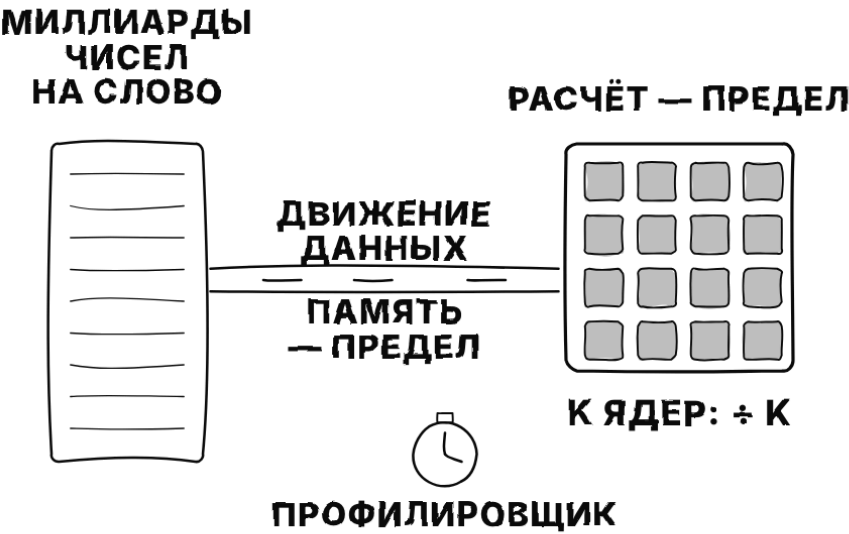
Рассуждение через O большое изящно и важно, но книга предупреждает, что оно не решает, какой метод действительно быстрее. Это зависит от деталей, таких как оборудование и вид параллельной работы, который оно поддерживает. Когда вы просите библиотеку вроде PyTorch перемножить две матрицы, библиотека выбирает конкретную низкоуровневую процедуру, называемую ядром, для оборудования, на котором работает ваша программа, будь то графический чип, обычный процессор или специализированный чип, на шаге, который называется диспетчеризацией; компиляторы вроде XLA идут дальше и сплавляют несколько операций в одну, настроенную под это оборудование. Поразительный пример: у лучших известных методов умножения двух квадратных таблиц работа порядка n в степени ниже двух целых четырёх десятых, намного лучше куба наивного метода. Для таблиц тысяча на тысячу куб — это миллиард операций, тогда как тысяча в степени две целых четыре десятых — лишь около шестнадцати миллионов. Но эти хитрые методы трудно эффективно распределить по тысячам маленьких ядер графического чипа, так что на практике их применяют редко.



Лучший рост, редкое применение

Ещё два предела O большого. Первое: k процессоров, работающих параллельно, ничего не меняют в порядке роста: в лучшем случае они делят время на k , постоянный множитель, так что кубический метод остаётся кубическим. Второе: O большое не учитывает время на перемещение данных с одного места на другое, а оно может стать главным узким местом. Чат-ассистент, отвечающий вам, часто ограничен не столько арифметикой, сколько тем, как быстро его миллиарды чисел можно вытащить из памяти для каждого слова, которое он пишет. Программа, ограниченная перемещением данных, называется ограниченной памятью, в противоположность ограниченной вычислениями, и единственный надёжный способ узнать, какая у вас, — запустить профилировщик, инструмент, который измеряет, куда на самом

деле уходит время. Книга заключает, что оценить стоимость алгоритма далеко не просто, потому что теоретический рост и наблюдаемая скорость взаимодействуют.



Ограничено вычислениями или памятью

Запомним

Сложность описывает, как работа растёт с размером: умножение таблицы a на b на таблицу b на c требует a умножить на c скалярных произведений длины b , это её временная сложность, и нуждается в памяти под результат, это её пространственная сложность. Оно большое сохраняет только форму роста для больших размеров, игнорируя константы: наивное матричное умножение имеет порядок a умножить на b умножить на c , кубический для квадратных таблиц, так что удвоение размера даёт восьмикратную работу. Реальная скорость зависит и от оборудования, ядра, которое диспетчеризует библиотека, и компиляторов; методы с лучшим ростом, ниже степени две целых четыре десятых, применяются редко, потому что плохо параллелятся. Параллельные процессоры дают лишь постоянное ускорение, а перемещение данных может доминировать, делая программу ограниченной памятью, а не вычислениями, что может выявить только профилировщик.

Вопросы

1. Наивное умножение двух квадратных таблиц кубическое. Что происходит с работой, когда размер таблиц удваивается?
 - а. Она удваивается

- b. Она вырастает вчетверо
 - c. Она вырастает ввосемьмеро
 - d. На графическом чипе она остаётся прежней
2. Почему методы умножения матриц с ростом ниже степени две целых четыре десятых редко применяются на практике?
- a. Их трудно эффективно распределить по сильно параллельному оборудованию, такому как графические чипы
 - b. Они дают неверные ответы для больших таблиц
 - c. Им нужно больше памяти, чем таблице результата
 - d. Они работают только для таблиц меньше тысячи строк
3. Программа работает на 8 процессорах вместо 1. Что говорит О большое о её порядке роста?
- a. Он падает с кубического до квадратичного
 - b. Он становится в восемь раз меньше по порядку
 - c. Его больше нельзя вычислить
 - d. Он не меняется: параллелизм даёт в лучшем случае постоянное ускорение
4. Что значит, что программа ограничена памятью?
- a. Она использует больше памяти, чем есть у компьютера
 - b. Её скорость ограничена главным образом перемещением данных, а не арифметикой
 - c. Она сохраняет каждый промежуточный результат на диск
 - d. У неё пространственная сложность порядка ноль

Ответы

1. **с)** Кубический рост означает, что работа идёт как размер, умноженный на себя три раза, так что удвоение размера умножает работу на два на два на два, восемь.
2. **а)** Их лучший асимптотический рост не переходит в скорость, потому что они плохо параллелятся на GPU.
3. **д)** С k процессорами время делится не более чем на k , постоянный множитель, который O большое игнорирует.
4. **б)** O большое не учитывает перемещение данных, которое может стать главным узким местом; тогда программа ограничена памятью, а не вычислениями, что может показать профилировщик.

Разобранные примеры

Считаем умножения Сколько умножений нужно наивному методу, чтобы перемножить таблицу 10 строк на 20 столбцов с таблицей 20 строк на 5 столбцов, и какова форма результата?

Внутренние размеры оба равны 20, так что произведение существует и имеет 10 строк и 5 столбцов.

Это 10 умножить на 5, то есть 50 элементов.

Каждый элемент — скалярное произведение длины 20, так что ему нужно 20 умножений.

Всего 50 умножить на 20 даёт 1000 умножений, то есть a умножить на b умножить на c при a 10, b 20 и c 5.

Ответ: 1000 умножений; результат — 10 на 5.

Задачи

1. Наивное умножение двух квадратных таблиц занимает одну секунду. Сколько примерно займут таблицы втрое большего

размера?

Подсказка: Работа кубическая по размеру.

Ответ: Около 27 секунд: три на три на три — двадцать семь, так что работа, а значит и примерно время, вырастает в двадцать семь раз.

2. Покажите, что программа, которой нужно пять размеров плюс сто шагов, имеет порядок размера.

Подсказка: Найдите кратное размера, которое не превышаетя после некоторой точки.

Ответ: Как только размер достигает ста, пять размеров плюс сто никогда не больше шести размеров, так что после этой точки работа не превышает фиксированного кратного размера: она имеет порядок размера.

3. Для квадратных таблиц из тысячи строк сравните работу наивного метода с работой метода, чья работа растёт как размер в степени две целых четыре десятых.

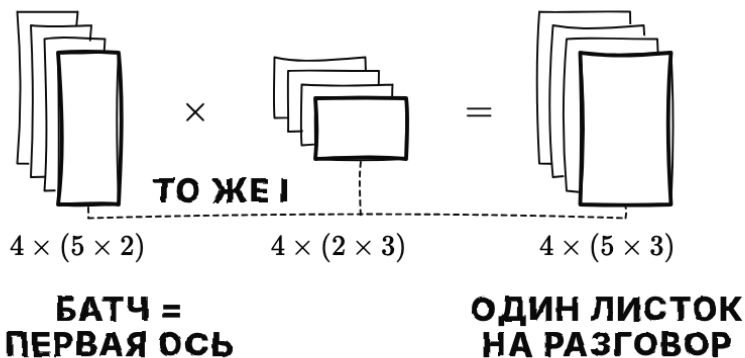
Подсказка: Тысяча в кубе и тысяча в степени две целых четыре десятых, что около шестнадцати миллионов.

Ответ: Наивному методу нужен около миллиарда операций, хитрому — около шестнадцати миллионов, примерно в шестьдесят раз меньше; и всё же хитрый применяют редко, потому что он плохо параллелится на графических чипах.

Стопки таблиц и сокращённая запись буквами

Представьте четырёх учеников, у каждого листок с домашним заданием: умножить таблицу пять на два на таблицу два на три. Каждый ученик работает один над своей парой и сдаёт результат пять на три. Собранные вместе, четыре ответа образуют стопку: четыре результата пять на три. Внутри не произошло ничего нового, одно и то же умножение просто сделали четыре раза бок о бок. Так работает большинство операций над тензорами более чем с двумя направлениями, и книга называет эту операцию батчевым матричным умножением.

БАТЧЕВЫЙ МАТМУЛ

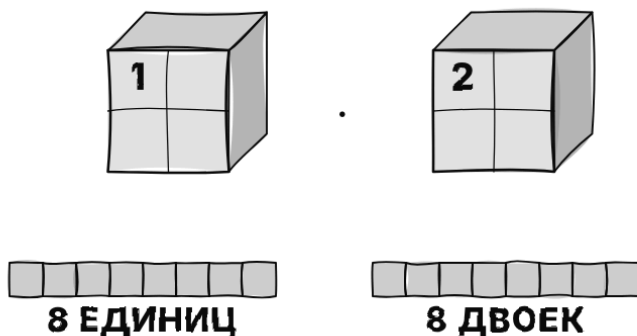


Батчевое матричное умножение

Векторы и матрицы особенные, потому что многие операции имеют смысл для них и становятся неудобными или неопределёнными дальше: матричное умножение, настоящая матричная экспонента, определители. Поэтому когда у тензоров больше направлений, нужные нам операции в основном двух видов: батчевые версии матричных операций или сочетания матричных операций с редукциями, такими как суммы. Для батчевого умножения возьмите первую стопку из n таблиц, каждая a на b , и вторую стопку из n таблиц, каждая b на c . Результат — стопка из n таблиц, каждая a на c , где таблица номер i — это просто таблица i первой стопки, умноженная на таблицу i второй. Библиотеки считают ведущее направление, первое, батчем и обрабатывают его незаметно: в PyTorch обычная

команда умножения, применённая к стопке из четырёх таблиц пять на два и стопке из четырёх таблиц два на три, возвращает стопку из четырёх таблиц пять на три, ровно как показывает окошко с кодом в книге. Именно так чат-ассистент отвечает на сообщения многих людей сразу: каждый разговор — один листок в стопке, и одни и те же умножения выполняются на всех листках вместе.

Для второго вида книга даёт обобщённое скалярное произведение. Возьмите два тензора одной формы с тремя направлениями, перемножьте их позиция за позицией и сложите каждое произведение. Это не что иное, как обычное скалярное произведение двух тензоров, развёрнутых в длинные списки. Например, у двух маленьких кубиков два на два на два, первый заполнен единицами, второй двойками, восемь позиций, каждое произведение равно двум, и обобщённое скалярное произведение равно шестнадцати. Книга добавляет, что этот короткий обзор покрывает большинство операций над тензорами, используемых в остальной книге, и что всё новое будет введено по мере надобности.



EINSUM

Скалярное произведение целых тензоров

Затем идёт необязательный раздел о хитрой сокращённой записи под названием `einsum`, по соглашению Эйнштейна о суммировании, которая может описать практически любую операцию над тензорами короткой строкой букв. Идея в том, чтобы дать каждому направлению каждого тензора букву. Для батчевого умножения у первой стопки направления i, j, z , у второй — i, z, k , а у результата — i, j, k . Для обобщённого скалярного произведения у обоих тензоров направления i, j, k , а у результата букв нет вовсе, поскольку это одно число. Правило, в упрощённой форме, которой пользуется книга, таково: любая буква, которая появляется во входах, но не в результате, суммируется. В батчевом умножении буква z появляется в обоих входах, но не в результате, так что это направление, по которому

перемножают и складывают, а это ровно скалярное произведение внутри каждого умножения таблиц. В обобщённом скалярном произведении все три буквы исчезают из результата, так что суммируется всё. Вся операция тогда записывается одной строкой: буквы входов через запятую, стрелка и буквы результата.

ВО ВХОДАХ, НЕ В РЕЗУЛЬТАТЕ → СУММА СУММИРУЕТСЯ


ijz, *izk* → *ijk*

СТОПКА 1 СТОПКА 2

ijk, *ijk* →
ОДНО ЧИСЛО

EINOPS

ijz, *ikz* → *ijk*

ПЕРЕВЁРНУТО

БАТЧ ИЗ 2, ВРУЧНУЮ

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Einsum: буквы для направлений

Большинство фреймворков предоставляют операцию einsum, которая принимает ровно такую строку. Её первое преимущество — не нужно помнить собственные команды каждой библиотеки: в одном PyTorch несколько функций умножения с разными правилами форм и поведением бродкастинга, тогда как einsum описывает их все одинаково. Второе преимущество —

переносимость: строка для батчевого умножения одинакова в PyTorch и в JAX, так что код переходит из одной библиотеки в другую без изменений. Развернуть направление тоже легко: если во второй стопке таблицы лежат по-другому, с направлениями i , k , z , вы просто пишете эти буквы, и `einsum` умножает на перевёрнутые таблицы без лишнего шага. По этим причинам `einsum` и пакеты, построенные на той же идее, такие как `einops`, стали очень популярны.

Опыт

Возьмите два листа бумаги и на каждом нарисуйте стопку из двух маленьких таблиц два на два. На первом листе напишите таблицы один, два над три, четыре и один, ноль над ноль, один; на втором — два, ноль над ноль, два и пять, шесть над семь, восемь. Умножьте таблицу один на таблицу один и таблицу два на таблицу два: должно получиться два, четыре над шесть, восемь, а затем пять, шесть над семь, восемь. Это батчевое умножение с батчем из двух, сделанное вручную.

Запомним

За пределами двух направлений большинство операций — батчевые матричные операции или матричные операции в сочетании с редукциями. Батчевое матричное умножение умножает таблицу i одной стопки на таблицу i другой, давая стопку результатов, и библиотеки обрабатывают ведущее батчевое направление автоматически. Обобщённое скалярное произведение перемножает два тензора одной формы позиция за позицией и складывает всё — скалярное произведение их развёрнутых форм. `Einsum` даёт каждому направлению букву и суммирует по буквам, которые есть во входах, но не в результате, так что одна строка описывает операцию одинаково в PyTorch и JAX, что и сделало его, и `einops`, популярными.

Вопросы

1. Стопку из 4 таблиц 5 на 2 умножают батчево на стопку из 4 таблиц 2 на 3. Что получается?
 - a. Одна таблица 20 на 3
 - b. Стопка из 4 таблиц 5 на 3
 - c. Стопка из 4 таблиц 2 на 2
 - d. Ничего, потому что формы не совпадают
2. В правиле `einsum`, которым пользуется книга, по каким направлениям идёт суммирование?

- a. По каждой букве, которая появляется в результате
 - b. Только по первой букве каждого входа
 - c. По буквам, которые появляются во входах, но не в результате
 - d. По буквам, которые появляются только в одном входе
3. Что такое обобщённое скалярное произведение двух тензоров одной формы?
- a. Сумма всех их попозиционных произведений, скалярное произведение их развёрнутых форм
 - b. Стопка матричных произведений
 - c. Произведение их наибольших элементов
 - d. Тензор той же формы, что и входы
4. Почему einsum полезен при переносе кода из PyTorch в JAX?
- a. Потому что иначе JAX не умеет перемножать матрицы
 - b. Потому что einsum работает быстрее любой другой команды
 - c. Потому что einsum избавляет от необходимости в батчах
 - d. Потому что одна и та же строка букв описывает операцию одинаково в обеих библиотеках

Ответы

1. **b)** Батчевое умножение умножает таблицу i на таблицу i ; каждое 5 на 2 умножить на 2 на 3 даёт 5 на 3, и таких четыре.
2. **c)** Буква, которая есть во входах, но отсутствует в результате, суммируется; в батчевом умножении это z , направление внутренних скалярных произведений.

3. a) Перемножить позиция за позицией и сложить всё — то же самое, что обычное скалярное произведение двух тензоров, записанных длинными списками.

4. d) Строка `einsum` для батчевого умножения одинакова в PyTorch и JAX, так что код переносится без изменений, и не нужно помнить команды каждой библиотеки.

Разобранные примеры

Формы в батчевом умножении В окошке с кодом в книге стопку формы 4 на 5 на 2 умножают на стопку формы 4 на 2 на 3. Найдите форму результата.

Ведущее направление, 4, — это батч: есть четыре пары таблиц.

Каждая пара — таблица 5 на 2 умножить на таблицу 2 на 3; внутренние размеры, 2 и 2, совпадают.

Значит, каждое произведение — 5 на 3.

Сложение четырёх произведений в стопку даёт стопку формы 4 на 5 на 3.

Ответ: Стопка формы 4 на 5 на 3.

Задачи

1. Два тензора формы 2 на 2 в развёрнутом виде читаются как один, два, три, четыре и два, ноль, один, один. Каково их обобщённое скалярное произведение?

Подсказка: Перемножьте позиция за позицией, затем сложите всё.

Ответ: Девять: один на два — два, два на ноль — ноль, три на один — три, четыре на один — четыре, а два плюс ноль плюс три плюс четыре — девять.

2. С помощью правила букв опишите обычное матричное умножение таблицы i на z на таблицу z на k .

Подсказка: Какая буква должна исчезнуть в результате?

Ответ: Входы несут буквы i , z и z , k , а результат сохраняет i , k . Буква z появляется во входах, но не в результате, так что по ней суммируют: каждый элемент — скалярное произведение вдоль z , а это матричное умножение.

3. Стопка содержит 32 разговора, каждый — таблица 10 на 64, и каждый нужно умножить на его собственную таблицу 64 на 64. Какова форма результата и как он вычисляется?

Подсказка: Думайте о каждом разговоре как об одном листке в батче.

Ответ: Стопка из 32 таблиц 10 на 64: таблица 10 на 64 каждого разговора на его таблицу 64 на 64 даёт 10 на 64, и 32 произведения выполняются бок о бок как одно батчевое умножение.

Наклон в точке

Возьмите калькулятор и возведите число три в квадрат: девять. Теперь возведите в квадрат три целых одну сотую: девять целых шестьсот одна десятитысячная. Вход сдвинулся на одну сотую, а выход сдвинулся на шесть сотых и чуточку больше, так что около тройки квадрат меняется примерно в шесть раз быстрее своего входа. Попробуйте три целых одну тысячную, и отношение подойдёт к шести ещё ближе. Это предельное отношение — производная, и это первый инструмент, нужный книге, потому что градиенты, её многомерные родственники, — то, что движет обучением каждой модели в книге.

x	x^2	ОТНОШЕНИЕ
3	9	
3.01	9.0601	6.01
3.001	9.006001	6.001 $\longrightarrow$ 6

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

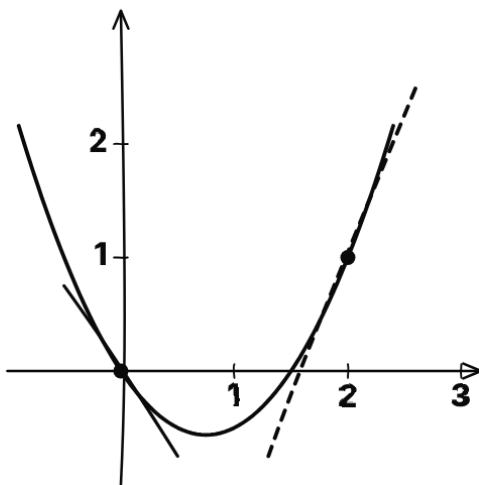
запись: $\frac{\partial f}{\partial x}$ или $f'(x)$

Квадрат около тройки

Для функции с одним числом на входе и одним на выходе производная в точке определяется ровно так, как вы только что сделали: сдвиньте вход на малую величину, посмотрите, насколько изменился выход, разделите изменение выхода на сдвиг и дайте сдвигу стремиться к нулю. Книга пишет её несколькими способами: завитая d для производных и градиентов любого размера, тот же знак с мелко подписанным входом, чтобы сказать, какой вход меняется, а для функций одного числа — штрих после имени функции, известный как обозначение Лагранжа. Она не беспокоится о том, всегда ли производная существует, что не гарантировано даже для гладкой на вид непрерывной кривой; она просто предполагает это и

возвращается к неудобным случаям, таким как острый угол модуля в нуле, в шестой главе.

Для знакомых функций определение даёт знакомые правила. У степени входа производная равна этой степени, умноженной на вход в степени на единицу меньше: у куба производная — три квадрата, так что в точке два это три на четыре, двенадцать. У логарифма производная — единица, делённая на вход, так что в точке четыре его наклон — одна четверть. У синуса производная равна косинусу, так что в нуле его наклон — единица.



$$(x^3)' = 3x^2 \rightarrow 12$$

$$(\log x)' = \frac{1}{x} \rightarrow \frac{1}{4}$$

$$(\sin x)' = \cos x \rightarrow 1$$

$$f'(x) = 2x - 1.5$$

**НАКЛОН В 0:
-1.5, ПАДАЕТ**

**НАКЛОН В 2:
+2.5, РАСТЁТ**

Наклоны касательных в долине

Геометрически производная — это наклон касательной, прямой, которая лишь касается кривой в точке, и это лучшее

прямолинейное приближение функции в этом месте. Рисунок в книге изображает кривую $x^2 - 1,5x$, пологую долину, с касательными, проведёнными в двух точках. Её производная — $2x - 1,5$. В нуле наклон равен минус полтора: кривая идёт вниз направо, так что там она убывает. В двойке наклон равен плюс двум с половиной: кривая поднимается направо. Вот урок на всю книгу: знак наклона говорит вам, в малой окрестности, куда направляется функция, и то же понимание перенесётся на функции многих входов. Когда обучают фотоприложение, именно такой наклон, вычисленный сразу для миллионов ручек, говорит каждой ручке, куда повернуться.

Три правила позволяют дифференцировать функции, собранные из более простых, и все три распространяются на много измерений. Линейность: производная суммы — сумма производных; для $x^2 + 3x$ в единице это два плюс три, пять. Правило произведения: производная произведения — это производная первого на второе плюс первое на производную второго; для $x^2 \cdot x^3$ это даёт два икс в четвёртой плюс три икс в четвёртой, пять икс в четвёртой, что и впрямь производная икс в пятой. Цепное правило: производная одной функции, применённой к результату другой, — это внешняя производная, взятая во внутреннем значении, умноженная на внутреннюю производную. Для $(2x + 1)^2$ в единице: внутреннее значение — три, внешняя производная там — шесть, внутренняя производная — два, так что наклон — двенадцать; если сначала раскрыть квадрат и продифференцировать, получится те же двенадцать. Цепное правило — то, что надо запомнить: наклон сквозь цепочку шагов — это произведение наклонов шагов, и эта идея понесёт на себе всю книгу.

СУММА $(f + g)' = f' + g'$ $(x^2 + 3x)'|_1 = 2 + 3 = 5$

ПРОИЗВ. $(fg)' = f'g + fg'$ $(x^2x^3)' = 2x^4 + 3x^4 = 5x^4$

ЦЕПЬ

$$(f(g(x)))' = f'(g(x)) g'(x)$$

$$((2x + 1)^2)'|_1 = \underbrace{6}_{\text{внѣшн.}} \cdot \underbrace{2}_{\text{внутр.}} = 12$$

Правила суммы, произведения и цепочки

Запомним

Производная функции одного числа — предел изменения выхода, делённого на уменьшающийся сдвиг входа; возведение в квадрат около тройки даёт шесть. У степеней, логарифмов и синусов знакомые производные: куб даёт три квадрата, логарифм — единицу на вход, синус — косинус. Производная — наклон касательной и лучшее прямолинейное приближение; её знак говорит, растёт или падает функция поблизости, как показывает долина на рисунке. Три правила строят производные составных функций: суммы складываются, произведения следуют правилу произведения, а цепочки перемножают свои наклоны — цепное правило.

Вопросы

1. Наклон функции в точке отрицателен. Что это говорит о функции около этой точки?
 - a. Она падает с ростом входа и растёт влево
 - b. Она растёт с ростом входа
 - c. Она достигла своего наименьшего значения
 - d. Она там не дифференцируема
2. Какое правило даёт наклон одной функции, применённой к результату другой?
 - a. Линейность

- b. Правило произведения
 - c. Цепное правило: внешний наклон во внутреннем значении на внутренний наклон
 - d. Правило степени
3. Почему книга так заботится о производных?
- a. Потому что каждая модель должна быть многочленом
 - b. Потому что производные заменяют потребность в данных
 - c. Потому что они важны только для синусов и логарифмов
 - d. Потому что градиенты, их форма для многих входов, движут обучением каждой модели в книге
4. Каков наклон логарифма при входе четыре?
- a. Четыре
 - b. Одна четверть
 - c. Ноль
 - d. Минус одна четверть

Ответы

1. **a)** Отрицательный наклон означает, что кривая идёт вниз направо в малой окрестности, как долина на рисунке в нуле.
2. **c)** Цепное правило перемножает наклоны шагов; для два x плюс один в квадрате в единице оно даёт шесть на два, двенадцать.
3. **d)** Градиенты, многомерные родственники производной, говорят каждому настраиваемому числу, куда повернуться во время обучения.
4. **b)** Производная логарифма — единица, делённая на вход, так что в точке четыре это одна четверть.

Разобранные примеры

Читаем наклон кривой из книги Рисунок в книге показывает кривую $y = x^2 - 1.5x$. Найдите её наклон в единице и точку, где кривая плоская.

Правило степени даёт два x для $y = x^2$, а линейность даёт -1.5 для второго слагаемого, так что производная — $2x - 1.5$.

В единице наклон — -0.5 , то есть одна вторая: кривая там слегка поднимается.

Кривая плоская там, где наклон равен нулю: $2x - 1.5 = 0$ равно нулю.

Это происходит при $x = 0.75$, на дне долины.

Ответ: Наклон одна вторая в единице; плоско при $x = 0.75$, в самой низкой точке.

Задачи

1. Оцените наклон квадрата в пятёрке, сдвинув вход на одну сотую, и сравните с правилом степени.

Подсказка: Возведите в квадрат пять целых одну сотую, вычитите двадцать пять, разделите на одну сотую.

Ответ: Пять целых одна сотая в квадрате — двадцать пять целых одна тысяча одна десятичная, так что изменение — около одной десятой, делённое на одну сотую даёт десять целых одну сотую.

Правило степени даёт два на пять, десять, значение, к которому стремится отношение при уменьшении сдвига.

2. С помощью цепного правила найдите наклон три x минус один в кубе при x равном единице.

Подсказка: Сначала найдите внутреннее значение, затем внешний наклон в нём, затем умножьте на внутренний наклон.

Ответ: Внутреннее значение — два. Внешняя функция — куб с наклоном три квадрата, так что в двойке это двенадцать. Внутренний наклон — три, так что наклон — двенадцать на три, тридцать шесть.

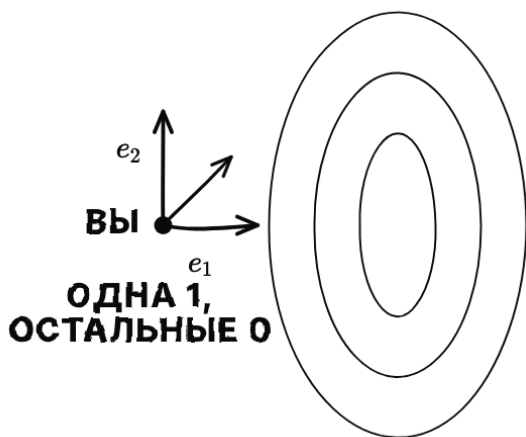
3. Примените правило произведения к x на логарифм x при x равном единице.

Подсказка: Производная первого на второе плюс первое на производную второго; логарифм единицы — ноль.

Ответ: Первое даёт один на логарифм единицы, то есть ноль; второе даёт один на единицу, делённую на один, то есть один. Наклон — ноль плюс один, один.

Наклоны во всех направлениях

Встаньте на склоне холма с картой в руке. Спросите, насколько крут склон, и честный ответ будет: зависит от того, куда вы идёте. Шагните строго на восток — и, возможно, круто подниметесь, шагните на север — и едва поднимитесь, шагните на северо-восток — и получите что-то среднее. На прямой было лишь два способа двигаться, влево и вправо, но на карте, а тем более во многих измерениях, направлений бесконечно много, так что наклон ничего не значит, пока вы не скажете, в какую сторону. Эта часть показывает, как один список чисел, градиент, отвечает на вопрос сразу для каждого направления.



$$\frac{\partial f}{\partial x_1}$$

$$\frac{\partial f}{\partial x_2}$$

$$\nabla f = \begin{pmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \end{pmatrix}$$

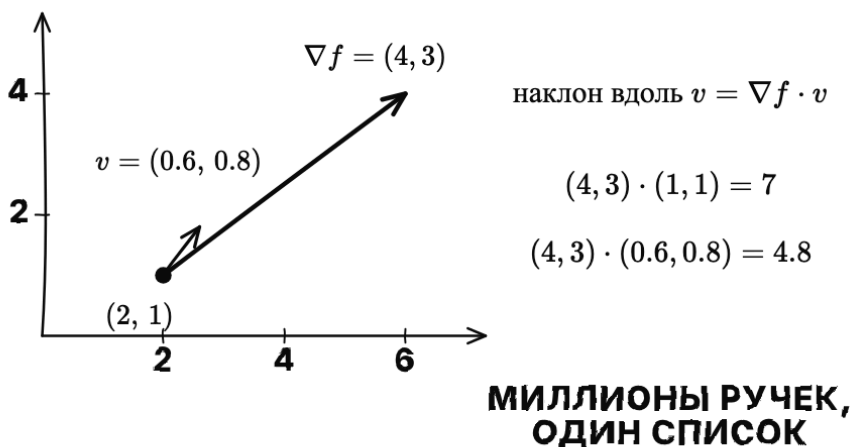
Наклону нужно направление

Возьмите функцию, превращающую вектор из d чисел в одно число, как высота земли в точке карты. Простейшие направления — оси: двигайтесь только вдоль одной оси, держа все остальные координаты фиксированными, и измерьте наклон ровно как в случае одного числа. Двигаться вдоль оси i означает прибавить малое кратное особого вектора, у которого единица в позиции i и нули в остальных, одна строка единичной таблицы, называемого базисным вектором. Полученный так наклон называется частной производной по этой координате. Сложите d частных производных в один вектор — и у вас градиент, настолько важный, что книга даёт ему собственный знак, перевернутый треугольник, называемый набла.

Вот пример. Пусть высота — первая координата в квадрате плюс три вторых координаты, и мы стоим в точке два, один. Двигаясь только вдоль первой оси, наклон — удвоенная первая координата, четыре; двигаясь только вдоль второй, наклон — три. Так что градиент там — вектор четыре, три.

$$f(x) = x_1^2 + 3x_2, \quad x = (2, 1)$$

$$\partial_1 f = 2x_1 = 4, \quad \partial_2 f = 3$$

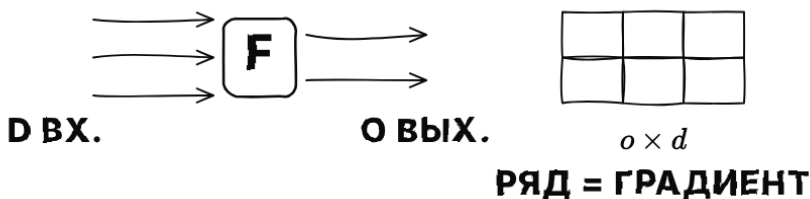


Градиент $(4, 3)$ и наклоны по направлениям

А как с направлением, которое не ось, скажем любой стрелкой v ? Наклон вдоль неё называется производной по направлению и снова определяется малым шагом, на этот раз вдоль v . Поскольку любое движение можно разложить на малые движения вдоль каждой оси, производная по направлению оказывается просто скалярным произведением градиента с направлением. В нашем примере ход вдоль один, один даёт четыре плюс три, семь; ход

вдоль стрелки шесть десятых, восемь десятых, которая имеет длину один, даёт четыре на шесть десятых плюс три на восемь десятых, две целых четыре десятых плюс две целых четыре десятых, четыре целых восемь десятых. Значит, знать градиент достаточно, чтобы знать наклон в каждом возможном направлении. Когда обучают фотоприложение, его ошибка зависит от миллионов ручек, и его градиент — один список из миллионов наклонов, который говорит, как меняется ошибка, куда бы ни двигали ручки вместе.

Теперь пусть функция возвращает несколько чисел вместо одного, скажем o выходов из d входов. У каждого выхода свой градиент, и если сложить эти градиенты как строки, получится таблица из o строк и d столбцов, называемая якобианом. При одном выходе якобиан — просто градиент, а при одном входе и одном выходе — обычная производная. Например, при двух входах, где первый выход — их произведение, а второй — их сумма, в точке два, один первая строка — один, два, а вторая строка — один, один. Якобиан наследует свойства производной, и самое важное — цепное правило: якобиан одной функции, применённой после другой, — это произведение их двух якобианов, где внешний взят в точке, которую выдавала внутренняя. Это цепочка наклонов из предыдущей части, теперь как матричное умножение, и это математическое сердце того, как обучают сети.



$$(x_1 x_2, x_1 + x_2) \text{ в } (2, 1) : J = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$$

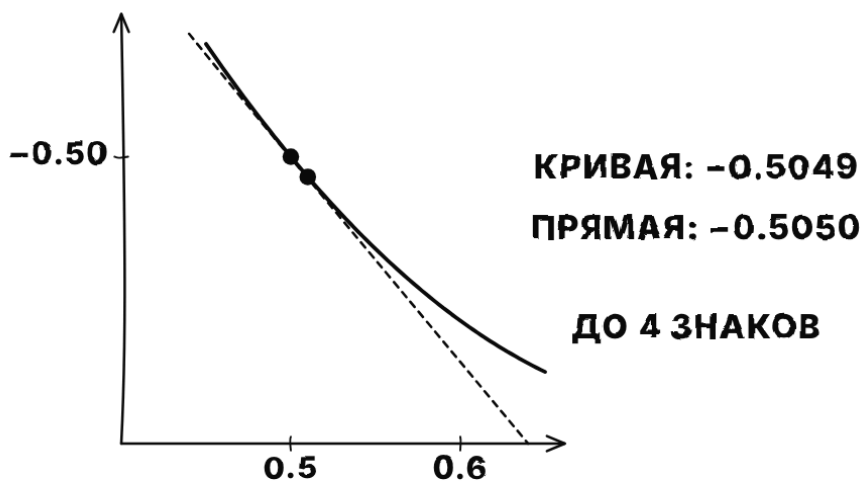
$$J_{f \circ g}(x) = J_f(g(x)) J_g(x)$$

ЦЕПНОЕ ПРАВИЛО = ПРОИЗВЕДЕНИЕ МАТРИЦ

Якобиан и цепное правило

Как и производная, градиент тоже даёт лучшее прямое приближение функции около точки — утверждение, известное как теорема Тейлора: около выбранной точки функция приблизительно равна своему значению там плюс скалярное произведение градиента со смещением от этой точки. Книга проверяет это на своей прежней кривой, $x^2 - 1,5x$, в точке одна вторая. Там значение — минус одна вторая, и наклон — минус одна вторая. При сдвиге на одну сотую вправо истинное значение — минус ноль целых пять тысяч сорок девять десятитысячных, а прямое приближение даёт минус одна вторая минус половина одной сотой, минус ноль целых пять тысяч пятьдесят десятитысячных. Они совпадают до четвёртого знака после запятой, а это и значит хорошее приближение первого порядка.

$$f(x + h) \approx f(x) + \nabla f(x) \cdot h$$



Приближение первого порядка, проверено

Запомним

Для функции многих входов наклону нужно направление; вдоль одной оси это частная производная, а набор всех частных производных — градиент. Наклон в любом направлении, производная по направлению, — это скалярное произведение градиента с этим направлением, так что один градиент даёт каждый направленный наклон: четыре, три вдоль один, один даёт семь. Для функции с несколькими выходами градиенты выходов, сложенные как строки, образуют якобиан, таблицу выходы на входы. Якобиан цепочки функций — произведение их якобианов, цепное правило в матричной форме. Около точки значение там плюс скалярное произведение градиента со смещением — лучшее прямое приближение, как проверка в одной второй показывает до четырёх знаков.

Вопросы

1. Градиент функции в точке — четыре, три. Каков её наклон вдоль направления один, один?
 - a. Семь, скалярное произведение градиента с направлением
 - b. Двенадцать, произведение четырёх и трёх
 - c. Пять, длина градиента
 - d. Один, потому что в направлении две единицы

2. Функция принимает 5 чисел и выдаёт 3 числа. Какова форма её якобиана?
- a. 5 строк на 3 столбца
 - b. 5 строк на 5 столбцов
 - c. 3 строки на 5 столбцов, по одной строке на выход
 - d. Один вектор из 15 чисел
3. Как получается якобиан одной функции, применённой после другой?
- a. Сложением двух якобианов
 - b. Выбором большего из двух
 - c. Их перемножением позиция за позицией
 - d. Перемножением двух якобианов, где внешний взят в результате внутренней функции
4. Почему знать градиент достаточно, чтобы знать наклон в любом направлении?
- a. Потому что у каждой функции один и тот же наклон во всех направлениях
 - b. Потому что любое движение раскладывается на движения вдоль осей, так что наклон — скалярное произведение градиента с направлением
 - c. Потому что градиент уже перечисляет каждое возможное направление
 - d. Потому что на практике важны только направления осей

Ответы

1. a) Производная по направлению — скалярное произведение градиента с направлением: четыре на один плюс три на один, семь.

- 2. c)** У каждого выхода есть градиент по 5 входам, и 3 градиента, сложенные как строки, дают таблицу 3 на 5.
- 3. d)** Цепное правило в матричной форме: якобиан композиции — матричное произведение отдельных якобианов.
- 4. b)** Разложение движения на его осевые части превращает производную по направлению в скалярное произведение градиента с направлением.

Разобранные примеры

Проверка книги по прямой Для кривой $x^2 - 1.5x$ используйте прямое приближение в одной второй, чтобы предсказать значение в одной второй плюс одна сотая, и сравните с истинным значением.

В одной второй значение — одна четверть минус три четверти, то есть минус одна вторая.

Наклон — два на одну вторую минус полтора, то есть минус одна вторая.

Приближение прибавляет наклон на шаг: минус одна вторая плюс минус одна вторая на одну сотую даёт минус ноль целых пятьсот пять тысячных.

Истинное значение в ноль целых пятьдесят одной сотой — ноль целых две тысячи шестьсот одна десятитысячная минус ноль целых семьсот шестьдесят пять тысячных, то есть минус ноль целых пять тысяч сорок девять десятитысячных, совпадение до четырёх знаков.

Ответ: Приближение минус ноль целых пять тысяч пятьдесят десятитысячных; истинное значение минус ноль целых пять тысяч сорок девять десятитысячных.

Задачи

1. Для функции, перемножающей два входа, найдите градиент в точке три, два и наклон вдоль направления один, минус один.

Подсказка: Каждая частная производная — это другой вход.

Ответ: Частная производная по первому входу — второй, два; по второму — первый, три. Градиент — два, три, а наклон вдоль один, минус один — два минус три, минус один.

2. У функции два выхода: удвоенный первый вход плюс второй, и первый вход минус второй. Запишите её якобиан.

Подсказка: По одной строке на выход, по одному столбцу на вход.

Ответ: Первая строка — два, один, а вторая строка — один, минус один: таблица две строки на два столбца, одинаковая в каждой точке, потому что функция линейна.

3. С помощью прямого приближения квадрата в тройке оцените три целых одну десятую в квадрате и сравните с истинным значением.

Подсказка: Значение в тройке — девять, а наклон там — шесть.

Ответ: Девять плюс шесть на одну десятую даёт девять целых шесть десятых. Истинное значение — девять целых шестьдесят одна сотая, так что приближение ошибается лишь на одну сотую для этого шага.

Где земля падает быстрее всего

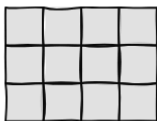
Снова на склоне холма из прошлой части, теперь в густом тумане. Вы хотите добраться до дна долины, но видите только землю у своих ног. Разумный план: нащупать, куда земля уходит вниз круче всего, шагнуть туда и повторить. Этот план, сделанный точным, и есть градиентный спуск, и эта часть готовит его в два хода: сначала внимательный взгляд на размеры якобианов, затем причина, почему самый крутой путь вниз — ровно противоположный градиенту.



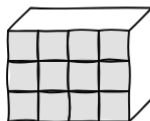
**НАЩУПАТЬ, ШАГНУТЬ,
ПОВТОРИТЬ**

$$y = Wx$$

D ВХ. О ВЫХ.



$$\partial y / \partial x = W$$



$$o \times o \times d$$



$$o \times (o d)$$

РАЗВЕРНУТЬ

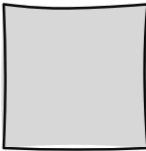
Книга начинается с того, что сама называет педантичным замечанием о размерах. Возьмите базовую операцию слоя сети: таблица весов умножает входной вектор и даёт выходной вектор, с d входами и o выходами. Как функция входа, её якобиан — просто сама таблица весов, o строк на d столбцов, и это осмысленно: каждый вес говорит, насколько сильно один выход откликается на один вход. Как функция весов, картина меняется, потому что теперь меняемая вещь сама является таблицей. Тогда у «якобиана» три направления: одно для выхода и два для строки и столбца сдвигаемого веса, o на o на d . Всегда можно развернуть таблицу весов в один длинный вектор из o умножить на d чисел, и тогда якобиан — обычная таблица из o строк и o умножить на d столбцов.

Книга разбирает это во вставке. Каждый выход — скалярное произведение одной строки весов со входом, так что сдвиг веса меняет только выход своей собственной строки, и со скоростью, равной значению входа, на которое этот вес умножается. Попробуем с двумя выходами и тремя входами, где вход — один, два, три. Первый выход откликается на три веса первой строки со скоростями один, два и три и совсем не откликается на веса второй строки. Второй выход откликается на веса второй строки с теми же скоростями, один, два, три, и не откликается на веса первой. Так что развёрнутый якобиан — таблица две строки на шесть столбцов: один, два, три, затем три нуля в первой строке; три нуля, затем один, два, три во второй. Половина его — нули, а остальное — вход, скопированный снова и снова. Хранить такую таблицу явно, материализовать её, — трата памяти на повторяющиеся значения, и шестая глава покажет, как этого

избежать, потому что на практике нам всегда нужна лишь таблица, применённая к другому тензору.

$$x = (1, 2, 3)$$

1	2	3	0	0	0
0	0	0	1	2	3



$$3200 \times 3200$$

> 10 МИЛЛИОНОВ ЭЛЕМЕНТОВ

$$32 \times 100 = 3200$$

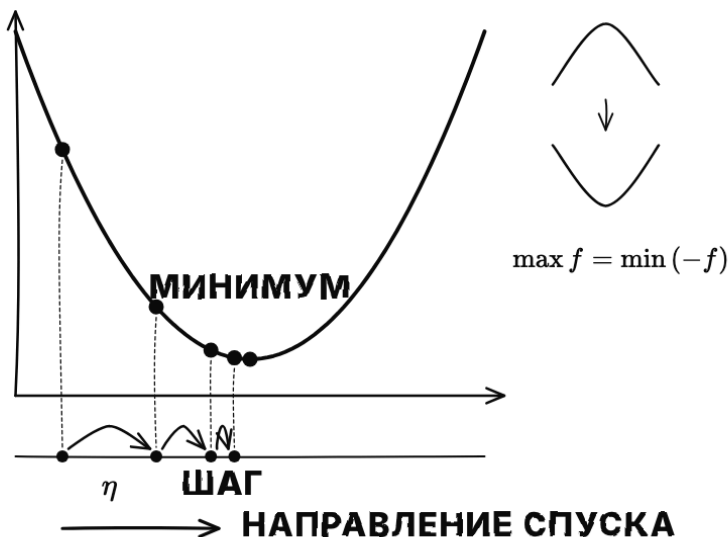
ВЕКТОР × ЯКОБИАН

В основном нули и копии

Вот предупреждение, которое стоит держать в голове. Книга часто будет записывать якобианы простыми таблицами, чтобы обозначения оставались чистыми, но их настоящие размеры скрыты в формах входов и выходов, включая размеры батчей. Для батча из тридцати двух примеров по сто чисел в каждом вход одного слоя уже содержит три тысячи двести чисел, а якобиан от такого входа к выходу того же размера был бы таблицей три тысячи двести на три тысячи двести, больше десяти миллионов элементов, для одного слоя и одного батча. Шестая глава избегает этого с помощью того, что она называет векторно-

якобианными произведениями, а книга отсылает к более формальному изложению, которое рассматривает якобианы как абстрактные линейные отображения.

Теперь к градиентному спуску. Допустим, мы хотим найти вход, при котором функция многих чисел как можно меньше, её минимум. Позже входом будут ручки модели, параметры, а функция будет измерять, насколько плохо модель справляется со своими данными; для фотоприложения — сколько картинок оно подписывает неверно. Минимизация ничего не теряет по сравнению с максимизацией: сделать функцию как можно больше — то же, что сделать её противоположность как можно меньше, точно как перевёрнутый холм превращает свою вершину в самую низкую точку долины. Лишь редко лучшую точку можно выписать напрямую формулой, как случится с методом наименьших квадратов в четвёртой главе. Обычно приходится искать шаг за шагом: начать со случайной догадки и на каждом шаге двигаться на определённую длину в определённом направлении. Длина называется размером шага или, в машинном обучении, скоростью обучения. Направление называется направлением спуска, если какой-то шаг вдоль него понижает функцию. Выбирайте каждый раз направление спуска, выбирайте размер шага с осторожностью, и процедура приближается к минимуму, в смысле, который следующая часть делает точным.

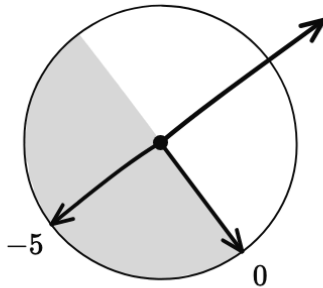


Шаг за шагом вниз

Какие направления ведут вниз? Ровно те, у которых производная по направлению отрицательна, а из прошлой части производная по направлению — это скалярное произведение градиента с направлением. Записав это скалярное произведение как длину градиента на длину направления на косинус угла между ними и взяв направления длины один, для выбора остаётся только косинус. Косинус отрицателен, когда угол лежит между четвертью оборота и тремя четвертями оборота, так что все эти направления ведут вниз. Самый отрицательный косинус, минус один, достигается на половине оборота, ровно противоположно градиенту: это направление наискорейшего спуска. С прежним градиентом четыре, три, длина которого пять, шаг в противоположном направлении, минус восемь десятых, минус

шесть десятых, даёт наклон минус пять, самый крутой из возможных; шаг под прямым углом к нему, вдоль шесть десятых, минус восемь десятых, даёт наклон ноль, ровную тропу вдоль склона.

$$\nabla f = (4, 3), \quad \|\nabla f\| = 5$$



ВНИЗ: $\cos < 0$

$(-0.8, -0.6)$: наклон -5

$(0.6, -0.8)$: наклон 0

Противоположно градиенту — круче всего

Запомним

Для слоя, умножающего вход на таблицу весов, якобиан по входу — сама таблица весов, а по весам у него три направления, или, в развёрнутом виде, o строк на o умножить на d столбцов. Этот якобиан по весам — в основном нули и копии входа, а реальные размеры, включая батчи, делают явные якобианы громадными, чего шестая глава избегает векторно-якобианными произведениями. Градиентный спуск ищет минимум шаг за шагом, двигаясь на размер шага, скорость обучения, вдоль направления спуска — такого, у которого производная по направлению отрицательна. Направления дальше четверти оборота от градиента ведут вниз, а направление, ровно противоположное градиенту, — направление наискорейшего спуска.

Вопросы

1. Для слоя, в котором таблица весов умножает вход, что такое якобиан по входу?
 - a. Сама таблица весов
 - b. Вход, скопированный o раз
 - c. Таблица нулей
 - d. Тензор с тремя направлениями

2. Почему книга предостерегает от материализации якобиана по весам?
- a. Потому что его вообще нельзя вычислить
 - b. Потому что он в основном нули и копии входа, тратит память, а реальные размеры батчей делают его громадным
 - c. Потому что он меняется каждый раз, когда меняется вход
 - d. Потому что он совпадает с градиентом
3. Направление образует с градиентом угол в четверть оборота, прямой угол. Что произойдёт, если шагнуть вдоль него?
- a. Функция падает быстрее всего
 - b. Функция растёт быстрее всего
 - c. В первом порядке функция не меняется: путь ровный
 - d. Размер шага становится нулём
4. Почему мы всегда можем говорить о минимизации, а не о максимизации?
- a. Потому что у каждой функции есть минимум, но нет максимума
 - b. Потому что для максимизации нужна формула в замкнутом виде
 - c. Потому что градиенты указывают только вниз
 - d. Потому что сделать функцию как можно больше — то же, что сделать её противоположность как можно меньше

Ответы

1. **a)** Каждый вес говорит, насколько сильно один выход откликается на один вход, так что якобиан по входу — это ровно таблица весов, d строк на d столбцов.
2. **b)** Его элементы — значения входа и нули; с батчами его размер взрывается, и шестая глава избегает его построения с помощью векторно-якобианных произведений.
3. **c)** Косинус прямого угла равен нулю, так что производная по направлению равна нулю; вниз ведут только углы больше четверти оборота.
4. **d)** Перевернутая функция превращает свои вершины в самые низкие точки, так что максимизировать f — это минимизировать минус f .

Разобранные примеры

Размер якобиана по весам У слоя 4 входа и 3 выхода. Сколько элементов у его якобиана по таблице весов и какова его форма, когда веса развёрнуты?

Таблица весов — 3 строки на 4 столбца, 12 весов.

По весам у якобиана одно направление для 3 выходов и два для весов 3 на 4: 3 на 3 на 4.

Это 3 умножить на 3 умножить на 4, то есть 36 элементов.

Разворачивание 12 весов в один вектор даёт таблицу 3 строки на 12 столбцов, всё ещё 36 элементов, большинство из них нули.

Ответ: 36 элементов; в развёрнутом виде 3 строки на 12 столбцов.

Задачи

1. Градиент в вашей текущей догадке — шесть, восемь. Каково направление наискорейшего спуска длины один и какой наклон оно даёт?

Подсказка: Длина шесть, восемь — десять; направьтесь ровно в противоположную сторону.

Ответ: Направление — минус шесть десятых, минус восемь десятых, противоположное градиенту. Его наклон — скалярное произведение: минус три целых шесть десятых минус шесть целых четыре десятых, то есть минус десять, длина градиента со знаком минус.

2. При том же градиенте шесть, восемь является ли направление ноль, один направлением спуска?

Подсказка: Вычислите скалярное произведение градиента с направлением и посмотрите на его знак.

Ответ: Нет. Скалярное произведение равно восьми, оно положительно, так что функция вдоль этого направления растёт; оно меньше чем на четверть оборота отстоит от градиента.

3. При весах в таблице из 2 строк и входе один, два, три с какой скоростью второй выход откликается на вес в строке два, столбце три, и на вес в строке один, столбце три?

Подсказка: Каждый выход зависит только от своей строки весов, со скоростью, равной значению входа, на которое вес умножается.

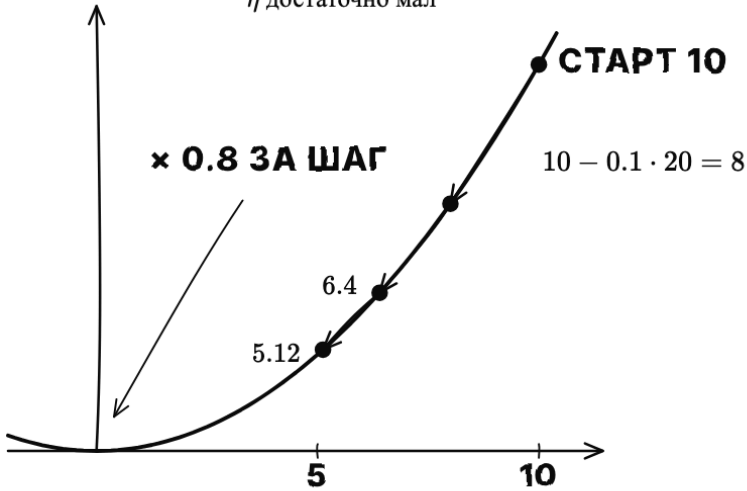
Ответ: На вес в строке два, столбце три он откликается со скоростью три, третье значение входа. На вес в строке один он не откликается вовсе: скорость равна нулю.

Спуск с горы и то, где вы оказываетесь

Запустим план туманного склона на простейшей долине из всех: высота — квадрат вашего положения, и вы начинаете с десяти. Наклон там — удвоенное положение, двадцать, так что вы шагаете против него с размером шага одна десятая: десять минус одна десятая от двадцати приводит вас к восьми. В восьми наклон — шестнадцать, и то же правило приводит вас к шести целым четырём десятым; затем к пяти целым двенадцати сотым, и так далее. Каждый шаг сохраняет четыре пятых расстояния до дна, и вы скользите к нулю, самой низкой точке. Этот простой цикл, движение против градиента снова и снова, и есть наискорейший градиентный спуск.

$$x \leftarrow x - \eta \nabla f(x)$$

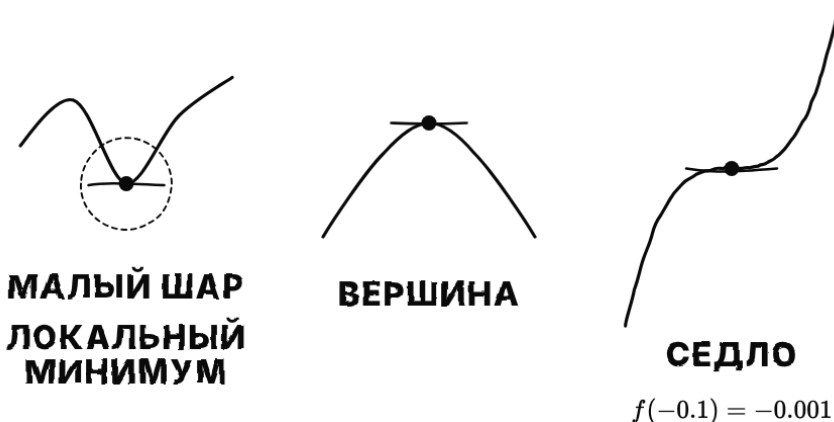
η достаточно мал



Спуск на x в квадрате

Словами правило такое: новая догадка равна старой догадке минус размер шага, умноженный на градиент в старой догадке. Книга не задерживается на том, как выбирать размер шага; она предполагает, что он достаточно мал, чтобы каждый шаг понижал функцию. Она также замечает, что градиентный спуск быстр лишь настолько, насколько быстр метод вычисления градиента, и шестая глава вводит для этого общий метод. Вопрос этой части другой: где заканчивается прогулка?

$$\nabla f = 0 : \text{стационарная}$$



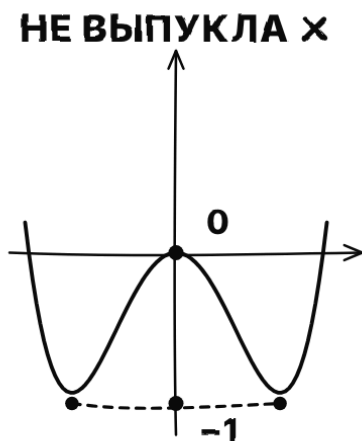
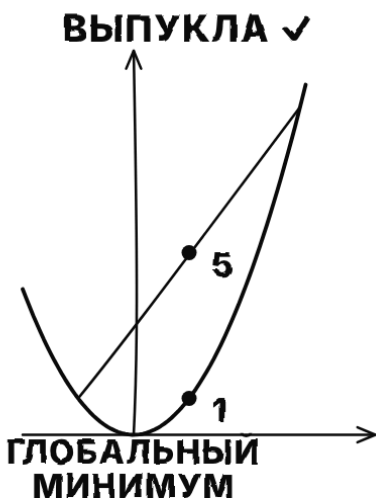
КАКАЯ — ЗАВИСИТ ОТ СТАРТА

Плоские места: минимум, вершина, седло

Книга сначала тщательно определяет свои цели, замечая, что доверчивый читатель может пропустить вперёд. Локальный минимум — это точка, значение в которой не больше значений во всех соседних точках внутри некоторого малого шара вокруг неё, как дно одной ямки в бугристом ландшафте. В такой точке касательная плоская, а вокруг неё земля поднимается. Плоскость схватывается понятием стационарной точки: точки, где градиент равен нулю, так что нет наклона ни в каком направлении. Но плоское место не всегда дно. Это может быть вершина, то есть просто дно перевернутой функции, или седловая точка, где кривая меняет направление изгиба. Рисунок в книге показывает такую точку; простой случай — куб x^3 в нуле, где наклон равен нулю, но кривая продолжает идти вниз влево, с минус одной

тысячной в минус одной десятой. В общем случае, без предположений о функции, самое большее, что можно доказать, — это что градиентный спуск достигает какой-то стационарной точки, и какой именно, зависит от того, где он начал.

Можно ли добиться большего? Представьте параболу в форме чаши: у неё нет седловых точек и только один минимум, и этот минимум — наименьшее значение на всей области входов, называемое глобальным минимумом. На параболе из любой начальной точки каждый градиент указывает назад к этому дну, так что градиентный спуск всегда его находит, как и наша прогулка от десяти. Свойство, обобщающее параболу, называется выпуклостью. Функция выпукла, если всякий раз, когда вы берёте две точки и натягиваете прямую нить между их значениями на графике, график между ними никогда не поднимается выше нити. Проверьте на квадрате: возьмите точки минус один и три, чьи значения — один и девять; посередине, в единице, нить находится на среднем, пять, а график — лишь на единице, ниже нити. Теперь попробуйте ландшафт с двумя долинами, четвёртая степень минус удвоенный квадрат: два его дна — в минус единице и плюс единице, оба со значением минус один, а посередине, в нуле, график стоит на нуле, выше нити на минус единице, так что эта функция не выпукла.



Выпукла или нет: проверка нитью

Выпуклость делит задачи оптимизации на лёгкие и трудные. Для невыпуклой функции градиентный спуск достигает стационарной точки, и ничего больше нельзя сказать, не глядя на производные производных. Для выпуклой функции он достигает глобального минимума, с какой бы точки ни начал. А если нить всегда строго выше графика, что называется строгой выпуклостью, этот глобальный минимум единственный. Найти глобальный минимум невыпуклой функции одним лишь градиентным спуском означало бы перезапускаться из каждой возможной начальной точки, бесконечно, что ставит задачу среди самых трудных, какие знает информатика, называемых NP-трудными.

У этого настоящий исторический вес. Пятая глава покажет, что каждая нетривиальная сеть невыпукла, так что при её обучении может быть много стационарных точек. Более старые методы обучения с учителем, такие как метод опорных векторов, сохраняли нелинейное поведение, при этом наслаждаясь выпуклой задачей, и годами это выглядело решающим преимуществом. Но сети, стоящие за вашим фотоприложением и чат-ассистентом, невыпуклы, и они работают. Интересно, замечает книга, что когда такие модели стартуют с разумной начальной догадки, их обучение обычно оседает в точках, которые на практике работают хорошо.

Запомним

Градиентный спуск повторяет одно правило: новая догадка равна старой минус размер шага на градиент; на квадрате от десяти с шагом одна десятая он идёт к восьми, затем шести целым четырём десятым, затем пяти целым двенадцати сотым. Локальный минимум — самый низкий в малой окрестности; стационарная точка имеет нулевой градиент и может быть минимумом, максимумом или седловой точкой, и в общем случае градиентный спуск достигает лишь какой-то стационарной точки. Глобальный минимум — самый низкий везде; для выпуклой функции, чей график никогда не поднимается выше нити между двумя её точками, градиентный спуск достигает его из любого старта, а строгая выпуклость делает его единственным. Сети невыпуклы, в отличие от метода опорных векторов, но из разумного старта их обучение достигает точек, которые хорошо работают.

Вопросы

1. Для выпуклой функции где заканчивается градиентный спуск при достаточно малом шаге?
 - a. В глобальном минимуме, с какой бы точки ни начали
 - b. В ближайшей седловой точке
 - c. В разной точке для каждого старта, без гарантий
 - d. Только в максимуме

2. Почему достичь стационарной точки — не то же самое, что достичь минимума?
- a. Потому что у стационарных точек всегда большой градиент
 - b. Потому что плоская точка может быть и вершиной, и седловой точкой
 - c. Потому что стационарные точки существуют только у выпуклых функций
 - d. Потому что у минимума должен быть отрицательный наклон
3. Посередине между точками минус один и плюс один функция с двумя долинами имеет значение ноль, а нить между её значениями находится на минус единице. Что это показывает?
- a. Функция выпукла
 - b. У функции единственный глобальный минимум
 - c. Функция не выпукла, поскольку её график поднимается выше нити
 - d. У функции нет стационарных точек
4. Почему исторически важно, что сети невыпуклы?
- a. Потому что это делает их обучение невозможным
 - b. Потому что это значит, что их градиенты всегда нулевые
 - c. Потому что выпуклые методы всегда медленнее
 - d. Потому что более старые методы вроде метода опорных векторов сохраняли выпуклую задачу, но сети всё же хорошо обучаются из разумного старта

Ответы

1. **a)** На выпуклой функции каждый путь вниз ведёт к глобальному минимуму, как на параболе в форме чаши.
2. **b)** Нулевой градиент означает плоское место, которое может быть минимумом, максимумом или седловой точкой, как куб в нуле.
3. **c)** Выпуклый график никогда не поднимается выше нити между двумя его точками; здесь он поднимается, так что функция не выпукла.
4. **d)** Невыпуклость означает много стационарных точек, предполагаемую слабость по сравнению с выпуклыми методами, но сети, разумно инициализированные, достигают точек, которые хорошо работают.

Разобранные примеры

Три шага градиентного спуска Минимизируйте квадрат икс градиентным спуском, начав с пяти, с размером шага одна четверть. Где вы после трёх шагов?

Градиент квадрата — удвоенный икс, так что каждый шаг вычитает четверть удвоенного икс, то есть половину икс.

Шаг один: из пяти вычитаем два с половиной, попадаем в два с половиной.

Шаг два: из двух с половиной вычитаем одну целую одну четверть, попадаем в одну целую одну четверть.

Шаг три: вычитаем пять восьмых, попадаем в пять восьмых, каждый раз вдвое сокращая расстояние до дна.

Ответ: Пять восьмых, приближаясь к глобальному минимуму в нуле.

Задачи

1. Найдите стационарную точку функции $x^2 - 4x$ и скажите, является ли она глобальным минимумом.

Подсказка: Приравняйте наклон, $2x - 4$, к нулю; функция — парабола в форме чаши.

Ответ: Наклон $2x - 4$ равен нулю при $x = 2$, где значение — -4 . Функция — выпуклая парабола, так что эта стационарная точка — её единственный глобальный минимум.

2. Проверьте условие выпуклости для квадрата между точками ноль и четыре в точке на четверти пути от четырёх к нулю, то есть в тройке.

Подсказка: Сравните значение графика в тройке со значением нити там, которое равно четверти значения в нуле плюс три четверти значения в четвёрке.

Ответ: График в тройке равен девяти. Нить там — четверть от нуля плюс три четверти от шестнадцати, то есть двенадцать. Девять ниже двенадцати, как и требует выпуклость.

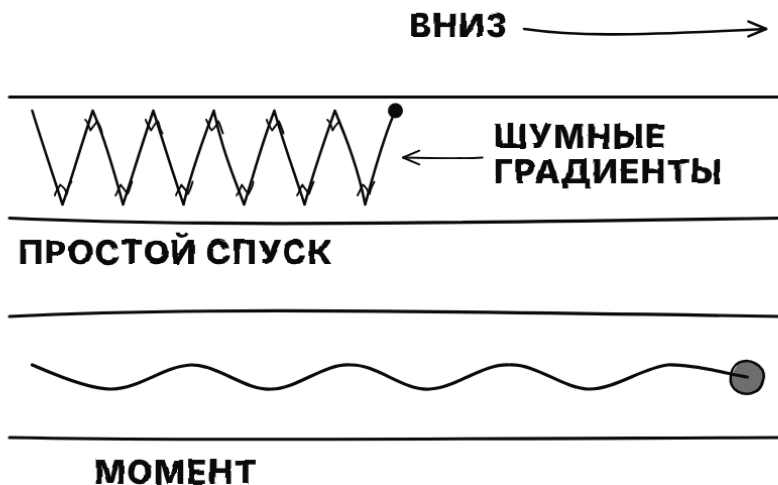
3. Запустите градиентный спуск на квадрате от пяти с размером шага один. Что происходит, и что это подсказывает о размере шага?

Подсказка: Каждый шаг вычитает единицу, умноженную на удвоенный x .

Ответ: Каждый шаг переводит x в $x - 2x$, то есть $-x$: пять, минус пять, пять и так далее, никогда не приближаясь к нулю. Шаг слишком велик; книга предполагает размер шага достаточно малым, чтобы каждый шаг понижал функцию.

Шар, который помнит

Пустите шарик по длинному узкому жёлобу, слегка наклонённому вдоль своей длины. Он не скользит прямо посередине; он мечется от одной боковой стенки к другой, медленно продвигаясь вдоль жёлоба. Теперь пустите по тому же жёлобу тяжёлый шар. Его вес несёт его вперёд: качания из стороны в сторону частично гасят друг друга, а постоянная тяга вниз накапливается, так что он спускается гораздо быстрее. Эта разница и есть идея момента.



Шарик и тяжёлый шар в жёлобе

Отрицательный градиент даёт самый крутой путь вниз, но лишь в крошечной окрестности текущей точки. Простой градиентный спуск ведёт себя как лёгкий шарик, беспорядочный путь, потому что каждый новый градиент может указывать куда-то совсем в другую сторону; книга замечает, что при идеально выбранном шаге на выпуклой функции два последовательных градиента даже стоят под прямым углом друг к другу. Пятая глава покажет, что в больших моделях эти направления ещё и очень шумные. Когда обучают чат-ассистента, градиент каждого шага приходит от маленькой горстки примеров, так что он сильно дрожит от шага к шагу. Многие техники пытаются ускорить дело, выбирая лучшие направления, и книга предпочитает те, которым не нужны производные производных, такие как таблица, называемая гессианом, и лишние вычисления функции, поскольку и то и другое дорого.

Момент — одна из таких техник. Вместо того чтобы двигаться на простой шаг вниз, каждый ход — это простой шаг вниз плюс доля предыдущего хода. Эта доля, число между нулём и единицей, — коэффициент момента, а самый первый предыдущий ход считается нулевым. Вот маленький расчёт с одной ручкой, размером шага одна десятая, градиентом, который остаётся равным единице, и коэффициентом одна вторая. Первый ход — минус одна десятая. Второй — минус одна десятая плюс половина от минус одной десятой, минус пятнадцать сотых. Третий — минус одна десятая плюс половина от этого, минус сто семьдесят пять тысячных, и ходы продолжают расти к минус двум десятым, вдвое больше простого шага, потому что постоянное направление продолжает накапливаться. Когда же градиент мечется туда-

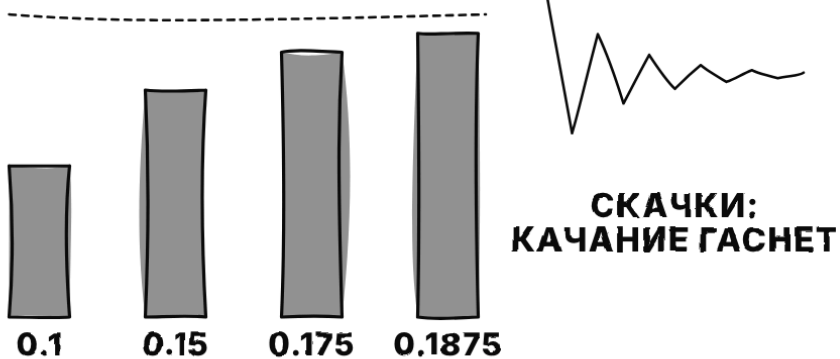
сюда, запомненная часть работает против новой, и значение уменьшается.

$$\text{ход} = -\eta \nabla f + \beta \cdot \text{предыдущий ход}$$

$$\beta \in (0, 1), \quad \text{первый предыдущий ход} = 0$$

$$\eta = 0.1, \quad \nabla f = 1, \quad \beta = 0.5$$

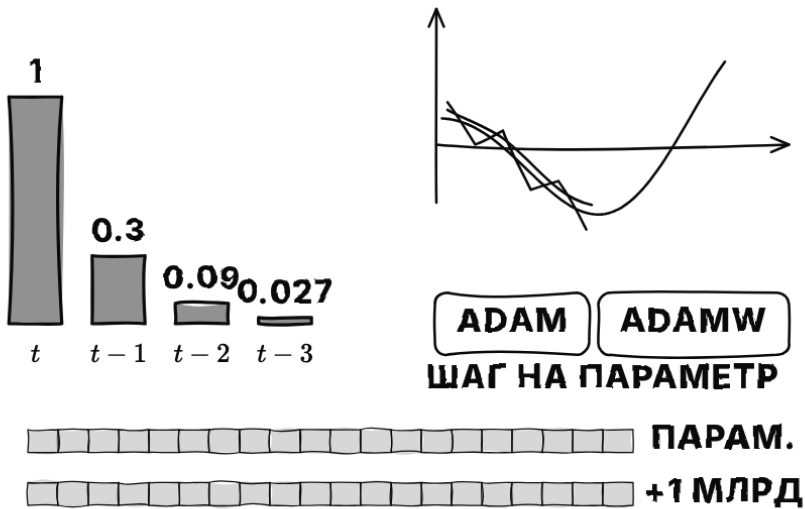
ПРЕДЕЛ 0.2 = 2 × ШАГ



Ходы, которые накапливаются

Если развернуть правило, видно, как угасает память. Новейший градиент считается полностью; предыдущий умножается на коэффициент; тот, что перед ним, — на коэффициент в квадрате, и так далее, так что градиент n шагов назад умножается на коэффициент в степени n минус один. При коэффициенте три десятых, использованном на рисунке в книге, градиент двумя шагами назад считается с весом три десятых, а тремя шагами назад — всего девять сотых. Рисунок сравнивает простой градиентный спуск и градиентный спуск с моментом на волнистой кривой, $x \cdot \sin(2x)$, начиная чуть за единицей, и путь с

моментом — более гладкий из двух. Действительно можно показать, что момент ускоряет обучение, сглаживая путь.



Угасающая память, Adam и его цена

Другой распространённый приём даёт каждому параметру собственный размер шага, подстроенный под то, насколько велики были его градиенты. Популярный метод под названием Adam сочетает несколько таких идей, и его ценят за то, что он довольно нечувствителен к собственным настройкам, гиперпараметрам, то есть числам, выбираемым пользователем, а не выучиваемым градиентным спуском; значения по умолчанию из библиотеки в большинстве случаев — хорошее начало. Adam и его варианты, такие как AdamW, остаются стандартным выбором

в глубоком обучении, а разработка оптимизатора, который явно их превосходит, всё ещё открытая исследовательская задача.

Цена ускорения — память. Момент должен хранить предыдущий ход, по одному лишнему числу на каждый параметр, что удваивает память, нужную самому оптимизатору: для модели с миллиардом параметров — ещё миллиард чисел. Однако в большинстве случаев, добавляет книга, память, нужная для вычисления градиента, значит куда больше, как покажет шестая глава.

Запомним

Отрицательный градиент — лучшее направление лишь очень локально, а в больших моделях он шумный, так что простой градиентный спуск идёт беспорядочным путём. Момент добавляет к каждому шагу долю предыдущего хода, начиная с нуля, так что постоянные направления накапливаются, а качания туда-сюда гасятся; старые градиенты угасают по степеням коэффициента. При шаге одна десятая, постоянном градиенте единица и коэффициенте одна вторая ходы растут от одной десятой к двум десятым. Adam сочетает идеи вроде момента с размерами шага для каждого параметра, устойчив к своим гиперпараметрам и является оптимизатором по умолчанию, а AdamW — среди его вариантов. Момент хранит предыдущий ход для каждого параметра, удваивая память оптимизатора, хотя вычисление градиента обычно требует больше.

Вопросы

1. Что момент добавляет к каждому шагу градиентного спуска?
 - a. Долю предыдущего хода
 - b. Второй градиент, вычисленный в случайной точке
 - c. Производную градиента, гессиан
 - d. Фиксированную величину в направлении первого шага
2. Почему момент уменьшает качание простого градиентного спуска?
 - a. Потому что он использует больший размер шага
 - b. Потому что он игнорирует градиент после первого шага
 - c. Потому что когда градиент мечется туда-сюда, запомненный ход работает против нового
 - d. Потому что он вычисляет градиент на большем числе примеров
3. При коэффициенте момента три десятых с каким весом считается градиент трёх шагов назад?
 - a. Три десятых
 - b. Вообще никак
 - c. Девять десятых
 - d. Девять сотых
4. Какова цена момента в памяти?
 - a. Он хранит предыдущий ход для каждого параметра, удваивая память оптимизатора

- b. Ему совсем не нужна память
- c. Он хранит каждый градиент с начала обучения
- d. Он хранит таблицу гессиана

Ответы

1. a) Каждый ход — это простой шаг вниз плюс коэффициент момента, умноженный на предыдущий ход, начиная с нулевого предыдущего хода.
2. c) Постоянные направления накапливаются, а чередующиеся частично гасятся запомненной частью, сглаживая путь.
3. d) Градиент n шагов назад умножается на коэффициент в степени n минус один; для трёх шагов назад это три десятых в квадрате, девять сотых.
4. a) Хранение одного предыдущего хода на параметр удваивает то, что нужно оптимизатору, хотя вычисление градиента обычно требует ещё больше памяти.

Разобранные примеры

Момент против мечущегося градиента Одна ручка, размер шага одна десятая, коэффициент момента одна вторая. Градиент равен единице, затем минус единице, затем единице. Вычислите три хода и сравните их с простым градиентным спуском.

Первый ход — минус одна десятая на единицу, то есть минус одна десятая, поскольку предыдущий ход нулевой.

Второй ход — минус одна десятая на минус единицу, плюс одна десятая, плюс половина от минус одной десятой: одна десятая минус пять сотых, то есть плюс пять сотых.

Третий ход — минус одна десятая плюс половина от пяти сотых: минус семьдесят пять тысячных.

Простой градиентный спуск двигался бы на минус одну десятую, плюс одну десятую, минус одну десятую; с моментом ходы туда-сюда меньше.

Ответ: Ходы минус одна десятая, плюс пять сотых и минус семьдесят пять тысячных: качание уменьшается.

Задачи

1. При размере шага одна десятая, постоянном градиенте единица и коэффициенте момента девять десятых к какому размеру приближаются ходы?

Подсказка: Ходы перестают расти, когда ход равен одной десятой плюс девять десятых самого себя.

Ответ: К единице: если ход размера m удовлетворяет m равно одна десятая плюс девять десятых m , то одна десятая m равна одной десятой, так что m равно единице. Ходы приближаются к десятикратному простому шагу.

2. У модели семь миллиардов параметров. Сколько лишних чисел должен хранить момент?

Подсказка: Один предыдущий ход на параметр.

Ответ: Семь миллиардов: один запомненный ход на каждый параметр, что удваивает память оптимизатора, хотя вычисление градиента обычно занимает ещё больше памяти.

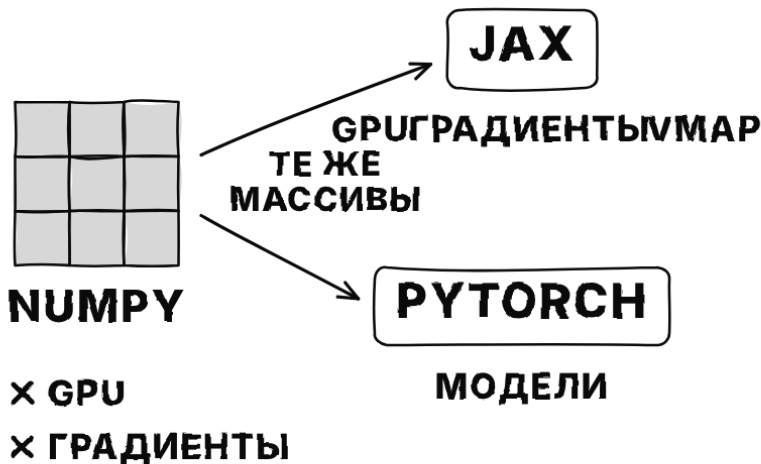
3. Объясните своими словами, почему книга предпочитает методы ускорения без вторых производных и без лишних вызовов функции.

Подсказка: Подумайте о размере таблицы вторых производных для модели с миллионами параметров.

Ответ: Для модели с миллионами параметров таблица вторых производных имела бы миллионы на миллионы элементов, а каждое лишнее вычисление функции стоит полного прохода по данным. Методы вроде момента улучшают направление, используя только уже вычисленные градиенты.

От бумаги к коду

Чтение о велосипеде никого не научило на нём ездить. Книга это знает, и в конце главы вручает вам не список учебных упражнений, а короткий путь самообучения через два настоящих программных инструмента, JAX и PyTorch, чтобы идти по нему параллельно с чтением. Решения должны быть опубликованы на сайте книги, и книга предупреждает, что многие её веб-ссылки могли переехать или устареть к тому времени, когда вы их откроете.



NumPy, затем JAX и PyTorch

Путь начинается с NumPy, классической библиотеки для работы с многомерными массивами, тензорами этой книги, и для индексации и преобразования их содержимого. Книга предлагает читать краткое руководство NumPy, пока обращение с массивами не станет естественным, особенно нарезка срезов, и тренироваться на известной, неспешной подборке из ста упражнений по NumPy.

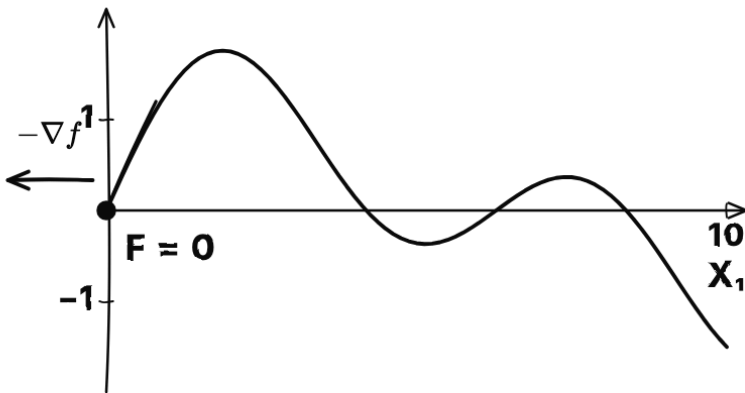
У NumPy, при всей его влиятельности, два ограничения. Сам по себе он мало использует параллельное оборудование вроде графических чипов и не умеет автоматически вычислять градиенты, тему шестой главы. JAX копирует манеру записи NumPy и добавляет широкую поддержку оборудования, автоматические градиенты и дополнительные преобразования, например такое, которое берёт функцию, написанную для одного примера, и прогоняет её по целому батчу, называемое векторизованным отображением. PyTorch тоже построен вокруг ядра в духе NumPy, с небольшими различиями в именах и поведении, плюс инструменты более высокого уровня для построения дифференцируемых моделей. Книга предлагает бегло посмотреть, как каждая библиотека создаёт массивы, чтобы увидеть, как много общего у них с NumPy, и пока не трогать высокоуровневые части для построения моделей; шестая глава объясняет, как устроены эти фреймворки. Чат-ассистент и фотоприложение, которыми вы пользуетесь, очень вероятно, обучены именно этими инструментами.

Затем идёт настоящее упражнение, которое надо сделать трижды, один раз в NumPy, один раз в JAX и один раз в PyTorch, каждый раз дело минут, когда синтаксис знаком. Исследуемая функция берёт два числа, каждое между нулём и десятью, и

возвращает синус первого на косинус второго плюс синус половины первого на косинус половины второго. Это волнистый ландшафт с несколькими холмами и впадинами. Книга добавляет примечательную сноску: эта функция была получена просьбой к ChatGPT дать красивую функцию с несколькими минимумами и максимумами, и ничто другое в книге не было сгенерировано языковой моделью, — оговорка, которую автор считает всё более важной.

$$f(x_1, x_2) = \sin x_1 \cos x_2 + \sin \frac{x_1}{2} \cos \frac{x_2}{2}$$

$$\nabla f = (1.5, 0)$$



Волнистая функция и её наклон

В начальном углу, где оба входа равны нулю, каждый синус равен нулю, и функция равна нулю. Её наклон вдоль первого входа там — косинус нуля на косинус нуля, единица, плюс одна вторая на такое же произведение, одна вторая: итого полтора. Вдоль

второго входа наклон содержит синусы нуля, так что он равен нулю. Градиент в углу, значит, полтора, ноль, и градиентный спуск шагнул бы оттуда в противоположную сторону, туда, куда функция падает.

Четыре шага упражнения таковы. Первый: напишите функцию так, чтобы она принимала целую таблицу из n точек, по одной на строку, и возвращала n значений сразу, что называется векторизованной. Второй: напишите вручную вторую функцию, возвращающую её градиент, поскольку автоматическое дифференцирование ещё не введено. Третий: запустите градиентный спуск из нескольких начальных точек и нарисуйте пути, по которым каждая идёт по ландшафту; в JAX или PyTorch здесь хорошо попробовать векторизованное отображение. Четвёртый: добавьте член момента и постройте график длины градиента по шагам: она должна уменьшаться к нулю по мере приближения прогулки к стационарной точке.

Опыт

Даже без программирования вы можете попробовать дух упражнения в электронной таблице. Поместите два входа в две ячейки, вычислите функцию в третьей, а два наклона — ещё в двух ячейках. Затем сделайте каждую новую строку равной предыдущим входам минус одна десятая наклонов, скопируйте строку вниз несколько десятков раз и смотрите, как значение падает, а наклоны уменьшаются к нулю.

Запомним

Глава заканчивается путём самообучения вместо классических упражнений, с решениями, запланированными на сайте книги. Он начинается с массивов и индексации NumPy, затем переходит к JAX, который добавляет поддержку оборудования, автоматические градиенты и векторизованное отображение, и PyTorch, который добавляет инструменты для построения моделей. Упражнение использует волнистую функцию двух входов, сгенерированную с помощью ChatGPT, единственный результат языковой модели в книге. В каждой библиотеке: вычислить функцию на многих точках сразу, закодировать её градиент вручную, нарисовать пути градиентного спуска из нескольких стартов, затем добавить момент и смотреть, как длина градиента уменьшается к нулю.

Вопросы

1. Что JAX добавляет к манере NumPy работать с массивами?
 - a. Широкую поддержку оборудования, автоматические градиенты и преобразования вроде векторизованного отображения
 - b. Новый способ записи массивов, не имеющий ничего общего с NumPy
 - c. Только более быструю печать результатов

- d. Высокоуровневое построение моделей, но без градиентов
2. Почему в упражнении градиент пока нужно писать вручную?
- a. Потому что у функции нет градиента
 - b. Потому что градиенты, написанные вручную, всегда быстрее
 - c. Потому что автоматическое дифференцирование ещё не введено; оно появится в шестой главе
 - d. Потому что NumPy запрещает автоматические градиенты
3. Что должно происходить с длиной градиента, когда градиентный спуск с моментом работает на функции из упражнения?
- a. Она должна неуклонно расти
 - b. Она должна оставаться постоянной
 - c. Она должна прыгать между нулём и единицей
 - d. Она должна уменьшаться к нулю по мере приближения прогулки к стационарной точке
4. Что значит написать функцию векторизованно?
- a. Нарисовать её стрелками
 - b. Сделать так, чтобы она принимала целую таблицу точек, по одной на строку, и возвращала все их значения сразу
 - c. Хранить её как один длинный вектор
 - d. Вычислять только её градиент

Ответы

1. **a)** JAX копирует интерфейс NumPy и добавляет поддержку оборудования, автоматическое вычисление градиентов и дополнительные преобразования вроде векторизованного отображения.
2. **c)** Упражнение идёт до автоматического дифференцирования, поэтому функции наклона прописываются вручную.
3. **d)** В стационарной точке градиент равен нулю, так что его длина падает к нулю по мере того, как прогулка успокаивается.
4. **b)** Векторизованная функция принимает n точек сразу и возвращает n значений, вместо того чтобы обрабатывать одну точку за раз.

Разобранные примеры

Один шаг по ландшафту упражнения В точке, где первый вход равен π , около трёх целых четырнадцати сотых, а второй — нулю, функция равна ноль плюс один, то есть единице, а её градиент — минус один, ноль. Сделайте один шаг градиентного спуска с размером шага одна десятая и проверьте, что функция падает.

Градиентный спуск вычитает размер шага на градиент: первый вход становится π минус одна десятая на минус единицу, π плюс одна десятая, около трёх целых двадцати четырёх сотых.

Второй вход остаётся нулём, поскольку его наклон равен нулю.

В новой точке первое слагаемое — синус примерно трёх целых двадцати четырёх сотых, грубо минус одна десятая, а второе слагаемое — синус примерно одной целой шестидесяти двух сотых, грубо ноль целых девятьсот девяносто девять тысячных.

Новое значение — около ноль целых девяти десятых, ниже начального значения единица, как и положено шагу спуска.

Ответ: Шаг переводит примерно в три целых двадцать четыре сотых, ноль, и значение падает с единицы до примерно ноль целых девяти десятых.

Задачи

1. Вычислите функцию из упражнения в начальном углу, где оба входа нули, и дайте её градиент там.

Подсказка: Каждый синус нуля — ноль, а каждый косинус нуля — единица.

Ответ: Функция равна нулю, поскольку оба слагаемых содержат синус нуля. Наклон вдоль первого входа — один плюс одна вторая, полтора; вдоль второго — ноль. Градиент — полтора, ноль.

2. Почему книга просит сделать одно и то же упражнение в трёх разных библиотеках?

Подсказка: Подумайте, что меняется и что остаётся прежним между NumPy, JAX и PyTorch.

Ответ: Три библиотеки делят ядро в духе NumPy, так что повторение короткого упражнения в каждой делает вас свободно владеющим всеми тремя и показывает, как мало между ними различий, одновременно выявляя, что добавляют JAX и PyTorch.

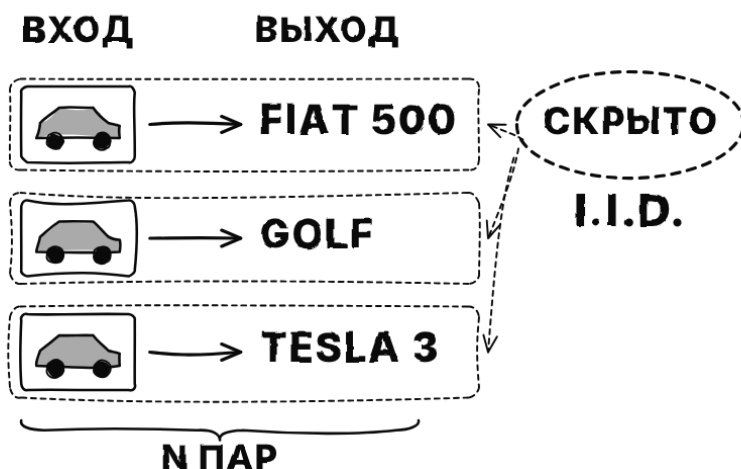
3. Опишите, как бы вы проследили градиентный спуск для этой функции в электронной таблице.

Подсказка: Каждая новая строка вычисляется из предыдущей.

Ответ: Поместите два входа в строку вместе с функцией и двумя её наклонами, написанными вручную. Сделайте входы каждой следующей строки равными предыдущим входам минус одна десятая наклонов, пересчитайте значение и наклоны и скопируйте строку вниз; значение должно падать, а наклоны уменьшаться к нулю.

Что делает кучу примеров хорошей

Допустим, вам нужно приложение, которое называет модель автомобиля по фотографии. Писать правила вручную безнадёжно: никто не опишет в коде каждый изгиб каждой машины с каждого угла. А собирать примеры легко: пройдите по улицам с камерой и подпишите модель под каждым снимком. Это ситуация, с которой начинается всё обучение с учителем, и третья глава, короткая, но названная в книге хребтом всего последующего, делает её точной. Она охватывает наборы данных, функции потерь, что значит минимизировать среднюю ошибку на примерах, предположения за всем этим и, наконец, версию на языке вероятностей, построенную на максимуме правдоподобия.

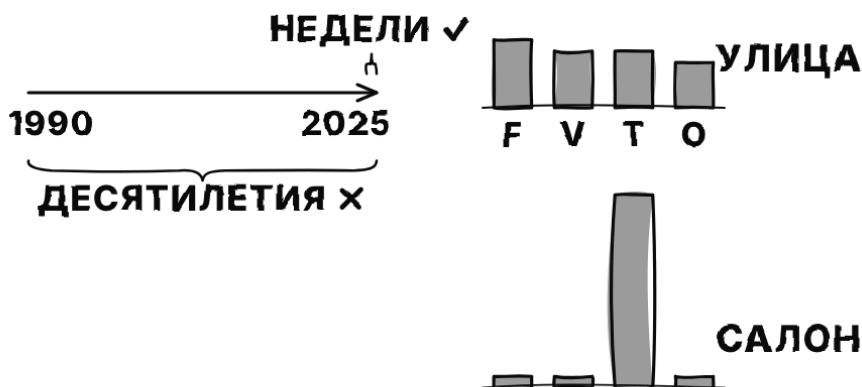


Набор данных из пар

Набор данных с учителем — это просто коллекция из n пар, каждая пара — один вход вместе с выходом, который мы для него хотим: фотография и модель автомобиля на ней. Такие задачи повсюду, от превращения речи в текст до управления роботом. Фотоприложение на вашем телефоне обучено именно на такой коллекции: миллионы картинок, каждая в паре с описанием того, что на ней. К определению прилагается одно дополнительное предположение, звучащее технически: каждая пара взята независимо из одного и того же неизвестного распределения вероятностей, и это распределение никогда нельзя узнать точно. Это называется предположением *i.i.d.*, сокращение от «независимые и одинаково распределённые».

Одинаково распределённые означает, что то, чему мы пытаемся подражать, остаётся стабильным во времени. Для приложения с автомобилями фотографии, собранные за несколько недель, этому удовлетворяют. Фотографии, собранные за последние несколько десятилетий, — нет, потому что машины на дорогах изменились: модель, обученная на них, не распознавала бы новые модели или работала бы хуже, чем должна.

ОДИНАКОВЫЕ НЕЗАВИСИМЫЕ



Стабильно во времени, честно в выборке

Независимые означает, что в коллекции нет перекоса и она честно представляет весь диапазон случаев. Если бы вы сделали все фотографии у автосалона Tesla, у вас было бы слишком много Tesla и слишком мало машин других марок. Выполняется ли предположение, зависит и от того, где модель используется.

Коллекция автомобилей, собранная в Италии, может прекрасно служить в Риме или Милане и плохо — в Токио или на Тайване, где улицы заполняют другие машины. Так что предположение нужно всегда проверять, прежде чем доверять инструментам обучения с учителем. Книга добавляет, что сегодняшние большие языковые модели обучены на таких огромных и разнообразных коллекциях, что становится трудно сказать, какие задачи внутри их обучающего распределения, а какие вне его, и как далеко они по-настоящему обобщают.

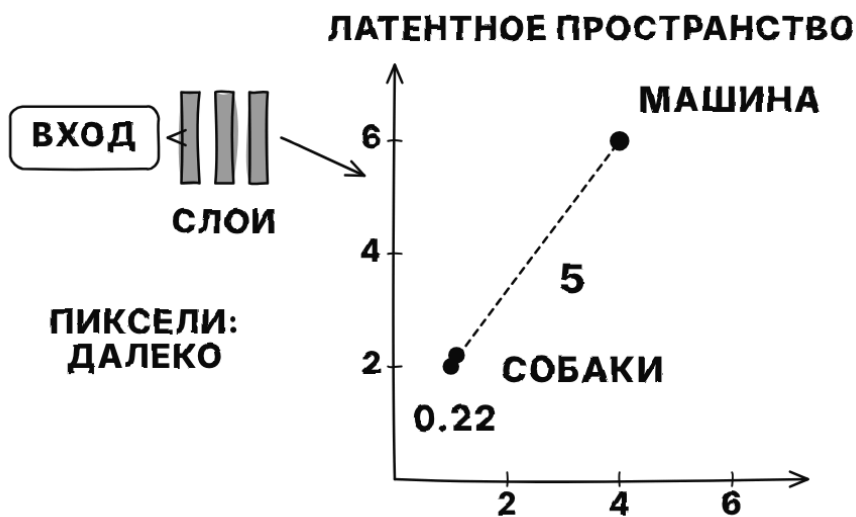


Рекомендатель кормит сам себя

Эта проверка не делается один раз и навсегда, как настаивает вставка в книге; она должна продолжаться всю жизнь модели. Если никто не следит, состав машин на дорогах медленно

меняется, и приложение тихо становится хуже — проблема, называемая сдвигом домена. Более тонкий случай — рекомендательная система, часть приложения, предлагающая, что посмотреть или купить. Как только она начинает предлагать, люди реагируют на её предложения, так что данные, на которых она учится дальше, частично сформированы её же прежними выборами. Это петля обратной связи, и она заставляет постоянно перепроверять, насколько хорошо работают система и приложение.

Открывающий рисунок главы заглядывает вперёд, к чему-то замечательному. Дифференцируемая модель преобразует свой вход шаг за шагом операциями линейной алгебры. После обучения промежуточные версии данных внутри такой модели, называемой тогда предобученной, часто имеют полезную геометрию: картинки с похожим смыслом оказываются точками, близкими друг к другу в этом скрытом, или латентном, пространстве. У сырых изображений нет разумного понятия расстояния; две фотографии одной собаки, сделанные с шага друг от друга, могут различаться почти в каждом пикселе. Перенос данных из такого пространства в пространство с осмысленными расстояниями, метрическое пространство, называется эмбедингом данных. Измерим это на игрушечном примере: если одна фотография собаки попадает в точку один, два, а другая — в одну целую одну десятую, две целых две десятых, а фотография машины — в четыре, шесть, то две собаки отстоят друг от друга примерно на две десятых, а машина — примерно на пять.



Бликие смыслы, близкие точки

Запомним

Набор данных с учителем — коллекция пар вход-выход, показывающих желаемое поведение, используемая, когда собирать примеры легко, а писать правила — нет.

Предполагается, что пары взяты независимо из одного неизменного неизвестного распределения — предположение i.i.d.. Одинаково распределённые означает, что цель стабильна во времени; независимые означает, что коллекция без перекоса и представительна, что зависит и от того, где модель будет использоваться. За предположением нужно следить всю жизнь модели из-за сдвига домена и петель обратной связи, таких как рекомендатель, реагирующий на собственные эффекты. Обученные модели часто встраивают данные в латентное пространство, где похожие смыслы лежат близко, пространство с осмысленными расстояниями.

Вопросы

1. Набор данных для распознавания автомобилей собран из фотографий за последние сорок лет. Какую часть предположения i.i.d. он нарушает?
 - а. Одинаково распределённые: машины на дорогах со временем изменились
 - б. Независимые: фотографии слишком далеки друг от друга

- c. Ни одну: старые фотографии всегда лучше
 - d. Обе, потому что фотографии не могут быть независимыми
2. Все фотографии сделаны у одного автосалона Tesla. Что не так с набором данных?
- a. Он слишком быстро меняется во времени
 - b. Он перекошен и непредставителен, что нарушает предположение независимости
 - c. В нём слишком много разных марок машин
 - d. Он слишком мал для эмбединга
3. Почему предположение i.i.d. нужно проверять всю жизнь модели?
- a. Потому что наборы данных растут автоматически
 - b. Потому что модель забывает своё обучение
 - c. Потому что данные могут дрейфовать, как при сдвиге домена, или перестраиваться самой системой, как в петлях обратной связи
 - d. Потому что предположение нужно только после обучения
4. Что такое эмбединг данных?
- a. Хранение данных внутри весов модели
 - b. Сжатие фотографии в меньшее число пикселей
 - c. Добавление меток к сырым примерам
 - d. Отображение данных в пространство с осмысленными расстояниями, где похожие объекты лежат близко

Ответы

1. **a)** Одинаково распределённые означает, что цель остаётся стабильной во времени; десятилетия меняющихся моделей машин это нарушают, так что модель провалилась бы на новых машинах.
2. **b)** Независимая выборка означает отсутствие перекоса при сборе; автосалон даёт слишком много Tesla и слишком мало других машин.
3. **c)** Медленные изменения в мире ухудшают качество, а системы вроде рекомендателей влияют на данные, на которых потом учатся.
4. **d)** У сырых изображений нет разумного расстояния; эмбеddинг переносит их в метрическое пространство, где семантически похожие объекты находятся рядом.

Разобранные примеры

Расстояния в эмбеddинге В латентном пространстве одна фотография собаки попадает в точку один, два, другая фотография собаки — в одну целую одну десятую, две целых две десятых, а фотография машины — в четыре, шесть. Какие фотографии близки?

Между двумя собаками разности — одна десятая и две десятых.

Их квадраты — одна сотая и четыре сотых, в сумме пять сотых, а квадратный корень — около двух десятых.

Между первой собакой и машиной разности — три и четыре, с квадратами девять и шестнадцать, в сумме двадцать пять.

Квадратный корень из двадцати пяти — пять, так что машина далеко от собак, а собаки близко друг к другу.

Ответ: Две собаки отстоят друг от друга примерно на две десятых; машина — на пять от первой собаки.

Задачи

1. Набор данных об автомобилях, собранный в Италии, используется для приложения в Милане и другого в Токио. Будет ли предположение i.i.d. выполняться в каждом случае?

Объясните.

Подсказка: Спросите, похожи ли машины, встречающиеся при использовании, на машины из коллекции.

Ответ: В Милане оно правдоподобно выполняется, поскольку машины на дорогах похожи на машины из итальянской коллекции. В Токио оно может нарушаться, потому что состав марок и моделей другой, так что коллекция не представляет того, что увидит приложение.

2. Объясните своими словами, как рекомендательная система может создать петлю обратной связи.

Подсказка: Откуда берутся данные, на которых она учится дальше?

Ответ: Как только рекомендатель предлагает объекты, люди нажимают частично из-за этих предложений, так что следующие данные, на которых он учится, отражают его собственные прежние выборы, а не независимые вкусы людей. Поэтому его качество и качество приложения нужно постоянно перепроверять.

3. Почему сырое изображение — плохое пространство для измерения сходства, и что вместо этого предлагает эмбединг?

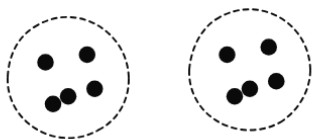
Подсказка: Подумайте о двух фотографиях одной собаки, сделанных с шага друг от друга.

Ответ: Две фотографии одной собаки могут различаться почти в каждом пикселе, так что попиксельное расстояние мало говорит о смысле. Эмбединг из обученной модели помещает похожие изображения близко друг к другу, так что расстояние в этом пространстве отражает сходство содержания.

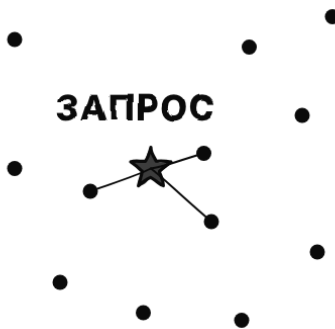
Обучение без меток и повторное использование выученного

Откройте фотоприложение и посмотрите на альбом, который собирает снимки одного и того же человека вместе, хотя вы никогда не говорили ему, кто есть кто. Никто не подписывал эти лица; приложение просто сложило похожие снимки в одну кучу. Это не обучение с учителем в строгом смысле, потому что не было целевых выходов, которым надо подражать. Большинство успешных приложений используют обучение с учителем в той или иной форме, но теперь книга обозревает его многочисленных родственников.

БЕЗ МЕТОК



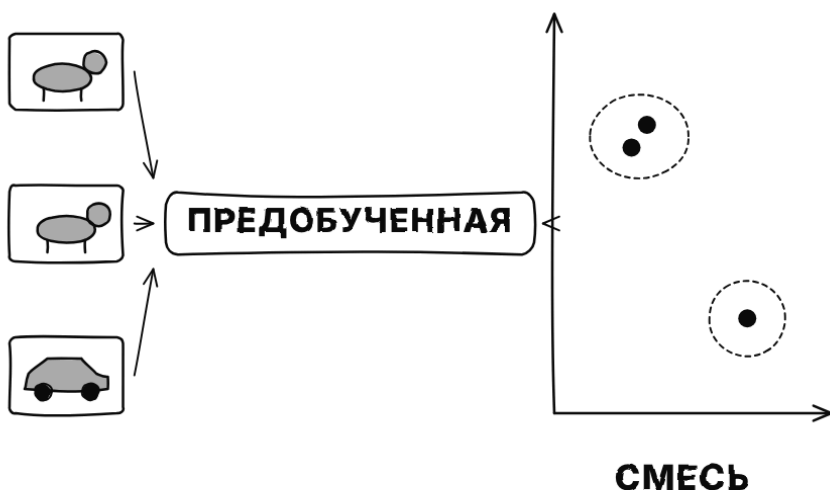
КЛАСТЕРЫ



К БЛИЖАЙШИХ

Кучи и ближайшие соседи

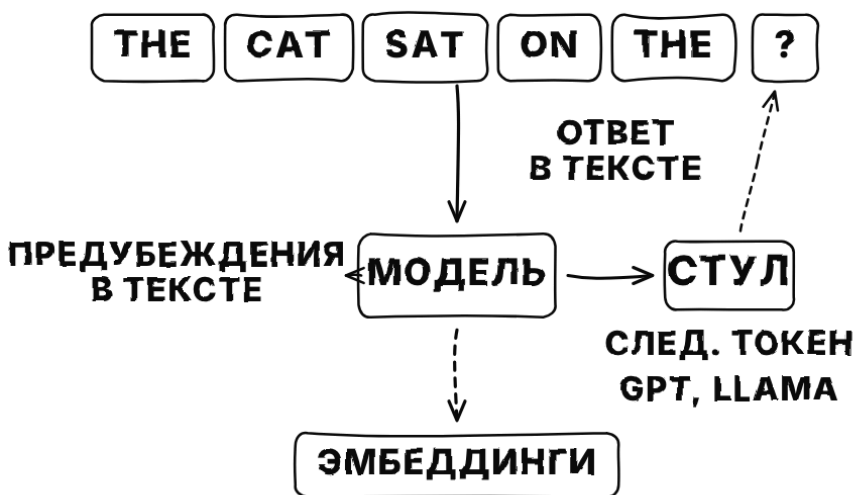
Когда в наборе данных есть входы, но нет целей, такая постановка называется обучением без учителя. Типичная задача — кластеризация: разбить данные на кучи так, чтобы объекты внутри кучи были похожи, а объекты из разных куч — нет. Другая — поиск: по запросу пользователя найти в большой коллекции k наиболее похожих на него объектов, скажем десять ближайших. С изображениями обе задачи непросты, потому что расстояние между сырыми картинками бессмысленно: крошечный сдвиг может изменить миллионы пикселей.



Встроить, затем кластеризовать

Хитрость в том, чтобы одолжить модель, уже обученную на какой-то широкой задаче, например называть предметы на фотографиях, — так называемую предобученную модель. Её внутренние состояния можно читать как векторы в пространстве многих измерений, и во многих случаях у этих векторов полезная геометрия: вещи с похожим смыслом встроены в близкие точки. Так что эти векторы можно подать обычному методу кластеризации, например смеси гауссиан, которая описывает данные как смещение нескольких колоколообразных облаков, по одному на кучу. Рисунок в книге показывает ровно это: изображения проходят через предобученную модель, и их векторы собираются в группы по смыслу.

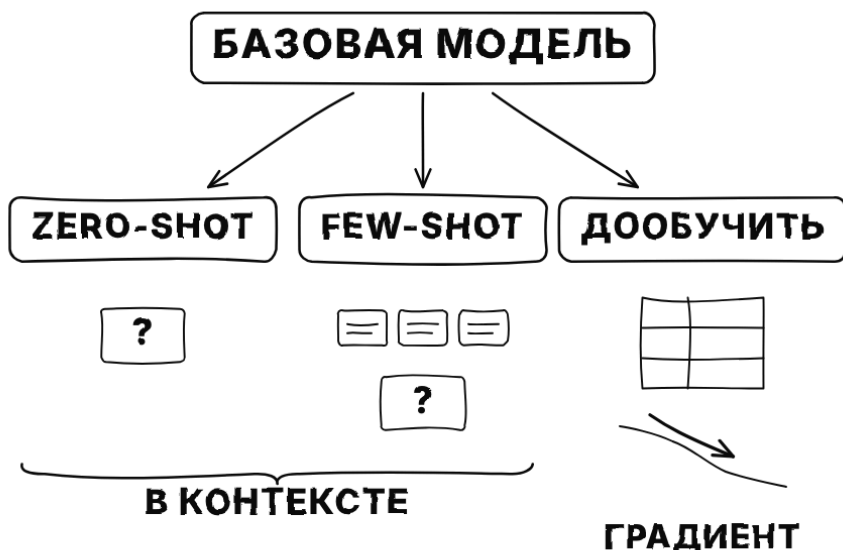
А если предобученной модели нет? Тогда очень популярный вариант, называемый самообучением, придумывает задачу с учителем из неразмеченных данных, чтобы обучить модель, которая потом может служить многим другим задачам. С большим корпусом текста, например, всегда можно обучить программу предсказывать, как продолжается короткий кусок текста, потому что истинное продолжение прямо здесь, в тексте. Открытие, что сети, обученные так, дают ещё и отличные эмбединги текста, оказало, словами книги, глубокое влияние на сообщество. Сноска добавляет отрезвляющее замечание: большие наборы данных, собранные из интернета, полны предубеждений, брани и вульгарного содержимого, модели, обученные на них, впитывают эти предубеждения, и это одна из главных претензий к фундаментальным моделям с закрытым кодом.



Предсказать, что дальше

Большие языковые модели вроде GPT и Llama — современные версии этой идеи: их обучение всегда начинается с самообучаемого предсказания следующего токена, угадывания следующего кусочка текста снова и снова. Такие модели иногда называют фундаментальными, и второй рисунок книги показывает три способа их использовать. При использовании без примеров, zero-shot, вы просто задаёте модели вопрос о новой задаче. Попросить чат-ассистента кратко изложить письмо, без единого примера, — это использование без примеров. При промптинге с несколькими примерами, few-shot, вы включаете в сам промпт несколько примеров новой задачи, например три письма с их краткими изложениями, а затем то, которое хотите изложить. И то и другое работает, только если модель обобщает

очень хорошо. Когда обучение таким образом не требуется, говорят, что модель учится в контексте. Наконец, при дообучении вы берёте предобученную модель и продолжаете обучать её параметры градиентным спуском на небольшом наборе данных для новой задачи, так же как при обучении с нуля. Книга сосредоточена на построении моделей с нуля, но замечает, что дообучение пользуется теми же средствами.



Три способа использовать модель

Дообучение облегчают большие открытые коллекции моделей в интернете, и оно не обязано трогать каждый параметр. Обучение лишь небольшого подмножества параметров или небольшого числа добавленных называется параметрически эффективным дообучением; книга оставляет его второму тому. Чтобы понять,

почему это важно, представьте, что подстраивается лишь один процент модели с семью миллиардами параметров: это семьдесят миллионов чисел для обучения вместо семи миллиардов.

Список вариантов продолжается, и книга лишь намекает на них. Если метки несёт только часть набора данных, постановка называется частичным обучением с учителем, с примерами в двенадцатой главе. С несколькими наборами данных из похожих распределений или из одного распределения в разное время появляется целое семейство задач, зависящее от порядка, в котором приходят задачи и данные: адаптация к новому домену, обучение учиться, непрерывное обучение, обучение расстояниям и даже разобучение; некоторые из них обещаны в следующем томе.

Запомним

Без целей обучение идёт без учителя: кластеризация группирует похожие объекты, а поиск находит ближайшие к запросу объекты, и обе задачи трудны на сырых изображениях. Предобученная модель встраивает данные так, что похожие смыслы лежат рядом, что позволяет обычной кластеризации, такой как смеси гауссиан, работать с её векторами. Самообучение придумывает задачу из неразмеченных данных, например предсказание продолжения текста, и так начинаются большие языковые модели вроде GPT и Llama; данные из интернета приносят в них и предубеждения. Фундаментальные модели можно использовать без примеров, с несколькими примерами в промпте или дообучать градиентным спуском, полностью или параметрически эффективно. Частичное обучение с учителем и многие другие постановки, такие как непрерывное обучение или разобучение, расширяют картину.

Вопросы

1. Почему кластеризация сырых изображений трудна без предобученной модели?
 - a. Потому что у изображений нет пикселей
 - b. Потому что расстояние между сырыми картинками бессмысленно: крошечный сдвиг меняет миллионы

пикселей

- c. Потому что кластеризация работает только с текстом
 - d. Потому что изображения нельзя хранить как векторы
2. Как самообучение получает обучающую задачу из неразмеченного текста?
- a. Просит людей разметить каждое предложение
 - b. Кластеризует слова в кучи
 - c. Предсказывает, как продолжается короткий кусок текста, поскольку истинное продолжение есть в тексте
 - d. Дообучает уже существующую модель
3. Вы даёте чат-ассистенту три письма с их краткими изложениями, а затем четвёртое письмо для изложения, без всякого обучения. Как это называется?
- a. Дообучение
 - b. Использование без примеров
 - c. Частичное обучение с учителем
 - d. Промптинг с несколькими примерами, форма обучения в контексте
4. Что отличает дообучение от использования без примеров и с несколькими примерами?
- a. Параметры модели дополнительно обучаются градиентным спуском на небольшом наборе данных
 - b. Модель вообще не получает входа
 - c. Модель обучается с нуля на интернете
 - d. Модель используется только для кластеризации

Ответы

1. **b)** Пиксельные расстояния не отражают смысл; эмбеddинг предобученной модели помещает похожие изображения рядом, так что кластеризация может работать с этими векторами.
2. **c)** Данные сами дают себе цели: настоящее продолжение каждого куска текста — это ответ, который надо предсказать.
3. **d)** Включение нескольких примеров задачи в промпт — это промптинг с несколькими примерами; параметры не меняются, так что модель учится в контексте.
4. **a)** Дообучение продолжает обучение предобученных параметров градиентным спуском, тогда как использование без примеров и с несколькими примерами оставляет их неизменными.

Разобранные примеры

Сколько экономит параметрически эффективное дообучение У модели семь миллиардов параметров. Если дообучение подстраивает лишь один процент из них, сколько чисел обучается?

Один процент — это одна сотая.

Семь миллиардов, делённые на сто, — семьдесят миллионов.

Значит, подстраиваются семьдесят миллионов чисел, а остальные шесть миллиардов девятьсот тридцать миллионов остаются как были.

Это в сто раз меньше чисел для обучения, чем при полном дообучении.

Ответ: Семьдесят миллионов параметров вместо семи миллиардов.

Задачи

1. Классифицируйте каждую ситуацию: группировка покупателей по покупательским привычкам без меток; набор данных, где размечена лишь десятая часть фотографий; обучение модели заполнять скрытые слова в предложениях.

Подсказка: Без учителя, частично с учителем или самообучение.

Ответ: Группировка без меток — обучение без учителя, конкретно кластеризация. Частично размеченный набор данных — частичное обучение с учителем. Заполнение скрытых слов придумывает цели из самих данных, а это самообучение.

2. Почему книга связывает самообучение с большими языковыми моделями, такими как GPT и Llama?

Подсказка: С чего начинается их обучение?

Ответ: Их обучение всегда начинается с самообучаемого предсказания следующего токена на огромных объёмах текста, что даёт им общие способности и хорошие эмбединги; затем их используют без примеров, с несколькими примерами или после дообучения.

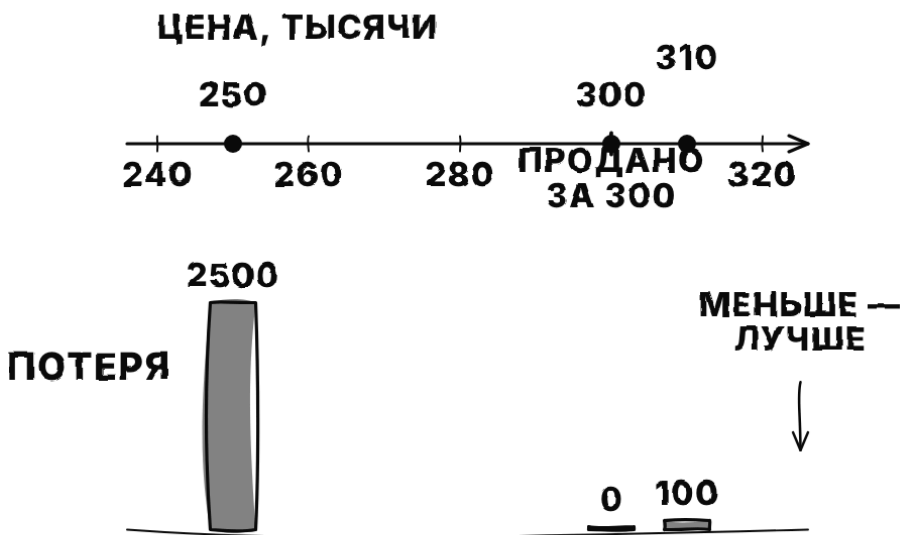
3. Какую критику книга высказывает о моделях, обученных на больших наборах данных из интернета?

Подсказка: Подумайте, что содержат такие наборы данных.

Ответ: Данные из интернета полны предубеждений, брани и вульгарного содержимого, и модели, обученные на них, впитывают эти предубеждения; это одна из главных претензий к фундаментальным моделям с закрытым кодом.

Оценка предсказания

Попросите трёх друзей угадать цену квартиры, которая на самом деле продана за триста тысяч. Один говорит триста десять, другой — двести пятьдесят, третий — ровно триста. Кто справился лучше, очевидно, но чтобы компьютер мог автоматически улучшать свои догадки, «лучше» должно стать числом, которое уменьшается, когда догадки становятся лучше. Это число — потеря, и когда данные собраны, её выбор — это способ точно сказать, что мы имеем в виду под подражанием желаемому поведению.



Три догадки, три потери

Функция потерь берёт цель и предсказание модели и возвращает одно число, и она должна быть дифференцируемой. Её значение отслеживает качество: меньшая потеря для одного предсказания, чем для другого, означает, что первое предсказание лучше относительно этой цели. Один распространённый выбор, например, — квадрат разности: догадки выше получают десять в квадрате, сто; пятьдесят в квадрате, две тысячи пятьсот; и ноль. Потеря несёт наше понимание задачи и наши предпочтения среди решений, записанные на шкале вещественных чисел, с которой может работать оптимизатор. Поскольку она дифференцируема, обучение становится математической задачей оптимизации, которую может решить градиентный спуск.

Естественная цель — наименьшая возможная средняя потеря по всему набору данных. Возьмите каждый пример, сравните предсказание модели с целью, вычислите потерю, сложите все n потерь и разделите на n ; затем ищите модель, которая делает это среднее как можно меньше. С нашими тремя догадками как тремя предсказаниями на трёх примерах средняя квадратичная потеря была бы сто плюс две тысячи пятьсот плюс ноль, делённое на три, около восьмисот шестидесяти семи. По историческим причинам эта цель называется минимизацией эмпирического риска, где риск — просто другое слово для потери, а эмпирический означает «измеренный на примерах, которые у нас реально есть».

$$\frac{100 + 2500 + 0}{3} \approx 867$$

ЭМПИРИЧЕСКИЙ РИСК

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i, \theta))$$

θ ПАРАМЕТРЫ

$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta)$$

ГРАДИЕНТНЫЙ СПУСК

Усредняем потери

Записанный так, поиск, кажется, идёт по всем возможным функциям входа. На практике, объясняет книга, модель всегда описывается набором тензоров, её параметрами, так что модель становится функцией и входа, и параметров. Обучение тогда означает поиск значений параметров, минимизирующих среднюю потерю, и поиск ведётся градиентным спуском по параметрам. Обучение фотоприложения — ровно это: миллионы подписанных картинок, потеря, которая мала, когда приложение правильно называет картинку, и градиентный спуск, крутящий параметры, чтобы уменьшить среднее.

Затем книга подчёркивает, почему потеря должна быть дифференцируемой. Рассмотрите различение кошек и собак с

целями плюс один для кошки и минус один для собаки и модель, которая выдаёт любое вещественное число, чей знак — её решение. Самая естественная оценка — потеря ноль-один: ноль, когда знак верный, единица, когда неверный. Она идеально совпадает с нашим представлением о правоте, но бесполезна для обучения, потому что её наклон почти везде равен нулю: сдвиг выхода с трёх десятых на тридцать одну сотую ничего не меняет, так что градиентный спуск никогда не сдвинется с места. Менее интуитивная, но лучшая величина — зазор, цель, умноженная на предсказание, положительный, когда знак верный, и отрицательный, когда неверный, и меняющийся плавно. Построенная на нём потеря — это шарнирная потеря: ноль, как только зазор достигает единицы, иначе единица минус зазор. Для кошки предсказание два имеет зазор два и ничего не стоит; предсказание три десятых верно, но робко и стоит семь десятых; предсказание минус одна вторая неверно и стоит полтора. И теперь маленький сдвиг с трёх десятых на тридцать одну сотую снижает цену до шестидесяти девяти сотых, что градиентный спуск может почувствовать. Шарнирная потеря — та, которой обучают метод опорных векторов, и пример показывает встроенное напряжение между потерями, выражающими то, что мы хотим, и потерями, которыми могут пользоваться оптимизаторы.



Потеря ноль-один против шарнирной

Вставка даёт предысторию: минимизация эмпирического и ожидаемого риска связана с российским учёным-информатиком Владимиром Вапником, чьи работы основали статистическую теорию обучения. Эта теория изучает, насколько хорошо средняя потеря на конечном наборе примеров замещает истинное среднее по всем возможным данным, для ограниченных видов моделей. Современные сети, которые продолжают хорошо обобщать долго после того, как от них ожидали бы переобучения, открыли для неё много новых вопросов.

Запомним

Функция потерь сравнивает предсказание с его целью и возвращает одно дифференцируемое число, меньшее для лучших предсказаний; она кодирует, чего мы хотим от модели. Обучение минимизирует среднюю потерю по набору данных, что называется минимизацией эмпирического риска, поиском по параметрам модели градиентным спуском. Потеря ноль-один совпадает с «быть правым», но имеет нулевой наклон почти везде, так что градиентный спуск не может ей пользоваться. Зазор, цель на предсказание, меняется плавно, и построенная на нём шарнирная потеря, используемая для метода опорных векторов, даёт градиентному спуску, за чем следовать. Минимизация риска идёт из статистической теории обучения Вапника, которой удивительное обобщение современных сетей продолжает бросать вызов.

Вопросы

1. Почему потеря ноль-один бесполезна для градиентного спуска?
 - a. Её наклон почти везде равен нулю, так что малые сдвиги её не меняют и параметры никогда не двигаются
 - b. Она всегда равна единице
 - c. Её нельзя усреднить по набору данных
 - d. Она даёт большие значения лучшим предсказаниям

2. Что минимизирует минимизация эмпирического риска?
- a. Наибольшую потерю на каком-либо одном примере
 - b. Потерю на одном самом трудном примере
 - c. Среднюю потерю по примерам набора данных
 - d. Число параметров модели
3. Для кошки (цель плюс один) модель выдаёт три десятых. Какова её шарнирная потеря?
- a. Ноль, потому что знак верный
 - b. Одна целая три десятых
 - c. Три десятых
 - d. Семь десятых, потому что зазор меньше единицы
4. О чём говорит зазор, цель, умноженная на предсказание?
- a. О размере набора данных
 - b. О том, верен ли знак (положительный) или неверен (отрицательный), меняясь плавно
 - c. О числе нужных параметров
 - d. О точной вероятности быть правым

Ответы

1. **a)** Потеря ноль-один скачет только тогда, когда меняется знак предсказания; в остальных местах малое изменение не имеет эффекта, так что градиентный спуск застревает на старте.
2. **c)** Она усредняет потерю по всем n примерам и ищет параметры, делающие это среднее наименьшим.
3. **d)** Зазор — единица на три десятых; шарнирная потеря — единица минус зазор, пока зазор меньше единицы, то есть семь десятых.

4. б) Зазор положителен, когда знак предсказания совпадает с целью, и отрицателен иначе, и, в отличие от потери ноль-один, меняется непрерывно.

Разобранные примеры

Средняя шарнирная потеря для трёх фотографий собак

У трёх фотографий собак цель минус один. Модель выдаёт минус два, минус четыре десятых и одну вторую. Вычислите шарнирные потери, их среднее и сравните с потерей ноль-один.

Зазоры — цель на выход: два, четыре десятых и минус одна вторая.

Шарнирные потери: первый зазор не меньше единицы, так что ноль; второй — единица минус четыре десятых, шесть десятых; третий — единица минус минус одна вторая, полтора.

Средняя шарнирная потеря — ноль плюс шесть десятых плюс полтора, две целых одна десятая, делённое на три, семь десятых.

Потери ноль-один — ноль, ноль и один, среднее одна треть; она не замечает, что второе предсказание было верным, но робким.

Ответ: Шарнирные потери ноль, шесть десятых и полтора, среднее семь десятых; среднее ноль-один — одна треть.

Задачи

1. Квартира продана за триста тысяч. Используя квадрат разности, измеренный в тысячах, какая догадка имеет меньшую потерю, двести девяносто или триста двадцать, и на сколько?

Подсказка: Возведите в квадрат разности десять и двадцать.

Ответ: Двести девяносто стоит десять в квадрате, сто; триста двадцать стоит двадцать в квадрате, четыреста. Первая догадка лучше, с потерей на триста меньше.

2. Для кошки выход модели сдвигается с одной второй на шесть десятых. Как меняются потеря ноль-один и шарнирная потеря?

Подсказка: У обоих выходов верный знак; вычислите единицу минус зазор для шарнирной потери.

Ответ: Потеря ноль-один остаётся нулём, поскольку знак уже был верным. Шарнирная потеря падает с одной второй до четырёх десятых, так что градиентный спуск получает сигнал двигаться дальше.

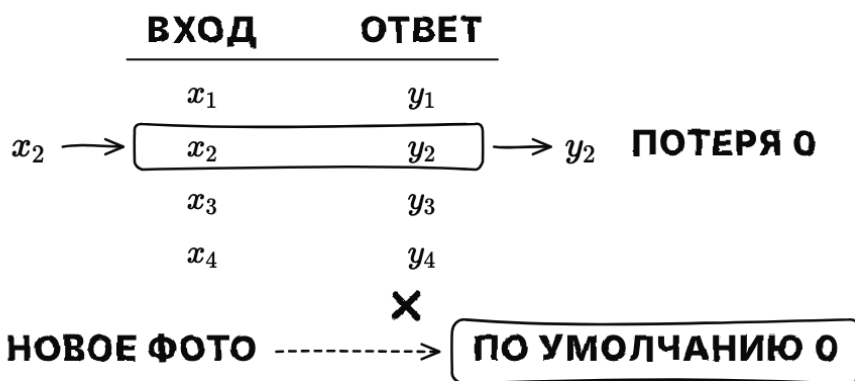
3. Объясните своими словами напряжение, которое книга описывает при выборе потери.

Подсказка: Сравните то, что мы хотим измерить, с тем, чем может пользоваться оптимизатор.

Ответ: Потеря должна выражать то, что нас по-настоящему волнует, например правоту, но она также должна быть дифференцируемой и информативной для градиентного спуска. Самая естественная мера, потеря ноль-один, не даёт наклона, так что мы используем заменители, такие как шарнирная потеря, которые отслеживают качество, меняясь плавно.

Запоминание — не обучение

Вот модель, которая никогда не ошибается на своих обучающих данных. Храните весь набор данных в таблице; когда приходит вход, который есть в таблице, возвращайте сохранённый ответ; когда приходит что-то другое, возвращайте какое-то фиксированное значение по умолчанию, скажем ноль. На обучающих примерах её предсказания в точности равны целям, так что, пока потеря наименьшая при равенстве предсказания и цели, её средняя потеря на наборе данных — наименьшая из возможных. И она бесполезна: покажите ей одну новую фотографию, и она ответит ноль. Книга использует эту справочную таблицу, чтобы подчеркнуть мысль, проходящую через всё машинное обучение: минимизация потери на примерах — не настоящая цель.

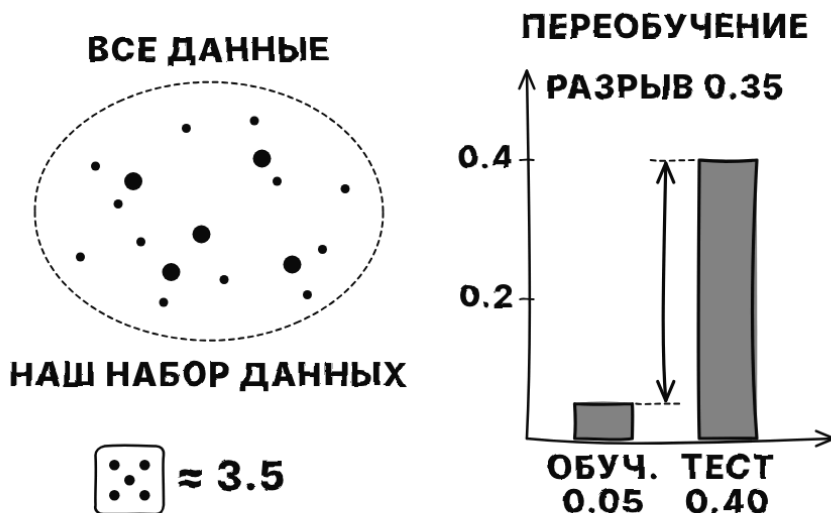


ЗАПОМНИТЬ ≠ ВЫУЧИТЬ

Справочная таблица ничему не учится

Чего мы по-настоящему хотим — малой потери на будущих входах, которых мы ещё не видели; обучающие примеры лишь их заместители. Книга формализует это как ожидаемый риск: среднюю потерю по всем парам вход-выход, которые модель могла бы когда-либо встретить, взвешенным по тому, насколько вероятна каждая, например по каждому письму, которое вообще может быть написано. Модель с низким ожидаемым риском гарантированно работала бы. Но его нельзя вычислить: мы никогда не сможем перечислить и разметить каждый возможный вход. Эмпирический риск, среднее по нашему набору данных, — это его оценка, то, что математики называют приближением Монте-Карло: среднее по случайным выборкам, замещающее среднее по всему. Это работает как бросание кубика много раз:

среднее ваших бросков приближается к трём с половиной, истинному среднему, без перечисления всех возможных исходов.

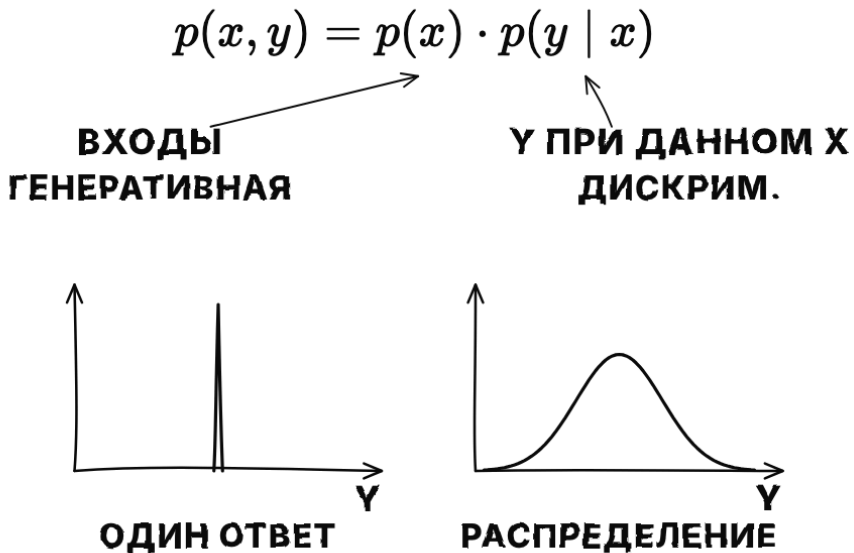


Разрыв обобщения

Разность между ожидаемым риском и эмпирическим риском называется разрывом обобщения. У чистого запоминателя, как справочная таблица, разрыв огромен: он подстраивается под свои конкретные обучающие данные и проваливается за их пределами, что и означает переобучение. На практике разрыв измеряют отдельным тестовым набором — примерами, отложенными и никогда не используемыми при обучении. Если средняя потеря — пять сотых на обучающем наборе и сорок сотых на тестовом, разрыв в тридцать пять сотых — грубая мера того, насколько модель переобучена. Прежде чем выпустить новую

версию фотоприложения, его создатели проверяют его на картинках, которых оно никогда не видело при обучении, именно по этой причине.

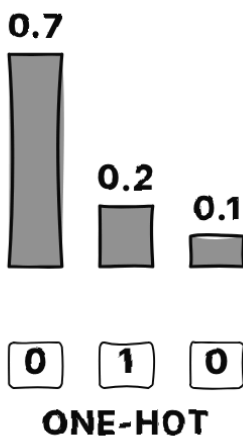
Как же выбирать саму потерю? Книга предлагает это как удачный момент прочитать её приложение о вероятности, особенно о распределениях, достаточных статистиках и максимуме правдоподобия. Потеря кодирует то, что мы заранее знаем о задаче, и сильно влияет на качество. Иногда достаточно здравого смысла, как с шарнирной потерей в прошлой части. Но есть более принципиальный путь, переписывающий всё обучение на языке вероятности, — точка зрения, которую многие книги предпочитают, и теперь книга изложит её основную идею.



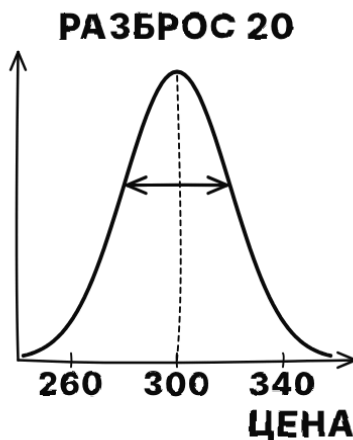
Расщепляем совместное распределение

Мы предположили, что примеры приходят из совместного распределения входов и выходов. По правилу произведения вероятностей оно расщепляется на две части: насколько вероятен каждый вход и насколько вероятен каждый выход, когда вход известен, — условное распределение выхода при данном входе. Сноска добавляет, что расщепить можно и наоборот; методы, моделирующие входы, называются генеративными, а те, что моделируют только выход при данном входе, — дискриминативными, и, не считая языковых моделей, книга сосредоточена на втором виде. Использовать модель, возвращающую одно предсказание для каждого входа, имеет смысл, когда для каждого входа почти вся вероятность сидит на одном выходе — пике, который математики называют дельтой Дирака. Это всё упрощает, но это можно ослабить.

Вместо того чтобы выдавать само предсказание, модель может выдавать числа, описывающие целое распределение по возможным выходам, его достаточные статистики. Возьмите классификацию с тремя классами. Модель возвращает три вероятности, дающие в сумме единицу, вектор в симплексе, встреченном раньше, скажем семь десятых, две десятых и одна десятая. Истинный класс записывается как вектор one-hot: все нули, кроме единицы в позиции класса. Вероятность, которую модель даёт истинному классу, — это тогда произведение каждой предсказанной вероятности в степени соответствующего элемента вектора one-hot, что просто выбирает предсказанную вероятность истинного класса. Если истинный класс второй, вектор one-hot — ноль, один, ноль, и произведение — две десятых в степени один, умноженные на остальные в степени ноль, то есть две десятых.



$$0.7^0 \cdot 0.2^1 \cdot 0.1^0 = 0.2$$



Предсказываем распределение

Для предсказания одного числа, называемого регрессией, модель может вместо этого вернуть два числа: центр и разброс колоколообразной кривой, гауссианы, над возможными выходами. Для примера с ценой квартиры модель могла бы сказать: около трёхсот тысяч, плюс-минус двадцать тысяч. Так модель выражает не только догадку, но и то, насколько она уверена, и следующая часть показывает, как такие вероятностные предсказания принципиальным образом приводят к потерям.

Запомним

Справочная таблица обучающих данных достигает наименьшего возможного эмпирического риска и бесполезна, что отделяет запоминание от обучения.

Настоящая цель — ожидаемый риск, средняя потеря по всем возможным данным, которую нельзя вычислить; эмпирический риск — её оценка Монте-Карло. Разность между ними — разрыв обобщения; переобучение означает большой разрыв, измеряемый на практике отдельным тестовым набором, никогда не используемым в обучении. В вероятностном взгляде совместное распределение расщепляется на распределение входов и условное распределение выходов при данных входах, которое и учат дискриминативные модели. Модель может выдавать описание распределения, например вероятности классов, оцениваемые относительно цели one-hot, или центр и разброс гауссианы для регрессии.

Вопросы

1. Почему модель-справочная таблица бесполезна, хотя её эмпирический риск — наименьший из возможных?
 - а. Потому что она лишь воспроизводит сохранённые обучающие ответы и возвращает значение по умолчанию для каждого нового входа

- b. Потому что её потеря на обучающих данных очень велика
 - c. Потому что она не может хранить весь набор данных
 - d. Потому что её градиент слишком велик
2. Почему ожидаемый риск нельзя вычислить напрямую?
- a. Потому что функция потерь не дифференцируема
 - b. Потому что потребовалось бы перечислить и разметить каждый возможный вход
 - c. Потому что он всегда равен нулю
 - d. Потому что тестовый набор слишком мал
3. Средняя потеря модели — пять сотых на обучающих данных и сорок сотых на отложенном тестовом наборе. О чём это говорит?
- a. Модель обобщает идеально
 - b. Тестовый набор использовался для обучения
 - c. Модель переобучена: разрыв обобщения около тридцати пяти сотых
 - d. Функция потерь неверна
4. Модель даёт вероятности семь десятых, две десятых и одна десятая трём классам, и истинный класс — третий. Какую вероятность она даёт истинному классу?
- a. Семь десятых
 - b. Две десятых
 - c. Единицу
 - d. Одну десятую

Ответы

1. **a)** Она идеально запоминает обучающие пары, но ничего не выучила о новых входах, так что её ожидаемый риск, настоящая цель, плох.
2. **b)** Ожидаемый риск усредняет по всем возможным парам вход-выход, которые никогда нельзя перечислить и разметить; вместо этого его оценивает эмпирический риск.
3. **c)** Разность между тестовой и обучающей потерей приближает разрыв обобщения; большой разрыв сигнализирует о переобучении.
4. **d)** С вектором one-hot ноль, ноль, один произведение выбирает предсказанную вероятность третьего класса, одну десятую.

Разобранные примеры

Выбираем истинный класс Модель предсказывает вероятности классов одна пятая, три пятых и одна пятая для фотографии, истинный класс которой — второй. Запишите вектор one-hot и вычислите вероятность, которую модель назначает истинному классу.

Истинный класс — второй, так что вектор one-hot — ноль, один, ноль.

Возведите каждую предсказанную вероятность в степень соответствующего элемента: одна пятая в степени ноль — единица, три пятых в степени один — три пятых, одна пятая в степени ноль — единица.

Перемножьте три результата: один на три пятых на один.

Произведение — три пятых, просто предсказанная вероятность истинного класса.

Ответ: Три пятых.

Задачи

1. Объясните разницу между запоминанием и обучением на примере фотоприложения.

Подсказка: Подумайте, что происходит с фотографией, которую приложение никогда не видело.

Ответ: Запоминающее приложение вспоминало бы метки своих обучающих фотографий и проваливалось бы на любой новой. Обучающееся приложение схватывает закономерности за фотографиями и правильно подписывает новые; поэтому его создатели проверяют его на фотографиях, никогда не использованных в обучении.

2. Почему среднее по набору данных — разумная оценка ожидаемого риска, и когда она подвела бы?

Подсказка: Вспомните сравнение с бросанием кубика и предположение $i.i.d.$

Ответ: Как усреднение многих бросков кубика оценивает три с половиной, усреднение потери по примерам, взятым из истинного распределения, оценивает среднее по всем данным. Это подводит, когда примеры непредставительны, например когда нарушено предположение $i.i.d.$, или когда модель просто их запомнила.

3. Для предсказания цены квартиры модель выдаёт центр и разброс. Что выражает «триста тысяч плюс-минус двадцать тысяч», чего не выражает одно число?

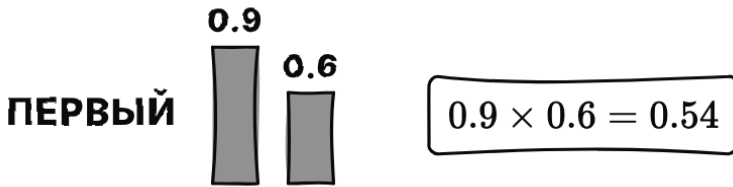
Подсказка: Подумайте о колоколообразной кривой.

Ответ: Это описывает гауссово распределение возможных цен: наиболее вероятное значение и то, насколько модель не уверена. Одно число даёт догадку, но не уверенность модели в ней.

Самое вероятное объяснение и много объяснений сразу

Два синоптика неделю предсказывали дождь. В дни, когда шёл дождь, первый давал дождю шансы девять из десяти и шесть из десяти; второй оба раза говорил «одна вторая». Какому синоптику вы доверяете больше? Перемножьте шансы, которые каждый дал тому, что действительно произошло: девять десятых на шесть десятых — пятьдесят четыре сотых у первого, одна вторая на одну вторую — четверть у второго. Первый синоптик сделал произошедшее гораздо более вероятным на вид, и эта простая идея, называемая максимумом правдоподобия, — то, как обучают вероятностные модели.

$$\text{разброс} = s^2$$



МАКСИМУМ ПРАВДОПОДОБИЯ

Два синоптика, два произведения

Сначала деталь, оставшаяся с прошлой части: когда модель предсказывает колоколообразную кривую, её второй выход возводится в квадрат, чтобы получить разброс, так что разброс никогда не станет отрицательным. Эта вероятностная постановка очень общая и включает всё сказанное о потерях до сих пор. Она даёт проектировщику больше свободы, поскольку бывает легче выбрать форму распределения выходов при данном входе, чем изобрести функцию потерь напрямую, и она естественно выражает неуверенность через разброс.

Как обучают такую модель? Поскольку примеры предполагаются независимыми и взятыми из одного распределения, вероятность, которую модель назначает всему набору данных, — это

произведение вероятностей, которые она назначает истинному выходу каждого примера. Это произведение называется правдоподобием набора данных. Случайно выбранная модель раскидывает свои вероятности более или менее случайно и даёт нашему конкретному набору данных малое правдоподобие. Так что разумный выбор — модель, которая делает правдоподобие как можно большим: решение максимального правдоподобия. Отличие от прежнего в том, что цель обучения теперь следует из правил вероятности, как только выбраны распределения, а не проектируется вручную. Для фотоприложения это означает предпочитать настройки, при которых истинные метки его обучающих картинок выглядят как можно более вероятными.

$$\text{правдоподобие} = p_1 p_2 \cdots p_n$$

СДЕЛАТЬ НАИБОЛЬШИМ

$$0.9^{1000} \approx 1.7 \times 10^{-46}$$

↓

$$-\log \text{правдоподобие} = \sum_{i=1}^n -\log p_i$$

МИНИМИЗИРОВАТЬ = ПОТЕРЯ

$$-\log 0.9 \approx 0.105, \quad -\log 0.6 \approx 0.511$$

$$0.105 + 0.511 \approx 0.616 = -\log 0.54$$

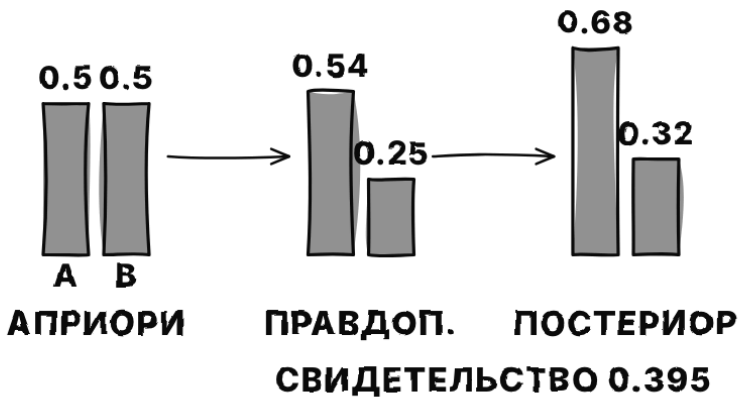
От произведения к сумме логарифмов

Две точки зрения тесно связаны. Произведения многих вероятностей становятся абсурдно малыми: у тысячи примеров, каждому из которых дано девять шансов из десяти, совместное правдоподобие — около двух на сто тысяч миллиардов миллиардов миллиардов миллиардов, десятичная точка и за ней сорок пять нулей. Взятие логарифма превращает произведение в сумму, а смена знака превращает максимизацию в минимизацию. Так что максимизировать правдоподобие — то же самое, что минимизировать сумму по примерам минус логарифма вероятности, данной каждому истинному выходу. У первого синоптика минус логарифм девяти десятых — около одной десятой, а минус логарифм шести десятых — около половины, в сумме около шести десятых, что ровно минус логарифм пятидесяти четырёх сотых. Этот член с минус логарифмом играет роль потери, и книга объявляет, что каждую потерю, используемую на практике, можно вывести таким способом из конкретного выбора распределения. Она призывает читателя держать в голове обе точки зрения.

Раздел три точка три делает ещё один шаг, к байесовскому обучению и байесовским нейронным сетям, которые книга лишь набрасывает, отсылая за глубиной к туториалам. Вероятностная модель уже может допускать несколько правдоподобных выходов для одного входа. Но обучение всё ещё выбирает одну-единственную функцию, тогда как несколько довольно разных настроек модели могли бы объяснять данные одинаково хорошо, и сохранение всех могло бы дать более верные предсказания. Поэтому мы задаём распределение вероятностей на самих функциях. Мы начинаем с априорного распределения, выражающего, какие функции мы предпочитаем до того, как

увидели какие-либо данные; поскольку модель описана своими параметрами, априорное — на самом деле распределение на весах. Например, функции с меньшей нормой часто предпочитают, потому что они устойчивее, так что априорное может дать им больше вероятности.

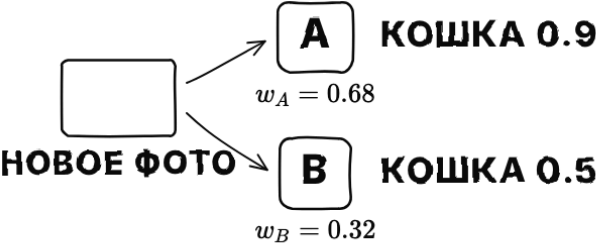
$$\text{постериор} = \frac{\text{правдоп.} \times \text{априори}}{\text{свидет.}}$$



Априорное на правдоподобие

Когда набор данных увиден, теорема Байеса обновляет априорное: апостериорное, распределение после данных, — это правдоподобие на априорное, делённое на нормирующее число, называемое свидетельством, которое заставляет всё давать в сумме единицу. Вот крошечный случай всего с двумя моделями-кандидатами, каждая с априорной вероятностью одна вторая, и данными, правдоподобие которых — пятьдесят четыре сотых при

первой и четверть при второй. Правдоподобие на апериорное даёт двадцать семь сотых и двенадцать с половиной сотых; их сумма, около сорока сотых, — свидетельство; доля, апостериорное даёт первой модели около шестидесяти восьми из ста, а второй — около тридцати двух. Чтобы предсказывать, мы усредняем предсказания всех моделей, каждое с весом её апостериорной вероятности. Если первая модель даёт новой фотографии шанс девять десятых быть кошкой, а вторая — одна вторая, усреднённое предсказание — около семидесяти семи из ста. С настоящими моделями кандидатов бесконечно много, и среднее приближают, вытягивая горстку моделей из апостериорного, — снова среднее Монте-Карло.



$$0.68 \cdot 0.9 + 0.32 \cdot 0.5 \approx 0.77$$



Среднее по моделям

Подвох, признаёт книга, в том, что апостериорное почти никогда нельзя вычислить точно, кроме очень особых выборов априорного и правдоподобия. Его нужно приближать, либо методом Монте-Карло по марковским цепям, который берёт выборки долгим случайным блужданием, либо вариационным выводом, который подгоняет вместо него более простое распределение; раздел девять точка три покажет один практический пример, называемый Monte Carlo dropout.

Запомним

Вероятностную модель обучают максимумом правдоподобия: выбирают модель, дающую наблюдаемому набору данных наибольшую вероятность, произведение вероятностей каждого истинного выхода. Взятие логарифмов превращает произведение в сумму, так что максимизация правдоподобия равна минимизации суммы минус логарифмов вероятностей, которая действует как потеря; все практические потери возникают так.

Байесовское обучение держит много моделей сразу: априорное на моделях, обновляемое теоремой Байеса в апостериорное, правдоподобие на априорное, делённое на свидетельство. Предсказания усредняют все модели с весами их апостериорных вероятностей, на практике через выборку нескольких; с двумя моделями с весами шестьдесят восемь и тридцать два из ста шансы кошки девять десятых и одна вторая усредняются примерно до семидесяти семи из ста. Апостериорное редко вычислимо точно, поэтому его приближают методом Монте-Карло по марковским цепям или вариационным выводом.

Вопросы

1. Что выбирает максимум правдоподобия?
 - а. Модель, при которой наблюдаемый набор данных наиболее вероятен

- b. Модель с наименьшим числом параметров
 - c. Модель с наибольшей априорной вероятностью
 - d. Модель, предсказывающую одну вторую для всего
2. Почему перед оптимизацией правдоподобия берут логарифмы?
- a. Потому что вероятности могут быть отрицательными
 - b. Потому что логарифмы делают модель байесовской
 - c. Потому что произведение многих вероятностей становится крошечным, а логарифм превращает его в удобную сумму
 - d. Потому что логарифмы избавляют от потребности в данных
3. Что такое апостериорное распределение в теореме Байеса?
- a. Распределение на моделях до данных
 - b. Вероятность набора данных при одной модели
 - c. Фиксированная функция потерь
 - d. Распределение на моделях после данных, правдоподобие на априорное, делённое на свидетельство
4. Почему байесовское обучение трудно на практике?
- a. Потому что априорные распределения нельзя задать
 - b. Потому что апостериорное почти никогда нельзя вычислить точно и его приходится приближать
 - c. Потому что оно использует только одну модель
 - d. Потому что оно не может делать предсказания

Ответы

1. а) Правдоподобие — произведение вероятностей, данных каждому истинному выходу; максимум правдоподобия выбирает модель, делающую это произведение наибольшим.

2. с) Произведение по тысячам примеров исчезающе мало; его логарифм — сумма, и минимизировать минус эту сумму — то же, что максимизировать правдоподобие.

3. d) Апостериорное обновляет априорное правдоподобием наблюдаемых данных, нормированным свидетельством.

4. b) Кроме особых выборов априорного и правдоподобия, у апостериорного нет замкнутой формы, поэтому используют метод Монте-Карло по марковским цепям или вариационный вывод.

Разобранные примеры

Обновляем две модели теоремой Байеса У двух моделей-кандидатов априорные вероятности одна пятая и четыре пятых. Правдоподобие данных — пятьдесят четыре сотых при первой и одна четверть при второй. Найдите апостериорную вероятность каждой.

Умножьте правдоподобие на априорное: пятьдесят четыре сотых на одну пятую — около одиннадцати сотых, точнее сто восемь тысячных; одна четверть на четыре пятых — две десятых.

Сложите их, чтобы получить свидетельство: сто восемь тысячных плюс двести тысячных — триста восемь тысячных.

Разделите каждое произведение на свидетельство: сто восемь на триста восемь — около тридцати пяти из ста.

Вторая получает двести на триста восемь, около шестидесяти пяти из ста.

Ответ: Апостериорная вероятность около тридцати пяти из ста для первой модели и шестьдесят пять для второй.

Задачи

1. Модель даёт истинным выходам трёх примеров вероятности восемь десятых, одна вторая и девять десятых. Вычислите правдоподобие и сумму минус натуральных логарифмов.

Подсказка: Перемножьте для правдоподобия; минус натуральные логарифмы — около двадцати двух сотых, шестидесяти девяти сотых и одиннадцати сотых.

Ответ: Правдоподобие — восемь десятых на одну вторую на девять десятых, тридцать шесть сотых. Сумма минус логарифмов — около двадцати двух плюс шестидесяти девяти плюс одиннадцати сотых, около одной целой двух сотых, что равно минус логарифму тридцати шести сотых.

2. У двух моделей апостериорные вероятности семь десятых и три десятых. Для новой фотографии первая говорит «кошка» с вероятностью восемь десятых, а вторая — с вероятностью четыре десятых. Каково усреднённое предсказание?

Подсказка: Взвесьте каждое предсказание его апостериорной вероятностью и сложите.

Ответ: Семь десятых на восемь десятых — пятьдесят шесть сотых, а три десятых на четыре десятых — двенадцать сотых; вместе шестьдесят восемь сотых.

3. Почему априорное распределение может предпочитать функции с меньшей нормой?

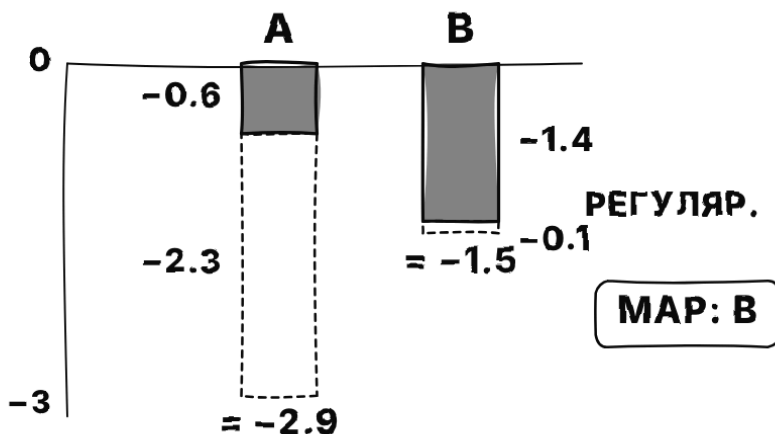
Подсказка: Подумайте, как сильно такая функция реагирует на малые изменения.

Ответ: Функции с меньшей нормой обычно устойчивее, они реагируют менее бурно на малые изменения входа, так что до данных разумно считать их более правдоподобными.

Штраф от априорного и простейший блок

Допустим, вы должны выбрать всего одну модель, а не целую толпу. Байесовское мышление всё равно помогает: выберите единственную модель с наибольшей апостериорной вероятностью. Поскольку свидетельство одно и то же для каждой модели, его можно игнорировать, а взятие логарифмов расщепляет цель на две складываемые части: логарифм правдоподобия, насколько хорошо модель объясняет данные, и логарифм априорного, насколько правдоподобной модель казалась заранее. Это называется решением максимума апостериорной вероятности.

$$\log p(\theta | D) = \log p(D | \theta) + \log p(\theta) + c$$

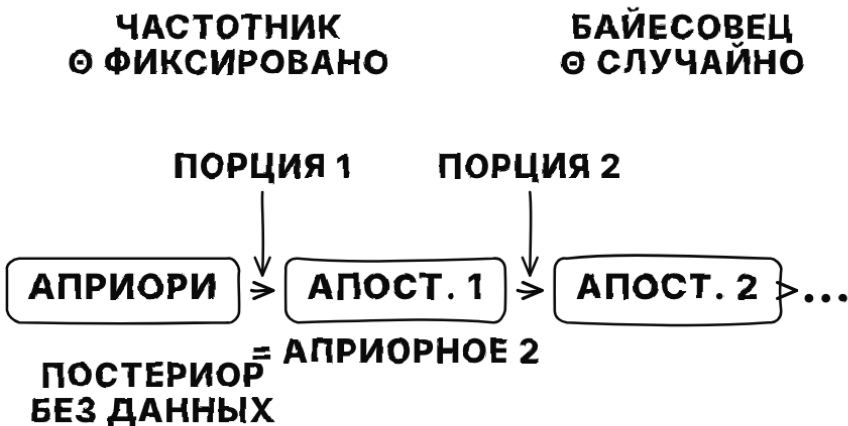


Априорное может перевернуть выбор

Вот как априорное может изменить вердикт. Скажем, модель А объясняет данные с правдоподобием пятьдесят четыре сотых, логарифм около минус шести десятых, а модель В — с четвертью, логарифм около минус одной целой четырёх десятых. Максимум правдоподобия выбирает А. Но если априорное давало А лишь один шанс из десяти, логарифм около минус двух целых трёх десятых, а В — девять из десяти, логарифм около минус одной десятой, итоги становятся около минус двух целых девяти десятых для А и минус одной целой пяти десятых для В, и выбор максимума апостериорной вероятности — В. Если бы у каждой модели было одно и то же априорное, вторая часть была бы константой, и мы вернулись бы к простому максимуму правдоподобия. В общем же случае априорное штрафует модели,

слишком далеко уходящие от того, чего мы ожидали, и книга называет это регуляризатором: он тянет решение к области, которую предпочитает априорное, — полезное оружие против переобучения и способ наложить ограничения.

Сноска слабо связывает это со старым спором о том, что означает вероятность. Для частотников вероятность — это то, как часто происходят события, и максимум правдоподобия считает параметры фиксированными неизвестными, а данные — случайными. Для байесовцев вероятность измеряет неуверенность, и данные фиксированы, а параметры случайны.



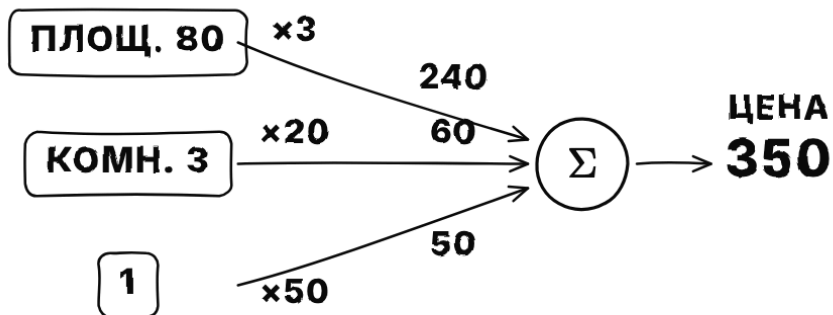
ОНЛАЙН-ОБУЧЕНИЕ БЕЗ ЗАБЫВАНИЯ

Сегодняшнее апостериорное — завтрашнее априорное

У полного байесовского подхода есть второе преимущество: новые данные легко вливаются. Когда приходит новая порция данных из того же распределения, апостериорное, вычисленное на первой порции, просто становится априорным для второй. Сноска в книге формулирует это изящно: исходное априорное — просто апостериорное после наблюдения ничего. Представьте, что фотоприложение учится на ваших новых фотографиях каждую неделю: такой перенос апостериорного вперёд помогает ему учить новые лица, не забывая старых. Обучение по мере поступления данных называется онлайн-обучением, и этот подход может облегчить его самую известную проблему, катастрофическое забывание, когда модель теряет старые знания, учась новому.

На этом начинается четвёртая глава, о линейных моделях. Программирование, напоминает книга, — это выбор хорошей последовательности базовых операций; построение модели — выбор хорошей последовательности дифференцируемых блоков. Простейший блок — линейная модель, где каждый вход добавляет к выходу свой собственный взвешенный вклад. Для квартиры линейная модель могла бы сказать: цена равна три на площадь плюс двадцать на число комнат плюс пятьдесят; квартира в восемьдесят квадратных метров с тремя комнатами тогда получает двести сорок плюс шестьдесят плюс пятьдесят, триста пятьдесят. В каком-то смысле, говорит книга, каждая дифференцируемая модель — хитрая вариация или комбинация таких линейных блоков.

□ → □ → □ БЛОКИ



ЛИНЕЙНАЯ МОДЕЛЬ

Простейший блок: линейный

Запомним

Решение максимума апостериорной вероятности выбирает единственную модель с наибольшей апостериорной вероятностью: логарифм правдоподобия плюс логарифм априорного, свидетельство не имеет значения. При равномерном априорном оно сводится к максимуму правдоподобия; иначе логарифм априорного действует как регуляризатор, штрафует маловероятные модели и борясь с переобучением, и он может опрокинуть выбор правдоподобия. Максимум правдоподобия соответствует частотному взгляду, с фиксированными параметрами и случайными данными; байесовский взгляд считает параметры случайными. В байесовском подходе апостериорное становится следующим априорным, когда приходят новые данные, что помогает онлайн-обучению против катастрофического забывания. Четвёртая глава начинается с линейных моделей, где входы добавляют взвешенные вклады, — базового блока всех дифференцируемых моделей.

Вопросы

1. Когда решение максимума апостериорной вероятности совпадает с максимумом правдоподобия?
 - а. Когда у каждой модели одно и то же априорное, так что логарифм априорного — константа

- b. Когда набор данных пуст
 - c. Когда правдоподобие равно единице
 - d. Никогда
2. Почему книга называет логарифм априорного регуляризатором?
- a. Потому что он увеличивает правдоподобие
 - b. Потому что он штрафует модели, уходящие от того, что предпочитает априорное, притягивая решение к ним
 - c. Потому что он избавляет от необходимости в градиентном спуске
 - d. Потому что он нормирует апостериорное
3. Как байесовский подход впитывает новую порцию данных?
- a. Отбрасывая старое апостериорное
 - b. Переобучаясь от исходного априорного только на новой порции
 - c. Используя апостериорное первой порции как априорное для второй
 - d. Усредняя правдоподобия двух наборов данных
4. Линейная модель оценивает квартиру как три на площадь плюс двадцать на комнаты плюс пятьдесят. Что она даёт для шестидесяти квадратных метров и двух комнат?
- a. Триста пятьдесят
 - b. Двести тридцать
 - c. Двести семьдесят
 - d. Двести семьдесят минус пятьдесят

Ответы

1. **a)** При равномерном априорном второй член добавляет каждой модели одну и ту же величину, так что решает только правдоподобие.
2. **b)** Добавление логарифма априорного штрафует маловероятные модели, что помогает против переобучения и налагает ограничения.
3. **c)** Апостериорное после первой порции становится исходным убеждением, обновляемым новыми данными, что помогает против катастрофического забывания.
4. **c)** Три на шестьдесят — сто восемьдесят, двадцать на два — сорок, плюс пятьдесят даёт двести семьдесят.

Разобранные примеры

Когда априорное опрокидывает правдоподобие У модели А логарифм правдоподобия около минус шести десятых, у модели В — около минус одной целой четырёх десятых. Их априорные вероятности — один из десяти и девять из десяти, с логарифмами около минус двух целых трёх десятых и минус одной десятой. Какую модель выбирает максимум правдоподобия и какую — максимум апостериорной вероятности?

Максимум правдоподобия сравнивает только логарифмы правдоподобия: минус шесть десятых побеждает минус одну целую четыре десятых, так что он выбирает А.

Максимум апостериорной вероятности добавляет логарифм априорного: для А минус шесть десятых плюс минус две целых три десятых — около минус двух целых девяти десятых.

Для В минус одна целая четыре десятых плюс минус одна десятая — около минус одной целой пяти десятых.

Большой итог — минус одна целая пять десятых, так что максимум апостериорной вероятности выбирает В.

Ответ: Максимум правдоподобия выбирает А; максимум апостериорной вероятности выбирает В.

Задачи

1. Объясните своими словами разницу между частотным и байесовским взглядом, как её набрасывает книга.

Подсказка: Что считается случайным: параметры или данные?

Ответ: В частотном взгляде вероятность — частота событий; параметры фиксированы, но неизвестны, а данные — случайная выборка. В байесовском взгляде вероятность измеряет неуверенность; данные фиксированы, а параметры — случайные величины с распределением.

2. Что такое катастрофическое забывание, и почему перенос апостериорного вперёд может помочь?

Подсказка: Подумайте о модели, обучаемой неделя за неделей на новых данных.

Ответ: Это когда модель, обученная на новых данных, теряет то, что выучила на старых. Использование прежнего апостериорного как нового априорного сохраняет старые знания как исходное убеждение, которое новые данные лишь подправляют, а не перезаписывают.

3. Почему книга говорит, что все дифференцируемые модели в каком-то смысле — вариации линейных блоков?

Подсказка: Сравните построение модели с написанием программы.

Ответ: Как программы — последовательности базовых операций, так и модели — последовательности дифференцируемых блоков, а самый базовый блок — линейный, где входы добавляют взвешенные

вклады; более мощные модели возникают из хитрых вариаций и композиций таких блоков.