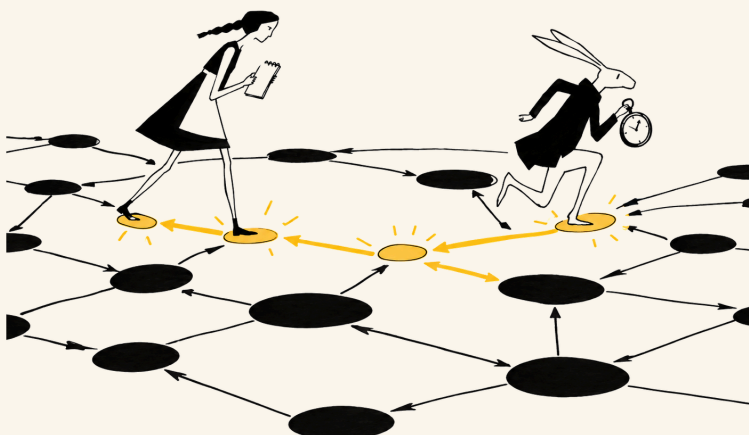


Приключения Алисы в Стране дифференцируемых чудес



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

Зачем заглядывать под капот

Обучение как поиск

Старые идеи, новая мощь

Коробки чисел и их формы

Чем эта книга является, а чем нет

Нарезаем тензор ломтиками

Стрелки, длины и совпадение

Матрицы как батчи и как функции

Как растёт стоимость

Много произведений сразу и короткая запись для всех них

Наклон в точке

Наклоны во многих направлениях

Скрытые размеры и первый шаг вниз по склону

Наискорейший спуск и где он заканчивается

Чаши, бутристые ландшафты и толчок импульса

Цена памяти и где практиковаться

Учебный ландшафт и что такое набор данных

Обучение без меток и повторное использование выученного

Оценка ошибок: функция потерь

Потери, за которыми можно идти, и модели, которые только запоминают

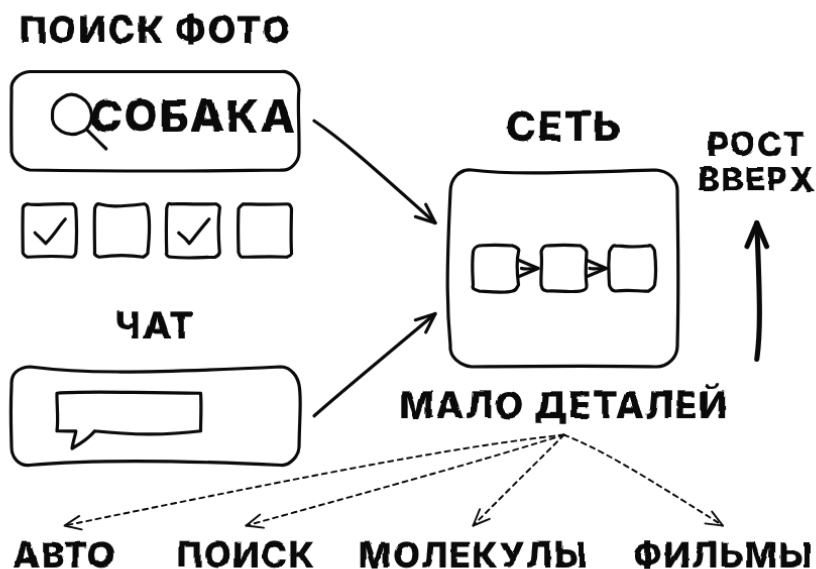
Предсказывать распределение вместо числа

Сделать данные вероятными и сохранить каждую хорошую модель

Приближения, априорные распределения как штрафы и обучение в рассрочку

Зачем заглядывать под капот

Откройте приложение с фотографиями на телефоне и наберите в поиске одно слово, скажем «собака». Через мгновение оно покажет вам все снимки, где есть собака, хотя никто никогда не подписывал ваши фотографии вручную. Теперь откройте чат-ассистента и попросите объяснить рецепт более простыми словами; он ответит гладкими, связными фразами. Оба фокуса, говорит нам книга, выполняют нейронные сети, и обе построены из удивительно небольшого набора руководящих принципов и компонентов. Вот об этих немногих деталях и весь наш курс.



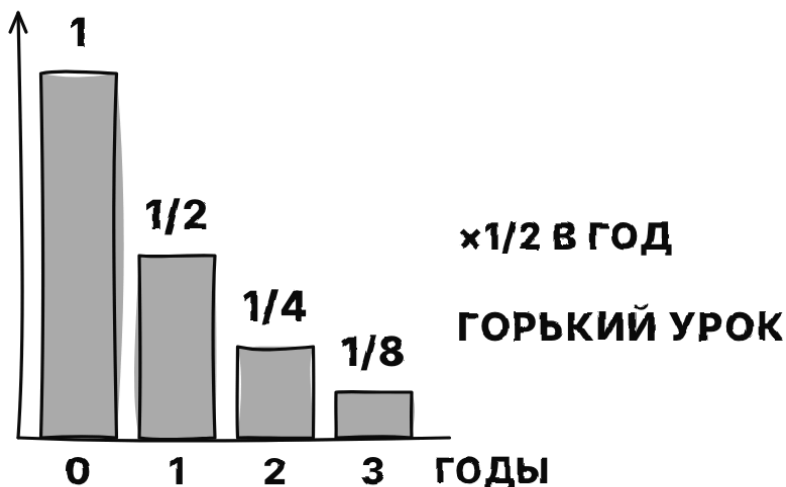
Два приложения, один набор деталей

Нейронные сети стали частью повседневной жизни, иногда открыто, как большие языковые модели, или LLM, а иногда незаметно. Они управляют дронами и автомобилями, работают в поисковых системах, помогают проектировать новые молекулы и подсказывают, что посмотреть дальше. Первое утверждение книги таково: всё это держится на очень немногих идеях, а усилия исследователей сместились с изобретения новых идей на масштабирование старых до пределов физически возможного.

У этого сдвига есть имя: законы масштабирования нейросетей. Неформально говоря, почти для любой задачи одновременное увеличение данных, вычислительной мощности и размера модели почти всегда даёт предсказуемый рост точности. Если сказать то же самое наоборот: вычислительная мощность, нужная для достижения заданной точности, уменьшается в постоянное число раз за каждый промежуток времени. Вот что значит постоянный множитель на числах, которые можно проверить: если стоимость достижения некоторой точности каждый год уменьшается вдвое, то через три года она составит одну вторую на одну вторую на одну вторую, то есть одну восьмую от прежней. Исследователь Ричард Саттон называл это горьким уроком: простые универсальные инструменты в сочетании с экспоненциально растущей вычислительной мощностью снова и снова побеждают более хитрые специальные конструкции.

ЗАКОНЫ РОСТА

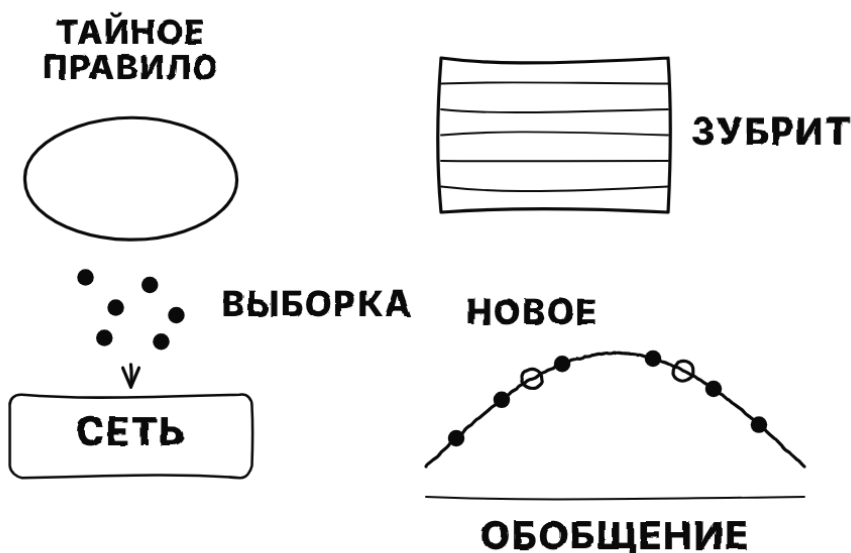
СТОИМОСТЬ



Стоимость падает вдвое каждый год

Если принять законы масштабирования как данность, говорит книга, у нас в руках оказывается почти волшебный инструмент. В двух словах, нейронную сеть оптимизируют так, чтобы она приближала некоторое распределение вероятностей по данным, взятым из этого распределения. Думайте о распределении как о скрытом правиле, которое порождает все фотографии собак на свете; сеть видит выборку из них и пытается уловить это правило. В принципе здесь возможен провал: современные сети так велики, что могли бы просто запомнить каждый показанный пример и превратиться в тривиальную таблицу поиска. Вместо этого обученные модели, как оказывается, хорошо обобщают, даже на задачи, которых в их обучающих данных явно не было. По мере роста наборов данных граница между тем, что внутри

обучающего распределения, и тем, что снаружи, размывается, и очень большие модели показывают признаки сильного обобщения с удивительно малой опорой на чистое запоминание, которое называют переобучением.



Выборка из скрытого правила

Чрезвычайно большие модели, которые можно повторно использовать для многих последующих задач, иногда называют фундаментальными моделями. Вместе с живым сообществом открытого кода они изменили то, как мы пользуемся этими системами. Многие задачи теперь решаются простым промптингом, то есть текстовыми или графическими инструкциями для предобученной модели, найденной в сети, при этом её внутренности остаются полностью чёрным ящиком.

Книга сравнивает это с переходом от написания собственных библиотек на языке вроде C++ к использованию готовых программ, исходный код которых вы никогда не видите. Сравнение не надуманное: сегодня лишь несколько команд в мире обладают вычислительной мощностью и опытом, чтобы спроектировать и выпустить по-настоящему большие модели, такие как семейство языковых моделей Llama, точно так же, как лишь немногие компании способны создавать крупное корпоративное программное обеспечение.



Чёрный ящик, потом открытый

Но так же, как открытый код позволяет настраивать программы или писать новые с нуля, обычное потребительское оборудование и немного изобретательности дают вам широкий простор для

экспериментов с этими моделями. Вы можете дообучить их под свою задачу, слить две модели в одну, квантовать их, то есть хранить их числа с меньшим количеством бит, чтобы они работали на маломощном оборудовании, проверить их устойчивость или спроектировать совершенно новые варианты. Для всего этого нужно заглянуть под капот и понять, как эти модели обрабатывают и преобразуют данные внутри, со всеми приёмами и причудами, усвоенными из опыта и отладки. Книга представляет себя как вход в этот мир и обращается к читателю как к человеку, который, подобно Алисе, любопытен от природы.

Опыт

Задайте тому же чат-ассистенту один и тот же вопрос дважды: один раз короткой простой фразой, а другой раз с несколькими дополнительными указаниями о тоне и длине ответа. Ожидайте, что два ответа будут разными, хотя сама модель ничуть не изменилась. Изменился только промпт; сеть внутри осталась тем же чёрным ящиком, и это ровно тот способ работы, который описывает книга, и причина, по которой она хочет, чтобы вы открыли этот ящик.

Запомним

Нейронные сети сегодня работают в поиске, фотографиях, чат-ассистентах и многом другом, и всё же все они построены из небольшого набора принципов и компонентов. Законы масштабирования нейросетей говорят, что одновременный рост данных, вычислительной мощности и размера модели почти всегда предсказуемо повышает точность, поэтому стоимость заданной точности постоянно уменьшается в одно и то же число раз; Саттон назвал это горьким уроком. Сеть оптимизируют так, чтобы она приближала распределение вероятностей по выборке, и хотя она могла бы просто всё запомнить, большие обученные модели обобщают удивительно хорошо. Фундаментальные модели часто используют только через промптинг, как чёрные ящики, но чтобы настраивать, дообучать, квантовать или изобретать модели, нужно понимать, что происходит внутри, и именно это путешествие предлагает книга.

Вопросы

1. Что, согласно законам масштабирования нейросетей, происходит, когда данные, вычислительную мощность и размер модели увеличивают одновременно?
 - а. Точность почти всегда растёт предсказуемым образом

- b. Точность растёт только на тех задачах, для которых модель была спроектирована
 - c. Точность перестаёт улучшаться, как только модель может запомнить данные
- 2. Почему современная нейронная сеть в принципе могла бы не выучить правило, стоящее за её данными?
 - a. Потому что она никогда не видит достаточно примеров, чтобы под них подстроиться
 - b. Потому что она достаточно велика, чтобы запомнить каждый пример и превратиться в таблицу поиска
 - c. Потому что распределение вероятностей нельзя приблизить по выборке
- 3. С чем сравнивается использование фундаментальной модели только через промптинг?
 - a. С написанием собственных библиотек с нуля на языке вроде C++
 - b. С починкой исходного кода скачанной программы
 - c. С использованием готовых программ, исходный код которых вы не видите
- 4. Какая из этих задач требует заглянуть под капот модели, а не просто подать ей промпт?
 - a. Квантовать её, чтобы она работала на маломощном оборудовании
 - b. Попросить её переписать рецепт более простыми словами
 - c. Переформулировать вопрос, чтобы получить другой ответ

Ответы

1. **a)** Почти для любой задачи одновременный рост данных, вычислений и размера модели почти всегда даёт предсказуемый прирост точности; иначе говоря, вычисления, нужные для заданной точности, постоянно уменьшаются в одно и то же число раз.
2. **b)** Современные сети так велики, что могли бы просто сохранить каждый показанный им пример; удивительно то, что обученные модели вместо этого обобщают, а не только запоминают.
3. **c)** Когда вы только подаёте промпты предобученной модели, её внутренности остаются чёрным ящиком, почти как при использовании программ, исходный код которых вам недоступен.
4. **a)** Дообучение, слияние, квантование, проверка устойчивости и проектирование новых вариантов требуют понимания того, как модель обрабатывает данные внутри, тогда как промптинг обращается с ней как с чёрным ящиком.

Задачи

1. Объясните своими словами, почему книга называет нейронную сеть, обученную по законам масштабирования, почти волшебным инструментом и что в принципе могло бы это испортить.

Подсказка: Подумайте о том, что сеть пытается приблизить и что очень большая сеть могла бы сделать вместо этого.

Ответ: Сеть оптимизируют так, чтобы она приближала распределение вероятностей, стоящее за её данными, и с масштабированием её точность растёт предсказуемо. В принципе огромная сеть могла бы просто запомнить каждый пример и стать таблицей поиска, но обученные модели вместо этого хорошо

обобщают, даже на задачи, которых явно не было в обучающих данных.

2. Книга сравнивает промптинг фундаментальной модели с использованием программы без исходного кода. Объясните, что это сравнение говорит о том, кто строит такие модели и что всё же могут делать все остальные.

Подсказка: Спросите себя, сколько команд способны построить самые большие модели и что позволяет открытый код.

Ответ: Лишь несколько команд обладают вычислениями и опытом, чтобы выпустить по-настоящему большие модели, так же как лишь немногие компании создают крупное корпоративное программное обеспечение. Но, как и с открытым кодом, обычное оборудование и немного изобретательности по-прежнему позволяют любому дообучать, сливать, квантовать, проверять или изобретать модели, если он понимает, как модели устроены внутри.

Обучение как поиск

Возьмите радиоприёмник со старой ручкой настройки и попробуйте найти станцию, которая вам нравится. Вы чуть поворачиваете ручку, слушаете, и если звук становится чище, продолжаете крутить в ту же сторону; если хуже, поворачиваете назад. Вам совсем не нужно знать, как радио устроено внутри, только в какую сторону маленький поворот делает звук лучше. Держите эту картинку в голове, потому что книга описывает почти весь современный ИИ как такое вот кручение ручек, только миллионов ручек одновременно.

УЧИТЕЛЬ: ПОДРАЖАЙ

ПОВОРОТ

ЧИСТОТА



НАЗАД
ПЕРЕБОР



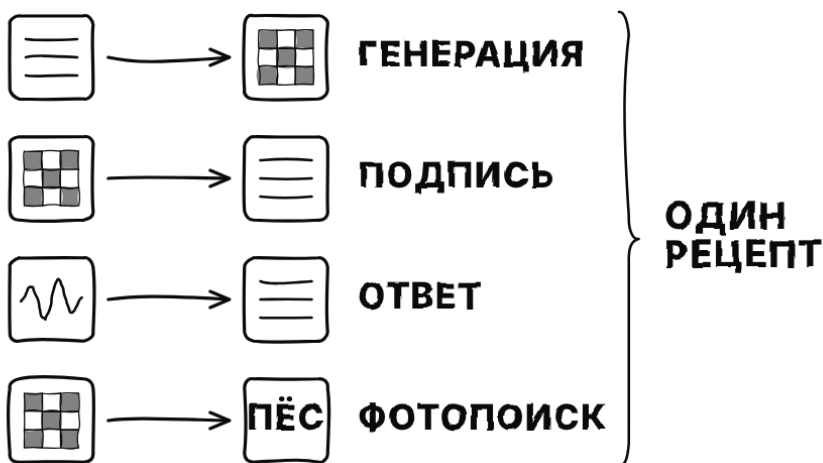
ВНУТРЬ НЕ ГЛЯДИМ



× МИЛЛИОНЫ

Повернуть, послушать, повернуть снова

Книга предполагает, что её читатели знают основы машинного обучения, ML, и конкретнее обучения с учителем, SL. Обучение с учителем решает задачу так: мы собираем примеры нужного нам поведения, а затем обучаем, то есть оптимизируем, систему, пока она не начнёт это поведение воспроизводить. Идея звучит просто, говорит книга, и она обманчиво сильна. Генерация изображений превращается в задачу собрать очень большой набор картинок с подписями. Подражание английскому языку превращается в задачу собрать много текста и научиться предсказывать каждое предложение по предыдущим. Чтение рентгеновского снимка превращается в задачу вести большую базу снимков вместе с решениями врачей по ним.



СЛЕД. ФРАЗА · СНИМОК → РЕШЕНИЕ

Что на входе, что на выходе

Рисунок в книге сортирует задачи по тому, что подаётся на вход и что должно получиться на выходе. Генерация изображений хочет получить картинку, упорядоченную сетку пикселей, из текста, последовательности символов; подпись к изображению — обратная задача, текст из картинки. Ответ на вопрос о записи хочет получить текст из аудиофрагмента, а это снова упорядоченная последовательность, на этот раз чисел.

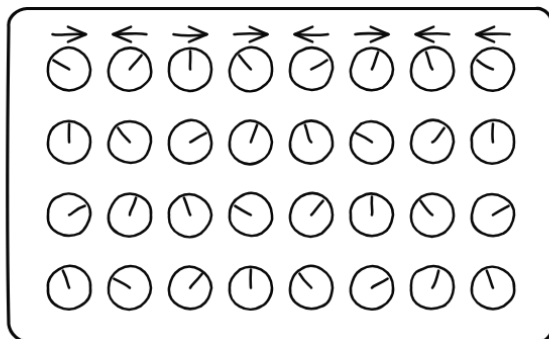
Увлекательное наблюдение книги в том, что модели для всех этих случаев проектируются по очень похожим рецептам.

Приложение с фотографиями в вашем телефоне — одно из них: на входе картинка, на выходе слово вроде «собака», выученное по множеству картинок, которые пришли вместе с подписями.

В общем виде, говорит книга, обучение — это задача поиска. Мы начинаем с программы с очень большим числом степеней свободы, которые называются параметрами; думайте о них как о ручках нашего радио. Затем мы меняем параметры, пока модель не станет работать достаточно хорошо. Чтобы это было практически выполнимо, нужны эффективные способы искать лучшую настройку, даже когда параметров миллионы, миллиарды или даже триллионы, слишком много, чтобы перебирать их по одному.

ОБУЧЕНИЕ = ПОИСК

ПРОГРАММА



$10^6 \dots 10^{12}$

**ПЕРЕБОР
ПО ОДНОМУ $\times$
ГРАДИЕНТ**

ПАРАМЕТРЫ

Программа со множеством ручек

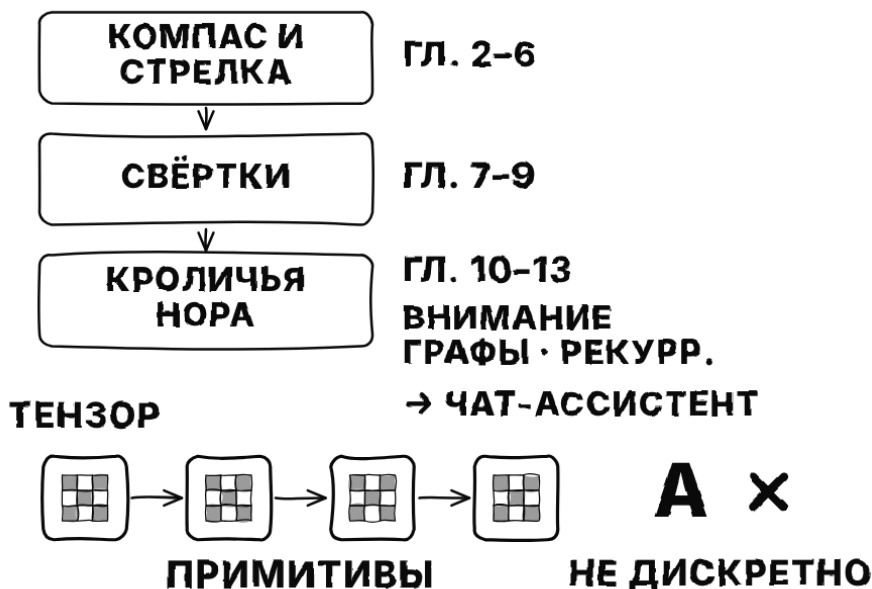
Дифференцируемые модели решают это, разрешая только дифференцируемые компоненты, математические функции, производную которых мы умеем вычислять. Знать производную функции многих входов, которая называется градиентом, значит знать, что произойдёт, если мы чуть сдвинем каждый параметр. Вот эта идея на одной ручке и на числах, которые можно проверить. Пусть ошибка модели равна $f(w) = w^2$, а ручка стоит на $w = 3$, так что ошибка равна девяти. Производная w^2 равна $2w$, что при тройке даёт шесть: это значит, что крошечный поворот ручки меняет ошибку примерно в шесть раз сильнее самого поворота. Проверьте: поверните ручку на 3,01, и ошибка станет 9,0601, то есть изменится на 0,0601, почти ровно в шесть раз

больше поворота на 0,01. Значит, мы знаем, не пробуя, что поворот ручки вниз уменьшит ошибку, и примерно на сколько.



Наклон w в квадрате при тройке

Это знание ведёт к автоматическим процедурам оптимизации, прежде всего к автоматическому дифференцированию, которое вычисляет такие производные для целой программы, и к градиентному спуску, который с их помощью чуть поворачивает каждую ручку в правильную сторону, снова и снова. Всё это устройство — предмет первой части книги под названием «Компас и стрелка», которая занимает главы со второй по шестую.



Карта книги

Как только нейронная сеть увидена как не более чем композиция дифференцируемых строительных блоков, называемых примитивами, книга задаёт два основных вопроса. Первый: какие виды данных такая сеть может принимать и выдавать? Второй: какие примитивы мы можем использовать? Дифференцируемость — сильное требование: оно исключает прямую работу со многими обычными типами данных, такими как символы или целые числа, потому что они скачут от одного значения к другому и поэтому разрывны. Букву нельзя чуть сдвинуть в сторону другой буквы. С чем дифференцируемые модели легко справляются, так это с большими массивами чисел, которые книга будет называть тензорами; фотография — один из них, сетка значений яркости, и такие массивы можно

преобразовывать алгебраически, сцепляя простые линейные и нелинейные шаги. Второй рисунок книги показывает именно это: сеть как цепочку примитивов, работающих со структурированными массивами, где каждый примитив отсортирован по тому, что он принимает и что выдаёт, а это задаёт правила их соединения.

Вторая часть книги подробно изучает один типичный дифференцируемый компонент, свёртку, в главах с седьмой по девятую. Свёртки можно применять всюду, где данные приходят как упорядоченная последовательность элементов: звук, изображения, текст, видео и многое другое. По пути книга вводит приёмы построения глубоких моделей, то есть моделей из многих шагов подряд, и такие идеи, как разбиение текста на токены, порождение последовательности по одному кусочку за раз и причинное моделирование, которые вместе лежат в основе сегодняшних больших языковых моделей.

Третья часть, «Вниз по кроличьей норе», исследует другие конструкции: для множеств, прежде всего слои внимания и модели-трансформеры в главах десятой и одиннадцатой; для графов в главе двенадцатой; и наконец рекуррентные слои для последовательностей во времени в главе тринадцатой. Чат-ассистент, которым вы пользуетесь каждый день, — это трансформер, так что эта часть пути ведёт прямо к нему.

Помимо книги, автор обещает сайт с дополнительными, более исследовательскими главами, которые можно читать в любом порядке: о генеративном моделировании, условных вычислениях, переносе обучения и объяснимости. Есть также серия практических занятий в виде блокнотов, которые следуют

за большей частью книги и доходят до продвинутых тем, таких как контрастивное обучение и слияние моделей.

Запомним

Обучение с учителем означает собрать примеры желаемого поведения и оптимизировать систему так, чтобы она его воспроизводила, и этот простой рецепт одинаково покрывает генерацию изображений, язык и медицинские снимки. Обучение — это поиск по параметрам программы, её ручкам, настройки, которая работает, даже когда их миллиарды. Дифференцируемые модели делают поиск практичным: градиент говорит, как каждый малый сдвиг меняет ошибку, как шестикратный поворот для дубль-вэ в квадрате при тройке, и на этом держатся автоматическое дифференцирование и градиентный спуск. Такие модели работают с тензорами, большими массивами чисел, а не с дискретными вещами вроде символов; три части книги охватывают основы, затем свёртки, затем внимание, графы и рекуррентные слои.

Вопросы

1. Почему книга называет обучение задачей поиска?
 - а. Потому что модель ищет ближайший сохранённый пример для каждого нового входа

- b. Потому что обучение меняет множество параметров программы, пока она не заработает достаточно хорошо
 - c. Потому что обучение с учителем ищет размеченные данные в интернете
2. Что нам даёт знание градиента ошибки модели?
- a. Как меняется ошибка, когда каждый параметр чуть сдвигают
 - b. Лучшую возможную настройку всех параметров сразу
 - c. Список обучающих примеров, на которых модель ошиблась
3. Почему дифференцируемая модель не может работать напрямую с символами или целыми числами?
- a. Потому что разных символов слишком много, чтобы их хранить
 - b. Потому что они слишком малы, чтобы умножать их на матрицы
 - c. Потому что они скачут от одного значения к другому, и их нельзя чуть сдвинуть
4. Чем в основном различаются генерация изображений и подпись к изображению?
- a. Одной нужны размеченные данные, а другой нет
 - b. Тем, что на входе и что должно быть на выходе: текст в картинку против картинки в текст
 - c. Одна — обучение с учителем, а другая — задача поиска

Ответы

1. б) Мы начинаем с программы с очень большим числом параметров, её ручек, и меняем их, пока модель не станет работать достаточно хорошо; трудность в том, чтобы эффективно искать среди миллиардов настроек.

2. а) Градиент говорит, что произойдёт, если каждый параметр чуть сдвинуть, и именно это позволяет автоматическим процедурам, таким как градиентный спуск, улучшать модель шаг за шагом.

3. с) Символы и целые числа дискретны и поэтому разрывны; букву нельзя чуть сдвинуть в сторону другой, а большие массивы чисел, такие как изображения, можно.

4. б) Книга сортирует задачи по входу и выходу: генерация делает сетку пикселей из текста, подпись делает текст из сетки пикселей, и модели в обоих случаях следуют похожим конструкциям.

Разобранные примеры

Одна ручка, один сдвиг Ошибка модели равна $f(w) = w^2$, а её единственный параметр стоит на $w = 3$. С помощью производной предскажите, насколько изменится ошибка, если w повернуть вверх на $0,01$, а затем проверьте прямым вычислением.

Производная w^2 равна $2w$, так что при $w = 3$ она равна 6 : ошибка меняется примерно в шесть раз сильнее сдвига.

Предсказанное изменение: $6 \times 0,01 = 0,06$.

Прямая проверка: $3,01^2 = 9,0601$, а $3^2 = 9$, так что настоящее изменение равно $0,0601$.

Предсказание совпадает почти точно, и именно поэтому градиент говорит нам, куда крутить ручку, без перебора всех значений.

Ответ: Примерно $0,06$ (точное изменение $0,0601$), так что уменьшение w снизило бы ошибку.

Задачи

1. При той же ошибке $f(w) = w^2$ ручка теперь стоит на $w = 5$. В какую сторону её повернуть, чтобы снизить ошибку, и примерно насколько изменится ошибка при повороте на $0,1$?

Подсказка: Сначала найдите производную $2w$ при $w = 5$.

Ответ: Повернуть вниз. Производная равна $2 \times 5 = 10$, так что поворот на $0,1$ меняет ошибку примерно на $10 \times 0,1 = 1$ (точно: $5,1^2 - 25 = 1,01$).

2. Выберите задачу из своего телефона, например поиск фотографий по слову «собака» или получение ответа от чат-ассистента, и опишите её так, как книга описывает задачи: что на входе, что на выходе и какие данные собрал бы подход обучения с учителем.

Подсказка: Посмотрите на примеры с генерацией изображений, языком и рентгеновскими снимками.

Ответ: Поиск фотографий принимает картинку и выдаёт метку; нужно собрать много фотографий в паре с их метками и оптимизировать модель так, чтобы она их воспроизводила. Чат-ассистент принимает текст и выдаёт текст; нужно собрать большое количество текста и научиться предсказывать каждый кусок по тому, что шло раньше.

3. Объясните, почему фотографию можно подать дифференцируемой модели, а слово, записанное буквами, напрямую подать нельзя.

Подсказка: Спросите себя, можно ли каждое из них изменить на крошечную величину.

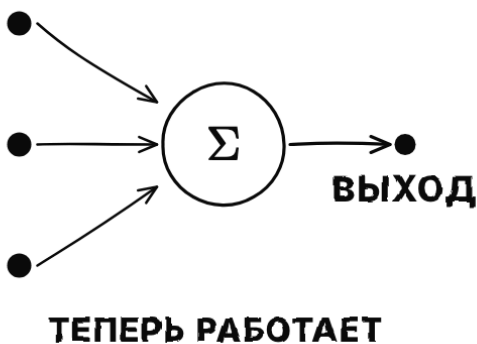
Ответ: Фотография — это массив чисел, тензор значений яркости, который можно чуть сдвинуть и преобразовать линейными и

нелинейными шагами. Буквы дискретны: между одной буквой и следующей ничего нет, значит, нет малого сдвига, а потому и производной, за которой можно следовать.

Старые идеи, новая мощь

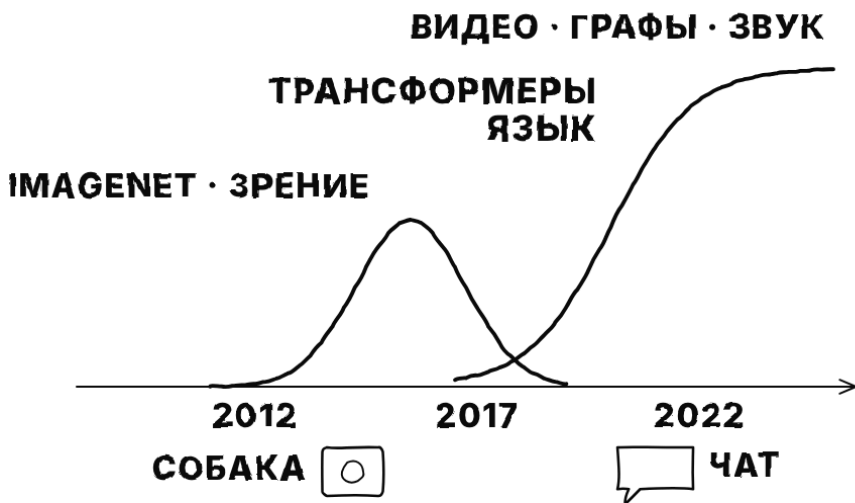
Поищите старые газетные заметки о «машине, которая учится», и вы найдёте восторженные заголовки, звучащие как новости этой недели о чат-ассистентах. Удивляет дата на них. Книга показывает один такой момент на рисунке, который она с подмигиванием подписывает «хайп вокруг ИИ»: идёт 1958 год, и американский психолог Фрэнк Розенблатт привлекает широкое внимание прессы своими перцептронами, одним из первых работающих прототипов нейронной сети. Так что восторг не нов; ново то, насколько хорошо эти машины наконец работают.

ХАЙП ИИ 1958 · ПЕРЦЕПТРОН



У нейронных сетей долгая и богатая история. Само их название восходит к попыткам начала двадцатого века моделировать биологические нейроны, нервные клетки мозга. Словарь прижился: чтобы соответствовать сегодняшним программным библиотекам, книга по-прежнему будет говорить о нейронах, слоях и активациях, хотя аналогия с мозгом давно поблёкла.

Интерес приходил несколькими волнами. Между 2012 и 2017 годами сети усложнялись как никогда прежде, подталкиваемые крупными открытыми тестами и соревнованиями, прежде всего соревнованием ImageNet Large Scale Visual Recognition Challenge, ILSVRC, о котором книга рассказывает в девятой главе. Вторая большая волна пришла в 2017 году с трансформерами, предметом десятой главы. Как несколько лет раньше компьютерное зрение захватили свёрточные модели, так обработку естественного языка за очень короткое время захватили трансформеры. Дальнейший прогресс последовал для видео, для графов, которым посвящена двенадцатая глава, и для звука, что привело к сегодняшнему восторгу вокруг больших языковых моделей, мультимодальных сетей, смешивающих текст, картинки и звук, и генеративных моделей, которые создают новое содержимое. Приложение с фотографиями, находящее вашу собаку, принадлежит первой волне; чат-ассистент, который вам отвечает, — второй.



Две волны

Слова менялись так же быстро, как модели. В 1980-е говорили о коннекционизме; позже «глубокое обучение» стало означать современные сети в противовес меньшим, более мелким моделям прошлого. Книга находит все эти термины безнадежно расплывчатыми, потому что современные искусственные сети почти не сохранили сходства с биологическими сетями или с неврологией. Что действительно объединяет современные сети, так это то, что они собраны из дифференцируемых блоков, частей, производные которых можно вычислить, как мы видели на примере ручки, которую можно чуть сдвинуть. По этой причине книга предпочитает называть их дифференцируемыми моделями везде, где может.

× **КОННЕКЦИОНИЗМ ДИФФЕРЕНЦИРУЕМОЕ
(1980-Е) ПРОГРАММИРОВАНИЕ**

× **ГЛУБОКОЕ ОБУЧЕНИЕ**

РАСПЛЫВЧАТО



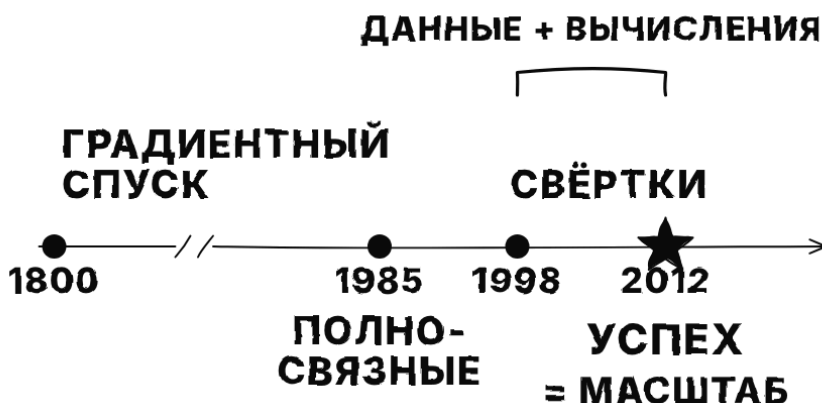
ЛЕКУН 2018 · РАФФЕЛ 2021

Названия, а потом название получше

Этот взгляд открывает более широкую область, называемую дифференцируемым программированием, молодую дисциплину, которая смешивает информатику и оптимизацию, чтобы изучать компьютерные программы, которые можно дифференцировать в общем виде. Книга отдаёт должное двум коротким манифестам, сформировавшим этот взгляд: одному Яна Лекуна 2018 года, отстаивающему это более широкое понимание нейронных сетей, и одному Колина Раффела 2021 года, призывающему строить модели так, как строится программное обеспечение с открытым кодом.

Путешествие по этой стране дифференцируемых моделей — это ещё и путешествие по истории, говорит книга. Численная

оптимизация линейных моделей градиентным спуском, предмет четвёртой главы, была известна по меньшей мере с девятнадцатого века. Полносвязные сети в том виде, в каком книга использует их позже, восходят к 1980-м годам. Свёрточные модели были уже известны и применялись в конце 1990-х. Нанесите эти даты на линию, и появится разрыв: метод из 1800-х, другой из 1980-х, ещё один из 1990-х, а большие успехи пришли только после 2012 года.



Старые идеи, поздний успех

Не хватало, объясняет книга, не идей, а средств: прошло много десятилетий, прежде чем появилось достаточно данных и достаточно вычислительной мощности, чтобы увидеть, как хорошо эти модели работают, когда у них вдоволь и того, и

другого, и вдоволь параметров. Это возвращает нас к законам масштабирования из первой части: старые рецепты, приготовленные в гораздо большем масштабе, оказались необыкновенно хороши.

Книга честно говорит о своих границах. Она не может глубоко разобрать каждую тему, отчасти потому, что исследования движутся так быстро. Вместо этого её цель — дать достаточно основ, чтобы читатель мог ориентироваться в новейших научных статьях и не теряться. Тем, кто хочет полной истории, она указывает на обзор современных нейронных сетей, на книгу интервью с некоторыми главными фигурами области и на длинную, небеспристрастную комментированную историю Юргена Шмидхубера.

Опыт

Напишите три даты на полоске бумаги: 1800 для градиентного спуска на линейных моделях, 1985 для полносвязных сетей, 1998 для свёрточных моделей. Затем добавьте 2012, начало великой волны, и 2017, приход трансформеров. Вы увидите, что большинство идей лежали на полоске задолго до успехов, и в этом мысль книги: перемена пришла от данных и вычислительной мощности, а не от изобретения идей.

Запомним

У нейронных сетей долгая история: от ранних моделей биологических нейронов через перцептрон Розенблатта 1958 года к волнам 2012–2017 годов, движимым ImageNet, и трансформеров с 2017 года. Названия вроде коннекционизма и глубокого обучения расплывчаты, потому что современные сети едва ли похожи на мозг; общее у них то, что они собраны из дифференцируемых блоков, поэтому книга называет их дифференцируемыми моделями. Этот взгляд ведёт к дифференцируемому программированию, дисциплине на стыке информатики и оптимизации. Градиентный спуск, полносвязные сети и свёртки были известны десятилетия назад; засиять им дали достаточные данные, вычислительная мощность и параметры.

Вопросы

1. Градиентный спуск, полносвязные сети и свёртки были известны задолго до 2012 года. Почему их большие успехи пришли только потом?
 - а. Потому что в старых методах были ошибки, которые исправили после 2012 года
 - б. Потому что понадобились десятилетия, чтобы появилось достаточно данных, вычислительной мощности и параметров и стало видно, как хорошо они работают

- с. Потому что мозг наконец поняли достаточно хорошо, чтобы его скопировать
- 2. Почему книга предпочитает название «дифференцируемые модели» названиям «нейронные сети» или «глубокое обучение»?
 - а. Потому что современные сети едва ли похожи на мозг, а общее у них то, что они собраны из дифференцируемых блоков
 - б. Потому что старые названия — охраняемые товарные знаки программных библиотек
 - с. Потому что дифференцируемые модели — это другой, более новый вид сетей
- 3. Какую роль сыграли трансформеры во второй волне по сравнению со свёрточными моделями в первой?
 - а. Трансформеры заменили свёртки в распознавании изображений во время соревнования ImageNet
 - б. Трансформеры были более старой идеей, которую затем обогнали свёрточные модели
 - с. Трансформеры за короткое время захватили обработку языка, как свёрточные модели захватили компьютерное зрение

Ответы

1. б) Идеи уже были; не хватало средств, достаточных данных и вычислительной мощности, а также множества параметров, чтобы раскрылось, как хорошо работают эти модели.

- 2. а)** Старые названия расплывчаты: современные искусственные сети почти не сохранили сходства с биологическими, а их существенная черта в том, что они составлены из дифференцируемых блоков.
- 3. с)** Как несколькими годами раньше компьютерное зрение захватили свёрточные модели, так обработку естественного языка за очень короткое время после 2017 года захватили трансформеры.

Задачи

- 1.** Объясните своими словами, почему книга показывает газетный момент 1958 года о перцептроне Розенблатта под подписью «хайп вокруг ИИ».

Подсказка: Сравните восторг тогда с восторгом сейчас и спросите себя, что на самом деле изменилось.

Ответ: Перцептрон Розенблатта, одна из первых работающих нейронных сетей, уже в 1958 году привлёк широкое внимание прессы. Подпись показывает, что восторг по поводу обучающихся машин стар; ново сегодня то, насколько хорошо такие машины наконец работают, благодаря данным и вычислительной мощности.

- 2.** Расположите на временной шкале и скажите, чему учит разрыв между ними: градиентный спуск для линейных моделей, полносвязные сети, свёрточные модели, волна 2012–2017 годов, трансформеры.

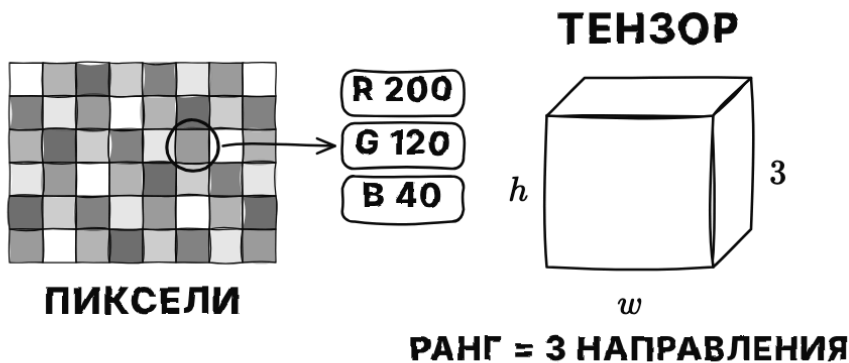
Подсказка: Используйте даты, которые даёт книга: девятнадцатый век, 1980-е, конец 1990-х, 2012, 2017.

Ответ: Градиентный спуск на линейных моделях известен по меньшей мере с девятнадцатого века, полносвязные сети — с 1980-х, свёрточные модели — с конца 1990-х, великая волна — с 2012 по 2017 год, трансформеры — с 2017 года. Большинство идей существовали

задолго до успехов, так что перемена пришла от достаточных данных, вычислительной мощности и параметров.

Коробки чисел и их формы

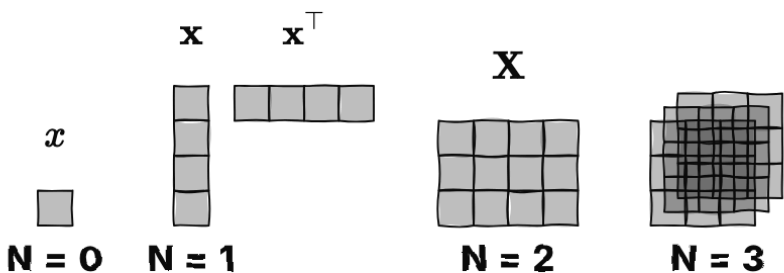
Откройте любую фотографию на телефоне и увеличивайте её, пока не увидите маленькие квадратики. Каждый квадратик — это пиксель, и каждый пиксель хранится как три числа: сколько красного, сколько зелёного и сколько синего. Так что для компьютера фотография вовсе не картинка; это коробка чисел, столько-то квадратиков в высоту, столько-то в ширину и три в глубину. Спросите, как выглядит такая коробка для целого альбома, и вы получите первую идею этой части.



Фотография — коробка чисел

Основной тип данных дифференцируемых моделей — тензор, который книга определяет как n -мерный массив объектов, обычно вещественных чисел. Книга признаёт, что название — небольшое злоупотребление: в научной литературе тензор — вещь более точная, мультилинейный оператор, а здесь это просто многомерный массив; название так распространено, что книга его сохраняет. С извинениями перед математиками она называет n рангом тензора, то есть числом направлений, в которых коробка простирается.

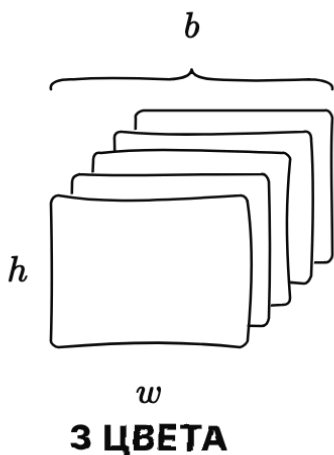
Обозначения меняются с рангом. Тензор с $n = 0$ содержит одно значение, скаляр, записываемый строчной буквой, такой как x или y . Скаляры называются так потому, что любой из них можно записать как кратное, то есть масштабированную копию, единицы. Тензор с $n = 1$ — это столбец значений, вектор, записываемый строчной жирной буквой, $\mathbf{x}$. Те же значения, выложенные в строку, записываются как $\mathbf{x}^T$, где маленькая T означает, что столбец положили на бок; когда контекст ясен, книга опускает T для удобства чтения. На рисунке книга изображает векторы строками только ради экономии места; в тексте каждый вектор — столбец. Тензор с $n = 2$ — это прямоугольник значений, матрица, записываемая прописной жирной буквой, $\mathbf{X}$ или $\mathbf{Y}$. Для рангов выше двух особых букв нет, а витиеватые каллиграфические буквы, такие как $\mathcal{X}$, оставлены для множеств и распределений вероятностей.



$\mathcal{X}$: множества

Ранг от нуля до трёх

Чаще всего, говорит книга, понять алгоритм или операцию — значит понять форму каждого участвующего тензора. Чтобы записывать формы компактно, она использует знак, похожий на маленькую волну: $X \sim (b, h, w, 3)$. Словами: X — тензор ранга четыре, четыре размера которого равны b , h , w и 3 . Откуда он берётся — из нашего фотоальбома: b — число фотографий в батче, h — высота и w — ширина в пикселях, а 3 — три цветовых канала. Некоторые размеры — фиксированные числа, как эта тройка, а другие — буквы, которые могут принимать любое значение.



$$X \sim (b, h, w, 3)$$

$$b = 32, h = w = 224$$

$$224 \times 224 \times 3 = 150,528$$

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

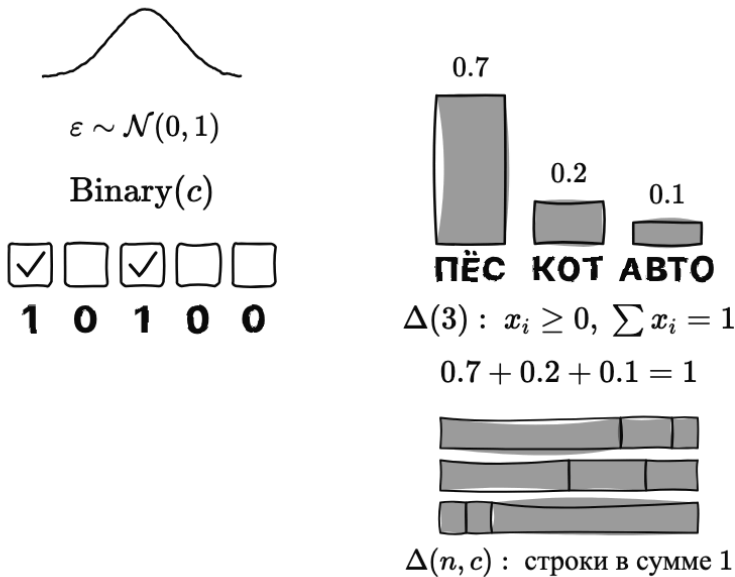
$$\approx 5 \text{ МИЛЛИОНОВ}$$

Батч фотографий

Давайте посчитаем вслух. Возьмём батч из 32 фотографий, каждая 224 пикселя в высоту и 224 в ширину. Одна фотография содержит 224 на 224, то есть 50 176 пикселей, умножить на 3 цвета, что даёт 150 528 чисел. Весь батч содержит в 32 раза больше: 4 816 896 чисел в одном тензоре ранга четыре, почти пять миллионов. Вот почему формы так важны: они с одного взгляда говорят, сколько всего течёт через модель.

Тот же знак волны означает также извлечение случайного значения из распределения вероятностей, как в $\epsilon \sim \mathcal{N}(0, 1)$, число, взятое из колоколообразной кривой с центром в нуле. Книга использует его так лишь изредка, и контекст всегда делает смысл ясным. Так что $x \sim (d)$ заменяет более привычное $x \in \mathbb{R}^d$, вектор

из d вещественных чисел, а $\mathbf{X} \sim (n, d)$ заменяет $\mathbf{X} \in \mathbb{R}^{n \times d}$, матрицу из n строк и d столбцов.



Правила для значений

Иногда значения в тензоре должны подчиняться правилу, и для этого тоже есть обозначение. $\mathbf{x} \sim \text{Binary}(c)$ — это тензор размера c , элементы которого только нули и единицы, например ряд галочек, говорящих, какие из c меток есть у фотографии. $\mathbf{x} \sim \Delta(a)$ — это вектор в так называемом симплексе: каждый элемент не меньше нуля, $x_i \geq 0$, и все они в сумме дают единицу, $\sum_i x_i = 1$. Именно так выглядит список вероятностей. Если ваше фотоприложение говорит: собака 0,7, кошка 0,2, машина 0,1, все три не меньше нуля, а 0,7 плюс 0,2 плюс 0,1 равно 1, так что этот вектор лежит в симплексе размера три. Для более высоких

рангов, как в $X \sim \Delta(n, c)$, правило действует вдоль последнего измерения: каждая из n строк — список из c вероятностей с суммой единица, по одной строке на фотографию.

Дальнейшие обозначения вводятся по мере надобности глава за главой.

Опыт

Возьмите лист бумаги в клетку и раскрасьте крошечную картинку три клетки в высоту и четыре в ширину. Теперь напишите для каждой клетки три числа — для её красного, зелёного и синего; у вас тензор формы три на четыре на три, в котором 36 чисел. Сложите два таких листа стопкой, и форма станет два на три на четыре на три, с 72 числами: вы только что собрали маленький батч, объект того же рода, что и X в книге.

Запомним

Тензор — это многомерный массив чисел, а его ранг — число направлений, в которых он простирается: у скаляра ранг ноль, у вектора ранг один, у матрицы ранг два. Скаляры пишутся строчными, векторы строчными жирными и всегда столбцами, матрицы прописными жирными; витиеватые буквы оставлены для множеств и распределений. Знак волны даёт форму тензора, как в $X \sim (b, h, w, z)$ для батча цветных фотографий, а батч из 32 фотографий по 224 на 224 пикселя содержит почти пять миллионов чисел. $\text{Binary}(c)$ означает только нули и единицы, а $\Delta(a)$, симплекс, означает неотрицательные значения с суммой единица, форму списка вероятностей.

Вопросы

1. Батч цветных фотографий записывается как $X \sim (b, h, w, z)$. Каков его ранг и почему последний размер — фиксированное число?
 - a. Ранг три, потому что размер батча b не считается измерением
 - b. Ранг четыре, а z фиксирована, потому что у каждого пикселя три значения цвета
 - c. Ранг четыре, а z фиксирована, потому что в каждом батче три фотографии

2. Какой из этих векторов лежит в симплексе $\Delta(3)$?
- a. (0,5, 0,5, 0,5)
 - b. (0,9, 0,3, -0,2)
 - c. (0,6, 0,3, 0,1)
3. Почему книга говорит, что понять операцию — это по большей части понять формы?
- a. Потому что форма каждого тензора с одного взгляда говорит, что входит, что выходит и сколько всего течёт через модель
 - b. Потому что значения внутри тензоров никогда не важны для работы модели
 - c. Потому что формы — единственное, что компьютер может хранить
4. Как в обозначениях книги отличить на странице вектор от матрицы?
- a. Вектор — прописная жирная буква, а матрица — строчная жирная
 - b. Вектор — строчная жирная буква, матрица — прописная жирная
 - c. Оба записываются витиеватыми каллиграфическими буквами

Ответы

1. b) Форма перечисляет четыре размера, значит, ранг равен четырём; b, h и w могут меняться, а u каждого пикселя всегда есть красный, зелёный и синий, поэтому последний размер равен 3.

2. c) У вектора в симплексе каждый элемент не меньше нуля, а сумма элементов равна единице; $0,6 + 0,3 + 0,1 = 1$, тогда как первый даёт в сумме 1,5, а у второго есть отрицательный элемент.

3. a) Следя за формой каждого тензора, видно, что операция принимает и что выдаёт и насколько она велика, как показывают почти пять миллионов чисел в батче из 32 фотографий.

4. b) Векторы записываются строчными жирными буквами, как жирный x , матрицы — прописными жирными, как жирный большой X , а витиеватые буквы оставлены для множеств и распределений вероятностей.

Разобранные примеры

Сколько чисел в батче Батч фотографий имеет форму $(b, h, w, 3)$, где $b = 32$, $h = 224$ и $w = 224$. Сколько чисел содержит тензор?

Одна фотография содержит $h \times w = 224 \times 224 = 50\,176$ пикселей.

Каждый пиксель хранит 3 значения цвета, так что одна фотография содержит $50\,176 \times 3 = 150\,528$ чисел.

Батч складывает стопкой 32 фотографии, так что он содержит $32 \times 150\,528$ чисел.

$32 \times 150\,528 = 4\,816\,896$.

Ответ: 4 816 896 чисел, почти пять миллионов

Задачи

1. Батч из 10 цветных фотографий, каждая 100 пикселей в высоту и 50 в ширину. Запишите его форму в обозначениях книги, назовите ранг и посчитайте число чисел.

Подсказка: Форма — $(b, h, w, 3)$; перемножьте все четыре размера.

Ответ: $X \sim (10, 100, 50, 3)$, ранг четыре. Он содержит $10 \times 100 \times 50 \times 3 = 150\,000$ чисел.

2. Классификатор даёт для каждой из 2 фотографий вероятности 3 классов: (0,5, 0,3, 0,2) и (0,1, 0,1, 0,8). Запишите форму этого результата и проверьте, что он принадлежит $\Delta(2, 3)$.

Подсказка: Правило симплекса действует вдоль последнего измерения, строка за строкой.

Ответ: Результат имеет форму (2, 3). В каждой строке элементы не меньше нуля и в сумме дают единицу: $0,5 + 0,3 + 0,2 = 1$ и $0,1 + 0,1 + 0,8 = 1$, так что он принадлежит $\Delta(2, 3)$.

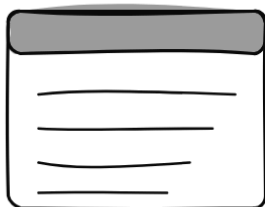
3. Перепишите $\mathbf{X} \in \mathbb{R}^{n \times d}$ и $\mathbf{x} \in \mathbb{R}^d$ в обозначениях формы из книги и скажите, что представляет собой каждый объект.

Подсказка: Знак волны заменяет «принадлежит $\mathbb{R}$ в степени ...».

Ответ: $\mathbf{X} \sim (n, d)$, матрица из n строк и d столбцов вещественных чисел; $\mathbf{x} \sim (d)$, вектор-столбец из d вещественных чисел.

Чем эта книга является, а чем нет

Прочитайте рецепт хлеба, а потом попробуйте его испечь, и вы выучите две разные вещи. Рецепт говорит вам, что происходит с мукой, водой и дрожжами; ваши руки учатся тому, насколько влажным должно быть тесто, а этого никакой рецепт записать не может. Книга проводит ту же границу для нейронных сетей: она даёт вам рецепт полностью и честно признаёт, что чувство приходит только с практикой.

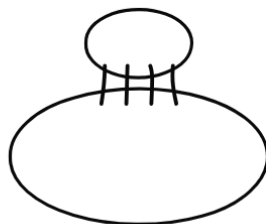


РЕЦЕПТ = КНИГА



КУРС В САПИЕНЦЕ

**ИНЖ. · ТЕЛЕКОМ
ШКОЛЫ**



ЧУВСТВО = ПРАКТИКА

Рецепт и чувство

Книга выросла из лекций. Автор много лет читал курс под названием «Нейронные сети для приложений науки о данных» в магистратуре по науке о данных Римского университета Сапиенца и хотел придать этим лекциям связную письменную форму. Основные главы составляют главную часть того курса; остальные главы покрывают темы, которые преподавались в одни годы и не преподавались в другие. Некоторые части опираются и на другое преподавание: курс нейронных сетей для компьютерной инженерии, введение в машинное обучение для инженеров телекоммуникаций и разные учебные занятия, аспирантские курсы и летние школы.

Уже есть несколько превосходных и недавних книг о современных глубоких нейронных сетях, и книга открыто об этом говорит. В начале она покрывает во многом ту же почву, что и они; позже способ объяснения, некоторые дополнительные части и несколько разделов продвинутых глав пересекаются меньше, следуя собственным исследовательским интересам автора. Её заявленная надежда — предложить дополнительный, взаимодополняющий взгляд на материал, который существует и в других местах.

ТЕОРИЯ + КОД

ДИЗАЙН $\rightleftharpoons$ КОД



НЕ РУКОВОДСТВО X

Библиотеки за возрождением

Название «дифференцируемые программы» выбрано намеренно: их понимание приходит и из теории, и из программирования. То, как модель спроектирована, и то, как она реализована, постоянно влияют друг на друга, и автоматическое дифференцирование, которое вычисляет градиенты для целых программ, — самый ясный тому пример. Книга связывает большую часть возрождения нейронных сетей, примерно с 2012 года, с мощными программными библиотеками: сначала Theano, затем Caffe и Chainer, а потом сегодняшние TensorFlow, PyTorch и JAX, среди прочих. Везде, где возможно, она связывает свои объяснения с идеями из этих библиотек, главным образом PyTorch и JAX. Но это не руководство по программированию, и

за полным введением в каждую библиотеку она отправляет читателей к документации самих библиотек.

Прежде чем отправиться в путь, книга перечисляет, чем она не является. Во-первых, она не пытается быть полной. Она выбирает идеи, которые и распространены сегодня, и достаточно общи, чтобы оставаться полезными какое-то время, но она не видит будущего, и некоторые части могут оказаться неполными или устаревшими к тому времени, когда вы их читаете. В области, где чат-ассистент в вашем телефоне может заменяться новой версией каждые несколько месяцев, такое предупреждение просто реалистично.

Во-вторых, для каждой идеи она показывает несколько вариаций из исследовательской литературы, например переход от батч-нормализации к нормализации по слою, двух способов перемасштабировать числа внутри сети. Существуют ещё сотни вариаций, и книга приглашает читателей исследовать их на сайте Papers With Code.

В-третьих, и это самое важное, это книга о фундаментальных компонентах дифференцируемых моделей. Чтобы строить их в большом масштабе и действительно заставлять работать, нужны ещё инженерное мастерство и некоторая доля интуиции. Об оборудовании книга говорит мало, а для интуиции, говорит она, нет ничего лучше опыта и небеспристрастных блогов, указывая в особенности на рецепт обучения нейронных сетей Андрея Карпатого и его видеосерию «Zero to Hero». Это снова наш хлеб: книга — это рецепт, а чувство теста выучивается выпечкой.

Опыт

Откройте страницу документации PyTorch или JAX и посмотрите, сколько отдельных функций она перечисляет. Затем сравните это с теми немногими идеями, которые этот курс назвал до сих пор: тензоры, градиенты, дифференцируемые строительные блоки. Вы должны обнаружить, что библиотеки огромны, а основных идей мало; книга учит немногому, а документация покрывает многое.

Запомним

Книга превращает многолетние лекции автора о нейронных сетях в Римском университете Сапиенца в связный текст с основными главами и отдельными дополнительными темами. В начале она пересекается с другими хорошими книгами, а затем идёт своим путём, предлагая взаимодополняющий взгляд. Теория и код идут вместе: возрождение нейронных сетей примерно с 2012 года во многом обязано библиотекам от Theano до PyTorch и JAX, на которые книга ссылается, не будучи руководством по программированию. Она не полна, показывает лишь немногие из многих вариантов и учит фундаментальным компонентам, тогда как строительство в большом масштабе требует ещё инженерии и интуиции, которые приходят с опытом.

Вопросы

1. Почему книга говорит, что для понимания дифференцируемых программ нужны и теория, и программирование?
 - a. Потому что теория слишком трудна, чтобы следить за ней, не запуская код
 - b. Потому что то, как модели проектируются, и то, как они реализуются, постоянно формируют друг друга, как показывает автоматическое дифференцирование
 - c. Потому что каждая глава написана как руководство по программированию
2. Что, по словам книги, она оставляет в основном опыту и блогам?
 - a. Интуицию и инженерию, нужные, чтобы строить модели в большом масштабе и заставлять их работать
 - b. Определения тензоров и градиентов
 - c. Историю нейронных сетей с 1958 года
3. Почему книга предупреждает, что некоторые части могут устареть к тому времени, когда вы их читаете?
 - a. Потому что её примеры написаны для старой версии PyTorch
 - b. Потому что курс, из которого она выросла, больше не преподаётся
 - c. Потому что она выбирает идеи, распространённые сегодня и общие на какое-то время, но область движется

быстро, и она не стремится к полноте

Ответы

1. **б)** Проектирование и реализация постоянно взаимодействуют, и автоматическое дифференцирование, вычисляющее градиенты для целых программ, — самый ясный пример; при этом книга не является руководством по программированию.
2. **а)** Она учит фундаментальным компонентам, но строительство в масштабе требует инженерного мастерства и интуиции, а для интуиции нет ничего лучше опыта и небеспристрастных блогов.
3. **с)** Она выбирает идеи, одновременно распространённые сейчас и достаточно общие, чтобы продержаться, но не может предвидеть будущее и не стремится к полноте.

Задачи

1. Объясните своими словами разницу, которую книга проводит между пониманием фундаментальных компонентов сети и тем, чтобы заставить большую сеть действительно работать.

Подсказка: Вспомните хлеб: рецепт и чувство теста.

Ответ: Книга полностью объясняет компоненты, как рецепт. Для строительства и обучения в большом масштабе нужны ещё инженерное мастерство, знание оборудования и интуиция, которая приходит с опытом, и за ней книга отправляет к практике и к небеспристрастным руководствам вроде руководства Карпатого.

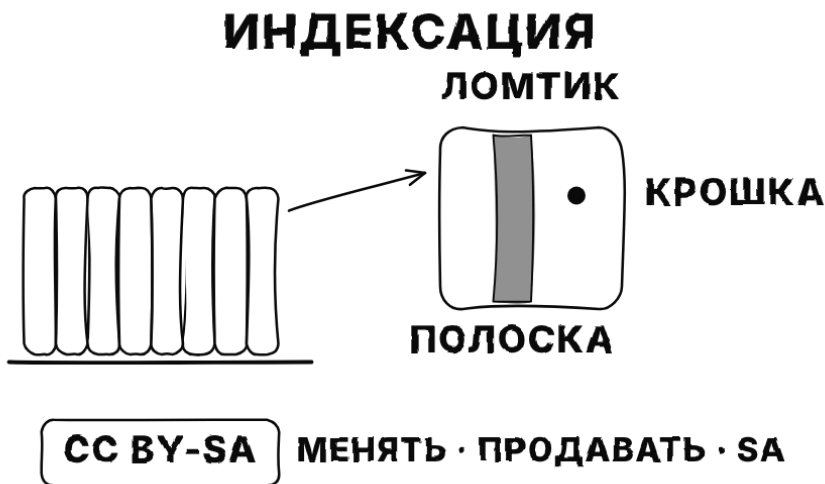
2. Почему читателю, у которого уже есть другая книга по глубокому обучению, эта всё же может пригодиться, по словам самой книги?

Подсказка: Посмотрите, где книга пересекается с другими, а где нет.

Ответ: В начале она покрывает во многом ту же почву, но её способ объяснения, некоторые дополнительные части и несколько продвинутых разделов следуют исследовательским интересам автора, так что она предлагает дополнительный, взаимодополняющий взгляд на материал.

Нарезаем тензор ломтиками

Достаньте из пакета нарезанный батон хлеба. Вы можете вынуть один целый ломтик, можете отрезать полоску из середины этого ломтика или отщипнуть одну крошку в точно выбранном месте. Каждое из этого — способ взять часть батона, сказав, где она лежит, и именно это значит индексировать тензор. Прежде чем мы начнём резать, книга завершает своё введение благодарностями и лицензией.



Ломтик, полоска, крошка

Книга благодарит тех, кто помог её создать: раскраска её формул взята из пакета LaTeX, написанного Эс-Ти Джоном; её цветные картинки с Алисой и символы на полях лицензированы из библиотеки стоковых изображений; а картинки с Алисой внутри её рисунков воспроизводят оригинальные иллюстрации Джона Тенниела через Викимедиа. Она благодарит Роберто Альму за отзывы на ранний черновик и за поддержку публикации, а Коррадо Дзокколо, Эмануэле Родолу, Марцина Слабого, Константина Бурлаченко и Диего Сандовала за обширные исправления, вместе со всеми, кто писал по электронной почте.

Книга выпущена под лицензией Creative Commons CC BY-SA. Словами самой лицензии, она позволяет другим распространять, перерабатывать, адаптировать материал и строить на его основе в любой среде и формате, пока указан автор; она разрешает коммерческое использование; и всё, что построено на её основе, должно выпускаться на тех же условиях. Курс, который вы слушаете, — такая адаптация, и он несёт ту же лицензию.

Вторая глава собирает математику, которая нужна остальной книге. Она предполагает, что вы уже встречались с этими темами, и сосредоточивается на обозначениях и связном обзоре, соединяя такие идеи, как тензоры, с тем, как они реализуются на практике. У неё три части, каждая строится на предыдущей: линейная алгебра, затем градиенты для объектов со многими измерениями, затем как оптимизировать функции с помощью этих градиентов. Отдельное приложение даёт самостоятельное изложение теории вероятностей с упором на принцип максимального правдоподобия. Книга предупреждает, что глава плотно набита определениями, и просит читателя потерпеть.

ЛИНЕЙНАЯ
АЛГЕБРА

ГРАДИЕНТЫ

ОПТИМИЗАЦИЯ

ПРИЛОЖЕНИЕ:
ВЕРОЯТНОСТЬ



$$X \sim (s_1, s_2, \dots, s_n)$$

$$X \sim (4, 5, 6) : 4 \times 5 \times 6 = 120$$

РАНГ 3

Вторая глава и форма тензора

Первая часть напоминает линейную алгебру, отчасти чтобы договориться об общих обозначениях, и начинается она с тензора.

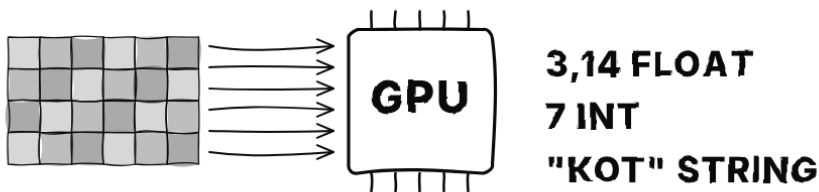
Определение: тензор X — это n -мерный массив элементов, все одного типа, а его форма записывается как $X \sim (s_1, s_2, \dots, s_n)$.

Словами: список в скобках даёт размер каждого из n измерений.

Например, батч из 4 маленьких серых фотографий, каждая 5 пикселей в высоту и 6 в ширину, имеет форму $X \sim (4, 5, 6)$ и содержит 4 на 5 на 6, то есть 120 чисел. При n , равном нулю, получаются скаляры, при единице — векторы, при двойке — матрицы, а выше — многомерные массивы, с теми буквами, о которых мы уже договорились: строчные для скаляров, строчные жирные для векторов, прописные жирные для матриц.

Почему тензоры так важны для глубокого обучения? Потому что они подходят для массово-параллельных вычислений: графические процессоры, GPU, и более специализированные чипы, такие как TPU и IPU, могут работать со всеми числами тензора одновременно. Вот почему телефон в вашем кармане может быстро разметить целый альбом фотографий: он обращается с ними как с одним большим тензором, а не с фотографиями одна за другой.

ПАРАЛЛЕЛЬНО



АЛЬБОМ

КНИГА

1 2 3 4 5



NUMPY

0 1 2 3 4

Все числа сразу

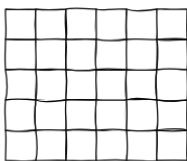
Тензор описывается двумя вещами: типом его элементов и его формой. Книга в основном использует тензоры чисел с плавающей точкой, чисел с десятичной запятой, точный формат которых она обсуждает позже; но тензоры могут содержать и

целые числа, например номера классов в классификации, или строки, для текста.

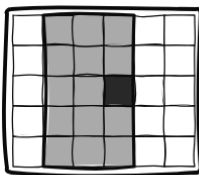
Тензоры можно индексировать, чтобы брать срезы, то есть подмножества их значений, и большинство соглашений библиотеки Python под названием NumPy здесь применимы.

Одно отличие: для удобства чтения книга отсчитывает позиции с 1, а не с 0, как делает NumPy. Для простых выражений она пишет индексы маленькими подстрочными знаками. Возьмём трёхмерный тензор $X \sim (a, b, c)$. Тогда X_i — один срез формы (b, c) : в нашем батче фотографий X_2 — вся вторая фотография, сетка 5 на 6 из 30 чисел. А X_{ijk} — один скаляр: X_{234} — яркость пикселя в строке 3, столбце 4 второй фотографии, одна крошка батона.

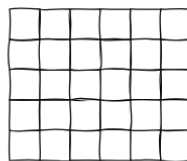
$$X \sim (a, b, c)$$



ЛИСТ1



ЛИСТ2



ЛИСТ3

$$X_2 : 5 \times 6 = 30$$

$$X_{234} : \text{строка 3, столбец 4}$$

$$X_{2,:;2:5} \rightarrow 5 \times 3$$

$$[X]_{ijk}$$

Режем батч фотографий

Для более сложных выборок книга разделяет индексы запятыми, как в $X_{i,:j:k}$. Двоеточие само по себе означает «взять всё вдоль этого измерения», а j двоеточие k означает «диапазон позиций», так что результат имеет форму $(b, k - j)$. С нашими фотографиями $X_{2,:,2:5}$ берёт вторую фотографию, все 5 строк и полосу столбцов; её форма — 5 на 5 минус 2, то есть 5 на 3, полоска из 15 чисел. Когда выражение стало бы загромождённым, книга записывает индексацию в светло-серых скобках, $[X]_{i:j:k}$, чтобы индексы отделялись от остального, а внутри скобок может стоять и целое выражение.

Опыт

Нарисуйте на бумаге сетку из 5 строк и 6 столбцов и напишите число в каждой клетке; это одна фотография, а ещё два таких листа дают батч формы 3 на 5 на 6. Укажите на второй лист: это икс два, 30 чисел. Укажите на нём на строку 3, столбец 4: это икс два три четыре, одно число. Затем заштрихуйте на этом листе все строки столбцов 2, 3 и 4: вы вырезали полосу 5 на 3, тот вид среза, который записывается двоеточиями.

Запомним

Книга благодарит своих помощников и выпущена под лицензией CC BY-SA, которая разрешает адаптировать и продавать, пока указан автор, а результат сохраняет ту же лицензию. Вторая глава собирает математику, нужную дальше: линейную алгебру, градиенты во многих измерениях и оптимизацию с помощью градиентов, а вероятность вынесена в приложение. Тензор — это n -мерный массив элементов одного типа с формой $X \sim (s_1, \dots, s_n)$; тензоры подходят для параллельного оборудования, такого как GPU. Индексация следует NumPy, но отсчитывает с 1: X_i — срез, $X_{i j k}$ — одно число, двоеточия выбирают целые измерения или диапазоны, а серые скобки отделяют индексы от остального.

Вопросы

1. Для тензора $X \sim (a, b, c)$ в чём разница между X_i и $X_{i j k}$?
 - a. X_i — целый срез формы (b, c) , а $X_{i j k}$ — одно число
 - b. X_i — одно число, а $X_{i j k}$ — срез формы (a, b, c)
 - c. Оба — срезы, но в $X_{i j k}$ вдвое больше чисел
2. Почему тензоры так хорошо подходят для глубокого обучения?
 - a. Потому что они хранят числа точнее других форматов

- b. Потому что они могут содержать только числа с плавающей точкой
 - c. Потому что оборудование вроде GPU может работать со всеми их числами параллельно
- 3. Чего лицензия книги CC BY-SA требует от адаптации, такой как этот курс?
 - a. Чтобы она никогда не продавалась
 - b. Чтобы она указывала автора и выпускалась под той же лицензией
 - c. Чтобы она сохраняла исходный текст без изменений
- 4. Чем индексация в книге отличается от NumPy?
 - a. Она отсчитывает позиции с 1, а не с 0
 - b. Она запрещает двоеточия для диапазонов
 - c. Она всегда записывает индексы в серых скобках

Ответы

- 1. a) Фиксация одного индекса оставляет два других измерения, срез формы (b, c); фиксация всех трёх выбирает один скаляр, как яркость одного пикселя одной фотографии.
- 2. c) Тензоры подходят для массово-параллельных реализаций на GPU и более специализированных чипах, которые обрабатывают много чисел одновременно.
- 3. b) Лицензия разрешает распространять, перерабатывать, адаптировать и использовать коммерчески, пока указан автор, а адаптированный материал несёт те же условия.
- 4. a) Большинство соглашений NumPy применимы, но для удобства чтения книга индексирует с 1, а не с 0; серые скобки используются

только когда выражение стало бы загромождённым.

Разобранные примеры

Формы срезов Батч серых фотографий имеет форму $X \sim (4, 5, 6)$. Назовите форму X_2 , форму $X_{2,:,2:5}$ (которая имеет форму $(b, k - j)$) и скажите, что такое X_{234} .

Здесь $a = 4$ фотографии, $b = 5$ строк, $c = 6$ столбцов.

X_2 фиксирует первый индекс, оставляя форму $(b, c) = (5, 6)$: вся вторая фотография, 30 чисел.

$X_{2,:,2:5}$ фиксирует фотографию, оставляет все $b = 5$ строк и берёт столбцы от $j = 2$ до $k = 5$, так что её форма $(b, k - j) = (5, 3)$, 15 чисел.

X_{234} фиксирует все три индекса, так что это одно число: строка 3, столбец 4 второй фотографии.

Ответ: X_2 имеет форму $(5, 6)$; $X_{2,:,2:5}$ имеет форму $(5, 3)$; X_{234} — один скаляр.

Задачи

1. Тензор имеет форму $(10, 8, 12)$. Сколько чисел он содержит и сколько содержит один срез X_i ?

Подсказка: Перемножьте все размеры для целого тензора; отбросьте первый для среза.

Ответ: Он содержит $10 \times 8 \times 12 = 960$ чисел; один срез X_i имеет форму $(8, 12)$ и содержит 96.

2. Для $X \sim (10, 8, 12)$ какова форма $X_{3,:,4:10}$?

Подсказка: Используйте правило книги: срез $X_{i,:,j:k}$ имеет форму $(b, k - j)$.

Ответ: $(8, 6)$: все 8 позиций среднего измерения и $10 - 4 = 6$ позиций последнего.

3. Объясните в нескольких предложениях, какие две вещи описывают тензор, с примером каждой из лекции.

Подсказка: Одна о том, что такое элементы, другая о том, как они расположены.

Ответ: Тензор описывается типом его элементов, обычно это числа с плавающей точкой, но возможны и целые числа, например номера классов, или строки для текста, и его формой, списком размеров его измерений, например (4, 5, 6) для четырёх фотографий по 5 на 6 пикселей.

Стрелки, длины и совпадение

Встаньте в комнате и укажите рукой на дверь, а друга попросите указать на окно. Если вы оба указываете в одну сторону, ваши руки полностью согласны; если в противоположные стороны, они полностью не согласны; если ваши руки образуют прямой угол, у них нет ничего общего. Одно число может измерить это согласие, и это самая важная операция всего нашего курса: скалярное произведение.

СКАЛЯР. ПРОИЗВ.


СОГЛАСНЫ


ПРОТИВ


НИЧЕГО

$$\begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \square \\ \hline \square \\ \hline \end{array} \mathbf{x} \begin{pmatrix} d, 1 \end{pmatrix} \quad \mathbf{x}^T \begin{pmatrix} 1, d \end{pmatrix} \quad \begin{array}{|c|c|c|c|} \hline \square & \square & \square & \square \\ \hline \end{array} \quad (d) \neq (d, 1)$$

Согласны, не согласны, ничего общего

Книга начинается с замечания, что большинство её моделей — композиции дифференцируемых операций, причём в основном простых: сумм, умножений и нескольких нелинейных функций, таких как экспонента $\exp(x)$, синусы и косинусы и квадратные корни. Векторы, записываемые как $x \sim (d)$, — это тензоры с одним измерением. Книжки по линейной алгебре тщательно различают векторы-столбцы x и векторы-строки x^T , и книга старается следовать этому соглашению. В коде всё хитрее: строка или столбец — это двумерный тензор формы $(1, d)$ или $(d, 1)$, а это не то же самое, что одномерный тензор формы (d) .

Эта разница важна, потому что большинство библиотек следуют правилам трансляции, заимствованным из NumPy. В двух словах, трансляция выравнивает две формы по правому краю и повторяет тензор везде, где может, чтобы формы совпали. Блок кода в книге показывает очень частую ошибку, которая из этого выходит: прибавьте «столбец» формы 4 на 1 к простому вектору формы 4, ожидая четыре суммы, и получите вместо этого матрицу 4 на 4. Столбец повторяется по четырём столбцам, а вектор по четырём строкам, и складывается каждая пара, давая шестнадцать чисел там, где вы хотели четыре. То же самое происходит в NumPy, TensorFlow, JAX и практически любой библиотеке, так что формы приходится проверять.

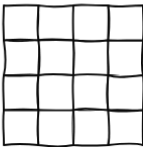
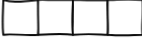
$$\mathbf{z} = a\mathbf{x} + b\mathbf{y}$$

(4, 1)



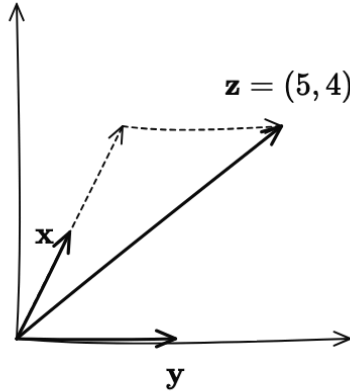
+

(4)



$4 \times 4 = 16$, а не 4

**ПРОВЕРЯЙ
ФОРМЫ**

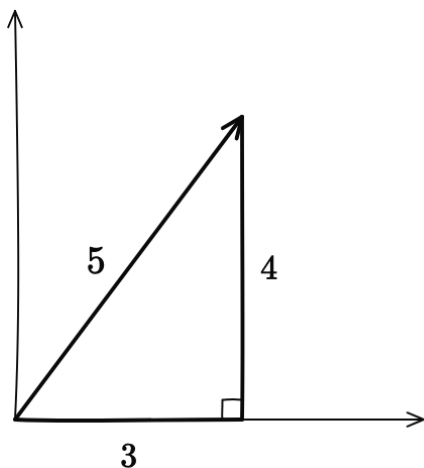


Трансляция и линейная комбинация

У векторов своя алгебра, называемая векторным пространством: любые два вектора одной формы можно линейно скомбинировать в третий, $\mathbf{z} = a\mathbf{x} + b\mathbf{y}$. Поэлементно, $z_i = ax_i + by_i$: каждая компонента результата — это a на компоненту x плюс b на компоненту y . Откуда это берётся — просто делаем одно и то же в каждой позиции. При $\mathbf{x} = (1, 2)$, $\mathbf{y} = (3, 0)$, a , равном 2, и b , равном 1, получаем 2 на 1 плюс 3, то есть 5, и 2 на 2 плюс 0, то есть 4: вектор $(5, 4)$.

Если представить вектор как точку, или стрелку из начала координат, в d -мерном пространстве, то сложение двух векторов означает достроить параллелограмм, а расстояние вектора от начала координат — это его евклидова норма, называемая также

ℓ_2 -нормой: $\|\mathbf{x}\| = \sqrt{(\sum_i x_i^2)}$. Словами: возведите каждую компоненту в квадрат, сложите их и извлеките квадратный корень. Это идёт от Пифагора: в двух измерениях стрелка — это длинная сторона прямоугольного треугольника, другие стороны которого — две компоненты. Для $\mathbf{x} = (3, 4)$: 9 плюс 16 равно 25, а корень из него 5. Квадрат нормы $\|\mathbf{x}\|^2$ особенно удобен, потому что это просто сумма квадратов компонент, здесь 25, без извлечения корня.



$$\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$$

$$\sqrt{9 + 16} = 5$$

$$\|\mathbf{x}\|^2 = 25$$

Длина по Пифагору

Теперь центральная операция, внутреннее произведение, называемое также скалярным: перемножьте два вектора покомпонентно и сложите результаты. Определение в книге читается так: $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_i x_i y_i$. Угловые скобки пришли из физики, и книга время от времени использует их для ясности;

важно то, что результат — одно число, скаляр. Собственный пример книги: при $\mathbf{x} = [0.1, 0, -0.3]$ и $\mathbf{y} = [-4.0, 0.05, 0.1]$ произведения равны минус 0,4, затем 0, затем минус 0,03, а в сумме они дают минус 0,43.

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = \sum_i x_i y_i$$

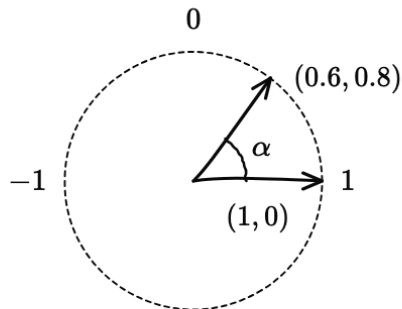
$$0.1 \times (-4.0) = -0.4$$

$$0 \times 0.05 = 0$$

$$-0.3 \times 0.1 = -0.03$$

$$\text{сумма} = -0.43$$

$$\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \alpha$$



$$\cos \alpha = 0.6$$

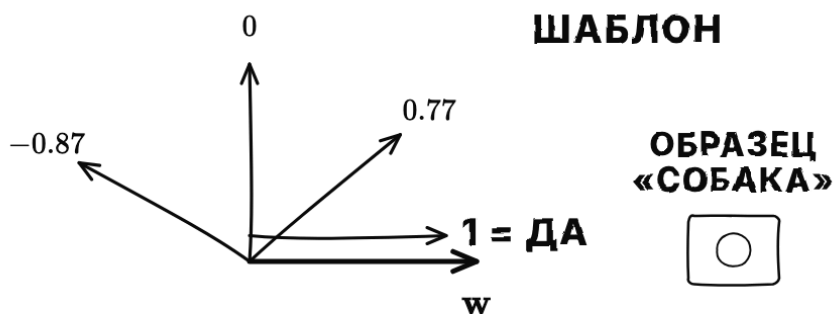
Скалярное произведение двумя способами

У скалярного произведения простой геометрический смысл через угол α между двумя стрелками: $\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos(\alpha)$. Словами: это две длины, перемноженные между собой, умноженные на множитель, который говорит, насколько стрелки смотрят в одну сторону. Для векторов, нормированных на длину один, длины выпадают, и скалярное произведение — это просто косинус угла, который тогда называют косинусным сходством. Оно меняется от 1, стрелки смотрят в одну сторону, до минус 1, стрелки смотрят в

противоположные стороны, и равно 0, когда они ортогональны, под прямым углом: снова ваша рука и рука вашего друга. Проверьте один случай: единичные стрелки $(1, 0)$ и $(0,6, 0,8)$ имеют скалярное произведение 0,6, и действительно, у второй длина 1, поскольку $0,36$ плюс $0,64$ равно 1, так что косинус угла между ними равен 0,6.

Перевернём вопрос. Зафиксируем единичный вектор x и спросим, какой единичный вектор y даёт наибольшее скалярное произведение: $y^* = \operatorname{argmax} \langle x, y \rangle = x$. Здесь аргмакс означает «то значение y , при котором выход как можно больше», и ответ — y , равный самому x , поскольку косинус достигает 1 только при нулевом угле. Вот идея, которую книга хочет посеять: позже x будет входом, а w — набором настраиваемых параметров, и скалярное произведение наибольшее, когда вход резонирует с параметрами, что называется сопоставлением с шаблоном. Ваше фотоприложение по духу работает именно так: оно сравнивает числа фотографии с выученным образцом «собака», и большое скалярное произведение означает, что фотография совпадает с образцом.

$$\mathbf{y}^* = \arg \max_y \langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}$$



$$(2, 5, 1) \cdot (1, 1, 1) = 8 = \text{сумма}$$

Лучшее совпадение — сам шаблон

Книга завершает наблюдением, которое пригодится позже: сумму компонент вектора саму можно записать как скалярное произведение с вектором **1**, все компоненты которого — единицы; умножение на единицу и сложение просто суммирует компоненты.

Опыт

Возьмите два карандаша и держите оба остриём в потолок: они согласны, и их косинусное сходство равно 1. Поверните один остриём в пол: косинус минус 1. Положите один плашмя, пока другой всё ещё смотрит вверх: они образуют прямой угол, и скалярное произведение их направлений равно 0. Каждый промежуточный угол даёт промежуточное число, и это всё, что измеряет скалярное произведение, когда длины положены равными единице.

Запомним

Векторы можно линейно комбинировать покомпонентно, $z_i = ax_i + by_i$, а их длина — евклидова норма, квадратный корень из суммы квадратов компонент, 5 для (3, 4). В коде строка или столбец формы (1, d) или (d, 1) отличается от простого вектора формы (d), и трансляция может молча превратить сумму двух векторов в матрицу 4 на 4. Скалярное произведение перемножает компоненты и складывает их, давая одно число, минус 0,43 в примере книги, и оно равно произведению двух длин на косинус угла между векторами. Для единичных векторов скалярное произведение — это косинусное сходство, от 1 до минус 1 с нулём для прямых углов, и оно наибольшее, когда два вектора совпадают, вот почему вектор параметров w работает как шаблон, с которым совпадают входы.

Вопросы

1. В библиотеке вы прибавляете тензор формы $(4, 1)$ к тензору формы (4) . Почему получается результат 4×4 ?
 - a. Потому что трансляция повторяет столбец по столбцам, а вектор по строкам, и складывает каждую пару
 - b. Потому что библиотека всегда превращает суммы векторов в матрицы
 - c. Потому что тензор формы (4) втайне является матрицей
2. Два вектора единичной длины имеют скалярное произведение 0. Что это о них говорит?
 - a. Они смотрят в одну сторону
 - b. Они смотрят в противоположные стороны
 - c. Они расположены под прямым углом друг к другу
3. Зафиксировав единичный вектор x , какой единичный вектор y делает скалярное произведение как можно большим, и почему это важно позже?
 - a. Любой вектор, ортогональный x , потому что ортогональные векторы независимы
 - b. y , равный x , так что вектор параметров w работает как шаблон, с которым резонируют совпадающие входы
 - c. y , равный минус x , так что обнаруживаются входы, противоположные параметрам
4. Почему квадрат нормы часто удобнее самой нормы?
 - a. Потому что он всегда меньше нормы

- б. Потому что это просто сумма квадратов компонент, без квадратного корня
- с. Потому что он не зависит от компонент вектора

Ответы

1. а) Трансляция выравнивает формы по правому краю и повторяет тензоры, чтобы они совпали, так что столбец плюс одномерный вектор становится матрицей 4 на 4 вместо четырёх сумм.
2. с) Для единичных векторов скалярное произведение — это косинус угла между ними; оно равно 0 ровно тогда, когда они ортогональны, под прямым углом.
3. б) Косинус достигает 1 только при нулевом угле, так что максимум — при $u = x$; если x — вход, а w — параметры, скалярное произведение наибольшее, когда вход совпадает с шаблоном w .
4. б) Норма — это квадратный корень из суммы квадратов компонент, так что её квадрат — просто эта сумма, 25 для вектора (3, 4).

Разобранные примеры

Скалярное произведение из книги Вычислите внутреннее произведение $x = [0.1, 0, -0.3]$ и $y = [-4.0, 0.05, 0.1]$.

Перемножьте покомпонентно: $0.1 \times (-4.0) = -0.4$.

$0 \times 0.05 = 0$.

$-0.3 \times 0.1 = -0.03$.

Сложите произведения: $-0.4 + 0 - 0.03 = -0.43$, одно число.

Ответ: -0.43

Сюрприз трансляции x имеет форму (4, 1), «столбец», а y имеет форму (4), одномерный тензор. Какова форма $x + y$ и

сколько чисел в ней?

Трансляция выравнивает формы по правому краю: $(4, 1)$ против (4) .

Размер 1 в x повторяется до 4, а y повторяется вдоль нового первого измерения размера 4.

Каждая компонента x складывается с каждой компонентой y .

Ответ: Форма $(4, 4)$, 16 чисел, вместо 4 сумм, которых, вероятно, хотели.

Задачи

1. Найдите норму $x = (6, 8)$ и квадрат её нормы.

Подсказка: Возведите каждую компоненту в квадрат и сложите.

Ответ: $36 + 64 = 100$, так что квадрат нормы равен 100, а норма равна $\sqrt{100} = 10$.

2. Вычислите $z = 3x - 2y$ для $x = (1, 4)$ и $y = (2, 1)$.

Подсказка: Используйте $z_i = ax_i + by_i$ при $a = 3$, $b = -2$.

Ответ: $z_1 = 3 \times 1 - 2 \times 2 = -1$ и $z_2 = 3 \times 4 - 2 \times 1 = 10$, так что $z = (-1, 10)$.

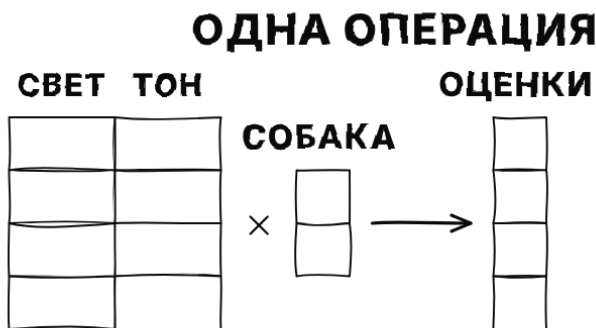
3. Единичные векторы $(1, 0)$, $(0, 8, 0, 6)$ и $(0, 1)$ сравниваются с $x = (1, 0)$. Упорядочите их по косинусному сходству с x .

Подсказка: Для единичных векторов косинусное сходство — это просто скалярное произведение.

Ответ: $(1, 0)$ даёт 1, $(0, 8, 0, 6)$ даёт 0,8, $(0, 1)$ даёт 0; так что порядок таков: $(1, 0)$, затем $(0, 8, 0, 6)$, затем $(0, 1)$, который ортогонален x .

Матрицы как батчи и как функции

Откройте электронную таблицу и впишите в неё признаки четырёх фотографий, по одной фотографии на строку: яркость, насыщенность и так далее по столбцам. Теперь вы хотите оценить каждую фотографию по одному и тому же образцу «собака»: умножить каждую строку на образец и сложить. Можно делать это строка за строкой, четыре раза, или одной-единственной операцией над всем листом. Эта единственная операция — умножение матриц, и эта часть показывает, почему оно — рабочая лошадка каждой модели.

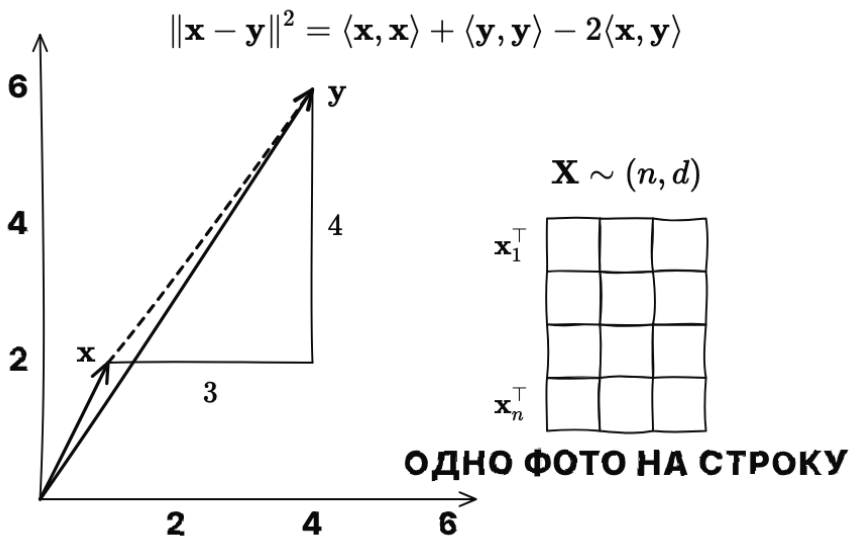


$$\langle \mathbf{x}, \mathbf{1} \rangle = (2, 5, 3) \cdot (1, 1, 1) = 10$$

Оценить все строки сразу

Сначала два последних факта о векторах, которыми книга будет часто пользоваться. Сумма компонент вектора — это его скалярное произведение с вектором из единиц: $\langle \mathbf{x}, \mathbf{1} \rangle = \sum_{i=1}^d x_i$. Словами: умножить каждую компоненту на единицу и сложить — значит просто сложить их. Для $\mathbf{x} = (2, 5, 3)$ это 2 плюс 5 плюс 3, то есть 10.

Второе: квадрат расстояния между двумя векторами можно записать одними скалярными произведениями: $\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle$. Это получается из раскрытия квадрата, совсем как школьное правило: a минус b в квадрате равно a в квадрате плюс b в квадрате минус два a b . Проверьте на $\mathbf{x} = (1, 2)$ и $\mathbf{y} = (4, 6)$: напрямую разности равны 3 и 4, а 9 плюс 16 равно 25. Через скалярные произведения: $\mathbf{x}$ с самим собой даёт 5, $\mathbf{y}$ с самим собой 52, $\mathbf{x}$ с $\mathbf{y}$ 16, и 5 плюс 52 минус 32 — снова 25. Если взять $\mathbf{y}$ нулевым вектором, получится частный случай $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$: квадрат длины вектора — это его скалярное произведение с самим собой.



Расстояние через скалярные произведения, затем батч

Теперь к двум измерениям. Матрица $\mathbf{X} \sim (n, d)$ имеет n строк и d столбцов, с элементами от X_{11} в левом верхнем углу до X_{nd} в правом нижнем. Для дальнейшего полезнее всего видеть её как стопку из n векторов $\mathbf{x}_1, \dots, \mathbf{x}_n$, выложенных строками, где каждая строка — это $\mathbf{x}_i^T$. Затем книга говорит, что $\mathbf{X}$ представляет батч векторов данных: нашу таблицу, по одной фотографии на строку. Модели по обыкновению определяются, и на бумаге, и в коде, так, чтобы работать с такими батчами.

Фундаментальная операция — умножение матриц. Для матриц согласованных форм, $\mathbf{X} \sim (a, b)$ и $\mathbf{Y} \sim (b, c)$, произведение $\mathbf{Z} = \mathbf{XY}$ имеет форму (a, c) , и каждый его элемент равен $Z_{ij} = \langle \mathbf{X}_i, \mathbf{Y}_j^T \rangle$. Словами: элемент в строке i и столбце j — это скалярное

произведение строки i первой матрицы со столбцом j второй, вот почему внутренние размеры, оба равные b , должны совпадать. Возьмём строки (1, 2) и (3, 4) как $\mathbf{X}$ и строки (5, 6) и (7, 8) как $\mathbf{Y}$. Левый верхний элемент — это 1 на 5 плюс 2 на 7, то есть 19; правый верхний — 1 на 6 плюс 2 на 8, то есть 22; левый нижний — 3 на 5 плюс 4 на 7, то есть 43; правый нижний — 3 на 6 плюс 4 на 8, то есть 50.

$$(a, b) (b, c) \rightarrow (a, c)$$

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

$$1 \cdot 5 + 2 \cdot 7 = 19$$

$$\mathbf{W}\mathbf{x} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 3 \\ 10 \end{pmatrix} \quad \mathbf{X}\mathbf{X}^T = \begin{pmatrix} 5 & 11 \\ 11 & 25 \end{pmatrix}$$

Строка на столбец

Когда второй множитель — один вектор, получается произведение матрицы на вектор, $\mathbf{z} = \mathbf{W}\mathbf{x}$: каждая компонента z — скалярное произведение одной строки $\mathbf{W}$ с $\mathbf{x}$. При $\mathbf{W}$ со строками (1, 0) и (2, 1) и $\mathbf{x}$, равном (3, 4), получаем 3 и 10. В терминах батчей $\mathbf{X}\mathbf{W}^T$ вычисляет одной операцией линейной

алгебры скалярное произведение каждой строки X с каждым образцом в W : наша таблица, оценённая по образцу «собака», все фотографии сразу. А $XX^T \sim (n, n)$ вычисляет сразу все скалярные произведения между парами строк: для строк (1, 2) и (3, 4) это даёт 5 и 11 в первой строке и 11 и 25 во второй, каждая фотография сравнена с каждой другой.

Важны ещё несколько операций. Произведение Адамара перемножает две матрицы одной формы поэлементно: $[X \odot Y]_{i,j} = X_{i,j} Y_{i,j}$. Ему не хватает богатой алгебры настоящего умножения матриц, но дифференцируемые модели постоянно используют его для маскирования, обнуления некоторых элементов, и для масштабирования. Перемножьте строки (1, 2) и (3, 4) поэлементно с маской, строки которой (1, 0) и (0, 1), и вы сохраните 1 и 4, а остальное обнулите. Мультипликативные взаимодействия такого рода стали популярны и в некоторых недавних семействах моделей, к которым книга вернётся позже.

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

МАСКА

$$[\exp(\mathbf{X})]_{ij} = \exp(X_{ij}), \quad \exp(0) = 1$$

$$\text{mat-exp} : 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} \approx 2.708 \approx e$$

ТОЛЬКО КВАДРАТНЫЕ

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \xrightarrow{\Sigma_j} \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

СУММА ВДОЛЬ ОСИ

Поэлементно и вдоль оси

Выражение вроде $\exp(\mathbf{X})$ понимается поэлементно: $[\exp(\mathbf{X})]_{ij} = \exp(X_{ij})$, экспонента берётся от каждого элемента отдельно, так что нулевой элемент становится единицей. Сравните с настоящей матричной экспонентой из линейной алгебры, $\text{mat-exp}(\mathbf{X}) = \sum_{k=0}^{\infty} \mathbf{X}^k / k!$, суммой степеней матрицы, каждая делённая на k факториал. Для матрицы один на один, содержащей число 1, первые члены — это 1 плюс 1 плюс одна вторая плюс одна шестая плюс одна двадцать четвёртая, примерно 2,708, уже близко к e , примерно 2,718. Поэлементная версия работает для тензоров любой формы, а матричная экспонента имеет смысл только для квадратных матриц; вот почему библиотеки вроде PyTorch держат такие чисто матричные операции, а также обратные матрицы и определители, в отдельном модуле линейной

алгебры, как показывает блок кода в книге с двумя экспонентами.

Редукции, такие как суммы и средние вдоль оси, записываются без пределов, когда идут по всей оси, $\sum_i X_i$, а в коде принимают аргумент `axis`. Суммирование строк (1, 2) и (3, 4) вдоль второй оси даёт по одному итогу на строку, 3 и 7.

Почему умножение матриц определено таким странным способом, строка на столбец, а не поэлементно, как у Адамара? Ответ книги: функция f от векторов линейна, если $f(\alpha x_1 + \beta x_2) = \alpha f(x_1) + \beta f(x_2)$, и каждую такую функцию можно записать как матрицу A . Тогда произведение матрицы на вектор Ax — это применение функции, а произведение матриц AB — композиция двух функций: $(f \circ g)(x) = f(g(x))$ при $g(x) = Bx$. Проверьте: пусть у B строки (1, 1) и (0, 1), а у A строки (2, 0) и (0, 3); для x , равного (1, 1), B даёт (2, 1), а затем A даёт (4, 3). Произведение AB имеет строки (2, 2) и (0, 3), и применённое к (1, 1) оно даёт (4, 3) напрямую: выполнить одну функцию после другой — то же самое, что сначала перемножить их матрицы.

$$f(\alpha \mathbf{x}_1 + \beta \mathbf{x}_2) = \alpha f(\mathbf{x}_1) + \beta f(\mathbf{x}_2)$$

$$(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$$

$$(1, 1) \xrightarrow{\mathbf{B} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}} (2, 1) \xrightarrow{\mathbf{A} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}} (4, 3)$$

=

$$(1, 1) \xrightarrow{\mathbf{AB} = \begin{pmatrix} 2 & 2 \\ 0 & 3 \end{pmatrix}} (4, 3)$$

Умножение матриц составляет функции

Опыт

На бумаге в клетку запишите слева матрицу со строками (1, 2) и (3, 4), а справа матрицу со строками (5, 6) и (7, 8).

Проведите пальцем вдоль первой строки левой матрицы и вниз по первому столбцу правой, умножая и складывая: должно получиться 19. Сделайте остальные три элемента и проверьте 22, 43 и 50; вы сделали руками то, что чип телефона делает миллиарды раз, когда размечает ваши фотографии.

Запомним

Сумма компонент вектора — это его скалярное произведение с вектором из единиц, а квадрат расстояния между двумя векторами можно записать скалярными произведениями, $\|x - y\|^2 = \langle x, x \rangle + \langle y, y \rangle - 2\langle x, y \rangle$. Матрицу лучше всего видеть как батч векторов-строк, а элемент Z_{ij} произведения XY — это скалярное произведение строки i матрицы X со столбцом j матрицы Y , как 19, 22, 43, 50 в нашем примере. XW^T оценивает каждую строку батча по каждому образцу сразу, а XX^T сравнивает все пары строк; произведение Адамара перемножает поэлементно и используется для маскирования и масштабирования. Функции вроде `exp` действуют на тензоры поэлементно, в отличие от настоящей матричной экспоненты квадратных матриц; а умножение матриц определено именно так, потому что оно составляет композицию линейных функций.

Вопросы

1. В произведении матриц $Z = XY$ что такое элемент Z_{ij} ?
 - a. Произведение элементов X_{ij} и Y_{ij}
 - b. Скалярное произведение строки i матрицы X со столбцом j матрицы Y
 - c. Сумма строки i матрицы X и столбца j матрицы Y

2. Почему умножение матриц определено как строка на столбец, а не поэлементно?
- a. Потому что тогда умножение матриц составляет композицию линейных функций, которые они представляют
 - b. Потому что его быстрее вычислять, чем поэлементные произведения
 - c. Потому что поэлементные произведения не допускаются для квадратных матриц
3. Если X — батч фотографий, по одной на строку, что даёт XX^T ?
- a. Сумму всех фотографий
 - b. Каждую фотографию, умноженную поэлементно на саму себя
 - c. Все скалярные произведения между парами фотографий в виде матрицы n на n
4. Почему библиотеки держат матричную экспоненту в отдельном модуле линейной алгебры, тогда как $\exp$ работает с любым тензором?
- a. Потому что матричная экспонента — лишь приближение $\exp$
 - b. Потому что поэлементная $\exp$ работает для тензоров любой формы, а матричная экспонента определена только для квадратных матриц
 - c. Потому что $\exp$ вообще нельзя применять к матрицам

Ответы

1. b) Каждый элемент — это скалярное произведение строки i первой матрицы со столбцом j второй, поэтому внутренние размеры должны совпадать; поэлементное умножение — это произведение Адамара.

2. a) Каждая линейная функция — это матрица; её применение — произведение матрицы на вектор, а произведение матриц AB — это функция, которая сначала применяет $g = B$, а затем $f = A$.

3. c) XX^T имеет форму (n, n) , и его элемент i, j — это скалярное произведение строк i и j : каждая фотография сравнена с каждой другой за одну операцию.

4. b) $\exp(X)$ применяет экспоненту к каждому элементу, что годится для любой формы, а настоящая матричная экспонента суммирует степени матрицы, что имеет смысл только для квадратных матриц.

Разобранные примеры

Перемножаем две маленькие матрицы Вычислите XY для X со строками $(1, 2)$ и $(3, 4)$ и Y со строками $(5, 6)$ и $(7, 8)$.

Левый верхний: строка 1 матрицы X со столбцом 1 матрицы Y , $1 \times 5 + 2 \times 7 = 19$.

Правый верхний: строка 1 со столбцом 2, $1 \times 6 + 2 \times 8 = 22$.

Левый нижний: строка 2 со столбцом 1, $3 \times 5 + 4 \times 7 = 43$.

Правый нижний: строка 2 со столбцом 2, $3 \times 6 + 4 \times 8 = 50$.

Ответ: XY имеет строки $(19, 22)$ и $(43, 50)$.

Расстояние через скалярные произведения Для $x = (1, 2)$ и $y = (4, 6)$ вычислите $\|x - y\|^2$ напрямую и через $\langle x, x \rangle + \langle y, y \rangle - 2\langle x, y \rangle$.

Напрямую: $x - y = (-3, -4)$, и $9 + 16 = 25$.

$\langle x, x \rangle = 1 + 4 = 5$ и $\langle y, y \rangle = 16 + 36 = 52$.

$\langle x, y \rangle = 4 + 12 = 16$, так что $2\langle x, y \rangle = 32$.

$5 + 52 - 32 = 25$, то же, что и при прямом вычислении.

Ответ: 25 обоими способами

Задачи

1. Вычислите Wx для W со строками $(2, 1)$ и $(0, 3)$ и $x = (4, 5)$.

Подсказка: Каждая компонента — скалярное произведение одной строки W с x .

Ответ: Строка 1: $2 \times 4 + 1 \times 5 = 13$; строка 2: $0 \times 4 + 3 \times 5 = 15$; так что $Wx = (13, 15)$.

2. Замаскируйте матрицу со строками $(5, 6)$ и $(7, 8)$ произведением Адамара с маской, строки которой $(0, 1)$ и $(1, 0)$. Что останется?

Подсказка: Перемножьте поэлементно.

Ответ: Строки $(0, 6)$ и $(7, 0)$: элементы, где маска равна 1, выживают, остальные становятся 0.

3. Пусть $g(x) = Vx$, где у V строки $(1, 0)$ и $(1, 1)$, а $f(x) = Ax$, где у A строки $(3, 0)$ и $(0, 2)$. Вычислите $f(g(x))$ для $x = (2, 1)$ двумя способами: одна функция после другой и через одну матрицу AB .

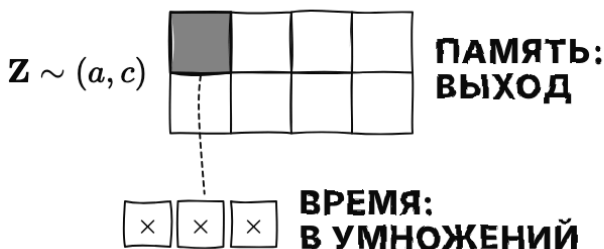
Подсказка: Сначала Vx , затем A на результат; потом вычислите AB строка на столбец.

Ответ: $Vx = (2, 3)$, затем A даёт $(6, 6)$. AB имеет строки $(3, 0)$ и $(2, 2)$, и AB на $(2, 1)$ равно $(6, 6)$: то же самое, потому что произведение матриц составляет композицию функций.

Как растёт стоимость

Засеките, сколько времени ваш телефон размечает один маленький альбом, а потом представьте альбом в десять раз шире в каждом направлении. Вы могли бы предположить, что в десять раз дольше, но для умножения матриц ответ — примерно в тысячу раз дольше. Подсчёт того, как работа растёт с размером, ещё до того, как вы что-либо запустили, — тема этой части.

10× ШИРЕ → 1000× РАБОТЫ



$$a = 2, b = 3, c = 4 : \quad 8 \times 3 = 24$$

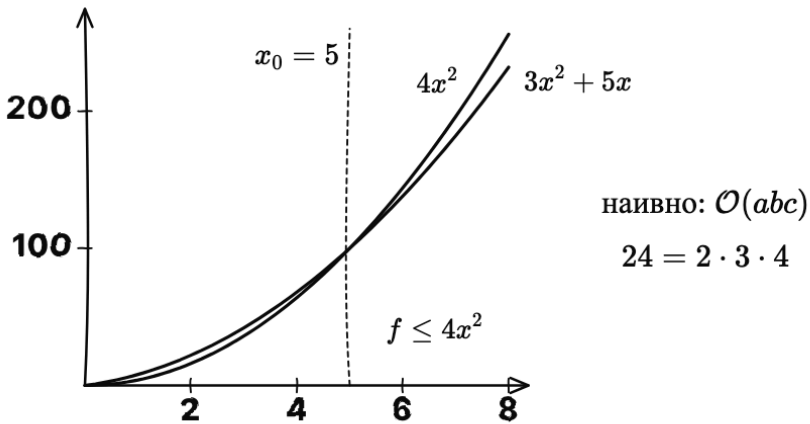
Считаем работу

Книга использует умножение матриц, чтобы ввести понятие сложности операции. Чтобы вычислить $Z \sim (a, c)$ из $X \sim (a, b)$ и Y

$\sim (b, c)$ прямо по определению, нам нужно a скалярных произведений, по одному на элемент Z , каждое длины b . Этот подсчёт работы называется временной сложностью. Память, которая нужна простой последовательной реализации, — это всего лишь размер выходной матрицы, и это называется пространственной сложностью. При a , равном 2, b , равном 3, и c , равном 4, у Z 8 элементов, каждый — скалярное произведение из 3 умножений, итого 24 умножения и 8 чисел для хранения.

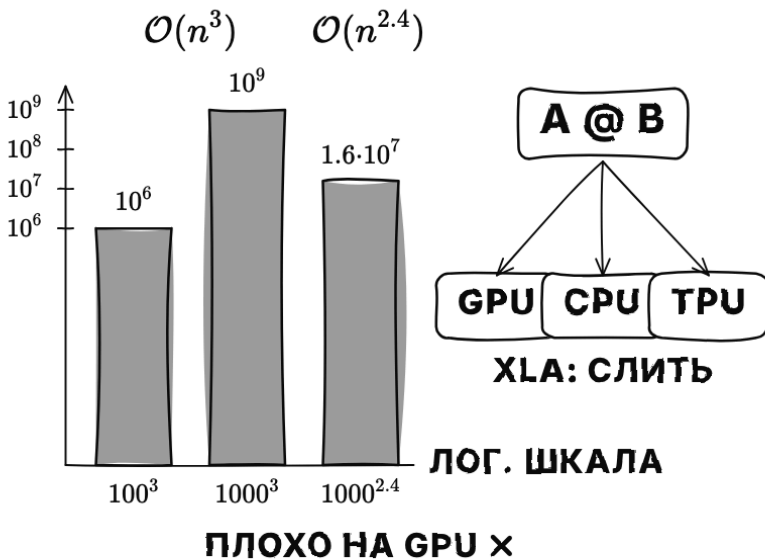
Чтобы уйти от деталей конкретного оборудования, информатика использует нотацию «О большое», от немецкого слова *Ordnung*, порядок приближения. Определение: функция $f(x)$ есть $O(g(x))$, где входы и выходы считаются неотрицательными, если существуют константа c и начальное значение x_0 такие, что $f(x) \leq c g(x)$ при любом $x \geq x_0$. Словами: как только x достаточно велик, f никогда не превышает фиксированного кратного g , так что для больших входов всё, кроме g , можно игнорировать. Это называется асимптотическим анализом. Например, $f(x) = 3x^2 + 5x$ есть $O(x^2)$: возьмите c , равное 4, и x ноль, равное 5, потому что три икс в квадрате плюс пять икс не больше четырёх икс в квадрате ровно тогда, когда пять икс не больше икс в квадрате, а это верно начиная с 5.

$$3x^2 + 5x \leq 4x^2 \iff 5x \leq x^2$$



О большое: в итоге ниже с·g

Так что наивное умножение матриц есть $\mathcal{O}(abc)$: работа растёт пропорционально каждому из трёх размеров, как показывает наше 24 равно 2 на 3 на 4. Для двух квадратных матриц размера (n, n) это $\mathcal{O}(n^3)$, кубическая зависимость от размера. При n , равном 100, это миллион умножений; при n , равном 1000, миллиард: в десять раз шире, в тысячу раз больше работы, то число, с которого мы начали.

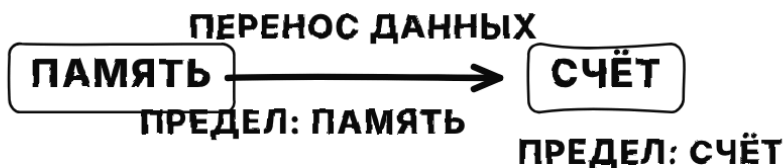
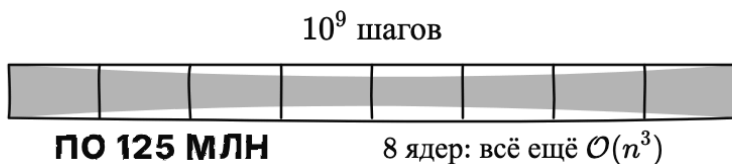


Кубическая стоимость и кто её выполняет

Асимптотическое мышление важно и элегантно, говорит книга, но выбор алгоритма по одному лишь «О большому» не гарантирует реальной скорости, которая зависит от оборудования, от того, сколько параллельной работы оно поддерживает, и от многого другого. Когда вы вызываете умножение матриц в библиотеке, записываемое как **A @ B** в PyTorch, низкоуровневая процедура, которая действительно выполняется, называемая ядром, выбирается во время выполнения под ваше оборудование, этот процесс называется диспетчеризацией; одна и та же строка кода может выполняться ядром для GPU, CPU или TPU. Компиляторы вроде XLA делают это ещё запутаннее, сливая и перестраивая операции под конкретный целевой чип.

Пример из книги: лучшие известные алгоритмы умножения двух матриц n на n масштабируются как $\mathcal{O}(n^2)$ с константой c ниже 2,4, гораздо лучше наивного $\mathcal{O}(n^3)$. Для n , равного 1000, n в степени 2,4 — это около 16 миллионов против миллиарда для куба. Однако эти хитрые алгоритмы трудно эффективно выполнять параллельно на оборудовании вроде GPU, так что на практике их применяют редко.

Ещё два предела асимптотического мышления. Первый: наличие k процессоров совсем не меняет «О большое»: в лучшем случае оно делит время на k , постоянный множитель; восемь ядер превращают миллиард шагов в 125 миллионов на каждое, по-прежнему кубически по n . Второй: «О большое» игнорирует время, потраченное на перемещение данных с места на место, а оно часто и есть настоящее узкое место. Когда так случается, реализацию называют ограниченной памятью, в противоположность ограниченной вычислениями, и единственный практический способ понять, какой случай у вас, — запустить профилировщик, инструмент, который измеряет, куда на самом деле уходит время. Книга обещает, что оценка стоимости алгоритма окажется далеко не тривиальной, потому что асимптотическая сложность и наблюдаемая скорость тянут в разные стороны.



ПРОФИЛИРОВЩИК

Ещё два предела

Опыт

Возьмите две сетки чисел 2 на 2 и перемножьте их вручную, считая умножения: вы насчитаете 8. Теперь сделайте две сетки 3 на 3: вы насчитаете 27. Удвоение с 2 до 4 дало бы 64, в восемь раз больше: размер вырос в два раза, работа — в два на два на два раза, вот что значит «кубически».

Запомним

Вычисление $Z \sim (a, c)$ из $X \sim (a, b)$ и $Y \sim (b, c)$ по определению требует ac скалярных произведений длины b , это его временная сложность, и памяти под выход, это его пространственная сложность. «О большое» говорит, что $f(x)$ есть $O(g(x))$, когда f остаётся ниже c на g начиная с некоторого x_0 , так что важна только самая быстрорастущая часть; наивное умножение матриц есть $O(abc)$, кубическое для квадратных матриц. Реальная скорость зависит от оборудования, ядер и компиляторов: алгоритмы ниже n в степени 2,4 существуют, но плохо работают на GPU, а k процессоров лишь делят время на константу. Перемещение данных может доминировать, делая код ограниченным памятью, а не вычислениями, и только профилировщик скажет, что именно.

Вопросы

1. Две квадратные матрицы растут с размера 100 до размера 1000. Примерно во сколько раз больше работы нужно наивному умножению?
 - а. Примерно в 10 раз больше
 - б. Примерно в 100 раз больше
 - с. Примерно в 1000 раз больше

2. Почему алгоритмы, перемножающие матрицы быстрее чем за n в степени 2,4 шагов, редко применяются на практике?
 - a. Потому что они дают слегка неверные результаты
 - b. Потому что их трудно эффективно распараллелить на оборудовании вроде GPU
 - c. Потому что им нужно больше памяти, чем выходной матрице
3. Почему наличие k процессоров не меняет сложность алгоритма в смысле «О большого»?
 - a. Потому что в лучшем случае оно делит время на константу k , а «О большое» игнорирует константы
 - b. Потому что процессоры не могут делить память
 - c. Потому что «О большое» учитывает только самый медленный процессор
4. Код ограничен памятью. Что это значит и как это выяснить?
 - a. У него кончается память; это видно из сообщения об ошибке
 - b. Он делает слишком много умножений; это показывает анализ «О большого»
 - c. Узкое место — перемещение данных, а не вычисления; это показывает профилировщик

Ответы

1. c) Наивное умножение матриц n на n есть $O(n^3)$: в десять раз больший размер даёт десять на десять на десять, в тысячу раз больше работы, от миллиона до миллиарда умножений.

2. б) Их асимптотическая стоимость гораздо лучше, но реальная производительность зависит от оборудования и параллелизма, и эти алгоритмы плохо работают на сильно параллельных чипах.

3. а) Параллелизм может дать ускорение не более чем в константу $1/k$ раз, а асимптотический анализ игнорирует постоянные множители.

4. с) «О большое» игнорирует перемещение данных, которое часто доминирует; такой код ограничен памятью, а не вычислениями, и только запуск профилировщика показывает, какой случай перед нами.

Разобранные примеры

Считаем работу произведения X имеет форму $(2, 3)$, а Y имеет форму $(3, 4)$. Сколько скалярных произведений и сколько умножений требует вычисление $Z = XY$ по определению и сколько чисел нужно хранить?

Z имеет форму $(a, c) = (2, 4)$, так что у него $ac = 8$ элементов.

Каждый элемент — скалярное произведение длины $b = 3$, то есть по 3 умножения.

Всего $8 \times 3 = 24$ умножения, что равно $abc = 2 \times 3 \times 4$.

Последовательная реализация хранит выход, 8 чисел.

Ответ: 8 скалярных произведений, 24 умножения, 8 хранимых чисел

Проверяем утверждение об «О большом» Покажите, что $f(x) = 3x^2 + 5x$ есть $O(x^2)$, найдя c и x_0 .

Попробуем $c = 4$: нам нужно $3x^2 + 5x \leq 4x^2$.

Вычтем $3x^2$: условие становится $5x \leq x^2$.

Для положительного x разделим на x : $5 \leq x$, так что это верно для всех $x \geq 5$.

Ответ: $c = 4$ и $x_0 = 5$ подходят, так что f есть $O(x^2)$.

Задачи

1. Сколько умножений требует наивное произведение двух матриц 10 на 10 и двух матриц 20 на 20?

Подсказка: Используйте n^3 для квадратных матриц.

Ответ: $10^3 = 1000$ и $20^3 = 8000$: удвоение размера умножает работу на восемь.

2. Произведение $X \sim (1000, 50)$ и $Y \sim (50, 10)$ вычисляется по определению. Какова его временная сложность в умножениях и сколько чисел содержит выход?

Подсказка: abc умножений; выход имеет форму (a, c).

Ответ: $1000 \times 50 \times 10 = 500\,000$ умножений; выход имеет форму (1000, 10), 10 000 чисел.

3. Объясните, почему выбор между двумя алгоритмами только по их «О большому» может ввести вас в заблуждение на реальном телефоне или GPU.

Подсказка: Подумайте о параллельном оборудовании, ядрах и перемещении данных.

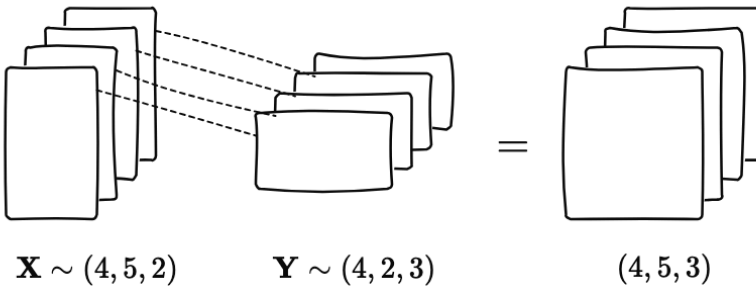
Ответ: «О большое» игнорирует постоянные множители, параллелизм и перемещение данных. Теоретически более быстрый алгоритм может плохо распараллеливаться на GPU, реально выполняемое ядро зависит от оборудования, а перемещение данных может быть настоящим узким местом, которое выявляет только профилировщик.

Много произведений сразу и короткая запись для всех них

Представьте стопку из четырёх листов бумаги в клетку, на каждом маленькая таблица чисел, а рядом вторую стопку из четырёх. Вы хотите умножить лист один на лист один, лист два на лист два и так далее, четыре отдельных произведения матриц бок о бок. Внутри каждой пары ничего нового не происходит; единственное новое — это стопка, и эта часть показывает, как библиотеки обращаются с такими стопками, и аккуратный способ записать любую из этих операций короткой строкой букв.

$$[\text{BMM}(\mathbf{X}, \mathbf{Y})]_i = \mathbf{X}_i \mathbf{Y}_i$$

$$\mathbf{X} @ \mathbf{Y}$$



$$4 \times (5 \cdot 2 \cdot 3) = 4 \times 30 = 120$$

Одно произведение на лист

Векторы и матрицы особенные, говорит книга, потому что для них определено много операций, которые в большем числе измерений не определены или неудобны: матричная экспонента, умножение матриц, определители. В более высоких измерениях большинство нужных нам операций — это либо батчевые версии матричных операций, либо сочетания матричных операций с редукциями, такими как суммы.

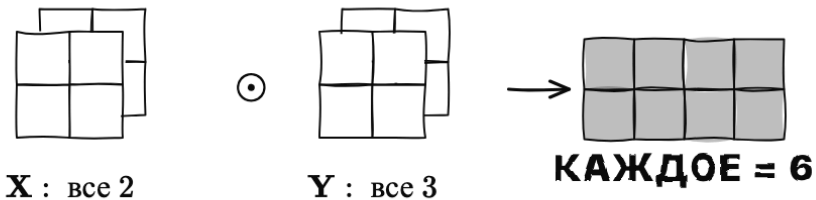
Сначала батчевый вид. Возьмём два тензора $X \sim (n, a, b)$ и $Y \sim (n, b, c)$, две стопки по n матриц в каждой. Батчевое умножение матриц, ВММ, — это $[ВММ(X, Y)]_i = X_i Y_i \sim (n, a, c)$. Словами: для каждой позиции i в стопке умножьте i -тую матрицу X на i -тую матрицу Y и сложите n результатов в стопку. Откуда это берётся — всего лишь наши листы бумаги, по одному обычному произведению матриц на лист. Блок кода в книге использует стопки по 4: X формы 4 на 5 на 2 и Y формы 4 на 2 на 3 дают результат формы 4 на 5 на 3. Считаем работу: каждое произведение — это 5 на 2 на 3, то есть 30 умножений, и таких 4, итого 120.

Библиотеки обращаются с такими стопками прозрачно: они предполагают, что измерения батча идут первыми, это ведущие измерения, и в PyTorch батчевое умножение записывается ровно как обычное, через `matmul` или знак `@`. Ваше фотоприложение делает это постоянно: целый батч фотографий проходит через один и тот же слой, матрица каждой фотографии умножается на одни и те же веса, всё за один вызов.

Теперь вид с редукцией. Для двух тензоров одной формы, $X, Y \sim (a, b, c)$, обобщённое скалярное произведение, GDT, — это $GDT(X, Y) = \sum_{i, j, k} [X \odot Y]_{i j k}$. Словами: перемножьте два тензора

поэлементно, а затем сложите всё. Это просто обычное скалярное произведение двух тензоров после того, как каждый развёрнут в один длинный вектор. Для двух тензоров формы 2 на 2 на 2, где каждый элемент X равен 2, а каждый элемент Y равен 3, есть 8 произведений по 6, так что результат равен 48. Эти два вида, говорит книга, покрывают большинство тензорных операций, используемых в остальной книге.

$$\text{GDT}(\mathbf{X}, \mathbf{Y}) = \sum_{i,j,k} [\mathbf{X} \odot \mathbf{Y}]_{ijk}$$



РАЗВЕРНУТЬ И ПЕРЕМНОЖИТЬ

$$8 \times 6 = 48$$

EINSUM →

Перемножить всё, сложить всё

Следующий раздел, отмеченный как необязательный, вводит `einsum`, соглашение, которое может выразить практически любую тензорную операцию, суммы, произведения и редукции, простой текстовой строкой. Запишем два примера так, чтобы был виден каждый индекс. Батчевое умножение: $M_{ijk} = \sum_z A_{ij} z$

$B_{i_z k}$. Обобщённое скалярное произведение: $M = \sum_i \sum_j \sum_k X_{i j k} Y_{i j k}$.

$$M_{ijk} = A_{ijz} B_{izk}$$

$$M = X_{ijk} Y_{ijk}$$

НЕТ СЛЕВА → СУММИРУЕТСЯ

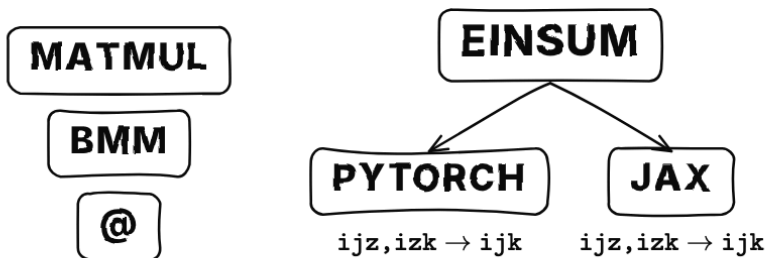
$$\begin{array}{ccc} i j z, i z k & \rightarrow & i j k \\ \text{А} & \text{В} & \text{ВЫХОД} \\ i j k, i j k & \rightarrow & \end{array}$$

От сумм к строке

Следуя обозначениям Эйнштейна, в упрощённой форме, которая не различает верхние и нижние индексы, мы опускаем знаки суммы с одним правилом: по любому индексу, который есть справа, но отсутствует слева, идёт суммирование. Так что $M_{i j k} = A_{i j z} B_{i_z k}$ означает «суммировать по z », потому что z отсутствует в M , а $M = X_{i j k} Y_{i j k}$ означает «суммировать по всем трём», потому что у M вообще нет индексов.

Затем мы оставляем только буквы индексов, сначала операнды, после стрелки результат. Батчевое умножение матриц становится

' $ijz, izk \rightarrow ijk$ '. Обобщённое скалярное произведение становится ' $ijk, ijk \rightarrow$ '. Прочитайте первое вслух: у первого тензора оси i, j, z ; у второго i, z, k ; выход сохраняет i, j, k ; z исчезает, значит, по нему суммируют. Между формулами и строками взаимно однозначное соответствие, и большинство библиотек реализуют это как операцию `einsum`.



РАЗНЫЕ ПРАВИЛА

$$A \sim (n, a, b), B \sim (n, c, b)$$

EINOPS

$$ijz, ikz \rightarrow ijk$$

Один синтаксис везде

Преимущество, говорит книга, в том, что больше не нужно помнить интерфейс каждой библиотеки. В одном PyTorch несколько функций умножения, таких как `matmul` и `bmm`, с разными правилами трансляции и требованиями к формам, а `einsum` даёт один единый синтаксис для всех них. Переход между библиотеками тоже становится прозрачным: строка `einsum` для

батчeveго умножения в JAX ровно та же, что и в PyTorch, как показывают два блока кода в книге.

Транспонированные оси тоже даются легко. Для $A \sim (n, a, b)$ и $B \sim (n, c, b)$ умножение каждой матрицы A на транспонированную парную матрицу B требует только поменять буквы у B :

`'ijz,ikz→ijk'`. Суммируемая буква z теперь стоит последней в обоих входах, а это ровно то, что значит умножение на транспонированную матрицу. По этим причинам `einsum` и его родственники, такие как популярный пакет `einops`, стали широко использоваться.

Опыт

Напишите на карточке строку `'ijz,ikz→ijk'`, а под ней формы 4 на 5 на 2 и 4 на 2 на 3. Сопоставьте буквы с размерами: i — это 4, j — 5, z — 2, k — 3. Прочитайте выход i, j, k : 4 на 5 на 3, та же форма, что печатает код в книге, а z , исчезнувшая буква, — это размер 2, который просуммировали.

Запомним

В более чем двух измерениях большинство операций — это батчевые матричные операции или матричные операции в сочетании с редукциями. Батчевое умножение матриц перемножает парные матрицы в двух стопках, (n, a, b) с (n, b, c) даёт (n, a, c) , и библиотеки автоматически считают ведущие измерения батчами. Обобщённое скалярное произведение перемножает два тензора одной формы поэлементно и суммирует всё, скалярное произведение развёрнутых тензоров. `Einsum` записывает любую такую операцию строкой букв-индексов, суммируя по каждой букве, отсутствующей в выходе, как в `'ijz,izk→ijk'` или `'ijk,ijk→'`, с одним синтаксисом во всех библиотеках.

Вопросы

1. X имеет форму (n, a, b) , а Y имеет форму (n, b, c) . Что даёт батчевое умножение матриц?
 - a. Одну матрицу формы (a, c) , сумму всех n произведений
 - b. Стопку из n матриц формы (a, c) , по одному произведению на позицию
 - c. Одно число, скалярное произведение развёрнутых тензоров
2. В строке `einsum 'ijz,izk→ijk'` почему по индексу z идёт суммирование?

- a. Потому что он есть в обоих входах, но отсутствует в выходе
 - b. Потому что это последняя использованная буква алфавита
 - c. Потому что это измерение батча
3. Что вычисляет строка `'ijk,ijk→'`?
- a. Тензор той же формы с перемноженными элементами
 - b. Батчевое произведение матриц
 - c. Обобщённое скалярное произведение: перемножить поэлементно и сложить всё в одно число
4. Почему книга находит `einsum` удобным при работе с разными библиотеками?
- a. Потому что он работает быстрее любого встроенного умножения
 - b. Потому что одна единая строка заменяет разные функции каждой библиотеки и одинакова в PyTorch и JAX
 - c. Потому что он избавляет от необходимости в измерениях батча

Ответы

1. **b)** Для каждой позиции i в стопке i -тая матрица X умножается на i -тую матрицу Y , и n результатов складываются в стопку формы (n, a, c) .
2. **a)** Соглашение таково: по любому индексу, который есть справа, но отсутствует в результате, идёт суммирование; z нет в `ijk`, так что он суммируется.

3. c) У выхода вообще нет индексов, так что суммирование идёт по каждому индексу: поэлементные произведения складываются, скалярное произведение развёрнутых тензоров.

4. b) В одном PyTorch несколько функций умножения с разными правилами, а `einsum` даёт один синтаксис для всех них, и та же строка работает в JAX.

Разобранные примеры

Батчевое произведение из книги В блоке кода книги X имеет форму $(4, 5, 2)$, а Y имеет форму $(4, 2, 3)$. Найдите форму $X @ Y$ и посчитайте умножения, сделанные по определению.

Ведущее измерение 4 — это батч: есть 4 отдельных произведения матриц.

Каждое произведение умножает матрицу 5 на 2 на матрицу 2 на 3, давая 5 на 3.

Так что результат имеет форму $(4, 5, 3)$.

Каждому произведению нужно $5 \times 2 \times 3 = 30$ умножений, а всего $4 \times 30 = 120$.

Ответ: Форма $(4, 5, 3)$; 120 умножений

Задачи

1. X и Y оба имеют форму $(2, 2, 2)$; каждый элемент X равен 1, а каждый элемент Y равен 5. Каково их обобщённое скалярное произведение?

Подсказка: Посчитайте элементы и перемножьте каждую пару.

Ответ: Есть $2 \times 2 \times 2 = 8$ элементов, каждое произведение равно $1 \times 5 = 5$, так что сумма равна 40.

2. А имеет форму (8, 10, 6), а В имеет форму (8, 6, 4). Используя 'ijz,izk→ijk', назовите размер каждой буквы и форму выхода.

Подсказка: Сопоставьте буквы каждого операнда с его размерами.

Ответ: $i = 8, j = 10, z = 6, k = 4$; выход сохраняет i, j, k , так что он имеет форму (8, 10, 4), а по $z = 6$ идёт суммирование.

3. А имеет форму (n, a, b), а В имеет форму (n, c, b). Какая строка `einsum` умножает каждую матрицу А на транспонированную парную матрицу В и какова форма выхода?

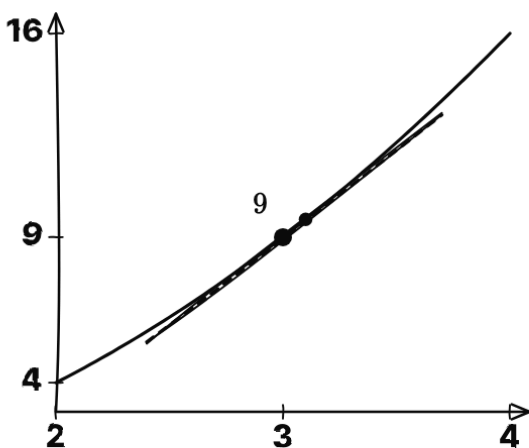
Подсказка: Суммируемая буква должна стоять на оси размера b в обоих входах.

Ответ: 'ijz,ikz→ijk': z — общий размер b , по нему идёт суммирование, а выход имеет форму (n, a, c).

Наклон в точке

Откройте калькулятор, наберите 3 и возведите в квадрат: 9. Теперь наберите 3,1 и возведите в квадрат: 9,61, то есть прирост 0,61 при шаге 0,1, примерно в шесть раз больше шага. Попробуйте 3,01: 9,0601, прирост 0,0601, ещё ближе к шести шагам. Отношение сходится к одному числу, и это число — производная, основа всего, что книга называет дифференцируемым.

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



ОТНОШЕНИЕ

$$h = 0.1 : 6.1$$

$$h = 0.01 : 6.01$$

$$h \rightarrow 0 : 6$$

Отношения сходятся к 6

Градиенты играют в книге центральную роль, как и подсказывает слово «дифференцируемый»: они позволяют оптимизировать модели полуавтоматическими методами, выведенными из градиентного спуска. Этот раздел напоминает основные определения для дифференцирования функций, сосредоточиваясь на том, что понадобится позже, иногда ценой полной математической строгости.

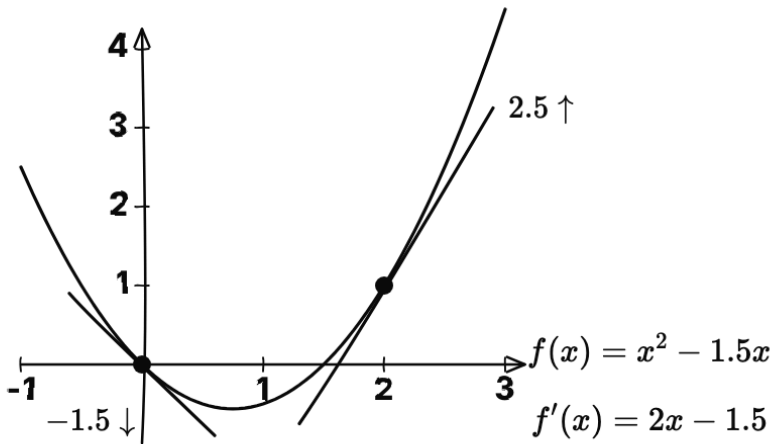
Начнём с простейшего случая, $y = f(x)$, одно число на входе и одно на выходе. Её производная определяется как $f'(x) = \lim_{h \rightarrow 0} (f(x + h) - f(x)) / h$. Словами: шагните чуть-чуть, на h , посмотрите, насколько изменился выход, разделите на шаг и дайте шагу сжаться к нулю. Именно это сделал наш калькулятор: при f , равном x в квадрате, в точке 3 шаги 0,1 и 0,01 дали отношения 6,1 и 6,01, подбирающиеся к 6.

Книга использует несколько обозначений. Символ ∂ обозначает производные и градиенты любой размерности; ∂_x , или ∂ над ∂x , называет переменную, по которой мы дифференцируем, когда это нужно сказать; а $f'(x)$, так называемая запись Лагранжа, оставлена для функций одного числа. Книга просто предполагает, что нужные ей производные существуют, что не гарантировано даже для непрерывных функций, и возвращается к этому вопросу только для функций с изломами, таких как модуль $f(x) = \text{abs}(x)$ в нуле, в шестой главе.

Для простых функций применение определения даёт знакомые правила. Для степени $\partial x^p = p x^{p-1}$: показатель спускается вперёд и уменьшается на единицу; для x в кубе в точке 2 это 3 на 2 в квадрате, то есть 12. Для логарифма $\partial \log(x) = 1/x$: в точке 2 это

одна вторая. Для синуса $\partial \sin(x) = \cos(x)$: в нуле это косинус нуля, то есть 1.

Геометрически производная — это наклон касательной, проходящей через точку, или, что то же самое, наилучшее приближение функции прямой линией в этой точке. Вот взгляд, который важен: наклон говорит, как функция ведёт себя поблизости. При положительном наклоне функция растёт вправо и убывает влево, на достаточно малом отрезке; при отрицательном наклоне наоборот. Рисунок в книге изображает $f(x) = x^2 - 1.5x$ с касательными в двух точках. Её производная равна $2x - 1.5$: в нуле наклон равен минус 1,5, так что кривая там идёт вниз, а в точке 2 он равен плюс 2,5, так что кривая идёт вверх. Такое прочтение наклона перенесётся на функции многих переменных.



Вниз в нуле, вверх в двойке

Три свойства распространяются на много измерений.

Линейность: производная суммы равна сумме производных,

$\partial[f(x) + g(x)] = f'(x) + g'(x)$; для x в квадрате плюс синус x

производная равна $2x$ плюс косинус x . Правило произведения:

$\partial[f(x)g(x)] = f'(x)g(x) + f(x)g'(x)$; при f , равном x , и g , равном x в

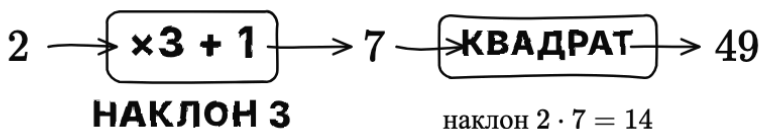
квадрате, это 1 на x в квадрате плюс x на $2x$, то есть $3x$ в

квадрате, ровно производная x в кубе.

$$\partial[f + g] = f' + g'$$

$$\partial[fg] = f'g + fg'$$

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



$$3 \times 14 = 42$$

СЕТЬ — ДЛИННАЯ ЦЕПОЧКА

Скорости перемножаются вдоль цепочки

И цепное правило, на котором держится вся эта книга: $\partial[f(g(x))]$ $= f'(g(x)) g'(x)$. Словами: когда одна функция питает другую, скорости перемножаются. Возьмите g , равное $3x$ плюс 1, и f , равное квадрату, при x , равном 2: g даёт 7 и имеет наклон 3, квадрат имеет наклон 2 на 7, то есть 14, в точке 7, и произведение равно 42. Это цепочка на калькуляторе: два, умножить на три, плюс один, в квадрате; крошечное изменение исходной двойки сдвигает итоговые 49 в сорок два раза сильнее. Нейронная сеть — длинная цепочка таких функций, и цепное правило — это способ найти её наклоны.

Опыт

На калькуляторе вычислите 2 на 3 плюс 1, в квадрате: 49. Теперь начните с 2,01: 2,01 на 3 плюс 1 равно 7,03, а в квадрате это 49,4209. Выход вырос на 0,4209 при шаге 0,01, примерно в 42 раза больше шага, ровно как предсказало цепное правило, умножив 14 на 3.

Запомним

Производная $f'(x)$ — это предел изменения выхода, делённого на сжимающийся шаг h , Δ для x в квадрате в точке 3. Отсюда знакомые правила: x^p даёт px^{p-1} , $\log x$ даёт $1/x$, синус даёт косинус; запись ∂ охватывает любую размерность, а f' — для скалярных функций. Производная — это наклон касательной, наилучшее приближение прямой: положительный значит рост, отрицательный — убывание, как для $x^2 - 1.5x$ в точке 2 и в нуле. Производные линейны, подчиняются правилу произведения и цепному правилу, $f'(g(x))$ на $g'(x)$, которое перемножает скорости вдоль цепочки, как 14 на 3 даёт 42.

Вопросы

1. Производная $f(x) = x^2 - 1.5x$ равна $2x - 1.5$. Что она говорит о функции вблизи $x = 0$?
 - а. Функция там убывает, поскольку наклон равен -1.5

- б. Функция там растёт, поскольку наклон равен 1,5
 - с. Функция там плоская, поскольку x равен нулю
- 2. Почему цепное правило — это правило, «на котором держится вся эта книга»?
 - а. Потому что в нейронных сетях нет сумм и произведений
 - б. Потому что сеть — длинная цепочка функций, и цепное правило перемножает их скорости, чтобы найти наклон целого
 - с. Потому что это единственное правило, верное для функций с изломами
- 3. Какую роль в определении производной играет h ?
 - а. Это высота функции в точке x
 - б. Это фиксированный шаг 0,1, используемый в каждом вычислении
 - с. Это шаг, сжимающийся к нулю, и отношение изменения выхода к h приближается к производной

Ответы

1. а) В нуле наклон равен $2 \times 0 - 1,5 = -1,5$; отрицательный наклон означает, что функция убывает вправо и растёт влево, на достаточно малом отрезке.
2. б) Когда одна функция питает другую, производная равна $f'(g(x))$ на $g'(x)$; сеть — длинная композиция, так что её наклоны получаются перемножением скоростей вдоль цепочки.
3. с) Производная — это предел $(f(x + h) - f(x)) / h$ при h , стремящемся к нулю; для x^2 в точке 3 отношения 6,1 и 6,01 подбираются к 6.

Разобранные примеры

Производная через сжимающиеся шаги Для $f(x) = x^2$ в точке $x = 3$ вычислите отношение $(f(x + h) - f(x)) / h$ при $h = 0,1$ и $h = 0,01$ и сравните с правилом $\partial x^p = px^{p-1}$.

$h = 0,1$: $3,1^2 = 9,61$, так что $(9,61 - 9) / 0,1 = 6,1$.

$h = 0,01$: $3,01^2 = 9,0601$, так что $(9,0601 - 9) / 0,01 = 6,01$.

Отношения приближаются к 6 по мере сжатия h .

Правило степени даёт $2 \times 3^1 = 6$, предел.

Ответ: 6,1, затем 6,01, приближаясь к производной 6

Цепное правило на цепочке калькулятора При $g(x) = 3x + 1$ и $f(u) = u^2$ найдите производную $f(g(x))$ в точке $x = 2$.

$g(2) = 7$, а $g'(x) = 3$.

$f'(u) = 2u$, так что $f'(g(2)) = 2 \times 7 = 14$.

Цепное правило: $f'(g(2)) \times g'(2) = 14 \times 3 = 42$.

Проверка: из 2,01, $(3 \times 2,01 + 1)^2 = 7,03^2 = 49,4209$, прирост 0,4209 при шаге 0,01, примерно в 42 раза.

Ответ: 42

Задачи

1. С помощью правил степени и линейности найдите производную $x^3 + 4x^2$ в точке $x = 1$.

Подсказка: Продифференцируйте каждое слагаемое и сложите.

Ответ: $3x^2 + 8x$, что в точке 1 равно $3 + 8 = 11$.

2. Примените правило произведения к $x \cdot \log(x)$ и вычислите производную в точке $x = 1$.

Подсказка: У $f = x$ производная $f' = 1$; у $g = \log x$ производная $g' = 1/x$.

Ответ: $1 \times \log(x) + x \times (1/x) = \log(x) + 1$, что в точке 1 равно $0 + 1 = 1$.

3. Найдите производную $\sin(2x)$ в точке $x = 0$ с помощью цепного правила.

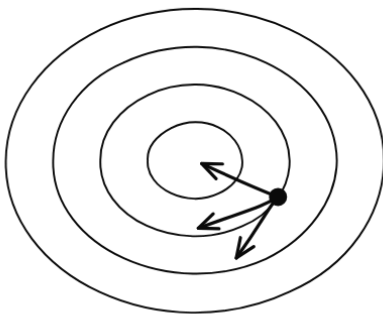
Подсказка: Внешняя функция — синус, внутренняя — $2x$.

Ответ: $\cos(2x) \times 2$, что в нуле равно $\cos(0) \times 2 = 2$.

Наклоны во многих направлениях

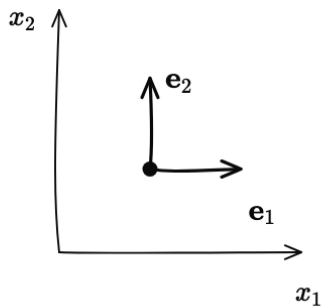
Встаньте на склоне холма и спросите, насколько он крут.

Честный ответ: зависит от того, куда вы смотрите. Прямо вверх — круто, вдоль склона — ровно, и у каждого промежуточного направления своя крутизна. У функции одного числа было только два направления, влево и вправо; у функции многих чисел их бесконечно много, и эта часть показывает, что один вектор, градиент, отвечает за все.



— КРУТО
— РОВНО

$$\partial_{x_i} f = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$



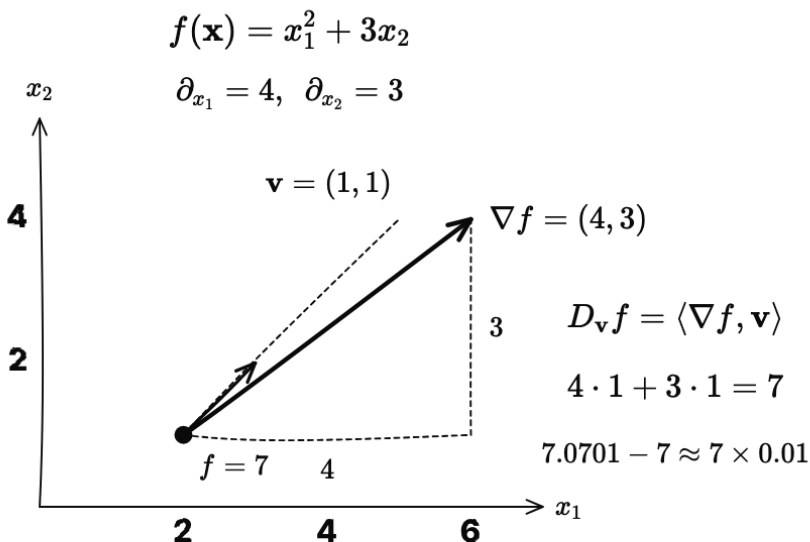
**ЧАСТНАЯ
ПРОИЗВОДНАЯ**

Крутизна зависит от направления

Возьмём функцию $y = f(\mathbf{x})$, вход которой — вектор $\mathbf{x} \sim (d)$.

Крошечный сдвиг ничего не значит, пока мы не скажем, в какую сторону. Простейший выбор — двигаться вдоль одной оси, i -той, держа все остальные входы неподвижными: $\partial_{x_i} f(\mathbf{x}) = \partial y / \partial x_i = \lim_{h \rightarrow 0} (f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})) / h$. Здесь $\mathbf{e}_i$ — i -тый базисный вектор, i -тая строка единичной матрицы: его j -тая компонента равна 1, когда j равно i , и 0 иначе, так что прибавление h на него сдвигает только i -тый вход. Это называется частной производной: обычная производная, взятая вдоль одной оси.

Будем всюду пользоваться маленькой функцией: $f(\mathbf{x}) = x_1^2 + 3x_2$ в точке $\mathbf{x} = (2, 1)$, где f равна 4 плюс 3, то есть 7. Двигая только x_1 , получаем наклон $2x_1$, то есть 4; двигая только x_2 , наклон 3.



Градиент отвечает за каждое направление

Сложив все частные производные в один столбец, получаем d -мерный вектор, градиент: $\nabla f(\mathbf{x}) = \partial f(\mathbf{x}) = [\partial_{x_1} f(\mathbf{x}), \dots, \partial_{x_d} f(\mathbf{x})]^T$. Поскольку он так фундаментален, у него есть собственный символ, перевёрнутый треугольник ∇ . Для нашей функции градиент в точке $(2, 1)$ равен $(4, 3)$.

А что насчёт сдвига в произвольном направлении $\mathbf{v}$? Это даёт производную по направлению, $D_{\mathbf{v}} f(\mathbf{x}) = \lim_{h \rightarrow 0} (f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})) / h$. Поскольку любое движение можно разложить на отдельные движения вдоль каждой оси, производная по направлению оказывается скалярным произведением: $D_{\mathbf{v}} f(\mathbf{x}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle = \sum_i \partial_{x_i} f(\mathbf{x}) v_i$. Каждое слагаемое — это наклон вдоль одной оси, умноженный на то, насколько мы сдвигаемся вдоль этой оси. Для $\mathbf{v} = (1, 1)$: 4 на 1 плюс 3 на 1, то есть 7. Проверьте напрямую: в точке $(2, 0.1, 1, 0.1)$ f равна 4,0401 плюс 3,03, то есть 7,0701, изменение на 0,0701 при шаге 0,01, примерно в 7 раз больше шага. Так что знать градиент достаточно, чтобы знать наклон в каждом возможном направлении: вот склон холма, на который отвечает один вектор.

Теперь общий случай: векторный вход $\mathbf{x} \sim (d)$ и векторный выход $\mathbf{y} \sim (o)$, который, по словам книги, самый общий случай из нужных нам. При нескольких выходах мы берём по одному градиенту на выход, и, сложив их строками, получаем матрицу o на d , якобиан: $\partial f(\mathbf{x}) \sim (o, d)$, элемент которого в строке j и столбце i равен $\partial y_j / \partial x_i$. Возьмём $f(\mathbf{x}) = (x_1 x_2, x_1 + x_2)$ в точке $(2, 3)$: у первого выхода наклоны 3 и 2, у второго 1 и 1, так что у якобиана строки $(3, 2)$ и $(1, 1)$. При одном выходе, o равном 1, якобиан — это градиент; при одном входе и одном выходе это обычная производная.

$$\partial f(\mathbf{x}) \sim (o, d), \quad [\partial f]_{ji} = \frac{\partial y_j}{\partial x_i}$$

$$f(\mathbf{x}) = (x_1 x_2, x_1 + x_2) \text{ в } (2, 3) : \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix}$$

0 = 1: ГРАДИЕНТ

$$\partial[f(g(\mathbf{x}))] = [\partial f] \partial g$$

$$g(3, 1) = (4, 2), \quad \partial f = (2, 4)$$

$$\begin{pmatrix} 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = (6, -2)$$

$$\nabla(x_1^2 - x_2^2) \text{ в } (3, 1) = (6, -2)$$

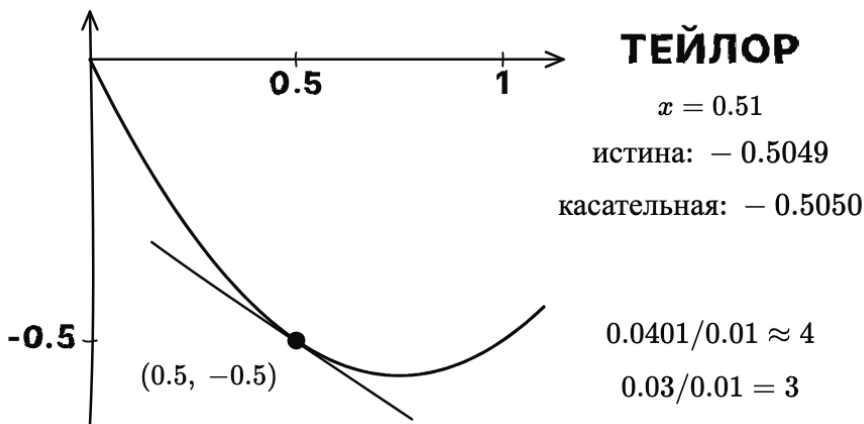
Якобианы перемножаются

Якобианы наследуют свойства производных, и прежде всего цепное правило становится произведением матриц: $\partial[f(g(\mathbf{x}))] = [\partial f(\bullet)] \partial g(\mathbf{x})$, где первый множитель вычислен в точке $g(\mathbf{x}) \sim (h)$. Проверим на $g(\mathbf{x}) = (x_1 + x_2, x_1 - x_2)$ и $f(\mathbf{u}) = u_1 u_2$ в точке $\mathbf{x}$, равной $(3, 1)$. g даёт $\mathbf{u}$, равное $(4, 2)$, с якобианом из строк $(1, 1)$ и $(1, -1)$; у f градиент (u_2, u_1) , то есть там $(2, 4)$. Перемножаем: 2 на 1 плюс 4 на 1 равно 6, а 2 на 1 плюс 4 на минус 1 равно минус 2: строка $(6, -2)$. Напрямую: f от g — это $x_1^2 - x_2^2$, градиент которой в точке $(3, 1)$ тоже равен $(6, -2)$.

Как и в одном измерении, градиенты и якобианы — линейные функции, касательные в точке. Градиент даёт наилучшее приближение первого порядка вблизи $\mathbf{x}_0$: $\tilde{f}(\mathbf{x}) = f(\mathbf{x}_0) + \langle \partial f(\mathbf{x}_0), \mathbf{x} \rangle$

– x_0), значение в точке плюс наклон, умноженный на смещение. Это теорема Тейлора. Блок кода в книге проверяет её для $f(x) = x^2 - 1.5x$ в точке 0,5, где f равна минус 0,5 и наклон равен минус 0,5: один шаг на 0,01 даёт истинное значение минус 0,5049 и приближение минус 0,5050, почти одно и то же.

$$\tilde{f}(x) = f(x_0) + \langle \partial f(x_0), x - x_0 \rangle$$



Касательное приближение

Опыт

На калькуляторе вычислите x с один в квадрате плюс 3 x с два в точке $(2, 1)$, получится 7, затем в точке $(2,01, 1)$, получится 7,0401, и в точке $(2, 1,01)$, получится 7,03. Два изменения, 0,0401 и 0,03, делённые на шаг 0,01, дают примерно 4 и 3, частные производные. Теперь шагните обоими сразу в точку $(2,01, 1,01)$: 7,0701, изменение примерно в 7 раз больше шага, та сумма, которую предсказал градиент.

Запомним

Для векторного входа частная производная сдвигает один вход вдоль его оси; сложив все вместе, получаем градиент $\nabla f(x)$, $(4, 3)$ для $x_1^2 + 3x_2$ в точке $(2, 1)$. Производная по направлению вдоль любого v — это скалярное произведение градиента и v , так что градиент даёт наклон в каждом направлении. Для векторного выхода по одному градиенту на выход складываются в якобиан, матрицу o на d , и цепное правило становится произведением якобианов, как показало $(6, -2)$ обоими способами. Вблизи точки f приближается своим значением плюс градиент, скалярно умноженный на смещение, теорема Тейлора, проверенная как $-0,5049$ против $-0,5050$.

Вопросы

1. Почему знания градиента достаточно, чтобы найти наклон функции в любом направлении?
 - a. Потому что производная по направлению вдоль v — это скалярное произведение градиента и v
 - b. Потому что у функции многих входов есть только одно направление, которое имеет значение
 - c. Потому что градиент перечисляет наклоны во всех возможных направлениях по одному
2. Функция принимает вектор размера 5 и возвращает вектор размера 3. Какова форма её якобиана?
 - a. $(5, 3)$
 - b. $(3, 5)$
 - c. (15)
3. Как выглядит цепное правило для якобианов?
 - a. Якобианы двух функций складываются
 - b. Якобианы перемножаются поэлементно
 - c. Якобиан внешней функции, вычисленный в точке $g(x)$, умножается как матрица на якобиан g
4. Что говорит теорема Тейлора в форме первого порядка о f вблизи точки x_0 ?
 - a. Что f везде поблизости равна своему значению в x_0
 - b. Что f лучше всего приближается своим значением в x_0 плюс градиент, скалярно умноженный на смещение

с. Что f растёт как квадрат расстояния от x_0

Ответы

1. **a)** Любое движение разлагается на движения вдоль каждой оси, так что производная по направлению — это сумма каждой частной производной, умноженной на движение вдоль этой оси, скалярное произведение $\langle \nabla f(x), v \rangle$.
2. **b)** Якобиан складывает по одному градиенту на выход в виде строк, так что у него $o = 3$ строки и $d = 5$ столбцов.
3. **c)** Производная композиции становится произведением матриц, ∂f , вычисленный в $g(x)$, на $\partial g(x)$, как показала строка $(6, -2)$ для $u_1 u_2$ после $(x_1 + x_2, x_1 - x_2)$.
4. **b)** Наилучшее линейное приближение — это $f(x_0) + \langle \partial f(x_0), x - x_0 \rangle$, которое в примере книги даёт $-0,5050$ против истинного $-0,5049$.

Разобранные примеры

Линеаризация из книги Для $f(x) = x^2 - 1.5x$ приблизьте $f(0,51)$ касательной в точке $0,5$ и сравните с истинным значением.

$$f(0,5) = 0,25 - 0,75 = -0,5.$$

$$f'(x) = 2x - 1,5, \text{ так что } f'(0,5) = -0,5.$$

$$\text{Приближение: } -0,5 + (-0,5) \times 0,01 = -0,505.$$

$$\text{Истинное значение: } 0,51^2 - 1,5 \times 0,51 = 0,2601 - 0,765 = -0,5049.$$

Ответ: Приближение $-0,5050$ против истинного $-0,5049$

Градиент и производная по направлению Для $f(x) = x_1^2 + 3x_2$ в точке $(2, 1)$ найдите градиент и производную по направлению вдоль $v = (1, 1)$.

$$\text{Частная по } x_1: 2x_1 = 4.$$

Частная по x_2 : 3.

Градиент: (4, 3).

Вдоль v : $4 \times 1 + 3 \times 1 = 7$; проверка: $f(2,01, 1,01) = 7,0701$, изменение 0,0701 при шаге 0,01.

Ответ: $\nabla f = (4, 3)$; производная по направлению 7

Задачи

1. Для $f(x) = x_1x_2 + x_2^2$ в точке (1, 2) найдите градиент и производную по направлению вдоль $v = (1, -1)$.

Подсказка: Продифференцируйте по x_1 , держа x_2 неподвижным, затем наоборот.

Ответ: $\partial/\partial x_1 = x_2 = 2$ и $\partial/\partial x_2 = x_1 + 2x_2 = 5$, так что $\nabla f = (2, 5)$; вдоль (1, -1): $2 - 5 = -3$.

2. Найдите якобиан $f(x) = (3x_1, x_1x_2)$ в точке (1, 4).

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

Ответ: Строка 1: (3, 0); строка 2: $(x_2, x_1) = (4, 1)$. У якобиана строки (3, 0) и (4, 1), форма (2, 2).

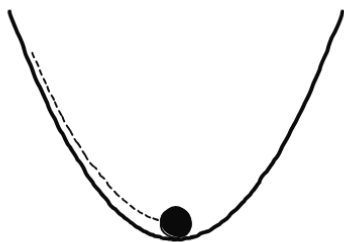
3. Для $f(x) = x_1^2 + 3x_2$ в точке $x_0 = (2, 1)$ с помощью приближения первого порядка оцените $f(2,1, 0,9)$, а затем вычислите истинное значение.

Подсказка: $f(x_0)$ плюс градиент (4, 3), скалярно умноженный на смещение.

Ответ: Смещение (0,1, -0,1): $7 + 4 \times 0,1 + 3 \times (-0,1) = 7,1$. Истинное: $2,1^2 + 3 \times 0,9 = 4,41 + 2,7 = 7,11$.

Скрытые размеры и первый шаг вниз по склону

Положите шарик в изогнутую салатницу и отпустите. Он не знает, где дно, но попадает туда: в каждое мгновение он катится туда, куда под ним наклонена чаша. Обучение модели — это такой шарик, и эта часть задаёт правило, которому он следует, после короткого, но важного предупреждения о размерах якобианов.



ОБУЧЕНИЕ

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$\partial_{\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbf{W}$$

$$\mathbf{W} = \begin{pmatrix} 1 & 2 & \boxed{0} \\ 0 & 1 & \boxed{4} \end{pmatrix}$$

**СДВИГ $x_3 \rightarrow$
ВЫХОДЫ НА 0 И 4**

Шарик в чаше; $\mathbf{W}$ как скорость

Книга называет предупреждение педантичным, но позже оно важно. Возьмём $y = \mathbf{W}x$, матрицу на вектор, сердце каждого слоя сети. Если смотреть на это как на функцию от x , её производная, как и раньше, — матрица o на d , и это просто $\partial_x[\mathbf{W}x] = \mathbf{W}$.

Словами: скорость, с которой выход меняется со входом, — это сама матрица, как производная z на x равна W . Для W со строками (1, 2, 0) и (0, 1, 4) малый сдвиг третьего входа сдвигает выходы на 0 и на 4 таких шага: третий столбец W .

Но если смотреть как на функцию от $\mathbf{W}$, вход сам является матрицей o на d , и «якобиан» тогда имеет форму (o, o, d) , по одной скорости на каждый выход и каждый элемент W . Мы всегда можем перейти к эквивалентной функции, принимающей развёрнутую версию W , $\text{vect}(\mathbf{W}) \sim (od)$, один длинный вектор, и тогда якобиан — обычная матрица формы (o, od) . Для нашего примера 2 на 3 форма — 2 на 2 на 3, то есть 12 чисел, или 2 на 6 после разворачивания, те же 12.

$$(o, o, d) = (2, 2, 3) \quad \text{или} \quad (o, od) = (2, 6)$$

$$y_i = \sum_j W_{ij} x_j$$

$$\partial y_i / \partial W_{ij} = x_j$$

1	2	3
---	---	---

0	0	0
---	---	---

ВЫХОД 1

0	0	0
---	---	---

1	2	3
---	---	---

ВЫХОД 2

$$1000 \times 1000 \times 1000 = 10^9$$

НЕ ХРАНИТСЯ: ВЕКТОР НА ЯКОБИАН

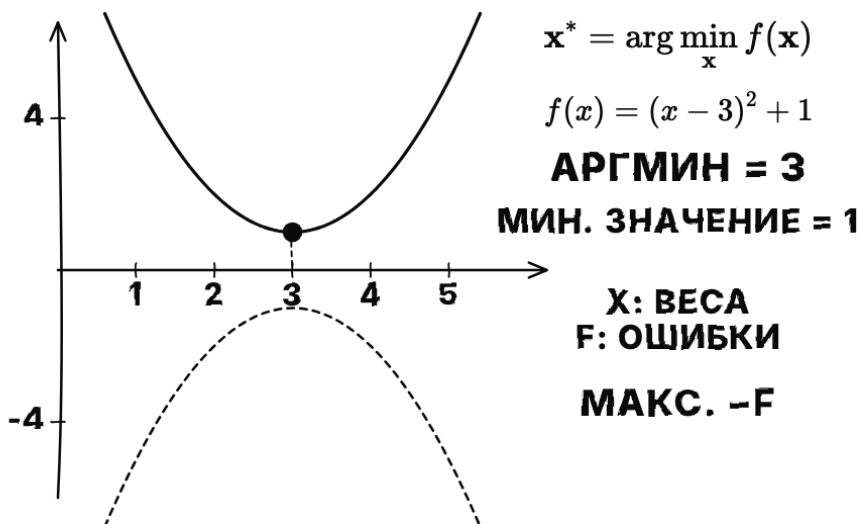
Якобиан по W

Блок в книге выписывает, что это за числа. Запишем произведение поэлементно: $y_i = \sum_j W_{ij} x_j$. Тогда сразу $\partial y_i / \partial W_{ij} = x_j$: сдвиг одного веса в строке i меняет выход i на входное значение этого веса и не трогает все остальные выходы. При $x = (1, 2, 3)$ выход 1 сдвигается на 1, 2 или 3 на единичный сдвиг весов строки 1 и вовсе не сдвигается при сдвиге весов строки 2. Так что из наших 12 чисел 6 — нули, а остальные 6 — просто x , скопированный дважды.

В этом и суть: хранить якобиан явно значило бы держать очень много повторяющихся значений. Для одного слоя с 1000 входов и 1000 выходов W содержит миллион весов, а его якобиан содержал бы 1000 на 1000 на 1000, миллиард чисел, в основном

нулей и копий. Как покажет шестая глава, нам никогда не нужно его строить, потому что на практике мы лишь применяем якобиан к другому тензору через так называемые произведения вектора на якобиан, и этот взгляд можно формализовать, рассматривая якобианы как абстрактные линейные отображения. Книга сохраняет векторную запись, потому что её достаточно, но предупреждает, что в шестой главе матричные якобианы будут прятать свои истинные размеры внутри форм входов и выходов, особенно размеров батча.

Теперь причина, по которой градиенты стоят всех этих хлопот: раздел 2.3, градиентный спуск. Задача — минимизировать функцию $f(\mathbf{x})$ при $\mathbf{x} \sim (d)$: $\mathbf{x}^* = \operatorname{argmin} f(\mathbf{x})$. Как и аргмакс, аргмин означает «то значение $\mathbf{x}$, которое даёт наименьшее возможное f ». Для $f(\mathbf{x}) = (\mathbf{x} - 3)^2 + 1$ аргмин равен 3, где f равна 1; минимальное значение равно 1, но аргмин — это место, 3. Книга предполагает один выход, оптимизацию с одной целью и отсутствие ограничений на то, куда может уходить $\mathbf{x}$.



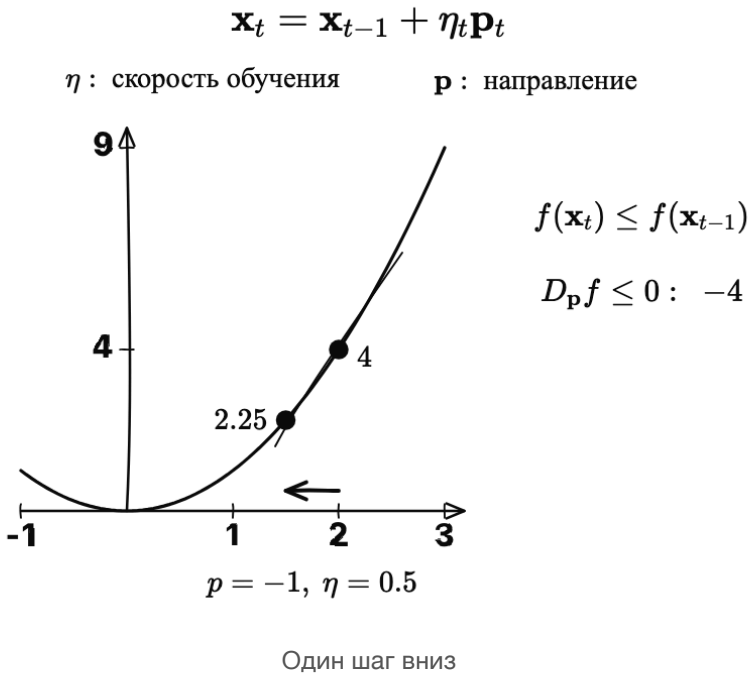
Аргмин — это место

В остальной книге x будет содержать параметры модели, а f будет измерять, насколько плохо модель справляется с данными, — постановка обучения с учителем из следующей главы. Для фотоприложения в вашем телефоне x — это его миллионы весов, а f считает его ошибки на размеченных фотографиях.

Минимизация ничего не теряет по сравнению с максимизацией, поскольку минимизировать f — то же, что максимизировать минус f : переверните кривую относительно горизонтальной оси, и её впадины станут пиками.

Редко минимум можно записать формулой, в замкнутом виде; книга встретит один такой случай с методом наименьших квадратов в разделе 4.1.2. Обычно приходится итерировать.

Начнём со случайной догадки x_0 и на каждой итерации сделаем шаг из длины η_t и направления $\mathbf{p}_t$: $\mathbf{x}_t = \mathbf{x}_{t-1} + \eta_t \mathbf{p}_t$. Словами: новая догадка — это старая догадка плюс смещение. η_t называется размером шага или, в машинном обучении, скоростью обучения.



Направление называется направлением спуска, если некоторая длина шага вдоль него не увеличивает f : $f(\mathbf{x}_t) \leq f(\mathbf{x}_{t-1})$. Если мы можем выбирать направление спуска на каждой итерации и аккуратно подбирать размеры шага, процедура достигает минимума, в том смысле, который книга вскоре уточнит. Для дифференцируемых функций производная по направлению называется каждое такое направление точно: $\mathbf{p}_t$ направление спуска $\Rightarrow D_{\mathbf{p}_t} f(\mathbf{x}_{t-1}) \leq 0$, направление, в котором f не растёт, по

крайней мере в первом порядке. Попробуйте $f(x) = x^2$ из точки 2: производная равна 4, так что у направления минус 1 производная по направлению равна минус 4, она отрицательна, и шаг 0,5 вдоль него приводит в 1,5, где f падает с 4 до 2,25.

Опыт

Нарисуйте на бумаге кривую x^2 в квадрате и положите на неё монетку в точке x , равной 2. Сдвиньте монетку на полединицы влево, в направлении, где, по наклону, кривая идёт вниз, и прочитайте высоту: 2,25 вместо 4. Сдвиньте её вместо этого вправо, и высота вырастет до 6,25; знак наклона заранее сказал вам, где спуск.

Запомним

Для $y = Wx$ производная по x — это просто W , матрица o на d . По W якобиан имеет форму (o, o, d) , или (o, od) после разворачивания, с $\partial y_i / \partial W_{ij} = x_j$: в основном нули и копии, поэтому шестая глава избегает его хранить. Оптимизация ищет $x^* = \operatorname{argmin} f(x)$; в книге x содержит параметры модели, а f — её ошибку, и минимизация — то же самое, что максимизация $-f$. Итеративные методы обновляют $x_t = x_{t-1} + \eta_t r_t$ со скоростью обучения η_t , а направление спуска — то, у которого производная по направлению неположительна, как показал шаг от 2 к 1,5 на x^2 .

Вопросы

1. Для $y = Wx$, где W имеет форму (o, d) , почему якобиан по W на практике никогда не хранят?
 - a. Потому что это всегда единичная матрица
 - b. Потому что он имеет форму (o, o, d) , в основном из нулей и копий x , а нам нужно лишь применять его к другому тензору
 - c. Потому что его нельзя вычислить для матричных входов
2. В чём разница между минимумом $f(x) = (x - 3)^2 + 1$ и его аргмином?
 - a. Минимум равен 1, это наименьшее значение; аргмин равен 3, это место, где оно достигается
 - b. Минимум равен 3, а аргмин равен 1
 - c. Это одно и то же число
3. Почему книга ничего не теряет, всегда говоря о минимизации?
 - a. Потому что ни у одной полезной функции нет максимума
 - b. Потому что градиентный спуск не умеет находить максимумы
 - c. Потому что максимизировать f — то же самое, что минимизировать $-f$
4. Когда направление p является направлением спуска для дифференцируемой f в текущей догадке?

- а. Когда оно указывает прочь от начальной догадки x_0
- б. Когда производная f по направлению p не больше нуля
- с. Когда его длина равна скорости обучения

Ответы

1. **б)** Поскольку $\partial y_i / \partial W_{ij} = x_j$, якобиан повторяет x , а в остальном нулевой; для слоя 1000 на 1000 он содержал бы миллиард чисел, и шестая глава избегает его с помощью произведений вектора на якобиан.
2. **а)** Аргмин называет вход, дающий наименьшее значение f ; здесь f наименьшая, равная 1, при $x = 3$.
3. **с)** Переворот функции относительно горизонтальной оси превращает её пики во впадины, так что любую максимизацию можно записать как минимизацию.
4. **б)** Направление спуска — то, вдоль которого f не растёт; для дифференцируемых функций это значит $D_p f \leq 0$, как направление -1 для x^2 в точке 2 с производной по направлению -4 .

Разобранные примеры

Якобиан Wx по W имеет форму $(2, 3)$, а $x = (1, 2, 3)$.

Используя $\partial y_i / \partial W_{ij} = x_j$, назовите форму якобиана $y = Wx$ по W и скажите, сколько его элементов равны нулю.

$o = 2$ и $d = 3$, так что якобиан имеет форму $(o, o, d) = (2, 2, 3)$, 12 элементов.

Выход y_1 зависит только от строки 1 матрицы W : его скорости равны $x_1, x_2, x_3 = 1, 2, 3$, и 0 для трёх весов строки 2.

Аналогично y_2 скорости 1, 2, 3 для строки 2 и 0 для строки 1.

Так что 6 элементов — нули, а остальные 6 — x , скопированный дважды.

Ответ: Форма (2, 2, 3), или (2, 6) после разворачивания; 6 из 12 элементов равны нулю.

Задачи

1. Для W со строками (1, 2, 0) и (0, 1, 4) какова производная Wx по x и как изменятся выходы, если только x_2 сдвинуть на 0,1?

Подсказка: Производная по x — это сама W ; посмотрите на её второй столбец.

Ответ: Это W . Сдвиг x_2 на 0,1 сдвигает выходы на $2 \times 0,1 = 0,2$ и $1 \times 0,1 = 0,1$, второй столбец W , умноженный на шаг.

2. $f(x) = x^2$, текущая догадка равна 3. Возьмите направление $p = -1$ с размером шага $\eta = 0,5$. Это направление спуска, и каковы новая догадка и новое значение f ?

Подсказка: Проверьте знак $f'(3) \times p$, затем примените $x_t = x_{t-1} + \eta p$.

Ответ: $f'(3) = 6$, так что производная по направлению равна $6 \times (-1) = -6 \leq 0$: направление спуска. Новая догадка $3 + 0,5 \times (-1) = 2,5$, и f падает с 9 до 6,25.

3. У слоя 500 входов и 200 выходов. Сколько чисел содержал бы его якобиан по W , если хранить его явно?

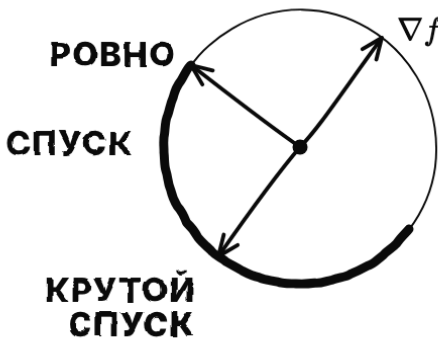
Подсказка: Форма — (0, 0, d).

Ответ: $200 \times 200 \times 500 = 20\,000\,000$ чисел, хотя сама W содержит лишь 100 000 весов.

Наискорейший спуск и где он заканчивается

Вернитесь на склон холма и встаньте лицом вверх, туда, где склон круче всего. Теперь развернитесь: вы смотрите прямо вниз, по самому крутому пути. Повернитесь вместо этого на четверть, и вы пойдёте по ровному, ни вверх, ни вниз; всё, что между четвертными поворотами с нижней стороны, ведёт вас ниже. Этот простой компас, собранный из градиента и косинуса, — и есть весь градиентный спуск.

$$D_{\mathbf{p}}f = \|\nabla f\| \|\mathbf{p}\| \cos \alpha$$



$$\nabla f(3, 4) = (6, 8), \quad \|\nabla f\| = 10$$

$$(-0.6, -0.8) : -10$$

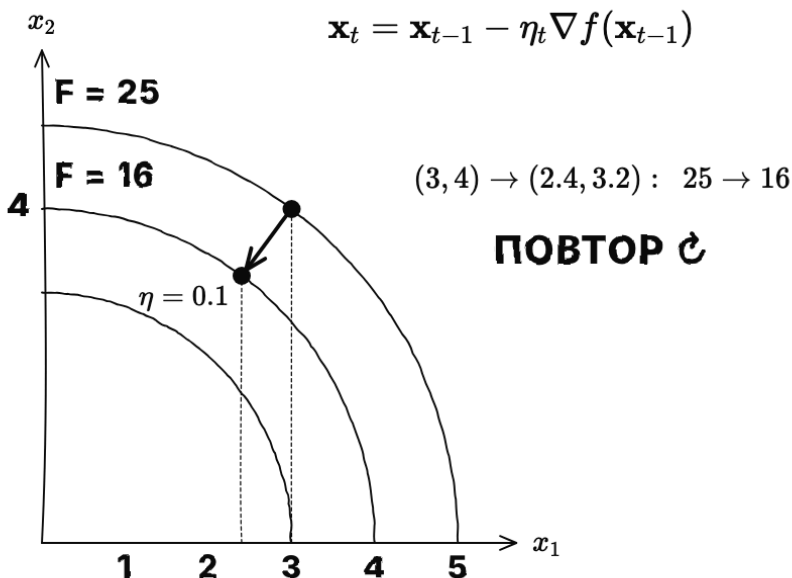
$$(-0.8, 0.6) : 0$$

Компас из градиента

Соединим производную по направлению с косинусной формой скалярного произведения: $D_{\mathbf{p}_t} f(\mathbf{x}_{t-1}) = \langle \nabla f(\mathbf{x}_{t-1}), \mathbf{p}_t \rangle = \|\nabla f(\mathbf{x}_{t-1})\| \|\mathbf{p}_t\| \cos(\alpha)$, где α — угол между направлением шага и градиентом. Первая длина не зависит от направления, которое мы выбираем, а поскольку направление лишь говорит, куда идти, его длину можно положить равной единице. Так что всё зависит от косинуса.

Косинус отрицателен для углов между $\pi/2$ и $3\pi/2$, четвертью оборота и тремя четвертями, так что каждое направление в этом диапазоне — направление спуска. Среди них то, что под углом π , указывающее точно против градиента, $\mathbf{p}_t = -\nabla f(\mathbf{x}_{t-1})$, даёт самую отрицательную производную по направлению, и оно называется направлением наискорейшего спуска. Возьмём $f(\mathbf{x}) = x_1^2 + x_2^2$ в точке $(3, 4)$: градиент равен $(6, 8)$, длиной 10. Единичное направление против него, $(-0,6, -0,8)$, даёт производную по направлению минус 10, самую крутую из возможных; боковое направление $(-0,8, 0,6)$ даёт минус 4,8 плюс 4,8, то есть ноль: идём по ровному.

Подставив это в итеративный шаг, получаем алгоритм, способный минимизировать любую дифференцируемую функцию, — наискорейший градиентный спуск. Дана начальная точка $\mathbf{x}_0$ и последовательность размеров шага η_t ; повторяем $\mathbf{x}_t = \mathbf{x}_{t-1} - \eta_t \nabla f(\mathbf{x}_{t-1})$. Словами: оттуда, где вы стоите, шагните немного против градиента. С нашей функцией шаг размером 0,1 из точки $(3, 4)$ переводит в 3 минус 0,6 и 4 минус 0,8, то есть $(2,4, 3,2)$, и f падает с 25 до 5,76 плюс 10,24, то есть 16.



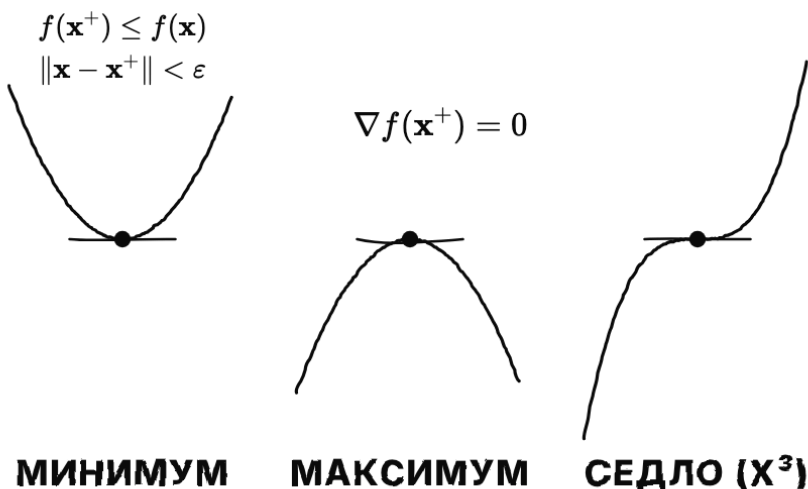
Один шаг градиентного спуска

Книга откладывает вопрос о выборе размера шага и просто предполагает, что он достаточно мал, чтобы каждый шаг уменьшал f . Она также замечает, что градиентный спуск быстр лишь настолько, насколько быстр наш способ вычисления градиента, поэтому шестая глава вводит для этого общий алгоритм. Когда обучали ваше фотоприложение, именно этот цикл повторялся огромное число раз: вычислить градиент ошибки по миллионам весов, шагнуть немного против него.

Куда приходит градиентный спуск? Сначала нужно сказать, что такое минимум. Локальный минимум — это точка $\mathbf{x}^+$ такая, что для некоторого малого радиуса $\varepsilon > 0$ выполняется $f(\mathbf{x}^+) \leq f(\mathbf{x})$ для всех $\mathbf{x}$ с $\|\mathbf{x} - \mathbf{x}^+\| < \varepsilon$. Словами: в достаточно малом шаре вокруг неё

ничего нет ниже. Интуитивно в такой точке касательная плоская, а градиент вокруг неё указывает вверх, прочь от неё.

ГС ДОХОДИТ ДО СТАЦИОНАРНОЙ ТОЧКИ



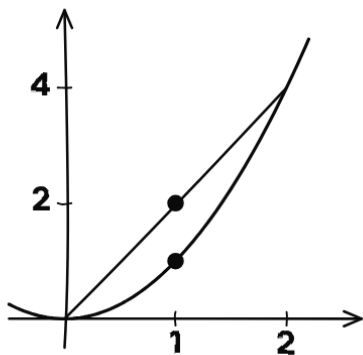
Стационарные точки

Идея плоской касательной получает своё имя: стационарная точка — та, где $\nabla f(\mathbf{x}^+) = 0$. Но стационарные точки — не только минимумы: это могут быть максимумы, которые являются минимумами минус f , или седловые точки, где меняется кривизна функции. Рисунок в книге показывает седло; в одном измерении классический случай — x в кубе в нуле: его наклон, $3x$ в квадрате, там равен нулю, но кривая продолжает подниматься с обеих сторон. Без предположений об f можно доказать лишь то, что градиентный спуск достигает какой-то

стационарной точки, а какой именно — зависит от того, откуда он начал.

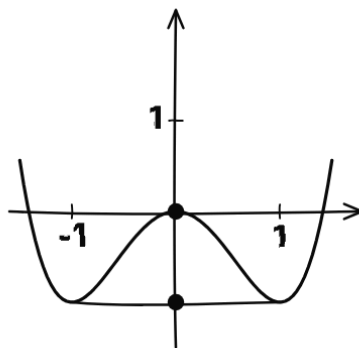
Можно ли лучше? Представьте параболу. У неё нет седловых точек и один-единственный минимум, и этот минимум особенный: наименьшее значение на всей области. Глобальный минимум — это точка $\mathbf{x}^*$ с $f(\mathbf{x}^*) \leq f(\mathbf{x})$ для любого $\mathbf{x}$. На параболе градиентный спуск достигает его из любой начальной точки, потому что каждый градиент указывает к нему.

$$f(\alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)$$



$$1 < 2 \checkmark$$

ГЛОБ. МИН. В 0



$$0 > -1 \times$$

ДВЕ ВПАДИНЫ

Ниже любой хорды

Выпуклость обобщает параболу. Функция выпукла, если для любых двух точек $\mathbf{x}_1$ и $\mathbf{x}_2$ и любого $\alpha \in [0, 1]$ выполняется $f(\alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)$. Словами: на отрезке между двумя

точками функция никогда не поднимается выше прямой хорды, соединяющей их значения. Для x в квадрате с точками 0 и 2 и α , равным одной второй: середина — это 1, где f равна 1, а середина хорды — среднее 0 и 4, то есть 2; 1 ниже 2, как и требует выпуклость. Для $x^4 - 2x^2$ с точками минус 1 и 1 середина 0 даёт 0, а среднее хорды от минус 1 и минус 1 равно минус 1: кривая выпирает выше хорды, так что она не выпукла, и действительно, у неё две отдельные впадины.

Опыт

Нарисуйте кривую в форме чаши, x в квадрате, и отметьте на ней две точки, в 0 и в 2. Соедините их высоты прямой линией по линейке: каждая точка кривой между ними лежит под линейкой. Теперь нарисуйте кривую в форме буквы W с двумя впадинами и положите линейку через обе впадины: горб посередине торчит над ней, а это ровно то, что выпуклость запрещает.

Запомним

Производная по направлению равна длине градиента на длину шага на $\cos \alpha$, так что каждое направление дальше четверти оборота от градиента идёт вниз, а самое крутое — минус градиент. Градиентный спуск повторяет $x_t = x_{t-1} - \eta_t \nabla f(x_{t-1})$: для $x_1^2 + x_2^2$ из $(3, 4)$ с шагом $0,1$ он приходит в $(2,4, 3,2)$, и f падает с 25 до 16. Локальный минимум — самая низкая точка в малом шаре; у стационарной точки нулевой градиент, и она может быть минимумом, максимумом или седлом, и в общем случае градиентный спуск достигает лишь какой-то стационарной точки. Глобальный минимум — самый низкий везде; выпуклые функции, лежащие ниже любой хорды, обобщают параболу, на которой градиентный спуск достигает глобального минимума откуда угодно.

Вопросы

1. Почему минус градиент называют направлением наискорейшего спуска?
 - a. Потому что это единственное направление, в котором f убывает
 - b. Потому что под углом π к градиенту косинус равен -1 , что даёт самую отрицательную производную по направлению
 - c. Потому что оно всегда ведёт прямо к глобальному минимуму

2. Градиентный спуск останавливается в точке, где градиент равен нулю. Что можно сказать об этой точке в общем случае?
- a. Это стационарная точка, которая может быть минимумом, максимумом или седловой точкой
 - b. Это всегда глобальный минимум
 - c. Это всегда локальный минимум
3. Что гарантирует выпуклость, из-за чего градиентный спуск ведёт себя как на параболе?
- a. Что размер шага можно выбирать свободно
 - b. Что градиент никогда не равен нулю
 - c. Что функция никогда не поднимается выше хорды между двумя точками, что исключает отдельные впадины и сёдла
4. Направление ровно под прямым углом к градиенту. Что малый шаг вдоль него делает с f в первом порядке?
- a. Уменьшает f как можно быстрее
 - b. Оставляет f неизменной, поскольку косинус прямого угла равен нулю
 - c. Увеличивает f как можно быстрее

Ответы

1. **b)** Для направлений единичной длины производная по направлению равна длине градиента на $\cos \alpha$; каждый угол между $\pi/2$ и $3\pi/2$ идёт вниз, но $\cos \alpha$ наименьший, -1 , при $\alpha = \pi$.
2. **a)** Нулевой градиент определяет стационарную точку, и без предположений об f можно доказать лишь, что градиентный спуск

достигает какой-то стационарной точки, в зависимости от того, откуда начал.

3. c) Выпуклая функция лежит ниже любой хорды, обобщая параболу, единственный минимум которой глобален и к которому указывают все градиенты.

4. b) Производная по направлению пропорциональна $\cos \alpha$, который равен 0 при четверти оборота, как $(-0,8, 0,6)$ против градиента $(6, 8)$ дало $-4,8 + 4,8 = 0$.

Разобранные примеры

Один шаг градиентного спуска Минимизируйте $f(x) = x_1^2 + x_2^2$, начиная с $(3, 4)$, с размером шага 0,1. Найдите градиент, новую точку и новое значение f .

Градиент равен $(2x_1, 2x_2) = (6, 8)$ в точке $(3, 4)$.

Обновление: $(3, 4) - 0,1 \times (6, 8) = (3 - 0,6, 4 - 0,8) = (2,4, 3,2)$.

Старое значение: $9 + 16 = 25$.

Новое значение: $2,4^2 + 3,2^2 = 5,76 + 10,24 = 16$.

Ответ: Новая точка $(2,4, 3,2)$; f падает с 25 до 16

Проверка выпуклости одной хордой Проверьте неравенство выпуклости для $f(x) = x^4 - 2x^2$ при $x_1 = -1$, $x_2 = 1$ и $\alpha = 0,5$.

Смешанная точка равна $0,5 \times (-1) + 0,5 \times 1 = 0$, и $f(0) = 0$.

$f(-1) = 1 - 2 = -1$ и $f(1) = -1$, так что значение хорды равно $0,5 \times (-1) + 0,5 \times (-1) = -1$.

Выпуклость требовала бы $0 \leq -1$, что неверно.

Ответ: Неравенство не выполняется, так что $x^4 - 2x^2$ не выпукла; у неё две отдельные впадины.

Задачи

1. Для $f(x) = x^2$, начиная с 5 с размером шага 0,25, вычислите два шага градиентного спуска.

Подсказка: Градиент равен $2x$; обновление $x \leftarrow x - 0,25 \times 2x$.

Ответ: Шаг 1: $5 - 0,25 \times 10 = 2,5$. Шаг 2: $2,5 - 0,25 \times 5 = 1,25$. Каждый шаг делит x пополам, двигаясь к минимуму в 0.

2. Является ли $x = 0$ стационарной точкой $f(x) = x^3$? Является ли она минимумом?

Подсказка: Вычислите производную в нуле, затем посмотрите на f с обеих сторон.

Ответ: $f'(x) = 3x^2$, что равно 0 в нуле, так что точка стационарная.

Это не минимум: f отрицательна чуть левее и положительна правее, так что это точка, где кривая выравнивается и продолжает расти, как седло в одном измерении.

3. Проверьте неравенство выпуклости для $f(x) = x^2$ при $x_1 = 1$, $x_2 = 3$ и $\alpha = 0,25$.

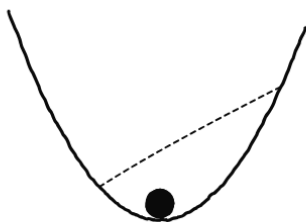
Подсказка: Сравните f в точке $0,25 \times 1 + 0,75 \times 3$ с $0,25 f(1) + 0,75 f(3)$.

Ответ: Точка равна $0,25 + 2,25 = 2,5$, и $f(2,5) = 6,25$. Хорда даёт $0,25 \times 1 + 0,75 \times 9 = 7$. Поскольку $6,25 \leq 7$, неравенство выполняется, как и требует выпуклость.

Чаши, бугристые ландшафты и толчок импульса

Пустите шарик в гладкую салатницу, и он всегда окажется на дне, откуда бы вы его ни бросили. Пустите его по лотку для яиц, и он уляжется в ту ячейку, какая окажется ближе. А придайте шарик у немного веса, чтобы он сохранял часть своей скорости, и он покатится ко дну плавнее и быстрее. Эти три картинки и есть эта часть: что гарантирует выпуклость, что мы теряем без неё и как импульс ускоряет дело.

ВЫПУКЛАЯ



**ЛЮБОЙ СТАРТ → МИН.
СТРОГАЯ → ОДИН**

НЕВЫПУКЛАЯ



**ЛИШЬ СТАЦИОНАРНАЯ
ТОЧКА**

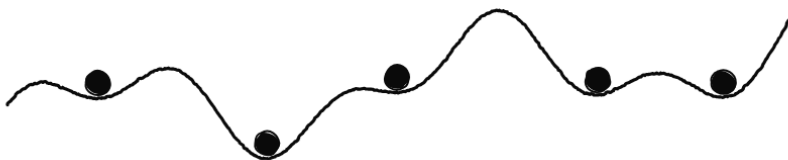
Чаша или лоток для яиц

Неравенство выпуклости сравнивает две вещи. Слева — значение f в любом месте отрезка от x_1 до x_2 ; справа — соответствующая точка на прямой, соединяющей $f(x_1)$ и $f(x_2)$. Если функция всегда лежит ниже каждой такой прямой, она выпукла; парабола, раскрытая вверх, — стандартный пример, та самая салатница.

Выпуклость говорит нам, насколько проста оптимизация. Для общей невыпуклой функции градиентный спуск достигает стационарной точки, и больше ничего нельзя сказать, не глядя на старшие производные, производные от производных. Для выпуклой функции градиентный спуск достигает глобального минимума, какой бы ни была начальная точка. А если неравенство выполняется строго, с «меньше» вместо «меньше или равно», что называется строгой выпуклостью, этот глобальный минимизатор ещё и единственный.

Разница серьёзная. Чтобы наверняка найти глобальный минимум невыпуклой задачи градиентным спуском, его пришлось бы запускать из каждой возможной начальной точки, бесконечно много раз, что делает задачу NP-трудной, одной из самых трудных для решения. Это лоток для яиц: каждый старт находит ячейку, но ничто не говорит вам, какая ячейка самая низкая.

? ГЛОБ. МИН.: НР-ТРУДНО



**СЕТИ:
НЕВЫПУКЛЫЕ**

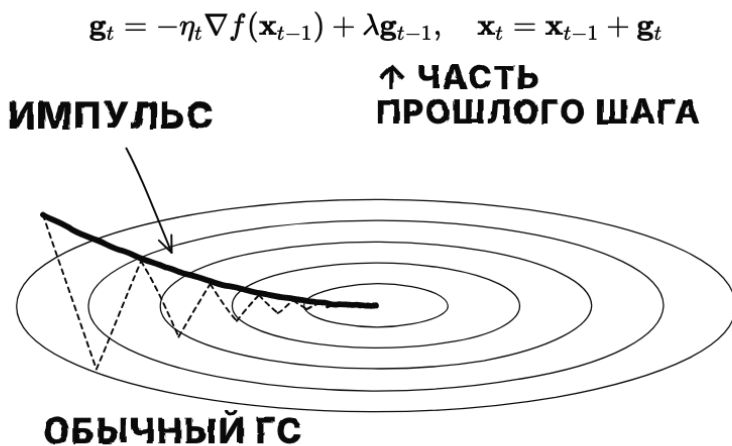
SVM: ВЫПУКЛЫЙ

✓ **ХОРОШИЕ ТОЧКИ НА ПРАКТИКЕ** ЧАТ · ФОТО

Какая ячейка самая низкая?

Это важно исторически. Как покажет пятая глава, каждая нетривиальная нейронная сеть невыпукла, так что в её оптимизации может быть много стационарных точек. Другие методы обучения, такие как метод опорных векторов, сохраняют нелинейное поведение, при этом имея выпуклую задачу оптимизации, что долго считалось сильным преимуществом. И всё же, что интересно, сложные дифференцируемые модели всё равно работают хорошо: начатая с разумной инициализации, их оптимизация приходит в точки, которые хорошо работают на практике. Чат-ассистент и фотоприложение в вашем телефоне были обучены именно так, на лотках для яиц с миллиардами измерений, и они работают.

Теперь ускорение. Отрицательный градиент — самый крутой путь вниз только в бесконечно малой окрестности текущей точки, и, как покажет пятая глава на примере стохастической оптимизации, для больших моделей эти направления могут быть очень шумными. Многие приёмы выбирают лучшие направления спуска; по вычислительным причинам книга предпочитает те, которым не нужны старшие производные, такие как гессиан, матрица вторых производных, и лишние вызовы функции.



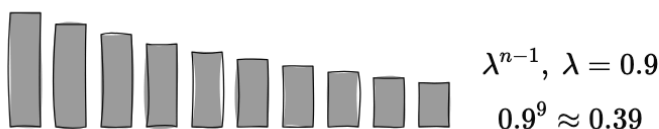
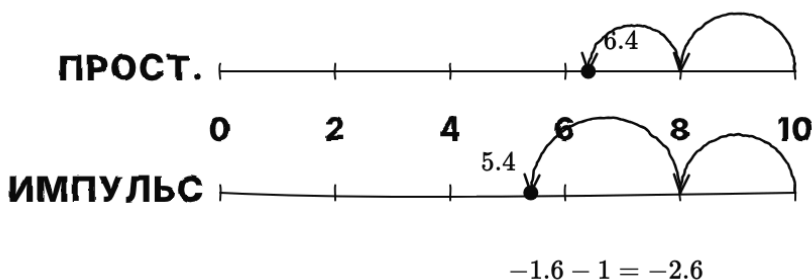
Зигзаг против импульса

Тот приём, который она описывает, — это импульс. Представьте градиентный спуск как шарик, катящийся вниз: его путь дёрганный, потому что каждый градиент может указывать совсем

в другую сторону; на самом деле при идеально выбранном шаге на выпуклой функции два последовательных градиента перпендикулярны. Импульс сглаживает это, сохраняя часть предыдущего шага: $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1}$, затем $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{g}_t$, начиная с $\mathbf{g}_0 = \mathbf{0}$. Словами: новый шаг — это обычный шаг наискорейшего спуска плюс доля λ предыдущего шага.

Давайте запустим его. Возьмём $f(x) = x^2$, начнём с 10, с размером шага 0,1 и $\lambda = 0,5$. Первый шаг: градиент равен 20, так что шаг равен минус 2, и x становится 8, как и при обычном градиентном спуске. Второй шаг: градиент равен 16, обычная часть равна минус 1,6, а импульс добавляет половину предыдущих минус 2, давая минус 2,6, так что x становится 5,4. Обычный градиентный спуск дошёл бы только до 6,4: запомненный толчок заставил шарик укатиться дальше.

$$\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) - \lambda \eta_t \nabla f(\mathbf{x}_{t-2}) + \lambda^2 \mathbf{g}_{t-2}$$



Импульс катится дальше

Развернув два шага, видим, как память угасает: $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) - \lambda \eta_t \nabla f(\mathbf{x}_{t-2}) + \lambda^2 \mathbf{g}_{t-2}$. В общем случае градиент из n шагов назад подавляется множителем λ^{n-1} . При λ , равном 0,9, градиент десятишаговой давности всё ещё учитывается с весом 0,9 в девятой степени, около 0,39: у импульса долгая память. Рисунок в книге сравнивает обычный градиентный спуск и спуск с импульсом на $x \sin(2x)$, начиная чуть дальше 1, с λ , равным 0,3, и импульс показывает более гладкий путь.

Другая распространённая идея — подстраивать размер шага для каждого параметра отдельно, в зависимости от того, насколько большими были его градиенты. Широко используемый оптимизатор Adam объединяет несколько таких идей. Одно из

его преимуществ в том, что он довольно устойчив к выбору своих гиперпараметров, настроек, которые выбирает пользователь, а не выучивает градиентный спуск, таких как размер шага и λ ; значения по умолчанию в библиотеках — хорошее начало в большинстве случаев. Разработка нового оптимизатора, который смог бы сместить Adam или его варианты, такие как AdamW, с места стандарта в глубоком обучении, остаётся открытой исследовательской задачей, включая недавние работы, проектирующие оптимизаторы для нейронных сетей из первых принципов.

Опыт

На калькуляторе выполните обычный градиентный спуск на x^2 в квадрате из 10 с шагом 0,1: каждый шаг умножает x на 0,8, давая 8, 6,4, 5,12. Теперь выполните спуск с импульсом при λ , равном 0,5: 8, затем 5,4, затем сами вычислите третий шаг: градиент 10,8 даёт минус 1,08, плюс половина от минус 2,6, итого минус 2,38, так что x равен 3,02. Сравните 3,02 с 5,12 обычного спуска после тех же трёх шагов.

Запомним

На выпуклой функции градиентный спуск достигает глобального минимума из любого старта, а при строгой выпуклости этот минимизатор единственный; на невыпуклой он достигает лишь стационарной точки, и поиск глобального минимума становится NP-трудным. Нейронные сети невыпуклы, в отличие от метода опорных векторов, но из разумного старта они приходят в точки, которые хорошо работают на практике. Импульс добавляет долю λ предыдущего шага, $g_t = -\eta_t \nabla f(x_{t-1}) + \lambda g_{t-1}$, так что старые градиенты угасают как λ^{n-1} ; на x^2 из 10 он за два шага дошёл до 5,4 вместо 6,4. Adam объединяет импульс с размерами шага для каждого параметра, устойчив к своим гиперпараметрам и остаётся стандартом, который пытаются превзойти новые оптимизаторы.

Вопросы

1. Что гарантирует градиентный спуск на выпуклой функции, чего он не может гарантировать на невыпуклой?
 - a. Достижение глобального минимума из любой начальной точки
 - b. Меньшее число шагов, чем с импульсом
 - c. Что он никогда не встретит точку с нулевым градиентом

2. Почему поиск глобального минимума невыпуклой задачи градиентным спуском называют NP-трудным?
- a. Потому что каждый шаг градиента слишком дорог
 - b. Потому что для уверенности пришлось бы запускать его из каждой возможной начальной точки
 - c. Потому что у невыпуклых функций нет минимума
3. В импульсе $g_t = -\eta_t \nabla f(x_{t-1}) + \lambda g_{t-1}$. Чем управляет λ ?
- a. Тем, насколько велик текущий градиентный шаг
 - b. Тем, сколько градиентов вычисляется за шаг
 - c. Тем, какая часть предыдущего шага сохраняется, так что старые градиенты угасают степенями λ
4. Почему Adam часто является удобным выбором?
- a. Потому что он делает нейронные сети выпуклыми
 - b. Потому что он довольно устойчив к своим гиперпараметрам, и значения по умолчанию работают в большинстве случаев
 - c. Потому что ему нужен гессиан, что делает его точным

Ответы

1. **a)** На выпуклой функции градиентный спуск достигает глобального минимума независимо от инициализации; на невыпуклой он достигает лишь какой-то стационарной точки.
2. **b)** Каждый запуск находит какую-то стационарную точку, как шарик, оседающий в одной ячейке лотка для яиц, и только перебор всех стартов гарантировал бы самую низкую.
3. **c)** λ — это доля предыдущего шага, которая переносится дальше; развёртывание показывает, что градиент из n шагов назад взвешен с

λ^{n-1} .

4. b) Adam объединяет импульс с размерами шага для каждого параметра и оказывается относительно устойчивым к выбору своих гиперпараметров, так что значения по умолчанию в библиотеках — хорошее начало.

Разобранные примеры

Два шага с импульсом Минимизируйте $f(x) = x^2$ из $x = 10$ с размером шага $0,1$ и $\lambda = 0,5$, начиная с $g_0 = 0$. Сравните с обычным градиентным спуском после двух шагов.

Шаг 1: градиент $2 \times 10 = 20$, так что $g_1 = -0,1 \times 20 + 0,5 \times 0 = -2$ и $x = 8$.

Шаг 2: градиент 16 , так что $g_2 = -1,6 + 0,5 \times (-2) = -2,6$ и $x = 8 - 2,6 = 5,4$.

Обычный спуск: $10 \rightarrow 8 \rightarrow 8 - 1,6 = 6,4$.

Импульс ближе к минимуму в 0 после тех же двух шагов.

Ответ: Импульс достигает $5,4$, обычный спуск $6,4$

Задачи

1. Продолжите пример с импульсом на третий шаг из $x = 5,4$ при $g_2 = -2,6$.

Подсказка: Градиент равен $2 \times 5,4$; добавьте половину g_2 .

Ответ: Градиент $10,8$, обычная часть $-1,08$, плюс $0,5 \times (-2,6) = -1,3$, так что $g_3 = -2,38$ и $x = 5,4 - 2,38 = 3,02$, против $5,12$ для обычного спуска.

2. При $\lambda = 0,9$ с каким множителем входит градиент из 4 шагов назад в текущий шаг импульса?

Подсказка: Градиент из n шагов назад подавляется множителем λ^{n-1} .

Ответ: $0,9^3 = 0,729$.

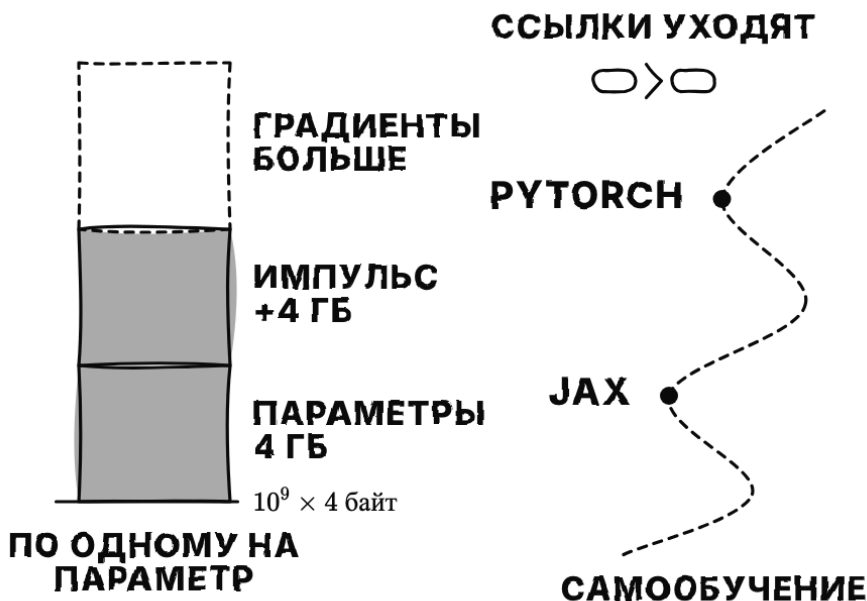
3. Объясните, почему нейронные сети успешно обучаются, хотя их оптимизация невыпукла.

Подсказка: Вспомните, что книга говорит о начальных точках и практической производительности.

Ответ: Невыпуклость означает много стационарных точек и отсутствие гарантии глобального минимума. Но, как замечает книга, при разумной инициализации оптимизация приходит в точки с хорошей производительностью на практике, так что глобальная гарантия не нужна, чтобы модели работали хорошо.

Цена памяти и где практиковаться

Шарик с импульсом помнит свой последний толчок, а память стоит места. Импульс держит в памяти предыдущий шаг, по одному числу на каждый параметр, что удваивает место, нужное самому оптимизатору. В числах: модель с миллиардом параметров, каждый хранится в 4 байтах, занимает 4 гигабайта; память импульса о последнем шаге добавляет ещё 4. Книга добавляет, что в большинстве случаев ещё важнее память, нужная для вычисления градиента, и возвращается к этому в разделе 6.3.



Импульс стоит памяти

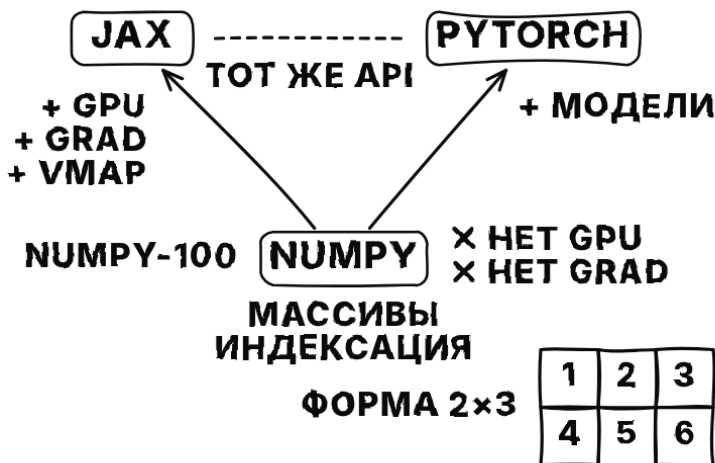
Затем глава переходит от теории к практике. В книге нет классических упражнений в конце глав, поскольку многие учебники их уже предлагают. Вместо этого она предлагает путь самостоятельного обучения через две библиотеки, JAX и PyTorch, который растёт по ходу книги, с решениями, которые будут опубликованы на сайте книги. Она предупреждает, что эти разделы полны ссылок на материалы в сети, которые могли переехать или устареть к тому времени, когда вы их откроете.

Отправная точка для любого, кто проектирует дифференцируемые модели, говорит книга, — внимательное изучение NumPy. NumPy даёт общий набор функций для работы с многомерными массивами, тем, что книга называет тензорами, вместе с функциями для индексации и преобразования их содержимого. Вы должны уверенно обращаться с массивами в NumPy, прежде всего индексировать их, те срезы и двоеточия, которые мы встречали раньше. Начать стоит с краткого руководства библиотеки, а сборник из ста упражнений по NumPy, репозиторий [rougier/numpy-100](https://github.com/rougier/numpy-100), даёт неспешный способ проверить себя.

ФОТОПРИЛОЖЕНИЕ



ЧАТ



NumPy, затем JAX и PyTorch

При всём своём влиянии у NumPy есть пределы. Без дополнительных библиотек у него слабая поддержка параллельного оборудования, такого как GPU, и нет автоматического дифференцирования, механизма шестой главы. JAX копирует интерфейс NumPy и добавляет более широкую поддержку оборудования, автоматическое вычисление градиентов и дополнительные преобразования, такие как векторизованное отображение `jax.vmap`, которое применяет функцию сразу к целому батчу. У PyTorch тоже ядро в духе NumPy, с небольшими отличиями в названиях и поведении, и он добавляет высокоуровневые инструменты для построения дифференцируемых моделей.

Книга предлагает пробежать документацию `jax.numpy.array` и `torch.tensor`, чтобы увидеть, как много у обоих общего с NumPy. Высокоуровневые модули, такие как `torch.nn`, пока можно не трогать; книга объясняет, как устроены эти библиотеки, в шестой главе, после того как введён их механизм градиентов. Модели за вашим фотоприложением и вашим чат-ассистентом почти наверняка написаны на одной из этих двух библиотек, так что этот путь ведёт прямо к инструментам реальных систем.

Опыт

Если у вас под рукой Python, создайте в NumPy маленький массив с числами от 1 до 6 в 2 строках по 3 и запросите его форму: он должен ответить 2 на 3. Сделайте то же самое с `jax.numpy` или `torch` и заметьте, что команды почти одинаковы. Эта почти полная одинаковость и есть то, что книга имеет в виду, говоря, что эти библиотеки воспроизводят интерфейс NumPy.

Запомним

Ускоренные оптимизаторы стоят памяти: импульс хранит предыдущий шаг, удваивая место оптимизатора, ещё 4 гигабайта для миллиарда параметров, хотя вычисление градиентов обычно требует ещё больше. Вместо классических упражнений книга предлагает путь самостоятельного обучения через JAX и PyTorch с решениями в сети. Начните с NumPy, стандартной библиотеки для массивов, и практикуйте индексацию, например на упражнениях numpy-100. У NumPy нет простой поддержки GPU и автоматического дифференцирования; JAX копирует его интерфейс и добавляет и то, и другое, плюс преобразования вроде vmap, а PyTorch предлагает ядро в духе NumPy с инструментами для построения моделей.

Вопросы

1. Почему импульс увеличивает память, нужную оптимизатору?
 - a. Потому что он хранит предыдущий шаг, по одному числу на параметр, удваивая место оптимизатора
 - b. Потому что он вычисляет вторые производные
 - c. Потому что он держит копию всего набора данных
2. Что JAX добавляет к интерфейсу NumPy?

- a. Другой способ индексации массивов
 - b. Поддержку строк и целых чисел в массивах
 - c. Более широкую поддержку оборудования, автоматические градиенты и преобразования вроде `vmmap`
3. Почему книга начинает практический путь с NumPy, а не с полноценной библиотеки?
- a. Потому что NumPy быстрее JAX и PyTorch на GPU
 - b. Потому что работа с массивами и их индексация в NumPy — основа, которую копируют JAX и PyTorch
 - c. Потому что NumPy автоматически вычисляет градиенты

Ответы

1. a) Импульсу приходится помнить предыдущий градиентный шаг для каждого параметра, удваивая место алгоритма оптимизации, хотя вычисление градиента обычно требует ещё больше памяти.
2. c) У NumPy нет простой поддержки GPU и автоматического дифференцирования; JAX воспроизводит его интерфейс, добавляя и то, и другое, плюс преобразования вроде векторизованного отображения.
3. b) NumPy — стандарт для работы с многомерными массивами, и JAX, и PyTorch строятся на интерфейсе в духе NumPy, так что уверенность в NumPy переносится на них.

Разобранные примеры

Стоимость импульса в памяти У модели 1 миллиард параметров, каждый хранится в 4 байтах. Сколько памяти занимают параметры и сколько добавляет импульс?

Параметры: $1\,000\,000\,000 \times 4$ байта = 4 000 000 000 байт, около 4 гигабайт.

Импульс хранит предыдущий шаг, по одному числу на параметр, тоже по 4 байта.

Так что он добавляет ещё 4 гигабайта, удваивая место оптимизатора.

Ответ: 4 ГБ для параметров плюс 4 ГБ для импульса

Задачи

1. У модели 7 миллиардов параметров, хранимых по 2 байта каждый. Сколько памяти занимает сохранённый шаг импульса?

Подсказка: По одному числу на параметр, того же размера, что и параметры.

Ответ: $7\,000\,000\,000 \times 2$ байта = 14 000 000 000 байт, около 14 гигабайт.

2. Объясните своими словами, почему книга предлагает пока не трогать модули вроде `torch.nn`.

Подсказка: Подумайте о том, что книга ещё не ввела.

Ответ: Это высокоуровневые инструменты для построения моделей, а книга объясняет устройство библиотек только в шестой главе, после того как вводит вычисление градиентов; пока цель — увидеть, как много общего у `jax.numpy.array` и `torch.tensor` с NumPy.

Учебный ландшафт и что такое набор данных

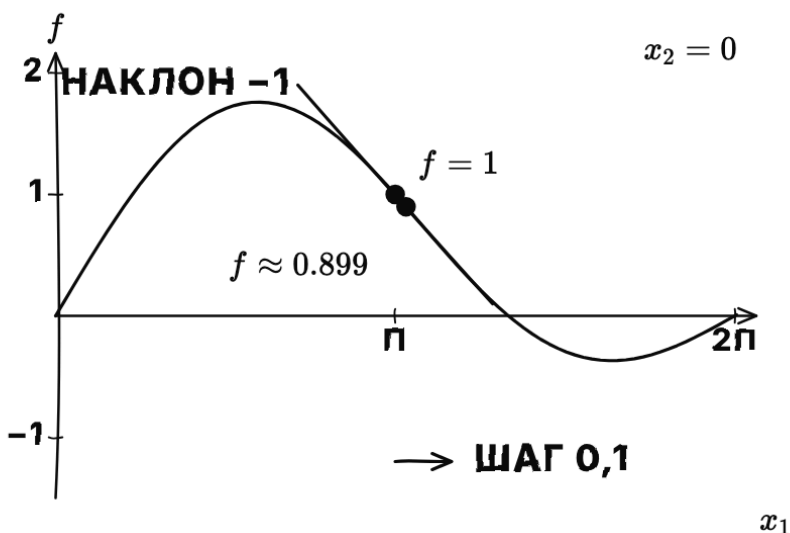
Представьте лист лотка для яиц, положенный плашмя на стол, десять на десять единиц, с бугорками и ямками, разбросанными по нему. Бросьте несколько шариков в разных местах и посмотрите, где каждый окажется. Это упражнение, которое книга задаёт перед тем, как двинуться дальше, и сразу за ним идёт вопрос, от которого зависит каждая обученная модель: откуда берутся примеры и можно ли им доверять?



Ландшафт из бугорков и ямок

Книга просит читателя повторить одно упражнение трижды, в NumPy, в JAX и в PyTorch, каждое занимает лишь минуты, как только синтаксис знаком. Ландшафт — это функция двух входов, $\mathbf{x} \sim (2)$, на квадрате от 0 до 10 в обоих направлениях: $f(\mathbf{x}) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$. Словами: две волны, перемноженные между собой, плюс более медленная копия того же узора, что даёт рельеф с несколькими пиками и впадинами. Книга добавляет честное примечание: эта функция получена вопросом к ChatGPT о функции с несколькими минимумами и максимумами, и ничто другое в книге не создано языковой моделью.

Задания идут в четыре шага. Первый: реализовать функцию в векторизованном виде: по матрице $\mathbf{X} \sim (n, 2)$ из n входных точек вернуть вектор из n значений, $[f(\mathbf{X})]_i = f(\mathbf{X}_i)$, все сразу, а не в цикле. Второй: написать функцию для градиента вручную, поскольку автоматическое дифференцирование ещё не введено. Третий: написать простой градиентный спуск и нарисовать пути из нескольких начальных точек, шарики на лотке; в JAX или PyTorch это хорошее место, чтобы попробовать vmap. Четвёртый: добавить импульс и построить график длины градиента, которая должна сжиматься к нулю по мере приближения метода к стационарной точке.



Один шаг вниз по склону

Давайте вычислим градиент вручную в одной точке, чтобы увидеть, что упражнение по силам. Частная производная по x_1 равна $\cos(x_1)\cos(x_2) + 0.5\cos(0.5x_1)\cos(0.5x_2)$, где цепное правило даёт множитель одна вторая для медленной волны. В точке $(\pi, 0)$ f равна 0 плюс синус $\pi/2$, то есть 1; первая частная производная равна минус 1 плюс одна вторая на 0, то есть минус 1, а вторая частная производная равна 0. Один шаг градиентного спуска на 0,1 поэтому поднимает x_1 примерно до 3,24, и f падает примерно до 0,899, снижение примерно на 0,1, ровно как предсказал наклон минус 1.

Теперь третья глава, которая формализует обучение с учителем: наборы данных, функции потерь, минимизацию эмпирического

риска, основные предположения и, наконец, вероятностную формулировку на основе максимального правдоподобия. Книга называет эту короткую главу хребтом всего последующего.

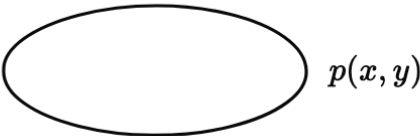
С УЧИТЕЛЕМ

x	y	
	СОБАКА	} N ПАР
	КОШКА	
	СОБАКА	
	КОШКА	
$n = 10,000$		

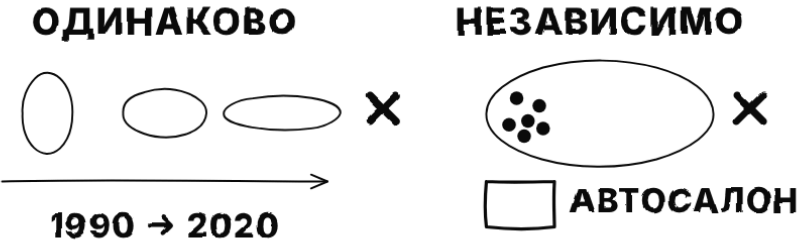
Набор данных — это список пар

Исходная ситуация: написать функцию вручную невозможно, например распознавать объекты на реальных фотографиях, но собрать примеры желаемого поведения достаточно легко. Распознавание речи и навигация роботов — ещё примеры. Определение: обучающий набор данных S_n размера n — это множество из n пар (x_i, y_i) , каждая — пример связи входа и выхода, которую мы хотим моделировать. Фотоприложение в вашем телефоне могло бы начать с 10 000 таких пар: фотография как x и слово «собака» или «кошка» как y .

Добавляется ещё одно предположение, и оно решающее: каждый пример — независимая и одинаково распределённая выборка, i.i.d., из некоторого неизвестного и, по сути, непознаваемого распределения вероятностей $p(x, y)$. Звучит технически, но именно это придаёт задаче смысл. Одинаково распределённые значит, что мы приближаем что-то устойчивое, не меняющееся со временем. Пример книги — распознавание моделей автомобилей по фотографиям: собранные за короткий период, они удовлетворяют предположению; собранные за несколько десятилетий — нет, потому что модели изменились, и модель, обученная на таких данных, пропустит новые машины или будет работать плохо.

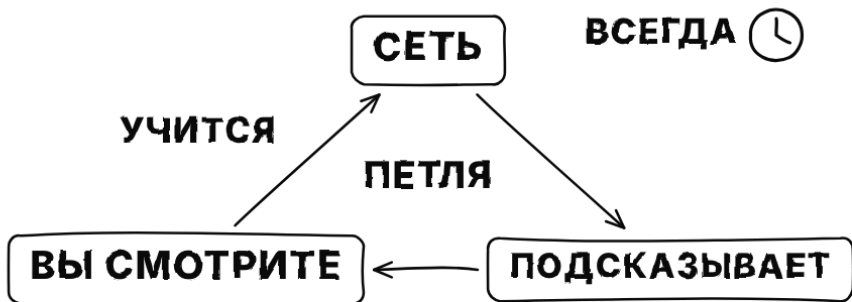


ПРОВЕРКА **I.I.D.**



Один источник, честная выборка

Независимо распределённые значат, что в сборе нет смещения и он представляет всё распределение. Соберите фотографии машин рядом с салоном Tesla, и вы получите слишком много одного вида и слишком мало всех остальных марок. Пригодность зависит и от того, где модель используется: набор данных о машинах, собранный в Италии, может служить в Риме или Милане, но не в Токио или на Тайване. Так что предположение i.i.d. всегда нужно проверять. А для современных языковых моделей, обученных на огромных распределениях данных, даже понять, какие задачи внутри распределения, а какие снаружи, и как далеко модели обобщают, становится размытым.



СДВИГ ДОМЕНА



Петля обратной связи

Блок в книге добавляет, что проверка никогда не делается раз и навсегда; её нужно продолжать всю жизнь модели. Оставленные без контроля медленные изменения в том, какие машины ездят по дорогам, ухудшат классификатор машин, пример сдвига домена. Рекомендательная система ещё хитрее: она меняет то, как люди пользуются приложением, потому что пользователи начинают реагировать на её собственные подсказки, что создаёт петли обратной связи, требующие постоянной переоценки системы и приложения. Видео, которые предлагает ваше стриминговое приложение, — живой пример: что вы смотрите, формирует то, что оно выучивает, а то, что оно выучивает, формирует то, что вы смотрите.

Опыт

На калькуляторе в радианах вычислите $\sin(3.14159)$ плюс $\sin(1.5708)$: получится около 0 плюс 1, то есть 1. Теперь вычислите $\sin(3.24159)$ плюс $\sin(1.62080)$: около минус 0,0998 плюс 0,9988, то есть 0,899. Ландшафт опустился примерно на 0,1, когда вы шагнули на 0,1 против градиента, ровно то, что обещает один шаг градиентного спуска.

Запомним

Предложенное упражнение реализует $f(x) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$ в NumPy, JAX и PyTorch:

векторизованно по батчу, с градиентом, написанным вручную, путями градиентного спуска из нескольких стартов и импульсом с убывающей нормой градиента. Обучающий набор данных — это множество из n пар вход-выход, используемое, когда написать функцию вручную невозможно, а примеры собрать легко. Пары предполагаются i.i.d. из неизвестного распределения: одинаково распределённые значит устойчивые во времени, независимо распределённые значит собранные без смещения, как показывают десятилетия фотографий машин и салон Tesla. За предположением нужно следить всю жизнь модели: сдвиг домена со временем ухудшает модели, а рекомендательные системы создают петли обратной связи, меняя то самое поведение, на котором учатся.

Вопросы

1. Классификатор машин обучен на фотографиях, собранных за последние сорок лет. Какая часть предположения i.i.d. нарушена и почему?
 - а. Одинаково распределённые, потому что модели машин менялись со временем, так что связь не устойчива

- b. Независимо распределённые, потому что фотографии сделаны в одном городе
 - c. Ни одна, потому что больше лет дают больше данных
- 2. Почему сбор фотографий машин рядом с салоном одной марки нарушил бы предположение i.i.d.?
 - a. Потому что фотографии были бы слишком похожи по освещению
 - b. Потому что сбор был бы смещён: слишком много машин одного вида и слишком мало других
 - c. Потому что салоны продают только новые машины
- 3. Почему рекомендательная система нуждается в постоянной переоценке?
 - a. Потому что её градиенты со временем становятся шумными
 - b. Потому что её набор данных всегда слишком мал
 - c. Потому что пользователи реагируют на её собственные подсказки, создавая петли обратной связи, которые меняют данные, на которых она учится
- 4. В практическом упражнении что должно происходить с длиной градиента по ходу градиентного спуска?
 - a. Она должна сжиматься к нулю по мере приближения метода к стационарной точке
 - b. Она должна расти, потому что импульс добавляется к каждому шагу
 - c. Она должна оставаться постоянной, если размер шага фиксирован

Ответы

1. **a)** Одинаково распределённые значит, что приближаемая вещь устойчива во времени; за десятилетия модели машин меняются, и обученная модель пропустит новые.
2. **b)** Независимо распределённые значит, что в сборе нет смещения и он представляет всё распределение; автосалон чрезмерно представляет одну марку.
3. **c)** Рекомендательная система меняет то, как пользователи взаимодействуют с приложением, так что данные сдвигаются в ответ на саму систему, что требует непрерывной переоценки.
4. **a)** В стационарной точке градиент равен нулю, так что по мере приближения оптимизатора к ней норма градиента сходится к нулю.

Разобранные примеры

Один шаг на учебном ландшафте Для $f(x) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$ в точке $(\pi, 0)$ вычислите f и градиент, затем сделайте один шаг градиентного спуска размером $0,1$.

$$f(\pi, 0) = \sin(\pi) \cdot 1 + \sin(\pi/2) \cdot 1 = 0 + 1 = 1.$$

$$\partial f / \partial x_1 = \cos(x_1)\cos(x_2) + 0.5\cos(0.5x_1)\cos(0.5x_2) = -1 + 0,5 \times 0 = -1.$$

$$\partial f / \partial x_2 = -\sin(x_1)\sin(x_2) - 0.5\sin(0.5x_1)\sin(0.5x_2) = 0 \text{ при } x_2 = 0.$$

Шаг: $x_1 = \pi - 0,1 \times (-1) \approx 3,2416$, x_2 остаётся 0; $f \approx -0,0998 + 0,9988 \approx 0,899$.

Ответ: f падает с 1 примерно до 0,899 после одного шага

Задачи

1. Запишите частную производную $f(x) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$ по x_2 и вычислите её в точке $(\pi/2, \pi/2)$.

Подсказка: Производная $\cos$ равна минус $\sin$, а цепное правило даёт множитель 0,5 для медленной волны.

Ответ: $\partial f / \partial x_2 = -\sin(x_1)\sin(x_2) - 0.5\sin(0.5x_1)\sin(0.5x_2)$. В точке $(\pi/2, \pi/2)$: $-1 \times 1 - 0.5 \times \sin(\pi/4)^2 = -1 - 0.5 \times 0.5 = -1.25$.

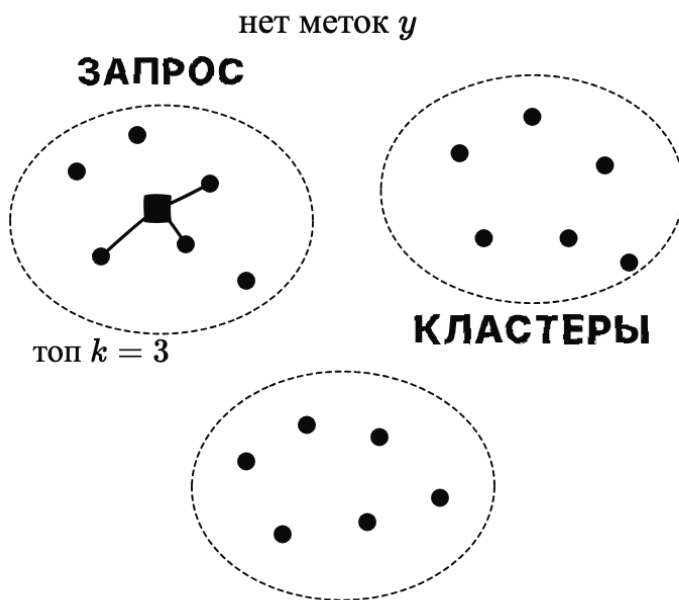
2. Команда обучает модель разблокировки по лицу на фотографиях из одной страны и выпускает её по всему миру. Объясните в терминах книги, что может пойти не так.

Подсказка: Вспомните набор данных о машинах, собранный в Италии и используемый в Токио.

Ответ: Данные могут быть пригодны там, где их собрали, но не в других местах: распределение при использовании отличается, так что предположение i.i.d. нарушается и качество может упасть для пользователей, не похожих на тех, что в обучающем наборе; это нужно проверять и перепроверять со временем.

Обучение без меток и повторное использование выученного

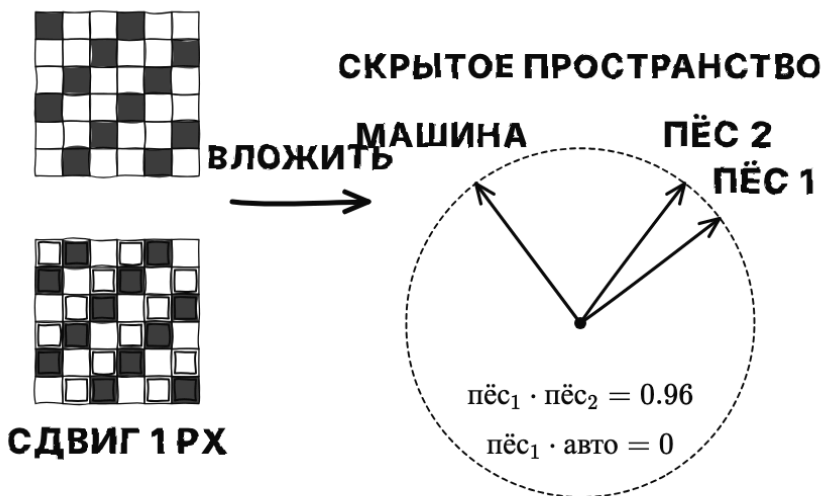
Откройте фотоприложение, выберите снимок своей собаки и найдите кнопку с надписью вроде «показать похожие». Оно вернёт другие фотографии собак, включая те, что никто никогда не подписывал, а фотографии вашей машины отодвинет далеко. Никто не говорил приложению, что на что похоже; это сделало что-то внутри обученной модели. Эта часть объясняет это «что-то» и многие способы повторно использовать обученную модель.



Группировка и поиск похожего

Большинство успешных приложений в той или иной форме используют обучение с учителем, но вариаций много. Когда в наборе данных вовсе нет целевых значений y_i , мы говорим об обучении без учителя. Классическое применение — кластеризация: группировка входов так, чтобы точки в одном кластере были похожи, а точки в разных кластерах — нет. Другое — поиск: отбор в большой базе данных k элементов, наиболее похожих на запрос пользователя.

Со сложными данными вроде изображений это трудно, потому что расстояния, вычисленные по сырым пикселям, мало что значат: даже крошечное изменение может поменять миллионы пикселей. Фотография с телефона из 12 миллионов пикселей, сдвинутая вбок на один пиксель, выглядит для нас одинаково, но почти каждое из её 36 миллионов цветовых чисел меняется.



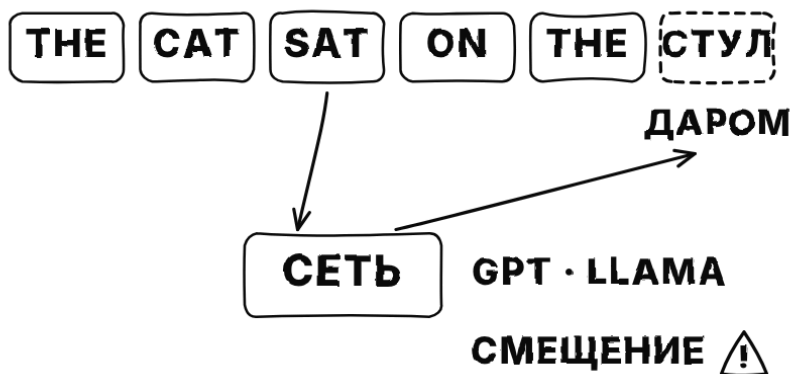
От пикселей к скрытому пространству

Теперь предположим, что у нас уже есть дифференцируемая модель, оптимизированная для какой-то другой, довольно общей задачи, например классификации изображений; это называется предобученной моделью. Как покажет книга, внутренние состояния такой модели можно читать как векторы в пространстве высокой размерности, и часто у них полезная геометрия: вещи, похожие по смыслу, отправляются, вкладываются, в близкие точки. Рисунок в книге называет это эмбедингом данных: переносом их из пространства без осмысленного расстояния, сырых изображений, в пространство, где оно есть, скрытое пространство.

Здесь возвращается скалярное произведение из предыдущих частей. Предположим, в крошечном двумерном скрытом пространстве одна фотография собаки попадает в $(0,8, 0,6)$, другая собака в $(0,6, 0,8)$, а машина в $(-0,6, 0,8)$, все длиной один. Косинусное сходство двух собак равно $0,48$ плюс $0,48$, то есть $0,96$, почти одно направление; первая собака и машина дают минус $0,48$ плюс $0,48$, то есть 0 , никакой связи. Вот как работает «показать похожие», и те же скрытые векторы можно подавать стандартным методам кластеризации, таким как смеси гауссиан.

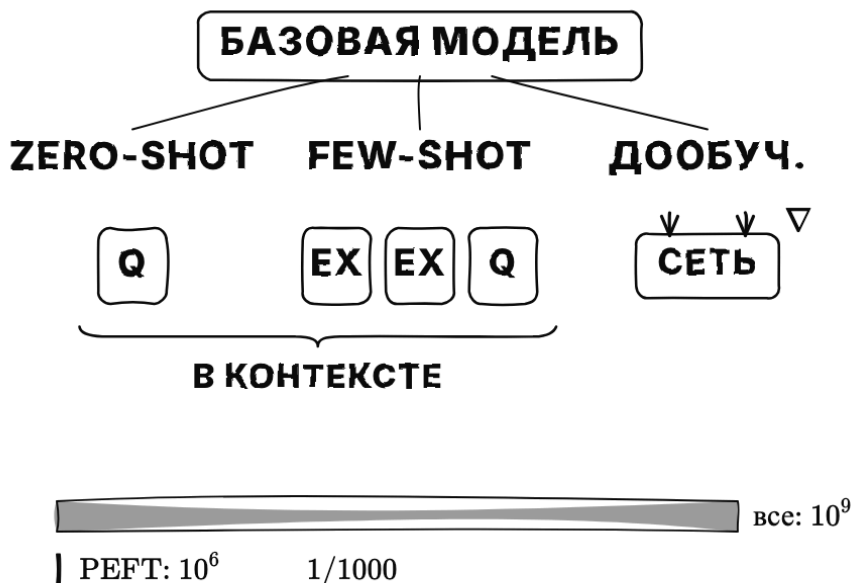
А если предобученной модели нет? Распространённый вариант обучения без учителя — самообучение, SSL. Его идея — найти задачу с учителем, спрятанную внутри неразмеченных данных, и использовать её для обучения модели, которая послужит многим последующим, нижележащим, задачам. С большим корпусом текста, например, мы всегда можем обучить программу предсказывать, как продолжается короткий фрагмент текста: следующие слова — бесплатные метки. Открытие, что нейронные сети, предобученные таким образом, дают ещё и эффективные эмбединги текста, глубоко повлияло на область. Книга добавляет предупреждение: большие веб-наборы данных полны смещений, ругательств и вульгарного содержимого, модели, обученные на них, впитывают эти смещения, и это серьёзная критика фундаментальных моделей с закрытым кодом.

САМООБУЧЕНИЕ



Метки, спрятанные в тексте

Большие языковые модели, как покажут главы восьмая и десятая, — современные версии этой идеи: такие модели, как GPT или Llama, всегда начинают с самообучения на предсказании следующего токена, маленького кусочка текста. Такие модели иногда называют фундаментальными моделями, и есть три способа их использовать, которые сравнивает рисунок в книге. Zero-shot: дать модели новую задачу напрямую, например вопрос, и использовать её ответ. Few-shot промптинг: сначала поместить в промпт несколько примеров новой задачи; оба способа работают только если модель очень хорошо обобщает. Дообучение: взять предобученную модель и оптимизировать её параметры градиентным спуском на маленьком наборе данных для новой задачи, почти как при обучении с нуля.



Три способа использовать модель

Когда вы просите чат-ассистента перевести одно предложение, это zero-shot; когда вы сначала показываете ему два переведённых примера в нужном стиле, это few-shot промптинг; а компания, которая дополнительно обучает его на собственных документах, его дообучает. Когда примеров в промпте достаточно и дообучение не нужно, книга говорит, что модель выполняет обучение в контексте; обучение по нескольким примерам можно делать и дообучением. Сама книга сосредоточена на построении моделей с нуля, замечая, что дообучение работает похожими средствами.

Дообучение сегодня особенно легко благодаря большим репозиториям моделей с открытым кодом. Оно может обновлять

все параметры исходной модели, или только малое подмножество, или небольшое число добавленных параметров; последние варианты называются параметрически эффективным дообучением, PEFT. Чтобы почувствовать экономию: обучение одного миллиона добавленных чисел поверх модели из миллиарда означает настройку в тысячу раз меньшего числа значений, чем при полном дообучении. Техники PEFT книга оставляет для следующего тома.

Опыт

Нарисуйте из центра листа три стрелки длиной один: одну вправо вверх для «собака один», одну чуть круче для «собака два» и одну влево вверх для «машина». Измерьте углы транспортиром: две собачьи стрелки близки, примерно в 16 градусах друг от друга, а стрелка машины под прямым углом к первой собаке. Малый угол означает большой косинус, то есть «похоже», а это всё, что нужно поисковой системе, чтобы ранжировать результаты.

Запомним

Без целевых значений у нас обучение без учителя, такое как кластеризация и поиск, но расстояния по сырым пикселям для изображений бессмысленны. Внутренние векторы предобученной модели вкладывают данные в скрытое пространство, где похожие вещи лежат близко, так что скалярные произведения и кластеризация там работают, как показали 0,96 для двух собак и 0 для собаки и машины. Самообучение изобретает метки из неразмеченных данных, например предсказывая следующие слова текста; так начинают языковые модели вроде GPT и Llama, и так же оно переносит смещения веб-данных. Фундаментальные модели можно использовать zero-shot, с few-shot промптами или дообучать градиентным спуском, полностью или параметрически эффективным дообучением малого набора параметров.

Вопросы

1. Почему трудно находить похожие изображения, измеряя расстояния напрямую по их пикселям?
 - a. Потому что пиксели нельзя хранить как числа
 - b. Потому что даже крошечное изменение, вроде сдвига на один пиксель, меняет миллионы значений пикселей
 - c. Потому что у изображений нет меток

2. Что делают внутренние векторы предобученной модели полезными для поиска и кластеризации?
 - a. Вещи, похожие по смыслу, вкладываются близко друг к другу в скрытом пространстве
 - b. Они всегда двумерны
 - c. Они содержат исходные пиксели в сжатом виде
3. Как самообучение получает метки из большого корпуса неразмеченного текста?
 - a. Просит людей разметить его малую часть
 - b. Сначала кластеризует текст
 - c. Обучает модель предсказывать, как продолжается фрагмент текста, так что следующие слова служат метками
4. Вы показываете чат-ассистенту два примера предпочитаемого стиля пересказа, а затем просите третий пересказ, не меняя его параметров. Как это называется?
 - a. Дообучение
 - b. Few-shot промптинг, пример обучения в контексте
 - c. Использование zero-shot

Ответы

1. **b)** Расстояния по сырым пикселям плохо определены: малые возмущения меняют миллионы пикселей, хотя картинка выглядит для нас одинаково.
2. **a)** Внутренние состояния модели часто имеют полезную геометрию, отправляя семантически похожие объекты в близкие точки, так что расстояния и скалярные произведения там становятся осмысленными.

3. c) Самообучение находит задачу с учителем внутри неразмеченных данных; с текстом всегда можно обучить модель предсказывать продолжение, и так начинают модели вроде GPT и Llama.

4. b) Несколько примеров в промпте — это few-shot промптинг; когда дообучение не нужно, модель выполняет обучение в контексте.

Разобранные примеры

Сходство в скрытом пространстве В двумерном скрытом пространстве собака один находится в $(0,8, 0,6)$, собака два в $(0,6, 0,8)$, а машина в $(-0,6, 0,8)$, все длиной один. Вычислите косинусное сходство собаки один с собакой два и с машиной.

Для единичных векторов косинусное сходство — это скалярное произведение.

Собака один с собакой два: $0,8 \times 0,6 + 0,6 \times 0,8 = 0,48 + 0,48 = 0,96$.

Собака один с машиной: $0,8 \times (-0,6) + 0,6 \times 0,8 = -0,48 + 0,48 = 0$.

Так что две собаки смотрят почти в одну сторону, а машина под прямым углом к собаке один.

Ответ: 0,96 для двух собак, 0 для собаки и машины

Задачи

1. Фотография имеет 4000 на 3000 пикселей по 3 значения цвета в каждом. Сколько чисел её описывают и почему это делает пиксельные расстояния плохой мерой сходства?

Подсказка: Перемножьте три размера; подумайте о сдвиге на один пиксель.

Ответ: $4000 \times 3000 \times 3 = 36\,000\,000$ чисел. Сдвиг фотографии на один пиксель меняет почти все из них, так что две одинаковые на вид фотографии могут быть очень далеки по пиксельному расстоянию.

2. У базовой модели 1 миллиард параметров. Полное дообучение обновляет все, а параметрически эффективный метод обучает 2 миллиона добавленных параметров. Какую долю от числа параметров обучает параметрически эффективный метод?

Подсказка: Разделите одно число на другое.

Ответ: $2\,000\,000 / 1\,000\,000\,000 = 0,002$, то есть две тысячных размера базовой модели.

3. Определите каждое использование языковой модели как zero-shot, few-shot промптинг или дообучение: (a) попросить её напрямую перевести предложение; (b) обучить её дополнительно на обращениях в поддержку компании; (c) показать ей три решённые загадки перед тем, как задать четвёртую.

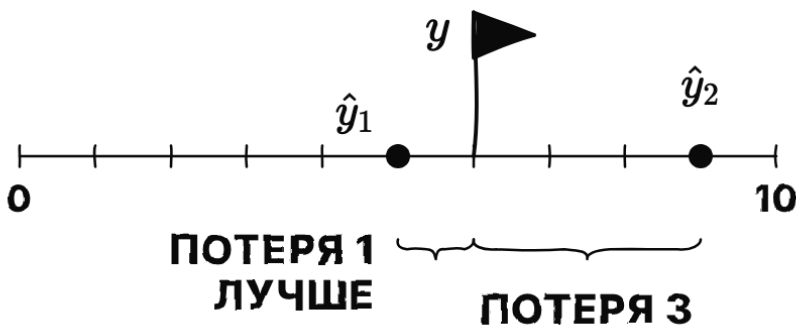
Подсказка: Спросите, есть ли примеры в промпте и меняются ли параметры.

Ответ: (a) zero-shot; (b) дообучение, поскольку её параметры оптимизируются градиентным спуском; (c) few-shot промптинг, поскольку примеры в промпте, а параметры не меняются.

Оценка ошибок: функция потерь

Сыграйте с другом в угадайку: он загадывает число, а вы угадываете. После каждой попытки он говорит, насколько вы промахнулись, и вы стараетесь сделать это число маленьким. Без этого одного числа, «насколько промахнулись», у вас не было бы способа улучшаться; с ним улучшение становится делом арифметики. Функция потерь — это такое число, а обучение модели — эта игра, сыгранная миллионы раз.

$$l(y, \hat{y})$$



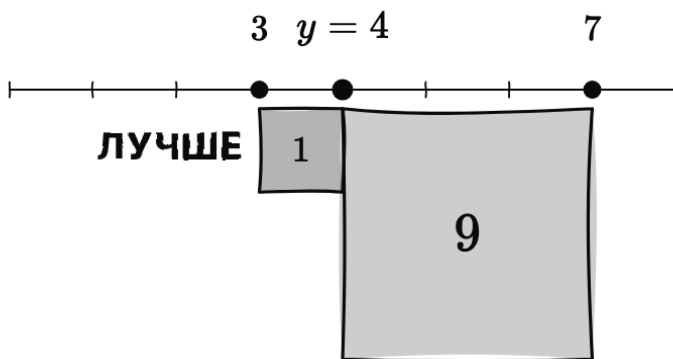
$$l(y, \hat{y}_1) < l(y, \hat{y}_2)$$

Насколько промахнулась догадка

Сначала книга дополняет свой список сценариев обучения несколькими указателями. Когда размечена лишь часть набора данных, это обучение с частичным привлечением учителя, примеры будут в двенадцатой главе. Когда есть несколько наборов данных из похожих распределений или одно распределение, меняющееся со временем, возникают бесчисленные задачи в зависимости от порядка, в котором приходят задания или данные: адаптация домена, метаобучение, непрерывное обучение, метрическое обучение, разобучение и другие, некоторые из которых разбирает следующий том.

Раздел 3.2 переходит к функциям потерь, которые делают идею «приближения» поведения точной. Определение: при целевом значении y и предсказании модели $\hat{y} = f(x)$ потеря $l(y, \hat{y})$ — это одно вещественное число, дифференцируемое, значение которого отслеживает, насколько хорошо работает модель. Конкретно, $l(y, \hat{y}_1) < l(y, \hat{y}_2)$ означает, что предсказание один лучше предсказания два для этого целевого значения.

Распространённый пример — квадратичная ошибка, разность в квадрате. Если настоящая цена дома равна 4, а две модели предсказывают 3 и 7, потери равны 1 в квадрате, то есть 1, и 3 в квадрате, то есть 9; меньшая потеря, 1, отмечает лучшее предсказание. Функция потерь превращает наше понимание задачи и наши предпочтения среди решений в одно вещественное число, с которым может работать оптимизатор. А поскольку она дифференцируема, она превращает обучение в задачу математической оптимизации, которую может решить градиентный спуск, минимизируя среднюю потерю по набору данных.



$$f^* = \arg \min_f \frac{1}{n} \sum_{i=1}^n l(y_i, f(x_i)) \quad \text{ERM}$$

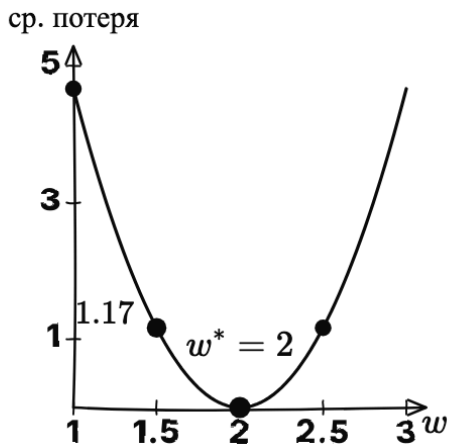
Квадратичная ошибка, затем среднее

Итак, при наборе данных $\mathcal{S}_n$ из пар и потере l разумная цель — модель с наименьшей средней потерей: $f^* = \arg \min_f (1/n) \sum_{i=1}^n l(y_i, f(x_i))$. Словами: испытайте модель на каждом примере, оцените каждое предсказание, усредните оценки и выберите модель с наименьшим средним. По историческим причинам это называется минимизацией эмпирического риска, ERM, где «риск» — просто другое слово для потери.

Давайте вычислим эмпирический риск. У трёх примеров целевые значения 2, 4 и 6, а модель предсказывает 3, 4 и 4. Квадратичные потери равны 1, 0 и 4; их среднее равно 5, делённым на 3, примерно 1,67.

y	$\hat{y}$	потеря
2	3	1
4	4	0
6	4	4

среднее = $5/3 \approx 1.67$



$$w^* = \arg \min_w \frac{1}{n} \sum_{i=1}^n l(y_i, f(x_i, w))$$

Средняя потеря и её впадина

Эта формула ищет среди всех возможных функций от x . На практике, как покажет книга, модель описывается набором тензоров w , её параметрами, и мы пишем $f(x, w)$. Тогда обучение ищет лучшие параметры численной оптимизацией, минимизируя эмпирический риск градиентным спуском: $w^* = \operatorname{argmin}_w (1/n) \sum_{i=1}^n l(y_i, f(x_i, w))$. Единственное изменение в том, что поиск теперь идёт по тензору параметров w .

Сделаем это конкретным на простейшей модели, $f(x, w) = wx$, на входах 1, 2, 3 с целевыми значениями 2, 4, 6. При w , равном 1,5, предсказания равны 1,5, 3 и 4,5, квадратичные потери 0,25, 1 и 2,25, а среднее примерно 1,17. При w , равном 2, каждое предсказание точно, и средняя потеря равна 0, так что здесь w со

звёздочкой равно 2. Фотоприложение в вашем телефоне обучали так же, только с миллионами фотографий, миллионами параметров и функцией потерь, подходящей для меток, а не для цен.

Опыт

Возьмите три пары (1, 2), (2, 4), (3, 6) и попробуйте несколько значений w в модели w на x : 1, 1,5, 2, 2,5. Для каждого вычислите три квадратичные ошибки и их среднее. Вы должны увидеть, как среднее падает к 0 при w , равном 2, и снова растёт за ним — маленькая впадина, дно которой и есть обученный параметр.

Запомним

За пределами простого обучения с учителем лежат обучение с частичным привлечением учителя и многие постановки с несколькими наборами данных, такие как адаптация домена, непрерывное обучение и разобучение. Потеря $l(y, \hat{y})$ — это дифференцируемое вещественное число, которое меньше для лучших предсказаний, как квадратичная ошибка 1 против 9. Минимизация эмпирического риска выбирает модель с наименьшей средней потерей по набору данных, примерно 1,67 для предсказаний 3, 4, 4 против целевых значений 2, 4, 6. При параметризованной модели $f(x, w)$ обучение ищет $w^* = \operatorname{argmin}_w$ по w средней потери с помощью градиентного спуска; для wx на наших трёх парах w^* равно 2.

Вопросы

1. Почему функция потерь должна быть дифференцируемой?
 - a. Чтобы её можно было напечатать как вещественное число
 - b. Чтобы обучение стало задачей оптимизации, которую может решить градиентный спуск
 - c. Чтобы она всегда была положительной
2. Что выбирает минимизация эмпирического риска?
 - a. Модель с наименьшей потерей на одном выбранном примере
 - b. Модель с наибольшим числом параметров
 - c. Модель с наименьшей средней потерей по всему набору данных
3. При целевом значении 4 предсказание A равно 3, а предсказание B равно 7. При квадратичной ошибке какое лучше и почему?
 - a. A, потому что его потеря равна 1 против 9 у B
 - b. B, потому что оно больше целевого значения
 - c. Они одинаково хороши, поскольку оба неверны
4. Что меняется при переходе от минимизации по всем функциям f к минимизации по параметрам w ?
 - a. Функция потерь больше не нужна
 - b. Поиск становится численной оптимизацией по тензорам параметров фиксированного семейства моделей

с. Набор данных заменяется одним примером

Ответы

1. **б)** Будучи дифференцируемой, потеря превращает обучение в задачу математической оптимизации, решаемую градиентным спуском, минимизацией средней потери на наборе данных.
2. **с)** ERM оценивает каждое предсказание на наборе данных, усредняет потери и выбирает модель или параметры с наименьшим средним; риск — синоним потери.
3. **а)** Меньшая потеря означает лучшее предсказание: $(4 - 3)^2 = 1$ меньше, чем $(4 - 7)^2 = 9$.
4. **б)** Модели описываются тензорами w , так что обучение ищет $w^* = \arg\min$ по w средней потери численной оптимизацией, такой как градиентный спуск.

Разобранные примеры

Эмпирический риск простой модели Модель $f(x, w) = wx$ испытывают на парах $(1, 2)$, $(2, 4)$, $(3, 6)$ с квадратичной потерей. Вычислите эмпирический риск при $w = 1,5$ и при $w = 2$.

$w = 1,5$: предсказания 1,5, 3, 4,5; квадратичные ошибки 0,25, 1, 2,25.

Среднее: $(0,25 + 1 + 2,25) / 3 = 3,5 / 3 \approx 1,17$.

$w = 2$: предсказания 2, 4, 6, все точные; квадратичные ошибки 0, 0, 0; среднее 0.

Так что $w = 2$ минимизирует эмпирический риск на этом наборе данных.

Ответ: Примерно 1,17 при $w = 1,5$ и 0 при $w = 2$, которое и есть w^*

Задачи

1. Целевые значения 1, 3 и 5; модель предсказывает 2, 3 и 3. Вычислите эмпирический риск с квадратичной потерей.

Подсказка: Возведите каждую разность в квадрат, сложите, разделите на 3.

Ответ: Квадратичные ошибки 1, 0 и 4; сумма 5; эмпирический риск $5/3 \approx 1,67$.

2. Для $f(x, w) = wx$ на парах (1, 2), (2, 4), (3, 6) вычислите эмпирический риск при $w = 2,5$ и сравните с $w = 1,5$.

Подсказка: Предсказания 2,5, 5, 7,5.

Ответ: Квадратичные ошибки 0,25, 1 и 2,25, среднее $\approx 1,17$, то же, что при $w = 1,5$: риск растёт симметрично по обе стороны от минимума при $w = 2$.

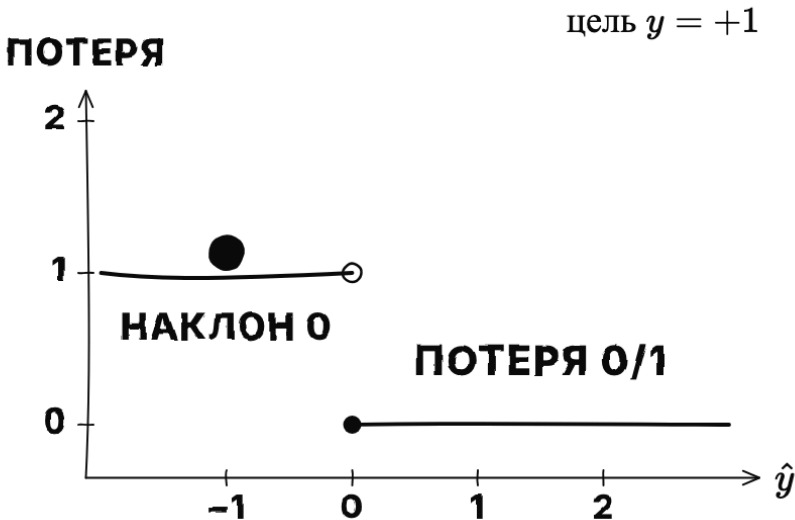
3. Спам-фильтр обучают на наборе данных, где размечено лишь 5% писем. Какая это постановка обучения, и что книга говорит о сценариях, где данные приходят со временем из меняющихся распределений?

Подсказка: Подумайте о частичных метках и о списке задач с несколькими наборами данных.

Ответ: Частично размеченные данные — это обучение с частичным привлечением учителя. Данные из похожих или меняющихся распределений, приходящие со временем, порождают такие задачи, как адаптация домена и непрерывное обучение, некоторые из которых разбирает следующий том.

Потери, за которыми можно идти, и модели, которые только запоминают

Сыграйте в угадайку снова, но теперь ваш друг говорит только «верно» или «неверно». Вы называете 30 и слышите «неверно»; называете 31 и снова слышите «неверно». Вы ничего не узнаете о том, куда двигаться, потому что ответ не меняется, когда вы чуть сдвигаете догадку. Функция потерь с таким изъяном не может вести градиентный спуск, и эта часть показывает, как этого избежать, а затем переходит к более глубокой ловушке: модели, которая получает идеальный балл, просто запоминая.



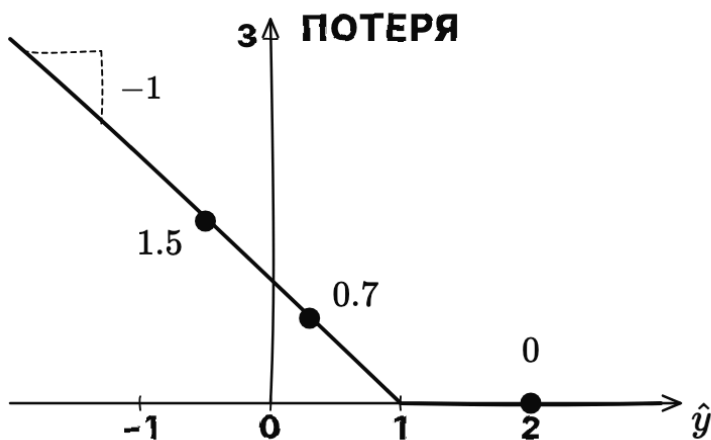
Потеря «верно или неверно» плоская

Первое наблюдение: дифференцируемость функции потерь существенна. Возьмём задачу бинарной классификации, где целевое значение $y = \pm 1$ принимает только два значения, и модель $f(x)$, выдающую любое вещественное число. Решение модели можно читать как знак её выхода, $\text{sign}(f(x))$, и определить потерю 0/1: $l(y, \hat{y}) = 0$, если $\text{sign}(\hat{y}) = y$, и 1 иначе. Она соответствует нашей интуиции о правоте, но бесполезна для обучения. Её градиент равен нулю почти везде, кроме ровно того места, где знак переключается, так что градиентный спуск никогда не сдвинется со стартовой точки: верно или неверно, и ничего между.

Менее интуитивная, но более полезная величина — отступ, $u\hat{y}$. Он положителен, когда знак модели согласуется с целевым значением, и отрицателен, когда нет, и, в отличие от потери 0/1, он меняется гладко. Одна из потерь, построенных на нём, — кусочно-линейная потеря, $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$, используемая для обучения метода опорных векторов.

отступ $y\hat{y}$

$$\max(0, 1 - y\hat{y})$$



У кусочно-линейной потери есть наклон

Вычислим её для целевого значения плюс 1. Предсказание 0,3 верно по знаку, так что потеря 0/1 равна 0, но кусочно-линейная потеря равна 1 минус 0,3, то есть 0,7: верно, но неуверенно. Предсказание 2 даёт отступ 2 и кусочно-линейную потерю 0. Предсказание минус 0,5 даёт отступ минус 0,5 и кусочно-линейную потерю 1,5, большую, потому что оно неверно. А ниже отступа 1 кусочно-линейная потеря падает на единицу с каждой единицей роста предсказания: наклон, за которым градиентный спуск может следовать. Книга выводит урок: есть встроенное напряжение между потерями, выражающими нашу идею хорошей работы, и потерями, которые хорошо работают для численной оптимизации.

Блок отмечает историю: минимизация эмпирического и ожидаемого риска связана с работами российского учёного Владимира Вапника, основавшего статистическую теорию обучения. Эта теория изучает, насколько хорошо эмпирический риск приближает ожидаемый по конечному числу выборок, для ограниченных семейств функций и мер их сложности. Современные нейронные сети, которые продолжают хорошо обобщать долго после того, как ожидалось бы переобучение, открыли в этой теории много новых исследовательских вопросов.

Второе наблюдение: эмпирический риск всегда тривиально минимизировать. Определим модель, которая ищет свой вход в обучающем наборе: если пара там есть, вернуть её u ; иначе вернуть фиксированное значение по умолчанию, например 0. Эта таблица поиска делает потерю на обучении настолько низкой, насколько возможно, но она бесполезна, потому что ничего не знает ни об одном входе, которого не видела. В этом разница между запоминанием и обучением. Мы оптимизируем среднюю потерю на обучающих данных, но наша настоящая цель — низкая потеря на будущих входах, ещё не виденных; обучающий набор лишь их заместитель.

ТАБЛИЦА

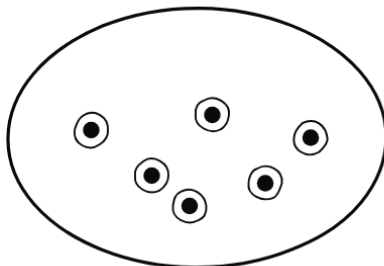
$1 \rightarrow 2$
$2 \rightarrow 4$
$3 \rightarrow 6$

ПОТЕРЯ 0

$5 \rightarrow 0$ ✗

ЗАПОМИНАНИЕ

$$ER[f] = \mathbb{E}_{p(x,y)}[l(y, f(x))]$$



ВСЕ ПАРЫ

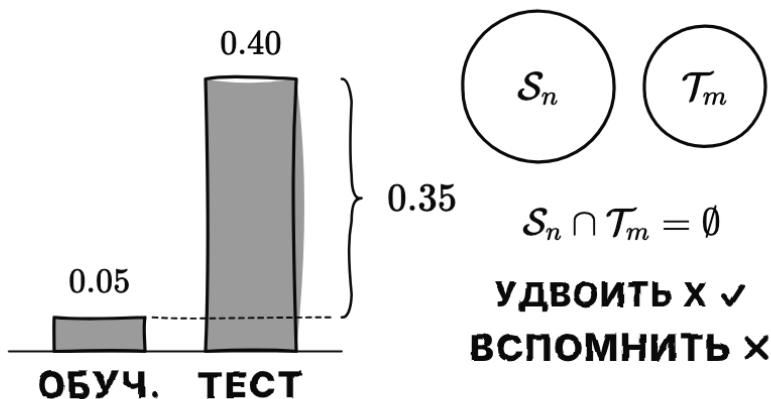
ЭМПИРИЧЕСКИЙ $\approx$

Запоминание против ожидаемого риска

Книга формализует это как ожидаемый риск: $ER[f] = \mathbb{E}_{p(x,y)}[l(y, f(x))]$. Словами: средняя потеря по всем парам вход-выход, которые модель могла бы когда-либо встретить, взвешенным по их вероятности, например по всем письмам, которые когда-либо мог бы получить спам-фильтр. Модель с низким ожидаемым риском гарантированно работала бы, но вычислить его невозможно, поскольку мы никогда не сможем перечислить и разметить все возможные входы. Эмпирический риск оценивает его по имеющемуся набору данных, как приближение Монте-Карло: среднее по случайным выборкам, замещающее среднее по всему.

Разница между ожидаемым и эмпирическим риском — это разрыв обобщения. У таблицы поиска огромный разрыв: она переобучается на своих обучающих данных. На практике мы проверяем обобщение отдельным тестовым набором $\mathcal{T}_m$ из m точек, никогда не использованных в обучении, не имеющим ничего общего с обучающим набором: $\mathcal{S}_n \cap \mathcal{T}_m = \emptyset$. Если классификатор фотографий достигает средней потери 0,05 на обучающих фотографиях, но 0,40 на отложенных тестовых, разрыв 0,35 — мера того, насколько он переобучен.

РАЗРЫВ



Разрыв обобщения

Опыт

Запишите пять пар, например (1, 2), (2, 4), (3, 6), (4, 8), (5, 10), и спрячьте две из них. Постройте «модель», которая просто вспоминает три оставленные, и другую, которая говорит «удвой вход». Проверьте обе на двух спрятанных парах: модель-воспоминание не имеет ответа и откатывается к 0, модель-удвоение угадывает обе; это запоминание против обучения, измеренное на тестовом наборе.

Запомним

Функция потерь должна быть дифференцируемой: потеря 0/1 соответствует правоте, но имеет нулевой градиент почти везде, так что градиентный спуск не может двигаться. Отступ $y\hat{u}$ меняется гладко, а кусочно-линейная потеря $\max(0, 1 - y\hat{u})$ даёт 0,7 для верного, но робкого 0,3, 0 для уверенного 2 и 1,5 для неверного -0,5. Эмпирический риск можно довести до минимума таблицей поиска, которая только запоминает; что нам действительно нужно — низкий ожидаемый риск, средняя потеря по всем возможным входам, которую можно лишь оценить. Разрыв между ожидаемым и эмпирическим риском измеряет переобучение, а отдельный тестовый набор, никогда не использованный в обучении, оценивает его на практике.

Вопросы

1. Почему потеря 0/1 бесполезна для обучения градиентным спуском?
 - a. Потому что она даёт неверный ответ для верных предсказаний
 - b. Потому что её градиент равен нулю почти везде, так что градиентный спуск никогда не сдвигается
 - c. Потому что её можно использовать только с методом опорных векторов
2. Для целевого значения +1 какое предсказание получает кусочно-линейную потерю 0,7?
 - a. 0,3
 - b. 2
 - c. -0,5
3. Таблица поиска, возвращающая сохранённый ответ для каждого обучающего входа, достигает наименьшего эмпирического риска. Почему она всё же бесполезна?
 - a. Потому что поиск слишком медленный
 - b. Потому что её потеря недифференцируема
 - c. Потому что она только запоминает и не может сказать ничего полезного о входах, которых не видела
4. Как переобучение измеряют на практике?
 - a. Сравнивая потерю на обучающем наборе с потерей на отдельном тестовом наборе, никогда не использованном

в обучении

- б. Вычисляя ожидаемый риск по всем возможным входам
- с. Подсчитывая параметры модели

Ответы

1. б) Потеря $0/1$ меняется только там, где переключается знак предсказания; везде остальном сдвиг предсказания ничего не меняет, так что градиент равен нулю, и оптимизатор остаётся на старте.
2. а) Кусочно-линейная потеря равна $\max(0, 1 - y\hat{y})$: для $0,3$ это $1 - 0,3 = 0,7$, для 2 это 0 , а для $-0,5$ это $1,5$.
3. с) Наша настоящая цель — низкая потеря на будущих, невиданных входах, ожидаемый риск; таблица поиска откатывается к значению по умолчанию и переобучается, с огромным разрывом обобщения.
4. а) Ожидаемый риск вычислить нельзя, так что его оценивает отложенный тестовый набор; разница между потерей на обучении и на тесте приближает разрыв обобщения.

Разобранные примеры

Кусочно-линейная потеря против потери $0/1$ Для целевого значения $y = +1$ вычислите отступ, потерю $0/1$ и кусочно-линейную потерю $\max(0, 1 - y\hat{y})$ для предсказаний $\hat{y} = 0,3, 2$ и $-0,5$.

$\hat{y} = 0,3$: отступ $0,3$; знак совпадает, так что потеря $0/1$ равна 0 ; кусочно-линейная $1 - 0,3 = 0,7$.

$\hat{y} = 2$: отступ 2 ; потеря $0/1$ равна 0 ; кусочно-линейная $\max(0, -1) = 0$.

$\hat{y} = -0,5$: отступ $-0,5$; знак не совпадает, так что потеря $0/1$ равна 1 ; кусочно-линейная $1 + 0,5 = 1,5$.

Потеря $0/1$ не отличает $0,3$ от 2 , а кусочно-линейная потеря всё ещё подталкивает $0,3$ к уверенному отступу.

Ответ: Кусочно-линейные потери $0,7$, 0 и $1,5$; потери $0/1$ равны 0 , 0 и 1

Задачи

1. Для целевого значения $y = -1$ вычислите кусочно-линейную потерю для предсказаний $\hat{y} = -3, -0,2$ и $0,4$.

Подсказка: Сначала вычислите отступ $y\hat{y}$.

Ответ: Отступы 3 , $0,2$ и $-0,4$; кусочно-линейные потери $\max(0, -2) = 0$, $0,8$ и $1,4$.

2. У одной модели средняя потеря $0,12$ на обучающем наборе и $0,15$ на тестовом; у другой $0,01$ на обучении и $0,45$ на тесте. Какая переобучена сильнее и какова оценка разрыва обобщения у каждой?

Подсказка: Вычтите потерю на обучении из потери на тесте.

Ответ: Разрывы: $0,03$ и $0,44$. Вторая модель переобучена гораздо сильнее: она почти запоминает обучающие данные, но плохо работает на невиданных.

3. Объясните своими словами, почему эмпирический риск называют приближением Монте-Карло ожидаемого риска.

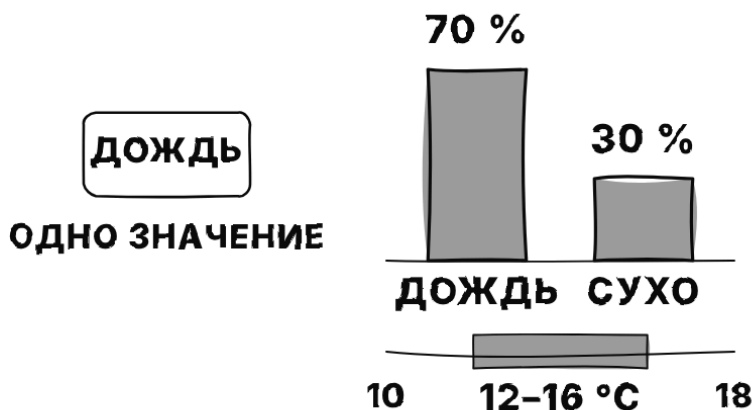
Подсказка: Подумайте, по чему усредняет ожидаемый риск и что даёт набор данных.

Ответ: Ожидаемый риск усредняет потерю по всем возможным парам вход-выход, взвешенным по распределению, что вычислить нельзя. Набор данных — случайная выборка из этого распределения, и усреднение потери по нему оценивает полное среднее, так же как методы Монте-Карло оценивают средние по случайным выборкам.

Предсказывать распределение вместо числа

Спросите погодное приложение о завтрашнем дне, и оно редко скажет просто «дождь». Оно скажет «70 процентов вероятности дождя» или «от 12 до 16 градусов». Это более богатый вид ответа: не одно значение, а насколько вероятно каждое. Книга теперь показывает, что модель можно построить так, чтобы она отвечала именно так, и что это подсказывает, какую функцию потерь использовать.

НАСКОЛЬКО ВЕРОЯТНО КАЖДОЕ



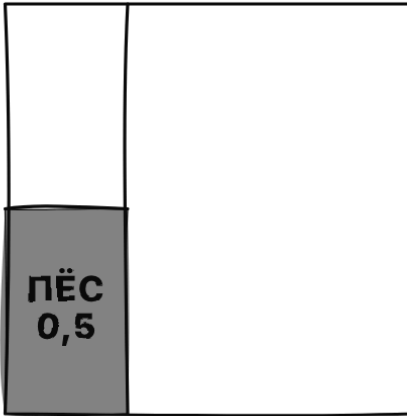
ВЕРОЯТНОСТНО

Одно значение или насколько вероятно каждое

Книга предлагает, что сейчас хороший момент изучить или пробежать её приложение о вероятности, особенно распределения, достаточные статистики и оценку максимального правдоподобия. Функция потерь, говорит она, кодирует наши апприорные знания о задаче и сильно влияет на качество. Иногда для её разработки достаточно простого рассуждения о задаче, как с кусочно-линейной потерей. Но есть более принципиальный путь: переформулировать всё обучение в чисто вероятностных терминах, точка зрения, которую многие книги предпочитают и которая в некоторых постановках может быть более интуитивной или более полезной.

Ключевое наблюдение начинается с предположения, что примеры приходят из распределения $p(x, y)$. По правилу произведения вероятностей $p(x, y) = p(x) p(y | x)$. Словами: шанс увидеть конкретный вход и выход вместе — это шанс входа, умноженный на шанс этого выхода, когда вход известен. Если 30 процентов фотографий сняты на улице, а половина уличных фотографий показывает собаку, то шанс уличной фотографии с собакой равен 0,3 на 0,5, то есть 0,15.

ВСЕ ФОТО



УЛИЦА 0,3

$$p(x, y) = p(x) p(y | x) \\ = p(x | y) p(y)$$

$$0.3 \times 0.5 = 0.15$$

$p(y | x)$: дискриминативный

$p(x)$: генеративный

Правило произведения как площади

Книга добавляет, что произведение можно разложить и в другую сторону, $p(x, y) = p(x | y) p(y)$. Методы, оценивающие $p(x)$ или $p(x | y)$, называются генеративными, а оценивающие $p(y | x)$ — дискриминативными; за исключением языкового моделирования, эта книга сосредоточена на дискриминативном виде и оставляет генеративное моделирование следующему тому.

Приближать $p(y | x)$ одной функцией $f(x)$, выдающей одно предсказание, имеет смысл, только если для каждого входа почти вся вероятность сидит на одном значении y . Такое распределение близко к тому, что математики называют дельтой Дирака, пиком в одной точке, и это предположение чрезвычайно упрощает задачу.

Ослабление: пусть выходы модели — не само предсказание, а достаточные статистики, параметры, условного распределения $p(y | f(x))$ над возможными выходами. Для задачи классификации с тремя классами, пронумерованными 1, 2 и 3, у модели может быть три выхода, задающих категориальное распределение: $p(y | f(x)) = \prod_{i=1}^3 f_i(x)^{y_i}$. Здесь $y \sim \text{Binary}(3)$ — это one-hot кодировка класса: все нули, кроме единицы в позиции класса, а $f(x) \sim \Delta(3)$ перечисляет предсказанную вероятность каждого класса.

Прочитаем формулу на числах. Предположим, фотоприложение предсказывает (0,7, 0,2, 0,1) для собаки, кошки и машины, а истинный класс — собака, one-hot (1, 0, 0). Произведение равно 0,7 в степени 1 на 0,2 в степени 0 на 0,1 в степени 0, а поскольку что угодно в степени 0 равно 1, результат — просто 0,7: вероятность, которую модель дала правильному классу. Если бы фотография на самом деле была машиной, one-hot (0, 0, 1), та же формула выбрала бы 0,1. Показатели степени работают как переключатель, который выбирает вероятность истинного класса.

Для регрессии, предсказания вещественного числа $y \in \mathbb{R}$, модель может выдавать два значения, $f(x) \sim (2)$, среднее и разброс гауссианы, колоколообразной кривой: $p(y | f(x)) = \mathcal{N}(y | f_1(x), f_2^2(x))$. Второй выход возводится в квадрат, чтобы дисперсия всегда была положительной. Если модель цены дома выдаёт 250 и минус 20, прогноз — колоколообразная кривая с центром в 250 и дисперсией 400, типичным разбросом 20; минус исчезает при возведении в квадрат.

$$p(y | f(x)) = \mathcal{N}(y | f_1(x), f_2^2(x))$$



Колоколообразная кривая для цены

Эта постановка очень общая и включает всё сказанное до сих пор. Она также даёт разработчику больше свободы, потому что выбрать вид $p(y | x)$ может быть проще, чем изобретать потерю $l(y, \hat{y})$ напрямую. И она естественно моделирует неопределённость: дисперсия гауссианы говорит, насколько модель уверена, так же как процент в погодном приложении.

Опыт

Напишите три вероятности, дающие в сумме единицу, скажем 0,6, 0,3 и 0,1, для трёх возможных ответов на вопрос викторины. Выберите верный ответ и запишите его one-hot вектор, например (0, 1, 0). Возведите каждую вероятность в соответствующую степень 0 или 1 и перемножьте: получите 0,3, вероятность, которую вы назначили истине, и это число следующие части будут стараться сделать большим.

Запомним

Функция потерь несёт наши априорные знания о задаче; вместо того чтобы проектировать её вручную, мы можем переформулировать обучение в вероятностных терминах. По правилу произведения $p(x, y) = p(x) p(y | x)$; оценка $p(y | x)$ дискриминативна, оценка $p(x)$ или $p(x | y)$ генеративна, и книга сосредоточена на дискриминативном виде. Модель может выдавать параметры распределения: три вероятности классов для категориального, где показатели one-hot выбирают вероятность истинного класса, 0,7 для собаки, предсказанной с 0,7. Для регрессии она может выдавать среднее гауссианы и квадрат разброса, что сохраняет дисперсию положительной и выражает неопределённость модели.

Вопросы

1. Классификатор выдаёт вероятности (0,7, 0,2, 0,1), а истинный класс — третий. Что даёт категориальная формула $\prod_i f_i(x)^{y_i}$?
 - a. 0,7
 - b. 0,1
 - c. 1,0
2. В гауссовой модели регрессии почему второй выход возводится в квадрат?
 - a. Чтобы сделать среднее больше
 - b. Чтобы предсказанная дисперсия всегда была положительной
 - c. Чтобы превратить регрессию в классификацию
3. Когда имеет смысл предсказывать одно значение $f(x)$, а не целое распределение по y ?
 - a. Когда для каждого входа почти вся вероятность сидит на одном значении y , как у дельты Дирака
 - b. Когда набор данных очень велик
 - c. Когда в задаче больше двух классов
4. Какой вид методов оценивает $p(y | x)$?
 - a. Генеративный
 - b. Без учителя
 - c. Дискриминативный

Ответы

1. **b)** One-hot вектор $(0, 0, 1)$ ставит показатель 1 только на третью вероятность, а что угодно в степени 0 равно 1, так что произведение выбирает 0,1, вероятность, данную истинному классу.
2. **b)** Дисперсия не может быть отрицательной; возведение второго выхода в квадрат, $f_2^2(x)$, сохраняет её положительной, что бы ни выдала модель, так что -20 даёт 400.
3. **a)** Одно предсказание — хорошее резюме, только если $p(y | x)$ сосредоточено вокруг одной точки; иначе моделирование распределения улавливает неопределённость.
4. **c)** Методы, оценивающие $p(x)$ или $p(x | y)$, генеративны, оценивающие $p(y | x)$ — дискриминативны; за исключением языкового моделирования, книга сосредоточена на последних.

Разобранные примеры

Категориальная формула как переключатель Модель предсказывает $f(x) = (0,7, 0,2, 0,1)$ для собаки, кошки, машины. Вычислите $p(y | f(x)) = \prod_i f_i(x)^{y_i}$, когда истинный класс — собака, и когда это машина.

У собаки one-hot $(1, 0, 0)$: $0,7^1 \times 0,2^0 \times 0,1^0$.

Любое число в степени 0 равно 1, так что произведение равно 0,7.

У машины one-hot $(0, 0, 1)$: $0,7^0 \times 0,2^0 \times 0,1^1 = 0,1$.

Ответ: 0,7 для собаки, 0,1 для машины: вероятность, которую модель дала истинному классу

Правило произведения на числах 30% фотографий сняты на улице, и половина уличных фотографий показывает собаку. Какова вероятность уличной фотографии с собакой?

$p(x) = 0,3$ для уличной фотографии.

$p(y | x) = 0,5$ для собаки при уличной фотографии.

Правило произведения: $p(x, y) = p(x) p(y | x) = 0,3 \times 0,5$.

Ответ: 0,15

Задачи

1. Модель с четырьмя классами предсказывает (0,1, 0,4, 0,4, 0,1); истинный класс — второй. Запишите его one-hot вектор и вычислите $\sum_i f_i(x) \wedge y_i$.

Подсказка: Только показатель при истинном классе равен 1.

Ответ: One-hot (0, 1, 0, 0); произведение равно 0,4.

2. Модель регрессии для температур выдаёт $f(x) = (14, -3)$. Опишите предсказанное распределение.

Подсказка: Первый выход — среднее; второй возводится в квадрат, чтобы дать дисперсию.

Ответ: Гауссиана с центром в 14 и дисперсией $(-3)^2 = 9$, то есть типичным разбросом 3 градуса.

3. Объясните, почему предсказание распределения может быть полезнее предсказания одного числа, на примере из повседневной жизни.

Подсказка: Подумайте, что прогноз погоды говорит вам помимо самого вероятного исхода.

Ответ: Распределение также говорит, насколько предсказание уверено: «70% вероятности дождя» или диапазон температур позволяют решить, брать ли зонт, а одно слово «дождь» скрывает неопределённость. Дисперсия гауссианы играет эту роль в модели регрессии из книги.

Сделать данные вероятными и сохранить каждую хорошую модель

Представьте двух синоптиков, которые оба давали прогноз на прошлую неделю. Для каждого дня вы смотрите, какой шанс каждый из них дал той погоде, что на самом деле случилась, и перемножаете эти шансы. Синоптик с большим произведением сделал прошлую неделю наименее удивительной, и ему вы доверяли бы больше. Это произведение — правдоподобие, и выбор модели, которая делает его наибольшим, — это способ обучения вероятностных моделей.

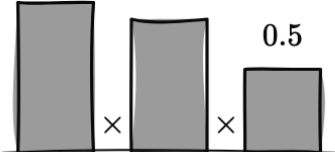
$$p(\mathcal{S}_n \mid f) = \prod_{i=1}^n p(y_i \mid f(x_i))$$

СЛУЧАЙНАЯ

1/3 каждому


$$\square \times \square \times \square = 0.037$$

ОБУЧЕННАЯ



0.9 0.8 0.5

$$\square \times \square \times \square = 0.36 \approx 10 \times$$

Правдоподобие: перемножьте шансы

Вспомните предположение, что набор данных $\mathcal{S}_n$ — это множество независимых, одинаково распределённых выборок. Поскольку выборки независимы, вероятность, которую данная модель f приписывает всему набору данных, — это произведение по примерам: $p(\mathcal{S}_n | f) = \prod_{i=1}^n p(y_i | f(x_i))$. Эта величина называется правдоподобием набора данных. Случайно выбранная модель размазывает свою вероятность более или менее равномерно по всем возможным выходам, так что нашим конкретным данным она даёт малое правдоподобие.

Возьмём фотоприложение с тремя классами и тремя размеченными фотографиями. Случайная модель даёт каждому истинному классу вероятность одна треть, так что правдоподобие равно одной трети в кубе, около 0,037. Обученная модель даёт истинным классам 0,9, 0,8 и 0,5, так что правдоподобие равно 0,9 на 0,8 на 0,5, то есть 0,36, примерно в десять раз больше.

Разумная стратегия, следовательно, — выбрать модель, которая делает набор данных как можно более вероятным, и это принцип максимального правдоподобия. Определение: при наборе данных и семействе распределений $p(y | f(x))$, параметризованном f , решение по максимальному правдоподобию — это $f^* = \operatorname{argmax}_f \prod_{i=1}^n p(y_i | f(x_i))$. Это снова задача оптимизации, но теперь, как только распределения выбраны, она следует из законов вероятности, тогда как раньше функцию потерь нам приходилось проектировать.

МАКС. ПРАВДОПОДОБИЕ

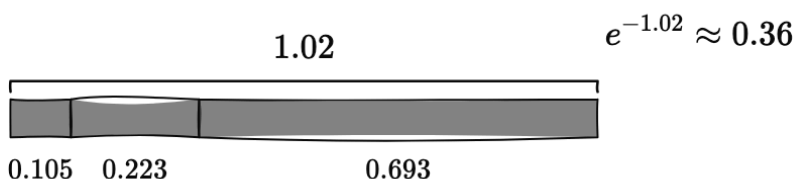
$$f^* = \arg \max_f \prod_{i=1}^n p(y_i | f(x_i))$$

log
↓

$$\arg \min_f \sum_{i=1}^n -\log p(y_i | f(x_i))$$

ПСЕВДОПОТЕРЯ

$$(1/2)^{1000} \approx 10^{-301} \quad \mathbf{X} \quad 1000 \times 0.693 = 693$$

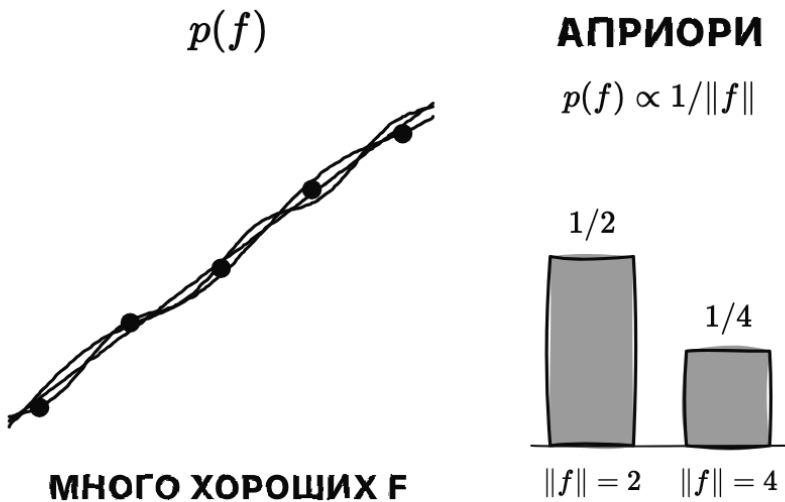


От произведения к сумме логарифмов

Два взгляда тесно связаны. Возьмём логарифмы, которые превращают произведения в суммы, и перейдём от максимизации к минимизации, сменив знак: $\arg \max_f \log \prod_i p(y_i | f(x_i)) = \arg \min_f \sum_i -\log p(y_i | f(x_i))$. Логарифм сохраняет порядок чисел, так что лучшая модель та же, а суммы куда добрее к компьютерам, чем длинные произведения: тысяча множителей по одной второй перемножаются в число с примерно тремя сотнями нулей после запятой, а их минус логарифмы просто складываются примерно в 693. Для нашей обученной модели минус логарифмы 0,9, 0,8 и 0,5 равны примерно 0,105, 0,223 и 0,693, в сумме около 1,02, и e в степени минус 1,02 действительно примерно 0,36. Так что две формулировки совпадают, если считать $-\log p(y | f(x))$ псевдопотерей, и книга говорит, что

каждая используемая на практике функция потерь возникает именно так, из конкретного выбора этого члена.

Раздел 3.3 идёт на шаг дальше, к байесовскому обучению и байесовским нейронным сетям, о которых книга даёт лишь общую идею. Вероятность над выходами уже позволяет модели выражать несколько правдоподобных предсказаний. Но обучение всё ещё возвращает одну-единственную функцию f , тогда как много разных наборов параметров могут быть одинаково хороши, и наличие всех их могло бы дать более верные предсказания.



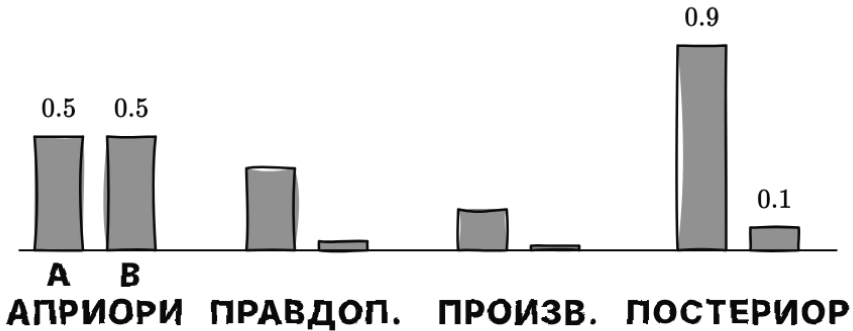
Много хороших моделей и априорное распределение

Решение снова — ещё одно распределение вероятностей, на этот раз над функциями. Мы начинаем с априорного распределения $p(f)$, нашего предпочтения до того, как увидели данные; поскольку f описывается конечным набором параметров, на практике это априорное распределение над весами. Например, функции с меньшей нормой часто предпочтительнее, потому что они устойчивее, и тогда можно взять $p(f) \propto 1/\|f\|$: функция с нормой 2 была бы априори вдвое вероятнее функции с нормой 4.

После наблюдения данных вероятность над функциями сдвигается согласно априорному распределению и правдоподобию, по теореме Байеса: $p(f \mid \mathcal{S}_n) = p(\mathcal{S}_n \mid f) p(f) / p(\mathcal{S}_n)$. Левая часть — апостериорное распределение, убеждение после данных; знаменатель, свидетельство, лишь заставляет апостериорное распределение давать в сумме единицу.

Предположим, есть всего две модели-кандидата, A и B, априори равновероятные, с правдоподобиями 0,36 и 0,04. Априорное на правдоподобие даёт 0,18 и 0,02; свидетельство — их сумма, 0,2; так что апостериорная вероятность равна 0,9 для A и 0,1 для B.

$$p(f | \mathcal{S}_n) = \frac{p(\mathcal{S}_n | f) p(f)}{p(\mathcal{S}_n)}$$



$$0.9 \times 0.8 + 0.1 \times 0.3 = 0.75$$

Байес с двумя моделями

В отличие от одной лучшей модели, апостериорное распределение может продолжать предпочитать несколько функций сразу, что может предсказывать лучше. Для нового входа мы усредняем по моделям, каждая взвешена своей апостериорной вероятностью: $p(y | x) = \int p(y | f(x)) p(f | \mathcal{S}_n) df$, что на практике приближается усреднением по конечной выборке из k моделей. С нашими двумя моделями, если А говорит «собака» с вероятностью 0,8, а В — с 0,3, усреднённое предсказание равно 0,9 на 0,8 плюс 0,1 на 0,3, то есть 0,75: в основном взгляд А, чуть смягчённый сомнением В.

Опыт

Запишите три дня погоды и вероятности двух синоптиков для того, что на самом деле случилось, например 0,9, 0,8 и 0,5 для первого и 0,4, 0,5 и 0,2 для второго. Перемножьте три числа каждого синоптика: 0,36 против 0,04. Затем вместо этого сложите на калькуляторе минус логарифмы: около 1,02 против 3,22; меньшая сумма принадлежит тому же синоптику, что и большее произведение, вот почему максимизация правдоподобия и минимизация логарифмической потери — один и тот же выбор.

Запомним

Для данных i.i.d. правдоподобие набора данных при модели — это произведение вероятностей, которые она даёт каждому истинному выходу, 0,36 для обученной модели против 0,037 для случайной. Максимальное правдоподобие выбирает модель, делающую данные наиболее вероятными; взятие логарифмов превращает это в минимизацию суммы $-\log p(y_i | f(x_i))$, псевдопотери, из которой выводятся практические функции потерь. Байесовское обучение сохраняет распределение над моделями: априорное $p(f)$, обновляемое теоремой Байеса в апостериорное, со свидетельством как нормировкой, 0,9 и 0,1 для наших двух моделей. Предсказания затем усредняют все модели, взвешенные апостериорным распределением, приближённо по конечной выборке моделей, как 0,75 из 0,8 и 0,3.

Вопросы

1. Почему правдоподобие набора данных можно записать как произведение по его примерам?
 - a. Потому что примеры предполагаются независимыми, так что их вероятности перемножаются
 - b. Потому что у модели по одному параметру на пример
 - c. Потому что функция потерь всегда является произведением
2. Почему максимизация правдоподобия — то же самое, что минимизация суммы $-\log p(y_i | f(x_i))$?
 - a. Потому что логарифм обращает порядок чисел
 - b. Потому что логарифм сохраняет порядок чисел и превращает произведение в сумму, а знак минус превращает максимизацию в минимизацию
 - c. Потому что оба всегда равны нулю на лучшей модели
3. Что сохраняет байесовское обучение из того, что максимальное правдоподобие выбрасывает?
 - a. Сами обучающие данные
 - b. Функцию потерь
 - c. Целое распределение над моделями, так что несколько хороших функций могут вносить вклад в предсказания
4. В теореме Байеса какова роль свидетельства $p(S_n)$?
 - a. Оно нормирует апостериорное распределение, чтобы оно давало в сумме единицу

- b. Это априорное предпочтение малых норм
- c. Это правдоподобие лучшей модели

Ответы

1. a) Выборки i.i.d., так что вероятность, которую модель приписывает всему набору данных, — это произведение вероятностей, которые она приписывает каждому истинному выходу.
2. b) Логарифм возрастает, так что лучшая модель не меняется; он превращает произведения в суммы, а смена знака даёт минимизацию псевдопотери.
3. c) Максимальное правдоподобие возвращает одну функцию, а апостериорное распределение может предпочитать несколько функций сразу, и предсказания усредняют их по апостериорному весу.
4. a) Апостериорное равно правдоподобию на априорное, делённому на свидетельство, которое обеспечивает, что апостериорное — настоящее распределение, как 0,18 и 0,02, делённые на 0,2, дали 0,9 и 0,1.

Разобранные примеры

Правдоподобие и логарифмическая потеря

Классификатор даёт истинным классам трёх фотографий вероятности 0,9, 0,8 и 0,5. Вычислите правдоподобие и отрицательное логарифмическое правдоподобие и сравните со случайной моделью, дающей по 1/3.

Правдоподобие: $0,9 \times 0,8 \times 0,5 = 0,36$.

Отрицательное логарифмическое правдоподобие: $-\ln 0,9 - \ln 0,8 - \ln 0,5 \approx 0,105 + 0,223 + 0,693 \approx 1,02$.

Проверка: $e^{(-1,02)} \approx 0,36$.

Случайная модель: $(1/3)^3 \approx 0,037$, примерно в десять раз меньше.

Ответ: Правдоподобие 0,36, отрицательное логарифмическое правдоподобие около 1,02; случайная модель получает около 0,037

Апостериорное распределение над двумя моделями У моделей А и В равные априорные вероятности 0,5 и правдоподобия 0,36 и 0,04. Найдите их апостериорные вероятности и усреднённую вероятность «собаки», если А предсказывает 0,8, а В предсказывает 0,3.

Априорное на правдоподобие: $0,5 \times 0,36 = 0,18$ и $0,5 \times 0,04 = 0,02$.

Свидетельство: $0,18 + 0,02 = 0,2$.

Апостериорные: $0,18 / 0,2 = 0,9$ для А и $0,02 / 0,2 = 0,1$ для В.

Усреднённое предсказание: $0,9 \times 0,8 + 0,1 \times 0,3 = 0,72 + 0,03 = 0,75$.

Ответ: Апостериорные 0,9 и 0,1; усреднённая вероятность собаки 0,75

Задачи

1. Модель даёт истинным выходам четырёх примеров вероятности 0,5, 0,5, 0,8 и 0,25. Вычислите правдоподобие.

Подсказка: Перемножьте четыре числа.

Ответ: $0,5 \times 0,5 \times 0,8 \times 0,25 = 0,05$.

2. У моделей А и В априорные вероятности 0,7 и 0,3 и правдоподобия 0,1 и 0,4. Вычислите их апостериорные вероятности.

Подсказка: Умножьте априорное на правдоподобие для каждой, затем разделите на сумму.

Ответ: 0,07 и 0,12; свидетельство 0,19; апостериорные $\approx 0,37$ для А и $\approx 0,63$ для В. Данные сдвинули убеждение от А к В.

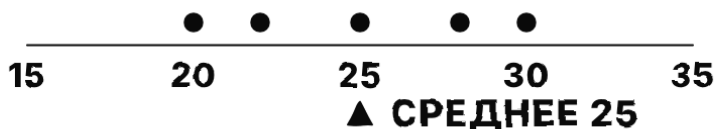
3. При априорном $p(f) \propto 1/\|f\|$ во сколько раз функция с нормой 1 априори вероятнее функции с нормой 5?

Подсказка: Сравните $1/1$ с $1/5$.

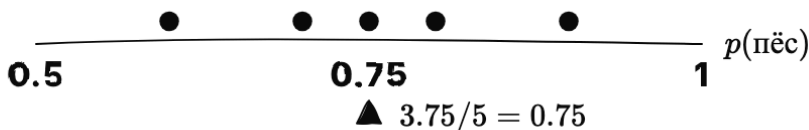
Ответ: В пять раз вероятнее, поскольку $1/1$ в пять раз больше $1/5$; функции с меньшей нормой предпочитаются как более устойчивые.

Приближения, априорные распределения как штрафы и обучение в рассрочку

Попросите пятерых друзей оценить, сколько займёт прогулка: 20, 25, 30, 22 и 28 минут. Их среднее, 25, обычно лучшая догадка, чем любая из них по отдельности. Байесовское предсказание — эта идея, сделанная точной: спросить много правдоподобных моделей и усреднить их, и эта часть показывает, как это делается на практике и какие более простые методы скрыты внутри.



К МОДЕЛЕЙ ИЗ АПОСТЕРИОРНОГО



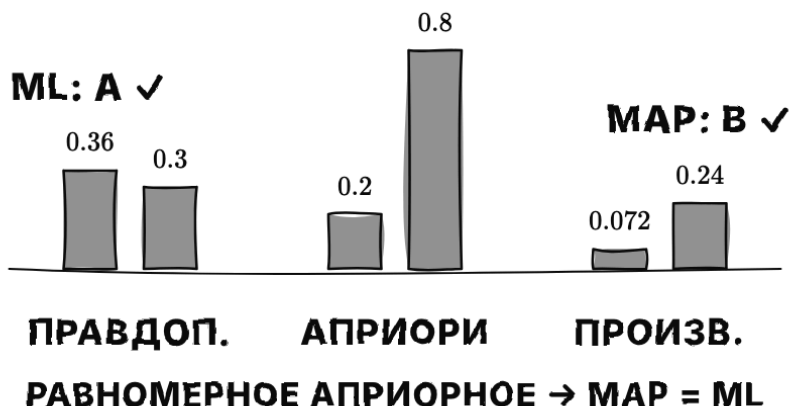
АПОСТЕРИОРНОЕ: РЕДКО ТОЧНО МСМС · ВАРИАЦИОННЫЙ

Усреднение многих догадок

Усреднённое предсказание по всем моделям — интеграл, и на практике его заменяют средним Монте-Карло по k моделям, случайно взятым из апостериорного распределения, $f_k \sim p(f | \mathcal{S}_n)$. Если пять выбранных моделей дают фотографии шанс быть собакой 0,8, 0,7, 0,9, 0,6 и 0,75, усреднённое предсказание — их сумма, 3,75, делённая на 5, то есть 0,75. Красоту этой конструкции, говорит книга, портит один факт: апостериорное распределение почти никогда нельзя вычислить точно, кроме очень особых выборов априорного распределения и правдоподобия. Его приходится приближать, либо методами Монте-Карло с марковскими цепями, либо вариационным выводом, и раздел 9.3.1 покажет одну байесовскую трактовку параметров модели, Monte Carlo dropout.

Книга завершает главу двумя замечаниями. Первое: предположим, нам нужна только одна функция с наибольшей апостериорной плотностью. Тогда свидетельство можно игнорировать, и задача распадается на два члена: $f^* = \operatorname{argmax}_f p(\mathcal{S}_n | f) p(f) = \operatorname{argmax}_f [\log p(\mathcal{S}_n | f) + \log p(f)]$. Первый член — правдоподобие, второй — регуляризационный член, а ответ называется решением максимума апостериорной вероятности, или MAP.

$$f^* = \arg \max_f [\log p(\mathcal{S}_n | f) + \log p(f)]$$



Правдоподобие или правдоподобие на априорное

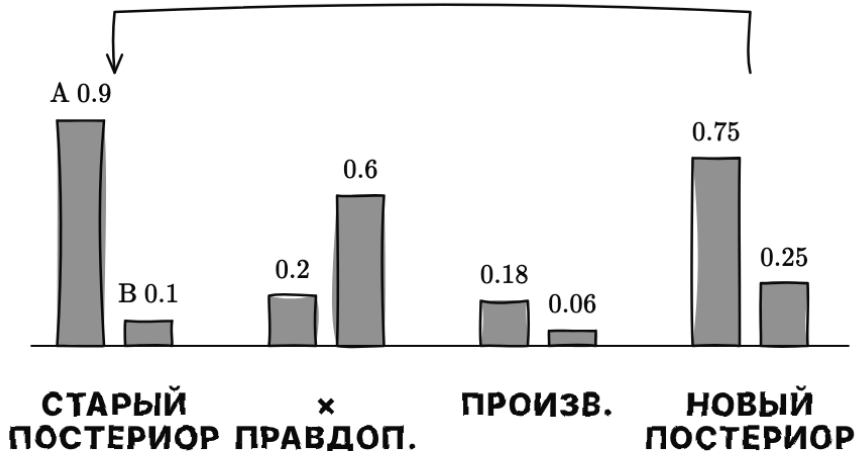
Сравним оба на числах. Модель А объясняет данные с правдоподобием 0,36, но имеет априорную вероятность 0,2; у модели В правдоподобие 0,30 и априорная вероятность 0,8. Максимальное правдоподобие выбирает А, поскольку 0,36 больше 0,30. MAP умножает на априорное: 0,36 на 0,2 даёт 0,072, а 0,30 на 0,8 даёт 0,24, так что MAP выбирает В, модель, которая подходит чуть хуже, но заранее куда правдоподобнее.

Если у каждой функции одинаковый априорный вес, равномерное априорное распределение, второй член постоянен, и MAP сводится к максимальному правдоподобию. Иначе MAP штрафует функции, которые слишком далеко уходят от априорного, что помогает против переобучения и позволяет

накладывать ограничения на f . Вот почему $\log p(f)$ называют регуляризатором: он тянет решение к области, которую предпочитает априорное распределение. Примечание связывает это с двумя интерпретациями вероятности: грубо говоря, максимальное правдоподобие считает параметры фиксированными неизвестными, а данные случайными, это частотный взгляд, тогда как байесовский взгляд считает данные фиксированными, а параметры случайными.

Второе замечание: полная байесовская трактовка легко впитывает новые данные. При новом наборе данных S'_n из того же распределения мы используем апостериорное, вычисленное на первых данных, как новое априорное и обновляем его новым правдоподобием. Книга предлагает смотреть на исходное априорное как на апостериорное после наблюдения вообще ничего.

ПОСТЕРИОР → НОВ. АПРИОРИ: ПОМНИМ



Вчерашнее апостериорное — сегодняшнее априорное

Возьмём модели А и В с апостериорными вероятностями 0,9 и 0,1 после первой порции. У новой порции правдоподобие 0,2 при А и 0,6 при В: умножение даёт 0,18 и 0,06, их сумма 0,24, так что новые апостериорные вероятности равны 0,75 и 0,25. Убеждение сдвинулось к В, но первая порция не забыта: А всё ещё впереди. Вот почему байесовский взгляд может облегчить известную проблему моделей, обучаемых онлайн, на данных, приходящих со временем: катастрофическое забывание, когда обучение на новых данных стирает выученное раньше. Чат-ассистент, обновляемый только на разговорах этого месяца, без какого-либо переноса, был бы подвержен именно этому риску.

Опыт

Возьмите две монеты и заранее решите, что вы верите 9 к 1, что первая из них честная. Подбросьте их несколько раз и для каждой монеты умножьте своё убеждение на то, насколько правдоподобно выглядят её результаты, затем перемасштабируйте два числа так, чтобы они давали в сумме единицу. Сделайте это снова после новых бросков, начиная с убеждений, которые только что вычислили, а не с 9 к 1: это апостериорное становится следующим априорным.

Запомним

Байесовские предсказания усредняют по моделям, взятым из апостериорного распределения, приближение Монте-Карло вроде 0,75 по пяти выборкам, но само апостериорное обычно требует приближения методами Монте-Карло с марковскими цепями или вариационным выводом. Сохранение только самой вероятной функции даёт решение MAP, аргмакс логарифма правдоподобия плюс логарифм априорного, где логарифм априорного действует как регуляризатор; при равномерном априорном оно равно максимальному правдоподобию. MAP может предпочесть чуть хуже подходящую модель, которая априори правдоподобнее, как B вместо A с 0,24 против 0,072. Новые данные впитываются использованием старого апостериорного как нового априорного, что помогает против катастрофического забывания при онлайн-обучении.

Вопросы

1. Когда решение МАР сводится к решению по максимальному правдоподобию?
 - a. Когда априорное распределение равномерно и даёт каждой функции одинаковый вес
 - b. Когда набор данных очень мал
 - c. Когда апостериорное распределение вычислено точно
2. Почему $\log p(f)$ называют регуляризатором?
 - a. Потому что он делает правдоподобие больше
 - b. Потому что он тянет решение к функциям, которые предпочитает априорное, штрафуя те, что далеко от него уходят
 - c. Потому что он убирает свидетельство из теоремы Байеса
3. Как байесовская трактовка впитывает новую порцию данных?
 - a. Переобучаясь с исходного априорного только на новой порции
 - b. Усредняя старый и новый наборы данных
 - c. Используя апостериорное по старым данным как априорное для новых данных
4. Почему байесовские предсказания обычно приходится приближать?
 - a. Потому что апостериорное распределение редко можно вычислить в замкнутом виде

- б. Потому что априорное распределение всегда неизвестно
- с. Потому что средние Монте-Карло точны

Ответы

- 1. а) МАР максимизирует логарифм правдоподобия плюс логарифм априорного; при равномерном априорном второй член — константа, и остаётся максимальное правдоподобие.
- 2. б) Логарифм априорного штрафует функции, отклоняющиеся от априорного, что помогает против переобучения и накладывает ограничения на f .
- 3. с) Предыдущее апостериорное становится исходным убеждением, обновляемым новым правдоподобием, так что старая информация сохраняется, что помогает против катастрофического забывания.
- 4. а) Кроме очень специальных априорных распределений и правдоподобий, у апостериорного нет замкнутого вида, так что используют Монте-Карло с марковскими цепями или вариационный вывод и усредняют по выбранным моделям.

Разобранные примеры

Максимальное правдоподобие против МАР У модели А правдоподобие 0,36 и априорная вероятность 0,2; у модели В правдоподобие 0,30 и априорная вероятность 0,8. Какую выбирает максимальное правдоподобие и какую выбирает МАР?

Максимальное правдоподобие сравнивает правдоподобия: $0,36 > 0,30$, так что оно выбирает А.

МАР сравнивает правдоподобие на априорное: А даёт $0,36 \times 0,2 = 0,072$.

В даёт $0,30 \times 0,8 = 0,24$.

$0,24 > 0,072$, так что МАР выбирает В.

Ответ: Максимальное правдоподобие выбирает А; MAP выбирает В

Обновление в рассрочку После первой порции у моделей А и В апостериорные вероятности 0,9 и 0,1. У новой порции правдоподобие 0,2 при А и 0,6 при В. Вычислите обновлённые апостериорные вероятности.

Используйте старое апостериорное как новое априорное.

Априорное на правдоподобие: $0,9 \times 0,2 = 0,18$ и $0,1 \times 0,6 = 0,06$.

Свидетельство: $0,18 + 0,06 = 0,24$.

Апостериорные: $0,18 / 0,24 = 0,75$ и $0,06 / 0,24 = 0,25$.

Ответ: 0,75 для А и 0,25 для В

Задачи

1. Четыре модели, выбранные из апостериорного распределения, предсказывают вероятности 0,6, 0,9, 0,7 и 0,8, что письмо — спам. Каково предсказание Монте-Карло?

Подсказка: Усредните четыре предсказания.

Ответ: $(0,6 + 0,9 + 0,7 + 0,8) / 4 = 3,0 / 4 = 0,75$.

2. При равномерном априорном над двумя моделями с правдоподобиями 0,4 и 0,1 какую модель выбирает MAP и согласуется ли это с максимальным правдоподобием?

Подсказка: Равномерное априорное умножает обе на одно и то же число.

Ответ: Обе умножаются на одно и то же априорное, так что MAP выбирает модель с правдоподобием 0,4, ровно как и максимальное правдоподобие.

3. Объясните катастрофическое забывание и как использование апостериорного в качестве следующего априорного может

ПОМОЧЬ.

Подсказка: Подумайте о модели, обучаемой на данных, приходящих со временем.

Ответ: Когда модель, обучаемая онлайн, учится только на новых данных, она может потерять выученное раньше. Использование апостериорного по прежним данным как априорного для новых переносит старую информацию вперёд, так что обновление сдвигает убеждение, не стирая его.