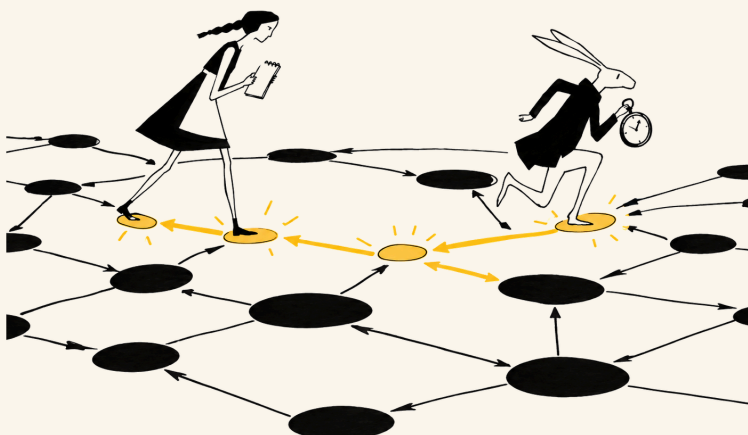


Приключения Алисы в Стране дифференцируемых чудес



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

Введение

Об этой книге

В стране дифференцируемости

Обозначения и символы

Последние мысли перед отправлением

Лицензия и математические предварительные сведения

Распространённые операции над матрицами

Вычислительная сложность

Нотация Эйнштейна и производные

Градиенты, производные по направлению и якобианы

Размерности якобианов и градиентный спуск

Сходимость градиентного спуска

Ускорение градиентного спуска

От теории к практике, и что такое набор данных

Разновидности обучения с учителем

Функции потерь

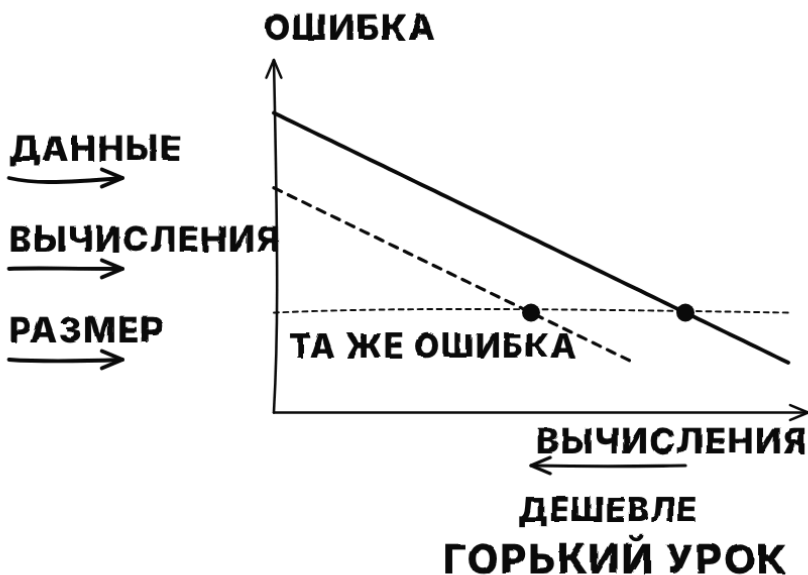
Ожидаемый риск и выбор функции потерь

Максимальное правдоподобие и байесовское обучение

Линейные модели и квадратичные потери

Введение

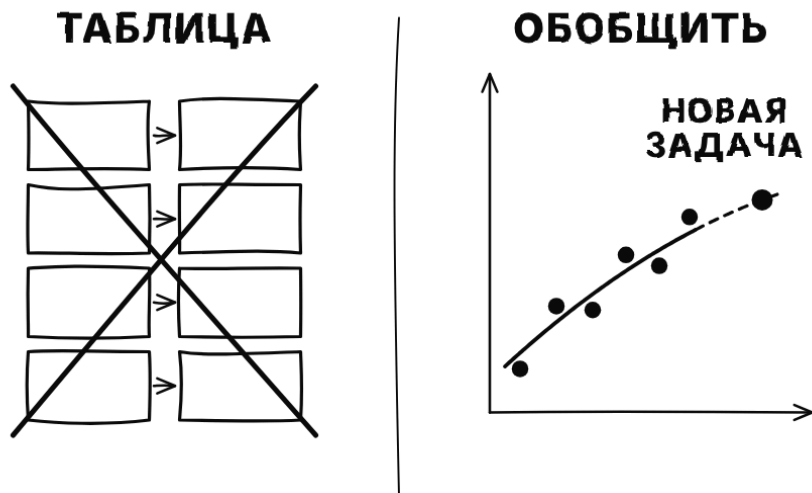
Нейронные сети стали неотъемлемой частью нашего повседневного мира: либо открыто, в облике больших языковых моделей (LLM), либо незаметно, питая бесчисленные технологии и научные открытия, включая дроны, автомобили, поисковые системы, проектирование молекул и рекомендательные системы. Как мы увидим, всё это сделано с опорой на очень небольшой набор руководящих принципов и компонентов, образующих ядро этой книги, тогда как фокус исследований сместился на их масштабирование до пределов физически возможного.



Больше данных, вычислений, размера

Сила масштабирования воплощена в сравнительно недавнем понятии нейронных законов масштабирования, которое, в свою очередь, сыграло ключевую роль в привлечении огромных инвестиций в искусственный интеллект (ИИ). Неформально говоря, практически для любой задачи одновременное увеличение данных, вычислительной мощности и размера моделей почти всегда даёт предсказуемый рост точности. Иначе говоря, вычислительная мощность, необходимая для достижения заданной точности в задаче, уменьшается в постоянное число раз за каждый период времени. Огромную силу сочетания простых, универсальных инструментов с экспоненциально растущей вычислительной мощностью в ИИ Ричард Саттон назвал горьким уроком.

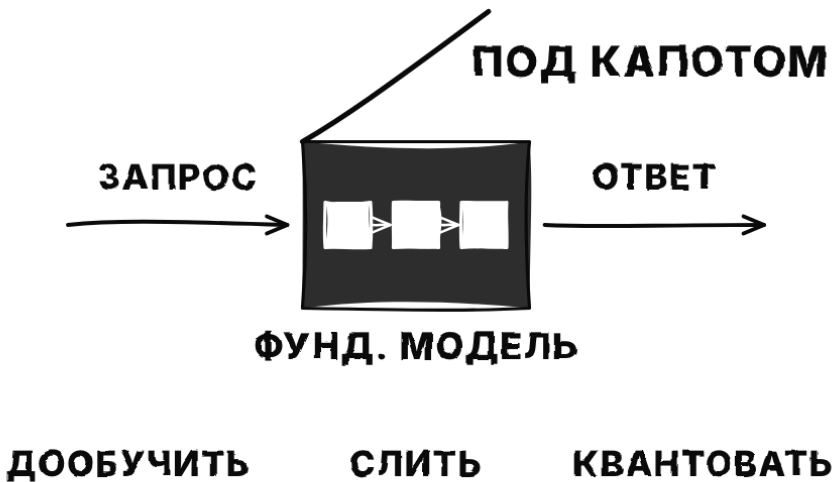
Если принять законы масштабирования как данность, у нас в руках оказывается почти волшебный инструмент. В двух словах, нейронные сети оптимизируются так, чтобы приближать некоторое распределение вероятностей по данным, извлечённым из него. В принципе, это приближение может не удалиться: например, современные нейронные сети настолько велики, что легко могут запомнить все показанные им данные и превратиться в тривиальную таблицу поиска. Однако обученные модели, как показано, хорошо обобщают даже на задачи, которые явно не рассматривались в обучающих данных. Более того, с ростом размера наборов данных граница между тем, что внутри распределения и что вне его, размывается, а крупномасштабные модели показывают признаки сильной способности к обобщению и удивительно слабую зависимость от чистого запоминания, то есть переобучения.



Запомнить или обобщить

Появление чрезвычайно больших моделей, которые можно задействовать для множества прикладных задач, иногда называемых фундаментальными моделями, вместе с живым сообществом открытого кода также изменило то, как мы взаимодействуем с этими моделями. Многие задачи теперь можно решить простым промптингом, то есть взаимодействуя с помощью текстовых или визуальных инструкций с предобученной моделью, найденной в сети, при этом внутреннее устройство модели остаётся полным чёрным ящиком. С высоты птичьего полёта это похоже на переход от необходимости писать свои библиотеки, например на C++, к опоре на открытое или коммерческое программное обеспечение, исходный код которого недоступен. Метафора не так натянута, как может показаться:

сегодня немногие команды в мире располагают вычислительными ресурсами и технической экспертизой, чтобы спроектировать и выпустить по-настоящему крупномасштабные модели, такие как LLM семейства Llama, точно так же как немногие компании имеют ресурсы для создания корпоративного CRM.



Чёрный ящик открыт

И точно так же, как открытое программное обеспечение даёт бесконечные возможности для настройки или создания своих программ с нуля, потребительское оборудование и немного изобретательности дают вам огромный выбор вариантов для экспериментов с дифференцируемыми моделями: от их дообучения под ваши задачи до слияния моделей, квантования

для маломощного оборудования, проверки их устойчивости или даже проектирования совершенно новых вариантов и идей. Для всего этого нужно заглянуть «под капот» и понять, как эти модели обрабатывают и преобразуют данные внутри, со всеми их уловками и особенностями, рождёнными опытом и отладкой. Эта книга — точка входа в этот мир: если вы, как Алиса, любопытны по природе, надеюсь, путешествие придётся вам по вкусу.

Вопросы

1. Что, неформально говоря, утверждают нейронные законы масштабирования?
 - a. Одновременное увеличение данных, вычислений и размера модели почти всегда даёт предсказуемый рост точности
 - b. Как только модель велика, поднять точность может лишь лучший алгоритм обучения
 - c. Большие модели всегда запоминают свои данные и потому хуже справляются с новыми задачами
 - d. Точность растёт только с размером модели, независимо от объёма данных
2. Почему можно было бы ожидать, что очень большие сети не сумеют приблизить распределение, и что наблюдается на деле?
 - a. Они слишком медленно обучаются, но на практике обучаются быстро

- b. Они могли бы просто запомнить данные, как таблица поиска, однако обученные модели хорошо обобщают даже на задачи, которых нет в обучающих данных
 - c. Им нужны размеченные данные, однако они работают вовсе без данных
 - d. Они забывают ранние данные, однако сохраняют всё
3. Что иллюстрирует сравнение с библиотеками на C++ и закрытым коммерческим программным обеспечением?
- a. Что нейронные сети пишутся на C++
 - b. Что промптинг медленнее программирования
 - c. Что использование предобученной модели через запросы похоже на опору на программы, внутренности которых вы не видите, созданные немногими, у кого есть ресурсы
 - d. Что открытые модели нельзя изменять
4. Почему автор говорит, что нужно заглянуть под капот этих моделей?
- a. Потому что промптинг больше не работает
 - b. Потому что фундаментальные модели незаконно использовать без проверки
 - c. Потому что только тогда их можно обучать на телефоне
 - d. Потому что дообучение, слияние, квантование, проверка или проектирование новых моделей требуют понимания того, как они обрабатывают данные внутри

Ответы

1. **a)** Автор описывает законы масштабирования как предсказуемый рост точности при одновременном увеличении данных, вычислительной мощности и размера модели, практически для любой задачи.
2. **b)** Современные сети достаточно велики, чтобы запомнить всё, что видят, но в книге отмечается, что обученные модели, напротив, хорошо обобщают.
3. **c)** Промптинг предобученной модели обращается с ней как с чёрным ящиком, подобно тому как коммерческими программами пользуются без исходного кода; немногие команды могут построить по-настоящему большие модели, как немногие компании могут создать корпоративное CRM.
4. **d)** Все возможности, открытые экспериментатору, от дообучения до новых конструкций, требуют понимания того, как модели преобразуют данные внутри, со всеми их уловками и особенностями.

Задачи

1. Своими словами: что Ричард Саттон назвал горьким уроком и как это связано с законами масштабирования?

Подсказка: Подумайте, что с чем сочетается.

Ответ: Горький урок в том, что простые, универсальные инструменты в сочетании с экспоненциально растущей вычислительной мощностью оказались чрезвычайно сильны в ИИ. Законы масштабирования говорят то же в измеримой форме: если дать таким моделям больше данных, вычислений и размера, точность предсказуемо растёт.

2. Объясните, почему с ростом наборов данных граница между тем, что внутри распределения и что вне его, размывается.

Подсказка: Спросите себя, какую долю возможных задач уже затрагивает огромный набор данных.

Ответ: В очень больших наборах данных присутствует столько видов примеров, что становится трудно сказать, что лежит вне обучающего распределения. Большие модели тогда показывают признаки сильного обобщения и мало зависят от чистого запоминания, то есть переобучения.

3. Перечислите четыре вещи, которые, по словам автора, можно делать с дифференцируемыми моделями на потребительском оборудовании, и скажите, что у них общего.

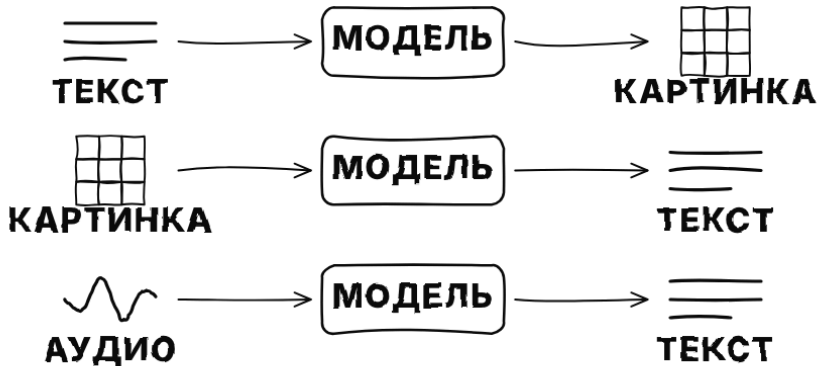
Подсказка: Посмотрите на перечисление после слов «от их дообучения».

Ответ: Дообучать их под свои задачи, сливать модели, квантовать их для маломощного оборудования и проверять их устойчивость (или проектировать новые варианты). Все они требуют заглянуть под капот и понять, как модели обрабатывают данные внутри.

Об этой книге

Мы предполагаем, что наши читатели знакомы с основами машинного обучения (ML), и точнее, обучения с учителем (SL). Обучение с учителем позволяет решать сложные задачи, собирая данные о желаемом поведении и «обучая», то есть оптимизируя, системы так, чтобы они приближали это поведение. Эта обманчиво простая идея чрезвычайно сильна. Например, генерацию изображений можно превратить в задачу сбора достаточно большой коллекции картинок с подписями; имитация английского языка становится задачей собрать большую коллекцию текстов и научиться предсказывать предложение по предыдущим; а диагностика рентгеновского снимка становится равносильной наличию большой базы снимков с соответствующими решениями врачей, как показывает рисунок F.1.2.

ОБУЧЕНА НА ДАННЫХ



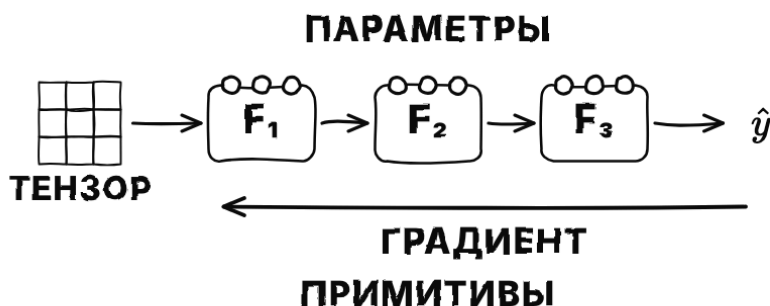
ТОТ ЖЕ РЕЦЕПТ

Задачи как вход и выход

Рисунок распределяет большинство задач по тому, какой вход и выход им нужны. Генерация изображений хочет получить изображение, упорядоченную сетку пикселей, из текста, последовательности символов, тогда как обратная задача, подпись к изображению, состоит в том, чтобы породить подпись по изображению. Другой пример: ответ на звуковой запрос требует текста из аудио, ещё одной упорядоченной последовательности, на этот раз числовой. Поразительно, но устройство моделей во всех случаях следует похожим спецификациям.

В общем виде обучение — это задача поиска. Мы начинаем с того, что задаём программу с большим числом степеней свободы,

которые называем параметрами, и подбираем параметры, пока качество модели нас не удовлетворит. Чтобы сделать эту идею практичной, нам нужны эффективные способы поиска оптимальной конфигурации даже при миллионах, миллиардах или триллионах параметров. Как следует из названия, дифференцируемые модели делают это, ограничивая выбор модели дифференцируемыми компонентами, то есть математическими функциями, которые мы умеем дифференцировать. Умение вычислить производную многомерной функции, градиент, означает знать, что произойдёт, если слегка изменить её параметры, а это, в свою очередь, ведёт к автоматическим процедурам их оптимизации, прежде всего к автоматическому дифференцированию и градиентному спуску. Описание этой конструкции — тема первой части книги, «Компас и стрелка», с главы 2 по главу 6.



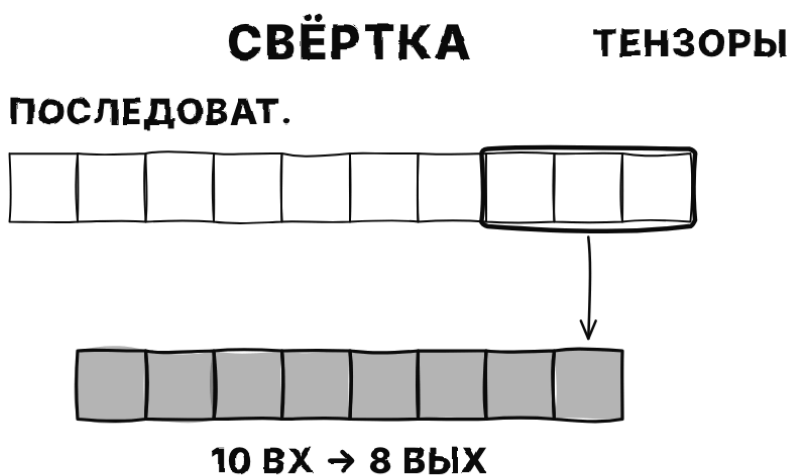
X A B C 1 2 3 **ДИСКРЕТНО**

Цепочка дифференцируемых блоков

Рассматривая нейронные сети просто как композиции дифференцируемых примитивов, мы можем задать два основных вопроса, показанных на рисунке F.1.3. Во-первых, какие типы данных мы можем принимать на вход или выдавать на выход? И во-вторых, какого рода примитивы мы можем использовать? Дифференцируемость — сильное требование, которое не позволяет нам напрямую работать со многими стандартными типами данных, такими как символы или целые числа, которые по своей сути дискретны, а значит, разрывны. Напротив, мы увидим, что дифференцируемые модели легко работают с более сложными данными, представленными большими массивами чисел, которые мы будем называть тензорами, например с изображениями, которыми можно алгебраически

манипулировать простыми композициями линейных и нелинейных преобразований.

Во второй части книги мы сосредоточимся на типичном примере дифференцируемого компонента, операторе свёртки, с главы 7 по главу 9. Свёртки применимы всякий раз, когда наши данные можно представить упорядоченной последовательностью элементов: сюда входят, среди прочего, аудио, изображения, текст и видео. По пути мы также введём ряд полезных приёмов для проектирования глубоких моделей, то есть моделей, составленных из многих шагов подряд, а также несколько важных идей, таких как токенизация текста, авторегрессионная генерация последовательностей и каузальное моделирование, которые лежат в основе современных LLM.



Свёртка скользит по последовательности

Рисунок F.1.3 показывает нейронные сети как последовательности дифференцируемых примитивов, действующих на структурированные массивы, тензоры. Каждый примитив можно отнести к категории по его входной и выходной сигнатуре, которая, в свою очередь, задаёт правила их композиции.

Третья часть книги, «Вниз по кроличьей норе», продолжает наше исследование дифференцируемых моделей, рассматривая альтернативные конструкции для множеств, прежде всего слои внимания и модели-трансформеры в главах 10 и 11, графы в главе 12 и, наконец, рекуррентные слои для временных последовательностей в главе 13.

ЧАСТЬ I

гл. 2–6

ЧАСТЬ II

гл. 7–9

ЧАСТЬ III

ВНИМАНИЕ

САЙТ

ЛАБЫ

Три части книги

Книгу дополняет сайт, где я, надеюсь, буду собирать дополнительные главы и материалы по интересным темам, не привязанным к конкретному типу данных, включая генеративное моделирование, условные вычисления, перенос обучения и объяснимость. Эти главы носят более исследовательский характер, и читать их можно в любом порядке. Кроме того, я предлагаю серию лабораторных занятий с инструкциями в виде ноутбуков, которые охватывают большую часть материала книги, а также продвинутые темы, такие как контрастивное обучение и слияние моделей.

Вопросы

1. Почему книга называет обучение задачей поиска?
 - a. Потому что модели ищут ответы в интернете
 - b. Потому что мы задаём программу с множеством параметров и подбираем их, пока качество нас не удовлетворит
 - c. Потому что обучающие данные нужно проверять на ошибки
 - d. Потому что у каждой задачи ровно одна правильная программа
2. Что даёт нам знание градиента многомерной функции?
 - a. Точные оптимальные параметры за один шаг
 - b. Список всех обучающих данных
 - c. Знание того, что произойдёт при небольшом изменении параметров, что ведёт к автоматической оптимизации
 - d. Способ напрямую работать с символами и целыми числами
3. Почему дифференцируемые модели не могут напрямую работать с символами или целыми числами?
 - a. Эти типы данных дискретны, а значит, разрывны, тогда как дифференцируемость требует непрерывности
 - b. Эти типы данных занимают слишком много памяти
 - c. Символы и целые числа вообще нельзя хранить в тензорах
 - d. Могут, но только со свёртками

4. Когда можно применять свёртки?
- a. Только к изображениям
 - b. Только к данным без порядка
 - c. Всякий раз, когда данные можно представить упорядоченной последовательностью элементов, такой как аудио, изображения, текст и видео
 - d. Только после преобразования данных в целые числа

Ответы

1. **b)** Обучение начинается с программы с большим числом степеней свободы, параметров, которые подбирают, пока модель не станет работать хорошо.
2. **c)** Градиент говорит, что произойдёт при небольшом возмущении параметров, и это ведёт к процедурам вроде автоматического дифференцирования и градиентного спуска.
3. **a)** Дифференцируемость — сильное требование; символы и целые числа по своей сути дискретны, а значит, разрывны.
4. **c)** В книге сказано, что свёртки применимы всякий раз, когда данные — упорядоченная последовательность элементов, и перечислены аудио, изображения, текст и видео.

Задачи

1. Выберите один из примеров книги (генерация изображений, имитация языка, диагностика рентгеновских снимков) и объясните, как он становится задачей обучения с учителем.

Подсказка: Скажите, какие данные вы бы собрали и какому поведению обучали.

Ответ: Для диагностики рентгеновских снимков собирают большую базу снимков с приложенными решениями врачей и обучают систему приближать это поведение: по снимку выдать решение. Задача сводится к сбору примеров желаемого поведения «вход — выход» и оптимизации модели, чтобы она его воспроизводила.

2. Назовите два основных вопроса, которые книга задаёт, когда нейронные сети рассматриваются как композиции дифференцируемых примитивов, и скажите, какие части книги отвечают на них для каких данных.

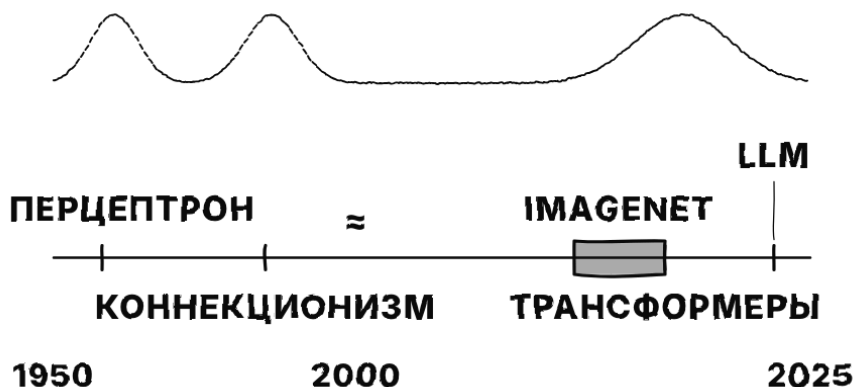
Подсказка: Один вопрос о типах данных, другой о строительных блоках.

Ответ: Какие типы данных мы можем принимать на вход или выдавать на выход, и какие примитивы мы можем использовать? Первая часть выстраивает дифференцируемый каркас, вторая изучает свёртки для упорядоченных последовательностей, таких как аудио, изображения, текст и видео, а третья охватывает множества (внимание и трансформеры), графы и временные последовательности (рекуррентные слои).

В стране дифференцируемости

У нейронных сетей долгая и богатая история. Само название — отголосок ранних попыток моделировать биологические нейроны в двадцатом веке, и похожая терминология осталась повсюду: ради согласия с существующими фреймворками в следующих главах мы можем говорить о нейронах, слоях или, например, активациях. После нескольких волн интереса период между 2012 и 2017 годами увидел беспрецедентный рост сложности сетей, подстёгнутый крупномасштабными бенчмарками и соревнованиями, прежде всего ImageNet Large Scale Visual Recognition Challenge (ILSVRC), о котором мы расскажем в главе 9. Вторая большая волна интереса пришла с появлением трансформеров, темы главы 10, в 2017 году: как компьютерное зрение несколькими годами раньше захватили свёрточные модели, так обработку естественного языка за очень короткий срок захватили трансформеры. Дальнейшие улучшения в эти годы были сделаны для видео, графов, которым посвящена глава 12, и аудио, что вылилось в нынешний ажиотаж вокруг LLM, мультимодальных сетей и генеративных моделей. Здесь не место для полного исторического обзора современных нейронных сетей.

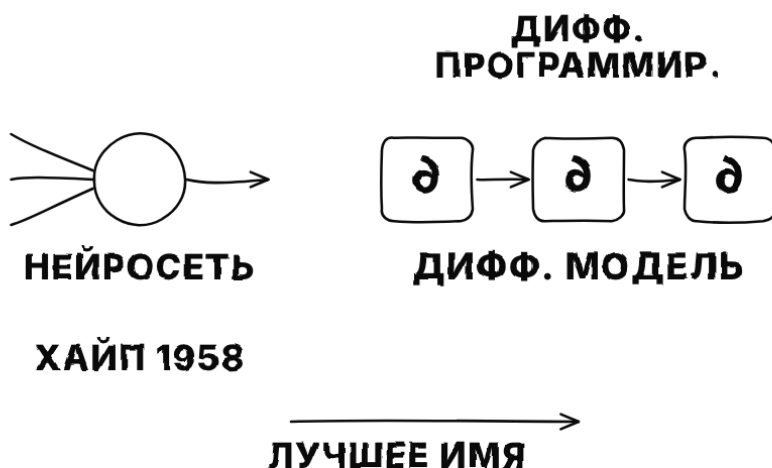
ВОЛНЫ ИНТЕРЕСА



Волны нейронных сетей

Этот период шёл параллельно быстрой эволюции терминологии: от коннекционизма восьмидесятых до использования термина глубокое обучение для современных сетей, в противоположность меньшим и более мелким моделям прошлого. Несмотря на это, все эти термины остаются неумолимо расплывчатыми, потому что современные искусственные сети почти не сохранили сходства с биологическими нейронными сетями и неврологией. Если посмотреть на современные нейронные сети, их главная черта — то, что они составлены из дифференцируемых блоков: по этой причине в этой книге я предпочитаю, где возможно, термин дифференцируемые модели. Взгляд на нейронные сети как на дифференцируемые модели прямо ведёт к более широкой теме дифференцируемого программирования, зарождающейся

дисциплины, которая соединяет информатику и оптимизацию, чтобы изучать дифференцируемые компьютерные программы в целом. Как и многих, меня вдохновил «манифест», опубликованный Янном Лекуном в Facebook в 2018 году, а за связь между нейронными сетями и открытым программированием и разработкой я также благодарен второму манифесту, опубликованному Колином Раффелом в 2021 году.

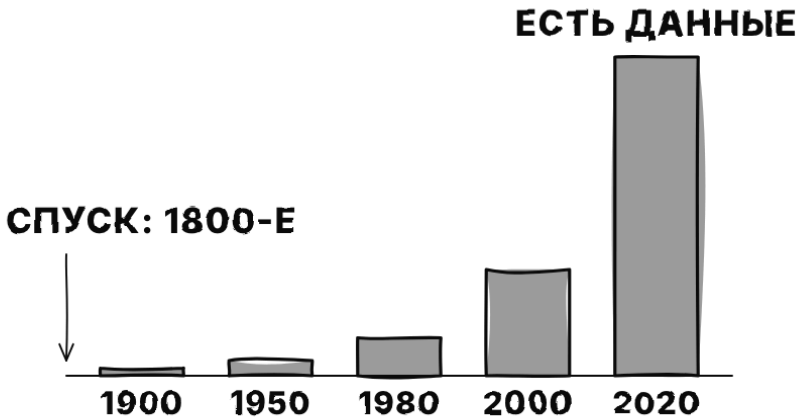


Не нейроны: дифференцируемые блоки

Рисунок в книге показывает ажиотаж вокруг ИИ, только на дворе 1958 год, и американский психолог Фрэнк Розенблатт привлёк большое внимание прессы своими исследованиями «перцептронов», одного из первых работающих прототипов нейронных сетей.

Путешествуя по этой стране дифференцируемых моделей, мы путешествуем и по истории. Основные понятия численной оптимизации линейных моделей градиентным спуском, о которых речь в главе 4, были известны по крайней мере с девятнадцатого века; так называемые «полносвязные сети» в том виде, в каком мы используем их далее, можно датировать восьмидесятыми годами; свёрточные модели были известны и использовались уже в конце девятых. Для подробной истории с собственным взглядом есть также аннотированная история нейронных сетей Юргена Шмидхубера. Однако понадобились многие десятилетия, чтобы накопить достаточно данных и мощности и понять, насколько хорошо эти модели могут работать при достаточном объёме данных и достаточном числе параметров.

ДАННЫЕ, РАСЧЁТ



Старые идеи, новый масштаб

Хотя у нас нет места углубляться во все возможные темы, в том числе из-за того, как быстро движутся исследования, я надеюсь, что книга даёт достаточно материала, чтобы читатель легко ориентировался в самой свежей литературе.

Вопросы

1. Почему автор предпочитает термин «дифференцируемые модели» термину «нейронные сети»?
 - а. Потому что современные сети биологически точны

- b. Потому что их главная черта — составленность из дифференцируемых блоков, а сходства с биологическими нейронами они почти не сохранили
 - c. Потому что «нейронные сети» — торговая марка
 - d. Потому что дифференцируемые модели меньше нейронных сетей
2. Что произошло с обработкой естественного языка после появления трансформеров в 2017 году?
- a. Её за очень короткий срок захватили трансформеры, как раньше компьютерное зрение захватили свёрточные модели
 - b. Она вернулась к системам на правилах
 - c. Она перестала использовать нейронные сети
 - d. Её объединили с ImageNet
3. Почему идеи, известные с девятнадцатого века или с восьмидесятых, так долго не раскрывали свою силу?
- a. Поначалу они были математически ошибочны
 - b. Их изобретатели держали их в секрете
 - c. Понадобились десятилетия, чтобы накопить достаточно данных и вычислительной мощности и увидеть, как хорошо они работают в большом масштабе
 - d. Они работают только с трансформерами

Ответы

1. b) Термины вокруг нейронов расплывчаты, поскольку современные сети едва напоминают биологию; определяет их то, что они построены из дифференцируемых блоков.

2. а) Книга проводит параллель: компьютерное зрение захватили свёрточные модели, а несколько лет спустя обработку языка — трансформеры.

3. с) Градиентный спуск, полносвязные сети и свёртки — всё это старые идеи; не хватало лишь данных и мощности.

Задачи

1. Расположите на шкале времени, опираясь только на сказанное в отрывке: градиентный спуск для линейных моделей, полносвязные сети, свёрточные модели, ажиотаж вокруг перцептрона Розенблатта, подъём благодаря ImageNet, трансформеры.

Подсказка: Отрывок даёт для каждого век, десятилетие или год.

Ответ: Градиентный спуск для линейных моделей: по крайней мере девятнадцатый век. Ажиотаж вокруг перцептрона: 1958.

Полносвязные сети: восьмидесятые. Свёрточные модели: конец девяностых. Подъём благодаря ImageNet: с 2012 по 2017.

Трансформеры: 2017.

2. Своими словами: что такое дифференцируемое программирование, согласно отрывку?

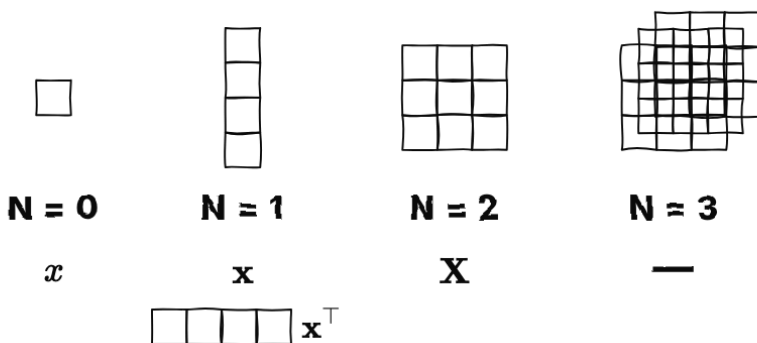
Подсказка: Оно соединяет две области.

Ответ: Это зарождающаяся дисциплина, которая соединяет информатику и оптимизацию, чтобы изучать дифференцируемые компьютерные программы в целом; взгляд на нейронные сети как на дифференцируемые модели прямо ведёт к ней.

Обозначения и символы

Рисунок в книге показывает основные типы данных: скаляры, векторы, матрицы и общие n -мерные массивы. Мы называем их тензорами, а n называем рангом тензора. Для удобства чтения на рисунке вектор показан строкой, но в тексте мы считаем все векторы векторами-столбцами.

ТЕНЗОРЫ



Скаляр, вектор, матрица, тензор

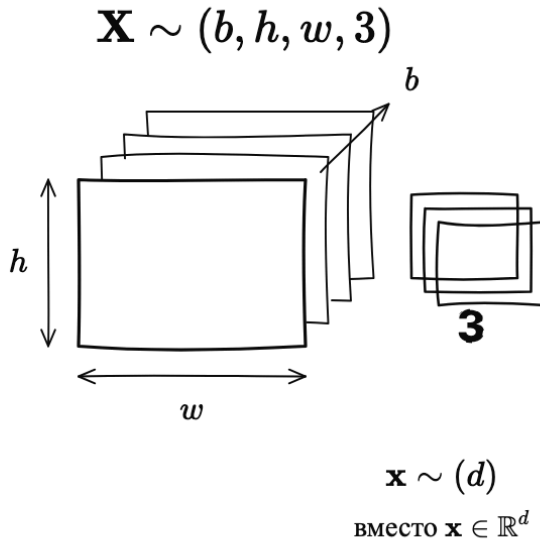
Основной тип данных при работе с дифференцируемыми моделями — тензор, который мы определяем как n -мерный массив объектов, как правило, вещественных чисел. В научной

литературе у тензоров есть более точное определение как мультилинейных операторов, тогда как объекты, которые мы используем в книге, — более простые многомерные массивы; хотя это неточное название, слово «тензор» так распространено, что мы сохраняем здесь эту договорённость. С извинениями перед каждым математиком, который нас читает, мы называем n рангом тензора. Обозначения в книге меняются в зависимости от n .

Тензор из одного элемента, с $n = 0$, — это просто одно значение, скаляр. Для скаляров мы используем строчные буквы, такие как x или y . Если вам любопытно, скаляры называются так потому, что их можно записать как скалярные кратные единицы; а ещё обещаю с этого момента сократить число сносок. Столбцы значений, с $n = 1$, — это векторы. Для векторов мы используем строчный полужирный шрифт, например x . Соответствующий вектор-строка обозначается x^T , когда их нужно различать, а для удобства чтения мы можем и опускать транспонирование, если оно ясно из контекста. Прямоугольные массивы значений, с $n = 2$, — это матрицы. Мы используем прописной полужирный шрифт, например X или Y . Для $n > 2$ специальных обозначений не вводится. Мы избегаем каллиграфических символов, таких как $\mathcal{X}$, которые мы оставляем для множеств или распределений вероятностей.

Для работы с тензорами мы используем разные стратегии индексации, подробнее описанные в разделе 2.1. В большинстве случаев понять алгоритм или операцию — значит понять форму каждого участвующего тензора. Чтобы кратко обозначить форму, мы используем такую запись: $X \sim (b, h, w, z)$. Это тензор ранга 4 с формой (b, h, w, z) . Некоторые размерности могут быть заданы

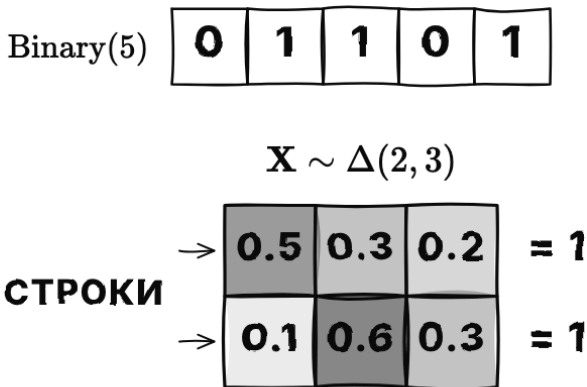
заранее, как эта 3, а другие размерности могут обозначаться переменными. Тот же символ мы используем для обозначения выборки из распределения вероятностей, например $\varepsilon \sim \mathcal{N}(0, 1)$, но делаем это редко, и смысл символа всегда должен быть ясен из контекста. Итак, $\mathbf{x} \sim (d)$ заменит более привычное $\mathbf{x} \in \mathbb{R}^d$, и аналогично $\mathbf{X} \sim (n, d)$ вместо $\mathbf{X} \in \mathbb{R}^{n \times d}$.



Формы: $\mathbf{X} \sim (b, h, w, 3)$

Наконец, нам может понадобиться ограничить элементы тензора, и для этого мы используем особую запись. $\mathbf{x} \sim \text{Binary}(c)$ обозначает тензор только с бинарными значениями, то есть с элементами, равными либо 0, либо 1. $\mathbf{x} \sim \Delta(a)$ обозначает вектор, принадлежащий так называемому симплексу, то есть $x_i \geq 0$ и $\sum_i x_i = 1$. Для тензоров более высокого ранга, например $\mathbf{X} \sim \Delta(n, c)$, мы

считаем, что нормировка выполняется по последней размерности; в этом случае каждая строка X_i принадлежит симплексу.



Бинарный и симплекс

Дополнительные обозначения вводятся в каждой главе по мере необходимости. Есть у нас и несколько символов на полях.

Вопросы

1. Каков ранг матрицы в обозначениях книги?

а. 0

- b. 1
 - c. 2
 - d. Зависит от числа строк
2. О чём говорит запись $X \sim (b, h, w, 3)$?
- a. Что X извлечён из распределения с четырьмя параметрами
 - b. Что X — тензор ранга 4 с формой b на h на w на 3
 - c. Что у X четыре элемента
 - d. Что X — матрица с b строками
3. Какой вектор принадлежит симплексу $\Delta(3)$?
- a. $(0.6, 0.6, -0.2)$
 - b. $(0.5, 0.5, 0.5)$
 - c. $(1, 1, 1)$
 - d. $(0.2, 0.5, 0.3)$
4. Как книга записывает вектор-строку, если её векторы — столбцы?
- a. С транспонированием, $\mathbf{x}^T$
 - b. Прописной полужирной буквой
 - c. Каллиграфической буквой
 - d. С тильдой

Ответы

1. c) Прямоугольные массивы значений, матрицы, имеют $n = 2$, а n — это то, что книга называет рангом.

2. b) Тильда с последующей формой обозначает форму тензора; здесь четыре размерности, значит, ранг 4, причём последняя зафиксирована равной 3.

3. d) У вектора из симплекса неотрицательные компоненты, в сумме дающие единицу; оба условия выполняет только (0.2, 0.5, 0.3).

4. a) Векторы — столбцы, а соответствующий вектор-строка обозначается x^T , когда их нужно различать.

Разобранные примеры

Форма пакета фотографий Пакет содержит 32 цветные фотографии, каждая 64 пикселя в высоту и 64 в ширину, с 3 цветовыми каналами. Запишите его форму в обозначениях книги, укажите ранг и число хранимых значений.

Книга записывает формы с тильдой: перечислите размерности по порядку, сначала пакет, затем высоту, ширину и каналы.

Итак, $X \sim (32, 64, 64, 3)$.

Ранг — это число размерностей, здесь четыре.

Число значений — произведение размерностей: $32 \times 64 \times 64 \times 3 = 32 \times 4096 \times 3 = 393\,216$.

Ответ: $X \sim (32, 64, 64, 3)$, тензор ранга 4, содержащий 393 216 значений.

Задачи

1. Принадлежит ли вектор (0.6, 0.6, −0.2) симплексу $\Delta(3)$?
Объясните.

Подсказка: Проверьте оба условия симплекса, а не только сумму.

Ответ: Нет. Его компоненты в сумме дают 1, но одна компонента отрицательна, а симплекс требует, чтобы каждое $x_i \geq 0$.

2. Таблица из 100 клиентов с 8 признаками у каждого записывается как $\mathbf{X} \sim (100, 8)$. Как то же утверждение выглядело бы в более привычной записи и каков его ранг?

Подсказка: Книга говорит, какую запись заменяет её тильда.

Ответ: $\mathbf{X} \in \mathbb{R}^{100 \times 8}$, матрица, то есть тензор ранга 2.

3. Если $\mathbf{Y} \sim \Delta(5, 4)$, что должно быть верно для каждой строки $\mathbf{Y}$?

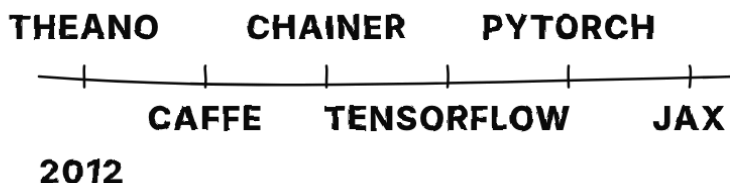
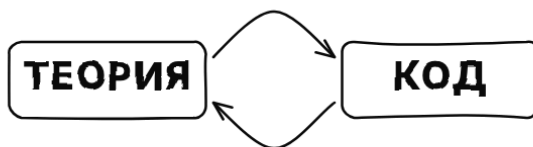
Подсказка: Для более высоких рангов нормировка применяется к последней размерности.

Ответ: Каждая из 5 строк содержит 4 неотрицательные компоненты, в сумме дающие 1, так что каждая строка принадлежит симплексу $\Delta(4)$.

Последние мысли перед отправлением

Книга выросла из моего желания придать связную форму моим лекциям по курсу «Нейронные сети для приложений науки о данных», который я много лет веду в магистратуре по науке о данных в Римском университете Ла Сапиенца. Основные главы книги составляют главную часть курса, а остальные главы — темы, которые я освещаю от случая к случаю, в зависимости от года. Некоторые части дополнены другими курсами, которые я вёл или собираюсь вести, включая части курса «Нейронные сети для компьютерной инженерии», введение в машинное обучение для телекоммуникационной инженерии, а также несколько учебных семинаров, аспирантских курсов и летних школ за эти годы.

КУРС SAPIENZA



Теория и код рука об руку

По теме современных, глубоких нейронных сетей уже есть немало прекрасных и недавних книг. Эта книга в начале охватывает содержание, похожее на все они, тогда как изложение и некоторые дополнительные части или несколько разделов в продвинутых главах пересекаются с ними меньше и зависят в основном от моих исследовательских интересов. Надеюсь, я смогу предложить дополнительный и взаимодополняющий взгляд на существующий материал.

Как подсказывает выбранное мной название, понимание дифференцируемых программ приходит и из теории, и из кода: между тем, как мы проектируем модели и как мы их реализуем, идёт постоянное взаимодействие, и лучший тому пример — такие

темы, как автоматическое дифференцирование. Нынешнее возрождение нейронных сетей, примерно с 2012 года, в значительной мере объясняется доступностью мощных программных библиотек: от Theano к Caffe, Chainer и затем прямо к современным версиям TensorFlow, PyTorch и JAX, среди прочих. Где только возможно, я стараюсь связать обсуждение с понятиями из существующих фреймворков программирования, с упором на PyTorch и JAX. Однако книга — не руководство по программированию, и за полным введением в каждую из библиотек я отсылаю к их документации.

Прежде чем двигаться дальше, я хотел бы перечислить ещё несколько вещей, которыми эта книга не является. Во-первых, я постарался отобрать несколько понятий, которые одновременно распространены сегодня и достаточно общи, чтобы пригодиться в ближайшем будущем. Однако я не могу предвидеть будущее и не стремлюсь к полноте, и некоторые части этих глав могут оказаться неполными или устаревшими к тому моменту, когда вы их читаете. Во-вторых, для каждого понятия я стараюсь привести несколько примеров вариаций, существующих в литературе, например от пакетной нормализации к нормализации по слою. Однако помните, что существуют ещё сотни других: для этого я приглашаю вас исследовать многочисленные страницы Papers With Code. Наконец, это книга об основных компонентах дифференцируемых моделей, но их реализация в большом масштабе и доведение до работы требуют и инженерной искущённости, и немного интуиции. Аппаратную сторону я почти не затрагиваю, а что до интуиции, ничто не заменит опыта и блог-постов с личной позицией, таких как

рецепт обучения Андрея Карпатого или его видеосерия Zero to Hero.

Благодарности. Раскраска уравнений — заслуга прекрасного пакета LaTeX от S. T. John. Цветные изображения Алисы в Стране чудес и чёрно-белые символы на полях лицензированы на Shutterstock.com. Изображения Алисы в Стране чудес на рисунках в основном тексте — репродукции оригинальных иллюстраций Джона Тенниела, спасибо Wikimedia. Я благодарю Роберто Альму за отзыв на предыдущий черновик книги и за то, что он подтолкнул меня к публикации. Я также благодарю Коррадо Дзокколо, Эмануэле Родолу, Марцина Слабого, Константина Бурлаченко и Диего Сандовала за обширные исправления и предложения к нынешней версии, а также всех, кто прислал мне отзывы по электронной почте.

Вопросы

1. Что автор называет главной причиной возрождения нейронных сетей примерно с 2012 года?
 - a. Открытие перцептрона
 - b. Доступность мощных программных библиотек, от Theano и Caffe до TensorFlow, PyTorch и JAX
 - c. Завершение соревнования ImageNet
 - d. Новую теорию биологических нейронов
2. Почему книга постоянно связывает идеи с PyTorch и JAX, не будучи руководством по программированию?

- a. Потому что между тем, как модели проектируются и как они реализуются, идёт постоянное взаимодействие
 - b. Потому что от читателя не ожидается знания математики
 - c. Потому что библиотеки заменяют теорию
 - d. Потому что автор написал эти библиотеки
3. Чем из перечисленного, по словам автора, книга не является?
- a. Книгой об основных компонентах дифференцируемых моделей
 - b. Полным и не стареющим каталогом всех вариантов и аппаратной стороны
 - c. Книгой на основе университетского курса
 - d. Книгой, связанной с существующими фреймворками

Ответы

1. **b)** Возрождение в значительной мере объясняется мощными программными библиотеками, поэтому книга и связывает теорию с кодом.
2. **a)** Понимание дифференцируемых программ приходит из теории и кода вместе; автоматическое дифференцирование — лучший пример этого взаимодействия.
3. **b)** Автор не стремится к полноте, ожидает, что части устареют, отсылает к сотням вариантов в других местах и почти не касается аппаратной стороны.

Задачи

1. Перечислите своими словами три вещи, которыми, по словам автора, эта книга не является.

Подсказка: Они начинаются со слов «во-первых», «во-вторых» и «наконец».

Ответ: Она не полна и не гарантированно актуальна, поскольку область движется и части могут устареть; она не каталог всех вариантов, а лишь несколько примеров из сотен; и она не руководство по реализации моделей в большом масштабе или по аппаратной стороне, для чего нужны инженерное мастерство, интуиция и опыт.

2. Почему автоматическое дифференцирование — лучший для автора пример взаимодействия проектирования и реализации?

Подсказка: Подумайте, что должно существовать в программном обеспечении, чтобы модель можно было обучить.

Ответ: Проектирование дифференцируемых моделей окупается лишь тогда, когда программы умеют эффективно вычислять градиенты; автоматическое дифференцирование — алгоритм, который фреймворки реализуют для этого, так что проектирование модели и её реализация связаны здесь яснее всего.

Лицензия и математические предварительные сведения

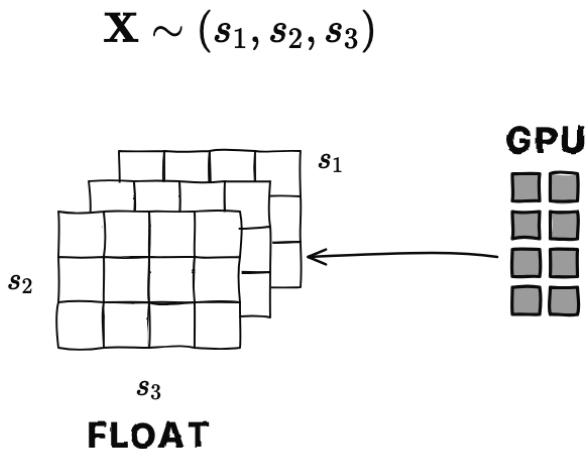
Книга выпущена под лицензией CC BY-SA. Эта лицензия позволяет «пользователям распространять, перерабатывать, адаптировать материал и опираться на него на любом носителе и в любом формате, при условии указания авторства. Лицензия допускает коммерческое использование. Если вы перерабатываете, адаптируете материал или опираетесь на него, вы должны лицензировать изменённый материал на тех же условиях».

Глава 2, Математические предварительные сведения. Коротко: здесь мы сжато излагаем математические понятия, необходимые, чтобы следовать книге. Мы предполагаем, что все эти темы вам уже знакомы, и сосредоточимся скорее на описании конкретных обозначений и на связном обзоре. Где возможно, мы подчёркиваем связь между частью этого материала, например тензорами, и его реализацией на практике.

Глава состоит из трёх частей, следующих одна из другой: начиная с линейной алгебры, переходя к определению градиентов для n -мерных объектов и, наконец, к тому, как оптимизировать функции, используя такие градиенты. Самодостаточный обзор теории вероятностей дан в приложении, с упором на принцип максимального правдоподобия.

Эта глава полна содержания и определений: потерпите немного!

Раздел 2.1, Линейная алгебра. Напомним здесь некоторые основные понятия линейной алгебры, которые пригодятся в дальнейшем и позволят договориться об общих обозначениях. Большая часть книги вращается вокруг идеи тензора.



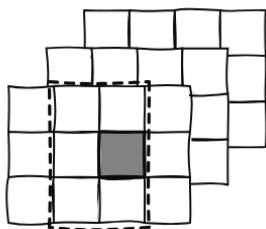
Тензор и его форма

Определение, тензоры: тензор $\mathbf{X}$ — это n -мерный массив элементов одного типа. В книге мы используем $\mathbf{X} \sim (s_1, s_2, \dots, s_n)$, чтобы быстро обозначить форму тензора.

При $n = 0$ мы получаем скаляры, одиночные значения, при $n = 1$ — векторы, при $n = 2$ — матрицы, а иначе — массивы большей размерности. Напомним, что мы используем строчную x для скаляров, строчную полужирную x для векторов и прописную x для матриц.

полужирную $\mathbf{X}$ для матриц. Тензоры в описанном здесь смысле фундаментальны для глубокого обучения, потому что они хорошо подходят для массово-параллельной реализации, например на GPU или более специализированном оборудовании, таком как TPU и IPU.

Тензор описывается типом его элементов и его формой. Большая часть нашего обсуждения будет сосредоточена на тензорах из чисел с плавающей точкой, конкретный формат которых мы рассмотрим позже, но их можно определить и для целых чисел, например в классификации, или для строк, например для текста. Тензоры можно индексировать, чтобы получать срезы, то есть подмножества их значений, и применимо большинство правил индексации NumPy; для удобства чтения в книге мы индексируем с 1, а не с 0. Для простых уравнений мы используем нижние индексы: например, для трёхмерного тензора $\mathbf{X} \sim (a, b, c)$ мы можем написать X_i , чтобы обозначить срез размера (b, c) , или $X_{i j k}$ для одного скаляра. Для более сложных выражений мы используем запятые, например $X_{i,:j:k}$, чтобы обозначить срез размера $(b, k - j)$. Когда нужно избежать нагромождения, мы используем светло-серую запись $[X]_{i j k}$, чтобы визуально отделить индексную часть от остального, причём аргумент внутри скобок может быть и выражением.



$$\mathbf{X}_i \sim (b, c)$$

$$X_{ijk}$$

$$\mathbf{X}_{i,:j:k}$$

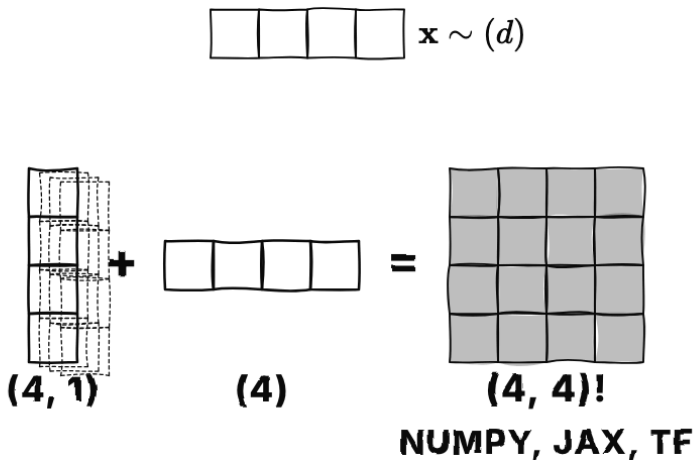
$$+ \quad \times \quad \text{EXP} \quad \text{SIN} \quad \sqrt{\quad}$$

Срез тензора

Распространённые операции над векторами. Нас в основном интересуют модели, которые можно записать как композиции дифференцируемых операций. На деле большинство наших моделей будут состоять из простых композиций сумм, умножений и некоторых дополнительных нелинейностей, таких как экспонента $\exp(x)$, синусы и косинусы и квадратные корни.

Векторы $\mathbf{x} \sim (d)$ — примеры одномерных тензоров. Книги по линейной алгебре заботятся о различии между векторами-столбцами $\mathbf{x}$ и векторами-строками $\mathbf{x}^T$, и мы постараемся по мере возможности придерживаться этой договорённости. В коде это хитрее, потому что векторы-строки и векторы-столбцы соответствуют двумерным тензорам формы $(1, d)$ или $(d, 1)$,

которые отличаются от одномерных тензоров формы (d) . Это важно помнить, потому что большинство фреймворков реализуют правила трансляции, вдохновлённые NumPy, что порождает неинтуитивное поведение; в двух словах, трансляция выравнивает формы тензоров справа и повторяет тензор всякий раз, когда это возможно, чтобы согласовать две формы. Врезка С.2.1 показывает пример очень распространённой ошибки, возникающей из-за неявной трансляции форм тензоров.

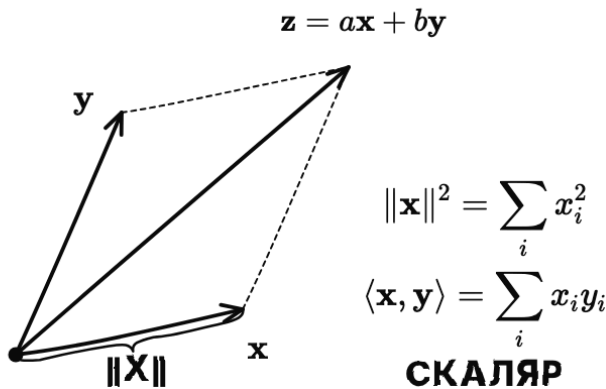


Ловушка трансляции

В этой врезке, которая, вероятно, представляет неправильную трансляцию, «столбец» формы 4 на 1 складывается с одномерным тензором формы 4, и результат имеет форму 4 на 4: поэлементная операция над двумя векторами даёт матрицу из-за

их форм. Тот же результат можно получить практически в любом фреймворке, таком как NumPy, TensorFlow или JAX.

У векторов есть собственная алгебра, которую мы называем векторным пространством, в том смысле, что любые два вектора $\mathbf{x}$ и $\mathbf{y}$ одной формы можно линейно скомбинировать, $\mathbf{z} = a\mathbf{x} + b\mathbf{y}$, получив третий вектор, где $z_i = ax_i + by_i$.



Сложение векторов, измерение длины

Если понимать вектор как точку в d -мерном евклидовом пространстве, сумма истолковывается построением параллелограмма, а расстояние вектора от начала координат даётся евклидовой нормой, или нормой ℓ_2 : $\|\mathbf{x}\| = \sqrt{(\sum_i x_i^2)}$.

Особый интерес представляет квадрат нормы $\|\mathbf{x}\|^2$, так как он соответствует сумме квадратов элементов. Основная операция над векторами, которая нас интересует, — скалярное произведение, которое получается поэлементным умножением двух векторов и суммированием полученных значений.

Определение, скалярное произведение: скалярное произведение двух векторов $\mathbf{x}, \mathbf{y} \sim (d)$ задаётся выражением $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_i x_i y_i$.

Запись с угловыми скобками распространена в физике, и мы иногда используем её для ясности. Важно, что скалярное произведение двух векторов — это скаляр. Например, если $\mathbf{x} = [0.1, 0, -0.3]$ и $\mathbf{y} = [-4.0, 0.05, 0.1]$, то $\langle \mathbf{x}, \mathbf{y} \rangle = -0.4 + 0 - 0.03 = -0.43$.

$$\mathbf{x} \begin{bmatrix} 0.1 & 0 & -0.3 \end{bmatrix}$$

$$\mathbf{y} \begin{bmatrix} -4 & 0.05 & 0.1 \end{bmatrix}$$

$$-0.4 \quad 0 \quad -0.03 = -0.43$$

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

КОСИНУСНОЕ СХ.



-1

Скалярное произведение на примере

Простое геометрическое истолкование скалярного произведения даёт его связь с углом α между двумя векторами: $\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos(\alpha)$.

Следовательно, для двух нормированных векторов, норма которых равна 1, скалярное произведение равносильно косинусу их угла, и в этом случае мы называем скалярное произведение косинусным сходством. Косинусное сходство $\cos(\alpha)$ колеблется между 1, для двух векторов, направленных одинаково, и -1 , для двух векторов, направленных противоположно, а особый случай $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ даёт ортогональные векторы, направленные перпендикулярно. Если посмотреть с другой стороны, для двух

нормированных векторов, с единичной нормой, зафиксировав первый аргумент x , мы получаем следующее.

Вопросы

1. Почему сложение тензора формы $(4, 1)$ с тензором формы (4) даёт результат $(4, 4)$?
 - a. Потому что сложение векторов всегда даёт матрицу
 - b. Потому что трансляция выравнивает формы справа и повторяет каждый тензор для согласования, превращая поэлементную сумму в таблицу 4 на 4
 - c. Потому что фреймворк автоматически транспонирует столбец
 - d. Потому что один из тензоров содержит целые числа
2. Что такое скалярное произведение двух векторов?
 - a. Скаляр: сумма поэлементных произведений
 - b. Вектор поэлементных произведений
 - c. Матрица, построенная из двух векторов
 - d. Угол между двумя векторами
3. Что означает скалярное произведение 0 для двух нормированных векторов?
 - a. Они направлены одинаково
 - b. Они направлены противоположно
 - c. Они ортогональны, направлены перпендикулярно
 - d. Один из них — нулевой вектор

4. Что книга понимает под рангом n тензора?

а. Число его элементов

б. Число его размерностей: 0 для скаляров, 1 для векторов, 2 для матриц

с. Его наибольшее значение

д. Число тензоров, из которых он составлен

Ответы

1. **б)** Трансляция выравнивает формы справа и повторяет тензоры, чтобы их согласовать, поэтому столбец плюс одномерный тензор становится полной матрицей: вероятно, не то, что задумывалось.

2. **а)** Скалярное произведение перемножает векторы поэлементно и суммирует результаты, так что это одно число.

3. **с)** Для единичных векторов скалярное произведение — косинус их угла, а косинус 0 означает перпендикулярные, ортогональные векторы.

4. **б)** Тензор — это n -мерный массив; $n = 0$ даёт скаляры, $n = 1$ — векторы, $n = 2$ — матрицы.

Разобранные примеры

Скалярное произведение вручную Вычислите скалярное произведение $x = [0.1, 0, -0.3]$ и $y = [-4.0, 0.05, 0.1]$.

Перемножьте векторы поэлементно: $0.1 \times (-4.0) = -0.4$, $0 \times 0.05 = 0$ и $-0.3 \times 0.1 = -0.03$.

Сложите три произведения, поскольку скалярное произведение — это сумма поэлементных произведений.

$-0.4 + 0 - 0.03 = -0.43$; результат — скаляр, как и любое скалярное произведение.

Ответ: -0.43

Задачи

1. Найдите евклидову норму $x = [3, 4]$ и квадрат её нормы.

Подсказка: Возведите элементы в квадрат, сложите их, затем извлеките квадратный корень.

Ответ: Квадрат нормы равен $3^2 + 4^2 = 9 + 16 = 25$, а норма равна $\sqrt{25} = 5$.

2. Даны единичные векторы $x = [1, 0]$ и $y = [0, 1]$. Каково их косинусное сходство и что оно говорит об их направлениях?

Подсказка: Для единичных векторов скалярное произведение равно косинусу угла.

Ответ: Их скалярное произведение равно $1 \times 0 + 0 \times 1 = 0$, так что косинусное сходство равно 0: векторы ортогональны, направлены перпендикулярно.

3. Вы складываете тензор формы $(3, 1)$ с тензором формы (3) . Какую форму даст трансляция и как вместо этого получить поэлементную сумму формы (3) ?

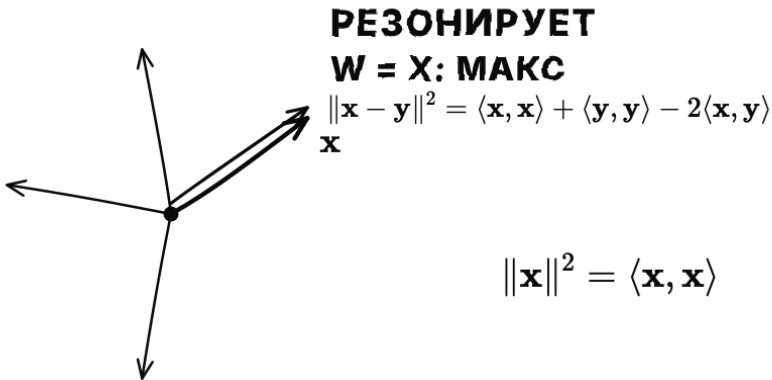
Подсказка: Трансляция выравнивает формы справа и повторяет для согласования.

Ответ: Результат имеет форму $(3, 3)$, потому что столбец повторяется по 3 столбцам, а одномерный тензор — по 3 строкам. Чтобы получить обычную поэлементную сумму, приведите оба тензора к одной форме, например сначала преобразовав тензор $(3, 1)$ в форму (3) .

Распространённые операции над матрицами

$y^* = \operatorname{argmax} \langle x, y \rangle = x$, где argmax обозначает операцию нахождения значения y , соответствующего наибольшему возможному значению на выходе. Из уравнения Е.2.3 мы видим, что для максимизации скалярного произведения второй вектор должен совпадать с первым. Это важно, потому что в следующих главах x будет представлять вход, а w — настраиваемые параметры, так что скалярное произведение максимально всякий раз, когда x «резонирует» с w , что называется сопоставлением с шаблоном.

$$\langle \mathbf{x}, \mathbf{1} \rangle = \sum_i x_i$$



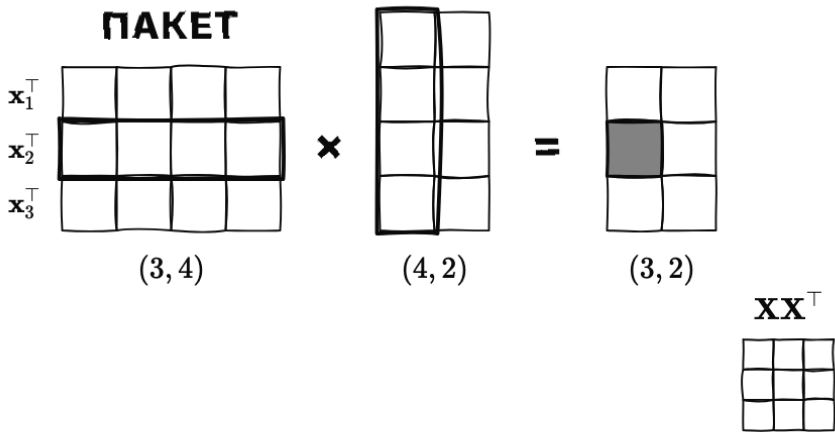
Лучшее совпадение — сам вектор

Закончим двумя дополнительными наблюдениями, которые пригодятся. Во-первых, сумму элементов вектора можно записать как его скалярное произведение с вектором $\mathbf{1}$, целиком состоящим из единиц, $\mathbf{1} = [1, 1, \dots, 1]^T$: $\langle \mathbf{x}, \mathbf{1} \rangle = \sum_{i=1}^d x_i$. Во-вторых, расстояние между двумя векторами тоже можно записать через их скалярные произведения: $\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle$. Случай $\mathbf{y} = \mathbf{0}$ даёт нам $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$. Оба уравнения могут быть полезны при записи формул или в коде.

Распространённые операции над матрицами. В двумерном случае мы имеем матрицы: $\mathbf{X} \sim (n, d)$, сетка элементов от X_{11} в левом верхнем углу до X_{nd} в правом нижнем. В этом случае мы можем говорить о матрице с n строками и d столбцами. Особенно

важно для дальнейшего, что матрицу можно понимать как стопку из n векторов $x_1, x_2, \dots, x_n$, где стопка организована построчно: строки X — это от x_1^T до x_n^T .

СКАЛ. ПРОИЗВ.



Строки на столбцы

Мы говорим, что X представляет пакет векторов данных. Как мы увидим, модели принято определять, и математически, и в коде, для работы с пакетированными данными такого рода. Основная операция для матриц — умножение.

Определение, умножение матриц: для любых двух матриц $X \sim (a, b)$ и $Y \sim (b, c)$ совместимой формы умножение матриц $Z = XY$, где $Z \sim (a, c)$, определяется поэлементно как $Z_{ij} = \langle X_i, Y_j^T \rangle$. То есть

элемент (i, j) произведения — это скалярное произведение i -той строки $\mathbf{X}$ и j -того столбца $\mathbf{Y}$.

Как частный случай, если второй множитель — вектор, мы получаем произведение матрицы на вектор: $\mathbf{z} = \mathbf{W}\mathbf{x}$.

Если истолковать $\mathbf{X}$ как пакет векторов, умножение матриц $\mathbf{X}\mathbf{W}^T$ — простой векторизованный способ вычислить n скалярных произведений, как в уравнении E.2.5, по одному на каждую строку $\mathbf{X}$, одной операцией линейной алгебры. Другой пример: умножение матрицы на её транспонированную, $\mathbf{X}\mathbf{X}^T \sim (n, n)$, — векторизованный способ вычислить сразу все возможные скалярные произведения пар строк $\mathbf{X}$.

Закончим упоминанием ещё нескольких операций над матрицами, которые будут важны.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \odot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$$

$\exp(\mathbf{X})$

e^1	e^2
e^3	e^4

$$\text{mat-exp}(\mathbf{X}) = \sum_k \mathbf{X}^k / k!$$

ТОЛЬКО КВАДР.

МОДУЛЬ LINALG

Элемент за элементом

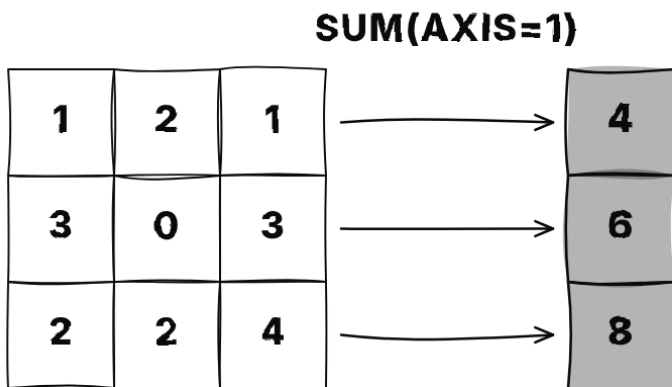
Определение, умножение Адамара: для двух матриц одной формы умножение Адамара определяется поэлементно как $[\mathbf{X} \odot \mathbf{Y}]_{ij} = X_{ij} Y_{ij}$.

Хотя умножение Адамара не обладает всеми интересными алгебраическими свойствами обычного умножения матриц, оно часто используется в дифференцируемых моделях для операций маскирования, например обнуления некоторых элементов, или операций масштабирования. Мультипликативные взаимодействия также стали популярны в некоторых недавних семействах моделей, как мы увидим далее.

Иногда мы пишем выражения вроде $\exp(\mathbf{X})$, которые следует понимать как поэлементное применение операции: $[\exp(\mathbf{X})]_{ij} = \exp(X_{ij})$.

Для сравнения, «настоящая» матричная экспонента определяется для квадратной матрицы как $\text{mat-exp}(\mathbf{X}) = \sum_{k=0}^{\infty} \mathbf{X}^k / k!$.

Важно, что уравнение Е.2.6 можно определить для тензоров любой формы, тогда как Е.2.7 верно только для квадратных матриц. Вот почему во всех фреймворках, таких как PyTorch, есть специальные модули, собирающие все специфически матричные операции, такие как обратные матрицы и определители. Врезка С.2.2 даёт пример, противопоставляя поэлементную экспоненту матрицы матричной экспоненте из учебников линейной алгебры: в PyTorch первая — обычная функция `exp`, вторая живёт в подпакете линейной алгебры как `matrix_exp`. Специализированные операции линейной алгебры обычно заключены в собственный подпакет.



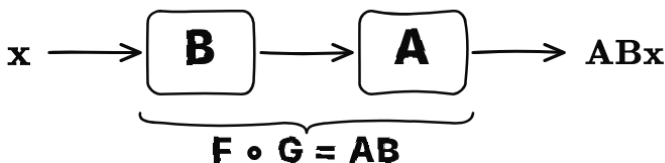
Свёртывание вдоль оси

Наконец, операции свёртывания, такие как сумма или среднее, вдоль осей мы можем записывать без указания нижних и верхних индексов, и в этом случае считаем, что суммирование идёт по всей оси: $\sum_i X_i = \sum_{i=1}^n X_i$. В PyTorch и других фреймворках операции свёртывания соответствуют методам с аргументом оси, например сумма вдоль оси 1.

Врезка в книге спрашивает: об определении умножения матриц, почему оно определено как в уравнении Е.2.4, а не как умножение Адамара? Рассмотрим вектор x и некоторую общую функцию f , определённую на нём. Функция называется линейной, если $f(\alpha x_1 + \beta x_2) = \alpha f(x_1) + \beta f(x_2)$. Любую такую функцию можно представить матрицей A ; это видно, если

разложить оба вектора по базису. Тогда произведение матрицы на вектор $A\mathbf{x}$ соответствует применению функции, $f(\mathbf{x}) = A\mathbf{x}$, а умножение матриц AB соответствует композиции функций $f \circ g$, где $(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$ и $g(\mathbf{x}) = B\mathbf{x}$.

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



Матрицы — линейные функции

Вопросы

1. Для единичных векторов какой $\mathbf{y}$ максимизирует скалярное произведение $\langle \mathbf{x}, \mathbf{y} \rangle$?
 - a. $\mathbf{y} = -\mathbf{x}$
 - b. $\mathbf{y} = \mathbf{x}$

- c. Любой y , ортогональный x
 - d. Нулевой вектор
2. Что такое элемент (i, j) матричного произведения XY ?
- a. $X_{i,j}$ умножить на $Y_{i,j}$
 - b. Сумма строки i матрицы X
 - c. Скалярное произведение i -той строки X и j -того столбца Y
 - d. Скалярное произведение i -того столбца X и j -той строки Y
3. Что вычисляет произведение XX^T для пакета X из n векторов-строк?
- a. Все скалярные произведения пар строк, в матрице n на n
 - b. Сумму всех строк
 - c. Поэлементные квадраты X
 - d. Норму каждого столбца
4. Почему умножение матриц определено как «строка на столбец», а не поэлементно?
- a. Потому что так быстрее на GPU
 - b. Потому что линейная функция — это матрица, и матричное произведение соответствует композиции двух линейных функций
 - c. Потому что произведение Адамара работает только для векторов
 - d. Потому что это избавляет от ошибок трансляции

Ответы

- 1. б)** Чтобы максимизировать скалярное произведение, второй вектор должен совпадать с первым; поэтому вход «резонирует» с параметрами, которые ему соответствуют, — сопоставление с шаблоном.
- 2. с)** Умножение матриц определяется поэлементно как скалярное произведение строки i первой матрицы и столбца j второй.
- 3. а)** Умножение матрицы на её транспонированную — векторизованный способ вычислить сразу все попарные скалярные произведения её строк.
- 4. б)** Любую линейную функцию можно представить матрицей; произведение матрицы на вектор — это её применение, а произведение матриц — композиция функций.

Разобранные примеры

Матричное произведение против произведения Адамара

Пусть $X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ и $Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$. Вычислите матричное произведение XY и произведение Адамара $X \odot Y$.

Для матричного произведения каждый элемент — строка X , скалярно умноженная на столбец Y .

Строка 1 на столбец 1: $1 \times 5 + 2 \times 7 = 19$; строка 1 на столбец 2: $1 \times 6 + 2 \times 8 = 22$.

Строка 2 на столбец 1: $3 \times 5 + 4 \times 7 = 43$; строка 2 на столбец 2: $3 \times 6 + 4 \times 8 = 50$.

Для произведения Адамара перемножьте соответствующие элементы: $1 \times 5 = 5$, $2 \times 6 = 12$, $3 \times 7 = 21$, $4 \times 8 = 32$.

Ответ: $XY = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$, тогда как $X \odot Y = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$.

Задачи

1. С помощью вектора из единиц найдите сумму элементов $x = [2, -1, 4]$.

Подсказка: Сумма элементов равна $\langle x, 1 \rangle$.

Ответ: $\langle x, 1 \rangle = 2 \times 1 + (-1) \times 1 + 4 \times 1 = 5$.

2. Для $x = [1, 2]$ и $y = [3, 1]$ вычислите квадрат расстояния $\|x - y\|^2$, используя только скалярные произведения, затем проверьте напрямую.

Подсказка: Используйте $\|x - y\|^2 = \langle x, x \rangle + \langle y, y \rangle - 2\langle x, y \rangle$.

Ответ: $\langle x, x \rangle = 5$, $\langle y, y \rangle = 10$, $\langle x, y \rangle = 3 + 2 = 5$, так что $5 + 10 - 10 = 5$.

Напрямую: $x - y = [-2, 1]$ и $(-2)^2 + 1^2 = 5$.

3. Матрица X имеет форму $(10, 3)$, а W — форму $(4, 3)$. Какова форма XW^T и что содержит каждый элемент?

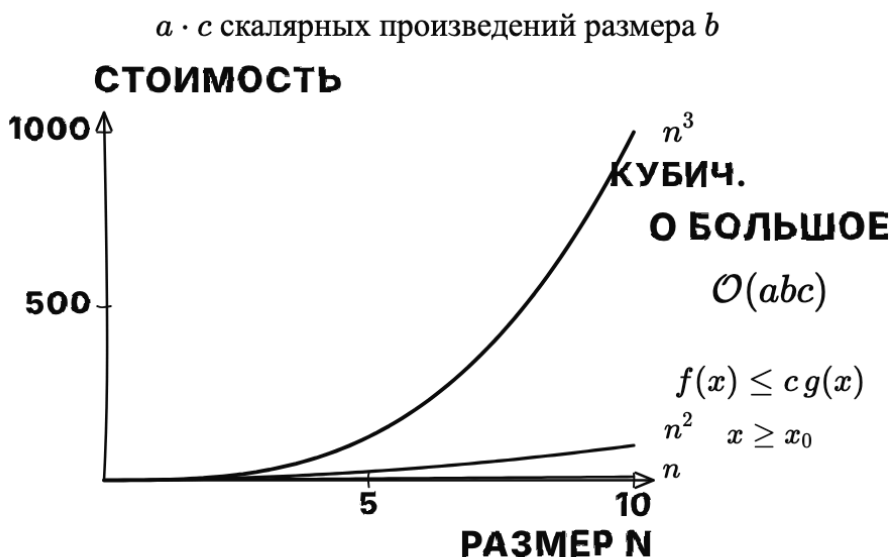
Подсказка: Сначала транспонируйте W , затем проверьте внутренние размерности.

Ответ: W^T имеет форму $(3, 4)$, так что XW^T имеет форму $(10, 4)$.

Каждый элемент (i, j) — скалярное произведение строки i матрицы X и строки j матрицы W .

Вычислительная сложность

Я воспользуюсь умножением матриц, чтобы ввести тему сложности операции. Глядя на уравнение Е.2.4, мы видим, что вычисление матрицы $Z \sim (a, c)$ из входных аргументов $X \sim (a, b)$ и $Y \sim (b, c)$ требует ac скалярных произведений размерности b , если прямо применить определение, и это мы называем временной сложностью, тогда как потребность в памяти для последовательной реализации — это просто размер выходной матрицы, и это мы называем пространственной сложностью.



Как стоимость растёт с размером

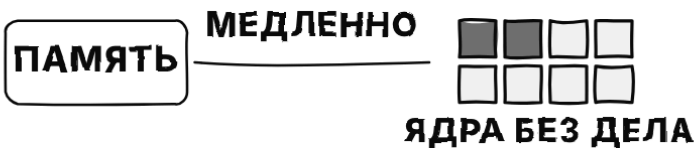
Чтобы отвлечься от конкретных деталей оборудования, информатика сосредоточивается на так называемой нотации $\mathcal{O}$, от немецкого *Ordnung*, что означает порядок приближения. Функция $f(x)$ называется $\mathcal{O}(g(x))$, где мы считаем входы и выходы неотрицательными, если можно найти константу c и значение x_0 такие, что $f(x) \leq c \cdot g(x)$ для любого $x \geq x_0$. Это означает, что как только x становится достаточно большим, мы можем игнорировать в нашем анализе все множители, кроме $g(x)$. Это называется асимптотическим анализом. Следовательно, мы можем сказать, что наивная реализация умножения матриц имеет сложность $\mathcal{O}(abc)$, растущую линейно по всем трём входным параметрам. Для двух квадратных матриц размера (n, n) мы говорим, что умножение матриц кубично по размерности входа.

Рассуждать в терминах асимптотической сложности важно и элегантно, но выбор алгоритма только по сложности $\mathcal{O}$ не обязательно оборачивается практическим выигрышем в производительности, который зависит от многих деталей: какое используется оборудование, какой поддерживается параллелизм и так далее. Когда вы вызываете конкретный примитив в фреймворке линейной алгебры, например умножение матриц в PyTorch, исполняемая низкоуровневая реализация, ядро, зависит от оборудования во время выполнения через процесс, известный как диспетчеризация; поэтому один и тот же код может работать через ядро для GPU, ядро для CPU, ядро для TPU и так далее. Ещё сложнее делают это компиляторы, такие как XLA, которые могут оптимизировать код, сливая операции с учётом конкретного целевого оборудования. Например, известно, что лучший асимптотический алгоритм умножения двух квадратных

матриц размера (n, n) масштабируется как $O(n^c)$ при константе $c < 2.4$, что гораздо лучше кубического требования $O(n^3)$ наивной реализации. Однако эти алгоритмы гораздо труднее эффективно распараллелить на высокопараллельном оборудовании, таком как GPU, поэтому на практике они редки.

$O(n^{2.37})$ плохо ложится на GPU

MEMORY-BOUND



COMPUTE-BOUND



ПРОФАЙЛЕР

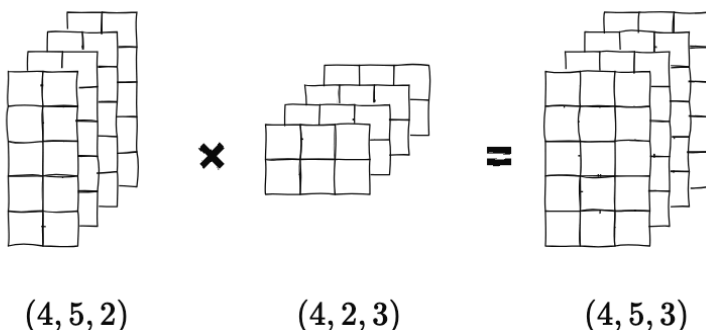
Ограничено памятью или вычислениями

Заметьте, что с точки зрения асимптотической сложности доступ к параллельной среде с k процессорами ничего не меняет, поскольку он может дать, в лучшем случае, лишь постоянное ускорение $1/k$ по сравнению с непараллельной реализацией. Кроме того, асимптотическая сложность не учитывает время, которое уходит на перемещение данных с одного места на другое, а оно во многих ситуациях может стать главным узким местом. В

таких случаях мы говорим, что реализация ограничена памятью, в противоположность ограниченной вычислениями. На практике это можно проверить только запустив профилировщик на коде. Мы увидим, что анализ сложности алгоритма далеко не тривиален из-за взаимодействия асимптотической сложности и наблюдаемой сложности.

Тензорные операции высших порядков. Векторы и матрицы интересны тем, что позволяют определить большое число операций, которые в высших размерностях не определены или сложны, например матричные экспоненты, умножение матриц и определители. При переходе к высшим размерностям большинство интересующих нас операций — это либо пакетные варианты матричных операций, либо конкретные сочетания матричных операций и операций свёртывания.

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



ПАКЕТ ПЕРВЫМ

Пакетное умножение матриц

Как пример первого, рассмотрим два тензора $\mathbf{X} \sim (n, a, b)$ и $\mathbf{Y} \sim (n, b, c)$. Пакетное умножение матриц, BMM, определяется как $[\text{BMM}(\mathbf{X}, \mathbf{Y})]_i = \mathbf{X}_i \mathbf{Y}_i \sim (n, a, c)$.

Операции в большинстве фреймворков прозрачно работают с пакетными версиями своих аргументов, которые, как и в этом случае, считаются ведущими размерностями, первыми.

Например, пакетное умножение матриц в PyTorch — то же самое, что обычное умножение матриц, как показывает врезка С.2.3: умножение тензора формы 4, 5, 2 на тензор формы 4, 2, 3 даёт результат формы 4, 5, 3. Большинство операций могут работать с входами, которые могут быть пакетными.

Как пример операции свёртывания, рассмотрим два тензора $X, Y \sim (a, b, c)$. Обобщённую версию скалярного произведения, GDT, можно записать как $\text{GDT}(X, Y) = \sum_{i,j,k} [X \odot Y]_{i,j,k}$, что есть просто скалярное произведение «расплющенных» версий входов. Этот краткий обзор охватывает большинство тензорных операций, которые мы будем использовать в остальной части книги, а дополнительный материал вводится по мере необходимости.

Вопросы

1. Почему наивное умножение матриц $X \sim (a, b)$ на $Y \sim (b, c)$ называют $\mathcal{O}(abc)$?
 - a. Потому что ему нужно $a \cdot c$ скалярных произведений, каждое размерности b
 - b. Потому что у выхода $a \cdot b \cdot c$ элементов
 - c. Потому что ему нужно $a + b + c$ операций
 - d. Потому что оно использует три процессора
2. Почему алгоритмы умножения матриц быстрее $\mathcal{O}(n^3)$ редко используются на практике?
 - a. Они дают неверные результаты
 - b. Им нужно больше памяти, чем есть у любого компьютера
 - c. Их гораздо труднее эффективно распараллелить на GPU
 - d. Они работают только для маленьких матриц
3. Что значит, что реализация ограничена памятью?

- a. Её время работы определяется перемещением данных, а не вычислениями
 - b. Она не может работать без GPU
 - c. Её сложность O большая кубическая
 - d. Она использует меньше памяти, чем ограниченная вычислениями
4. Почему наличие k процессоров не меняет сложность O большая?
- a. Потому что процессоры не могут делить данные
 - b. Потому что это даёт в лучшем случае постоянное ускорение $1/k$, а O большая игнорирует постоянные множители
 - c. Потому что O большая применимо только к GPU
 - d. Потому что k процессоров всегда медленнее

Ответы

1. **a)** Прямое применение определения требует abc скалярных произведений, каждое по b элементам, так что работа растёт как abc .
2. **c)** Асимптотически лучшие алгоритмы существуют, с показателем ниже 2.4, но они плохо распараллеливаются на высокопараллельном оборудовании, таком как GPU.
3. **a)** Асимптотический анализ игнорирует перемещение данных, которое может стать главным узким местом; тогда реализация ограничена памятью, и узнать это может только профилировщик.
4. **b)** Параллелизм даёт не более чем постоянный множитель $1/k$, а асимптотическая нотация отбрасывает константы.

Разобранные примеры

Подсчёт работы матричного произведения Сколько скалярных произведений и сколько скалярных умножений нужно наивной реализации, чтобы умножить $X \sim (100, 200)$ на $Y \sim (200, 50)$?

Выход Z имеет форму $(a, c) = (100, 50)$, так что нужно вычислить $100 \times 50 = 5000$ элементов.

Каждый элемент — одно скалярное произведение размерности $b = 200$, которому нужно 200 умножений.

Всего умножений: $5000 \times 200 = 1\,000\,000$, то есть ровно $a \cdot b \cdot c$, что согласуется с $\mathcal{O}(abc)$.

Ответ: 5000 скалярных произведений и 1 000 000 умножений.

Задачи

1. Две квадратные матрицы размера (n, n) перемножаются наивно. Если n удваивается, во сколько раз растёт работа?

Подсказка: Наивный алгоритм кубичен по n .

Ответ: В $2^3 = 8$ раз, так как работа растёт как n^3 .

2. В PyTorch тензор формы $(8, 3, 4)$ матрично умножается на тензор формы $(8, 4, 6)$. Какова форма результата и почему?

Подсказка: Ведущая размерность трактуется как пакет.

Ответ: $(8, 3, 6)$: первая размерность — пакет из 8, и каждая пара срезов, $(3, 4)$ на $(4, 6)$, даёт матрицу $(3, 6)$.

3. Объясните своими словами, почему $\text{GDT}(X, Y)$ для двух тензоров формы (a, b, c) — это просто скалярное произведение.

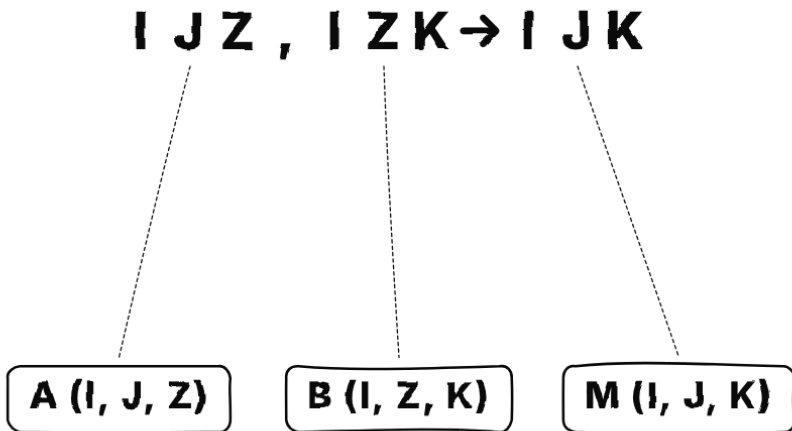
Подсказка: Представьте, что каждый тензор выложен в один длинный вектор.

Ответ: GDT перемножает тензоры поэлементно и суммирует всё; если оба тензора расплющить в векторы длины abc в одном и том же

порядке, это в точности сумма поэлементных произведений,
обычное скалярное произведение.

Нотация Эйнштейна и производные

Это необязательный раздел об einsum, наборе соглашений, который позволяет пользователю задать практически любую тензорную операцию, включая свёртывания, суммы и умножения, простым синтаксисом на основе текстовых строк. Врезка С.2.4 показывает два примера einsum в PyTorch: пакетное умножение матриц, записанное строкой «и же зет, и зет ка, в и же ка», и обобщённое скалярное произведение, записанное «и же ка, и же ка, в ничего».



PYTORCH = JAX

Читаем строку einsum

Чтобы ввести нотацию, рассмотрим снова два примера, показанные ранее в уравнениях Е.2.9 и Е.2.10, выписав явно все оси: $M_{ijk} = \sum_z A_{ij} B_{z_k}$ и $M = \sum_i \sum_j \sum_k X_{ijk} Y_{ijk}$.

В духе нотации Эйнштейна, в упрощённой версии, не различающей верхние и нижние индексы, мы можем упростить оба уравнения, убрав суммы, по соглашению, что по любому индексу, стоящему справа, но не слева, идёт суммирование. Итак, $M_{ijk} = A_{ij} B_{z_k}$ означает сумму по z , а $M = X_{ijk} Y_{ijk}$ означает сумму по i, j и k .

Затем мы можем сжать оба определения, выделив индексы в одну строку, где операнды теперь слева: «и же зет, и зет ка, в и же ка» для пакетного умножения матриц и «и же ка, и же ка, в ничего» для обобщённого скалярного произведения.

Между определениями в уравнениях Е.2.13 и Е.2.14 и их упрощёнными строковыми определениями есть прямое взаимно-однозначное соответствие. В большинстве фреймворков это реализовано в операции `einsum`, как показывает врезка С.2.4.

Преимущество этой нотации в том, что нам не нужно помнить API фреймворка, чтобы реализовать данную операцию, а перевод из одного фреймворка в другой прозрачен, потому что синтаксис `einsum` одинаков. Например, в PyTorch несколько методов умножения матриц, включая `matmul` и `bmm`, с разными правилами трансляции и ограничениями на формы, а `einsum` даёт для всех них единый синтаксис. Кроме того, определение нашего пакетного умножения матриц через `einsum` совпадает, например, с определением в JAX, как показывает врезка С.2.5, с той же самой строкой.

Работа с транспонированными осями тоже проста. Например, для $A \sim (n, a, b)$ и $B \sim (n, c, b)$ пакетное умножение $[A]_i$ на $[B^T]_i$ получается перестановкой соответствующих осей в определении einsum, на «и же зет, и ка зет, в и же ка».

$$\partial$$

$$ij\ z, i\ zk \rightarrow ijk$$

$$\updownarrow$$

$$ij\ z, i\ kz \rightarrow ijk$$

ГРАДИЕНТЫ И ЯКОБИАНЫ

Транспонирование перестановкой букв

По этим причинам einsum и его обобщения, такие как популярный пакет einops, недавно обрели широкую популярность.

Раздел 2.2, Градиенты и якобианы. Как подсказывает слово «дифференцируемые», градиенты играют в книге ключевую роль, давая способ оптимизировать наши модели полуавтоматическими механизмами, происходящими от

градиентного спуска. Напомним здесь некоторые базовые определения и понятия, касающиеся дифференцирования многозначных функций. Мы сосредоточимся на свойствах, которые будут существенны позже, отчасти за счёт математической точности.

Производные скалярных функций. Начнём с простой функции $y = f(x)$ со скалярным входом и скалярным выходом; её производная определяется так.

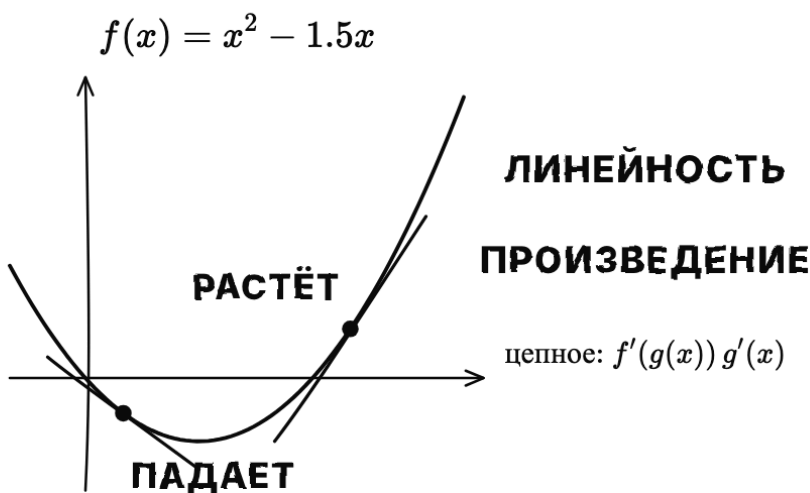
Определение, производная: производная $f(x)$ определяется как $f'(x) = \lim_{h \rightarrow 0} (f(x + h) - f(x)) / h$.

Для обозначения производных мы используем разные записи. ∂ будет обозначать в общем производные и градиенты любой размерности, векторы и матрицы; ∂_x или $\partial/\partial x$ подчёркивает входной аргумент, по которому мы дифференцируем, когда это нужно; а $f'(x)$ относится только к скалярным функциям и иногда называется нотацией Лагранжа.

Нас здесь не заботит существование производной функции, которое не гарантировано всюду даже для непрерывной функции и которое мы принимаем как данность. Мы коснёмся этого вопроса, лишь обсуждая производные негладких функций, таких как модуль $f(x) = \text{abs}(x)$ в точке 0, позже в главе 6.

Производные простых функций можно получить прямым применением определения; например, производные многочлена, логарифма или синуса должны быть знакомы: $\partial x^p = p x^{p-1}$, $\partial \log(x) = 1/x$ и $\partial \sin(x) = \cos(x)$.

Геометрически производную можно понимать как наклон касательной, проходящей через точку, или, что равносильно, как лучшее приближение первого порядка самой функции в этой точке, как показано на рисунке F.2.1. Это фундаментальный взгляд, потому что наклон прямой говорит нам, как функция ведёт себя в близкой окрестности: при положительном наклоне функция растёт справа и убывает слева, опять же на достаточно малом интервале, а при отрицательном наклоне верно обратное. Как мы увидим, это понимание распространяется и на векторные функции. На рисунке изображена функция $f(x) = x^2 - 1.5x$ вместе с её производными в двух разных точках.



Наклоны $x^2 - 1.5x$

Напомним некоторые важные свойства производных, которые распространяются на многомерный случай. Линейность: производная линейна, поэтому производная суммы — это сумма производных, $\partial[f(x) + g(x)] = f'(x) + g'(x)$. Правило произведения: $\partial[f(x)g(x)] = f'(x)g(x) + f(x)g'(x)$. Цепное правило: производная композиции функций получается перемножением соответствующих производных, $\partial[f(g(x))] = f'(g(x)) g'(x)$.

Вопросы

1. В нотации Эйнштейна, как она используется в книге, по каким индексам идёт суммирование?
 - a. По каждому индексу, который встречается дважды слева
 - b. По любому индексу, который стоит справа от знака равенства, но не слева
 - c. Только по индексу z
 - d. Ни по каким; суммы всегда нужно выписывать
2. Что вычисляет строка `einsum «i j k, i j k, в ничего»`?
 - a. Пакетное умножение матриц
 - b. Транспонирование
 - c. Обобщённое скалярное произведение: все индексы суммируются, остаётся одно число
 - d. Поэлементное произведение, сохранённое как тензор
3. Что положительная производная в точке говорит о функции поблизости?

- a. Функция растёт справа и убывает слева, на достаточно малом интервале
 - b. Функция там положительна
 - c. У функции там максимум
 - d. Функция поблизости постоянна
4. Что говорит цепное правило о производной $f(g(x))$?
- a. Она равна $f'(x) + g'(x)$
 - b. Она равна $f'(g(x))$ умножить на $g'(x)$
 - c. Она равна $f'(x)$ умножить на $g'(x)$
 - d. Она равна $f(g'(x))$

Ответы

1. **b)** Соглашение убирает знаки суммы: индекс, присутствующий справа, но отсутствующий слева, суммируется.
2. **c)** На стороне выхода ничего не остаётся, поэтому все три индекса суммируются: обобщённое скалярное произведение.
3. **a)** Производная — наклон касательной; положительный наклон означает, что локально функция поднимается справа и опускается слева.
4. **b)** Производная композиции — произведение производных, причём внешняя берётся в точке $g(x)$.

Разобранные примеры

Наклоны функции из примера книги Для $f(x) = x^2 - 1.5x$, функции с рисунка F.2.1, найдите производную и её значение при $x = 0$ и при $x = 1$. Скажите, растёт ли f или убывает около каждой точки.

Дифференцируйте почленно, используя линейность и правило для степеней: производная x^2 равна $2x$, а производная $1.5x$ равна 1.5 .

Итак, $f'(x) = 2x - 1.5$.

При $x = 0$: $f'(0) = -1.5$, наклон отрицательный, значит, f убывает около 0 .

При $x = 1$: $f'(1) = 2 - 1.5 = 0.5$, наклон положительный, значит, f растёт около 1 .

Ответ: $f'(x) = 2x - 1.5$; $f'(0) = -1.5$ (убывает), $f'(1) = 0.5$ (растёт).

Задачи

1. С помощью правила произведения продифференцируйте $f(x) = x \cdot \sin(x)$.

Подсказка: Возьмите первым множителем x , а вторым $\sin(x)$.

Ответ: $f'(x) = 1 \cdot \sin(x) + x \cdot \cos(x) = \sin(x) + x \cos(x)$.

2. С помощью цепного правила продифференцируйте $h(x) = \log(x^2)$ при $x > 0$ и упростите.

Подсказка: Внешняя функция — $\log$, внутренняя — x^2 .

Ответ: $h'(x) = (1/x^2) \cdot 2x = 2/x$.

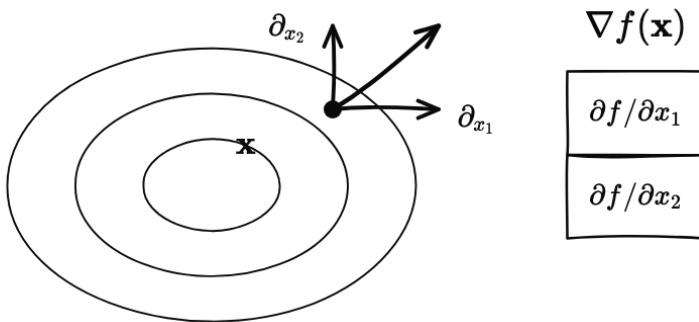
3. Напишите строку `einsum` для произведения матрицы на вектор $z = Wx$, где у W индексы i, j , а у x индекс j .

Подсказка: Какой индекс исчезает из выхода?

Ответ: `<i j, j, в i>`: j стоит на стороне входов, но не в выходе, поэтому по нему идёт суммирование, что даёт $z_i = \sum_j W_{ij} x_j$.

Градиенты, производные по направлению и якобианы

Рассмотрим теперь функцию $y = f(\mathbf{x})$, принимающую на вход вектор $\mathbf{x} \sim (d)$. Говорить здесь о бесконечно малых возмущениях бессмысленно, пока мы не укажем направление возмущения: если в скалярном случае у нас были только «влево» и «вправо», то здесь в евклидовом пространстве бесконечно много возможных направлений. В простейшем случае мы можем рассмотреть движение вдоль i -той оси, оставляя все остальные значения неизменными: $\partial_{x_i} f(\mathbf{x}) = \partial y / \partial x_i = \lim_{h \rightarrow 0} (f(\mathbf{x} + h \mathbf{e}_i) - f(\mathbf{x})) / h$. Здесь $\mathbf{e}_i \sim (d)$ — i -тый базисный вектор, i -тая строка единичной матрицы: его j -тый элемент равен 1, если $i = j$, и 0 иначе.



$$\mathbf{e}_i = [0, 1, 0]$$

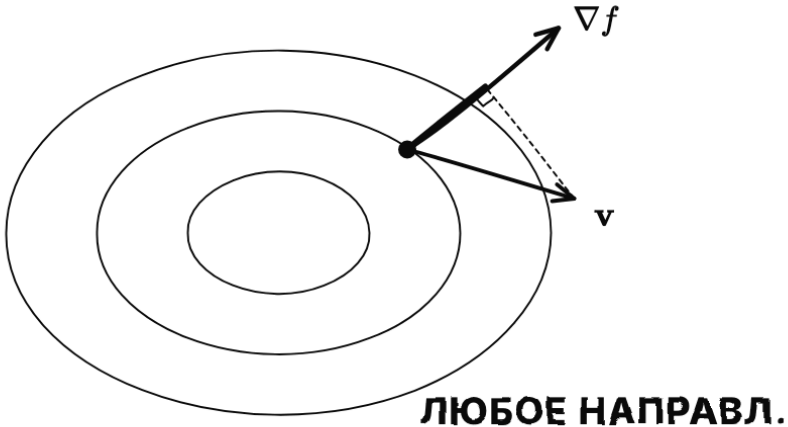
Частные производные образуют градиент

Уравнение E.2.17 называется частной производной. Сложив все частные производные в столпку, мы получаем d-мерный вектор, называемый градиентом функции.

Определение, градиент: градиент функции $y = f(\mathbf{x})$ — это вектор-столбец $\nabla f(\mathbf{x}) = \partial f(\mathbf{x})$, элементы которого — частные производные от $\partial_{x_1} f(\mathbf{x})$ до $\partial_{x_d} f(\mathbf{x})$.

Поскольку градиенты фундаментальны, мы используем особое обозначение $\nabla f(\mathbf{x})$, чтобы их выделить. А что со смещениями в произвольном направлении $\mathbf{v}$? В этом случае мы получаем производную по направлению: $D_{\mathbf{v}} f(\mathbf{x}) = \lim_{h \rightarrow 0} (f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})) / h$.

$$D_{\mathbf{v}}f(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})}{h} = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$$



Любое направление из градиента

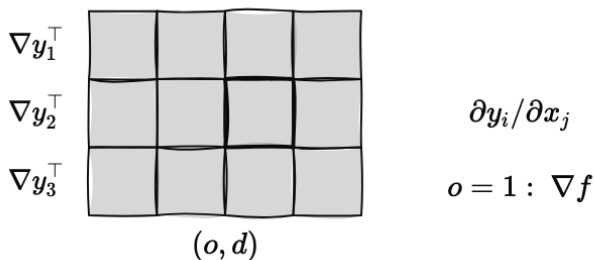
Движение в пространстве можно разложить на отдельные смещения вдоль каждой оси; поэтому легко доказать, что производная по направлению даётся скалярным произведением градиента на вектор смещения $\mathbf{v}$: $D_{\mathbf{v}}f(\mathbf{x}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle = \sum_i \partial_{x_i} f(\mathbf{x}) v_i$, где каждое v_i — смещение по i -той оси.

Следовательно, умения вычислять градиент функции достаточно, чтобы вычислить все возможные производные по направлению.

Якобианы. Рассмотрим теперь общий случай функции $y = f(\mathbf{x})$ с векторным входом $\mathbf{x} \sim (d)$, как раньше, и на этот раз с векторным выходом $y \sim (o)$. Как мы увидим, это самый общий случай, который нам нужно рассмотреть. Поскольку выходов у нас

больше одного, мы вычисляем градиент для каждого выхода, и их стопка даёт матрицу (o, d) , называемую якобианом f .

$$\mathbf{x} \sim (d) \rightarrow f \rightarrow \mathbf{y} \sim (o)$$



$$\partial[f \circ g] = \partial f \cdot \partial g$$



Якобиан: градиенты в стопке

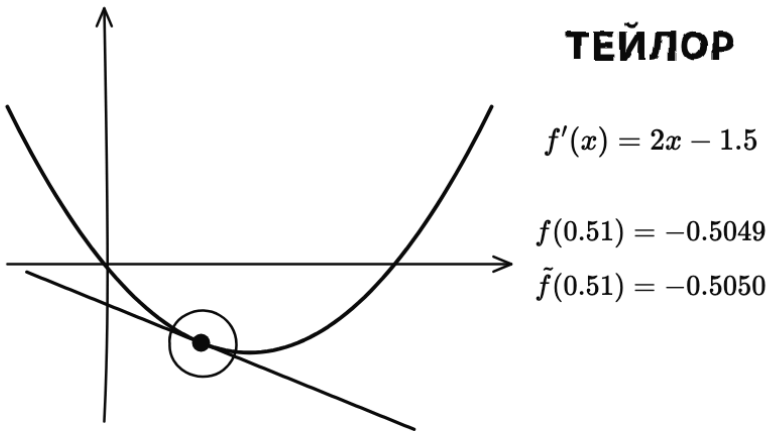
Определение, якобиан: матрица Якоби функции $\mathbf{y} = f(\mathbf{x})$, где $\mathbf{x} \sim (d)$ и $\mathbf{y} \sim (o)$, — это матрица $\partial f(\mathbf{x}) \sim (o, d)$, элемент которой в строке i и столбце j равен $\partial y_i / \partial x_j$, от $\partial y_1 / \partial x_1$ в левом верхнем углу до $\partial y_o / \partial x_d$ в правом нижнем.

При $o = 1$ мы получаем градиент, а при $d = o = 1$ — обычную производную. Якобианы наследуют все свойства производных: важно, что якобиан композиции функций — это теперь матричное произведение соответствующих отдельных

якобианов, $\partial[f(g(\mathbf{x}))] = [\partial f(\bullet)] \partial g(\mathbf{x})$, где первый якобиан вычисляется в точке $g(\mathbf{x}) \sim (\mathbf{h})$.

Как и в скалярном случае, градиенты и якобианы можно понимать как линейные функции, касательные в конкретной точке. В частности, градиент — лучшее «приближение первого порядка» в следующем смысле. Для точки $\mathbf{x}_0$ лучшее линейное приближение в бесконечно малой окрестности $f(\mathbf{x}_0)$ даётся $\tilde{f}(\mathbf{x}) = f(\mathbf{x}_0) + \langle \partial f(\mathbf{x}_0), \mathbf{x} - \mathbf{x}_0 \rangle$, где градиент играет роль наклона прямой, а $\mathbf{x} - \mathbf{x}_0$ — смещение от $\mathbf{x}_0$.

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}_0) + \langle \partial f(\mathbf{x}_0), \mathbf{x} - \mathbf{x}_0 \rangle$$



Касательная как приближение

Это называется теоремой Тейлора. Врезка С.2.6 и рисунок F.2.2 показывают её наглядно в скалярном случае $f(x) = x^2 - 1.5x$.

Врезка записывает функцию, её производную $2x - 1.5$, пока вычисленную вручную, и линеаризацию в точке $x = 0.5$. Для численной проверки: при $x = 0.51$ функция даёт -0.5049 , а линеаризация даёт -0.5050 .

Вопросы

1. Как получить производную f по любому направлению $\mathbf{v}$, если известен градиент?
 - a. Взять скалярное произведение градиента на $\mathbf{v}$
 - b. Умножить градиент только на длину $\mathbf{v}$
 - c. Взять наибольшую компоненту градиента
 - d. Нужно вычислять новый предел для каждого направления
2. Какова форма якобиана функции из векторов размера d в векторы размера o ?
 - a. (d, o)
 - b. (o, d)
 - c. (d)
 - d. (o)
3. Как выглядит цепное правило для якобианов?
 - a. Якобиан композиции — сумма якобианов
 - b. Это произведение Адамара якобианов
 - c. Это матричное произведение отдельных якобианов, причём внешний вычисляется в точке $g(\mathbf{x})$

- d. Оно верно только для скалярных функций
4. Почему в проверке из книги $f(0.51) = -0.5049$ и её линейная -0.5050 так близки?
- a. Потому что f линейна
 - b. Потому что линейная — лучшее приближение первого порядка около точки 0.5 , а 0.51 очень близко к ней
 - c. Потому что производная в точке 0.5 равна нулю
 - d. Потому что округление скрывает разницу

Ответы

1. **a)** Движение раскладывается по осям, поэтому производная по направлению равна $\langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$; градиента достаточно для любого направления.
2. **b)** На каждый из n выходов приходится одна строка-градиент с n компонентами, так что якобиан имеет размер n на n .
3. **c)** Якобианы наследуют свойства производных; для композиции они перемножаются как матрицы.
4. **b)** Теорема Тейлора говорит, что касательная — лучшее линейное приближение в малой окрестности; шаг 0.01 остаётся в этой окрестности.

Разобранные примеры

Линеаризация из книги в точке 0.5 Для $f(x) = x^2 - 1.5x$ постройте приближение первого порядка в точке $x = 0.5$ и сравните его с f при $x = 0.51$.

Вычислите функцию в точке: $f(0.5) = 0.25 - 0.75 = -0.5$.

Вычислите производную $2x - 1.5$ в точке: $2 \times 0.5 - 1.5 = -0.5$, наклон касательной.

Линеаризация: $\tilde{f}(x) = -0.5 + (-0.5)(x - 0.5)$; при 0.51 смещение равно 0.01, так что $\tilde{f}(0.51) = -0.5 - 0.005 = -0.505$.

Истинное значение $f(0.51) = 0.2601 - 0.765 = -0.5049$, что отличается лишь на 0.0001.

Ответ: $\tilde{f}(0.51) = -0.5050$ против $f(0.51) = -0.5049$.

Задачи

1. Для $f(x_1, x_2) = x_1^2 + 3x_1x_2$ найдите градиент в точке $(1, 2)$ и производную по направлению $v = (1, -1)$.

Подсказка: Продифференцируйте по каждой переменной, держа другую фиксированной, затем возьмите скалярное произведение с v .

Ответ: $\partial f / \partial x_1 = 2x_1 + 3x_2 = 2 + 6 = 8$ и $\partial f / \partial x_2 = 3x_1 = 3$, так что градиент равен $(8, 3)$. Вдоль v : $8 \times 1 + 3 \times (-1) = 5$.

2. Найдите якобиан $y = (x_1x_2, x_1 + x_2)$ в точке $(2, 3)$ и укажите его форму.

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

Ответ: Строка 1: $(\partial(x_1x_2)/\partial x_1, \partial(x_1x_2)/\partial x_2) = (x_2, x_1) = (3, 2)$. Строка 2: $(1, 1)$. Якобиан равен $[[3, 2], [1, 1]]$, форма $(2, 2)$.

Размерности якобианов и градиентный спуск

Закончим педантичным замечанием о размерности, которое пригодится в дальнейшем. Рассмотрим функцию $y = \mathbf{W}\mathbf{x}$. Если смотреть на неё как на функцию от $\mathbf{x}$, производная, как и раньше, — матрица (o, d) , и можно показать, что $\partial_{\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbf{W}$.

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

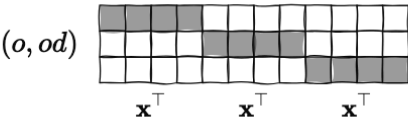


$$\partial_{\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbf{W}$$

(o, d)

$$\partial_{\mathbf{W}}[\mathbf{W}\mathbf{x}] \sim (o, o, d)$$

ПОВТОРЫ



ВЕКТОР-ЯКОБИАН

Якобианы $y = \mathbf{W}\mathbf{x}$

Рисунок F.2.2 показывает функцию $f(x) = x^2 - 1.5x$ и её приближение первого порядка в точке 0.5.

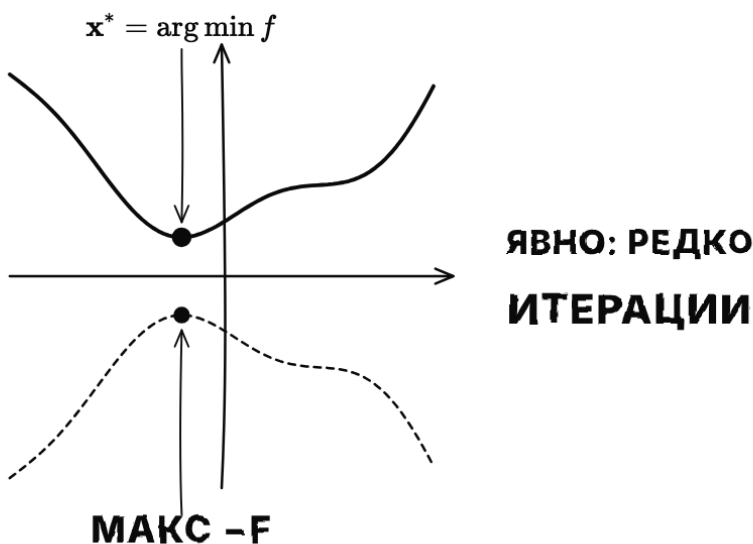
Если же смотреть на неё как на функцию от $\mathbf{W}$, вход сам является матрицей (o, d) , и «якобиан» в этом случае имеет форму (o, o, d) , как показывает следующая врезка. Однако мы всегда можем представить тождественную, изоморфную, функцию, принимающую на вход векторизованную версию $\mathbf{W}$, $\text{vect}(\mathbf{W}) \sim (od)$, и тогда якобиан будет матрицей формы (o, od) .

Во врезке якобиан выводится. Чтобы вычислить якобиан $\partial_{\mathbf{W}} \mathbf{W}x$, мы можем переписать выражение поэлементно как $y_i = \sum_j W_{ij} x_j$, откуда сразу находим, что $\partial y_i / \partial W_{ij} = x_j$. Заметьте, что чтобы материализовать якобиан явно, то есть сохранить его в памяти, нам понадобилось бы много повторяющихся значений. Как мы увидим в главе 6, этого можно избежать, потому что на практике нас интересует лишь применение якобиана к другому тензору.

Этот короткий пример поясняет, что мы имеем в виду, утверждая, что с точки зрения обозначений достаточно работать с векторными входами и выходами. Однако важно помнить об этом в главе 6, где мы будем использовать матричные якобианы для простоты обозначений, в частности чтобы избежать нагромождения индексов, но размеры этих якобианов могут «скрываться» внутри реальных форм входов и выходов, прежде всего размеров пакетов. В главе 6 мы увидим, что явного вычисления якобианов на практике можно избежать, рассматривая так называемые произведения вектора на якобиан. Это можно также формализовать, рассматривая якобианы как абстрактные линейные отображения.

Раздел 2.3, Градиентный спуск. Чтобы понять пользу доступа к градиентам, рассмотрим задачу минимизации произвольной

функции $f(x)$, где $x \sim (d)$: $x^* = \operatorname{argmin}_x f(x)$. Здесь, по аналогии с argmax , $\operatorname{argmin} f(x)$ обозначает операцию нахождения значения x , соответствующего наименьшему возможному значению $f(x)$. Мы предполагаем, что у функции один выход, то есть это однокритериальная оптимизация, и что область, в которой мы оптимизируем x , не ограничена.



В поисках низшей точки

В остальной части книги x будет кодировать параметры нашей модели, а f описывать качество самой модели на наших данных — постановка, называемая обучением с учителем, которую мы введём в следующей главе. Мы можем без потери общности рассматривать минимизацию вместо максимизации, поскольку минимизировать $f(x)$ равносильно тому, чтобы максимизировать

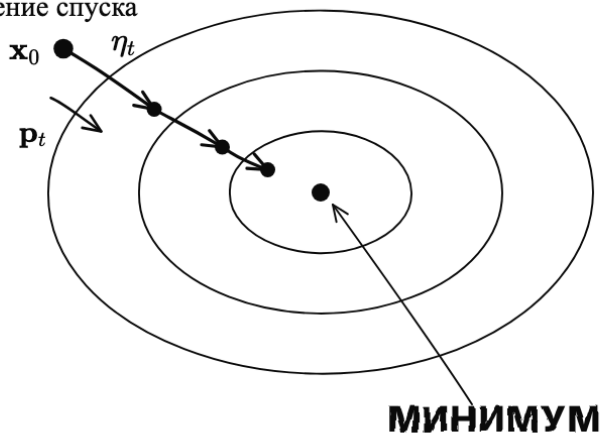
$-f(x)$, и наоборот; чтобы это представить, подумайте о функции одной переменной и отразите её относительно оси x , глядя, что происходит с её низшими точками.

В очень редких случаях нам удаётся выразить решение в замкнутом виде; один пример мы увидим в контексте оптимизации по методу наименьших квадратов в разделе 4.1.2. В общем же случае мы вынуждены прибегать к итеративным процедурам. Предположим, мы начинаем со случайного приближения x_0 и на каждой итерации делаем шаг, который разложим на его величину η_t , длину шага, и направление p_t : $x_t = x_{t-1} + \eta_t p_t$, то есть приближение на итерации t — это предыдущее приближение плюс смещение на итерации t .

$$x_t = x_{t-1} + \eta_t p_t$$

η_t : скорость обучения

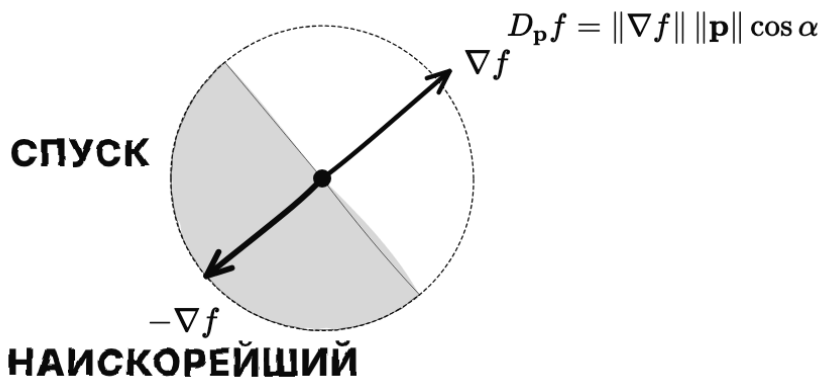
p_t : направление спуска



Шаг за шагом вниз

Мы называем η_t размером шага или, в терминологии машинного обучения, скоростью обучения, по причинам, которые станут ясны в следующей главе. Направление $\mathbf{p}_t$, для которого существует η_t такое, что $f(\mathbf{x}_t) \leq f(\mathbf{x}_{t-1})$, называется направлением спуска. Если мы можем выбирать направление спуска на каждой итерации и аккуратны в выборе размера шага, итеративный алгоритм из уравнения Е.2.26 сойдётся к минимуму, в смысле, который вскоре будет описан.

Для дифференцируемых функций мы можем точно описать все направления спуска с помощью производной по направлению из уравнения Е.2.20, поскольку их можно определить как направления, дающие отрицательное изменение относительно нашего предыдущего приближения $\mathbf{x}_{t-1}$: если $\mathbf{p}_t$ — направление спуска, то $D_{\mathbf{p}t} f(\mathbf{x}_{t-1}) \leq 0$.



Какие направления ведут вниз

Используя то, что мы узнали в разделе 2.2, и определение скалярного произведения через косинусное сходство из уравнения Е.2.2, получаем $D_{\mathbf{p}_t} f(\mathbf{x}_{t-1}) = \langle \nabla f(\mathbf{x}_{t-1}), \mathbf{p}_t \rangle = \|\nabla f(\mathbf{x}_{t-1})\| \|\mathbf{p}_t\| \cos(\alpha)$, где α — угол между $\mathbf{p}_t$ и $\nabla f(\mathbf{x}_{t-1})$.

Если взглянуть на выражение справа, первый множитель — константа относительно $\mathbf{p}_t$. Поскольку мы договорились, что $\mathbf{p}_t$ кодирует только направление движения, мы можем спокойно ограничить его условием $\|\mathbf{p}_t\| = 1$, превратив второй множитель в ещё одну константу. Следовательно, по свойствам косинуса мы заключаем, что любое $\mathbf{p}_t$, угол которого с $\nabla f(\mathbf{x}_{t-1})$ лежит между $\pi/2$ и $3\pi/2$, — направление спуска. Среди них направление $\mathbf{p}_t = -\nabla f(\mathbf{x}_{t-1})$, с углом π , имеет наименьшую возможную

производную по направлению, и мы называем его направлением наискорейшего спуска.

Соединив это наблюдение с итеративной процедурой из уравнения Е.2.26, мы получаем алгоритм минимизации любой дифференцируемой функции, который называем наискорейшим градиентным спуском.

Определение, наискорейший градиентный спуск: для дифференцируемой функции $f(\mathbf{x})$, начальной точки $\mathbf{x}_0$ и последовательности размеров шага η_t градиентный спуск идёт так: $\mathbf{x}_t = \mathbf{x}_{t-1} - \eta_t \nabla f(\mathbf{x}_{t-1})$.

Нас не будет занимать задача подбора подходящего размера шага, который мы просто будем считать «достаточно малым», чтобы итерация градиентного спуска давала уменьшение f . В следующем разделе мы сосредоточимся на том, какие точки получаются при запуске градиентного спуска из произвольной инициализации. Заметьте, что градиентный спуск настолько эффективен, насколько эффективна процедура, которой мы вычисляем градиент: общий алгоритм для этого мы введём в главе 6.

Вопросы

1. Для $\mathbf{y} = \mathbf{W}\mathbf{x}$, рассматриваемой как функция от $\mathbf{W} \sim (o, d)$, какую форму имеет якобиан?
 - a. (o, d)
 - b. (o, o, d) , или (o, od) , если $\mathbf{W}$ расплющить

- c. (d, o)
 - d. (o)
2. Какое направление $\mathbf{p}_t$ имеет наименьшую производную по направлению, наискорейший спуск?
- a. Сам градиент
 - b. Любое направление под прямым углом к градиенту
 - c. Минус градиент, под углом π к нему
 - d. Любой единичный вектор
3. Почему книга может без потери общности изучать только минимизацию?
- a. Потому что максимизация невозможна для дифференцируемых функций
 - b. Потому что минимизировать f — то же самое, что максимизировать $-f$
 - c. Потому что у всех функций один минимум
 - d. Потому что градиенты указывают только вниз
4. Почему книга избегает материализации якобианов, таких как $\partial_{\mathbf{W}} \mathbf{W}\mathbf{x}$, в памяти?
- a. В них было бы много повторяющихся значений, а на практике нужно лишь их применение к другому тензору
 - b. Их нельзя вычислить
 - c. Они всегда равны нулю
 - d. Они определены только для скаляров

Ответы

1. б) Вход сам является матрицей o на d , поэтому якобиан имеет форму (o, o, d) ; расплющив W в вектор из od элементов, получаем матрицу (o, od) .

2. с) При фиксированных нормах производная по направлению пропорциональна $\cos(\alpha)$, который минимален, -1 , при $\alpha = \pi$: отрицательный градиент.

3. б) Отражение функции относительно оси x превращает её высшие точки в низшие, так что одну задачу можно превратить в другую.

4. а) Элементы якобиана — просто повторяющиеся x_j , и глава 6 показывает, что нужны только произведения с ним, произведения вектора на якобиан.

Разобранные примеры

Два шага градиентного спуска Минимизируйте $f(x) = x^2$ градиентным спуском из $x_0 = 3$ с постоянным размером шага $\eta = 0.1$.
0.1. Вычислите два шага и значение f в каждой точке.

Градиент x^2 равен $2x$, а обновление: $x_t = x_{t-1} - \eta \cdot 2x_{t-1}$.

Шаг 1: градиент в точке 3 равен 6, так что $x_1 = 3 - 0.1 \times 6 = 2.4$.

Шаг 2: градиент в точке 2.4 равен 4.8, так что $x_2 = 2.4 - 0.1 \times 4.8 = 1.92$.

Значения функции падают с $f(3) = 9$ до $f(2.4) = 5.76$ и до $f(1.92) = 3.6864$, так что каждый шаг — спуск.

Ответ: $x_1 = 2.4$, $x_2 = 1.92$; f идёт $9 \rightarrow 5.76 \rightarrow 3.6864$.

Задачи

1. В некоторой точке градиент f равен $(2, -1)$. Является ли $p = (1, 1)$ направлением спуска? А $p = (-1, 0)$?

Подсказка: Вычислите производную по направлению $\langle \nabla f, p \rangle$ и проверьте её знак.

Ответ: Для $(1, 1)$: $2 \times 1 + (-1) \times 1 = 1 > 0$, так что это не направление спуска. Для $(-1, 0)$: $2 \times (-1) + 0 = -2 < 0$, так что это направление спуска.

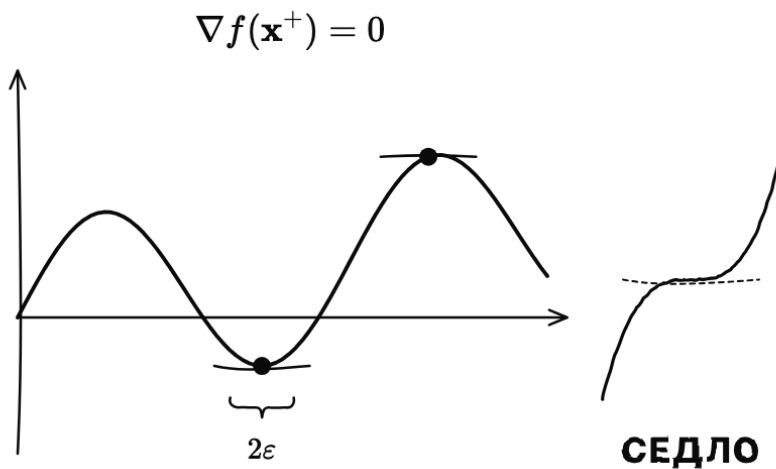
2. Для $y = Wx$ с W формы $(3, 4)$ укажите форму якобиана по x , по W и по расплющенной W .

Подсказка: По x это сама W .

Ответ: По x : $(3, 4)$, равен W . По W : $(3, 3, 4)$. По $\text{vect}(W)$, вектору из 12 элементов: $(3, 12)$.

Сходимость градиентного спуска

Обсуждая сходимость градиентного спуска, нужно уточнить, что мы понимаем под «минимизатором» функции. Если сходимость вас не волнует и вы доверяете градиентному спуску, без колебаний переходите к следующему разделу.



ЗАВИСИТ ОТ СТАРТА

Минимумы, максимумы и сёдла

Определение, минимум: локальный минимум $f(\mathbf{x})$ — это точка $\mathbf{x}^+$ такая, что для некоторого $\varepsilon > 0$ выполняется $f(\mathbf{x}^+) \leq f(\mathbf{x})$ для всех $\mathbf{x}$ с $\|\mathbf{x} - \mathbf{x}^+\| < \varepsilon$, то есть для каждой точки шара радиуса ε с центром в $\mathbf{x}^+$.

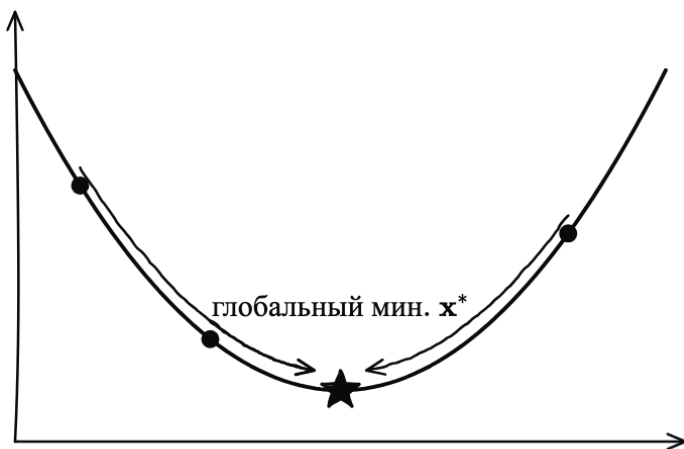
Словами: значение $f(x^+)$ — минимум, если рассматривать достаточно малую окрестность x^+ . Интуитивно, в такой точке наклон касательной будет равен 0, а градиент везде в остальной окрестности x^+ будет указывать вверх. Первую идею мы можем формализовать понятием стационарных точек.

Определение, стационарные точки: точка x^+ называется стационарной точкой $f(x)$, если $\nabla f(x^+) = 0$.

Стационарные точки не сводятся к минимумам: это могут быть максимумы, то есть минимумы $-f(x)$, или седловые точки — точки перегиба, где кривизна функции меняется, как в примере на рисунке F.2.3. В общем случае, без каких-либо ограничений на f , для градиентного спуска можно доказать лишь сходимость к некоторой стационарной точке, зависящей от его инициализации.

Рисунок — простой пример седловой точки; попробуйте представить касательную в этой точке, чтобы увидеть, что она и впрямь стационарна.

Можно ли сделать лучше? Представьте параболу: в этом случае у функции нет седловых точек и есть лишь один минимум. Этот минимум ещё и особенный в том смысле, что в этой точке функция достигает наименьшего возможного значения на всей области определения: мы говорим, что это глобальный минимум.



У параболы один глобальный минимум

Определение, глобальный минимум: глобальный минимум $f(\mathbf{x})$ — это точка $\mathbf{x}^*$ такая, что $f(\mathbf{x}^*) \leq f(\mathbf{x})$ для любого возможного входа $\mathbf{x}$.

Интуитивно, градиентный спуск на параболе сойдётся к этому глобальному минимуму из любой возможной инициализации, потому что все градиенты будут указывать на него. Эту идею можно обобщить понятием выпуклости функции. Определений выпуклости много; мы выбираем приведённое ниже для простоты изложения.

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$$



Выпуклая: ниже любой хорды

Определение, выпуклая функция: функция $f(x)$ выпукла, если для любых двух точек x_1 и x_2 и любого $\alpha \in [0, 1]$ выполняется $f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$.

Левая часть уравнения Е.2.28 — это значение f в любой точке внутри интервала от x_1 до x_2 , а правая — соответствующее значение на отрезке, соединяющем $f(x_1)$ и $f(x_2)$. Если функция всегда лежит ниже прямой, соединяющей любые две точки, она выпукла; например, парабола ветвями вверх выпукла.

Выпуклость характеризует простоту оптимизации функции в следующем смысле. Для произвольной невыпуклой функции градиентный спуск сходится к стационарной точке, и ничего

больше сказать нельзя, если не смотреть на производные высших порядков, производные производных. Для выпуклой функции градиентный спуск сойдётся к глобальному минимуму независимо от инициализации. Если неравенство в уравнении E.2.28 выполняется строго, что называется строгой выпуклостью, глобальный минимизатор будет ещё и единственным.

Это жёсткое свойство: чтобы найти глобальный минимум в невыпуклой задаче градиентным спуском, единственный выход — запустить оптимизатор бесконечно много раз из каждой возможной инициализации, что превращает задачу в NP-трудную.

У этого обсуждения большое историческое значение. Как мы увидим в главе 5, любая нетривиальная модель невыпукла, то есть у её задачи оптимизации может быть несколько стационарных точек. Это контрастирует с альтернативными алгоритмами обучения с учителем, такими как метод опорных векторов, которые сохраняют нелинейность, допуская при этом выпуклую оптимизацию. Интересно, что сложные дифференцируемые модели, по-видимому, хорошо работают даже при этом ограничении, в том смысле, что их оптимизация, начатая с разумной инициализации, сходится к точкам с хорошим эмпирическим качеством.

Вопросы

1. Без каких-либо ограничений на f , к чему, как можно доказать, приходит градиентный спуск?

- a. Всегда к глобальному минимуму
 - b. К некоторой стационарной точке, зависящей от инициализации
 - c. Всегда к локальному максимуму
 - d. Ни к чему
2. Что выпуклость гарантирует градиентному спуску?
- a. Сходимость к глобальному минимуму при любой инициализации
 - b. Сходимость ровно за один шаг
 - c. Сходимость к седловой точке
 - d. Что градиент никогда не равен нулю
3. Почему невыпуклость нейронных сетей исторически значима?
- a. Из-за неё нейронные сети стало невозможно обучать
 - b. Она контрастирует с методами вроде опорных векторов, сохраняющими нелинейность при выпуклой оптимизации, и всё же сети приходят к хорошим точкам из разумных стартов
 - c. Она доказывает, что сети всегда находят глобальные минимумы
 - d. Она означает, что сетям не нужны градиенты
4. Что такое седловая точка?
- a. Глобальный минимум
 - b. Стационарная точка, где кривизна функции меняется, ни минимум, ни максимум
 - c. Точка, где градиент очень велик

д. Начальная точка градиентного спуска

Ответы

1. **б)** Стационарные точки включают минимумы, максимумы и седловые точки; для произвольной f градиентный спуск приходит лишь к одной из них, в зависимости от того, откуда стартует.
2. **а)** Для выпуклой функции градиентный спуск сходится к глобальному минимуму независимо от инициализации.
3. **б)** Любая нетривиальная модель невыпукла, в отличие от SVM; и всё же дифференцируемые модели, разумно инициализированные, сходятся к точкам, которые хорошо работают.
4. **б)** Седловые точки стационарны, градиент равен нулю, но это точки перегиба, где меняется кривизна.

Разобранные примеры

Проверка неравенства выпуклости Проверьте неравенство выпуклости для $f(x) = x^2$ при $x_1 = 0$, $x_2 = 2$ и $\alpha = 0.5$.

Точка внутри интервала: $\alpha x_1 + (1 - \alpha) x_2 = 0.5 \times 0 + 0.5 \times 2 = 1$.

Левая часть — функция в этой точке: $f(1) = 1$.

Правая часть — прямая между концами: $0.5 \times f(0) + 0.5 \times f(2) = 0.5 \times 0 + 0.5 \times 4 = 2$.

Поскольку $1 \leq 2$, функция в этой точке лежит ниже отрезка, как и требует выпуклость.

Ответ: $1 \leq 2$: неравенство выполняется.

Задачи

1. Покажите при $x_1 = -1$, $x_2 = 1$ и $\alpha = 0.5$, что $f(x) = -x^2$ не выпукла.

Подсказка: Вычислите обе части неравенства выпуклости.

Ответ: Середина — 0, и $f(0) = 0$. Отрезок даёт $0.5 \times f(-1) + 0.5 \times f(1) = 0.5 \times (-1) + 0.5 \times (-1) = -1$. Поскольку 0 не ≤ -1 , неравенство нарушается, так что $-x^2$ не выпукла.

2. У функции $f(x) = x^3$ производная $f'(0) = 0$. Является ли $x = 0$ минимумом? Какого рода это стационарная точка?

Подсказка: Посмотрите на значения f чуть левее и чуть правее 0.

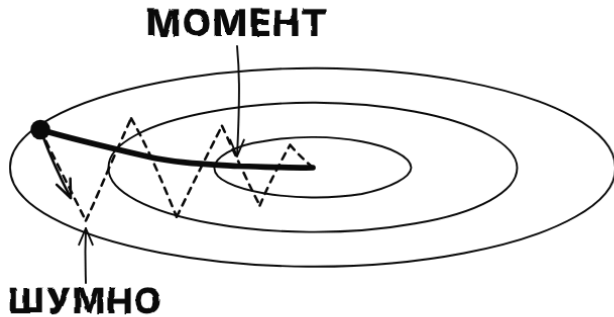
Ответ: Нет. f отрицательна чуть левее 0 и положительна чуть правее, так что 0 — ни минимум, ни максимум: это перегиб, стационарная точка седлового типа, где меняется кривизна.

Ускорение градиентного спуска

Отрицательный градиент описывает направление наискорейшего спуска, но лишь в бесконечно малой окрестности точки. Как мы увидим в главе 5, где введём стохастическую оптимизацию, эти направления могут быть крайне шумными, особенно при работе с большими моделями. Разработано множество приёмов, ускоряющих сходимость алгоритма оптимизации за счёт выбора лучших направлений спуска. По вычислительным причинам нас особенно интересуют методы, не требующие производных высших порядков, таких как гессиан, или многократных вызовов функции.

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

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

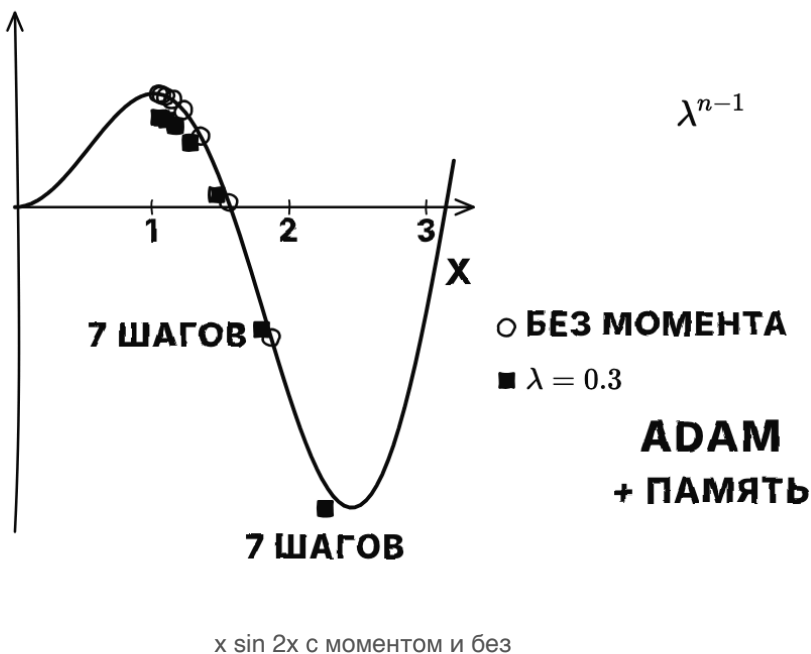


Момент сглаживает путь

Здесь мы опишем один такой приём, момент, а за более широким введением отсылаем к другим источникам, таким как пост Себастьяна Рудера в блоге 2016 года. Если представить градиентный спуск как шарик, «скатывающийся с холма», движение довольно хаотично, потому что каждый градиент может указывать в совершенно другом направлении; на деле при идеальном выборе размера шага и выпуклой функции потерь любые два градиента на соседних итерациях будут ортогональны. Мы можем сгладить это поведение, введя член «момента», который сохраняет часть направления с предыдущей градиентной итерации: $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1}$, наискорейший спуск плюс член момента, а затем $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{g}_t$.

Здесь мы инициализируем $\mathbf{g}_0 = \mathbf{0}$; рисунок F.2.4 показывает пример. Коэффициент λ определяет, насколько гасится предыдущий член. В самом деле, раскрыв два члена, получаем $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) - \lambda \eta_t \nabla f(\mathbf{x}_{t-2}) + \lambda^2 \mathbf{g}_{t-2}$.

Рисунок сравнивает градиентный спуск с моментом и без него при минимизации функции $x \sin(2x)$, начиная с $x = 1 + \epsilon$, при $\lambda = 0.3$.



Обобщая, итерация в момент времени $t - n$ гасится множителем λ^{n-1} . Можно показать, что момент ускоряет обучение, сглаживая траекторию оптимизации. Другой распространённый приём — адаптация размера шага для каждого параметра по величине

градиентов. Распространённый алгоритм оптимизации, сочетающий несколько таких идей, — Adam. Одно из преимуществ Adam в том, что он оказывается довольно устойчив к выбору своих гиперпараметров, где гиперпараметр — это параметр, выбираемый пользователем, а не выучиваемый градиентным спуском, причём значения по умолчанию в большинстве фреймворков — хорошая отправная точка в большинстве случаев. Разработка новых оптимизаторов, способных «сместить» Adam или его вариации, такие как AdamW, с места фактически стандартного оптимизатора в глубоком обучении остаётся открытой исследовательской задачей, и недавние работы посвящены проектированию специальных оптимизаторов для нейронных сетей из первых принципов.

Недостатком ускоренных алгоритмов оптимизации может быть повышенная потребность в памяти: например, момент требует хранить в памяти предыдущую градиентную итерацию, удваивая объём, нужный алгоритму оптимизации, хотя в большинстве случаев самый весомый фактор — память, необходимая для вычисления градиента, как мы увидим в разделе 6.3.

От теории к практике. Врезка об упражнениях объясняет, что в этой книге нет классических упражнений в конце глав, которых хватает во многих существующих учебниках. Вместо этого я предлагаю путь самостоятельного обучения, который поможет вам исследовать два фреймворка, JAX и PyTorch, по мере продвижения по книге. Решения упражнений будут опубликованы на сайте книги. Эти разделы полны ссылок на материалы в сети, которые к моменту, когда вы их будете искать, могли устареть или переехать.

Вопросы

1. Что добавляет член момента к простому градиентному спуску?
 - a. Вторую производную функции
 - b. Долю λ предыдущего шага, которая сглаживает хаотичный путь
 - c. Случайное возмущение на каждом шаге
 - d. Только больший размер шага
2. Почему путь простого градиентного спуска называют хаотичным?
 - a. Потому что каждый градиент может указывать в совершенно другом направлении, чем предыдущий
 - b. Потому что размер шага случаен
 - c. Потому что он использует вторые производные
 - d. Потому что он всегда проскакивает минимум
3. Какова цена использования момента?
 - a. Его нельзя применять к выпуклым функциям
 - b. Ему нужен гессиан
 - c. Он должен хранить предыдущий шаг, удваивая память оптимизатора
 - d. Ему нужно несколько вызовов функции на шаг
4. Почему Adam — популярный оптимизатор по умолчанию?
 - a. Он довольно устойчив к выбору своих гиперпараметров, и значения по умолчанию обычно работают

- b. Он находит глобальный минимум любой функции
- c. Ему не нужны градиенты
- d. Он не использует память

Ответы

1. **b)** Момент сохраняет часть направления с предыдущей итерации, λg_{t-1} , так что шарик, скатывающийся с холма, движется менее хаотично.
2. **a)** Каждый градиент может указывать куда-то ещё; при идеальном размере шага на выпуклой функции потеря соседние градиенты даже ортогональны.
3. **c)** Предыдущую градиентную итерацию нужно держать в памяти, что удваивает объём, нужный самому алгоритму оптимизации.
4. **a)** Adam сочетает несколько идей и оказывается устойчив к своим гиперпараметрам, а значения по умолчанию во фреймворках — хорошая отправная точка.

Разобранные примеры

Два шага с моментом Минимизируйте $f(x) = x^2$ из $x_0 = 3$ при $\eta = 0.1$, $\lambda = 0.5$ и $g_0 = 0$, используя момент. Сравните x_2 с простым градиентным спуском, который даёт 1.92 после двух шагов.

Градиент x^2 равен $2x$; обновления: $g_t = -\eta \nabla f(x_{t-1}) + \lambda g_{t-1}$ и $x_t = x_{t-1} + g_t$.

Шаг 1: градиент в точке 3 равен 6, так что $g_1 = -0.6 + 0.5 \times 0 = -0.6$ и $x_1 = 3 - 0.6 = 2.4$.

Шаг 2: градиент в точке 2.4 равен 4.8, так что $g_2 = -0.48 + 0.5 \times (-0.6) = -0.78$ и $x_2 = 2.4 - 0.78 = 1.62$.

Момент перенёс часть первого шага дальше, так что после двух шагов он ближе к минимуму в 0, чем простой спуск.

Ответ: $x_1 = 2.4$ и $x_2 = 1.62$ с моментом против 1.92 без него.

Задачи

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

Подсказка: Момент хранит один предыдущий шаг на каждый параметр.

Ответ: Параметры занимают 4 миллиарда байт, около 4 ГБ. Момент хранит ещё одно значение на параметр, так что добавляет ещё около 4 ГБ, удваивая долю оптимизатора.

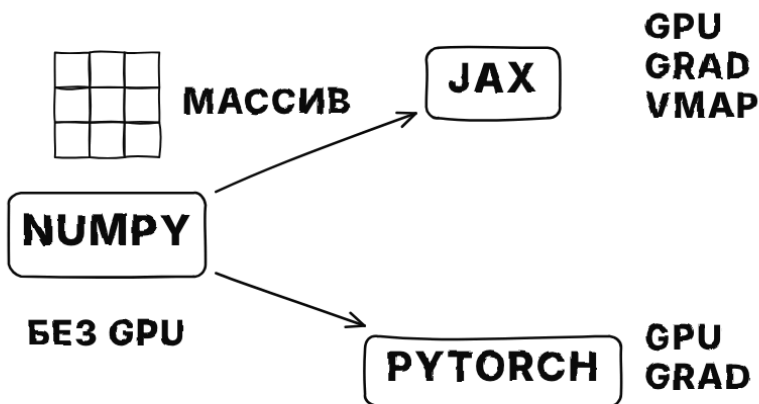
2. При $\lambda = 0.3$ чему равно λ^3 и что, по правилу гашения из книги, оно измеряет?

Подсказка: Итерация в момент времени $t - n$ гасится множителем λ^{n-1} .

Ответ: $\lambda^3 = 0.027$. По правилу λ^{n-1} это множитель гашения итерации в момент времени $t - 4$, которая, стало быть, всё ещё вносит вклад, но лишь примерно на 3 процента от полной силы.

От теории к практике, и что такое набор данных

Начнём с основ. Отправная точка для любого проектировщика дифференцируемых моделей — внимательное изучение NumPy. NumPy реализует общий набор функций для работы с многомерными массивами, которые мы в книге называем тензорами, а также функции для индексации и преобразования их содержимого. Подробнее можно прочитать в кратком руководстве библиотеки. Вы должны уверенно обращаться с массивами в NumPy, прежде всего с их индексацией: репозиторий `numpy-100` Николя Ружье предлагает хорошую, неспешную серию упражнений для проверки знаний.



NumPy и его наследники

Переход к реалистичному фреймворку. Несмотря на своё влияние, NumPy ограничен в поддержке параллельного оборудования, такого как GPU, если не использовать дополнительные библиотеки, и отсутствием автоматического дифференцирования, которое вводится в главе 6. JAX воспроизводит интерфейс NumPy, добавляя расширенную поддержку оборудования, автоматическое вычисление градиентов и дополнительные преобразования, такие как векторизованное отображение, vmap. Фреймворки вроде PyTorch тоже реализуют в своей основе интерфейс в духе NumPy, но вносят небольшие изменения в названия и функциональность и добавляют высокоуровневые утилиты для построения дифференцируемых моделей. Не торопясь просмотрите

документацию типа массива в JAX и типа тензора в PyTorch, чтобы понять, как много у них общего с NumPy. Пока можно не обращать внимания на высокоуровневые модули, такие как модуль нейронных сетей PyTorch. О том, как устроены эти фреймворки, мы подробнее поговорим в главе 6, после того как введём их механизм вычисления градиентов.

Реализация алгоритма градиентного спуска. Чтобы освоить все три фреймворка, NumPy, JAX и PyTorch, я предлагаю повторить упражнение ниже три раза; каждый вариант займёт лишь несколько минут, если вы знаете синтаксис. Рассмотрим двумерную функцию $f(\mathbf{x})$, где $\mathbf{x} \sim (2)$, а областью возьмём $[0, 10]$: $f(\mathbf{x}) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$. Я попросил ChatGPT сгенерировать приятную функцию с несколькими минимумами и максимумами; больше ничего в книге не создано языковой моделью, и мне кажется, такая оговорка становится важной.

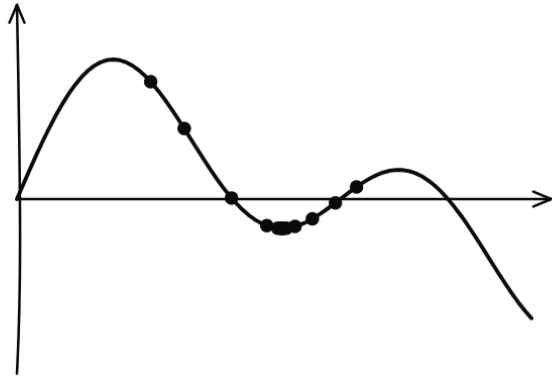
$$f(x) = \sin x + \sin 0.5x$$

1 F(X)

2 ГРАДИЕНТ

3 СПУСК

4 МОМЕНТ

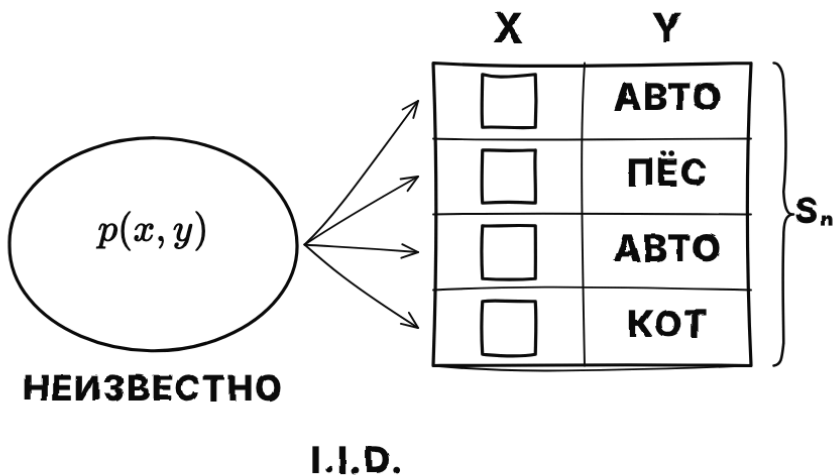


Градиентный спуск на бугристой функции

Прежде чем двигаться дальше по книге, повторите это для каждого фреймворка. Во-первых, реализуйте функцию векторизованно: получив матрицу $\mathbf{X} \sim (n, 2)$ из n входов, она должна вернуть вектор $f(\mathbf{X}) \sim (n)$, где $[f(\mathbf{X})]_i = f(X_i)$. Во-вторых, реализуйте ещё одну функцию для вычисления её градиента, записанного вручную, поскольку автоматического дифференцирования мы ещё не касались. В-третьих, напишите простую процедуру градиентного спуска и визуализируйте пути, которые проходит процесс оптимизации из нескольких начальных точек. В-четвёртых, попробуйте добавить член момента и визуализировать норму градиентов, которая должна сходиться к нулю по мере того, как алгоритм движется к стационарной точке.

Если для упражнения вы используете JAX или PyTorch, третий пункт — хорошее место, чтобы поэкспериментировать с `vmmap` для векторизации функции.

Глава 3, Наборы данных и функции потерь. Коротко: эта глава формализует сценарий обучения с учителем. Мы вводим понятия наборов данных, функций потерь, минимизации эмпирического риска и основные предположения, принимаемые в обучении с учителем. В завершение мы даём вероятностную формулировку обучения с учителем, построенную на понятии максимального правдоподобия. Эта короткая глава служит каркасом для остальной части книги.



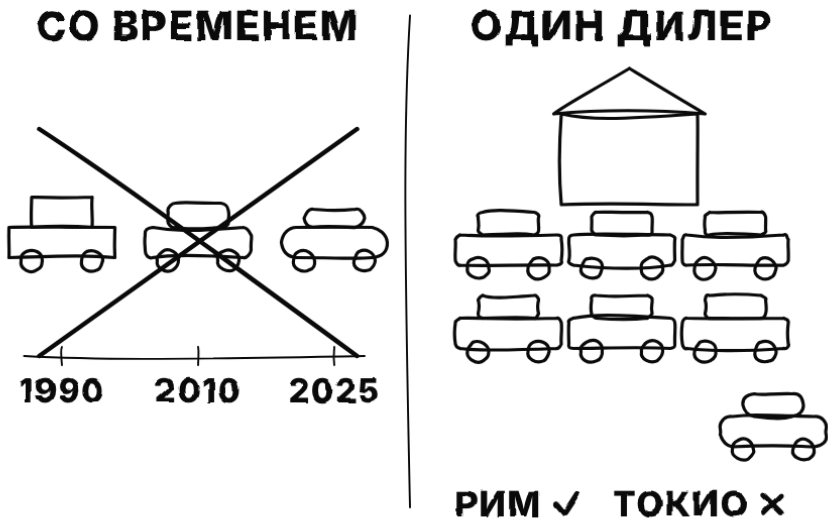
Набор данных из пар

Раздел 3.1, Что такое набор данных? Мы рассматриваем сценарий, в котором вручную запрограммировать некоторую функцию невозможно, например распознавание объектов на реальных изображениях, но собрать примеры желаемого поведения достаточно легко. Таких примеров множество, от распознавания речи до навигации роботов. Формализуем эту идею следующим определением.

Определение, набор данных: размеченный набор данных $\mathcal{S}_n$ размера n — это множество из n пар, $\mathcal{S}_n = (x_1, y_1), \dots, (x_n, y_n)$, где каждая пара (x_i, y_i) — пример связи «вход — выход», которую мы хотим смоделировать. Далее мы предполагаем, что каждый пример — независимая и одинаково распределённая, i.i.d., выборка из некоторого неизвестного, и непознаваемого, распределения вероятностей $p(x, y)$.

Загляните в приложение, если, прочитав определение, вы захотите освежить теорию вероятностей. Последнее предположение выглядит техническим, но оно нужно, чтобы связь, которую мы пытаемся смоделировать, была осмысленной. В частности, одинаковая распределённость выборок означает, что мы пытаемся приблизить нечто достаточно стабильное и неизменное во времени. В качестве характерного примера рассмотрим задачу сбора набора данных для распознавания моделей автомобилей по фотографиям. Это предположение будет выполнено, если мы соберём снимки за короткий промежуток времени, но нарушится, если мы соберём снимки за последние несколько десятилетий, поскольку модели автомобилей со временем менялись. В последнем случае обучение и внедрение модели на этом наборе данных

провалится, так как она не сможет распознавать новые модели или будет работать хуже возможного.



Когда i.i.d. нарушается

Аналогично, независимость выборок означает, что в сборе нашего набора данных нет смещения и что он достаточно репрезентативен для всего распределения. Возвращаясь к предыдущему примеру: собирать снимки рядом с дилерским центром Tesla будет неправильно, поскольку мы наберём избыток снимков определённого типа, упустив снимки других марок и моделей. Заметьте, что справедливость этих предположений зависит от контекста: набор данных об автомобилях, собранный в Италии, может подойти для внедрения нашей модели в Риме или Милане, но не подойти для

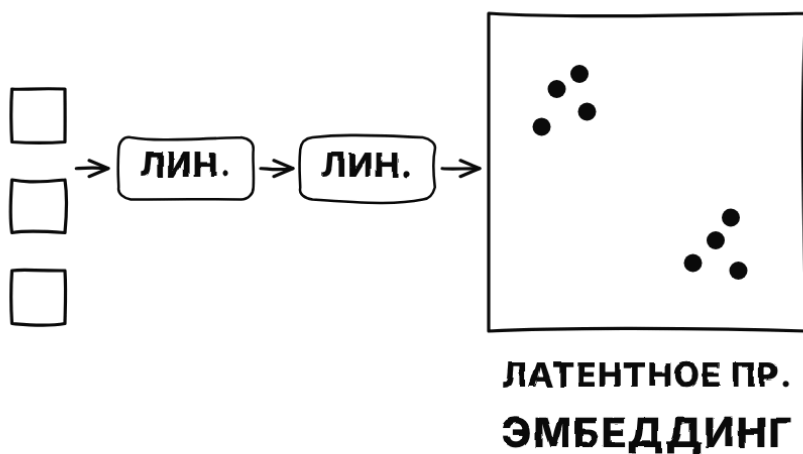
внедрения в Токио или на Тайване. Предположение i.i.d. всегда следует тщательно проверять, чтобы убедиться, что мы применяем инструменты обучения с учителем в подходящем сценарии. Интересно, что современные LLM обучаются на столь обширных распределениях данных, что даже понимание того, какие задачи действительно внутри распределения, а какие вне его, и насколько модели способны обобщать, размывается.



Сдвиг домена со временем

Врезка добавляет подробностей о свойстве i.i.d.. Важно, что обеспечение свойства i.i.d. — не одноразовый процесс, и его нужно проверять постоянно на протяжении всей жизни модели. В случае классификации автомобилей, если за этим не следить, малозаметные изменения распределения автомобилей со

временем ухудшат качество модели машинного обучения — пример сдвига домена. Другой пример: рекомендательная система изменит то, как пользователи взаимодействуют с некоторым приложением, поскольку они начнут реагировать на подсказки самой рекомендательной системы. Это создаёт петли обратной связи, требующие постоянной переоценки качества системы и приложения.



Петли и эмбединги

Рисунок показывает, что дифференцируемые модели обрабатывают данные, последовательно преобразуя их операциями линейной алгебры. Во многих случаях после оптимизации этих программ внутренние представления входных данных в модели, которую мы называем предобученной

моделью, обладают геометрическими свойствами: например, семантически похожие изображения проецируются в близкие точки этого «латентного» пространства. Преобразование данных из неметрического пространства, исходных изображений, в метрическое пространство называется эмбедингом данных.

Вопросы

1. Что JAX добавляет поверх интерфейса NumPy?
 - a. Ничего; это переименованная копия
 - b. Расширенную поддержку оборудования, автоматические градиенты и преобразования, такие как vmap
 - c. Только модуль нейронных сетей
 - d. Другое правило индексации, начинающееся с единицы
2. Почему набор фотографий автомобилей, собранный за несколько десятилетий, нарушит предположение i.i.d.?
 - a. Потому что старые фотографии хуже по разрешению
 - b. Потому что модели автомобилей меняются со временем, так что выборки не одинаково распределены
 - c. Потому что фотографий было бы слишком много
 - d. Потому что фотографии не были бы независимы друг от друга по освещению
3. Почему сбор фотографий рядом с дилерским центром Tesla — проблема?
 - a. Это смещает сбор в сторону одного типа автомобилей, так что выборки не представительны: нарушается часть о

независимости

- b. Tesla трудно распознать
 - c. Это нарушает только часть об одинаковой распределённости
 - d. Из-за этого набор данных слишком мал
4. Почему свойство i.i.d. нужно проверять на протяжении всей жизни модели?
- a. Потому что модели забывают то, чему обучились
 - b. Потому что распределения дрейфуют, например из-за сдвига домена или петель обратной связи, когда рекомендательная система меняет поведение пользователей
 - c. Потому что фреймворки меняют значения по умолчанию
 - d. Потому что наборы данных удаляются после обучения

Ответы

1. **b)** JAX воспроизводит интерфейс NumPy, добавляя поддержку оборудования, автоматическое вычисление градиентов и преобразования вроде векторизованного отображения.
2. **b)** Одинаковая распределённость означает, что изучаемое стабильно во времени; модели автомобилей за десятилетия меняются.
3. **a)** Независимая распределённость означает отсутствие смещения при сборе; один дилерский центр даёт избыток одной марки.
4. **b)** Малозаметные изменения данных со временем ухудшают качество, а рекомендательная система даже меняет то поведение, на котором учится.

Разобранные примеры

Один градиентный шаг на функции из упражнения

книги Для $f(x) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$ вычислите градиент в точке $(\pi, 0)$, сделайте один шаг градиентного спуска с $\eta = 0.1$ и проверьте, что f уменьшается.

Частные производные: $\partial f / \partial x_1 = \cos(x_1)\cos(x_2) + 0.5\cos(0.5x_1)\cos(0.5x_2)$ и $\partial f / \partial x_2 = -\sin(x_1)\sin(x_2) - 0.5\sin(0.5x_1)\sin(0.5x_2)$.

В точке $(\pi, 0)$: $\partial f / \partial x_1 = (-1)(1) + 0.5 \times \cos(\pi/2) \times 1 = -1$, а $\partial f / \partial x_2 = 0$, потому что $\sin(0) = 0$; значит, градиент равен $(-1, 0)$.

Шаг: $x - \eta \nabla f = (\pi, 0) - 0.1 \times (-1, 0) = (\pi + 0.1, 0)$, примерно $(3.2416, 0)$.

Проверка: $f(\pi, 0) = 0 + \sin(\pi/2) = 1$, а $f(\pi + 0.1, 0) = \sin(\pi + 0.1) + \sin(\pi/2 + 0.05) \approx -0.0998 + 0.9988 = 0.8990$, что меньше.

Ответ: Градиент $(-1, 0)$; новая точка примерно $(3.2416, 0)$; f падает с 1 до примерно 0.899.

Задачи

1. Ваша векторизованная f получает матрицу X формы $(5, 2)$. Какой формы должен быть выход и что такое его третий элемент?

Подсказка: Книга требует $[f(X)]_i = f(X_i)$.

Ответ: Вектор формы (5) ; его третий элемент — f , применённая к третьей строке X .

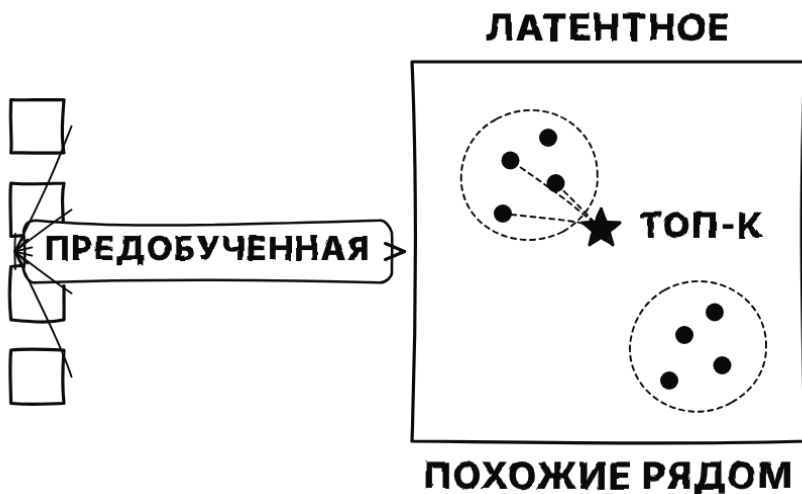
2. Спам-фильтр, обученный на письмах 2015 года, внедряют сегодня. Назовите, какая часть предположения i.i.d. под наибольшей угрозой, и почему.

Подсказка: Спросите, осталось ли изучаемое тем же со временем.

Ответ: Часть об одинаковой распределённости: стили и темы спама за годы дрейфуют, так что сегодняшние письма приходят из другого распределения, чем обучающие данные, — случай сдвига домена.

Разновидности обучения с учителем

Существует много вариаций стандартного сценария обучения с учителем, хотя большинство успешных приложений в той или иной форме используют именно обучение с учителем. Например, в некоторых наборах данных могут отсутствовать целевые значения y_i , и тогда мы говорим об обучении без учителя. Типичные приложения обучения без учителя — алгоритмы кластеризации, в которых мы хотим объединить входные данные в кластеры так, чтобы точки внутри кластера были похожи, а точки из разных кластеров — непохожи. Другой пример: в поисковой системе мы можем захотеть найти в большой базе k элементов, наиболее похожих на запрос пользователя.



Кластеры в латентном пространстве

При работе со сложными данными, такими как изображения, это нетривиально, потому что расстояния между изображениями плохо определены, если работать с пикселями: даже небольшие возмущения могут изменить миллионы пикселей. Однако предположим, что у нас есть некоторая дифференцируемая модель, которую мы уже оптимизировали для другой задачи, которую считаем достаточно общей, например классификации изображений. Мы называем её предобученной моделью. Как мы увидим, внутренние состояния этой модели можно истолковать как векторы в многомерном пространстве. Во многих случаях эти векторы, как показано, обладают полезными геометрическими свойствами в том смысле, что семантически похожие объекты отправляются, встраиваются, в близкие точки этих

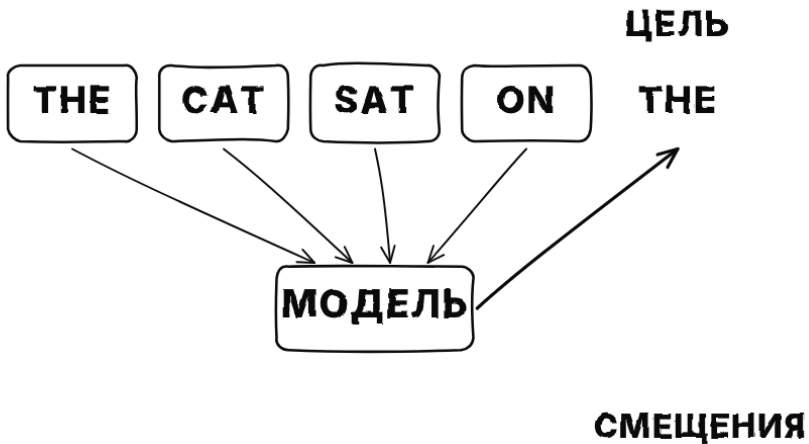
представлений. Поэтому мы можем использовать эти латентные представления со стандартными моделями кластеризации, такими как смеси гауссиан. Рисунок F.3.1 даёт общий обзор этой идеи.

А если у нас нет доступа к предобученной модели?

Распространённая вариация обучения без учителя называется самообучением, SSL. Цель самообучения — автоматически найти некую задачу с учителем в обычном неразмеченном наборе данных, чтобы оптимизировать модель, пригодную для большого набора прикладных задач. Например, если у нас есть большой корпус текстов, мы всегда можем оптимизировать программу, предсказывающую, как, скорее всего, продолжится небольшой кусок текста. Осознание того, что нейронные сети могут эффективно встраивать и текст, если они предобучены самообучением, оказало глубокое влияние на сообщество.

Крупные наборы данных из сети также полны предубеждений, ругательств и вульгарного содержания; признание того, что модели, обученные на этих данных, усваивают эти предубеждения, стало ещё одним важным осознанием, и это одна из главных претензий к закрытым фундаментальным моделям.

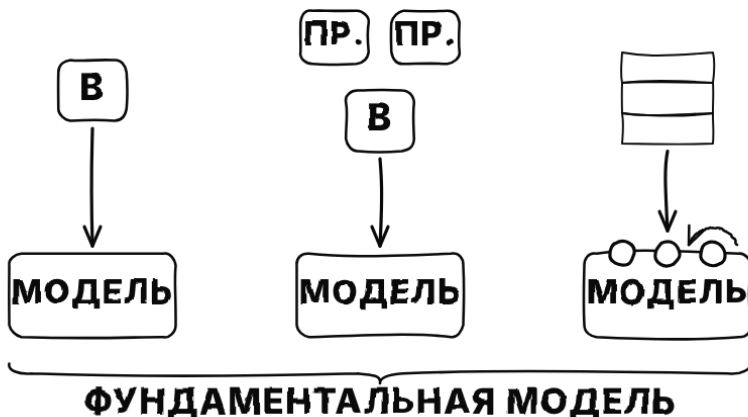
САМООБУЧЕНИЕ



Учимся у самого текста

Рисунок F.3.2 показывает три способа использования обученных моделей. Без примеров: вопрос задаётся модели напрямую, что возможно с генеративными языковыми моделями, которые вводятся в главе 8. Промптинг с несколькими примерами похож, но на вход подаётся несколько примеров. Обе техники применимы, только если базовая модель показывает высокую способность к обобщению. Дообучение: модель оптимизируется градиентным спуском на небольшом наборе примеров, что проходит похоже на обучение модели с нуля.

ZERO-SHOT FEW-SHOT ДООБУЧЕНИЕ

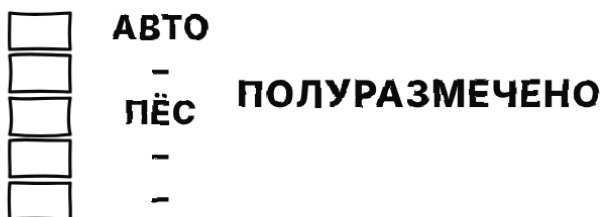


Без примеров, с примерами, дообучение

Как мы увидим в главах 8 и 10, LLM можно рассматривать как современные воплощения этой базовой идеи, поскольку оптимизация моделей вроде GPT или Llama всегда начинается с базового самообучения в виде предсказания следующего токена. Эти модели иногда называют фундаментальными моделями. В простейшем случае их можно использовать «из коробки» для новой задачи, например для ответа на запрос: тогда мы говорим, что они используются без примеров. Для LLM также можно дать небольшое число примеров новой задачи во входном запросе, и тогда мы говорим о промптинге с несколькими примерами. В самом общем случае мы можем взять предобученную фундаментальную модель и оптимизировать её параметры градиентным спуском на новой задаче: это называется

дообучением модели. Рисунок F.3.2 сравнивает три подхода. В этой книге мы сосредоточены на построении моделей с нуля, но дообучение можно выполнять похожими средствами.

Дообучение особенно упрощается благодаря большим открытым репозиториям в сети. Дообучать можно весь набор параметров исходной модели или только меньшее подмножество либо небольшое число дополнительных параметров: это называется параметрически эффективным дообучением, PEFT. Обучение по нескольким примерам также можно выполнять дообучением модели; в случаях, когда дообучение не нужно, мы говорим, что модель выполняет обучение в контексте. Техники параметрически эффективного дообучения мы рассмотрим в следующем томе.



Настройка немногих параметров

Возможны и многие другие вариации обучения с учителем, перечислить которые подробно здесь нет места, кроме некоторых общих намёков. Если размечена лишь часть набора данных, мы имеем частично размеченный сценарий; несколько примеров обучения с частичной разметкой мы увидим в главе 12. Кроме того, возможны сценарии с несколькими наборами данных из «похожих» распределений или из одного распределения за разные периоды времени, что порождает бесчисленные задачи в зависимости от порядка, в котором поступают задачи или данные, включая адаптацию домена, мета-обучение, непрерывное обучение, метрическое обучение, разобучение и так далее. Некоторые из них будут рассмотрены в следующем томе.

Вопросы

1. Почему нельзя просто кластеризовать изображения, сравнивая их пиксели?
 - a. Потому что пиксели — не числа
 - b. Потому что расстояния по пикселям плохо определены: малые возмущения могут изменить миллионы пикселей
 - c. Потому что у изображений слишком мало пикселей
 - d. Потому что кластеризация работает только с текстом
2. Что делает самообучение возможным на неразмеченном корпусе текстов?
 - a. Люди размечают каждое предложение
 - b. Задача с учителем находится в данных автоматически, например предсказание того, как продолжится кусок текста
 - c. Модели заранее дают ответы
 - d. Кластеризация заменяет обучение
3. Чем дообучение отличается от промптинга с несколькими примерами?
 - a. Дообучение меняет параметры модели градиентным спуском; промптинг с примерами лишь помещает примеры во вход
 - b. Промптингу с примерами нужно больше данных
 - c. Дообучению не нужны примеры
 - d. Это одно и то же

4. Что такое частично размеченный сценарий?
- a. Сценарий вовсе без данных
 - b. Сценарий, где размечена лишь часть набора данных
 - c. Сценарий с двумя моделями
 - d. Сценарий, где метки меняются со временем

Ответы

1. **b)** Пиксельные расстояния не отражают смысл; вместо этого книга использует внутренние векторы предобученной модели, где похожие объекты оказываются рядом.
2. **b)** Самообучение находит задачу с учителем внутри неразмеченных данных, например предсказание следующего токена, как делают LLM.
3. **a)** Промптинг с примерами даёт примеры в запросе, в контексте; дообучение оптимизирует параметры на небольшом наборе данных.
4. **b)** Если метки есть лишь у части набора данных, сценарий частично размеченный.

Задачи

1. Вы хотите найти среди миллиона отпускных фотографий десять самых похожих на новую. Опираясь на отрывок, опишите рабочий подход.

Подсказка: Не сравнивайте пиксели; подумайте о предобученной модели.

Ответ: Пропустите каждую фотографию через предобученную модель, например классификатор изображений, и возьмите её внутренний вектор для каждой фотографии. Семантически похожие фотографии встраиваются рядом, так что вы извлекаете десять

фотографий, чьи векторы ближе всего к вектору новой фотографии, — поиск топ- k в латентном пространстве.

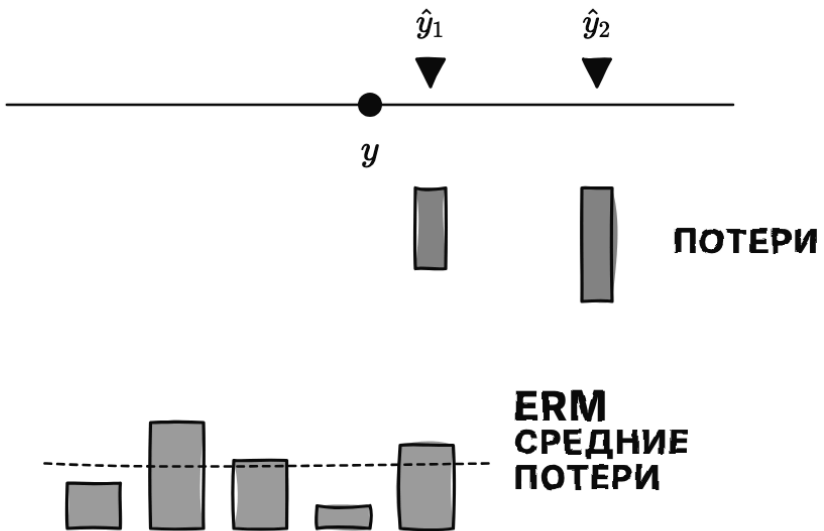
2. Назовите три способа использования обученной модели с рисунка F.3.2 и скажите, какие из них меняют её параметры.

Подсказка: Два меняют только вход.

Ответ: Без примеров, когда вопрос задаётся напрямую; промптинг с несколькими примерами, когда во вход добавляется несколько примеров; и дообучение, когда модель оптимизируется градиентным спуском на небольшом наборе данных. Параметры меняет только дообучение.

Функции потерь

Когда данные собраны, нам нужно формализовать идею «приближения» желаемого поведения, что мы делаем, вводя понятие функций потерь.



Потери ранжируют предсказания

Определение, функция потерь: для желаемого целевого значения y и предсказанного значения $\hat{y} = f(x)$ от модели f функция потерь $l(y, \hat{y}) \in \mathbb{R}$ — это скалярная дифференцируемая функция, значение которой коррелирует с качеством модели. То есть $l(y,$

$\hat{y}_1) < l(y, \hat{y}_2)$ означает, что предсказание $\hat{y}_1$ лучше предсказания $\hat{y}_2$ относительно опорного значения, целевого y .

Функция потерь воплощает наше понимание задачи и наши предпочтения в пространстве решений на вещественной шкале, которую можно использовать в алгоритме оптимизации. Будучи дифференцируемой, она позволяет превратить задачу обучения в задачу математической оптимизации, решаемую градиентным спуском через минимизацию средних потерь на нашем наборе данных.

Для этого, имея набор данных $\mathcal{S}_n$ из пар (x_i, y_i) и функцию потерь l , разумная задача оптимизации — минимальные средние потери на наборе данных, достижимые любой возможной дифференцируемой моделью f : $f^* = \operatorname{argmin}_f (1/n) \sum_{i=1}^n l(y_i, f(x_i))$, среднее по набору данных потерь предсказания на i -том примере.

По историческим причинам уравнение Е.3.1 называют минимизацией эмпирического риска, ERM, где риск — общий синоним потерь. Следующая врезка говорит больше о происхождении термина.

В уравнении Е.3.1 мы неявно предполагаем, что минимизируем по пространству всех возможных функций, определённых на нашем входе x . Вскоре мы увидим, что наши модели всегда можно параметризовать набором тензоров w , называемых параметрами модели, и минимизация выполняется поиском оптимального значения этих параметров численной оптимизацией, что мы обозначаем $f(x, w)$. Следовательно, имея набор данных $\mathcal{S}_n$, функцию потерь l и пространство моделей f , мы можем обучить модель, оптимизируя эмпирический риск уравнения Е.3.1 градиентным спуском, уравнение Е.2.27: $w^* =$

$\operatorname{argmin}_w (1/n) \sum_{i=1}^n l(y_i, f(x_i, w))$, где минимизация теперь ведётся по тензору параметров w .

О дифференцируемости функции потерь. Прежде чем двигаться дальше, сделаем два замечания о подходе ERM. Во-первых, заметьте, что требование дифференцируемости l фундаментально. Рассмотрим простую задачу бинарной классификации, которую мы как следует введём в следующей главе, где y может принимать лишь два значения, -1 или $+1$. Для вещественной модели $f(x) \in \mathbb{R}$ мы можем отождествить два решения со знаком f , который обозначим $\operatorname{sign}(f(x))$, и определить потери 0/1 так: $l(y, \hat{y})$ равна 0, если $\operatorname{sign}(\hat{y}) = y$, и 1 иначе.



Потери 0/1 против шарнирных

Хотя это согласуется с нашим интуитивным понятием «быть правым», как функция потерь это бесполезно, поскольку её градиент почти всегда будет нулевым, кроме мест, где знак f меняется, и любой алгоритм градиентного спуска застрянет на инициализации. Менее интуитивная величина в этом случае — отступ $u\hat{y}$, положительный или отрицательный в зависимости от того, совпадает ли знак модели с желаемым или нет, но меняющийся непрерывно, в отличие от потерь 0/1 из уравнения Е.3.3. Возможная функция потерь в этом случае — шарнирные потери $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$, которые используются для обучения моделей опорных векторов. Если отбросить детали, это показывает внутреннее напряжение между проектированием функций потерь, кодирующих наше понятие качества, и одновременно их пригодностью для численной оптимизации.

Врезка о риске и потерях: минимизацию эмпирического и ожидаемого риска в такой постановке обычно связывают с работами русского учёного в области информатики Владимира Вапника, которые породили область статистической теории обучения, SLT. Статистическая теория обучения особенно интересуется поведением уравнения Е.3.1, рассматриваемого как приближение уравнения Е.3.5 по конечной выборке, при некотором ограниченном классе функций f и некоторой мере их сложности. Противоречащие интуиции свойства современных нейронных сетей, такие как сильное обобщение долго после того, как следовало бы ожидать переобучения, открыли много новых направлений исследований в статистической теории обучения; см. также введение к главе 9.

Вопросы

1. Почему потери 0/1 бесполезны для градиентного спуска?
 - a. Потому что их слишком дорого вычислять
 - b. Потому что их градиент почти всегда равен нулю, так что градиентный спуск застревает на инициализации
 - c. Потому что они могут быть отрицательными
 - d. Потому что они работают только для регрессии
2. Что минимизирует минимизация эмпирического риска?
 - a. Наибольшие потери на наборе данных
 - b. Средние потери предсказаний модели по набору данных
 - c. Число параметров
 - d. Напрямую потери на будущих, невиданных данных
3. Что такое отступ $u\hat{y}$ в бинарной классификации?
 - a. Величина, положительная, когда знак модели согласуется с целевым, и отрицательная иначе, меняющаяся непрерывно
 - b. Потери 0/1, умноженные на два
 - c. Расстояние между двумя классами в пикселях
 - d. Всегда положительное число

Ответы

1. **b)** Потери 0/1 плоские везде, кроме мест, где меняется знак f , так что их градиент не даёт направления движения.

2. б) ERM ищет модель, на практике параметры w , минимизирующую среднее $l(y_i, f(x_i))$ по n примерам.

3. а) Отступ положителен или отрицателен в зависимости от того, совпадают ли знаки, и, в отличие от потерь $0/1$, меняется непрерывно.

Разобранные примеры

Шарнирные потери против потерь $0/1$ Вычислите шарнирные потери $\max(0, 1 - y\hat{y})$ и потери $0/1$ для трёх предсказаний: $y = +1$ при $\hat{y} = 0.3$; $y = +1$ при $\hat{y} = 2$; $y = -1$ при $\hat{y} = 0.5$. Затем вычислите средние шарнирные потери.

Первый случай: отступ равен $1 \times 0.3 = 0.3$, так что шарнирные потери $\max(0, 0.7) = 0.7$; знак верный, так что потери $0/1$ равны 0 .

Второй случай: отступ равен 2 , так что шарнирные потери $\max(0, -1) = 0$; потери $0/1$ равны 0 .

Третий случай: отступ равен $-1 \times 0.5 = -0.5$, так что шарнирные потери $\max(0, 1.5) = 1.5$; знак неверный, так что потери $0/1$ равны 1 .

Эмпирический риск при шарнирных потерях — это среднее: $(0.7 + 0 + 1.5) / 3 = 2.2 / 3 \approx 0.733$.

Ответ: Шарнирные потери 0.7 , 0 и 1.5 (среднее ≈ 0.733); потери $0/1$ равны 0 , 0 и 1 .

Задачи

1. В первом случае выше предсказание меняется с $\hat{y} = 0.3$ на $\hat{y} = 0.5$. Как меняются шарнирные потери и потери $0/1$ и что это говорит об их градиентах?

Подсказка: Пересчитайте обе потери.

Ответ: Шарнирные потери падают с 0.7 до 0.5 , а потери $0/1$ остаются 0 . Шарнирные потери реагируют на изменение, значит, у

них полезный градиент; потери 0/1 там плоские, с нулевым градиентом.

2. Объясните своими словами, почему в минимизации эмпирического риска риск используется как синоним потерь и что в ней эмпирического.

Подсказка: Подумайте, по чему берётся среднее.

Ответ: Риск — историческое название ожидаемых потерь. Она эмпирическая, потому что среднее берётся по конечному набору данных, который у нас реально есть, выборке примеров, а не по всему неизвестному распределению данных.

Ожидаемый риск и выбор функции потерь

Второе замечание: эмпирический риск всегда тривиально минимизировать, определив $f(x)$ так: она возвращает y , если пара (x, y) есть в обучающем наборе S_n , и значение по умолчанию $\bar{y}$, например 0, иначе.

ЗАПОМНЕНО

X	Y
x_1	y_1
x_2	y_2
x_3	y_3

ПОТЕРИ 0

$x_{new} \rightarrow$ **0** **ПОТЕРИ ВЕЛИКИ**

ВЫУЧЕНО?

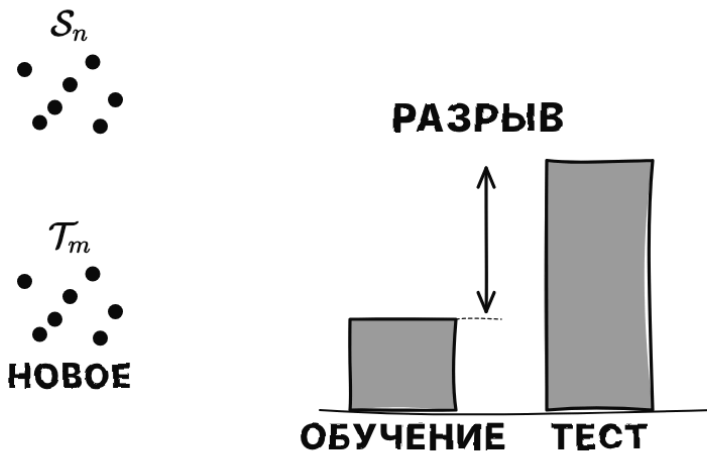
Таблица поиска жульничает

Это таблица поиска, которая возвращает предсказание y , если пара (x, y) содержится в наборе данных, а иначе выдаёт некое постоянное предсказание $\bar{y}$, например 0. Если предположить, что

потери достигают нижней границы всякий раз, когда $y = \hat{y}$, эта модель всегда будет достигать наименьшего возможного эмпирического риска, не принося никакой практической пользы.

Это показывает разницу между запоминанием и обучением, или оптимизацией. Хотя мы ищем модель, оптимизируя некие средние потери на обучающих данных, как в уравнении Е.3.1, наша истинная цель — минимизировать эту величину на неизвестных будущих входах, которых мы ещё не видели. Элементы обучающего набора — лишь посредник для этой цели. Эту идею можно формализовать, определив задачу минимизации ожидаемого риска.

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



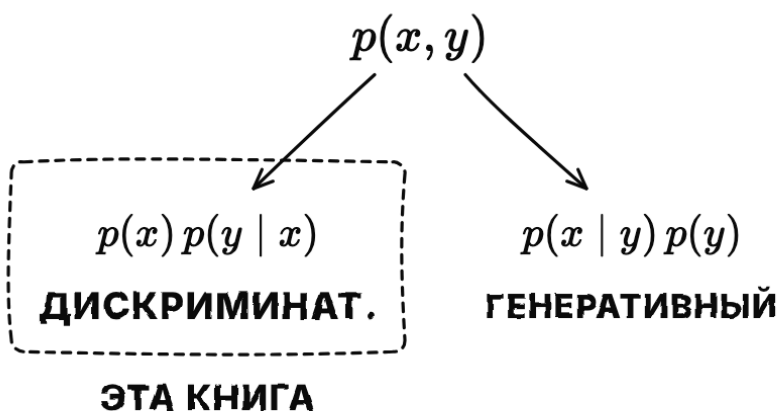
Разрыв обобщения

Определение, ожидаемый риск: для распределения вероятностей $p(x, y)$ и функции потерь l ожидаемый риск, ER , определяется как $ER[f] = \mathbb{E}_p(x,y)[l(y, f(x))]$.

Минимизацию уравнения Е.3.5 можно понимать как минимизацию средних, или ожидаемых, потерь по всем возможным парам «вход — выход», например по всем возможным письмам, которые могла бы увидеть наша модель. Ясно, что модель с низким ожидаемым риском гарантированно работала бы правильно. Однако величину из уравнения Е.3.5 невозможно вычислить на практике, поскольку перечислить и разметить все точки данных нельзя. Эмпирический риск даёт оценку ожидаемого риска при выборе конкретного набора данных и может рассматриваться как приближение ожидаемого риска методом Монте-Карло.

Разница в потерях между ожидаемым и эмпирическим риском называется разрывом обобщения: алгоритм чистого запоминания, как в уравнении Е.3.4, будет плохо обобщать или, иначе говоря, переобучится на конкретных обучающих данных, которые мы дали. Обобщение можно проверить на практике, выделив отдельный тестовый набор $\mathcal{T}_m$ из m точек данных, никогда не использовавшихся при обучении, так что $\mathcal{S}_n \cap \mathcal{T}_m = \emptyset$. Тогда разницу эмпирических потерь между $\mathcal{S}_n$ и $\mathcal{T}_m$ можно использовать как приближённую меру переобучения.

Как выбрать подходящую функцию потерь? Если вы ещё этого не сделали, сейчас хороший момент изучить или просмотреть материал приложения, особенно распределения вероятностей, достаточные статистики и оценку максимального правдоподобия.



Два способа разложить $p(x, y)$

Как мы увидим в следующих главах, функция потерь кодирует наши априорные знания о решаемой задаче и сильно влияет на качество. В некоторых случаях простых соображений о задаче достаточно, чтобы спроектировать подходящие потери, как это было сделано для шарнирных потерь в предыдущем разделе.

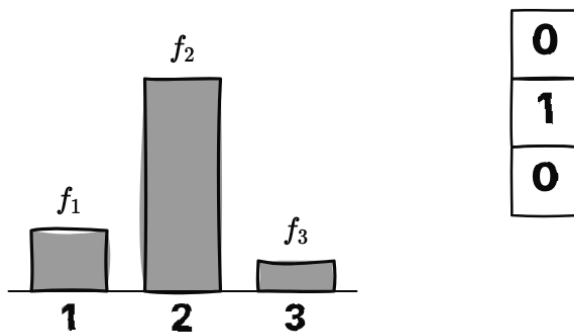
Однако можно работать более принципиально, переформулировав весь процесс обучения в чисто вероятностных терминах, что мы сейчас и покажем. Эта формулировка даёт альтернативный взгляд на обучение, который в некоторых сценариях может быть более интуитивным или более полезным. Это также предпочтительный взгляд многих книг. В этом разделе

мы даём основные идеи, а конкретные приложения рассматриваем далее в книге.

Ключевое наблюдение следующее. В разделе 3.1 мы начали с предположения, что наши примеры приходят из распределения $p(x, y)$. По правилу произведения вероятностей мы можем разложить $p(x, y)$ как $p(x, y) = p(x) p(y | x)$, так что $p(x)$ даёт вероятность наблюдать каждый вход x , а условный член $p(y | x)$ описывает вероятность наблюдать некий выход y при входе x . Можно разложить и так: $p(x, y) = p(x | y) p(y)$; методы, которым нужно оценивать $p(x)$ или $p(x | y)$, называются генеративными, а методы, оценивающие $p(y | x)$, — дискриминативными. Кроме языкового моделирования, в этой книге мы сосредоточены на втором случае, а генеративное моделирование в более широком смысле рассматриваем в следующем томе. Приближать $p(y | x)$ функцией $f(x)$ имеет смысл, если предположить, что вероятностная масса в основном сосредоточена вокруг одной точки y , то есть $p(y | x)$ близка к так называемой дельта-функции Дирака, и это резко упрощает общую формулировку задачи.

СУММА = 1

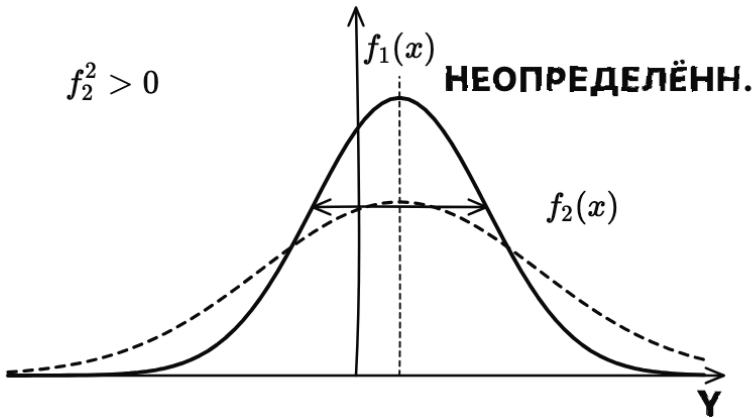
ONE-HOT Y



Модель выдаёт распределение

Однако мы можем ослабить это, предположив, что наша модель $f(x)$ не даёт предсказание напрямую, а используется для параметризации достаточных статистик условного распределения вероятностей $p(y | f(x))$ над возможными выходами. Например, рассмотрим задачу классификации, где y может принимать три значения: 1, 2 или 3. Мы можем считать, что y модели три выхода, параметризующие категориальное распределение над этими классами, так что $p(y | f(x)) = \prod_{i=1}^3 f_i(x)^{y_i}$. Здесь $y \sim \text{Binary}(3)$ — это one-hot кодирование класса y , вектор из нулей, кроме элемента данного класса, равного 1, как формально вводится в разделе 4.2, а $f(x) \sim \Delta(3)$ — предсказанные вероятности каждого класса. Другой пример: предположим, мы хотим предсказать одно скалярное значение $y \in \mathbb{R}$, то есть

регрессию. Это можно смоделировать двузначной функцией $f(x) \sim (2)$, так что предсказание — гауссиана с подходящим средним и дисперсией: $p(y | f(x)) = \mathcal{N}(y | f_1(x), f_2^2(x))$.



Регрессия как колокол

Здесь второй выход $f(x)$ возводится в квадрат, чтобы предсказанная дисперсия оставалась положительной. Как видно, это очень общая постановка, которая включает наше предыдущее обсуждение и даёт проектировщику больше гибкости, поскольку выбрать конкретную параметризацию для $p(y | x)$ может быть проще, чем выбрать конкретную функцию потерь $l(y, \hat{y})$. Кроме того, этот подход даёт более прямой способ моделировать неопределённость, например дисперсию в уравнении Е.3.6.

Вопросы

1. Почему таблица поиска, запоминающая обучающий набор, не имеет практической ценности, хотя и минимизирует эмпирический риск?
 - a. Потому что она слишком медленная
 - b. Потому что наша истинная цель — низкие потери на невиданных будущих входах, где она просто возвращает значение по умолчанию
 - c. Потому что таблицы поиска нельзя дифференцировать
 - d. Потому что ей нужен тестовый набор
2. Как на практике можно оценить разрыв обобщения?
 - a. Точно вычислив ожидаемый риск
 - b. Сравнив эмпирические потери на обучающем наборе с потерями на отдельном тестовом наборе, никогда не использовавшемся при обучении
 - c. Подсчитав параметры
 - d. Обучив дважды на одних и тех же данных
3. Во вероятностном взгляде на классификацию с тремя классами что выдаёт модель?
 - a. Один номер класса
 - b. Три вероятности, по одной на класс, параметризующие категориальное распределение
 - c. Значение потерь
 - d. One-hot кодирование входа

4. Почему второй выход регрессионной модели возводится в квадрат?
- a. Чтобы увеличить среднее
 - b. Чтобы предсказанная дисперсия гауссианы всегда была положительной
 - c. Чтобы ускорить обучение
 - d. Чтобы превратить регрессию в классификацию

Ответы

1. **b)** Запоминание достигает наименьшего эмпирического риска, но ничего не говорит о новых входах; обучающий набор — лишь посредник для будущих данных.
2. **b)** Ожидаемый риск вычислить нельзя, поэтому его приближает отложенный тестовый набор, а разница потерь измеряет переобучение.
3. **b)** $f(x) \sim \Delta(3)$ даёт предсказанные вероятности трёх классов, достаточные статистики категориального распределения.
4. **b)** Дисперсия должна быть положительной; возведение $f_2(x)$ в квадрат это гарантирует.

Разобранные примеры

Вероятность истинного класса Трёхклассовая модель предсказывает $f(x) = (0.2, 0.7, 0.1)$. Истинный класс — 2. Запишите его one-hot кодирование и вычислите $p(y | f(x))$ по формуле произведения из книги.

One-hot кодирование класса 2 имеет 1 на позиции 2 и нули в остальных: $y = (0, 1, 0)$.

Формула перемножает $f_i(x)$ в степени y_i для $i = 1, 2, 3$.

Любое число в степени 0 равно 1, так что считается только второй множитель: $0.2^0 \times 0.7^1 \times 0.1^0 = 1 \times 0.7 \times 1$.

Результат — просто вероятность, которую модель даёт истинному классу.

Ответ: $p(y | f(x)) = 0.7$

Задачи

1. Регрессионная модель выдаёт $f(x) = (3.0, -0.5)$. Какую гауссиану она предсказывает для y ?

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

Ответ: Нормальное распределение со средним 3.0 и дисперсией $(-0.5)^2 = 0.25$, то есть со стандартным отклонением 0.5.

2. У модели средние потери 0.1 на обучающем наборе и 0.6 на отложенном тестовом наборе. Каков примерный разрыв обобщения и о чём он говорит?

Подсказка: Вычтите одни эмпирические потери из других.

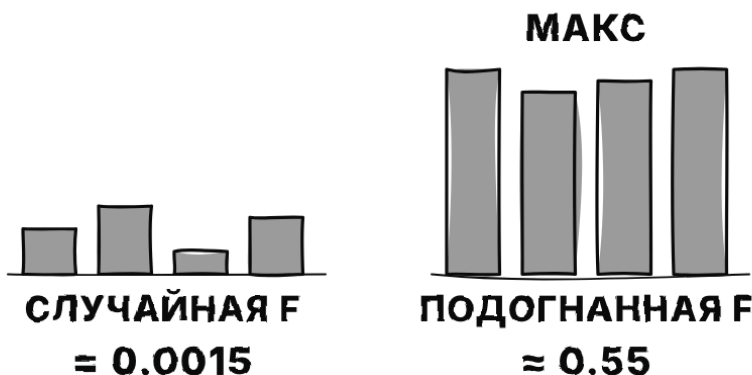
Ответ: Около 0.5. Модель работает гораздо лучше на виденных данных, чем на новых, что говорит о переобучении, склонности к запоминанию.

Максимальное правдоподобие и байесовское обучение

Как обучить вероятностную модель? Вспомните, что мы предположили, что примеры в нашем наборе данных $\mathcal{S}_n$ — это i.i.d. выборки из распределения вероятностей $p(x, y)$.

Следовательно, для модели $f(x)$ вероятность, которую конкретный выбор функции f приписывает самому набору данных, даётся произведением по примерам набора: $p(\mathcal{S}_n | f) = \prod_{i=1}^n p(y_i | f(x_i))$.

$$p(\mathcal{S}_n | f) = \prod_i p(y_i | f(x_i))$$



Правдоподобие: перемножаем вероятности

Величина $p(\mathcal{S}_n | f)$ называется правдоподобием набора данных. При случайном выборе $f(x)$ модель будет приписывать вероятности более или менее случайно по всем возможным входам и выходам, и правдоподобие нашего конкретного набора данных будет малым. Разумная стратегия, стало быть, — выбрать модель так, чтобы правдоподобие набора данных, напротив, было максимальным. Это прямое применение подхода максимального правдоподобия, описанного в приложении.

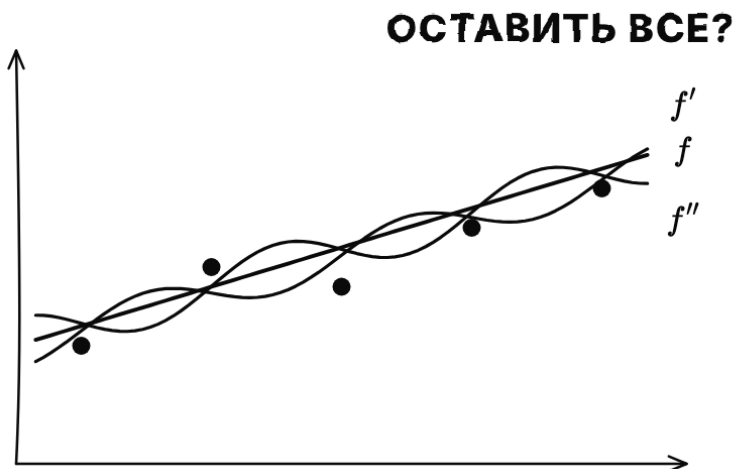
Определение, максимальное правдоподобие: для набора данных $\mathcal{S}_n$ из пар (x_i, y_i) и семейства распределений вероятностей $p(y | f(x))$, параметризованных $f(x)$, решение максимального правдоподобия даётся $f^* = \operatorname{argmax}_f \prod_{i=1}^n p(y_i | f(x_i))$.

Хотя мы снова остались с задачей оптимизации, теперь она прямо следует из законов вероятности, как только выбраны все распределения, в отличие от прежнего, где конкретная функция потерь была частью пространства проектирования. Однако два взгляда тесно связаны. Переходя в логарифмическое пространство и к задаче минимизации, получаем, что $\operatorname{argmax} \log \prod_i p(y_i | f(x_i))$ равен аргмину $\sum_i -\log p(y_i | f(x_i))$.

Следовательно, две формулировки тождественны, если считать $-\log p(y | f(x))$ «псевдопотерями», которые нужно оптимизировать. Как мы увидим, все функции потерь, используемые на практике, можно получить из принципа максимального правдоподобия при конкретном выборе этого члена. Оба взгляда интересны, и мы призываем читателей держать их в уме по мере продвижения по книге.

Раздел 3.3, Байесовское обучение. Здесь мы обсудим дальнейшее обобщение вероятностной формулировки, называемое

байесовскими нейронными сетями, BNN, которое представляет интерес в литературе. Мы даём лишь общую идею и отсылаем читателя за подробностями к одному из многих обстоятельных руководств.

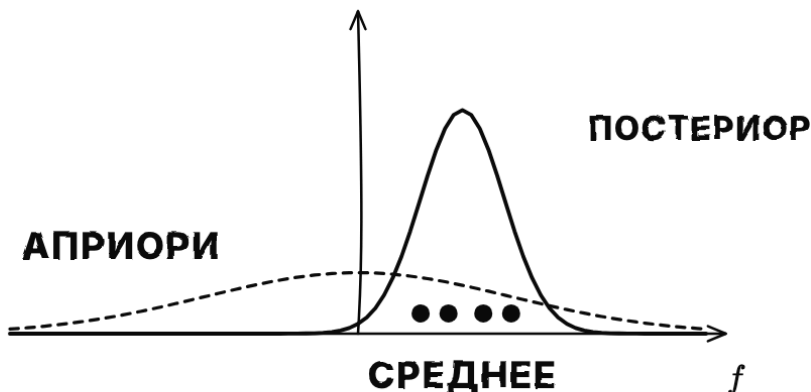


Данным подходят многие функции

Проектируя функцию вероятности $p(y | f(x))$ вместо $f(x)$ напрямую, мы можем работать с ситуациями, где интересно больше одного предсказания, то есть где у функции вероятности больше одной моды. Однако наша процедура всё ещё возвращает одну функцию $f(x)$ из пространства всех возможных функций, тогда как может оказаться, что во всём пространстве моделей верна не одна параметризация. В этом случае было бы полезно иметь доступ ко всем им для более точного предсказания.

И снова мы можем достичь этой цели, спроектировав ещё одно распределение вероятностей и позволив правилам вероятности вести нас. Поскольку теперь мы собираемся получить распределение по всем возможным функциям, мы начинаем с определения априорного распределения вероятностей $p(f)$ по всем возможным функциям; вспомните, что на практике f описывается конечным набором параметров, и тогда априорное $p(f)$ становится априорным распределением над этими весами. Например, мы увидим, что во многих ситуациях предпочтительны функции с меньшей нормой, как более устойчивые, и тогда мы могли бы определить априорное распределение $p(f) \propto 1/\|f\|$ для некоторой нормы $\|f\|$ функции f .

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



От априорного к апостериорному

Когда набор данных наблюдён, вероятность над f смещается в зависимости от априорного распределения и правдоподобия, и обновление даётся теоремой Байеса: $p(f \mid \mathcal{S}_n) = p(\mathcal{S}_n \mid f) p(f) / p(\mathcal{S}_n)$, где $p(f)$ — априорное распределение, до наблюдения набора данных, а $p(f \mid \mathcal{S}_n)$ — апостериорное, после наблюдения.

Член $p(f \mid \mathcal{S}_n)$ называется апостериорной функцией распределения, а член $p(\mathcal{S}_n)$ в знаменателе называется эвиденсом, и он нужен, чтобы апостериорное распределение было правильно нормировано. Предположим пока, что у нас есть доступ к апостериорному распределению. В отличие от прежнего, это распределение может кодировать предпочтение не одной функции f , что может дать лучшую предсказательную силу. Для входа x мы можем сделать предсказание, усреднив все возможные модели по их апостериорному весу: $p(y \mid x) = \int p(y \mid f(x)) p(f \mid \mathcal{S}_n) df$, что приблизительно равно $(1/k) \sum_{i=1}^k p(y \mid f_i(x)) p(f_i \mid \mathcal{S}_n)$.

В уравнении Е.3.10 мы приблизили интеграл средним Монте-Карло по k случайным выборкам из апостериорного распределения, $f_k \sim p(f \mid \mathcal{S}_n)$. Всю красоту этой конструкции портит то, что апостериорное распределение в общем случае невозможно вычислить в замкнутом виде, кроме очень специальных выборов априорного распределения и правдоподобия. За отсутствием этого приходится прибегать к приближённым решениям: либо методами Монте-Карло с марковскими цепями, либо вариационным выводом. В разделе 9.3.1 мы увидим один пример байесовской трактовки параметров модели, называемый Монте-Карло дропаутом.

MAP

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

ПРАВДОП. РЕГУЛЯР.

ML: ДАННЫЕ СЛ. БАЙЕС: F СЛУЧ.

MAP: правдоподобие плюс регуляризатор

Прежде чем закрыть раздел, отметим два интересных факта об апостериорном распределении. Во-первых, предположим, что нас интересует только функция с наибольшей апостериорной плотностью. В этом случае член эвиденса можно игнорировать, и решение распадается на два отдельных члена: $f^* = \operatorname{argmax}_f p(\mathcal{S}_n | f) p(f) = \operatorname{argmax}_f (\log p(\mathcal{S}_n | f) + \log p(f))$, член правдоподобия плюс член регуляризации.

Это называется решением максимума апостериорной вероятности, MAP. Если все функции имеют одинаковый вес априори, то есть если $p(f)$ равномерно на пространстве функций, то второй член — константа, и задача сводится к решению максимального правдоподобия. В общем же случае решение

максимума апостериорной вероятности наложит штраф на функции, слишком отклоняющиеся от нашего априорного распределения. Мы увидим, что это полезная идея для борьбы с переобучением и для наложения конкретных ограничений на функцию f . Член $\log p(f)$ обычно называют регуляризатором на пространстве функций, поскольку он толкает решение к области притяжения, заданной априорным распределением. Разница между решениями максимального правдоподобия и максимума апостериорной вероятности нестрога связана с разницей между частотной и байесовской интерпретациями вероятности, то есть вероятностями как частотами событий или вероятностями как мерой неопределённости. С самой общей точки зрения, максимальное правдоподобие считает параметры неизвестной фиксированной величиной, а данные — случайной выборкой, тогда как байесовская трактовка считает данные фиксированными, а параметры — случайными величинами.

Вопросы

1. Почему правдоподобие набора данных можно записать как произведение по его примерам?
 - a. Потому что примеры считаются i.i.d., так что их вероятности перемножаются
 - b. Потому что у всех примеров одна метка
 - c. Потому что модель линейна
 - d. Потому что набор данных мал

2. Как максимальное правдоподобие связано с минимизацией потерь?
- a. Никак; они не связаны
 - b. Логарифмирование и смена знака превращают его в минимизацию суммы $-\log p(y_i | f(x_i))$, псевдопотерь
 - c. Оно требует шарнирных потерь
 - d. Оно заменяет градиентный спуск
3. Когда решение максимума апостериорной вероятности сводится к максимальному правдоподобию?
- a. Когда априорное распределение равномерно, так что $\log p(f)$ — константа
 - b. Когда набор данных пуст
 - c. Когда эвиденс равен нулю
 - d. Никогда
4. Почему байесовские нейронные сети трудно использовать точно?
- a. Потому что априорные распределения нельзя определить
 - b. Потому что апостериорное распределение в общем случае невозможно вычислить в замкнутом виде, так что нужны приближения вроде МСМС или вариационного вывода
 - c. Потому что у них нет правдоподобия
 - d. Потому что им нужны размеченные тестовые наборы

Ответы

1. **a)** У независимых и одинаково распределённых примеров совместная вероятность равна произведению отдельных вероятностей.
2. **b)** Максимизация логарифма произведения равна минимизации суммы отрицательных логарифмов вероятностей, которая играет роль потерь.
3. **a)** При равном априорном весе всех функций член регуляризации постоянен, и остаётся только правдоподобие.
4. **b)** Кроме очень специальных априорных распределений и правдоподобий, у апостериорного распределения нет замкнутого вида.

Разобранные примеры

Правдоподобие и отрицательный логарифм

правдоподобия На трёх примерах модель даёт истинным меткам вероятности 0.9, 0.8 и 0.5. Вычислите правдоподобие набора данных и отрицательный логарифм правдоподобия, используя натуральные логарифмы.

При i.i.d. примерах правдоподобие — произведение: $0.9 \times 0.8 \times 0.5 = 0.36$.

Псевдопотери — сумма $-\log p$ по каждому примеру: $-\ln 0.9 \approx 0.105$, $-\ln 0.8 \approx 0.223$, $-\ln 0.5 \approx 0.693$.

Их сумма около 1.022.

Проверка: $-\ln 0.36 \approx 1.022$, то же самое, поскольку логарифм произведения — сумма логарифмов.

Ответ: Правдоподобие 0.36; отрицательный логарифм правдоподобия ≈ 1.022 .

Задачи

1. Вторая модель даёт тем же трём истинным меткам вероятности 0.6, 0.6 и 0.6. Какую модель предпочитает максимальное правдоподобие?

Подсказка: Вычислите её правдоподобие и сравните с 0.36.

Ответ: Её правдоподобие равно $0.6^3 = 0.216$, меньше 0.36, так что максимальное правдоподобие предпочитает первую модель.

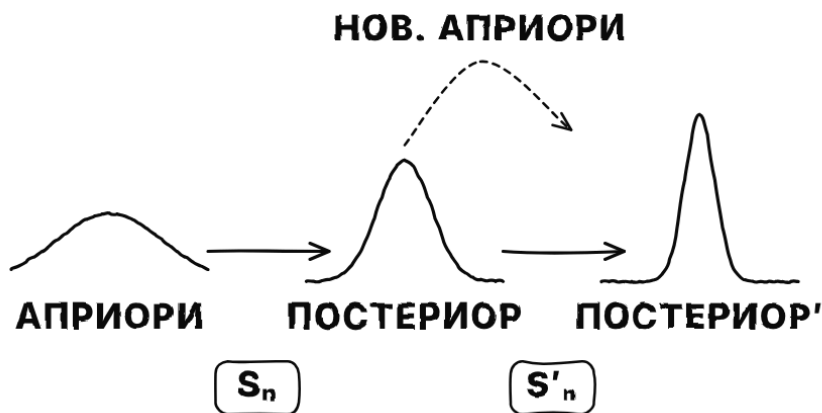
2. При равных априорных вероятностях 0.5 для двух моделей выше найдите по теореме Байеса апостериорную вероятность первой модели.

Подсказка: Умножьте каждое правдоподобие на его априорную вероятность, затем разделите на сумму обоих, эвиденс.

Ответ: Априорная на правдоподобие: $0.5 \times 0.36 = 0.18$ и $0.5 \times 0.216 = 0.108$. Эвиденс равен 0.288, так что апостериорная вероятность первой модели $0.18 / 0.288 = 0.625$.

Линейные модели и квадратичные потери

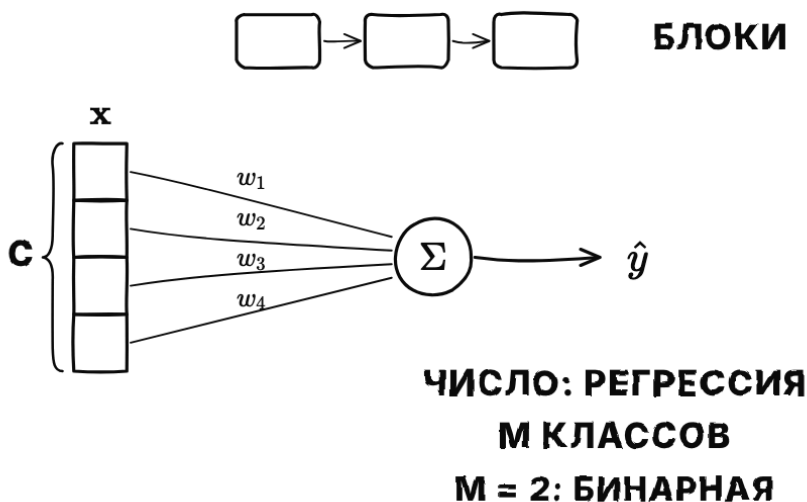
Во-вторых, полная байесовская трактовка даёт простой способ включать новые данные, например новый набор данных S'_n из того же распределения. Для этого мы заменяем априорную функцию в уравнении Е.3.8 апостериорным распределением, вычисленным на первой части набора данных, которое теперь представляет исходное предположение о возможных значениях f и обновляется при взгляде на новые данные. Думайте об исходной априорной функции как о распределении на f после наблюдения начального пустого множества значений. Это может смягчить проблемы при онлайн-обучении моделей, прежде всего так называемое катастрофическое забывание старой информации.



БЕЗ ЗАБЫВАНИЯ

Вчерашнее апостериорное — сегодняшнее априорное

Глава 4, Линейные модели. Коротко: программирование — это выбор подходящей последовательности примитивных операций для решения задачи. По аналогии, построение модели — это выбор правильной последовательности дифференцируемых блоков. В этой главе мы вводим простейший блок, линейные модели, которые предполагают, что входы действуют на выход аддитивно через взвешенное среднее. В некотором смысле все дифференцируемые модели — это хитрые вариации и композиции линейных блоков.



Линейная модель: взвешенная сумма

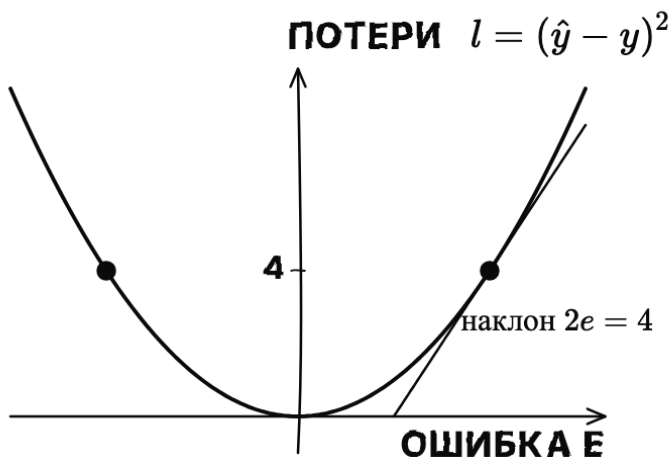
Раздел 4.1, Регрессия по методу наименьших квадратов.

Подытоживая предыдущую главу: задачу обучения с учителем можно задать выбором типа входа x , типа выхода y , модели f и функции потерь l . В этой главе мы рассматриваем простейшие возможные варианты для всех них.

Вход — вектор $x \sim (c)$, соответствующий набору признаков, например с личных признаков клиента банка. Мы используем скаляр c , сокращение от каналов, для обозначения числа признаков, чтобы быть последовательными со следующими главами. Выход — одно вещественное значение $y \in \mathbb{R}$. В неограниченном случае мы говорим, что это задача регрессии. Если y может принимать лишь одно из m возможных значений,

от 1 до m , мы говорим, что это задача классификации. В частном случае $m = 2$ мы говорим, что это задача бинарной классификации. В качестве f мы берём линейную модель, что в некоторых случаях даёт нам простые решения в замкнутом виде, прежде всего регрессию по методу наименьших квадратов, в разделе 4.1.2.

Основные формы, которые нужно помнить, сведены в таблице Т.4.1: n — размер набора данных, s — число признаков, m — число классов; для единообразия мы будем по возможности использовать те же буквы на протяжении всей книги. Начнём с обсуждения выбора функции потерь в случае регрессии. Мы начинаем со случая регрессии, поскольку, как мы покажем позже, классификацию можно решить небольшими изменениями случая регрессии.



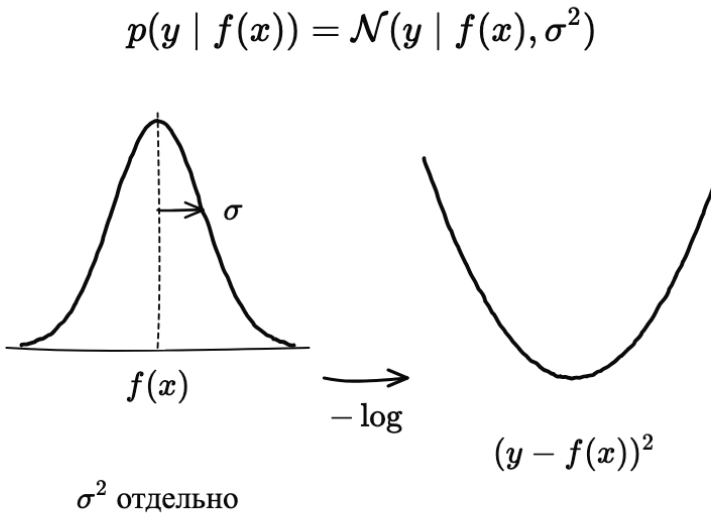
Квадратичные потери

Квадратичные потери и их варианты. Найти функцию потерь для регрессии сравнительно просто, поскольку ошибка предсказания $e = \hat{y} - y$ между предсказанным выходом модели $\hat{y} = f(x)$ и истинным желаемым выходом y — хорошо определённая цель, будучи непрерывной функцией выхода модели, которая монотонно убывает. Поскольку в общем случае знак ошибки предсказания нас не волнует, распространённый выбор — квадратичные потери: $l(\hat{y}, y) = (\hat{y} - y)^2$.

Здесь и далее мы используем символ $\hat{y}$ для обозначения предсказания модели. Как мы увидим, работа с уравнением Е.4.1 даёт нашему решению несколько преимуществ. Среди прочего, градиент квадратичных потерь — линейная функция выхода

модели, что позволяет найти оптимальное решение в замкнутом виде.

Вспоминая принцип максимального правдоподобия, квадратичные потери можно получить, предположив, что выходы модели следуют гауссову распределению с центром в $f(\mathbf{x})$ и постоянной дисперсией σ^2 : $p(y | f(\mathbf{x})) = \mathcal{N}(y | f(\mathbf{x}), \sigma^2)$.

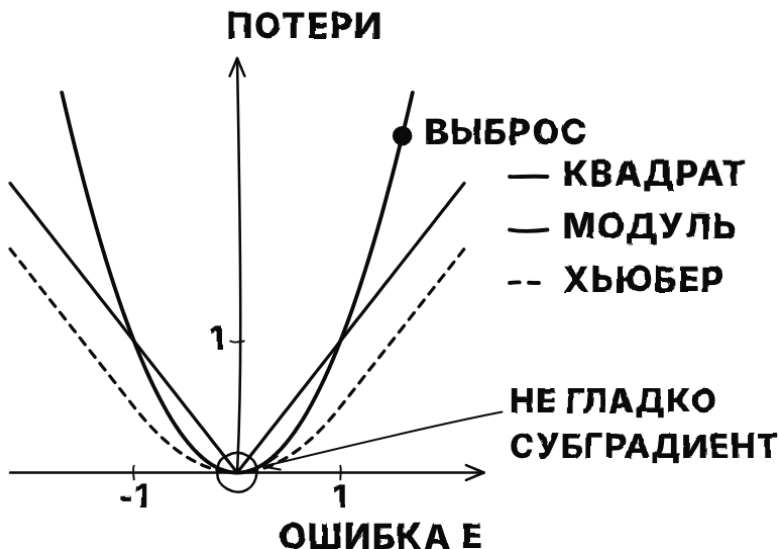


Гауссов шум даёт квадратичные потери

В этом случае логарифм правдоподобия для одной точки можно записать, вспомнив, что логарифм произведения — сумма логарифмов, а логарифм степени — показатель на логарифм, как $\log p(y | f(\mathbf{x}), \sigma^2) = -\log \sigma - \frac{1}{2} \log(2\pi) - (y - f(\mathbf{x}))^2 / (2\sigma^2)$.

Минимизируя уравнение Е.4.2 по f , мы видим, что первые два члена в правой части постоянны, а третий сводится к квадратичным потерям. Минимизацию по σ^2 можно выполнить независимо от оптимизации f , с простым решением в замкнутом виде, уравнение Е.4.9 ниже.

Придумывать варианты квадратичных потерь тоже легко. Например, один недостаток квадратичных потерь в том, что большие ошибки штрафуются с силой, растущей квадратично по ошибке, что может дать чрезмерное влияние выбросам, то есть точкам с сильно неверными метками. Другие варианты, уменьшающие влияние выбросов, — потери по модулю, $l(\hat{y}, y) = \text{abs}(\hat{y} - y)$, или потери Хьюбера, сочетание квадратичных потерь и потерь по модулю. Потери Хьюбера равны $\frac{1}{2}(y - \hat{y})^2$, когда модуль ошибки не больше 1, и $\text{abs}(y - \hat{y}) - \frac{1}{2}$ иначе.



Квадрат, модуль и Хьюбер

Рисунок F.4.1 показывает квадратичные потери, потери по модулю и потери Хьюбера в зависимости от ошибки предсказания $e = \hat{y} - y$.

Потери Хьюбера квадратичны вблизи нулевой ошибки и линейны иначе, а член $-1/2$ добавлен для непрерывности.

Потери по модулю кажутся неподходящим выбором в нашем контексте, поскольку у них есть точка недифференцируемости в 0, из-за модуля. Позже мы увидим, что функции с одной или небольшим числом таких точек на деле не проблематичны. Математически с ними можно работать через понятие субградиента, небольшого обобщения производной.

Практически же можно представить, что если начать со случайной инициализации, градиентный спуск никогда не попадёт в эти точки с идеальной точностью, а производная модуля ε при любом $\varepsilon > 0$ всегда определена.

Вопросы

1. При каком предположении максимальное правдоподобие даёт в точности квадратичные потери?
 - a. Выходы следуют гауссиане с центром в $f(x)$ и постоянной дисперсией
 - b. Выходы следуют равномерному распределению
 - c. Выходы бинарны
 - d. У модели нет параметров
2. Почему потери Хьюбера или потери по модулю могут быть предпочтительнее квадратичных?
 - a. Они всегда дифференцируемы
 - b. Квадратичные потери растут квадратично по ошибке, давая выбросам чрезмерное влияние; другие при больших ошибках растут лишь линейно
 - c. Они дают решения в замкнутом виде
 - d. Они игнорируют знак ошибки, а квадратичные — нет
3. Почему угол потерь по модулю в 0 на практике не настоящая проблема?
 - a. Потому что градиентный спуск со случайного старта практически никогда не попадает ровно в 0, а в

остальных точках производная существует;
математически точку обрабатывают субградиенты

- b. Потому что потери по модулю никогда не используются
 - c. Потому что угол сглаживает GPU
 - d. Потому что потери там постоянны
4. С точки зрения книги, чем линейные модели являются для всех дифференцируемых моделей?
- a. Ничем не связаны
 - b. Простейшим блоком; все дифференцируемые модели — хитрые вариации и композиции линейных блоков
 - c. Всегда лучше
 - d. Частным случаем только классификации

Ответы

1. a) При гауссиане с постоянной дисперсией логарифм правдоподобия имеет два постоянных члена и третий, пропорциональный $(y - f(x))^2$, квадратичным потерям.
2. b) Квадратичные потери штрафуют большие ошибки квадратично, так что сильно неверно размеченные точки доминируют; линейный рост ограничивает их влияние.
3. a) Одна точка недифференцируемости обрабатывается субградиентами, а на практике в неё никогда не попадают с идеальной точностью.
4. b) Глава начинается с того, что называет линейные модели простейшим дифференцируемым блоком, вариациями и композициями которого являются все остальные.

Разобранные примеры

Три функции потерь на двух ошибках

Вычислите квадратичные потери, потери по модулю и потери Хьюбера для малой ошибки $\epsilon = 0.5$ и большой ошибки $\epsilon = 4$.

Квадратичные потери: $0.5^2 = 0.25$ и $4^2 = 16$.

Потери по модулю: 0.5 и 4 .

Потери Хьюбера: для 0.5 модуль ошибки не больше 1, так что это $\frac{1}{2} \times 0.5^2 = 0.125$; для 4 это $4 - \frac{1}{2} = 3.5$.

Большая ошибка весит в 64 раза больше малой при квадратичных потерях, но лишь в 28 раз при Хьюбере, поэтому Хьюбер ограничивает выбросы.

Ответ: Квадратичные 0.25 и 16 ; по модулю 0.5 и 4 ; Хьюбер 0.125 и 3.5 .

Задачи

1. Проверьте, что два куска потерь Хьюбера сходятся при модуле ошибки 1.

Подсказка: Вычислите обе формулы при ошибке 1.

Ответ: Квадратичный кусок даёт $\frac{1}{2} \times 1^2 = 0.5$, а линейный — $1 - \frac{1}{2} = 0.5$: они совпадают, для чего и нужен член $-\frac{1}{2}$.

2. Банковская модель использует $s = 12$ признаков клиента и набор данных из $n = 5000$ клиентов. Какова форма одного входа, и является ли предсказание кредитного лимита в евро задачей регрессии или классификации?

Подсказка: Посмотрите, какого рода значение принимает выход.

Ответ: Один вход — вектор $\mathbf{x} \sim (12)$. Кредитный лимит — неограниченное вещественное значение, так что это задача регрессии.