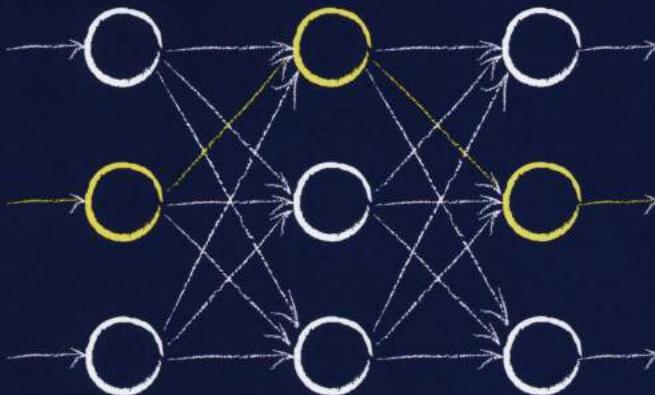


ПОЛНОЦВЕТНОЕ ИЗДАНИЕ

СОЗДАЕМ НЕЙРОННУЮ СЕТЬ



*Математические идеи, лежащие в основе
работы нейронных сетей, и поэтапное создание
собственной нейронной сети на языке Python*

ТАРИК РАШИД

Пролог

Попытки создания разумных машин

На протяжении тысячелетий человечество пытается разгадать тайну работы мозга и создать устройства, способные мыслить.

Мы изобрели кремниевые зажигалки, с помощью которых можем в любой момент получить огонь, блоки для поднятия тяжестей и даже калькуляторы, способные выполнять для нас расчеты, но все эти простые механические и электронные устройства, облегчающие нашу жизнь, нас уже не удовлетворяют.

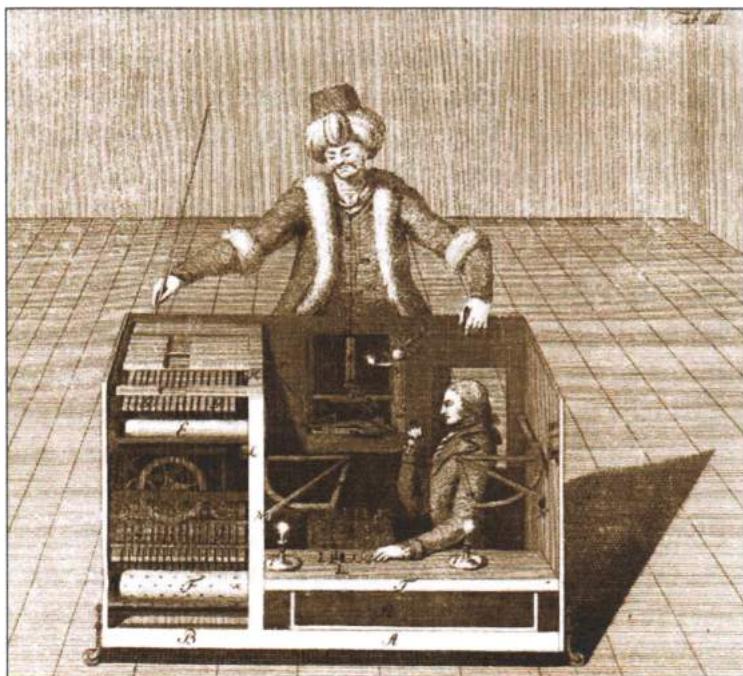
Теперь мы хотим автоматизировать более сложные задачи, такие как группирование схожих фотографий, отделение больных клеток от здоровых и даже игра в шахматы. По-видимому, для решения таких задач требуется человеческий интеллект или по крайней мере некие загадочные возможности человеческой психики, которых вы не найдете в простых устройствах типа калькуляторов.

Идея машин, обладающих интеллектом наподобие человеческого, в равной степени соблазнительная и пугающая, что породило в обществе множество фантазий и страхов на эту тему. Невероятно способный, но и чрезвычайно опасный HAL 9000 из фильма *Космическая одиссея 2001 года* Стэнли Кубрика, боевые роботы в захватывающей кинофраншизе *Терминатор* и говорящий автомобиль KITT с ярко выраженной индивидуальностью в классическом телесериале *Рыцарь дорог* — это лишь некоторые из огромного числа возможных примеров.

Когда в 1997 году чемпион мира по шахматам гроссмейстер Гарри Каспаров потерпел поражение от компьютера IBM Deep Blue, мы не только радовались этому историческому достижению, но и задумались о возможных опасностях со стороны разгулявшегося машинного интеллекта.

Интерес к разумным машинам был настолько велик, что некоторые изобретатели не смогли устоять перед искушением пойти на прямой обман публики, прибегая к всевозможным ухищрениям

и трюкам. Когда секрет шахматной машины *Турок* был раскрыт, оказалось, что внутри ящика скрывался игрок — человек, который и передвигал фигуры посредством системы рычагов.



open-hide.biz

Природа вдохновила новый золотой век

Оптимизм и амбиции по созданию искусственного интеллекта взмыли до новых высот после формализации этого предмета в 1950-х годах. Начальные успехи ознаменовались разработкой компьютеров, играющих в простые игры и доказывающих теоремы. Кое-кто был убежден, что машины с интеллектом на уровне человеческого появятся в течение ближайших десяти лет.

Однако искусственный интеллект оказался твердым орешком, и дальнейший прогресс застопорился. Осознание теоретических трудностей нанесло сокрушительный удар по амбициям создателей искусственного интеллекта, вслед за чем последовали урезание финансирования и потеря интереса к этой области исследований.

Казалось, что машины с их жесткой аппаратной логикой, состоящей сплошь из единиц и нулей, никогда не смогут соперничать с органической гибкостью, иногда размытой, мыслительных процессов биологического мозга.

По прошествии некоторого периода относительно слабого прогресса возникла невероятно мощная идея о том, как вывести исследования в области искусственного интеллекта из их привычной колеи. Почему бы не попытаться создать искусственный мозг, копируя работу биологического мозга? Реального мозга с нейронами вместо логических вентилей, наделенного способностью делать умозаключения, а не управляемого традиционными жестко закодированными черно-белыми абсолютистскими алгоритмами.

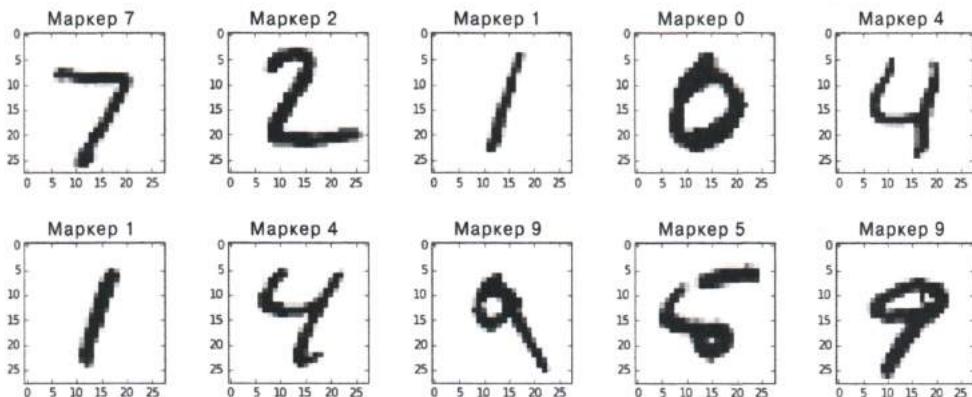
Ученых вдохновляла видимая простота мозга пчел или попугаев по сравнению со сложностью тех задач, которые они могли решать. Мозг весом не более долей грамма демонстрировал способность управлять полетом и адаптироваться к ветру, распознавать пищу и хищников и быстро принимать решения относительно того, стоит ли вступить в схватку или лучше обратиться в бегство. Исследователей интересовало, смогут ли компьютеры с их колоссальными электронными ресурсами имитировать работу такого мозга и даже достичь большего. Если мозг пчелы насчитывает примерно 950 тысяч нейронов, то смогут ли современные компьютеры с их ресурсами памяти, исчисляемыми гигабайтами и терабайтами, превзойти пчел?

Но при использовании традиционных подходов к решению проблем даже суперкомпьютеры с огромным объемом памяти и сверхбыстрыми процессорами не могли обеспечить то, на то способен мозг птицы или пчелы.

Идея проектирования интеллектуальных вычислительных устройств по образу и подобию биологических систем привела к созданию теории **нейронных сетей**, ставшей одним из самых мощных и полезных подходов к разработке искусственного интеллекта. Если говорить о сегодняшних достижениях, то, например, нейронные сети являются основным направлением деятельности компании Deepmind (ныне собственность компании Google), добившейся таких фантастических успехов, как создание нейронной сети, способной учиться играть в видеоигры, и еще одной, которая смогла победить в невероятно сложной игре го гроссмейстера мирового уровня. Нейронные

сети уже составляют саму сердцевину многих повседневных технологий, таких как системы автоматического распознавания автомобильных номеров или системы считывания почтовых индексов, написанных от руки.

В книге я расскажу вам о нейронных сетях, о том, как они работают и как создать собственную нейронную сеть, способную научиться распознавать рукописные символы (задача, решить которую в рамках традиционных компьютерных подходов очень трудно).



ГЛАВА 1

Как работают нейронные сети

Черпайте вдохновение в окружающих вас мелочах.

Что легко одному, трудно другому

Компьютеры, в сущности, — это не более чем калькуляторы, способные выполнять арифметические операции с огромной скоростью.

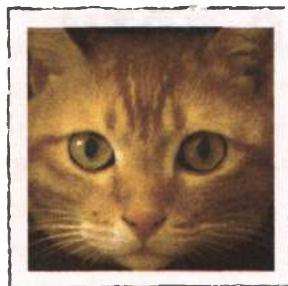
Эта особенность компьютеров позволяет им отлично справляться с задачами, аналогичными тем, которые решаются с помощью калькуляторов: суммирование чисел с целью определения объемов продаж, применение процентных ставок для начисления налогов или построение графиков на основе существующих данных.

Даже просмотр телевизионных программ или прослушивание потоковой музыки через Интернет с помощью компьютера не требует чего-то большего, чем многократное выполнение простых арифметических операций. Возможно, вы будете удивлены, но реконструкция видеокадра, состоящего сплошь из единиц и нулей, которые поступают на ваш компьютер по сети, осуществляется путем выполнения арифметических действий, лишь ненамного более сложных, чем суммирование чисел, которое вы проходили в школе.

Безусловно, сложение чисел с гигантской скоростью — тысячи или миллионы операций в секунду — это впечатляющий эффект, но его нельзя назвать проявлением искусственного интеллекта. Даже если человеку трудно складывать в уме большие числа, данный процесс все же не требует особого интеллекта. Для таких вычислений достаточно способности следовать элементарным инструкциям, и именно это происходит внутри любого компьютера.

А теперь перевернем все вверх тормашками и поставим столы на компьютеры!

Взгляните на приведенные ниже иллюстрации и убедитесь в том, что для вас не составляет труда распознать то, что на них изображено.



Мы с вами, посмотрев на эти фотографии, легко определим, что на них изображены соответственно люди, кот и дерево. Мы способны практически мгновенно и с высокой точностью распознавать объекты, на которые направляем свой взгляд, и при этом очень редко ошибаемся.

В процессе анализа изображений и классификации объектов наш мозг обрабатывает огромные объемы информации. Компьютеру трудно решать подобные задачи, а точнее — невероятно трудно.

Задача	Компьютер	Человек
Быстрое умножение тысяч больших чисел	Легко	Трудно
Распознавание конкретного человека среди толпы на фотографии	Трудно	Легко

Мы догадываемся, что для распознавания образов требуется человеческий интеллект — то, чего недостает машинам, какими бы сложными и мощными мы их ни создавали, а все потому, что они — не люди.

Но это как раз тот тип задач, в отношении которых мы и хотели бы сделать компьютеры более эффективными, поскольку они работают быстрее и никогда не устают. В свою очередь, именно для решения подобных задач и ведутся работы по созданию искусственного интеллекта.

Конечно же, компьютеры всегда будут начинаться электроникой, и потому задачей искусственного интеллекта является поиск предписаний, или **алгоритмов**, основанных на новых подходах к решению трудных задач, о которых идет речь.

Резюме

- Одни задачи, как, например, перемножение миллионов пар чисел, просты для компьютера, но трудны для человека.
- Но есть и такие задачи, как, к примеру, распознавание лиц людей в толпе на фотографии, которые трудны для компьютера, но просты для человека.

Простая прогнозирующая машина

Начнем с простого и будем постепенно усложнять задачу.

Вообразите простую прогнозирующую машину (предиктор), которая получает вопрос, совершает некий “мыслительный” процесс и выдает ответ. Все происходит примерно так, как в приведенном выше примере с распознаванием образов, в котором входная информация воспринималась нашими глазами, далее наш мозг анализировал изображение, после чего мы делали выводы относительно того, какие объекты имеются на данном изображении. Это можно представить с помощью следующей схемы.



Но компьютеры не могут по-настоящему думать (вспомните, что они всего лишь усовершенствованные калькуляторы), поэтому мы используем другую терминологию, более точно соответствующую тому, что происходит на самом деле.



Компьютер получает входную информацию, выполняет некоторые расчеты и выдает результат. Этот процесс схематически представлен на следующей иллюстрации. Входная информация, заданная в виде “ 3×4 ”, обрабатывается с возможной заменой операции умножения более простыми операциями сложения, и выдается выходной результат “12”.



Возможно, вы подумали: “Ну и что здесь особенного?” Не переживайте, все нормально. Пока что мы используем простые и хорошо знакомые примеры для введения понятий, которые далее будут применены к более интересным нейронным сетям.

Давайте чуть усложним задачу.

Представьте, что машина должна преобразовывать километры в мили.



Предположим, что формула, преобразующая километры в мили, нам неизвестна. Все, что мы знаем, — это то, что данные единицы измерения связаны между собой **линейной** зависимостью. Это означает, что если мы удвоим количество миль, то количество километров, соответствующее данному расстоянию, также удвоится. Такая зависимость воспринимается нами интуитивно.

Существование линейного соотношения между километрами и милями дает нам ключ к разгадке формулы для вычислений. Она должна иметь следующий вид: мили = километры \times **c**, где **c** — константа, величину которой мы пока что не знаем.

Единственными дополнительными подсказками нам могут служить отдельные примеры правильного выражения расстояний в километрах и милях. Эти примеры будут выступать как бы в роли экспериментальных данных, отражающих истинное положение вещей, которые мы используем для проверки своей научной теории.

Истинный пример	Километры	Мили
1	0	0
2	100	62,37

Что нам нужно сделать для того, чтобы определить недостающую величину константы? Давайте просто подставим в формулу какое-либо **случайное** значение! Например, предположим, что **c=0,5**, и посмотрим, что при этом произойдет.

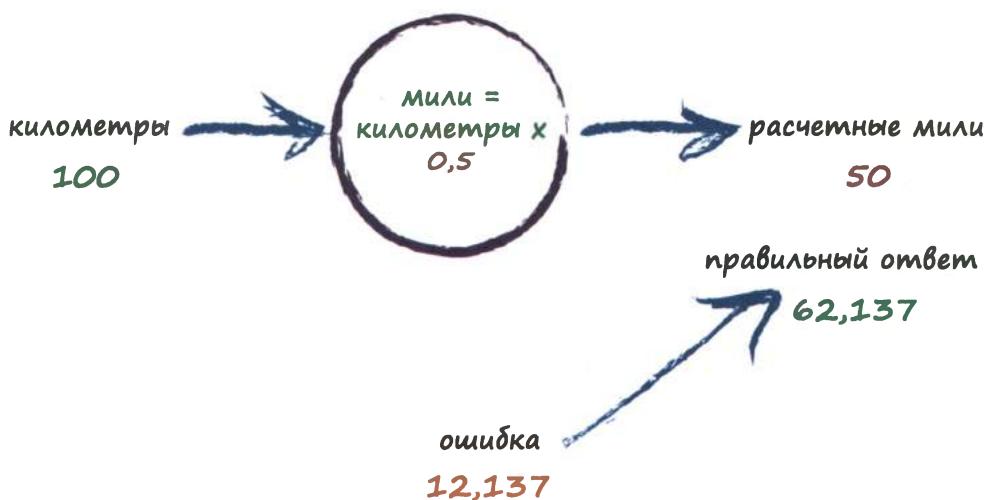


Здесь мы подставляем в формулу $\text{мили} = \text{километры} \times \mathbf{c}$ значение 100 вместо **километры** и текущее пробное значение 0,5 вместо константы **c**. В результате мы получаем ответ: **50 миль**.

Ну хорошо. Это вовсе неплохо, если учесть, что значение $c=0,5$ было выбрано случайным образом! Но мы знаем, что оно не совсем точное, поскольку пример 2 истинного соотношения говорит нам о том, что правильный ответ — 62,137.

Мы ошиблись на 12,137. Это число представляет величину ошибки, т.е. разность между истинным значением из нашего списка примеров и расчетным значением.

$$\begin{aligned}\text{ошибка} &= \text{истина} - \text{расчет} \\ &= 62,137 - 50 \\ &= 12,137\end{aligned}$$



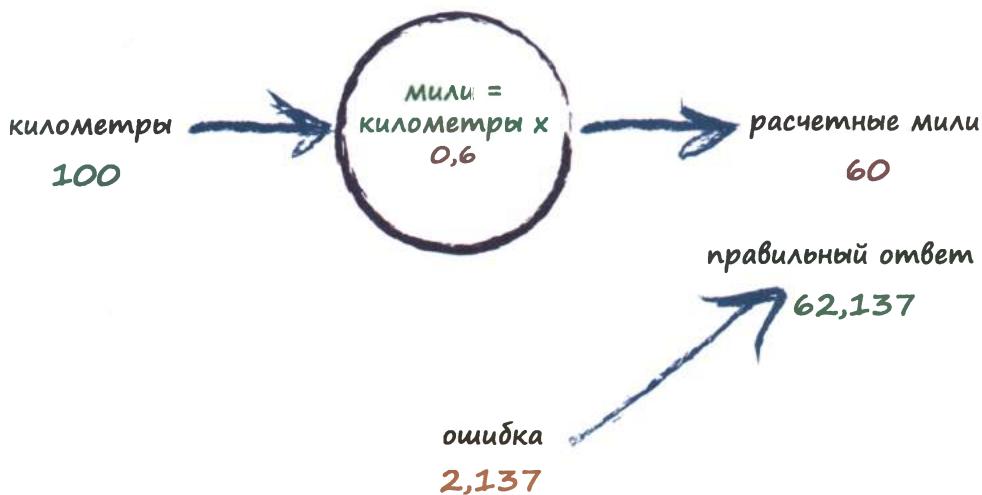
Что дальше? Мы знаем, что ошиблись, и нам известна величина ошибки. Вместо того чтобы видеть в этой ошибке повод для отчаяния, мы используем эту информацию для того, чтобы предложить более удачное пробное значение константы c , чем первое.

Вернемся к ошибке. Мы ошиблись на 12,137 в сторону меньших значений. Так как формула для преобразования километров в мили линейная, мили = километры × c , мы знаем, что увеличение c приведет к увеличению результирующего значения.

Давайте немного подправим c , заменив значение 0,5 значением 0,6, и посмотрим, к чему это приведет.

Приняв для c значение 0,6, мы получаем мили = километры $\times c = 100 \times 0,6 = 60$. Это уже лучше, чем предыдущий ответ — 50. Налицо явный прогресс!

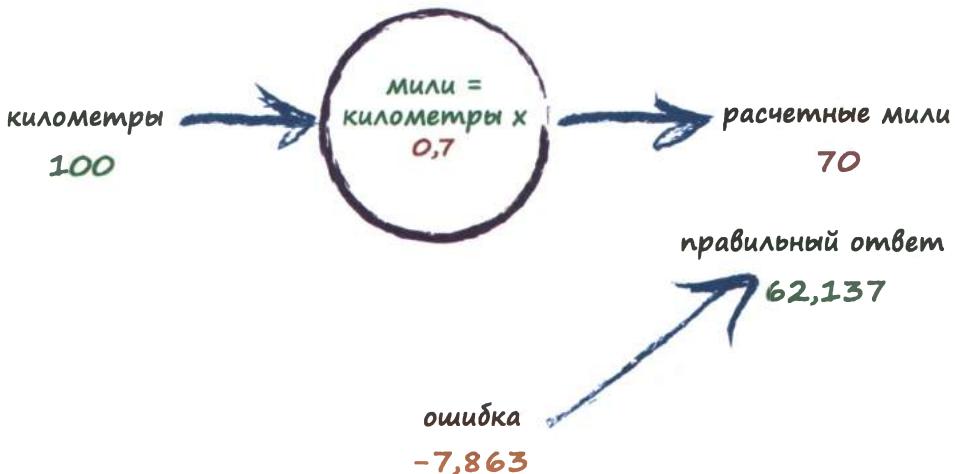
Теперь ошибка уменьшилась до 2,137. Вполне возможно, что с такой ошибкой мы могли бы даже смириться.



Здесь важно то, что величина ошибки подсказала нам, в каком направлении следует откорректировать величину c . Мы хотели увеличить выходной результат 50, поэтому немного увеличили c .

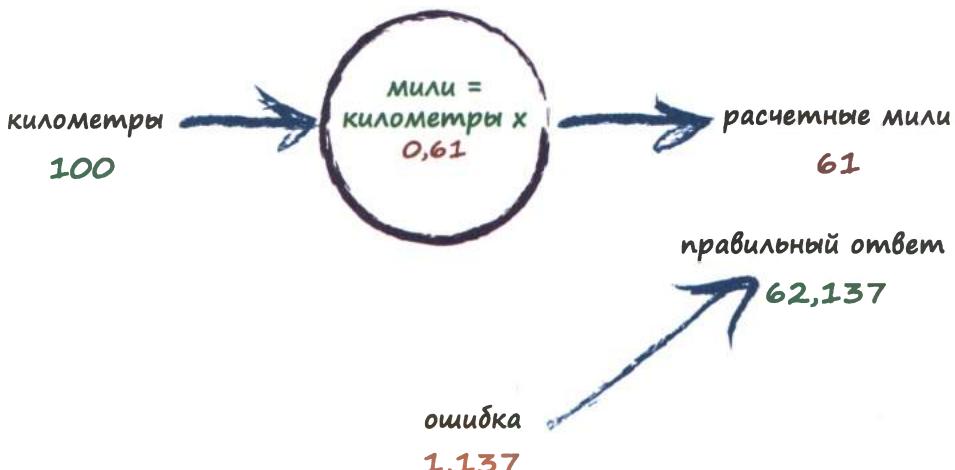
Вместо того чтобы использовать алгебру для нахождения точной величины поправки, на которую следует изменить значение c , мы продолжим использовать подход, заключающийся в постепенном уточнении значения этой константы. Если вы не уверены в правильности такого подхода и считаете, что было бы намного проще сразу определить точный ответ, то имейте в виду, что существует множество более интересных задач, для которых не существует простых математических формул, связывающих между собой входные и выходные значения. Именно поэтому нам и нужны более сложные методы наподобие нейронных сетей.

Повторим уже знакомые нам действия. Выходной результат 60 все еще слишком мал. Давайте вновь немного изменим константу c , увеличив ее значение с 0,6 до 0,7.



О, нет! Мы перестарались и получили результат, превышающий правильный ответ. Предыдущая ошибка была равна 2,137, а теперь она составляет $-7,683$. Знак “минус” просто свидетельствует о том, что вместо недооценки истинного результата произошла его переоценка (напомню, что величина ошибки определяется выражением *правильное значение минус расчетное значение*).

Итак, $c=0,6$ было гораздо лучше, чем $c=0,7$. Сейчас мы могли бы признать величину ошибки при $c=0,6$ удовлетворительной и закончить это упражнение. Но мы все-таки продвинемся еще чуть дальше. Почему бы нам не попытаться ввести очень малую поправку и увеличить значение c с 0,6 до, скажем, 0,61?



Это дает нам гораздо лучший результат, чем предыдущие, поскольку теперь выходное значение 61 отличается от правильного значения 62,137 всего лишь на 1,137.

Итак, последняя попытка научила нас тому, что величину поправки к величине **c** необходимо каждый раз определять заново. Если выходной результат приближается к правильному ответу, т.е. если ошибка уменьшается, то не следует оставлять величину поправки прежней. Тем самым мы избегаем переоценки значения по сравнению с истинным, как это было ранее.

Опять-таки, не отвлекаясь на поиск точных способов определения величины **c** и по-прежнему фокусируя внимание на идее ее постепенного уточнения, мы можем предположить, что поправка должна выражаться некоторой долей ошибки. Это интуитивно понятно: большая ошибка указывает на необходимость введения большей поправки, тогда как малая ошибка нуждается в незначительной поправке.

Хотите — верьте, хотите — нет, но то, что мы сейчас сделали, передает суть процесса обучения нейронной сети. Мы тренировали машину так, чтобы ее предсказания становились все более и более точными.

Нам стоит сделать небольшую паузу, чтобы поразмышлять над следующим: мы не находили точного решения задачи в один прием, как это часто делается при решении школьных или научных задач. Вместо этого мы предприняли совершенно иной подход, заключающийся в многократных попытках проверки пробного значения и его уточнения. Такие процессы иногда называют **итеративными**, что как раз и означает постепенное, шаг за шагом, улучшение искомого результата.

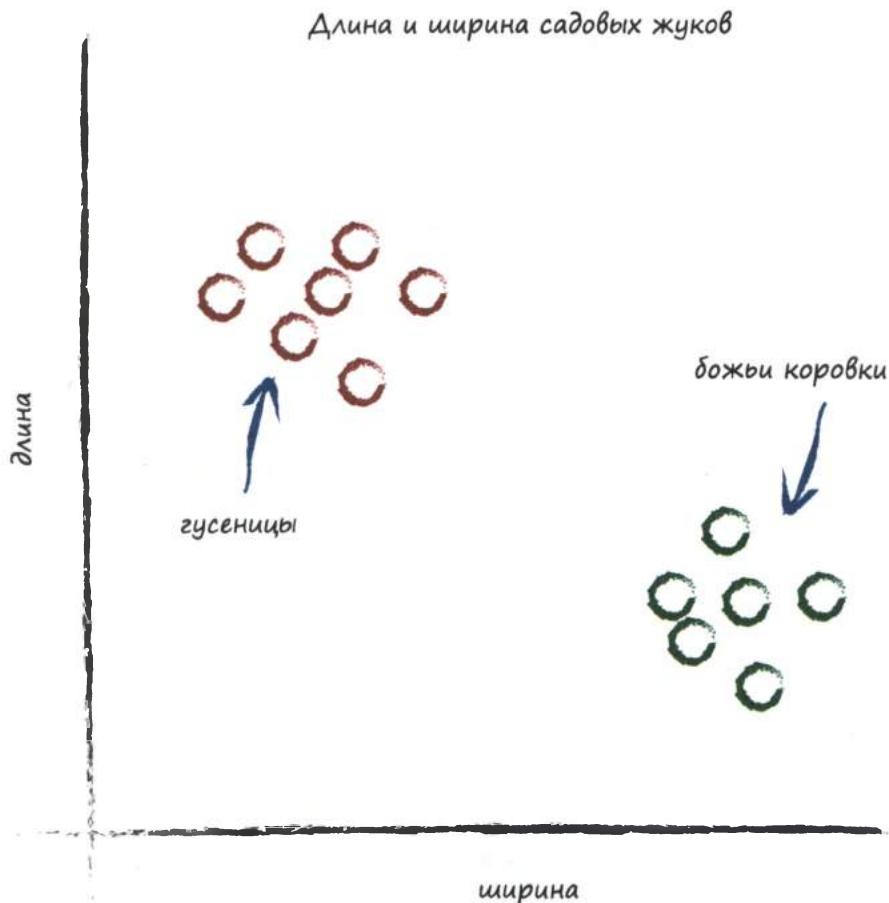
Резюме

- У всех полезных компьютерных систем имеются каналы ввода и вывода, между которыми над данными выполняются некоторые вычисления. В случае нейронных сетей это не так.
- Если точные принципы функционирования какой-либо системы нам неизвестны, то мы пытаемся получить представление о том, как она работает, используя модель с регулируемыми параметрами. Если бы мы не знали, как преобразовать километры в мили, то могли бы использовать для этой цели линейную функцию в качестве модели с регулируемым наклоном.
- Неплохим способом улучшения подобных моделей является настройка параметров на основании сравнения результатов модели с точными результатами в известных примерах.

Задачи классификации и прогнозирования очень близки

Мы назвали описанную ранее простую машину прогнозирующей, поскольку она получает входные данные и делает определенный прогноз относительно того, какими должны быть выходные данные. Мы улучшали прогнозы, регулируя внутренний параметр на основании величины ошибки, которую определяли, сравнивая прогноз с известным точным значением.

Взгляните на иллюстрацию ниже, на которой приведена диаграмма, представляющая результаты измерения размеров садовых жуков.

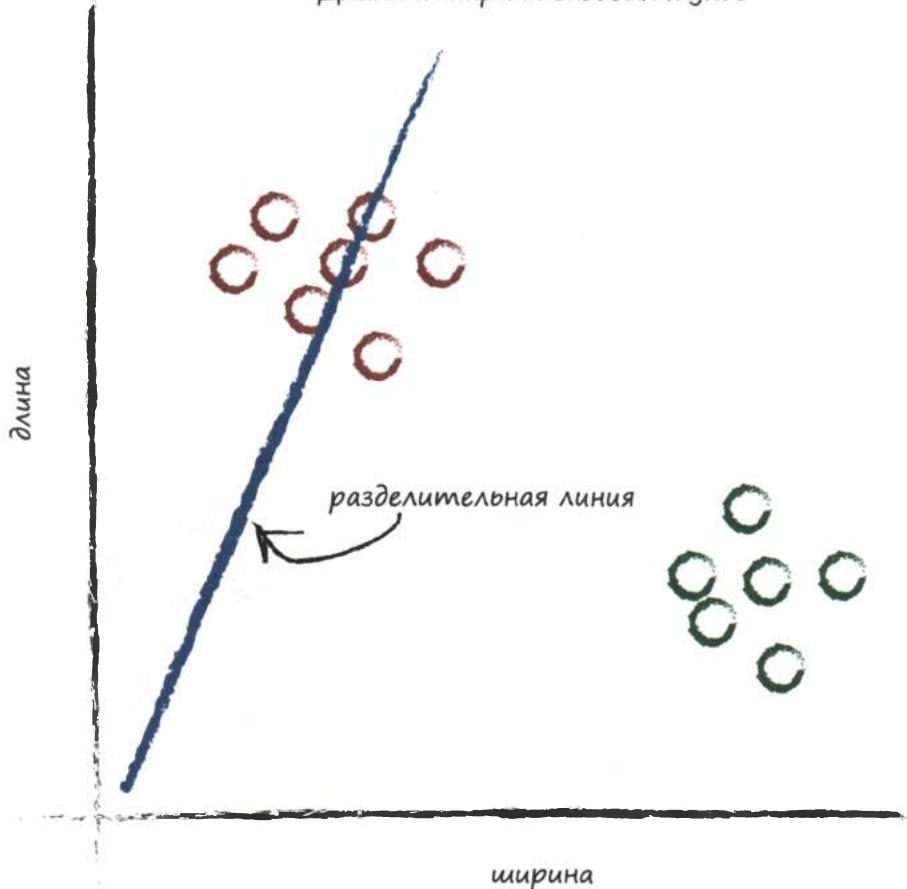


На диаграмме отчетливо видны две группы данных. Гусеницы уже и длиннее, а божьи коровки шире и короче.

Помните наш предиктор, который пытался правильно вычислить количество миль, соответствующее заданному количеству километров? В основу этого предиктора была положена настраиваемая линейная функция. Надеюсь, вы не забыли, что график зависимости выходных значений линейной функции от ее входных значений представляет собой прямую линию. Изменение настраиваемого параметра с приводит к изменению крутизны наклона этой прямой линии.

Что получится, если мы наложим на этот график прямую линию?

Длина и ширина садовых жуков

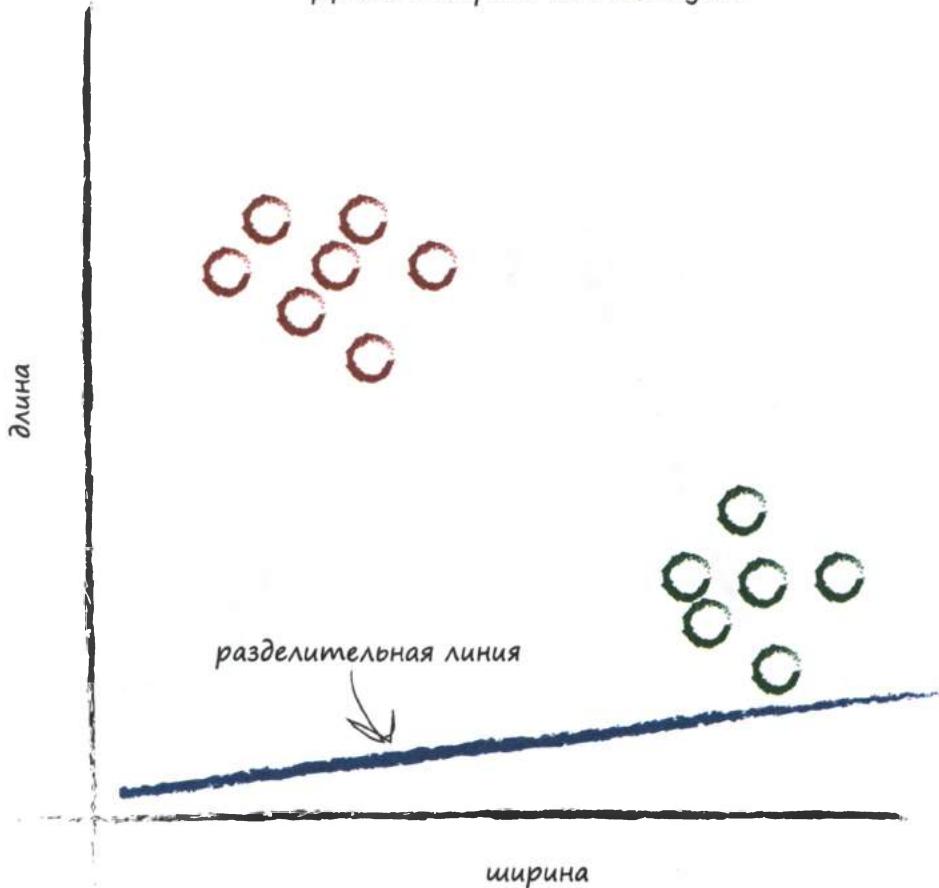


Мы не можем использовать прямую линию точно так же, как раньше, когда преобразовывали одно число (километры) в другое (мили), но, возможно, в данном случае нам удастся отделить с ее помощью один тип данных от другого.

Если бы на показанном выше графике прямая линия отделяла гусениц от божьих коровок, то мы могли бы воспользоваться ею для классификации неизвестных жуков, исходя из результатов измерений. Однако имеющаяся линия не справляется с этой задачей, поскольку половина гусениц находится по ту же сторону линии, что и божьи коровки.

Давайте проведем другую линию, изменив наклон, и посмотрим, что при этом произойдет.

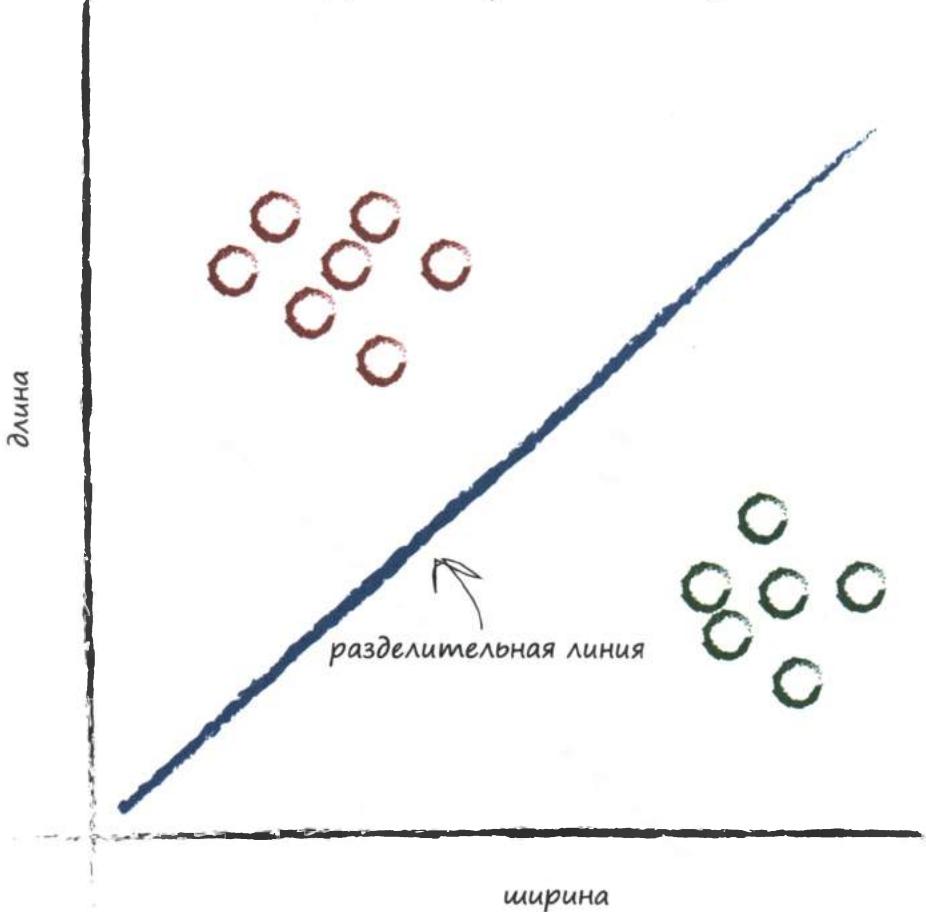
Длина и ширина садовых жуков



На этот раз линия оказалась еще менее полезной, поскольку вообще не отделяет один вид жуков от другого.

Сделаем еще один заход.

Длина и ширина садовых жуков

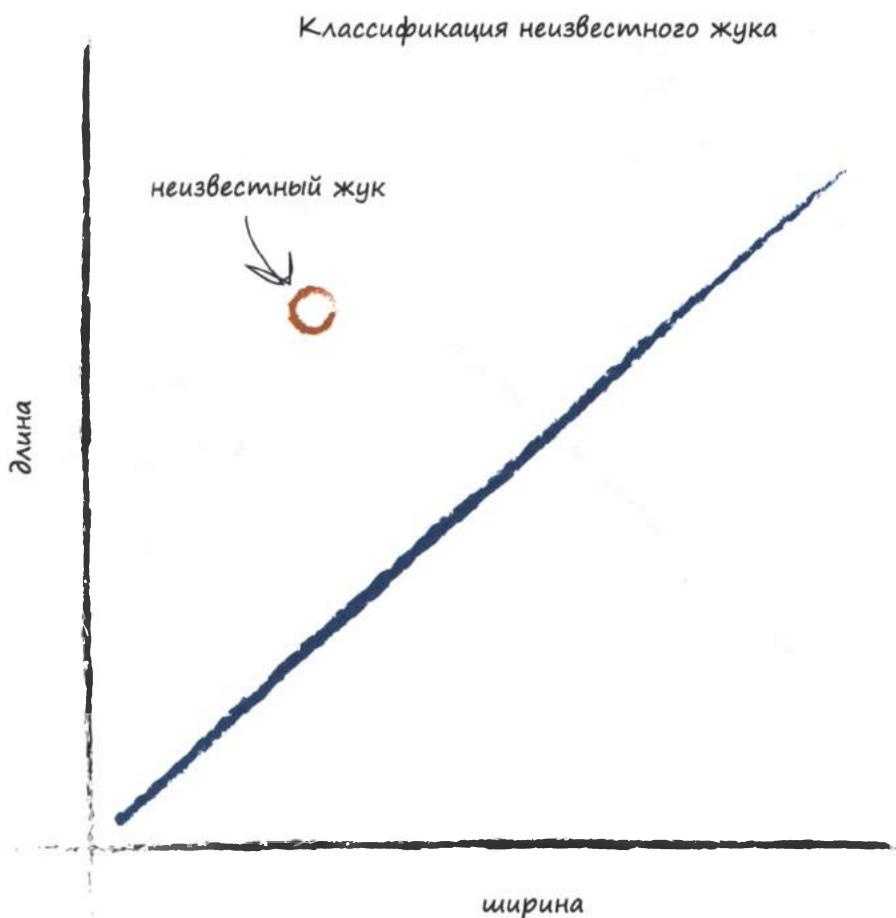


Вот это другое дело! Линия отчетливо отделяет гусениц от божьих коровок. Теперь мы можем использовать ее в качестве классификатора жуков.

Мы предполагаем, что не существует никаких других видов жуков, кроме тех, которые показаны на диаграмме, но на данном этапе это не является недостатком подхода — мы просто пытаемся проиллюстрировать суть идеи простого классификатора.

А теперь представьте, что в следующий раз наш компьютер использует робота для того, чтобы тот отобрал очередного жука и выполнил необходимые замеры. Тогда полученную линию можно будет использовать для корректного отнесения жука к семейству гусениц или семейству божьих коровок.

Взглянув на следующий график, вы увидите, что неизвестный жук относится к семейству гусениц, поскольку попадает в область над линией. Несмотря на свою простоту эта классификация уже является довольно мощным инструментом!



Только что вы имели возможность убедиться в том, насколько полезными могут быть предикторы с линейной функцией в качестве инструмента классификации вновь поступающих данных.

Однако мы обошли вниманием один существенный момент. Как нам узнать, какой наклон прямой является подходящим? Как улучшить линию, если она не разделяет должным образом две разновидности жуков?

Ответ на этот вопрос также имеет самое непосредственное отношение к способности нейронной сети обучаться, к рассмотрению чего мы и переходим.

Тренировка простого классификатора

Сейчас мы займемся тренировкой (обучением) нашего линейного классификатора и научим его правильно классифицировать жуков, относя их к гусеницам или божьим коровкам. Как вы видели ранее, речь идет об уточнении наклона разграничительной линии, отделяющей на графике одну группу точек данных, соответствующих парам значений длины и ширины, от другой.

Как мы это сделаем?

Вместо того чтобы заранее разработать подходящую научную теорию, мы нащупаем правильный путь методом проб и ошибок. Это позволит нам лучше понять математику, скрытую за этими действиями.

Нам нужны примеры для тренировки классификатора. Чтобы не усложнять себе жизнь, мы ограничимся двумя простыми примерами, приведенными ниже.

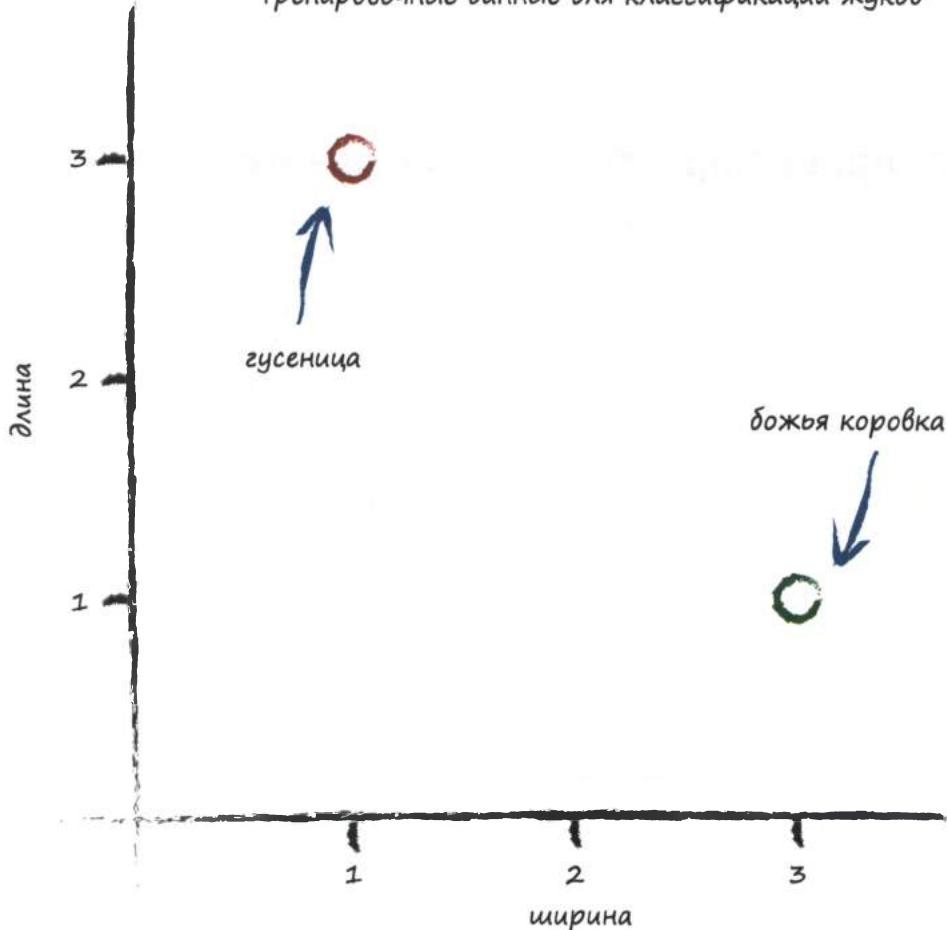
Пример	Ширина	Длина	Жук
1	3,0	1,0	Божья коровка
2	1,0	3,0	Гусеница

У нас есть пример жука, имеющего ширину 3,0 и длину 1,0, который, как нам известно, является божьей коровкой. Второй пример относится к жуку, имеющему большую длину — 3,0 и меньшую ширину — 1,0, которым является гусеница.

Мы знаем, что данные в этом наборе примеров являются истинными. Именно с их помощью будет уточняться значение константы в функции классификатора. Примеры с истинными значениями, которые используются для обучения предиктора или классификатора, называют **тренировочными данными**.

Отобразим эти два примера тренировочных данных на диаграмме. Визуализация данных часто помогает лучше понять их природу, почувствовать их, чего нелегко добиться, просто взглянувши в список или таблицу, заполненную числами.

Тренировочные данные для классификации жуков



Начнем со случайной разделительной линии, потому что нужно ведь с чего-то начинать. Вспомните линейную функцию из примера с преобразованием километров в мили, параметр которой мы настраивали. Мы можем сделать то же самое и сейчас, поскольку разделительная линия в данном случае прямая:

$$y = Ax$$

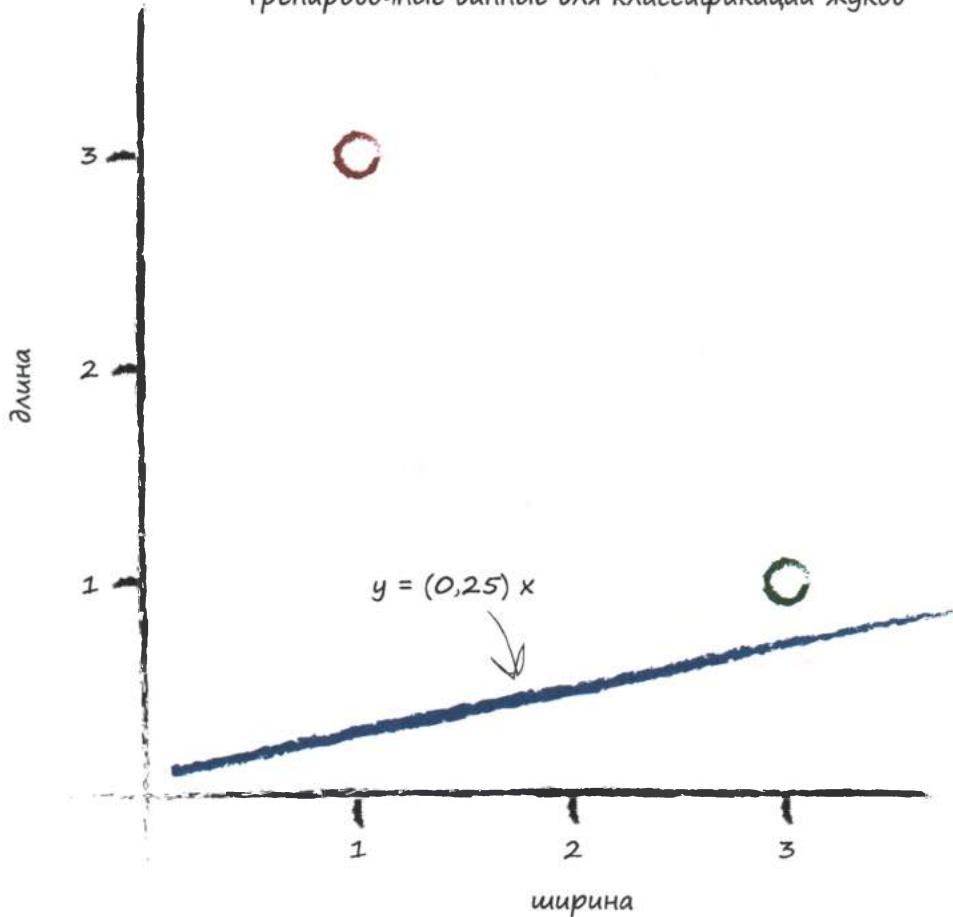
Мы намеренно используем здесь имена x и y , а не длина и ширина, поскольку, строго говоря, в данном случае линия не служит предиктором. Она не преобразует ширину в длину, как ранее километры переводились в мили. Вместо этого она выступает в качестве разделительной линии, классификатора.

Вероятно, вы обратили внимание на неполную форму уравнения $y = Ax$, поскольку полное уравнение прямой линии имеет следующий вид: $y = Ax + B$. Мы намеренно делаем этот сценарий с садовыми жуками максимально простым. Ненулевое значение B просто соответствует линии, которая не проходит через начало координат на диаграмме, что не добавляет в наш сценарий ничего нового.

Ранее было показано, что параметр A управляет наклоном линии. Чем больше A , тем больше крутизна наклона.

Для начала примем, что $A=0,25$. Тогда разделительная линия описывается уравнением $y=0,25x$. Отобразим эту линию в графическом виде на той же диаграмме, на которой отложены тренировочные данные.

Тренировочные данные для классификации жуков



Благодаря графику мы безо всяких вычислений сразу же видим, что линия $y=0,25x$ не является хорошим классификатором. Она не отделяет один тип жуков от другого. Например, мы не можем делать утверждения наподобие “Если точка данных располагается над линией, то она соответствует гусенице”, поскольку точно там же располагаются и данные божьей коровки.

Интуитивно мы понимаем, что правый край линии следует немногоХ приподнять. Мы устоим перед соблазном сделать это, глядя на диаграмму и проводя подходящую линию вручную. Мы хотим проверить, не удастся ли нам подобрать для этого подходящий повторяющийся рецепт, последовательность команд, которую компьютерщики называют алгоритмом.

Обратимся к первому тренировочному примеру, соответствующему божьей коровке: ширина — 3,0 и длина — 1,0. Если бы мы тестировали функцию $y=Ax$ с этим примером, в котором x равен 3,0, то получили бы следующий результат:

$$y = (0,25) * (3,0) = 0,75$$

Функция, в которой для параметра A установлено начальное случайноХ выбранное значение, равное 0,25, сообщает, что для жука шириной 3,0 длина должна быть равна 0,75. Мы знаем, что это слишком мало, поскольку согласно тренировочным данным длина жука равна 1,0.

Итак, налицо расхождение, или ошибка. Точно так же, как в примере с предиктором, преобразующим километры в мили, мы можем использовать величину этой ошибки для получения информации о том, каким образом следует корректировать параметр A .

Однако сначала давайте подумаем, каким должно быть значение y . Если положить его равным 1,0, то линия пройдет через точку с координатами $(x,y) = (3,0; 1,0)$, соответствующую божьей коровке. Само по себе это неплохо, но это не совсем то, что нам нужно. Нам желательно, чтобы линия проходила над этой точкой. Почему? Да потому, что мы хотим, чтобы точки данных божьей коровки лежали под линией, а не на ней. Линия должна служить разделителем между точками данных божьих коровок и гусениц, а не предсказывать длину жука по известной ширине.

В связи с этим попробуем нацелиться на значение $y=1,1$ при $x=3,0$. Оно лишь ненамного больше 1,0. Вместо него можно было бы взять значение 1,2 или 1,3, но никак не 10 или 100, поскольку с большой долей вероятности это приведет к тому, что прямая будет проходить над всеми точками данных, как божьих коровок, так и гусениц, в результате чего она станет полностью бесполезной в качестве разделителя.

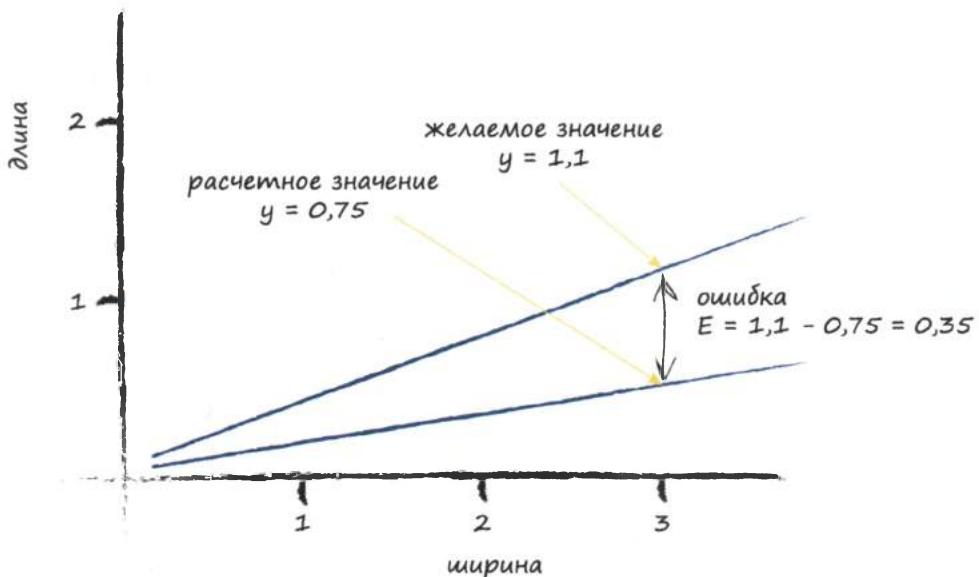
Следовательно, мы выбираем целевое значение 1,1 и определяем ошибку E с помощью следующей формулы:

$$\text{ошибка} = \text{желаемое целевое значение} - \text{фактический результат}$$

Или (после подстановки значений):

$$E = 1,1 - 0,75 = 0,35$$

Помня о пользе визуализации информации, обратимся к приведенной ниже диаграмме, на которой в графическом виде представлены ошибки, а также целевое и расчетное значения.



Вы спросите: а каким образом наше знание величины ошибки E может помочь в нахождении лучшего значения для параметра A ? Это очень важный вопрос.

Давайте на время отступим от этой задачи и немного порассуждаем. Мы хотим использовать ошибку в значении y , которую назвали E , для нахождения искомого изменения параметра A . Для этого нам нужно знать, как эти две величины связаны между собой. Каково соотношение между A и E ? Если бы это было нам известно, то мы могли бы понять, как изменение одной величины влияет на другую.

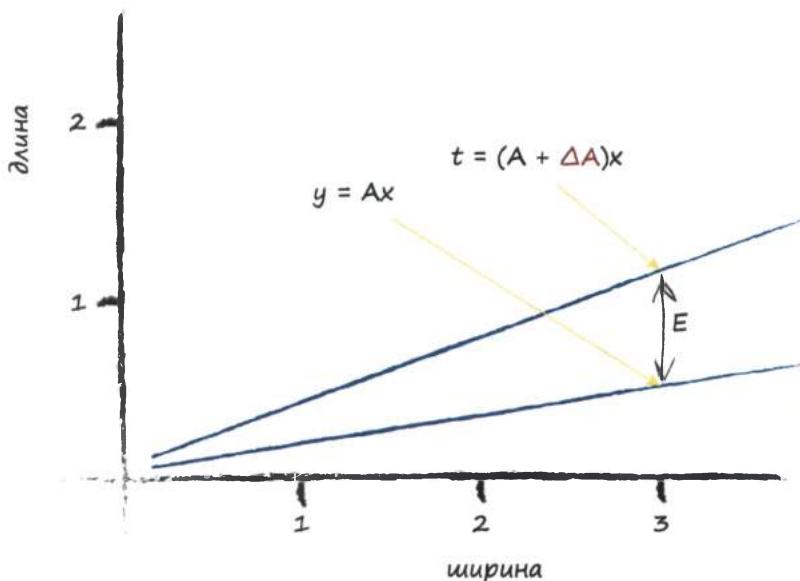
Начнем с линейной функции для классификатора:

$$y = Ax$$

Нам уже известно, что начальные попытки присвоения пробных значений параметру A привели к неверным значениям y , если ориентироваться на тренировочные данные. Пусть t — корректное целевое значение. Чтобы получить его, мы должны ввести в A небольшую поправку. Для таких поправок в математике принято использовать символ Δ , означающий “небольшое изменение”. Запишем соответствующее уравнение:

$$t = (A + \Delta A)x$$

Отобразим это соотношение в графическом виде на диаграмме, на которой показаны линии для двух значений наклона: A и $A + \Delta A$.



Вспомните, что ошибку **E** мы определили как разность между желаемым корректным значением **y** и расчетным значением, полученным для текущего пробного значения **A**. Таким образом, $E = t - y$.

Запишем это в явном виде:

$$t - y = (A + \Delta A)x - Ax$$

Раскрыв скобки и приведя подобные члены, получаем:

$$\begin{aligned} E &= t - y = Ax + (\Delta A)x - Ax \\ E &= (\Delta A)x \end{aligned}$$

Это просто замечательно! Ошибка **E** связана с **ΔA** очень простым соотношением. Оно настолько простое, что поначалу я даже засомневался, не кроется ли где-то ошибка, но оно оказалось действительно верным. Как бы то ни было, это простое соотношение значительно упрощает нашу работу.

Делая подобного рода выкладки, можно легко забыть о первоначальной задаче. Сформулируем простыми словами то, чего мы хотели добиться.

Мы хотели узнать, каким образом можно использовать информацию об ошибке **E** для определения величины поправки к **A**, которая изменила бы наклон линии таким образом, чтобы классификатор лучшеправлялся со своими функциями. Преобразуем последнее уравнение, чтобы найти выражение для **ΔA**:

$$\Delta A = E / x$$

Есть! Это и есть то волшебное выражение, которое мы искали. Теперь мы можем использовать ошибку **E** для изменения наклона классифицирующей линии на величину **ΔA** в нужную сторону.

Примемся за дело — обновим начальный наклон линии.

Когда **x** был равен 3,0, ошибка была равна 0,35. Таким образом, $\Delta A = E/x$ превращается в $0,35 / 3,0 = 0,1167$. Это означает, что текущее значение **A=0,25** необходимо изменить на величину 0,1167. Отсюда следует, что новое, улучшенное значение **A** равно **(A + ΔA)**, т.е. $0,25 + 0,1167 = 0,3667$. Не составляет труда убедиться в том, что

расчетное значение y при новом значении A равно, как и следовало ожидать, $1,1$ — желаемому целевому значению.

Ух ты! У нас получилось! Все работает, и мы располагаем методом для улучшения параметра A , если известна текущая ошибка.

Давайте поднажмем.

Закончив с первым примером, потренируемся на втором. Он дает нам следующие истинные данные: $x=1,0$ и $y=3,0$.

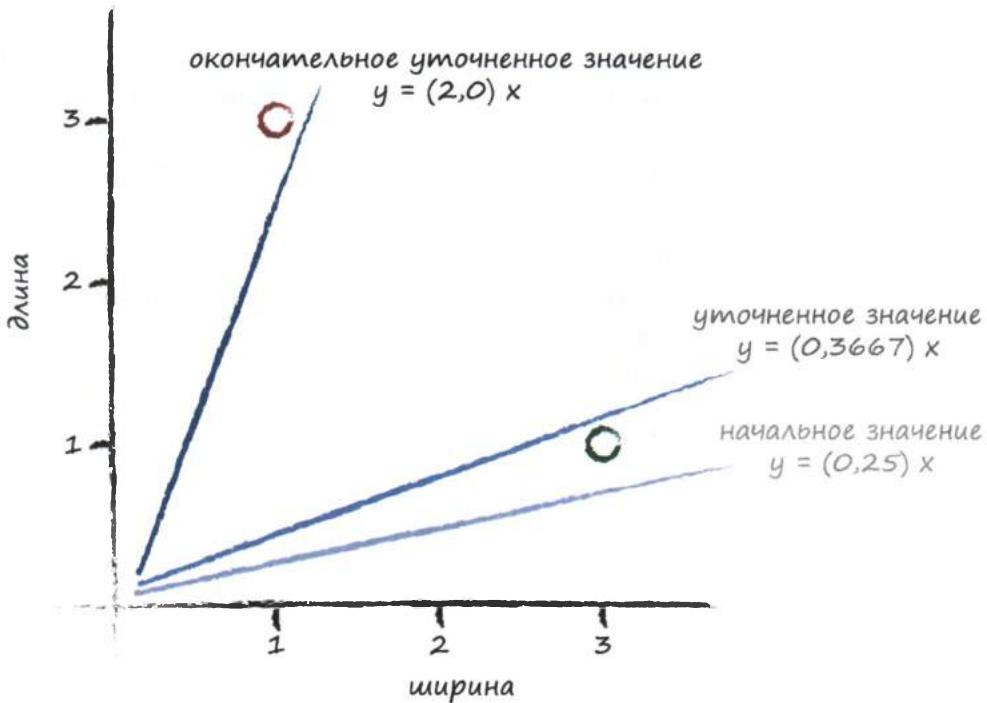
Посмотрим, что получится, если вставить $x=1,0$ в линейную функцию, в которой теперь используется обновленное значение $A=0,3667$. Мы получаем $y = 0,3667 * 1,0 = 0,3667$. Это очень далеко от значения $y=3,0$ в тренировочном примере.

Используя те же рассуждения, что и перед этим, когда мы нашули путь к построению такой линии, которая не пересекала бы тренировочные данные, а проходила над ними или под ними, мы можем задать желаемое целевое значение равным $2,9$. При этом данные тренировочного примера, соответствующего гусеницам, находятся над линией, а не на ней. Ошибка E равна $(2,9 - 0,3667) = 2,5333$.

Эта ошибка больше предыдущей, но если хорошо подумать, то у нас ведь был всего лишь один пример для обучения линейной функции, который отчетливо смешал функцию в своем направлении.

Опять обновим A , как делали до этого. Соотношение $\Delta A = E/x$ дает $2,5333 / 1,0 = 2,5333$. Это означает, что после очередного обновления параметр A принимает значение $0,3667 + 2,5333 = 2,9$. Отсюда следует, что для $x=1,0$ функция возвращает в качестве ответа значение $2,9$, которое и является желаемым целевым значением.

Мы проделали довольно большую работу, поэтому можем снова передохнуть и визуализировать полученные результаты. На следующей диаграмме представлены начальная линия, линия, обновленная после обучения на первом тренировочном примере, и окончательная линия, обновленная на втором тренировочном примере.



Но погодите! Что произошло? Глядя на график, мы видим, что нам не удалось добиться того наклона прямой, которого мы хотели. Она не обеспечивает достаточно надежное разделение областей диаграммы, занимаемых точками данных божьих коровок и гусениц.

Ну что тут сказать? Мы получили то, что просили. Линия обновляется, подстраиваясь под то целевое значение y , которое мы задаем.

Что-то здесь не так? А ведь действительно, если мы будем продолжать так и далее, т.е. просто обновлять наклон для очередного примера тренировочных данных, то все, что мы будем каждый раз получать в конечном счете, — это линию, проходящую вблизи точки данных последнего тренировочного примера. В результате этого мы отбрасываем весь предыдущий опыт обучения, который могли бы использовать, и учимся лишь на самом последнем примере.

Как исправить эту ситуацию?

Легко! И эта идея играет ключевую роль в **машиином обучении**. Мы **сглаживаем обновления**, т.е. немножко уменьшаем величину поправок. Вместо того чтобы каждый раз с энтузиазмом заменять A

новым значением, мы используем лишь некоторую долю поправки ΔA , а не всю ее целиком. Благодаря этому мы движемся в том направлении, которое подсказывает тренировочный пример, но делаем это осторожно, сохраняя некоторую часть предыдущего значения, которое было получено в результате, возможно, многих предыдущих тренировочных циклов. Мы уже видели, как работает эта идея сглаживания в примере с преобразованием километров в мили, когда изменили параметр с лишь на некоторую долю фактической ошибки.

У такого сглаживания есть еще один очень мощный и полезный побочный эффект. Если тренировочные данные не являются надежными и могут содержать ошибки или шум (а в реальных измерениях обычно присутствуют оба этих фактора), то сглаживание уменьшает их влияние.

Ну что ж, сделаем перерасчет, на этот раз добавив сглаживание в формулу обновления:

$$\Delta A = L (E / x)$$

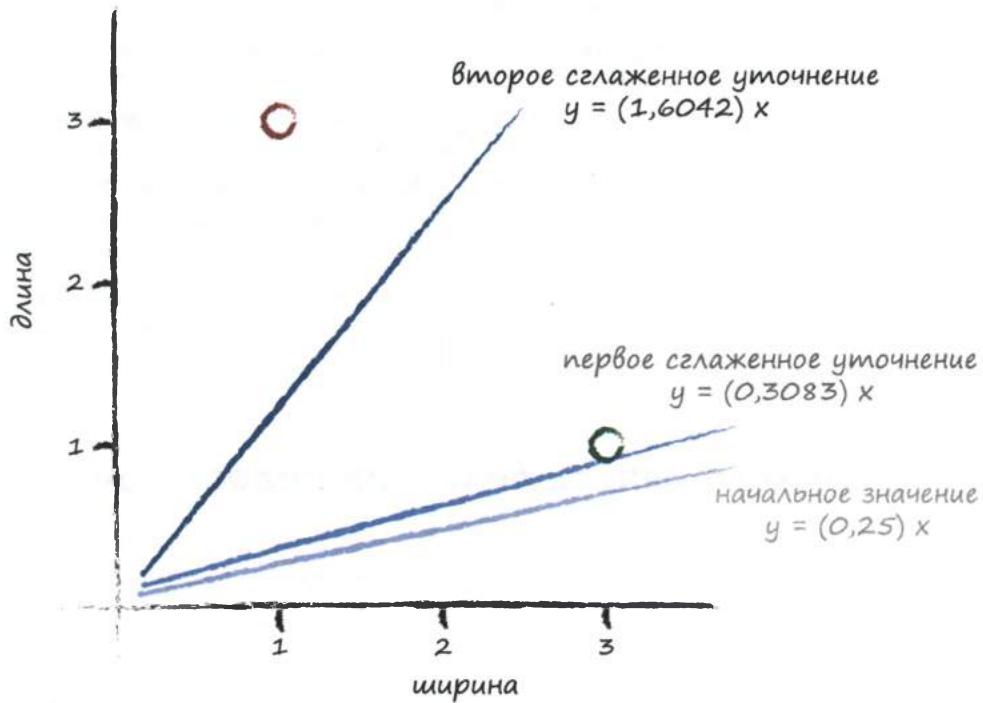
Фактор сглаживания, обозначенный здесь как L , часто называют коэффициентом **скорости обучения**. Выберем $L=0,5$ в качестве разумного начального приближения. Это означает, что мы собираемся использовать поправку вдвое меньшей величины, чем без сглаживания.

Повторим все расчеты, используя начальное значение $A=0,25$. Первый тренировочный пример дает нам $y = 0,25 * 3,0 = 0,75$. При целевом значении 1,1 ошибка равна 0,35. Поправка равна $\Delta A = L (E / x) = 0,5 * 0,35 / 3,0 = 0,0583$. Обновленное значение A равно $0,25 + 0,0583 = 0,3083$.

Проведение расчетов с этим новым значением A для тренировочного примера при $x=3,0$ дает $y = 0,3083 * 3,0 = 0,9250$. Как видим, расположение этой линии относительно тренировочных данных оказалось неудачным — она проходит ниже значения 1,1, но этот результат не так уж и плох, если учесть, что это была всего лишь первая попытка. Главное то, что мы движемся в правильном направлении от первоначальной линии.

Перейдем ко второму набору тренировочных данных при $x=1,0$. Используя $A=0,3083$, мы получаем $y = 0,3083 * 1,0 = 0,3083$. Желаемым значением было 2,9, поэтому ошибка составляет $(2,9 - 0,3083) = 2,5917$. Поправка $\Delta A = L(E/x) = 0,5 * 2,5917 / 1,0 = 1,2958$. Теперь обновленное значение A равно $0,3083 + 1,2958 = 1,6042$.

Вновь отобразим на диаграмме начальный, улучшенный и окончательный варианты линии, чтобы убедиться в том, что сглаживание обновлений приводит к более удовлетворительному расположению разделительной линии между областями данных божьих коровок и гусениц.



Это действительно отличный результат!

Всего лишь с двумя простыми тренировочными примерами и относительно простым методом обновления мы, используя сглаживание скорости обучения, смогли очень быстро получить хорошую разделительную линию $y=Ax$, где $A=1,6042$.

Не будем преуменьшать свои достижения. Нам удалось создать автоматизированный метод обучения классификации на примерах, который, несмотря на простоту подхода, продемонстрировал замечательную эффективность.

Великолепно!

Резюме

- Чтобы понять соотношение между выходной ошибкой линейного классификатора и параметром регулируемого наклона, достаточно владеть элементарной математикой. Зная это соотношение, можно определить величину изменения наклона, необходимую для устранения выходной ошибки.
- Недостаток прямолинейного подхода к регулировке параметров заключается в том, что модель обновляется до наилучшего соответствия только последнему тренировочному примеру, тогда как все предыдущие примеры не принимаются во внимание. Одним из неплохих способов устранения этого недостатка является уменьшение величины обновлений с помощью коэффициента скорости обучения, чтобы ни один отдельно взятый тренировочный пример не доминировал в процессе обучения.
- Тренировочные примеры, взятые из реальной практики, могут быть искажены шумом или содержать ошибки. Сглаживание обновлений способствует ограничению влияния подобных ложных примеров.

Иногда одного классификатора недостаточно

Как вы имели возможность убедиться, рассмотренные нами простые предикторы и классификаторы, относящиеся к числу тех, которые получают некоторые входные данные, выполняют соответствующие вычисления и выдают ответ, довольно эффективны. И все же их возможностей недостаточно для решения более интересных задач, к которым мы намереваемся применять методы нейронных сетей.

Ограничения линейного классификатора продемонстрированы ниже на простом, но весьма показательном примере. Но почему мы должны заниматься этим вместо того, чтобы сразу же перейти к обсуждению нейронных сетей? Дело в том, что от понимания сути указанных ограничений зависит один из ключевых элементов проектирования нейронных сетей, так что этот вопрос стоит того, чтобы его обсудить.

Мы отойдем от темы садовых жучков и обратимся к логическим (булевым) функциям. Не беспокойтесь, если эта терминология вам ни о чем не говорит. Джордж Буль — математик и философ, с именем которого связаны такие простые функции, как логические И и ИЛИ.

Функции булевой логики — это своего рода “мыслительные” функции со своим языком. Если мы говорим “Ты получишь пудинг только в том случае, если уже съел овощи И все равно голоден”, то мы используем булеву функцию И. Результат булевой функции И истинен только тогда, когда истинны (выполняются) оба условия. Он не будет истинным, если истинно только одно из условий. Поэтому, если я голоден, но еще не съел овощи, то не получу свой пудинг.

Аналогично, если мы говорим “Ты можешь погулять в парке, если сегодня выходной ИЛИ у тебя отпуск”, то используем булеву функцию ИЛИ. Результат булевой функции ИЛИ истинен, если истинным является хотя бы одно из условий. Вовсе не обязательно, чтобы все условия были истинными, как в случае функции И. Поэтому, если сегодня не выходной день, но у меня отпуск, то я могу погулять в парке.

Ранее мы представляли функцию в виде машины, которая принимает некоторые входные данные, выполняет определенные действия и выдает один ответ.



Значение **истина** часто представляется в компьютерах как число 1, а значение **ложь** — как число 0. В приведенной ниже таблице результаты работы логических функций И и ИЛИ представлены с использованием этой лаконичной нотации для всех комбинаций входных значений А и В.

Входное значение A	Входное значение B	Логическое И	Логическое ИЛИ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Здесь отчетливо видно, что результат функции И будет истинным только тогда, когда истинны и A, и B.

Булевы функции играют очень важную роль в информатике, и первые электронные вычислительные устройства фактически собирались из крохотных электрических схем, выполняющих логические операции. Даже арифметические операции выполнялись с использованием этих схем, которые сами по себе всего лишь выполняли простые булевые операции.

Представьте себе простой линейный классификатор, который должен использовать тренировочный набор данных для выяснения того, управляются ли данные логической функцией. Задачи такого рода естественным образом возникают перед учеными, исследующими причинно-следственные связи или корреляцию наблюдаемых величин. Например, требуется найти ответ на следующий вопрос: “Повышается ли вероятность заболевания малярией в тех местностях, в которых постоянно идут дожди И температура превышает 35 градусов?” Или такой: “Повышается ли вероятность заболевания малярией, если выполняется любое (булевая функция ИЛИ) из этих условий?”

Взгляните на приведенную ниже диаграмму, где значения на двух входах логической функции, A и B, представляют координаты точек на графике. Вы видите, что только в том случае, когда оба входных значения истинны, т.е. каждое из них равно 1, выходной результат, выделенный зеленым цветом, является истинным. Ложные выходные значения обозначены красным цветом.

логическое И



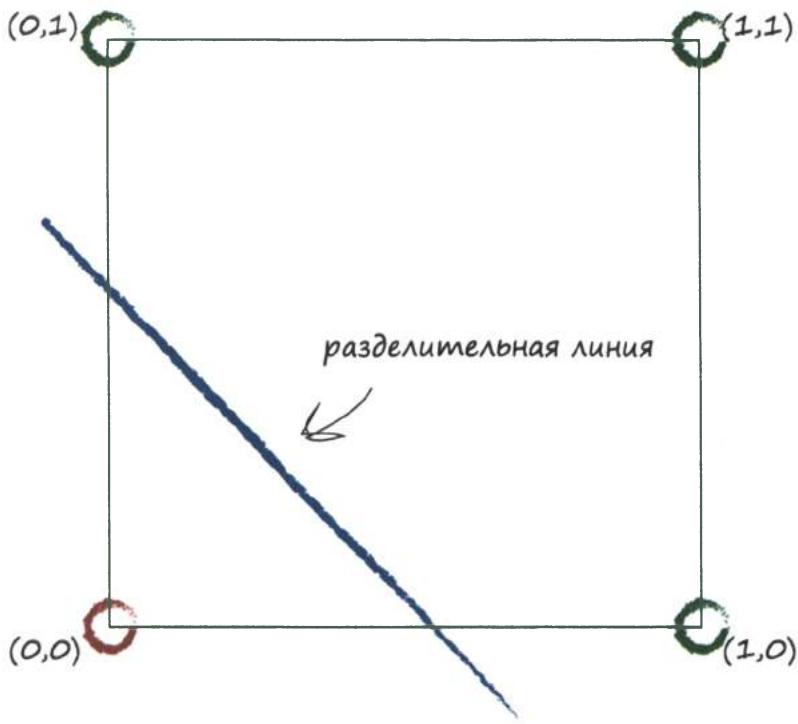
На диаграмме также показана прямая линия, отделяющая красную область от зеленой. Эта линия представляет линейную функцию, которую можно использовать для обучения линейного классификатора, что мы делали ранее.

Мы не будем проводить соответствующие вычисления, как в предыдущих примерах, поскольку ничего принципиально нового это не даст.

В действительности можно было бы предложить много других вариантов проведения разделительной линии, которые работали бы столь же удовлетворительно, но главный вывод из этого примера заключается в том, что простой линейный классификатор вида $y=ax+b$ можно обучить работе с булевой функцией И.

А теперь взгляните на аналогичное графическое представление булевой функции ИЛИ.

логическое ИЛИ

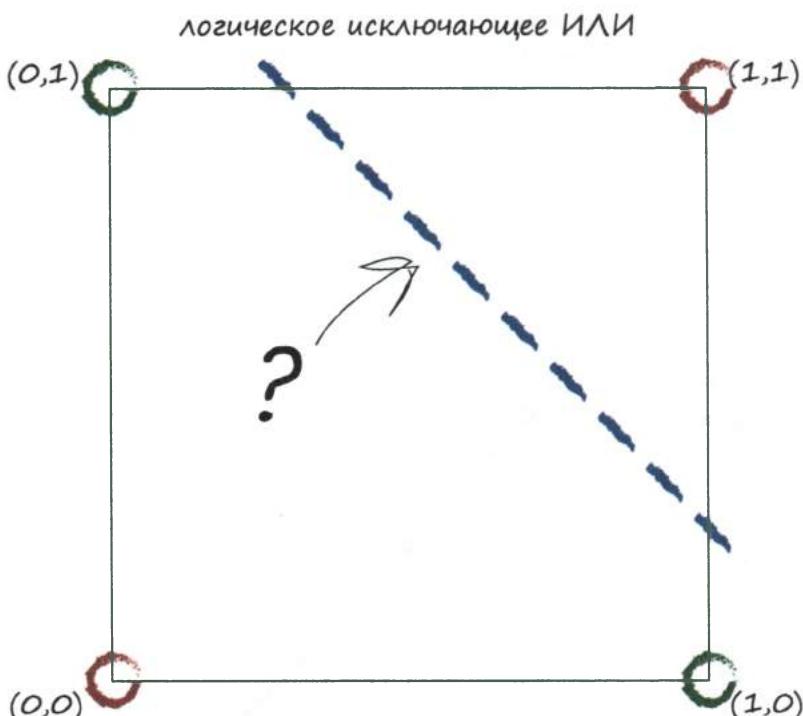


На этот раз красной оказалась лишь точка $(0,0)$, поскольку ей соответствуют ложные значения на обоих входах, А и В. Во всех других комбинациях значений хотя бы одно из них является истинным, и поэтому для них результат является истинным. Вся прелест этой диаграммы заключается в том, что она наглядно демонстрирует возможность обучения линейного классификатора работе с функцией ИЛИ.

Существует еще одна булева функция, исключающее ИЛИ, которая дает истинный результат только в том случае, если лишь одно из значений на входах А и В истинно, но не оба. Таким образом, если оба входных значения ложны или оба истинны, то результат будет ложным. Все вышесказанное резюмировано в приведенной ниже таблице.

Входное значение А	Входное значение В	Исключающее ИЛИ
0	0	0
0	1	1
1	0	1
1	1	0

Взгляните на диаграмму, соответствующую этой функции.



Вот вам и проблема! Мы не видим способа разделить зеленую и красную области одной прямой линией.

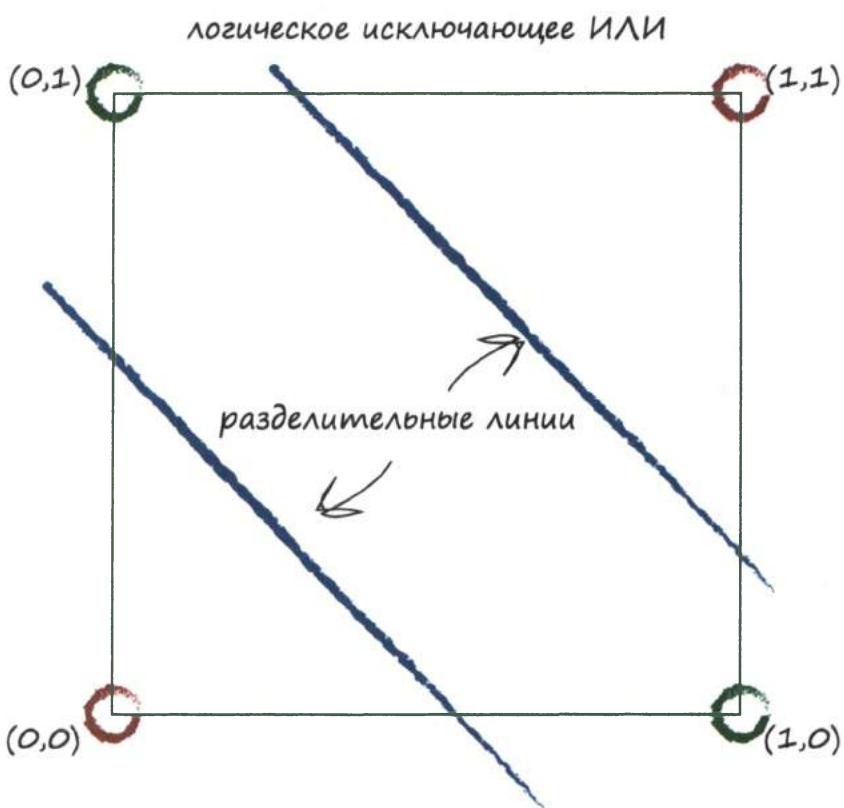
В действительности так оно и есть: невозможно провести одну прямую линию таким образом, чтобы она успешно отделила красные точки данных от зеленых для функции исключающего ИЛИ. Это означает, что простой линейный классификатор не в состоянии обучиться работе с функцией исключающего ИЛИ, если предоставить ему тренировочные данные, управляемые этой функцией.

Только что мы продемонстрировали главное ограничение простого линейного классификатора: такие классификаторы оказываются непригодными, если базовая задача не допускает разделения данных одной прямой линией.

Но ведь мы хотим, чтобы нейронные сети можно было использовать для решения самого широкого круга задач, даже тех, которые не допускают линейного разделения данных.

Следовательно, нам необходимо найти какой-то выход из этой ситуации.

К счастью, такой выход существует: совместное использование нескольких классификаторов. Его иллюстрирует приведенная ниже диаграмма с двумя разделительными линиями. Эта идея занимает центральное место в теории нейронных сетей. Теперь вам должно быть ясно, что с помощью множества прямых линий можно разделить желаемым образом любую сколь угодно сложную конфигурацию областей, подлежащих классификации.



Прежде чем мы займемся созданием нейронных сетей, которые предполагают существование нескольких классификаторов, работающих совместно, снова обратимся к природе и рассмотрим работу мозга животных, аналогии с которым послужили толчком к разработке подходов на основе нейронных сетей.

Резюме

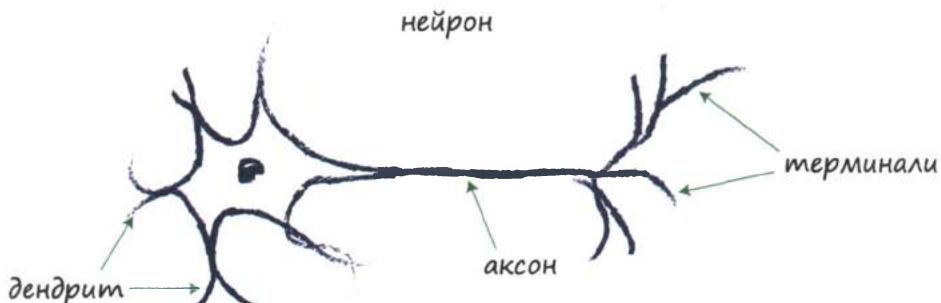
- Простой линейный классификатор не в состоянии разделить нужным образом области данных, если данные не управляются простым линейным процессом. Это было продемонстрировано на примере данных, управляемых логической функцией исключающего ИЛИ.
- Однако эта проблема решается очень просто: для разграничения данных, которые не удается разделить одной прямой линией, следует использовать множество линейных классификаторов.

Нейроны — вычислительные машины, созданные природой

Ранее говорилось о том, что мозг животных ставил ученых в тупик, поскольку даже у столь малых представителей живой природы, как попугай, он демонстрирует несравненно большие способности, чем цифровые компьютеры с огромным количеством электронных вычислительных элементов и памятью невероятных объемов, работающих на частотах, недостижимых для живого мозга из плоти и крови.

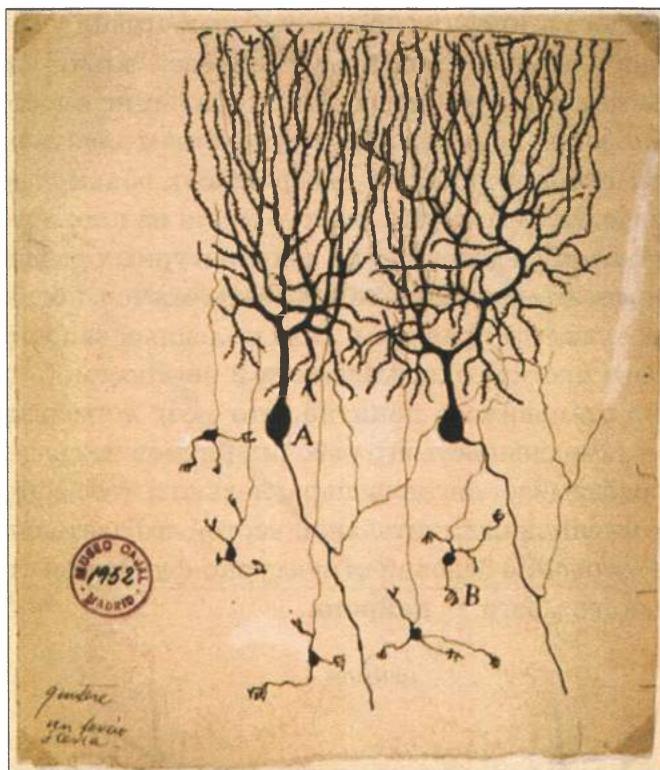
Тогда внимание сосредоточили на архитектурных различиях. В традиционных компьютерах данные обрабатываются последовательно, по четко установленным правилам. В их холодных запрограммированных расчетах нет места неоднозначности и неясности. С другой стороны, постепенно становилось понятно, что мозг животных, несмотря на кажущуюся замедленность его рабочих ритмов по сравнению с компьютерами, обрабатывает сигналы параллельно и что неопределенность является существенной отличительной чертой его деятельности.

Рассмотрим строение базовой структурно-функциональной единицы биологического мозга — нейрона.



Несмотря на то что нейроны существуют в различных формах, все они передают электрические сигналы от одного конца нейрона к другому — от дендритов через аксоны до терминалей. Далее эти сигналы передаются от одного нейрона к другому. Именно благодаря такому механизму вы способны воспринимать свет, звук, прикосновение, тепло и т.п. Сигналы от специализированных рецепторных нейронов доставляются по вашей нервной системе до мозга, который в основном также состоит из нейронов.

На приведенной ниже иллюстрации показана схема строения нейронов мозга попугая, представленная одним испанским нейробиологом в 1899 году. На ней отчетливо видны важнейшие компоненты нейрона — дендриты и терминали.



Сколько нейронов нам нужно для выполнения интересных, более сложных задач?

В головном мозге человека, являющемся самым развитым, насчитывается около 100 миллиардов нейронов. Мозг дрозофилы (плодовой мушки) содержит примерно 100 тысяч нейронов, но она способна летать, питаться, избегать опасности, находить пищу и решать множество других довольно сложных задач. Это количество — 100 тысяч — вполне сопоставимо с возможностями современных компьютеров, и поэтому в попытке имитации работы такого мозга есть смысл. Мозг нематоды (круглого червя) насчитывает всего 302 нейрона — ничтожно малая величина по сравнению с ресурсами современных цифровых компьютеров! Но этот червь способен решать такие довольно сложные задачи, которые традиционным компьютерам намного больших размеров пока что не по силам.

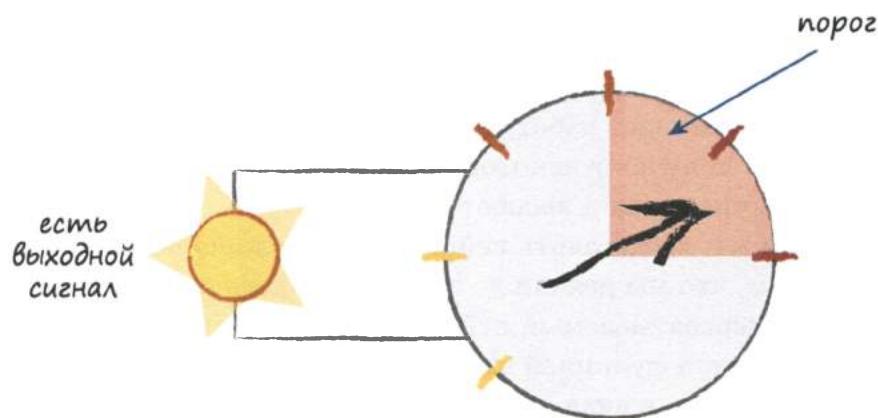
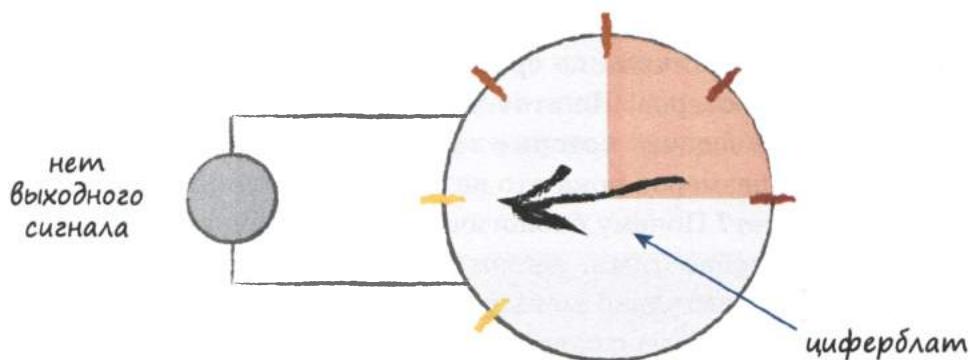
В чем же секрет? Почему биологический мозг обладает столь замечательными способностями, несмотря на то что работает медленнее и состоит из относительно меньшего количества вычислительных элементов по сравнению с современными компьютерами? Сложные механизмы функционирования мозга (например, наличие сознания) все еще остаются загадкой, но мы знаем о нейронах достаточно много для того, чтобы можно было предложить различные способы выполнения вычислений, т.е. различные способы решения задач.

Рассмотрим, как работает нейрон. Он принимает поступающий электрический сигнал и вырабатывает другой электрический сигнал. Это очень напоминает работу моделей классификатора или предиктора, которые получают некоторые входные данные, выполняют определенные вычисления и выдают результат.

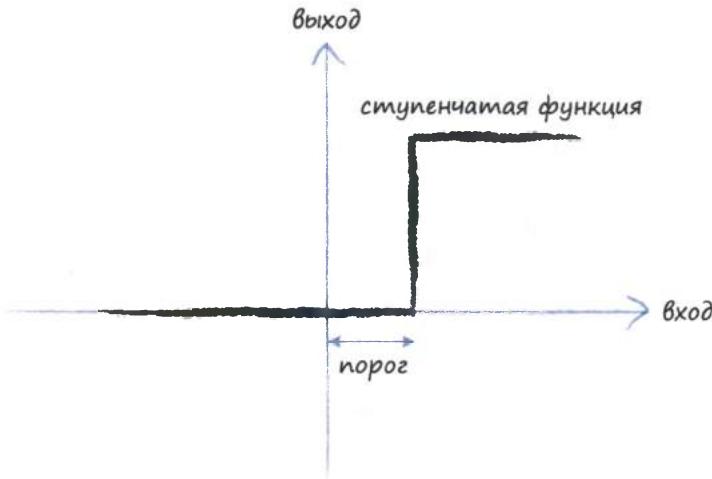
Так можем ли мы представить нейроны в виде линейных функций по аналогии с тем, что мы делали до этого? Нет, хотя сама по себе эта идея неплохая. Вырабатываемый нейроном выходной сигнал не является простой линейной функцией входного сигнала, т.е. выходной сигнал нельзя представить в виде `выход = (константа * вход) + (возможная другая константа)`.

Согласно результатам наблюдений нейроны не реагируют немедленно, а подавляют входной сигнал до тех пор, пока он не возрастет до такой величины, которая запустит генерацию выходного сигнала. Это можно представить себе как наличие некоего порогового значения, которое должно быть превышено, прежде чем будет сгенерирован выходной сигнал. Сравните поведение воды в чашке: вода не

сможет выплыть, пока чашка не наполнится. Интуитивно это понятно — нейроны пропускают лишь сильные сигналы, несущие полезную информацию, но не слабый шум. Идея выработывания выходного сигнала лишь в случаях, когда амплитуда входного сигнала достигает достаточной величины, превышающей некоторый порог, проиллюстрирована ниже в графическом виде.

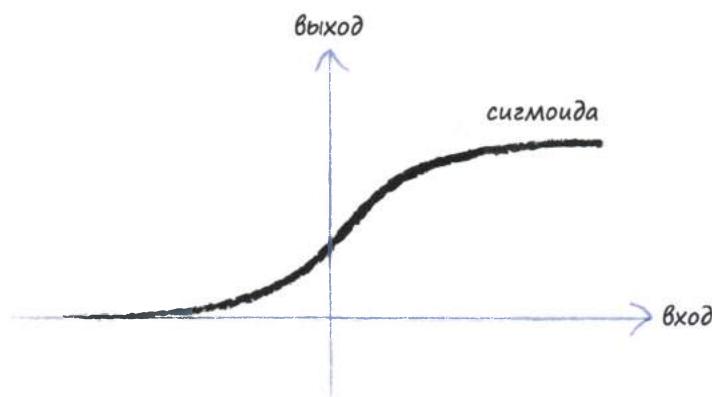


Функция, которая получает входной сигнал, но генерирует выходной сигнал с учетом порогового значения, называется **функцией активации**. С математической точки зрения существует множество таких функций, которые могли бы обеспечить подобный эффект. В качестве примера можно привести **ступенчатую функцию**.



Нетрудно заметить, что для слабых входных значений выходное значение равно нулю. Но стоит превысить входной порог, как на выходе появляется сигнал. Искусственный нейрон с таким поведением напоминал бы настоящий биологический нейрон. Это очень хорошо описывается термином, который используется учеными: они говорят, что нейрон **возбуждается** при достижении входным сигналом порогового значения.

Ступенчатую функцию можно усовершенствовать. Представленную ниже S-образную функцию называют **сигмоидой**, или сигмоидальной функцией. Резкие прямоугольные границы ступенчатой функции в ней сглажены, что делает ее более естественной и реалистичной. Природа не любит острых углов!



Сглаженная S-образная сигмоида — это и есть то, что мы будем использовать для создания собственной нейронной сети. Исследователями в области искусственного интеллекта используются также другие функции аналогичного вида, но сигмоида проста и очень популярна, так что она будет для нас отличным выбором.

Сигмоида, которую иногда называют также логистической функцией, описывается следующей формулой:

$$y = \frac{1}{1 + e^{-x}}$$

Это выражение не настолько устрашающее, как поначалу может показаться. Буквой e в математике принято обозначать константу, равную 2,71828... Это очень интересное число, которое встречается во многих областях математики и физики, а причина, по которой я использовал в нем многоточие (...), заключается в том, что запись десятичных знаков может быть продолжена до бесконечности. Для подобных чисел существует причудливое название — трансцендентные числа. Все это, конечно, интересно, но для наших целей вполне достаточно считать, что это число просто равно 2,71828. Входное значение берется с отрицательным знаком, и e возводится в степень $-x$. Результат прибавляется к 1, что дает нам $1+e^{-x}$. Наконец, мы обращаем последнюю сумму, т.е. делим 1 на $1+e^{-x}$. Это и есть то, что делает приведенная выше функция с входным значением x для того, чтобы предоставить нам выходное значение y . Поэтому на самом деле ничего страшного в ней нет.

Просто ради интереса следует отметить, что при нулевом значении x выражение e^{-x} принимает значение 1, поскольку возведение любого числа в нулевую степень всегда дает 1. Поэтому y становится равным $1 / (1 + 1)$ или просто $1/2$, т.е. половине. Следовательно, базовая сигмоида пересекает ось y при $y=1/2$.

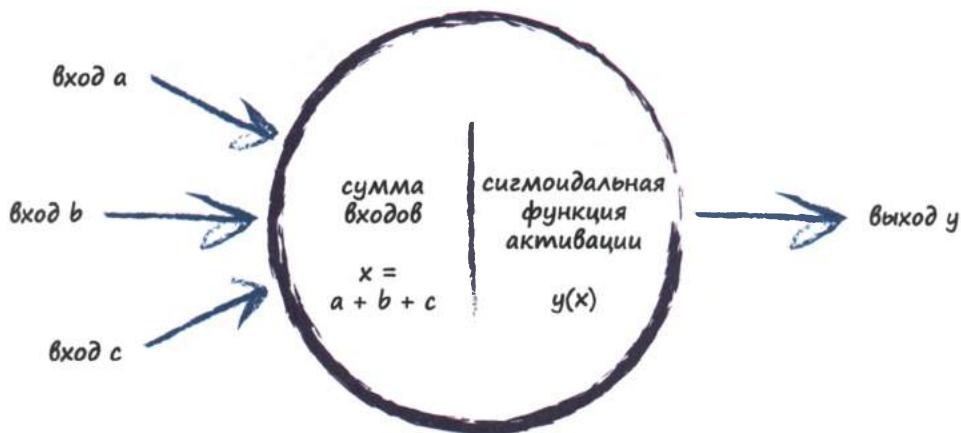
Существует еще одна, очень веская, причина для того, чтобы из множества всех S-образных функций, которые можно было бы использовать для определения выходного значения нейрона, выбрать

именно сигмоиду. Дело в том, что выполнять расчеты с сигмоидой намного проще, чем с любой другой S-образной функцией, и вскоре вы на практике убедитесь в том, что это действительно так.

Вернемся к нейронам и посмотрим, как мы можем смоделировать искусственный нейрон.

Первое, что следует понять, — это то, что реальные биологические нейроны имеют несколько входов, а не только один. Мы уже сталкивались с этим на примере булевой логической машины с двумя входами, поэтому идея более чем одного входа не будет для вас чем-то новым или необычным.

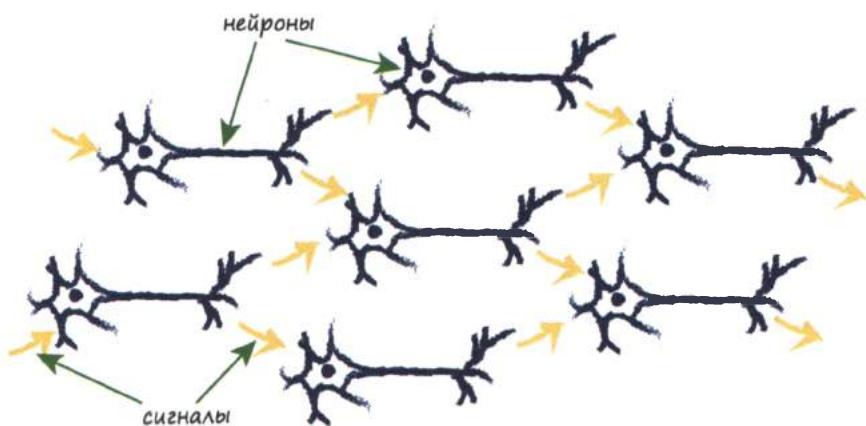
Но что нам делать со всеми этими входами? Мы будем просто комбинировать их, суммируя соответствующие значения, и результирующая сумма будет служить входным значением для сигмоиды, управляющей выходным значением. Такая схема отражает принцип работы нейронной сети. Приведенная ниже диаграмма иллюстрирует идею комбинирования входных значений и сравнения результирующей суммы с пороговым значением.



Если комбинированный сигнал недостаточно сильный, то сигмоида подавляет выходной сигнал. Если же сумма x достаточно велика, то функция возбуждает нейрон. Интересно отметить, что даже если только один сигнал достаточно сильный, в то время как все остальные имеют небольшую величину, то и этого может вполне хватить для возбуждения нейрона. Более того, нейрон может возбудиться

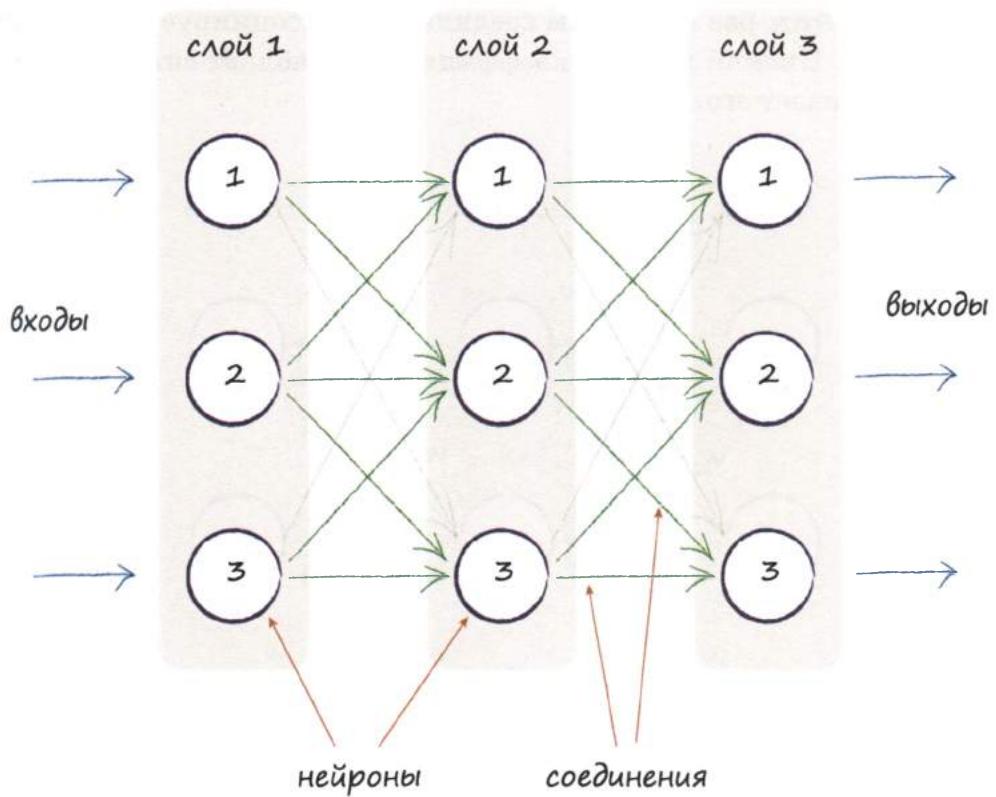
и тогда, когда каждый из сигналов, взятых по отдельности, имеет недостаточную величину, но, будучи взятыми вместе, они обеспечивают превышение порога. В этом уже чувствуется прототип более сложных и в некотором смысле неопределенных вычислений, на которые способны подобные нейроны.

Электрические сигналы воспринимаются дендритами, где они комбинируются, формируя более сильный сигнал. Если этот сигнал превышает порог, нейрон возбуждается, и сигнал передается через аксон к терминалам, откуда он поступает на дендриты следующего нейрона. Связанные таким способом нейроны схематически изображены на приведенной ниже иллюстрации.



На этой схеме бросается в глаза то, что каждый нейрон принимает входной сигнал от нескольких находящихся перед ним нейронов и, в свою очередь, также передает сигнал многим другим в случае возбуждения.

Одним из способов воспроизведения такого поведения нейронов, наблюдаемого в живой природе, в искусственной модели является создание многослойных нейронных структур, в которых каждый нейрон соединен с каждым из нейронов в предшествующем и последующем слоях. Эта идея поясняется на следующей иллюстрации.



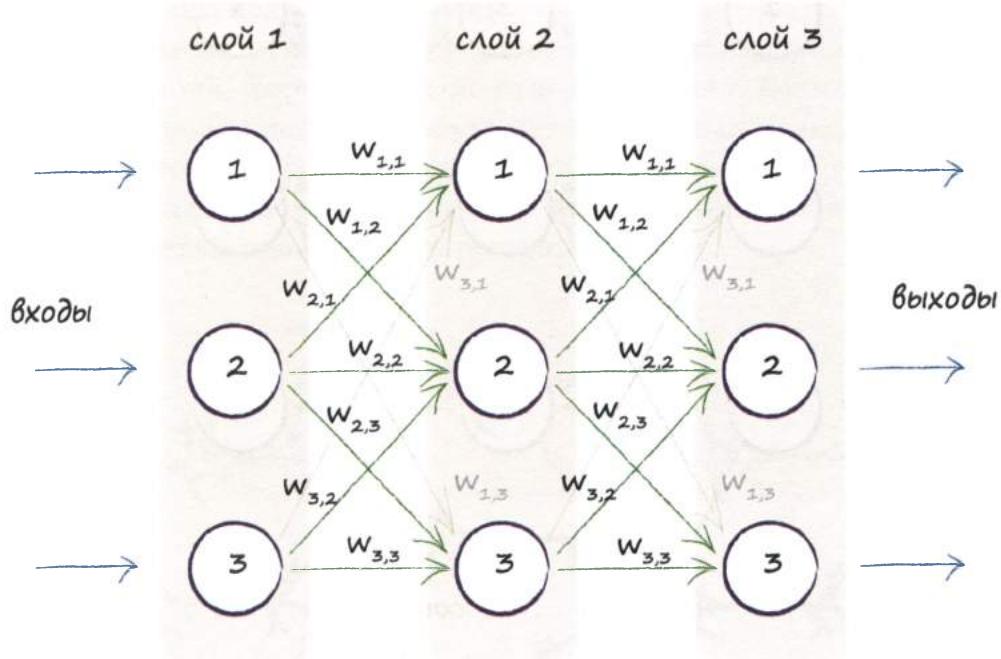
На этой иллюстрации представлены три слоя, каждый из которых включает три искусственных нейрона, или узла. Как нетрудно заметить, здесь каждый узел соединен с каждым из узлов предшествующего и последующего слоев.

Прекрасно! Но в какой части этой застывшей структуры заключена способность к обучению? Что мы должны регулировать, реагируя на данные тренировочных примеров? Есть ли здесь параметр, который можно было бы улучшать подобно тому, как ранее мы делали это с наклоном прямой линейного классификатора?

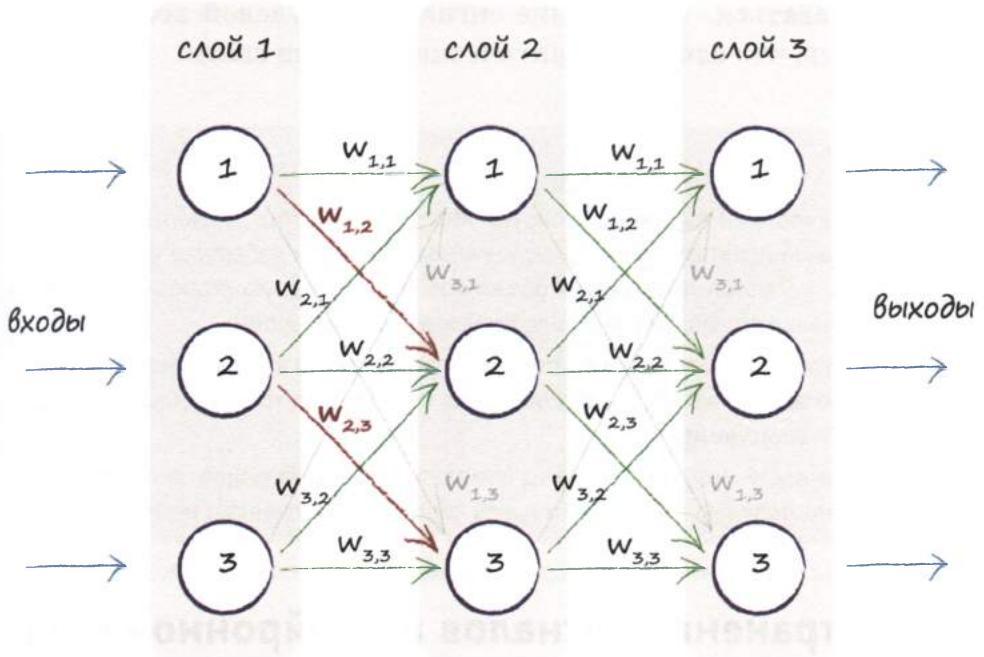
Наиболее очевидной величиной, регулировать которую мы могли бы, является сила связи между узлами. В пределах узла мы могли бы регулировать суммирование входных значений или же форму сигмоиды, но это уже немножко сложнее предыдущей регулировки.

Если работает более простой подход, то давайте им и ограничимся! На следующей диаграмме вновь показаны соединенные между собой

узлы, но на этот раз с каждым соединением ассоциируется определенный вес. Низкий весовой коэффициент ослабляет сигнал, высокий — усиливает его.



Следует сказать несколько слов о небольших индексах, указанных рядом с коэффициентами. Например, символ $w_{2,3}$ обозначает весовой коэффициент, связанный с сигналом, который передается от узла 2 данного слоя к узлу 3 следующего слоя. Следовательно, $w_{1,2}$ — это весовой коэффициент, который ослабляет или усиливает сигнал, передаваемый от узла 1 к узлу 2 следующего слоя. Чтобы проиллюстрировать эту идею, на следующей диаграмме оба этих соединения между первым и вторым слоями выделены цветом.



Вы могли бы вполне обоснованно подвергнуть сомнению данный замысел и задаться вопросом о том, почему каждый узел слоя должен быть связан с каждым из узлов предыдущего и последующего слоев. Это требование не является обязательным, и слои можно соединять между собой любым мыслимым способом. Мы не рассматриваем здесь другие возможные способы по той причине, что благодаря однородности описанной схемы полного взаимного соединения нейронов закодировать ее в виде компьютерных инструкций на самом деле значительно проще, чем любую другую схему, а также потому, что наличие большего количества соединений, чем тот их обязательный минимум, который может потребоваться для решения определенной задачи, не принесет никакого вреда. Если дополнительные соединения действительно не нужны, то процесс обучения ослабит их влияние.

Что под этим подразумевается? Это означает, что, как только сеть научится улучшать свои выходные значения путем уточнения весовых коэффициентов связей внутри сети, некоторые веса обнулятся или станут близкими к нулю. В свою очередь, это означает, что такие связи не будут оказывать влияния на сеть, поскольку их сигналы не

будут передаваться. Умножение сигнала на нулевой вес дает в результате нуль, что означает фактический разрыв связи.

Резюме

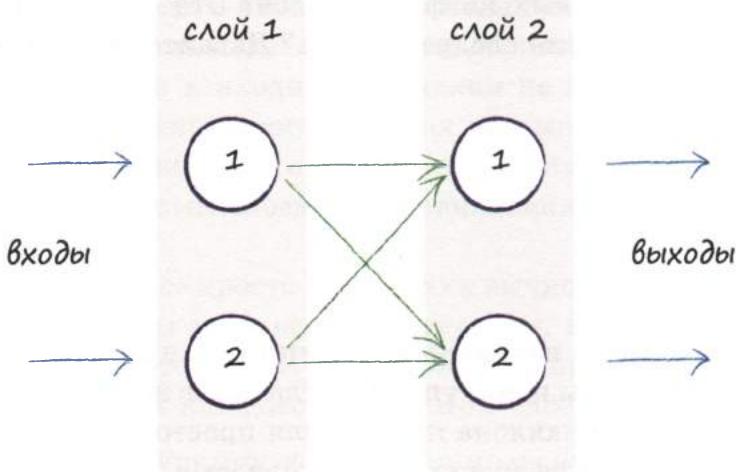
- Биологический мозг демонстрирует выполнение таких сложных задач, как управление полетом, поиск пищи, изучение языка или избегание встреч с хищниками, несмотря на меньший объем памяти и меньшую скорость обработки информации по сравнению с современными компьютерами.
- Кроме того, биологический мозг невероятно устойчив к повреждениям и несовершенству обрабатываемых сигналов по сравнению с традиционными компьютерными системами.
- Биологический мозг, состоящий из взаимосвязанных нейронов, является источником вдохновения для разработчиков систем искусственного интеллекта.

Распространение сигналов по нейронной сети

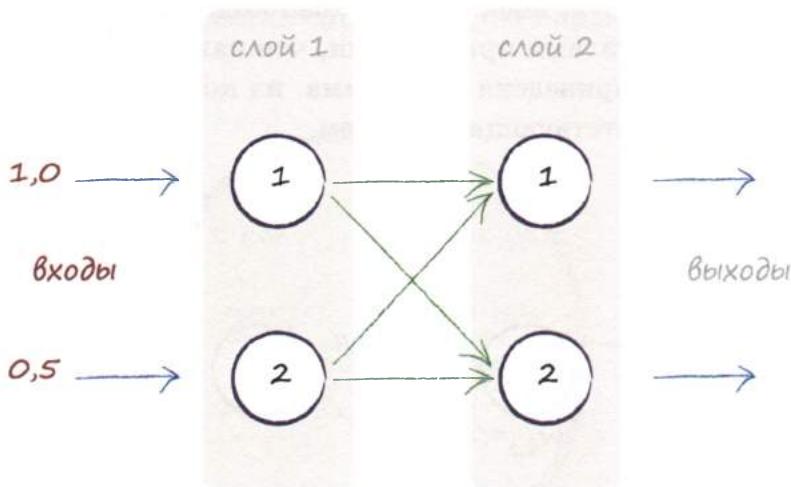
Приведенная выше картинка с тремя слоями нейронов, каждый из которых связан с каждым из нейронов в предыдущем и следующем слоях, выглядит довольно привлекательно.

Однако интуиция подсказывает нам, что рассчитать распространение входных сигналов по всем слоям, пока они не превратятся в выходные сигналы, не так-то просто, и нам, скажем так, придется хорошенько потрудиться!

Я согласен, что эта работа непростая, но также важно показать, как работает этот механизм, чтобы мы всегда четко представляли себе, что происходит в нейронной сети на самом деле, даже если впоследствии мы используем компьютер, который выполнит вместо нас всю работу. Поэтому мы попытаемся проделать необходимые вычисления на примере меньшей нейронной сети, состоящей всего лишь из двух слоев, каждый из которых включает по 2 нейрона, как показано ниже.



Предположим, что сигналам на входе соответствуют значения 1,0 и 0,5.



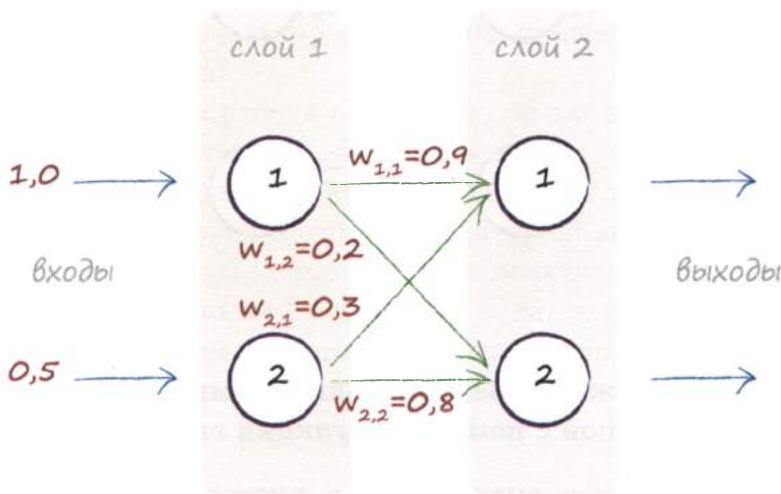
Как и раньше, каждый узел превращает сумму двух входных сигналов в один выходной с помощью функции активации. Мы также будем использовать сигмоиду $y = \frac{1}{1 + e^{-x}}$, с которой вы до этого познакомились, где x — это сумма сигналов, поступающих в нейрон, а y — выходной сигнал этого нейрона.

А что насчет весовых коэффициентов? Это очень хороший вопрос: с какого значения следует начать? Давайте начнем со случайных весов:

- $w_{1,1} = 0,9$
- $w_{1,2} = 0,2$
- $w_{2,1} = 0,3$
- $w_{2,2} = 0,9$

Выбор случайных начальных значений — не такая уж плохая идея, и именно так мы и поступали, когда ранее выбирали случайное начальное значение наклона прямой для простого линейного классификатора. Случайное значение улучшалось с каждым очередным тренировочным примером, используемым для обучения классификатора. То же самое должно быть справедливым и для весовых коэффициентов связей в нейронных сетях.

В данном случае, когда сеть небольшая, мы имеем всего четыре весовых коэффициента, поскольку таково количество всех возможных связей между узлами при условии, что каждый слой содержит по два узла. Ниже приведена диаграмма, на которой все связи помаркированы соответствующим образом.

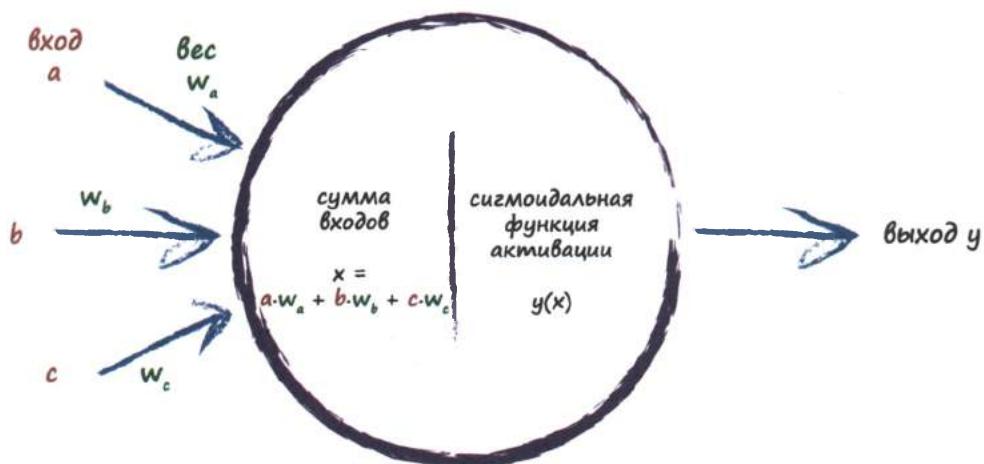


Приступим к вычислениям!

Первый слой узлов — входной, и его единственное назначение — представлять входные сигналы. Таким образом, во входных узлах функция активации к входным сигналам не применяется. Мы не выдвигаем в отношении этого никаких разумных доводов и просто принимаем как данность, что первый слой нейронных сетей является всего лишь входным слоем, представляющим входные сигналы. Вот и все.

С первым слоем все просто — никаких вычислений.

Далее мы должны заняться вторым слоем, в котором потребуется выполнить некоторые вычисления. Нам предстоит определить входной сигнал для каждого узла в этом слое. Помните сигмоиду $y = \frac{1}{1 + e^{-x}}$? В этой функции x — комбинированный сигнал на входе узла. Данная комбинация образуется из необработанных выходных сигналов связанных узлов предыдущего слоя, сглаженных весовыми коэффициентами связей. Приведенная ниже диаграмма аналогична тем, с которыми вы уже сталкивались, но теперь на ней указано сглаживание поступающих сигналов за счет применения весовых коэффициентов связей.



Для начала сосредоточим внимание на узле 1 слоя 2. С ним связаны оба узла первого, входного слоя. Исходные значения на этих входных узлах равны 1,0 и 0,5. Связи первого узла назначен весовой

коэффициент 0,9, связи второго — 0,3. Поэтому сглаженный входной сигнал вычисляется с помощью следующего выражения:

$$\begin{aligned}x &= (\text{выход первого узла} * \text{вес связи}) + \\&+ (\text{выход второго узла} * \text{вес связи}) \\x &= (1,0 * 0,9) + (0,5 * 0,3) \\x &= 0,9 + 0,15 \\x &= 1,05\end{aligned}$$

Без сглаживания сигналов мы просто получили бы их сумму $1,0 + 0,5$, но мы этого не хотим. Именно с весовыми коэффициентами будет связан процесс обучения нейронной сети по мере того, как они будут итеративно уточняться для получения все лучшего и лучшего результата.

Итак, мы уже имеем значение $x=1,05$ для комбинированного сглаженного входного сигнала первого узла второго слоя и теперь располагаем всеми необходимыми данными, чтобы рассчитать для этого узла выходной сигнал с помощью функции активации $y = \frac{1}{1+e^{-x}}$. Попробуйте справиться с этим самостоятельно, используя калькулятор. Вот правильный ответ: $y = 1 / (1 + 0,3499) = 1 / 1,3499$. Таким образом, $y=0,7408$.

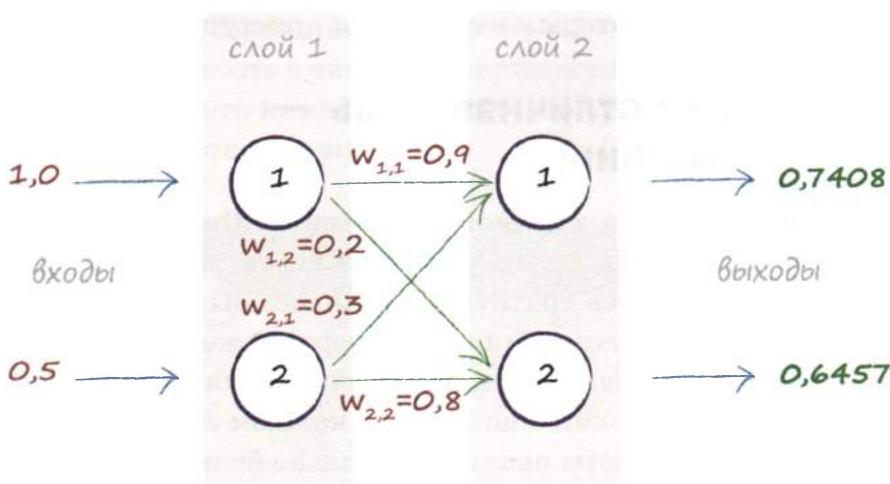
Отличная работа! Мы рассчитали фактический выходной сигнал для одного из двух выходных узлов сети.

Повторим те же вычисления для оставшегося узла — узла 2 второго слоя, т.е. вновь вычислим сглаженный входной сигнал с помощью следующего выражения:

$$\begin{aligned}x &= (\text{выход первого узла} * \text{вес связи}) + \\&+ (\text{выход второго узла} * \text{вес связи}) \\x &= (1,0 * 0,2) + (0,5 * 0,8) \\x &= 0,2 + 0,4 \\x &= 0,6\end{aligned}$$

Располагая значением x , можно рассчитать выходной сигнал узла с помощью функции активации: $y = 1 / (1 + 0,5488) = 1 / 1,5488$. Таким образом, $y=0,6457$.

Рассчитанные нами выходные сигналы сети представлены на приведенной ниже диаграмме.



Как видите, чтобы получить всего лишь два выходных сигнала для весьма простой нейронной сети, нам пришлось хорошо потрудиться. Лично я не был бы в восторге от необходимости проведения аналогичных расчетов вручную в случае более крупных сетей! К счастью, для выполнения подобных вычислений идеально подходят компьютеры, которые могут быстро справиться с этой работой, избавив вас от лишних хлопот.

Однако даже с учетом этого я не горел бы желанием выписывать соответствующие программные инструкции для сети, насчитывающей более двух слоев, каждый из которых может включать 4, 8 или даже 100 узлов. Стоит ли говорить о том, насколько это утомительное занятие, к тому же оно чревато большим количеством ошибок и опечаток, что вообще может сделать выполнение расчетов практически невозможным.

Но нам повезло. Математики разработали чрезвычайно компактный способ записи операций того типа, которые приходится выполнять для вычисления выходного сигнала нейронной сети, даже если она содержит множество слоев и узлов. Эта компактность благоприятна не только для людей, которые выписывают или читают соответствующие

формулы, но и для компьютеров, поскольку программные инструкции получаются более короткими и выполняются намного быстрее.

В этом компактном подходе предполагается использование матриц, к рассмотрению которых мы сейчас и приступим.

Какая все-таки отличная вещь — умножение матриц!

Матрицы пользуются ужасной репутацией. Они вызывают воспоминания о тех долгих и утомительных часах, которые в студенческие годы приходилось тратить на, казалось бы, совершенно бесполезное и никому не нужное занятие — перемножение матриц.

Перед этим мы вручную выполнили вычисления для двухслойной сети, содержащей всего лишь по 2 узла в каждом слое. Однако даже в этом случае объем работы оказался довольно большим, а что тогда можно сказать, например, о сети, состоящей из пяти слоев по 100 узлов? Уже сама по себе запись всех необходимых выражений была бы сложной задачей. Составление комбинаций сигналов, умножение на подходящие весовые коэффициенты, применение сигмоиды для каждого узла... Ух!

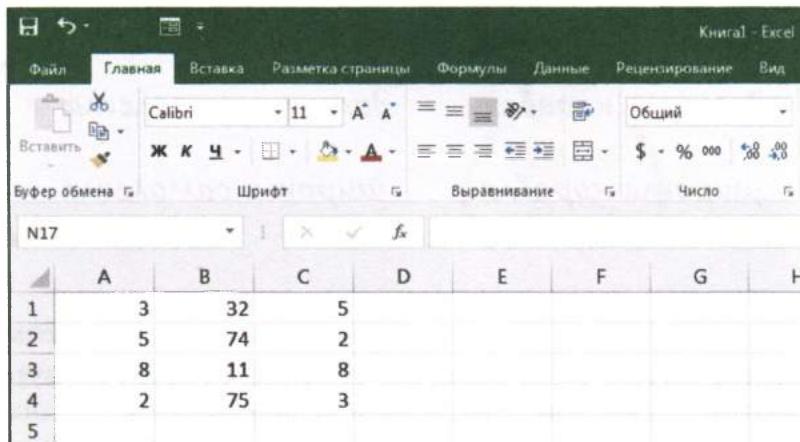
Так чем же нам могут помочь матрицы? Они будут нам полезны в двух отношениях. Во-первых, они обеспечивают сжатую запись операций, придавая им чрезвычайно компактную форму. Это большой плюс для нас, поскольку такая работа утомительна, требует концентрации внимания и чревата ошибками. Во-вторых, в большинстве компьютерных языков программирования предусмотрена работа с матрицами, а поскольку многие операции при этом повторяются, компьютеры распознают это и выполняют их очень быстро.

Резюмируя, можно сказать, что матрицы позволяют формулировать задачи в более простой и компактной форме и обеспечивают их более быстрое и эффективное выполнение.

Теперь, когда вам известны причины, по которым мы собираемся воспользоваться матрицами несмотря на возможный горький опыт знакомства с ними в студенческие годы, давайте демистифицируем матрицы, начав работать с ними.

Матрица — это всего лишь таблица, прямоугольная сетка, образованная числами. Вот и все. Все, что связано с матрицами, ничуть не сложнее этого.

Если вы уже пользовались электронными таблицами, то никаких сложностей при работе с числами, располагающимися в ячейках сетки, у вас возникнуть не должно. Ниже показан пример рабочего листа Excel с числовой таблицей.



The screenshot shows a Microsoft Excel spreadsheet titled "Книга1 - Excel". The ribbon menu is visible at the top, with "Главная" selected. The formula bar shows "N17". The main area contains a 5x4 grid of numbers:

1	3	32	5
2	5	74	2
3	8	11	8
4	2	75	3
5			

Это и есть матрица — таблица или сетка, образованная числами, точно так же, как и следующий пример матрицы размером 2×3 .

$$\begin{pmatrix} 23 & 43 & 22 \\ 43 & 12 & 54 \end{pmatrix}$$

При обозначении матриц принято указывать сначала количество строк, а затем количество столбцов. Поэтому размерность данной матрицы не 3×2 , а 2×3 .

Кроме того, иногда матрицы заключают в квадратные скобки, иногда — в круглые, как это сделали мы.

На самом деле матрицы не обязательно состоять из чисел. Они могут включать величины, которым мы дали имена, но не присвоили фактических значений. Поэтому на следующей иллюстрации также показана матрица с элементами в виде **переменных**, каждая из которых имеет определенный смысл и может иметь значение, просто мы пока не указали, что это за значения.

$$\begin{pmatrix} \text{длгота корабля} & \text{длгота самолета} \\ \text{широта корабля} & \text{широта самолета} \end{pmatrix}$$

Чем полезны матрицы, вам станет понятно, когда мы рассмотрим, как они умножаются. Возможно, вы это помните еще из курса высшей математики, а если нет, то освежите свою память.

Вот пример умножения одной простой матрицы на другую.

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

Как видите, эта операция не сводится к простому перемножению соответствующих элементов. Левый верхний элемент не равен $1*5$, как и правый нижний — не $4*8$.

Вместо этого матрицы умножаются по другим правилам. Возможно, вы и сами догадаетесь, по каким именно, если внимательно присмотритесь к примеру. Если нет, то взгляните на приведенную ниже иллюстрацию, где показано, как получается ответ для левого верхнего элемента.

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

Вы видите, что верхний левый элемент вычисляется с использованием верхней строки первой матрицы и левого столбца второй. Мысленно перебирая обе последовательности элементов, перемножьте их попарно и сложите полученные результаты. Таким образом, чтобы вычислить левый верхний элемент результирующей матрицы, мы начинаем перемещаться вдоль верхней строки первой матрицы, где находим число 1, а начиная перемещение вниз по левому столбцу второй матрицы, находим число 5. Мы перемножаем их и запоминаем результат, который равен 5. Мы продолжаем перемещаться вдоль ряда и вниз по столбцу и находим элементы 2 и 7. Результат их перемножения, равный 14, мы также запоминаем. При этом мы уже достигли конца строки и столбца, поэтому суммируем числа, которые перед этим запомнили, и получаем в качестве результата число 19. Это и есть левый верхний элемент результирующей матрицы.

Описание получилось длинным, но выполнение самих действий не отнимает много времени. Продолжите далее самостоятельно. На приведенной ниже иллюстрации показано, как вычисляется правый нижний элемент результирующей матрицы.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} (1*5) + (2*7) & (1*6) + (2*8) \\ (3*5) + (4*7) & (3*6) + (4*8) \end{pmatrix}$$

$$= \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Вы вновь можете видеть, как, используя строку и столбец, которые соответствуют искомому элементу (в данном случае вторая строка и второй столбец), мы получаем два произведения, $(3*6)$ и $(4*8)$, сложение которых дает $18 + 32 = 50$.

Продолжая действовать в таком же духе, находим, что левый нижний элемент равен $(3*5) + (4*7) = 15 + 28 = 43$, а правый верхний — $(1*6) + (2*8) = 6 + 16 = 22$.

На следующей иллюстрации продемонстрировано использование переменных вместо чисел:

$$\begin{pmatrix} a & b & .. \\ c & d & .. \\ .. & .. & .. \end{pmatrix} \begin{pmatrix} e & f \\ g & h \\ .. & .. \end{pmatrix} = \begin{pmatrix} (a*e) + (b*g) + ... & (a*f) + (b*h) + ... \\ (c*e) + (d*g) + ... & (c*f) + (d*h) + ... \end{pmatrix}$$

$$= \begin{pmatrix} ae+bg+... & af+bh+... \\ ce+dg+... & cf+dh+... \end{pmatrix}$$

Это всего лишь другой способ описания подхода, с помощью которого мы умножаем матрицы. Используя буквы, вместо которых могут быть подставлены любые числа, мы даем общее описание этого подхода. Такое описание действительно является более общим, чем предыдущее, поскольку применимо к матрицам различных размеров.

Говоря о применимости описанного подхода к матрицам любого размера, следует сделать одну важную оговорку: умножаемые матрицы должны быть совместимыми. Вы поймете, в чем здесь дело, вспомнив, как отбирали попарно элементы из строк первой матрицы и столбцов второй: этот метод не сможет работать, если количество элементов в строке не будет совпадать с количеством элементов в столбце. Поэтому вы не сможете умножить матрицу 2×2 на матрицу 5×5 . Попытайтесь это сделать, и сами увидите, почему это невозможно. Чтобы матрицы можно было умножить, количество столбцов в первой из них должно совпадать с количеством строк во второй.

В разных руководствах этот тип матричного умножения может встречаться под разными названиями: **скалярное произведение**, **точечное произведение** или **внутреннее произведение**. В математике возможны и другие типы умножения матриц, такие как перекрестное произведение, но мы будем работать с точечным произведением матриц.

Но почему мы должны углубляться в дебри этого ужасного матричного умножения и отвратительной алгебры? На то есть серьезная причина... Крепитесь!

Посмотрим, что произойдет, если заменить буквы словами, имеющими более непосредственное отношение к нашей нейронной сети. На приведенной ниже иллюстрации вторая матрица имеет размерность 2×1 , но матричная алгебра остается прежней.

$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} \text{input_1} \\ \text{input_2} \end{pmatrix} = \begin{pmatrix} (\text{input_1} * w_{1,1}) + (\text{input_2} * w_{2,1}) \\ (\text{input_1} * w_{1,2}) + (\text{input_2} * w_{2,2}) \end{pmatrix}$$

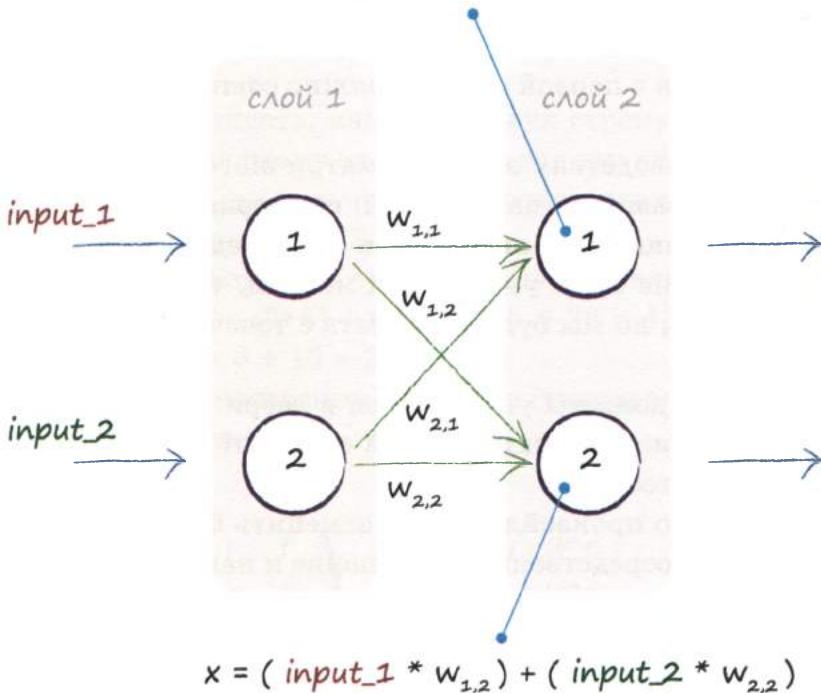
Иначе как магией это никак не назовешь!

Первая матрица содержит весовые коэффициенты для связей между узлами двух слоев, вторая — сигналы первого, входного слоя. Результатом умножения этих двух матриц является объединенный слаженный сигнал, поступающий на узлы второго слоя. Присмотритесь повнимательнее, и вы это поймете. На первый узел поступает сумма двух сигналов — сигнала `input_1`, умноженного

на весовой коэффициент $w_{1,1}$, и сигнала `input_2`, умноженного на весовой коэффициент $w_{2,1}$. Это значения x до применения к ним функции активации.

Представим все это в более наглядной форме с помощью иллюстрации.

$$x = (\text{input}_1 * w_{1,1}) + (\text{input}_2 * w_{2,1})$$



Описанный подход чрезвычайно полезен.

Почему? Потому что в его рамках все вычисления, которые необходимы для расчета входных сигналов, поступающих на каждый из узлов второго слоя, могут быть выражены с использованием матричного умножения, обеспечивающего чрезвычайно компактную форму записи соответствующих операций:

$$\mathbf{X} = \mathbf{W} \cdot \mathbf{I}$$

Здесь \mathbf{W} — матрица весов, \mathbf{I} — матрица входных сигналов, а \mathbf{X} — результирующая матрица комбинированных сглаженных сигналов, поступающих в слой 2. Символы матриц часто записывают с использованием **полужирного** шрифта, чтобы подчеркнуть, что данные символы представляют матрицы, а не просто одиночные числа.

Теперь нам не нужно заботиться о том, сколько узлов входит в каждый слой. Увеличение количества слоев приводит лишь к увеличению размера матриц. Но количество символов в записи при этом не увеличивается. Она остается по-прежнему компактной, и мы просто записываем произведение матриц в виде $\mathbf{W} \cdot \mathbf{I}$, независимо от количества элементов в каждой из них, будь это 2 или же 200!

Если используемый язык программирования распознает матричную нотацию, то компьютер выполнит все трудоемкие расчеты, связанные с вычислением выражения $\mathbf{X} = \mathbf{W} \cdot \mathbf{I}$, без предоставления ему отдельных инструкций для каждого узла в каждом слое.

Это просто фантастика! Затратив немного времени на то, чтобы понять суть матричного умножения, мы получили в свое распоряжение мощнейший инструмент для реализации нейронных сетей без каких-либо особых усилий с нашей стороны.

А что насчет функции активации? Здесь все просто и не требует применения матричной алгебры. Все, что нам нужно сделать, — это применить сигмоиду $y = \frac{1}{1 + e^{-x}}$ к каждомуциальному элементу матрицы \mathbf{X} .

Это звучит слишком просто, но так оно и есть, поскольку нам не приходится комбинировать сигналы от разных узлов: это уже сделано, и вся необходимая информация уже содержится в \mathbf{X} . Как мы уже видели, роль функции активации заключается в применении пороговых значений и подавлении ненужных сигналов с целью имитации поведения биологических нейронов. Поэтому результирующий выходной сигнал второго слоя можно записать в таком виде:

$$\mathbf{O} = \text{сигмоида}(\mathbf{X})$$

Здесь символом \mathbf{O} , выделенным полужирным шрифтом, обозначена матрица, которая содержит все выходные сигналы последнего слоя нейронной сети.

Выражение $X = W \cdot I$ применяется для вычисления сигналов, проходящих от одного слоя к следующему слою. Например, при наличии трех слоев мы просто вновь выполним операцию умножения матриц, использовав выходные сигналы второго слоя в качестве входных для третьего, но, разумеется, предварительно скомбинировав их и сгладив с помощью дополнительных весовых коэффициентов.

На данном этапе мы закончили с теорией и теперь должны посмотреть, как она работает, обратившись к конкретному примеру, который на этот раз будет представлять нейронную сеть несколько большего размера, с тремя слоями по три узла.

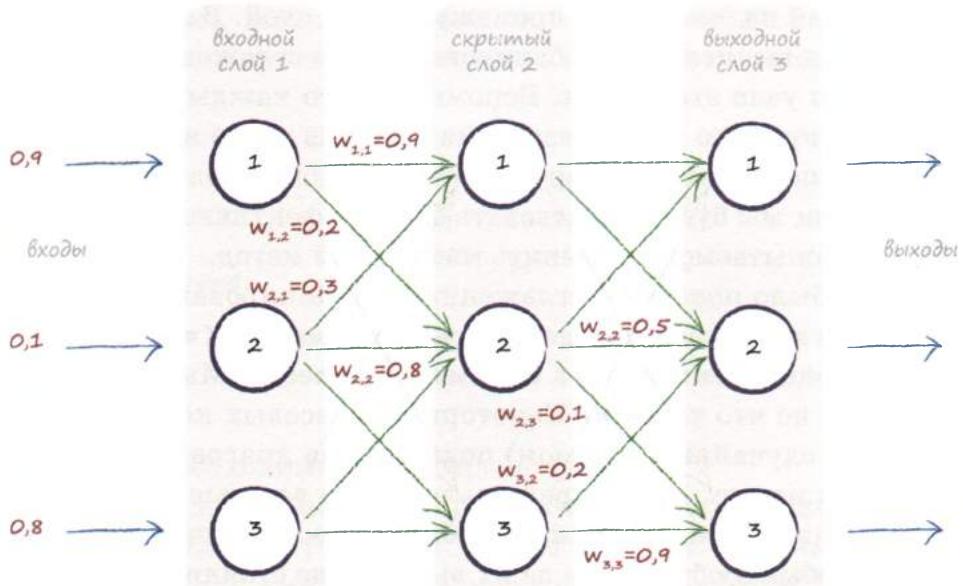
Резюме

- Многие вычисления, связанные с распространением сигналов по нейронной сети, могут быть выполнены с использованием операции **матричного умножения**.
- Использование матричного умножения обеспечивает компактную запись выражений, размер которых не зависит от размеров нейронной сети.
- Что немаловажно, операции матричной алгебры изначально предусмотрены в ряде языков программирования, благодаря чему могут выполняться быстро и эффективно.

Пример использования матричного умножения в сети с тремя слоями

Мы пока еще не использовали матрицы для выполнения расчетов, связанных с вычислением сигналов на выходе нейронной сети. Кроме того, мы пока что не обсуждали примеры с более чем двумя слоями, что было бы гораздо интереснее, поскольку мы должны посмотреть, как обрабатываются выходные сигналы промежуточного слоя в качестве входных сигналов последнего, третьего слоя.

На следующей диаграмме представлен пример нейронной сети, включающей три слоя по три узла. Чтобы избежать загромождения диаграммы лишними надписями, некоторые веса на ней не указаны.



Мы начнем со знакомства с общепринятой терминологией. Как мы знаем, первый слой — **входной**, а второй — **выходной**. Промежуточный слой называется **скрытым слоем**. Это звучит загадочно и таинственно, но, к сожалению, здесь нет ничего загадочного и таинственного. Такое название закрепилось за промежуточным слоем из-за того, что его выходные сигналы не обязательно проявляются как таковые, отсюда и термин “скрытый”.

Приступим к работе над примером, представленным на этой диаграмме. Входными сигналами нейронной сети являются следующие: 0,9; 0,1 и 0,8. Поэтому входная матрица I имеет следующий вид.

$$I = \begin{pmatrix} 0,9 \\ 0,1 \\ 0,8 \end{pmatrix}$$

Это было просто, т.е. с первым слоем мы закончили, поскольку его единственная задача — просто представлять входной сигнал.

Следующий на очереди — промежуточный слой. В данном случае нам придется вычислить комбинированные (и сглаженные) сигналы для каждого узла этого слоя. Вспомните, что каждый узел промежуточного скрытого слоя связан с каждым из узлов входного слоя, поэтому он получает некоторую часть каждого входного сигнала. Однако сейчас мы будем действовать более эффективно, чем раньше, поскольку попытаемся применить матричный метод.

Как уже было показано, сглаженные комбинированные входные сигналы для этого слоя определяются выражением $\mathbf{X} = \mathbf{W} \cdot \mathbf{I}$, где \mathbf{I} — матрица входных сигналов, а \mathbf{W} — матрица весов. Мы располагаем матрицей \mathbf{I} , но что такое \mathbf{W} ? Некоторые из весовых коэффициентов (выбранные случайным образом) показаны на диаграмме для этого примера, но не все. Ниже представлены все весовые коэффициенты (опять-таки, каждый из них выбирался как случайное число). Никакие особые соображения за их выбором не стояли.

$$W_{\text{входной_скрытый}} = \begin{pmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{pmatrix}$$

Как нетрудно заметить, весовой коэффициент для связи между первым входным узлом и первым узлом промежуточного скрытого слоя $w_{1,1}=0,9$, как и на приведенной выше диаграмме. Точно так же весовой коэффициент для связи между вторым входным узлом и вторым узлом скрытого слоя $w_{2,2}=0,8$. На диаграмме не показан весовой коэффициент для связи между третьим входным узлом и первым узлом скрытого слоя $w_{3,1}=0,4$.

Постойте-ка, а почему мы снабдили матрицу \mathbf{W} индексом “входной_скрытый”? Да потому, что матрица $\mathbf{W}_{\text{входной_скрытый}}$ содержит весовые коэффициенты для связей между входным и скрытым слоями. Коэффициенты для связей между скрытым и выходным слоями будут содержаться в другой матрице, которую мы обозначим как $\mathbf{W}_{\text{скрытый_выходной}}$.

Эта вторая матрица $\mathbf{W}_{\text{скрытый_выходной}}$, элементы которой, как и элементы предыдущей матрицы, представляют собой случайные числа,

приведена ниже. Например, вы видите, что весовой коэффициент для связи между третьим скрытым узлом и третьим выходным узлом $w_{3,3} = 0,9$.

$$W_{\text{скрытый_выходной}} = \begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix}$$

Отлично, необходимые матрицы получены.

Продолжим нашу работу и определим комбинированный сглаженный сигнал для скрытого слоя. Кроме того, мы должны присвоить ему описательное имя, снабженное индексом, который указывает на то, что это входной сигнал для скрытого, а не последнего слоя. Назовем соответствующую матрицу $X_{\text{скрытый}}$:

$$X_{\text{скрытый}} = W_{\text{входной_скрытый}} \cdot I$$

Мы не собираемся выполнять вручную все действия, связанные с перемножением матриц. Выполнение этой трудоемкой работы мы будем поручать компьютеру, ведь именно с этой целью мы используем матрицы. В данном случае готовый ответ выглядит так.

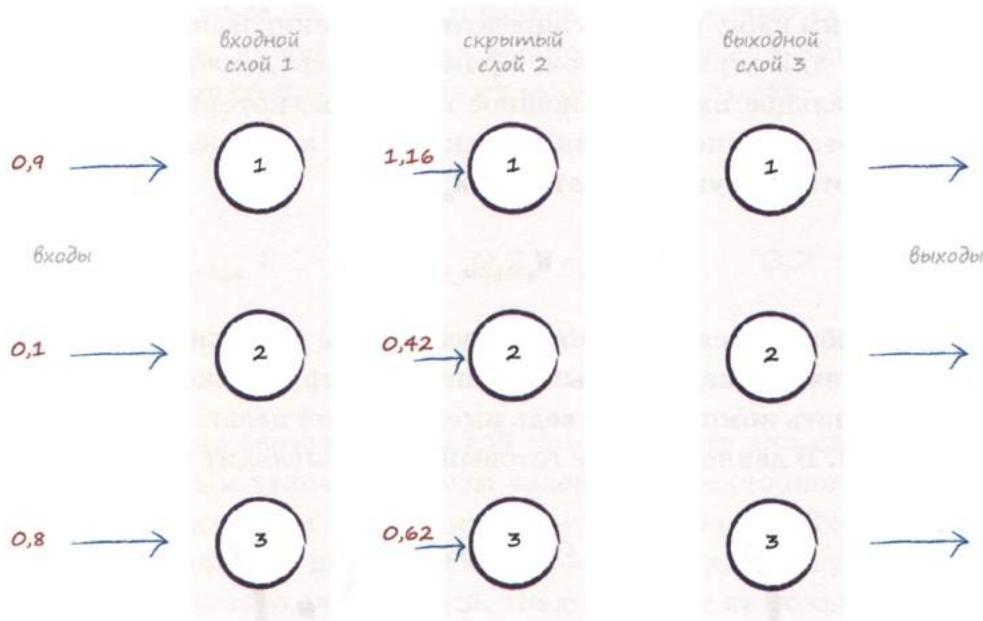
$$X_{\text{скрытый}} = \begin{pmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 0,9 \\ 0,1 \\ 0,8 \end{pmatrix}$$

$$X_{\text{скрытый}} = \begin{pmatrix} 1,16 \\ 0,42 \\ 0,62 \end{pmatrix}$$

Я получил этот ответ с помощью компьютера, и вы научитесь делать это, используя язык программирования Python, о чем мы говорим далее. Мы не будем заниматься этим сейчас, чтобы пока что не отвлекаться на обсуждение программного обеспечения.

Итак, мы располагаем комбинированными сглаженными входными сигналами скрытого промежуточного слоя, значения которых равны 1,16; 0,42 и 0,62. Для выполнения всех необходимых вычислений были использованы матрицы. Это достижение, которым мы вправе гордиться!

Отобразим входные сигналы для скрытого второго слоя на диаграмме.



Пока что все хорошо, но нам еще остается кое-что сделать. Как вы помните, чтобы отклик слоя на входной сигнал как можно лучше имитировал аналогичный реальный процесс, мы должны применить к узлам функцию активации. Так и поступим:

$$O_{\text{скрытый}} = \text{СИГМОИДА} (X_{\text{скрытый}})$$

Применяя сигмоиду к каждому элементу матрицы $X_{\text{скрытый}}$, мы получаем матрицу выходных сигналов скрытого промежуточного слоя.

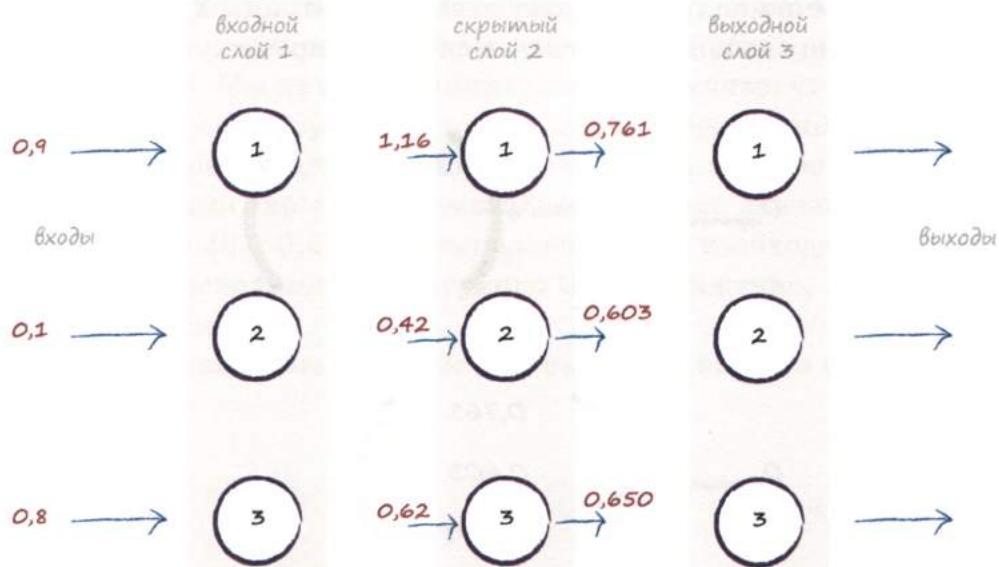
$$O_{\text{скрытый}} = \text{сигмоида} \begin{pmatrix} 1,16 \\ 0,42 \\ 0,62 \end{pmatrix}$$

$$O_{\text{скрытый}} = \begin{pmatrix} 0,761 \\ 0,603 \\ 0,650 \end{pmatrix}$$

Для уверенности давайте проверим, правильно ли вычислен первый элемент. Наша сигмоида имеет вид $y = \frac{1}{1 + e^{-x}}$. Полагая $x=1,16$, получаем $e^{1,16} = 0,3135$. Это означает, что $y = 1 / (1 + 0,3135) = 0,761$.

Нетрудно заметить, что все возможные значения этой функции находятся в пределах от 0 до 1. Вернитесь немного назад и взгляните на график логистической функции, если хотите убедиться в том, что это действительно так.

Ух! Давайте немного передохнем и подытожим, что мы к этому времени успели сделать. Мы рассчитали прохождение сигнала через промежуточный слой, т.е. определили значения сигналов на его выходе. Для полной ясности уточним, что эти значения были получены путем применения функции активации к комбинированным входным сигналам промежуточного слоя. Обновим диаграмму в соответствии с этой новой информацией



Если бы это была двухслойная нейронная сеть, то мы сейчас остановились бы, поскольку получили выходные сигналы второго слоя. Однако мы продолжим, поскольку есть еще и третий слой.

Как рассчитать прохождение сигнала для третьего слоя? Точно так же, как и для второго, поскольку эта задача на самом деле ничем не отличается от предыдущей. Нам известна величина входных сигналов, получаемых третьим слоем, как ранее была известна величина входных сигналов, получаемых вторым слоем. У нас также есть весовые коэффициенты для связей между узлами, ослабляющие сигналы. Наконец, чтобы отклик сети максимально правдоподобно имитировал естественный процесс, мы по-прежнему можем применить функцию активации. Поэтому вам стоит запомнить следующее: независимо от количества слоев в нейронной сети, вычислительная процедура для каждого из них одинакова — комбинирование входных сигналов, сглаживание сигналов для каждой связи между узлами с помощью весовых коэффициентов и получение выходного сигнала с помощью функции активации. Нам безразлично, сколько слоев образуют нейронную сеть — 3, 53 или 103, ведь к любому из них применяется один и тот же подход.

Итак, продолжим вычисления и рассчитаем сглаженный комбинированный входной сигнал $X = W \cdot I$ для третьего слоя.

Входными сигналами для третьего слоя служат уже рассчитанные нами выходные сигналы второго слоя $O_{\text{скрытый}}$. При этом мы должны использовать весовые коэффициенты для связей между узлами второго и третьего слоев $W_{\text{скрытый_выходной}}$, а не те, которые мы уже использовали для первого и второго слоев. Следовательно, мы имеем:

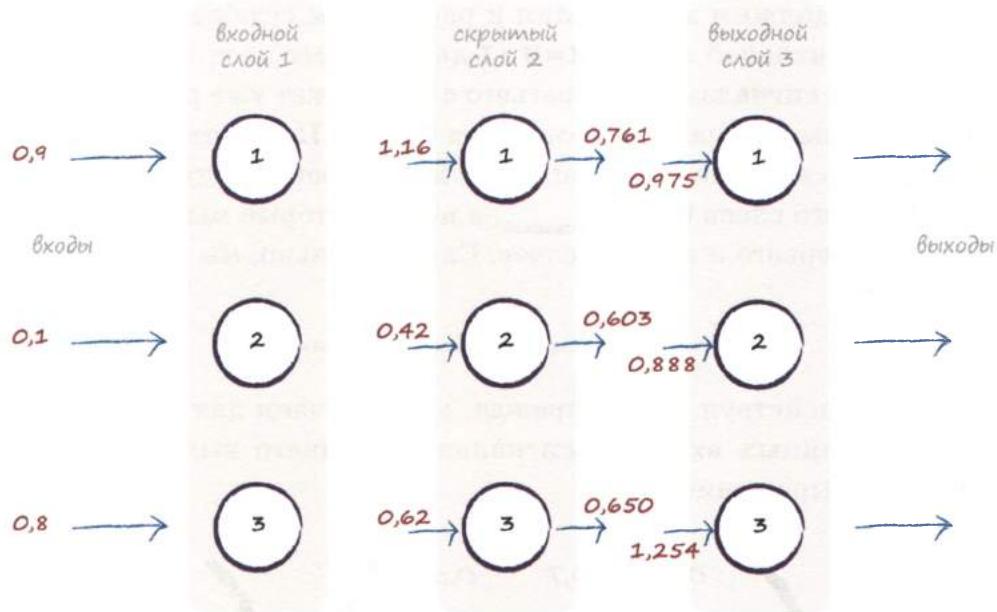
$$X_{\text{выходной}} = W_{\text{скрытый_выходной}} \cdot O_{\text{скрытый}}$$

Поэтому, действуя, как и прежде, мы получаем для сглаженных комбинированных входных сигналов последнего выходного слоя следующее выражение:

$$X_{\text{выходной}} = \begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix} \cdot \begin{pmatrix} 0,761 \\ 0,603 \\ 0,650 \end{pmatrix}$$

$$X_{\text{выходной}} = \begin{pmatrix} 0,975 \\ 0,888 \\ 1,254 \end{pmatrix}$$

Обновленная диаграмма отражает наш прогресс в расчете преобразования начальных сигналов, поступающих на узлы первого слоя, в сглаженные комбинированные сигналы, поступающие на узлы последнего слоя, в процессе их распространения по нейронной сети.

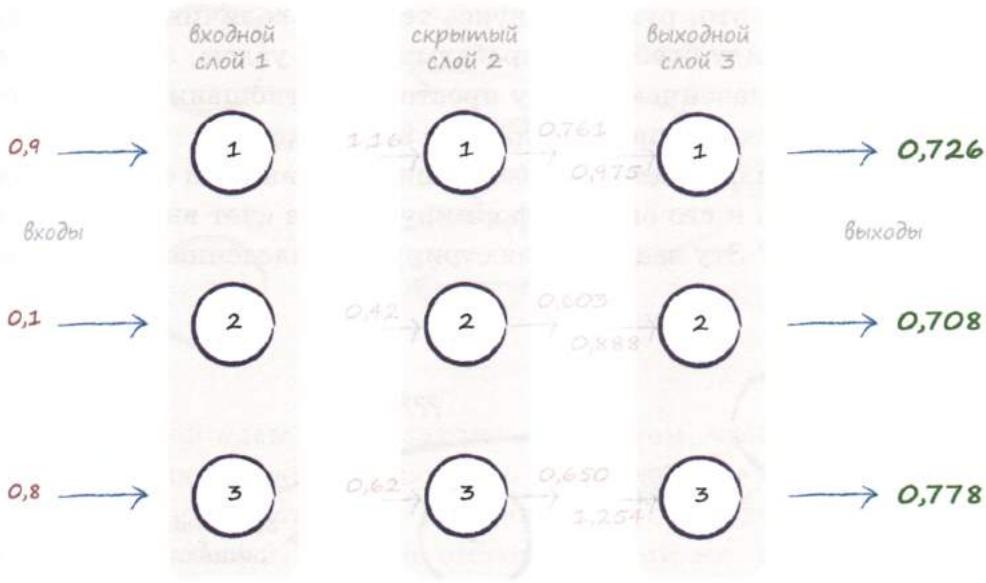


Все, что нам остается, — это применить сигмоиду, и сделать это не составляет труда.

$$O_{\text{выходной}} = \text{сигмоида} \begin{pmatrix} 0,975 \\ 0,888 \\ 1,254 \end{pmatrix}$$

$$O_{\text{выходной}} = \begin{pmatrix} 0,726 \\ 0,708 \\ 0,778 \end{pmatrix}$$

Есть! Мы получили сигналы на выходе нейронной сети. Опять-таки, отобразим текущую ситуацию на обновленной диаграмме.



Таким образом, в нашем примере нейронной сети с тремя слоями выходные сигналы имеют следующую величину: 0,726; 0,708 и 0,778.

Итак, нам удалось успешно описать распространение сигналов по нейронной сети, т.е. определить величину выходных сигналов при заданных величинах входных сигналов.

Что дальше?

Наш следующий шаг заключается в сравнении выходных сигналов нейронной сети с данными тренировочного примера для определения ошибки. Нам необходимо знать величину этой ошибки, чтобы можно было улучшить выходные результаты путем изменения параметров сети.

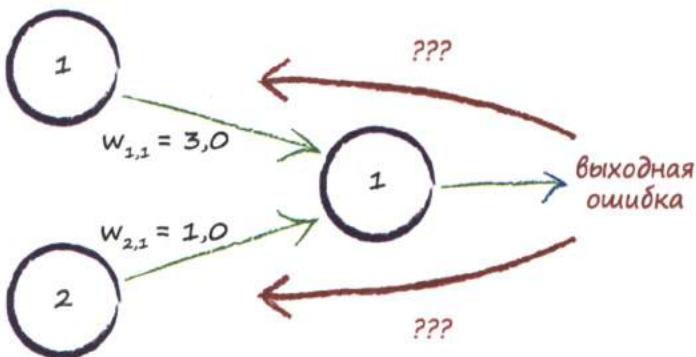
Пожалуй, эта часть работы наиболее трудна для понимания, поэтому мы будем продвигаться не спеша, постепенно знакомясь с основными идеями в процессе работы.

Корректировка весовых коэффициентов в процессе обучения нейронной сети

Ранее мы улучшали поведение простого линейного классификатора путем регулирования параметра наклона линейной функции

узла. Мы делали это, руководствуясь текущей величиной ошибки, т.е. разности между ответом, вырабатываемым узлом, и известным нам истинным значением. Ввиду простоты соотношения между величинами ошибки и поправки это было несложно.

Как нам обновлять весовые коэффициенты связей в случае, если выходной сигнал и его ошибка формируются за счет вкладов более чем одного узла? Эту задачу иллюстрирует приведенная ниже диаграмма.

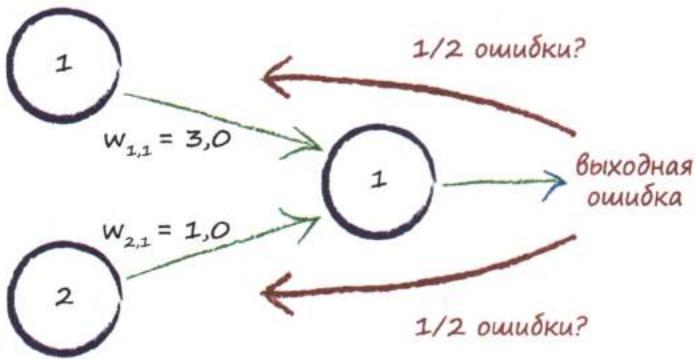


Когда на выходной узел поступал сигнал только от одного узла, все было намного проще. Но как использовать ошибку выходного сигнала при наличии двух входных узлов?

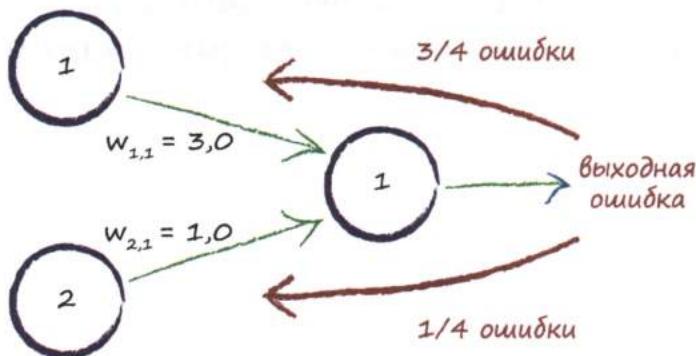
Использовать всю величину ошибки для обновления только одного весового коэффициента не представляется разумным, поскольку при этом полностью игнорируются другая связь и ее вес. Но ведь величина ошибки определяется вкладами всех связей, а не только одной.

Правда, существует небольшая вероятность того, что за ошибку ответственна только одна связь, но эта вероятность пренебрежимо мала. Если мы изменили весовой коэффициент, уже имеющий "правильное" значение, и тем самым ухудшили его, то при выполнении нескольких последующих итераций он должен улучшиться, так что не все потеряно.

Один из возможных способов состоит в том, чтобы распределить ошибку поровну между всеми узлами, вносящими вклад, как показано на следующей диаграмме.



Суть другой идеи также заключается в том, чтобы распределять ошибку между узлами, однако это распределение не обязано быть равномерным. Вместо этого большая доля ошибки приписывается вкладам тех связей, которые имеют больший вес. Почему? Потому что они оказывают большее влияние на величину ошибки. Эту идею иллюстрирует следующая диаграмма.



В данном случае сигнал, поступающий на выходной узел, формируется за счет двух узлов. Весовые коэффициенты связей равны 3,0 и 1,0. Распределив ошибку между двумя узлами пропорционально их весам, вы увидите, что для обновления значения первого, большего веса следует использовать $3/4$ величины ошибки, тогда как для обновления значения второго, меньшего веса — $1/4$.

Мы можем расширить эту идею на случаи с намного большим количеством узлов. Если бы выходной узел был связан со ста входными узлами, мы распределили бы выходную ошибку между всеми ста

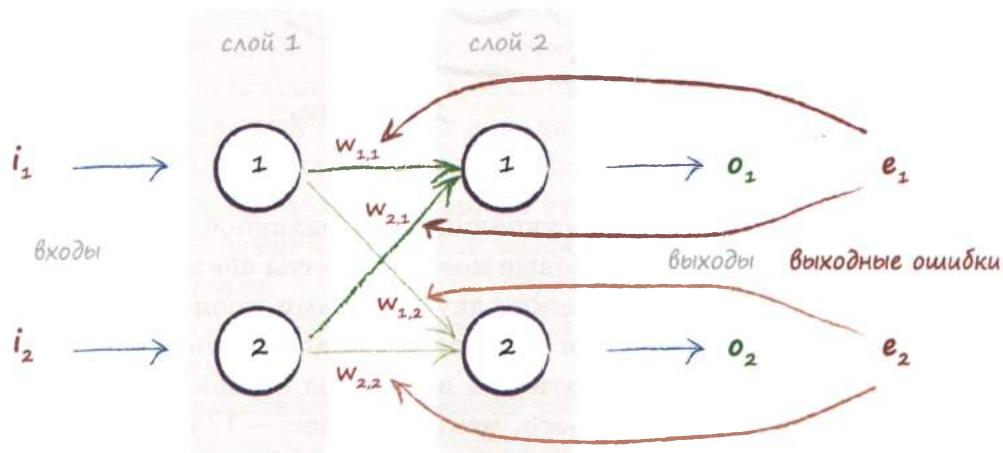
связями пропорционально их вкладам, размер которых определяется весами соответствующих связей.

Как вы могли заметить, мы используем весовые коэффициенты в двух целях. Во-первых, они учитываются при расчете распространения сигналов по нейронной сети от входного слоя до выходного. Мы интенсивно использовали их ранее именно в таком качестве. Во-вторых, мы используем веса для распространения ошибки в обратном направлении — от выходного слоя вглубь сети. Думаю, вы не будете удивлены, узнав, что этот метод называется **обратным распространением ошибки** (обратной связью) в процессе обучения нейронной сети.

Если бы выходной слой имел два узла, мы повторили бы те же действия и для второго узла. Второй выходной узел будет характеризоваться собственной ошибкой, распределаемой аналогичным образом между соответствующим количеством входных узлов. Теперь продолжим наше рассмотрение.

Обратное распространение ошибок от большего количества выходных узлов

На следующей диаграмме отображена простая сеть с двумя входными узлами, но на этот раз с двумя выходными узлами.



Ошибка может формироваться на обоих узлах — фактически эта ситуация очень похожа на ту, которая возникает, когда сеть еще не обучалась. Вы видите, что для коррекции весов внутренних связей нужна информация об ошибках в обоих узлах. Мы можем использовать прежний подход и распределить ошибку выходного узла между связанными с ним узлами пропорционально весовым коэффициентам соответствующих связей.

В действительности тот факт, что сейчас имеется более чем один выходной узел, ничего не меняет. Мы просто повторяем для второго узла те же действия, которые уже выполняли для первого. Почему все так просто? Эта простота объясняется тем, что связи одного выходного узла не зависят от связей другого. Между этими двумя наборами связей отсутствует какая-либо зависимость.

Вернемся к диаграмме, на которой ошибка на первом выходном узле обозначена как e_1 . Не забывайте, что это разность между желаемым значением, предоставляемым тренировочными данными t_1 , и фактическим выходным значением o_1 . Таким образом, $e_1 = (t_1 - o_1)$. Ошибка на втором выходном узле обозначена как e_2 .

На диаграмме видно, что ошибка e_1 распределяется пропорционально весам связей, обозначенным как w_{11} и w_{21} . Точно так же ошибка e_2 должна распределяться пропорционально весам w_{12} и w_{22} .

Чтобы у вас не возникало никаких сомнений в правильности получаемых результатов, запишем эти доли в явном виде. Ошибка e_1 информирует о величинах поправок для весов w_{11} и w_{21} . При ее распределении между узлами доля e_1 , информации о которой используется для обновления w_{11} , определяется следующим выражением:

$$w_{11}$$

$$\frac{w_{11}}{w_{11} + w_{21}}$$

Доля e_1 , используемая для обновления w_{21} , определяется аналогичным выражением:

$$\underline{w_{21}}$$

$$w_{11} + w_{21}$$

Возможно, эти выражения несколько смущают вас, поэтому рассмотрим их более подробно. За всеми этими символами стоит очень простая идея, которая заключается в том, что узлы, сделавшие больший вклад в ошибочный ответ, получают больший сигнал об ошибке, тогда как узлы, сделавшие меньший вклад, получают меньший сигнал.

Если w_{11} в два раза превышает w_{21} (скажем, $w_{11}=6$, а $w_{21}=3$), то доля e_1 , используемая для обновления w_{11} , составляет $6/(6+3) = 6/9 = 2/3$. Тогда для другого, меньшего веса w_{21} должно остаться $1/3 e_1$, что можно подтвердить с помощью выражения $3/(6+3) = 3/9$, результат которого действительно равен $1/3$.

Как и следовало ожидать, при равных весах будут равны и соответствующие доли. Давайте в этом убедимся. Пусть $w_{11}=4$ и $w_{21}=4$, тогда в обоих случаях доля будет составлять $4/(4+4) = 4/8 = 1/2$.

Прежде чем продолжить, сделаем паузу и вернемся на шаг назад, чтобы оценить то, что мы уже успели сделать. Мы знали, что при определении величины поправок для некоторых внутренних параметров сети, в данном случае — весов связей, необходимо использовать величину ошибки. Вы видели, как это делается для весов, которые сглаживают входные сигналы последнего, выходного слоя нейронной сети. Вы также видели, что увеличение количества выходных узлов не усложняет задачу, поскольку мы повторяем одни и те же действия для каждого выходного узла. Замечательно!

Но как быть, если количество слоев превышает два? Как обновлять веса связей для слоев, далеко отстоящих от последнего, выходного слоя?

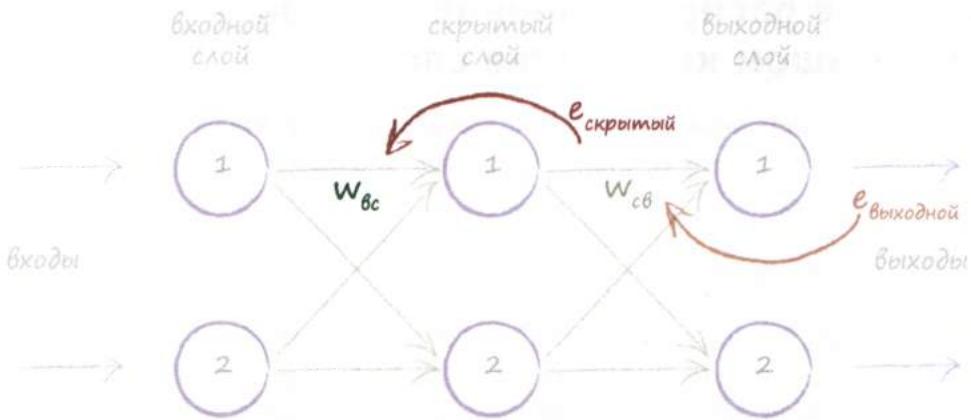
Обратное распространение ошибок при большом количестве слоев

На следующей диаграмме представлен пример простой нейронной сети с тремя слоями: входным, скрытым и выходным.



Продвигаясь в обратном направлении от последнего, выходного слоя (крайнего справа), мы видим, как информация об ошибке в выходном слое используется для определения величины поправок к весовым коэффициентам связей, со стороны которых к нему поступают сигналы. Здесь использованы более общие обозначения $e_{\text{выходной}}$ для выходных ошибок и w_{cv} для весов связей между скрытым и выходным слоями. Мы вычисляем конкретные ошибки, ассоциируемые с каждой связью, путем распределения ошибки пропорционально соответствующим весам.

Графическое представление позволяет лучше понять, какие вычисления следует выполнить для дополнительного слоя. Мы просто берем ошибки $e_{\text{скрытый}}$ на выходе скрытого слоя и вновь распределяем их по предшествующим связям между входным и скрытым слоями пропорционально весовым коэффициентам w_{bc} . Следующая диаграмма иллюстрирует эту логику.

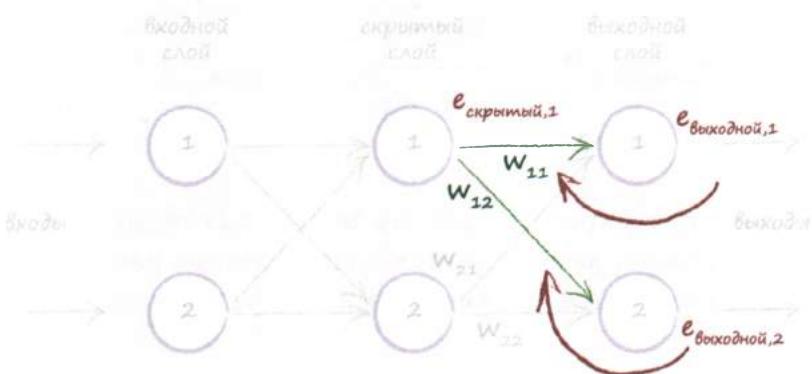


При наличии большего количества слоев мы просто повторили бы описанную процедуру для каждого слоя, продвигаясь в обратном направлении от последнего, выходного слоя. Идея распространения этого потока информации об ошибке (сигнала об ошибке) интуитивно понятна. Вы еще раз имели возможность увидеть, почему этот процесс описывается термином **обратное распространение ошибок**.

Если мы сначала использовали ошибку $e_{\text{выходной}}$ выходного сигнала узлов выходного слоя, то какую ошибку $e_{\text{скрытый}}$ мы собираемся использовать для узлов скрытого слоя? Это хороший вопрос, поскольку мы не можем указать очевидную ошибку для узла в таком слое. Из расчетов, связанных с распространением входных сигналов в прямом направлении, мы знаем, что у каждого узла скрытого слоя в действительности имеется только один выход. Вспомните, как мы применяли функцию активации к взвешенной сумме всех входных сигналов данного узла. Но как определить ошибку для такого узла?

У нас нет целевых или желаемых выходных значений для скрытых узлов. Мы располагаем лишь целевыми значениями узлов последнего, выходного слоя, и эти значения происходят из тренировочных примеров. Попытаемся найти вдохновение, взглянув еще раз на приведенную выше диаграмму. С первым узлом скрытого слоя ассоциированы две исходящие из него связи, ведущие к двум узлам выходного слоя. Мы знаем, что можем распределить выходную ошибку между этими связями, поступая точно так же, как и прежде.

Это означает, что с каждой из двух связей, исходящих из узла промежуточного слоя, ассоциируется некоторая ошибка. Мы могли бы воссоединить ошибки этих двух связей, чтобы получить ошибку для этого узла в качестве второго наилучшего подхода, поскольку мы не располагаем фактическим целевым значением для узла промежуточного слоя. Следующая диаграмма иллюстрирует эту идею:



На диаграмме отчетливо видно, что именно происходит, однако для уверенности давайте разберем ее более детально. Нам необходимы величины ошибок для узлов скрытого слоя, чтобы использовать их для обновления весовых коэффициентов связей с предыдущим слоем. Обозначим эти ошибки как $e_{\text{скрытый}}$. Но у нас нет очевидного ответа на вопрос о том, какова их величина. Мы не можем сказать, что ошибка — это разность между желаемым или целевым выходным значением этого узла и его фактическим выходным значением, поскольку данные тренировочного примера предоставляют лишь целевые значения для узлов последнего, выходного слоя. Они не говорят абсолютно ничего о том, какими должны быть выходные сигналы узлов любого другого слоя. В этом и заключается суть сложившейся головоломки.

Мы можем воссоединить ошибки, распределенные по связям, используя обратное распространение ошибок, с которым вы уже познакомились. Поэтому ошибка на первом скрытом узле представляет собой сумму ошибок, распределенных по всем связям, исходящим из

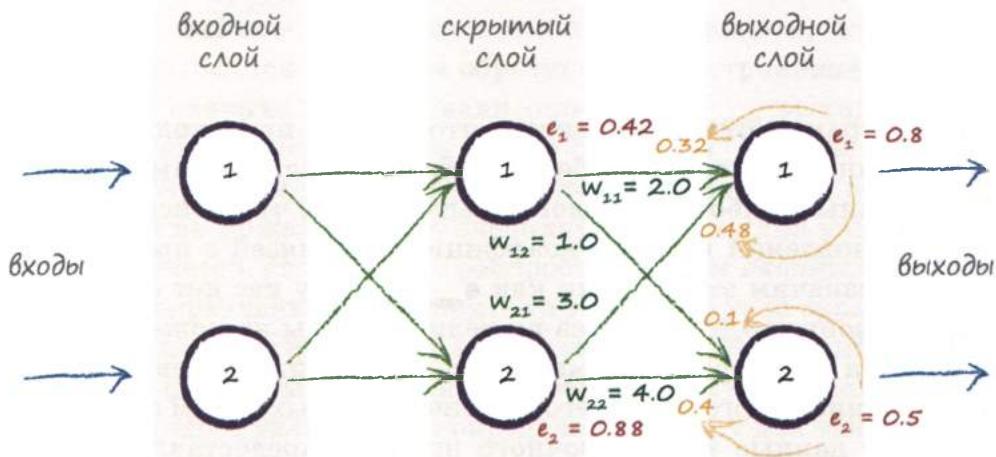
этого узла в прямом направлении. На приведенной выше диаграмме показано, что имеется некоторая доля выходной ошибки $e_{\text{выходной},1}$, приписываемая связи с весом w_{11} , и некоторая доля выходной ошибки $e_{\text{выходной},2}$, приписываемая связи с весом w_{12} .

Вышесказанное можно записать в виде следующего выражения:

$$e_{\text{скрытый},1} = \text{сумма ошибок, распределенных по связям } w_{11} \text{ и } w_{12}$$

$$= e_{\text{выходной},1} * \frac{w_{11}}{w_{11} + w_{21}} + e_{\text{выходной},2} * \frac{w_{12}}{w_{12} + w_{22}}$$

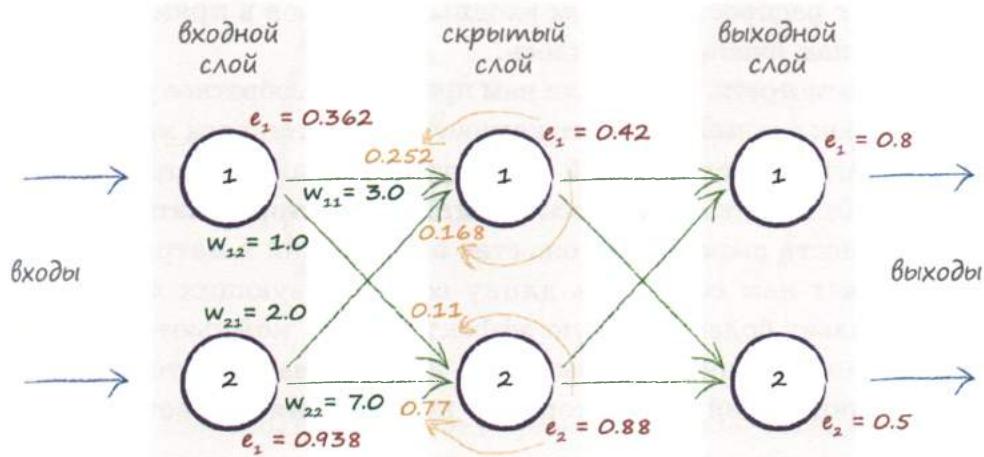
Чтобы проиллюстрировать, как эта теория выглядит на практике, приведем диаграмму, демонстрирующую обратное распространение ошибок в простой трехслойной сети на примере конкретных данных.



Проследим за обратным распространением одной из ошибок. Вы видите, что после распределения ошибки 0,5 на втором узле выходного слоя между двумя связями с весами 1,0 и 4,0 мы получаем доли, равные 0,1 и 0,4 соответственно. Также можно видеть, что объединенная ошибка на втором узле скрытого слоя представляет

собой сумму распределенных ошибок, в данном случае равных 0,48 и 0,4, сложение которых дает 0,88.

На следующей диаграмме демонстрируется применение той же методики к слою, который предшествует скрытому.



Резюме

- Нейронные сети обучаются посредством уточнения весовых коэффициентов своих связей. Этот процесс управляет ошибкой — разностью между правильным ответом, предоставляемым тренировочными данными, и фактическим выходным значением.
- Ошибка на выходных узлах определяется простой разностью между желаемым и фактическим выходными значениями.
- В то же время величина ошибки, связанной с внутренними узлами, не столь очевидна. Одним из способов решения этой проблемы является распределение ошибок выходного слоя между соответствующими связями пропорционально весу каждой связи с последующим объединением соответствующих разрозненных частей ошибки на каждом внутреннем узле.

Описание обратного распространения ошибок с помощью матричной алгебры

Можем ли мы упростить трудоемкие расчеты, используя возможности матричного умножения? Ранее, когда мы проводили расчеты, связанные с распространением входных сигналов в прямом направлении, это нам очень пригодилось.

Чтобы посмотреть, удастся ли нам представить обратное распространение ошибок с помощью более компактного синтаксиса матриц, опишем все шаги вычислительной процедуры, используя матричные обозначения. Кстати, тем самым мы попытаемся **векторизовать** процесс.

Возможность выразить множество вычислений в матричной форме позволяет нам сократить длину соответствующих выражений и обеспечивает более высокую эффективность компьютерных расчетов, поскольку компьютеры могут использовать повторяющийся шаблон вычислений для ускорения выполнения соответствующих операций.

Отправной точкой нам послужат ошибки, возникающие на выходе нейронной сети в последнем, выходном слое. В данном случае выходной слой содержит только два узла с ошибками e_1 и e_2 :

$$\text{ошибка}_{\text{выходной}} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Далее нам нужно построить матрицу для ошибок скрытого слоя. Эта задача может показаться сложной, поэтому мы будем выполнять ее по частям. Первую часть задачи представляет первый узел скрытого слоя. Взглянув еще раз на приведенные выше диаграммы, вы увидите, что ошибка на первом узле скрытого слоя формируется за счет двух вкладов со стороны выходного слоя. Этими двумя сигналами ошибок являются $e_1 * w_{11} / (w_{11} + w_{21})$ и $e_2 * w_{12} / (w_{12} + w_{22})$. А теперь обратите внимание на второй узел скрытого слоя, и вы вновь увидите, что ошибка на нем также формируется за счет двух вкладов:

$e_1 * w_{21} / (w_{21} + w_{11})$ и $e_2 * w_{22} / (w_{22} + w_{12})$. Ранее мы уже видели, как работают эти выражения.

Итак, для скрытого слоя мы имеем следующую матрицу, которая выглядит немного сложнее, чем мне хотелось бы.

$$\text{ошибка}_{\text{скрытый}} = \begin{pmatrix} \frac{w_{11}}{w_{11} + w_{21}} & \frac{w_{12}}{w_{12} + w_{22}} \\ \frac{w_{21}}{w_{21} + w_{11}} & \frac{w_{22}}{w_{22} + w_{12}} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Было бы здорово, если бы это выражение можно было переписать в виде простого перемножения матриц, которыми мы уже располагаем. Это матрицы весовых коэффициентов, прямого сигнала и выходных ошибок. Преимущества, которые можем при этом получить, огромны.

К сожалению, легкого способа превратить это выражение в сверхпростое перемножение матриц, как в случае распространения сигналов в прямом направлении, не существует. Распутать все эти доли, из которых образованы элементы большой матрицы, непросто. Было бы замечательно, если бы мы смогли представить эту матрицу в виде комбинации имеющихся матриц.

Что можно сделать? Нам позарез нужен способ, обеспечивающий возможность использования матричного умножения, чтобы повысить эффективность вычислений.

Ну что ж, дерзнем!

Взгляните еще раз на приведенное выше выражение. Вы видите, что наиболее важная для нас вещь — это умножение выходных ошибок e_n на связанные с ними веса w_{ij} . Чем больше вес, тем большая доля ошибки передается обратно в скрытый слой. Это важный момент. В дробях, являющихся элементами матрицы, нижняя часть играет роль нормирующего множителя. Если пренебречь этим фактором, можно потерять лишь масштабирование ошибок, передаваемых по меха-

низму обратной связи. Таким образом, выражение $e_1 * w_{11} / (w_{11} + w_{21})$ упростится до $e_1 * w_{11}$.

Сделав это, мы получим следующее уравнение.

$$\text{ошибка}_{\text{скрытый}} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Эта матрица весов напоминает ту, которую мы строили ранее, но она повернута вокруг диагонали, так что правый верхний элемент теперь стал левым нижним, а левый нижний — правым верхним. Такая матрица называется **транспонированной** и обозначается как w^T .

Ниже приведены два примера транспонирования числовых матриц, которые помогут вам лучше понять смысл операции транспонирования. Вы видите, что она применима даже в тех случаях, когда количество столбцов в матрице отличается от количества строк.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Итак, мы достигли того, чего хотели, — применили матричный подход к описанию обратного распространения ошибок:

$$\text{ошибка}_{\text{скрытый}} = w^T_{\text{скрытый_выходной}} \cdot \text{ошибка}_{\text{выходной}}$$

Конечно, это просто замечательно, но правильно ли мы поступили, отбросив нормирующий множитель? Оказывается, что эта упрощенная модель обратного распространения сигналов ошибок работает ничуть не хуже, чем более сложная, которую мы разработали перед этим. В блоге, посвященном данной книге, вы найдете публикацию, в которой представлены результаты расчетов обратного распространения ошибки, выполненных несколькими различными способами:

<http://makeyourownneuralnetwork.blogspot.co.uk/2016/07/error-backpropagation-revisted.html>

Если наш простой подход действительно хорошо работает, мы оставим его!

Немного подумав, можно прийти к выводу, что даже в тех случаях, когда в обратном направлении распространяются слишком большие или слишком малые ошибки, сеть сама все исправит при выполнении последующих итераций обучения. Важно то, что при обратном распространении ошибок учитываются весовые коэффициенты связей, и это наилучший показатель того, что мы пытаемся справедливо распределить ответственность за возникающие ошибки.

Мы проделали большую работу, очень большую!

Резюме

- Обратное распространение ошибок можно описать с помощью матричного умножения.
- Это позволяет нам записывать выражения в более компактной форме, независимо от размеров нейронной сети, и обеспечивает более эффективное и быстрое выполнение вычислений компьютерами, если в языке программирования предусмотрен синтаксис матричных операций.
- Отсюда следует, что использование матриц обеспечивает повышение эффективности расчетов как для распространения сигналов в прямом направлении, так и для распространения ошибок в обратном направлении.

Сделайте вполне заслуженный перерыв, поскольку следующий теоретический раздел потребует от вас концентрации внимания и приложения умственных усилий.

Как мы фактически обновляем весовые коэффициенты

Однако мы пока что не приступили к решению главной задачи — обновлению весов связей в нейронной сети. Мы работали в этом направлении и уже почти достигли намеченной цели. Нам осталось разобрать лишь одну ключевую идею, чтобы больше не было никаких неясностей.

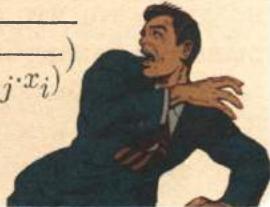
К этому моменту мы научились рассчитывать обратное распространение ошибок до каждого слоя сети. Зачем нам это нужно? Затем, что ошибки подсказывают нам, как должны быть изменены веса связей, чтобы улучшить результирующий общий ответ на выходе нейронной сети. В основном это то, что мы делали с линейным классификатором еще в начале главы.

Однако узлы — это не простые линейные классификаторы. В узлах сигналы суммируются с учетом весов, после чего к ним применяется сигмоида. Но как нам все-таки справиться с обновлением весов для связей, соединяющих эти более сложные узлы? Почему бы не использовать алгебру для непосредственного вычисления весов?

Последний путь нам не подходит ввиду громоздкости соответствующих выкладок. Существует слишком много комбинаций весов и слишком много функций, зависящих от функций, зависящих от других функций и т.д., которые мы должны комбинировать в ходе анализа распространения сигнала по сети. Представьте себе хотя бы небольшую нейронную сеть с тремя слоями и тремя нейронами в каждом слое, подобную той, с которой мы перед этим работали. Как отрегулировать весовой коэффициент для связи между первым входным узлом и вторым узлом скрытого слоя, чтобы сигнал на выходе третьего узла увеличился, скажем, на 0,5? Даже если бы вам повезло и вы сделали это, достигнутый результат мог бы быть разрушен настройкой другого весового коэффициента, улучшающего сигнал другого выходного узла. Как видите, эти расчеты далеко не тривиальны.

Чтобы убедиться в том, насколько они не тривиальны, достаточно взглянуть на приведенное ниже устрашающее выражение, которое представляет выходной сигнал узла выходного слоя как функцию входных сигналов и весовых коэффициентов связей для простой нейронной сети с тремя слоями по три узла. Входной сигнал на узле i равен x_i , весовой коэффициент для связи, соединяющей входной узел i с узлом j скрытого слоя, равен $w_{i,j}$. Аналогичным образом выходной сигнал узла j скрытого слоя равен x_j , а весовой коэффициент для связи, соединяющей узел j скрытого слоя с выходным узлом k , равен $w_{j,k}$. Необычный символ Σ_a^b означает суммирование следующего за ним выражения по всем значениям между a и b .

$$o_k = \frac{1}{1 + e^{-\sum_{j=1}^3 (w_{j,k} \cdot \frac{1}{1 + e^{-\sum_{i=1}^3 (w_{i,j} \cdot x_i)}})}}$$



Ничего себе! Лучше держаться от этого подальше.

А не могли бы мы вместо того, чтобы пытаться выглядеть очень умными, просто перебирать случайные сочетания весовых коэффициентов, пока не будет получен устраивающий нас результат?

Этот вопрос не столь уж наивен, как могло бы показаться, особенно когда задача действительно трудная. Такой подход называется **методом грубой силы**. Некоторые люди пытаются использовать методы грубой силы для того, чтобы взламывать пароли, и это может сработать, если паролем является какое-либо осмысленное слово, причем не очень длинное, а не просто набор символов. Такая задача вполне по силам достаточно мощному домашнему компьютеру. Но представьте, что каждый весовой коэффициент может иметь 1000 возможных значений в диапазоне от -1 до $+1$, например 0,501, -0,203 или 0,999. Тогда в случае нейронной сети с тремя слоями

по три узла, насчитывающей 18 весовых коэффициентов, мы должны были бы протестировать 18 тысяч возможностей. Если бы у нас была более типичная нейронная сеть с 500 узлами в каждом слое, то нам пришлось бы протестировать 500 миллионов различных значений весов. Если бы для расчета каждого набора комбинаций требовалась одна секунда, то для обновления весов с помощью всего лишь одного тренировочного примера понадобилось бы примерно 16 лет. Тысяча тренировочных примеров — и мы имели бы 16 тысяч лет!

Как видите, подход, основанный на методе грубой силы, практически нереализуем. В действительности по мере добавления в сеть новых слоев, узлов или вариантов перебора весов ситуация очень быстро ухудшается.

Эта головоломка не поддавалась математикам на протяжении многих лет и получила реальное практическое разрешение лишь в 1960–1970-х годах. Существуют различные мнения относительно того, кто сделал это впервые или совершил ключевой прорыв, но для нас важно лишь то, что это запоздалое открытие послужило толчком к взрывному развитию современной теории нейронных сетей, которая сейчас способна решать некоторые весьма впечатляющие задачи.

Но все же, как нам разрешить эту явно трудную проблему? Хотите верьте, хотите нет, но вы уже владеете средствами, с помощью которых сможете справиться с этим самостоятельно. Все, что вам для этого нужно знать, мы уже обсуждали. Теперь можем продолжить.

Самое главное, что требуется от нас, — это не бояться пессимизма.

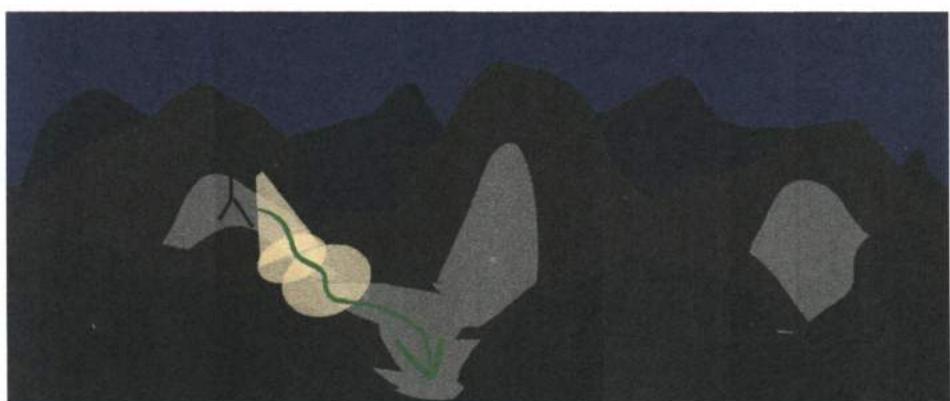
Математические выражения, позволяющие определить выходной сигнал нейронной сети при известных весовых коэффициентах, необычайно сложны, и в них нелегко разобраться. Количество различных комбинаций слишком велико для того, чтобы пытаться тестировать их поочередно.

Для пессимизма есть и ряд других причин. Тренировочных данных может оказаться недостаточно для эффективного обучения сети. В тренировочных данных могут быть ошибки, в связи с чем справедливость нашего предположения о том, что они истинны и на них можно учиться, оказывается под вопросом. Количество слоев или узлов в самой сети может быть недостаточным для того, чтобы правильно моделировать решение задачи.

Это означает, что предпринимаемый нами подход должен быть реалистичным и учитывать указанные ограничения. Если мы будем

следовать этим принципам, то, возможно, найдем решение, которое, даже не будучи идеальным с математической точки зрения, даст нам лучшие результаты, поскольку не будет основано на ложных идеалистических допущениях.

Проиллюстрируем суть наших рассуждений на следующем примере. Представьте себе ландшафт с очень сложным рельефом, имеющим возвышения и впадины, а также холмы с предательскими буграми и ямами. Вокруг так темно, что ни зги не видно. Вы знаете, что находитесь на склоне холма, и вам нужно добраться до его подножия. Точной карты местности у вас нет. Но у вас есть фонарь. Что вы будете делать? Пожалуй, вы воспользуетесь фонарем и осмотритесь вокруг себя. Света фонаря не хватит для дальнего обзора, и вы наверняка не сможете осмотреть весь ландшафт целиком. Но вы сможете увидеть, по какому участку проще всего начать спуск к подножию холма, и сделаете несколько небольших шагов в этом направлении. Действуя подобным образом, вы будете медленно, шаг за шагом, продвигаться вниз, не располагая общей картой и заблаговременно проложенным маршрутом.



Математическая версия этого подхода называется методом градиентного спуска, и причину этого нетрудно понять. После того как вы сделали шаг в выбранном направлении, вы вновь осматриваетесь, чтобы увидеть, какой путь ведет вас к цели, и делаете очередной шаг в этом направлении. Вы продолжаете действовать точно так же до тех пор, пока благополучно не спуститесь к подножию холма.

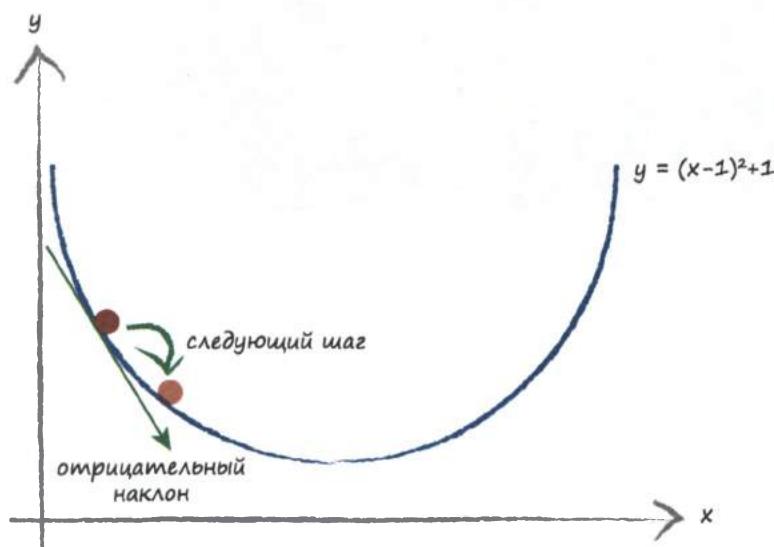
А теперь представьте, что этим сложным ландшафтом является математическая функция. Метод градиентного спуска позволяет

находить минимум, даже не располагая знаниями свойств этой функции, достаточными для нахождения минимума математическими методами. Если функция настолько сложна, что простого способа нахождения минимума алгебраическими методами не существует, то мы можем вместо этого применить метод градиентного спуска. Ясное дело, он может не дать нам точный ответ, поскольку мы приближаемся к ответу шаг за шагом, постепенно улучшая нашу позицию. Но это лучше, чем вообще не иметь никакого ответа. Во всяком случае, мы можем продолжить уточнение ответа еще более мелкими шагами по направлению к минимуму, пока не достигнем желаемой точности.

А какое отношение имеет этот действительно эффективный метод градиентного спуска к нейронным сетям? Если упомянутой сложной функцией является ошибка сети, то спуск по склону для нахождения минимума означает, что мы минимизируем ошибку. Мы улучшаем выходной сигнал сети. Это именно то, чего мы хотим!

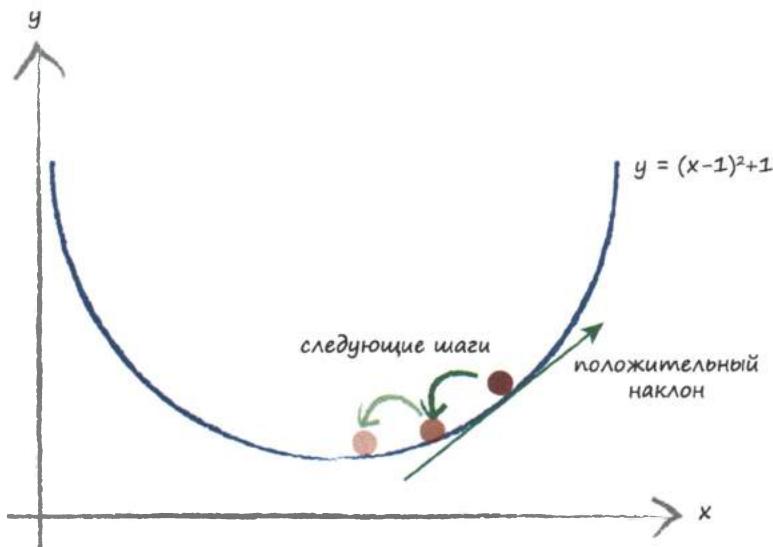
Чтобы вы все наглядно усвоили, рассмотрим использование метода градиентного спуска на простейшем примере.

Ниже приведен график простой функции $y = (x - 1)^2 + 1$. Если бы это была функция, описывающая ошибку, то мы должны были бы найти значение x , которое минимизирует эту функцию. Представим на минуту, что мы имеем дело не со столь простой функцией, а с гораздо более сложной.



Градиентный спуск должен с чего-то начинаться. На графике показана случайно выбранная начальная точка. Подобно покорителю гор, мы исследуем место, в котором находимся, и видим, в каком направлении идет спуск. Наклон кривой также обозначен на графике, и в данном случае ему соответствует отрицательный градиент. Мы хотим следовать в направлении вниз, поэтому движемся вдоль оси x вправо. Таким образом, мы немного увеличиваем x . Это первый шаг нашего альпиниста. Вы видите, что мы улучшили нашу позицию и продвинулись ближе к фактическому минимуму.

Представим, что мы начали спуск с какого-то другого места, как показано на следующем графике.

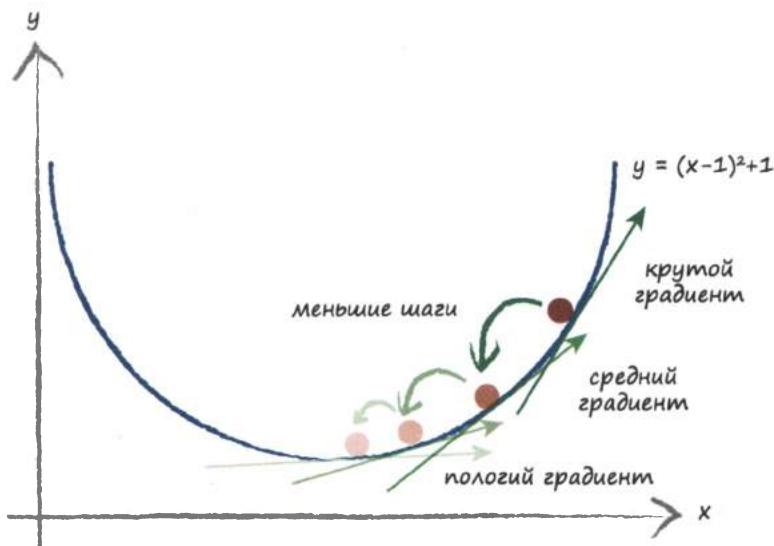


На этот раз наклон почвы под нашими ногами положителен, поэтому мы движемся влево, т.е. немного уменьшаем x . И вновь вы видите, что мы улучшили наше положение, приблизившись к фактическому минимуму. Мы можем продолжать действовать в том же духе до тех пор, пока изменения не станут настолько малыми, что мы будем считать, что достигли минимума.

Необходимым усовершенствованием этого метода должно быть изменение величины шагов во избежание перескока через минимум, что приведет к бесконечным прыжкам вокруг него. Вообразите, что мы оказались на расстоянии 0,5 метра от истинного минимума, но

длина нашего шага всегда равна 2 метра. Тогда мы будем постоянно пропускать минимум, поскольку на каждом шаге в его направлении мы будем перескакивать через него. Если мы будем уменьшать величину шага пропорционально величине градиента, то по мере приближения к минимуму будем совершать все более мелкие шаги. При этом мы предполагаем, что чем ближе к минимуму, тем меньше наклон. Для большинства гладких (непрерывно дифференцируемых) функций такое предположение вполне приемлемо. Оно не будет справедливым лишь по отношению к экзотическим зигзагообразным функциям со взлетами и провалами в точках, которые математики называют точками разрыва.

Идея уменьшения величины шага по мере уменьшения величины градиента, являющейся хорошим индикатором близости к минимуму, иллюстрируется следующим графиком.

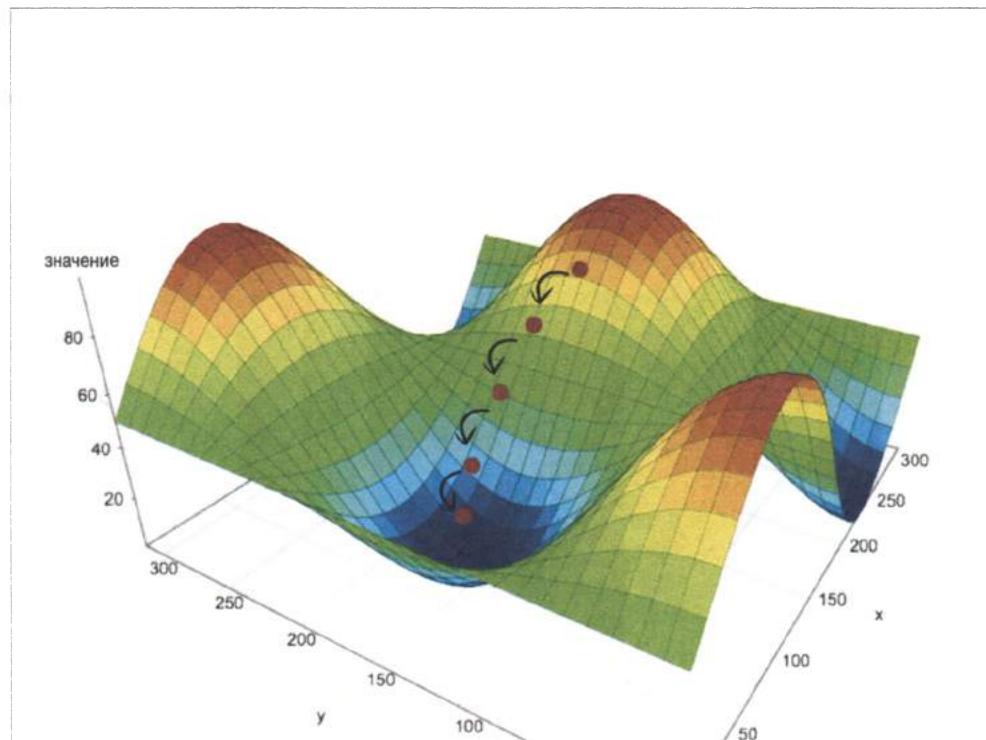


Кстати, обратили ли вы внимание на то, что мы изменяем x в направлении, противоположном направлению градиента? Положительный градиент означает, что мы должны уменьшить x . Отрицательный градиент требует увеличения x . Все это отчетливо видно на графике, но это легко упустить из виду и пойти совершенно неправильным путем.

Используя метод градиентного спуска, мы не пытались находить истинный минимум алгебраическим методом, поскольку сделали вид, будто функция $y = (x - 1)^2 + 1$ для этого слишком сложна. Даже если бы мы не могли определить наклон кривой с математической точностью, мы могли бы оценить его, и, как нетрудно заметить, это все равно вело бы нас в правильном общем направлении.

Вся мощь этого метода по-настоящему проявляется в случае функций, зависящих от многих параметров. Например, вместо зависимости y от x мы можем иметь зависимость от a, b, c, d, e и f . Вспомните, что функция выходного сигнала, а вместе с ней и функция ошибки зависят от множества весовых коэффициентов, которые часто исчисляются сотнями!

Следующий график вновь иллюстрирует метод градиентного спуска, но на этот раз применительно к более сложной функции, зависящей от двух параметров. График такой функции можно представить в трех измерениях, где высота представляет значение функции.



Возможно, глядя на эту трехмерную поверхность, вы задумались над тем, может ли метод градиентного спуска привести в другую долину, которая расположена справа. В действительности этот вопрос можно обобщить: не приводит ли иногда метод градиентного спуска в ложную долину, поскольку некоторые сложные функции имеют множество долин?

Что такое ложная долина? Это долина, которая не является самой глубокой. Тогда на поставленный вопрос следует дать утвердительный ответ: да, такое может происходить.

На следующей иллюстрации показаны три варианта градиентного спуска, один из которых приводит к ложному минимуму.



Давайте немного передохнем и соберемся с мыслями.

Резюме

- Метод градиентного спуска — это действительно хороший способ нахождения минимума функции, и он прекрасно работает, когда функция настолько сложна, что ее математическая обработка алгебраическими методами сопряжена с большими трудностями.
- Более того, этот метод хорошо работает в случае функций многих переменных, когда другие методы не срабатывают либо их невозможно реализовать на практике.
- Данный метод также **устойчив** к наличию дефектных данных и не заведет вас далеко в неправильную сторону, если функция не описывается идеально или же время от времени мы совершаем неверные шаги.

Выходной сигнал нейронной сети представляет собой сложную, трудно поддающуюся описанию функцию со многими параметрами, весовыми коэффициентами связей, которые влияют на выходной

сигнал. Так можем ли мы использовать метод градиентного спуска для определения подходящих значений весов? Можем, если правильно выберем функцию ошибки.

Функция выходного сигнала сама по себе не является функцией ошибки. Но мы знаем, что можем легко превратить ее в таковую, поскольку ошибка — это разность между целевыми тренировочными значениями и фактическими выходными значениями.

Однако здесь есть кое-что, чего следует осторегаться. Взгляните на приведенную ниже таблицу с тренировочными данными и фактическими значениями для трех выходных узлов вместе с кандидатами на роль функции ошибок.

Выход сети	Целевой результат	Ошибка (целевое – фактическое)	Ошибка $ целевое – фактическое $	Ошибка $(целевое – фактическое)^2$
0,4	0,5	0,1	0,1	0,01
0,8	0,7	-0,1	0,1	0,01
1,0	1,0	0	0	0
Сумма		0	0,2	0,02

Нашим первым кандидатом на роль функции ошибки является простая разность значений (**целевое – фактическое**). Это кажется вполне разумным, не так ли? Но если вы решите использовать сумму ошибок по всем узлам в качестве общего показателя того, насколько хорошо обучена сеть, то эта сумма равна нулю!

Что случилось? Ясно, что сеть еще недостаточно натренирована, поскольку выходные значения двух узлов отличаются от целевых значений. Но нулевая сумма означает отсутствие ошибки. Это объясняется тем, что положительная и отрицательная ошибки взаимно сократились. Отсюда следует, что простая разность значений, даже если бы их взаимное сокращение было неполным, не годится для использования в качестве меры величины ошибки.

Пойдем другим путем, взяв абсолютную величину разности. Формально это записывается как $|целевое – фактическое|$ и означает, что знак результата вычитания игнорируется. Это могло бы сработать, поскольку в данном случае ничто ни с чем не может сократиться. Причина, по которой данный метод не получил популярности,

связана с тем, что при этом наклон не является непрерывной функцией вблизи минимума, что затрудняет использование метода градиентного спуска, поскольку мы будем постоянно совершать скачки вокруг V-образной долины, характерной для функции ошибки подобного рода. При приближении к минимуму наклон, а вместе с ним и величина шага изменения переменной, не уменьшается, а это означает риск перескока.

Третий вариант заключается в том, чтобы использовать в качестве меры ошибки квадрат разности: (**целевое-фактическое**)². Существует несколько причин, по которым третий вариант предпочтительнее второго, включая следующие:

- он упрощает вычисления, с помощью которых определяется величина наклона для метода градиентного спуска;
- функция ошибки является непрерывно гладкой, что обеспечивает нормальную работу метода градиентного спуска ввиду отсутствия провалов и скачков значений функции;
- по мере приближения к минимуму градиент уменьшается, что означает снижение риска перескока через минимум, если используется уменьшение величины шагов.

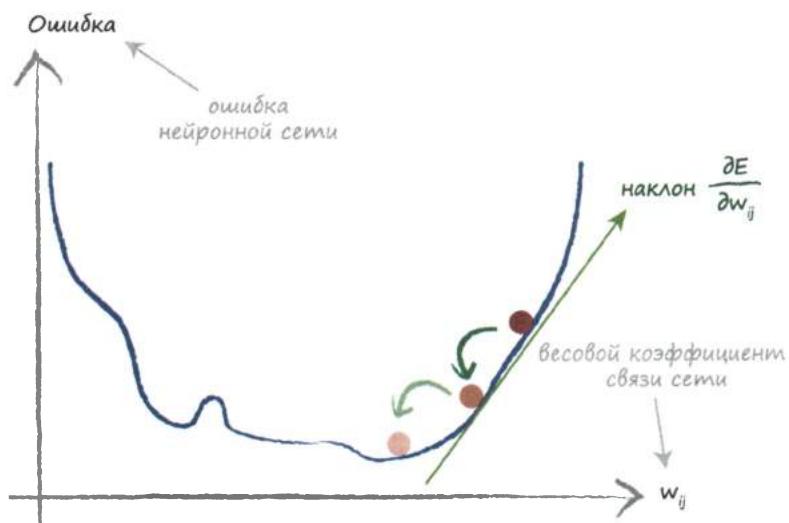
А возможны ли другие варианты? Да, вы вправе сконструировать любую сложную функцию, которую считаете нужной. Одни из них могут вообще не работать, другие могут хорошо работать для определенного круга задач, а третьи могут работать действительно хорошо, но их чрезмерная сложность приводит к неоправданным затратам ресурсов.

А тем временем мы вышли на финишную прямую!

Чтобы воспользоваться методом градиентного спуска, нам нужно определить наклон функции ошибки по отношению к весовым коэффициентам. Это требует применения **дифференциального исчисления**. Возможно, вы уже знакомы с ним, а если не знакомы или вам требуется освежить свою память, то обратитесь к приложению А. Дифференциальное исчисление — это просто математически строгий подход к определению величины изменения одних величин при изменении других. Например, оно позволяет ответить на вопрос о том, как изменяется длина пружины в зависимости от величины усилия,

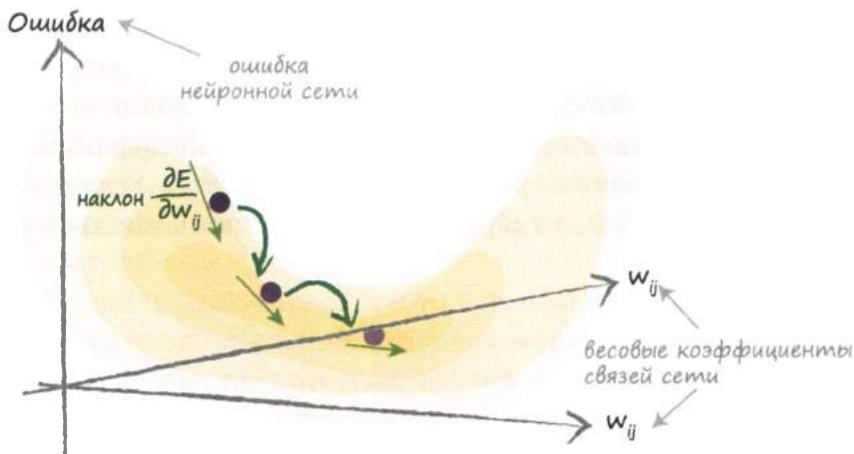
приложенного к ее концам. В данном случае нас интересует зависимость функции ошибки от весовых коэффициентов связей внутри нейронной сети. Иными словами, нас интересует, насколько величина ошибки чувствительна к изменениям весовых коэффициентов.

Начнем с рассмотрения графика, поскольку это всегда позволяет почувствовать под ногами твердую почву, когда пытаешься решать трудную задачу.



Этот график в точности повторяет один из тех, которые приводились ранее, чтобы подчеркнуть, что мы не делаем ничего принципиально нового. На этот раз функцией, которую мы пытаемся минимизировать, является ошибка на выходе нейронной сети. В этом простом примере показан лишь один весовой коэффициент, но мы знаем, что в нейронных сетях их будет намного больше.

На следующей диаграмме отображаются два весовых коэффициента, и поэтому функция ошибки графически отображается в виде трехмерной поверхности, высота расположения точек которой изменяется с изменением весовых коэффициентов связей. Как видите, теперь процесс минимизации ошибки больше напоминает спуск в долину по рельефной местности.



Визуализировать многомерную поверхность ошибки как функции намного большего количества параметров значительно труднее, но идея нахождения минимума методом градиентного спуска остается той же.

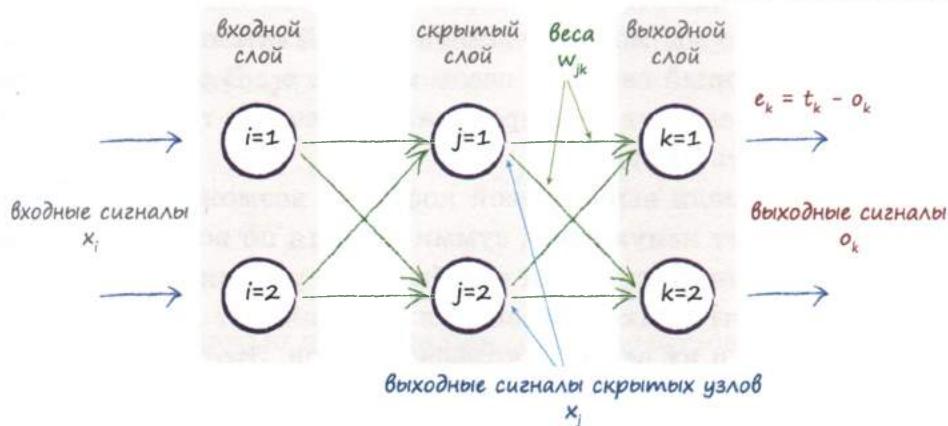
Сформулируем на языке математики, чего мы хотим:

$$\frac{\partial E}{\partial w_{jk}}$$

Это выражение представляет изменение ошибки E при изменении веса w_{jk} . Это и есть тот наклон функции ошибки, который нам нужно знать, чтобы начать градиентный спуск к минимуму.

Прежде чем мы развернем это выражение, временно сосредоточим внимание только на весовых коэффициентах связей между скрытым слоем и последним выходным слоем. Интересующая нас область выделена на приведенной ниже диаграмме. К связям между входным и скрытым слоями мы вернемся позже.

ошибка узла = целевое значение - фактическое значение



Мы будем постоянно ссылаться на эту диаграмму, дабы в процессе вычислений не забыть о том, что в действительности означает каждый символ. Пусть вас это не смущает, поскольку шаги вычислительной процедуры не являются сложными и будут объясняться, а все необходимые понятия ранее уже обсуждались.

Прежде всего запишем в явном виде функцию ошибки, которая представляет собой сумму возвещенных в квадрат разностей между целевым и фактическим значениями, где суммирование осуществляется по всем n выходным узлам.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_n (t_n - o_n)^2$$

Здесь мы всего лишь записали, что на самом деле представляет собой функция ошибки E .

Мы можем сразу же упростить это выражение, заметив, что выходной сигнал o_n на узле n зависит лишь от связей, которые с ним соединены. Для узла k это означает, что выходной сигнал o_k зависит лишь от весов w_{jk} , поскольку эти веса относятся к связям, ведущим к узлу k .

Это можно рассматривать еще и как то, что выходной сигнал узла k не зависит от весов w_{jb} , где b не равно k , поскольку связь между

этими узлами отсутствует. Вес w_{jb} относится к связи, ведущей к выходному узлу b , но не k .

Это означает, что мы можем удалить из этой суммы все сигналы o_n кроме того, который связан с весом w_{jk} , т.е. o_k . В результате мы полностью избавляемся от суммирования! Отличный трюк, который вам стоит запомнить на будущее.

Если вы уже успели выпить свой кофе, то, возможно, сообразили, что это означает ненужность суммирования по всем выходным узлам для нахождения функции ошибки. Мы уже видели, чем это объясняется: тем, что выходной сигнал узла зависит лишь от ведущих к нему связей и их весовых коэффициентов. Этот момент часто остается нераскрытым во многих учебниках, которые просто приводят выражение для функции без каких-либо пояснений.

Как бы то ни было, теперь мы имеем более простое выражение:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - o_k)^2$$

А сейчас мы используем некоторые средства дифференциального исчисления. Помните, что при необходимости восстановить в памяти необходимые знания вы всегда можете заглянуть в приложение А.

Член t_k — константа и поэтому не изменяется при изменении w_{jk} , т.е. не является функцией w_{jk} . Было бы очень странно, если бы истинные примеры, предоставляющие целевые значения, изменялись в зависимости от весовых коэффициентов! В результате у нас остается член o_k , который, как мы знаем, зависит от w_{jk} , поскольку весовые коэффициенты влияют на распространение в прямом направлении сигналов, которые затем превращаются в выходные сигналы o_k .

Чтобы разбить эту задачу дифференцирования на более простые части, мы воспользуемся цепным правилом (правило дифференцирования сложных функций). Прочитать о нем можно в приложении А.

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{jk}}$$

Теперь мы можем разделаться с каждой из этих частей по отдельности. С первой частью мы справимся легко, поскольку для этого нужно всего лишь взять простую производную от квадратичной функции. В результате получаем:

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) \cdot \frac{\partial o_k}{\partial w_{jk}}$$

Со второй частью придется немножко повозиться, но и это не вызовет больших затруднений. Здесь o_k — это выходной сигнал узла k , который, как вы помните, получается в результате применения сигмоиды к сигналам, поступающим на данный узел. Для большей ясности запишем это в явном виде:

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - o_k) \cdot \frac{\partial \text{сигмоида}(\sum_j w_{jk} \cdot o_j)}{\partial w_{jk}}$$

Здесь o_j — выходной сигнал узла предыдущего скрытого слоя, а не выходной сигнал узла последнего слоя.

Как продифференцировать сигмоиду? Мы могли бы это сделать самостоятельно, проведя сложные и трудоемкие вычисления в соответствии с изложенными в приложении А фундаментальными идеями, однако эта работа уже проделана другими людьми. Поэтому мы просто воспользуемся уже известным ответом, как это ежедневно делают математики по всему миру.

$$\frac{\partial}{\partial x} \text{сигмоида}(x) = \text{сигмоида}(x)(1 - \text{сигмоида}(x))$$

Дифференцирование некоторых функций приводит к выражениям устрашающего вида. В случае же сигмоиды результат получается очень простым. Это одна из причин широкого применения сигмоиды в качестве функции активации в нейронных сетях.

Используя этот впечатляющий результат, получаем следующее выражение:

$$\begin{aligned}\frac{\partial E}{\partial w_{jk}} &= -2(t_k - o_k) \cdot \text{сигмоида}(\sum_j w_{jk} \cdot o_j) (1 - \text{сигмоида}(\sum_j w_{jk} \cdot o_j)) \cdot \frac{\partial}{\partial w_{jk}} (\sum_j w_{jk} \cdot o_j) \\ &= -2(t_k - o_k) \cdot \text{сигмоида}(\sum_j w_{jk} \cdot o_j) (1 - \text{сигмоида}(\sum_j w_{jk} \cdot o_j)) \cdot o_j\end{aligned}$$

А откуда взялся последний сомножитель? Это результат применения цепного правила к производной сигмоиды, поскольку выражение под знаком функции сигмоида () также должно быть проинтегрировано по переменной w_{jk} . Это делается очень просто и дает в результате o_j .

Прежде чем записать окончательный ответ, избавимся от множителя 2 в начале выражения. Мы вправе это сделать, поскольку нас интересует только направление градиента функции ошибки, так что этот множитель можно безболезненно отбросить. Нам совершенно безразлично, какой множитель будет стоять в начале этого выражения, 2, 3 или даже 100, коль скоро мы всегда будем его игнорировать. Поэтому для простоты избавимся от него.

Окончательное выражение, которое мы будем использовать для изменения веса w_{jk} , выглядит так.

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - o_k) \cdot \text{сигмоида}(\sum_j w_{jk} \cdot o_j) (1 - \text{сигмоида}(\sum_j w_{jk} \cdot o_j)) \cdot o_j$$

Ух ты! У нас все получилось!

Это и есть то магическое выражение, которое мы искали. Оно является ключом к тренировке нейронных сетей.

Проанализируем вкратце это выражение, отдельные части которого выделены цветом. Первая часть, с которой вы уже хорошо знакомы, — это ошибка (целевое значение минус фактическое значение).

Сумма, являющаяся аргументом сигмоиды, — это сигнал, поступающий на узел выходного слоя, и для упрощения вида выражения мы могли бы обозначить этот сигнал просто как i_k . Он выступает в качестве входного сигнала узла k до применения функции активации. Последняя часть — это выходной сигнал узла j предыдущего скрытого слоя. Рассмотрение полученного выражения именно в таком ракурсе позволяет лучше понять связь физической картины происходящего с наклоном функции и в конечном счете с уточнением весовых коэффициентов.

Это поистине фантастический результат, и мы можем заслуженно гордиться собой. Путь к этому результату многим людям дается с большим трудом.

Нам осталось сделать совсем немного, и мы достигнем цели. Выражение, с которым мы совладали, предназначено для уточнения весовых коэффициентов связей между скрытым и выходным слоями. Мы должны завершить работу и найти наклон аналогичной функции ошибки для коэффициентов связей между входным и скрытым слоями.

Можно было бы вновь произвести полностью все математические выкладки, но мы не будем этого делать. Мы воспользуемся описанной перед этим физической интерпретацией составляющих выражения для производной и реконструируем его для интересующего нас нового набора коэффициентов. При этом необходимо учесть следующие изменения.

- Первая часть выражения для производной, которая ранее была выходной ошибкой (целевое значение минус фактическое значение), теперь становится рекомбинированной выходной ошибкой скрытых узлов, рассчитываемой в соответствии с механизмом обратного распространения ошибок, с чем вы уже знакомы. Назовем ее e_j .
- Вторая часть выражения, включающая сигмоиду, остается той же, но выражение с суммой, передаваемое функции, теперь относится к предыдущим слоям, и поэтому суммирование осуществляется по всем входным сигналам скрытого слоя, сглаженным весами связей, ведущих к скрытому узлу j . Мы могли бы назвать эту сумму i_j .

- Последняя часть выражения приобретает смысл выходных сигналов o_i узлов первого слоя, и в данном случае эти сигналы являются входными.

Тем самым нам удалось изящно избежать излишних трудоемких вычислений, в полной мере воспользовавшись всеми преимуществами симметрии задачи для конструирования нового выражения. Несмотря на всю ее простоту, это очень мощная методика, взятая на вооружение выдающимися математиками и учеными. Овладев ею, вы, несомненно, произведете на коллег большое впечатление!

Итак, вторая часть окончательного ответа, к получению которого мы стремимся (градиент функции ошибки по весовым коэффициентам связей между входным и скрытым слоями), приобретает следующий вид:

$$\frac{\partial E}{\partial w_{ij}} = - (e_j) \cdot \text{сигмоида} (\sum_i w_{ij} \cdot o_i) (1 - \text{сигмоида} (\sum_i w_{ij} \cdot o_i)) \cdot o_i$$

На данном этапе нами получены все ключевые магические выражения, необходимые для вычисления искомого градиента, который мы используем для обновления весовых коэффициентов по результатам обучения на каждом тренировочном примере, чем мы сейчас и займемся.

Не забывайте о том, что направление изменения коэффициентов противоположно направлению градиента, что неоднократно демонстрировалось на предыдущих диаграммах. Кроме того, мы сглаживаем интересующие нас изменения параметров посредством коэффициента обучения, который можно настраивать с учетом особенностей конкретной задачи. С этим подходом вы также уже сталкивались, когда при разработке линейных классификаторов мы использовали его для уменьшения негативного влияния неудачных примеров на эффективность обучения, а при минимизации функции ошибки — для того, чтобы избежать постоянных перескоков через минимум. Выразим это на языке математики:

$$\text{новый } w_{jk} = \text{старый } w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

Обновленный вес w_{jk} — это старый вес с учетом отрицательной поправки, величина которой пропорциональна производной функции ошибки. Поправка записана со знаком “минус”, поскольку мы хотим, чтобы вес увеличивался при отрицательной производной и уменьшался при положительной, о чем ранее уже говорилось. Символ α (альфа) — это множитель, сглаживающий величину изменений во избежание перескоков через минимум. Этот коэффициент часто называют **коэффициентом обучения**.

Данное выражение применяется к весовым коэффициентам связей не только между скрытым и выходным, но и между входным и скрытым слоями. Эти два случая различаются градиентами функции ошибки, выражения для которых приводились выше.

Прежде чем закончить с этим примером, посмотрим, как будут выглядеть те же вычисления в матричной записи. Для этого сделаем то, что уже делали раньше, — запишем, что собой представляет каждый элемент матрицы изменений весов.

$$\begin{pmatrix} \Delta w_{1,1} & \Delta w_{2,1} & \Delta w_{3,1} & \dots \\ \Delta w_{1,2} & \Delta w_{2,2} & \Delta w_{3,2} & \dots \\ \Delta w_{1,3} & \Delta w_{2,3} & \Delta w_{j,k} & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix} = \begin{pmatrix} E_1 * S_1 (1-S_1) \\ E_2 * S_2 (1-S_2) \\ E_k * S_k (1-S_k) \\ \dots \end{pmatrix} \cdot \begin{pmatrix} O_1 & O_2 & O_j & \dots \end{pmatrix}$$

↑
значения из следующего слоя

↑
значения из предыдущего слоя

Я опустил коэффициент обучения α , поскольку это всего лишь константа, которая никак не влияет на то, как мы организуем матричное умножение.

Матрица изменений весов содержит значения поправок к весовым коэффициентам w_{jk} для связей между узлом j одного слоя и узлом k следующего слоя. Вы видите, что в первой части выражения справа

от знака равенства используются значения из следующего слоя (узел **k**), а во второй — из предыдущего слоя (узел **j**).

Возможно, глядя на приведенную выше формулу, вы заметили, что горизонтальная матрица, представленная одной строкой, — это транспонированная матрица сигналов o_j на выходе предыдущего слоя. Цветовое выделение элементов матриц поможет вам понять, что скалярное произведение матриц отлично работает и в этом случае.

Используя символическую запись матриц, мы можем привести эту формулу к следующему виду, хорошо приспособленному для реализации в программном коде на языке, обеспечивающем эффективную работу с матрицами.

$$\Delta W_{jk} = \alpha \cdot E_k \cdot o_k (1 - o_k) \cdot o_j^T$$

Фактически это выражение совсем не сложное. Сигмоиды исчезли из поля зрения, поскольку они скрыты в матрицах выходных сигналов o_k узлов.

Вот и все! Работа сделана.

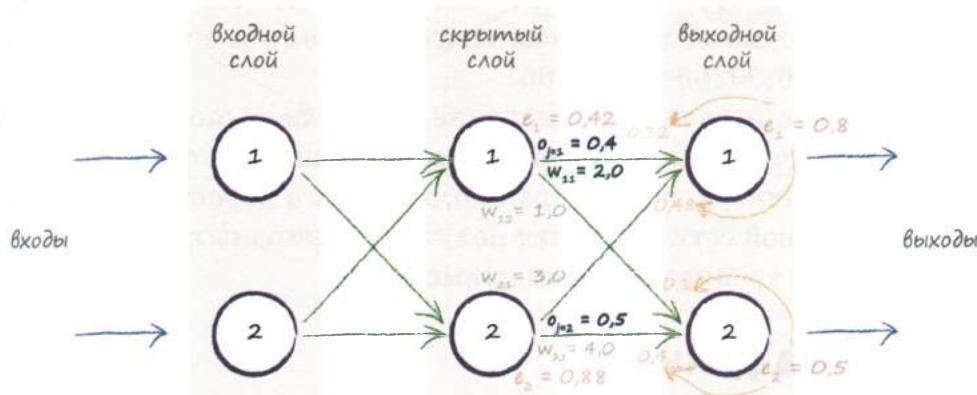
Резюме

- Ошибка нейронной сети является функцией весов внутренних связей.
- Улучшение нейронной сети означает уменьшение этой ошибки посредством изменения указанных весов.
- Непосредственный подбор подходящих весов наталкивается на значительные трудности. Альтернативный подход заключается в итеративном улучшении весовых коэффициентов путем уменьшения функции ошибки небольшими шагами. Каждый шаг совершается в направлении скорейшего спуска из текущей позиции. Этот подход называется **градиентным спуском**.
- Градиент ошибки можно без особых трудностей рассчитать, используя дифференциальное исчисление.

Пример обновления весовых коэффициентов

Проиллюстрируем применение описанного метода обновления весовых коэффициентов на конкретном примере с использованием числовых данных.

С представленной ниже сетью мы уже работали, но в этот раз будут использованы заданные значения выходных сигналов первого и второго узлов скрытого слоя, $o_{j=1}$ и $o_{j=2}$. Эти значения лишь иллюстрируют применение методики и выбраны произвольно, а не вычислены, как это следовало бы сделать, по известным выходным сигналам входного слоя.



Мы хотим обновить весовой коэффициент w_{11} для связи между скрытым и выходным слоями, текущее значение которого равно 2,0.

Запишем еще раз выражение для градиента ошибки.

$$\frac{\partial E}{\partial w_{jk}} = - (t_k - o_k) \cdot \text{сигмоида} (\sum_j w_{jk} \cdot o_j) (1 - \text{сигмоида} (\sum_j w_{jk} \cdot o_j)) \cdot o_j$$

Разберем это выражение по частям.

- Первая часть ($t_k - o_k$) — это уже известная вам по предыдущим диаграммам ошибка $e_1=0.8$.

- Сумма $\sum_j w_{jk} o_j$, передаваемая сигмоидам, равна $(2,0 * 0,4) + (3,0 * 0,5) = 2,3$.
- Тогда сигмоида $1/(1 + e^{-2,3})$ равна 0,909. Следовательно, промежуточное выражение равно $0,909 * (1 - 0,909) = 0,083$.
- Последняя часть — это просто сигнал o_j , которым в данном случае является сигнал o_{j-1} , так как нас интересует вес w_{11} , где $j=1$. Поэтому данная часть просто равна 0,4.

Перемножив все три части этого выражения и не забыв при этом о начальном знаке “минус”, получаем значение $-0,0265$.

При коэффициенте обучения, равном 0,1, изменение веса составит $-0,1 * (-0,0265) = +0,002650$. Следовательно, новое значение w_{11} , определяемое суммой первоначального значения и его изменения, составит $2,0 + 0,00265 = 2,00265$.

Это довольно небольшое изменение, но после выполнения сотен или даже тысяч итераций весовые коэффициенты в конечном счете образуют устойчивую конфигурацию, которая в хорошо натренированной нейронной сети обеспечит получение выходных сигналов, согласующихся с тренировочными примерами.

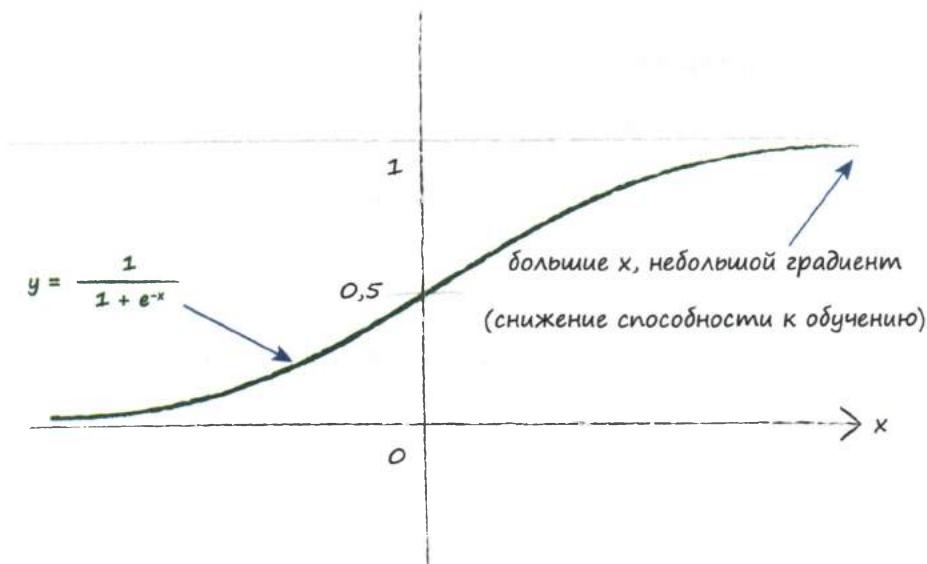
Подготовка данных

В этом разделе мы обсудим, как лучше всего подготовить тренировочные данные и случайные начальные значения весовых коэффициентов и даже спланировать выходные значения таким образом, чтобы процесс обучения имел все шансы на успех.

Совершенно верно, вы все правильно прочитали! Не все попытки использования нейронных сетей заканчиваются успехом, и на то есть множество причин. Некоторые из них можно устраниć за счет продуманного отбора тренировочных данных и начальных значений весовых коэффициентов, а также тщательного планирования схемы выходных сигналов. Рассмотрим все эти факторы поочередно.

Входные значения

Взгляните на приведенный ниже график сигмоиды. Как нетрудно заметить, при больших значениях входного сигнала кривая функции заметно спрямляется.



Значительное спрямление функции активации служит источником проблем, поскольку для усваивания сетью новых значений весов используется градиент. Посмотрите еще раз на выражение для изменений весовых коэффициентов. Оно зависит от градиента функции активации. Использование небольших значений градиента равносильно ограничению способности сети к обучению. Это называется **насыщением** нейронной сети. Отсюда следует, что для входных сигналов лучше задавать небольшие значения.

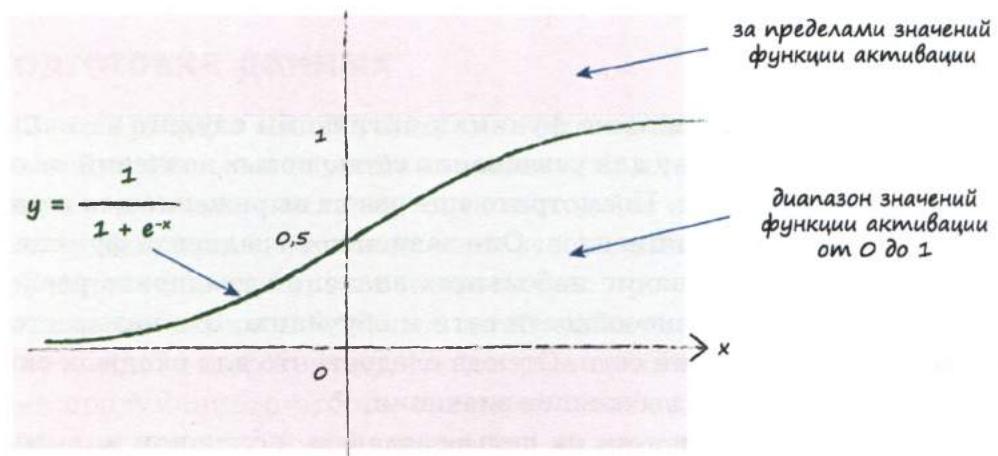
Любопытно, что при этом их нельзя задавать и слишком малыми, поскольку они тоже входят в указанное выражение для изменений весовых коэффициентов. Слишком малые значения входных сигналов могут быть проблематичными еще и потому, что при обработке очень больших или очень малых значений точность компьютерных вычислений значительно снижается.

Неплохим решением этой проблемы является масштабирование входных сигналов до значений в интервале от 0,0 до 1,0. Иногда для входных сигналов вводят небольшое смещение, скажем, 0,01, с целью недопущения нулевых входных сигналов, которые неприятны тем, что при $o_j=0$ выражение для поправки к весам обнуляется, тем самым лишая сеть способности к обучению.

Выходные значения

Выходные значения нейронной сети — это сигналы, появляющиеся на узлах последнего слоя. Если мы используем функцию активации, которая не обеспечивает получение значений свыше 1,0, то было бы глупо пытаться устанавливать значения большей величины в качестве целевых. Вспомните о том, что логистическая функция не дотягивает даже до значения 1,0 — она только приближается к нему. В математике это называется асимптотическим стремлением к 1,0.

Следующий график наглядно демонстрирует, почему логистическая функция активации не может обеспечить получение значений, превышающих 1,0 или меньших нуля.



Если мы все же установим целевые значения в этих недостижимых, запрещенных диапазонах, то тренировка сети приведет к еще большим весовым коэффициентам в попытке добиться все больших и больших значений выходных сигналов, которые фактически никогда не могут быть достигнуты вследствие использования

функции активации. Мы понимаем, что это так же плохо, как и насыщение сети.

Поэтому мы должны масштабировать наши целевые выходные значения таким образом, чтобы они были допустимыми при данной функции активации, одновременно заботясь о том, чтобы избежать значений, которые в действительности никогда не могут быть достигнуты.

Общепринято использовать диапазон значений от 0,0 до 1,0, но некоторые разработчики используют диапазон от 0,01 до 0,99, поскольку значения 0,0 и 1,0, с одной стороны, не являются подходящими целевыми значениями, а с другой — могут приводить к чрезмерно большим значениям весовых коэффициентов.

Случайные начальные значения весовых коэффициентов

Здесь применима та же аргументация, что и в случае входных и выходных сигналов. Мы должны избегать больших начальных значений весовых коэффициентов, поскольку использование функции активации в этой области значений может приводить к насыщению сети, о котором мы только что говорили, и снижению способности сети обучаться на лучших значениях.

Один из возможных вариантов — прибегнуть к выбору значений из случайного равномерного распределения чисел в диапазоне от -1,0 до +1,0. Это гораздо лучше, чем использовать, скажем, диапазон чисел от -1000 до +1000.

Возможен ли лучший вариант? Вполне вероятно.

Математики и ученые-компьютерщики разработали подходы, позволяющие определять эмпирические правила для задания случайных начальных значений весовых коэффициентов в зависимости от конкретной конфигурации сети и используемой функции активации. Соответствующие рецепты в высшей степени специфичны, но, невзирая на это, мы рискнем подступиться к ним!

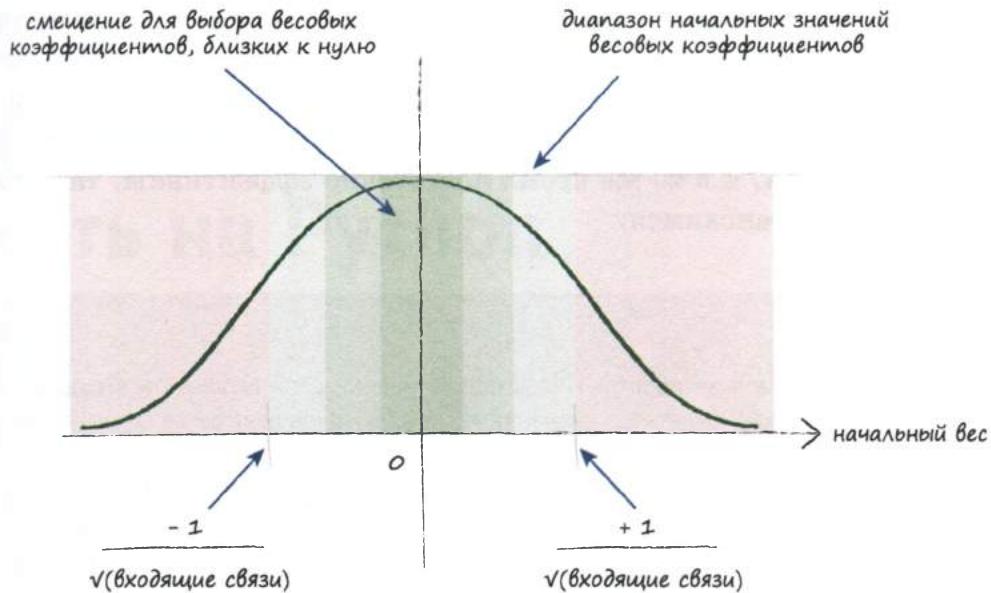
Мы не будем вдаваться в детали математических выкладок, но центральная идея заключается в том, что если на узел нейронной сети поступает множество сигналов, причем поведение этих сигналов хорошо определено, они не достигают слишком больших значений и не распределены каким-то невероятным образом, то весовые коэффициенты не должны нарушать такое состояние

сигналов в процессе их объединения и обработки функцией активации. Иными словами, мы не должны использовать весовые коэффициенты, разрушающие результаты наших попыток тщательно масштабировать входные сигналы. Суть эмпирического правила, к которому пришли математики, заключается в том, что весовые коэффициенты инициализируются числами, случайно выбираемыми из диапазона, грубая оценка которого определяется обратной величиной квадратного корня из количества связей, ведущих к узлу. Таким образом, если к каждому узлу ведут три связи, то начальные значения весов не должны превышать значение $1/(\sqrt{3}) = 0,577$. Если же каждый узел имеет 100 входящих связей, то веса должны находиться в диапазоне, ограниченном значением $1/(\sqrt{100}) = 0,1$.

Интуитивно это понятно. Некоторые слишком большие веса сметили бы функцию активации в область больших значений, что привело бы к ее **насыщению**. И чем больше связей приходится на узел, тем больше складывается весовых коэффициентов. Поэтому эмпирическое правило, которое уменьшает диапазон значений весовых коэффициентов с увеличением количества связей на узел, находит логическое объяснение.

Если вы уже знакомы с идеей выборочных значений, извлекаемых из распределений вероятностей, то поймете, что это правило фактически относится к нормальному распределению, для которого среднее значение равно нулю, а стандартное отклонение равно обратной величине корня из количества связей, ведущих к узлу. Однако не будем слишком строго придерживаться этой рекомендации, поскольку она предполагает выполнение довольно большого количества условий, которые не всегда соблюдаются, таких как альтернативное использование гиперболического тангенса `th()` в качестве функции активации и специфическое распределение входных сигналов.

Следующий график иллюстрирует в наглядной форме как простой подход, так и более сложный, в котором используется нормальное распределение.



В любом случае никогда не задавайте для начальных весов равные значения, особенно нулевые. Это был бы крайне неудачный вариант!

Этот вариант был бы неудачным по той причине, что в таком случае все узлы сети получили бы одинаковые сигналы, и сигналы на выходе каждого узла были бы одинаковыми. Если затем приступить к обновлению весов с использованием механизма обратного распространения ошибки, то ошибка распределится равномерно. Вы ведь не забыли, что ошибка распределяется между узлами пропорционально весам. Это приведет к одинаковым поправкам для всех весовых коэффициентов, что, в свою очередь, вновь приведет к весам, имеющим одинаковые значения. Подобная симметрия играет крайне отрицательную роль, ведь если правильно натренированная сеть должна иметь неодинаковые значения весовых коэффициентов (что характерно для большинства задач), то вы никогда не достигнете этого состояния.

Еще худший выбор — нулевые значения, поскольку они полностью “убивают” входной сигнал. В этом случае функция обновления весов, которая зависит от входных сигналов, обнуляется, тем самым полностью исключая саму возможность обновления весов.

Существует множество других мер, которые можно предпринять для улучшения процедур подготовки входных данных, задания весовых коэффициентов и организации желаемых выходных значений. С учетом целей книги изложенные идеи достаточно просты, чтобы быть понятными, и в то же время достаточно эффективны, так что на этом мы и остановимся.

Резюме

- Нейронные сети не работают удовлетворительно, если входные и выходные данные, а также начальные значения весовых коэффициентов не согласуются со структурой сети и спецификой конкретной задачи.
- Распространенной проблемой является **насыщение** сети — ситуация, когда большие значения сигналов, часто обусловленные большими значениями весовых коэффициентов, приводят к сигналам, попадающим в область близких к нулю значений градиента функции активации. Результатом этого является снижение способности сети обучаться на лучших значениях весовых коэффициентов.
- Другую проблему представляют **нулевые** значения сигналов или весов. Эти значения также полностью лишают сеть возможности обучаться на лучших значениях весовых коэффициентов.
- Значения весовых коэффициентов внутренних связей должны быть **случайными** и **небольшими**, но не нулевыми. Иногда используют более сложные правила, включающие, например, уменьшение значений весовых коэффициентов с увеличением количества связей, ведущих к узлу.
- **Входные сигналы** должны масштабироваться до небольших, но ненулевых значений. Обычно используют диапазоны значений от 0,01 до 0,99 и от -1,0 до +1,0 в зависимости от того, какой из них лучше соответствует специфике задачи.
- **Выходные сигналы** должны находиться в пределах диапазона, который способна обеспечить функция активации. Значения, меньшие или равные 0 и большие или равные 1, не совместимы с логистической сигмоидой. Установка тренировочных целевых значений за пределами допустимого диапазона приведет к еще большим значениям весов и в конечном счете к насыщению сети. Неплохим диапазоном является диапазон значений от 0,01 до 0,99.

ПРИЛОЖЕНИЕ А

Краткое введение в дифференциальное исчисление

Представьте, что вы движетесь в автомобиле с постоянной скоростью 30 миль в час. Затем вы нажимаете на педаль акселератора. Если вы будете держать ее постоянно нажатой, скорость движения постепенно увеличится до 35, 40, 50, 60 миль в час и т.д.

Скорость движения автомобиля изменяется.

В этом разделе мы исследуем природу изменения различных величин и обсудим способы математического описания этих изменений. А что подразумевается под “математическим описанием”? Это означает установление соотношений между различными величинами, что позволит нам точно определить степень изменения одной величины при изменении другой. Например, речь может идти об изменении скорости автомобиля с течением времени, отслеживаемого по наручным часам. Другими возможными примерами могут служить зависимость высоты растений от уровня выпавших осадков или изменение длины пружины в зависимости от приложенного к ее концам усилия.

Математики называют это **дифференциальным исчислением**. Я долго сомневался, прежде чем решился вынести этот термин в название приложения. Мне казалось, что многих людей он отпугнет, поскольку они посчитают изложенный ниже материал слишком сложным для того, чтобы на него тратить время. Однако такое отношение к данному предмету обсуждения могли привить только плохие учителя или плохие школьные учебники.

Прочитав все приложение до конца, вы убедитесь в том, что математическое описание изменений — а именно для этого и предназначено дифференциальное исчисление — оказывается совсем не сложным во многих полезных сценариях.

Даже если вы уже знакомы с дифференциальным исчислением, возможно, еще со школы, вам все равно стоит прочитать это приложение, поскольку вы узнаете много интересного об истории математики. Вам будет полезно взять на вооружение идеи и инструменты математического анализа, чтобы использовать их в будущем при решении различных задач.

Если вам нравятся исторические эссе, почитайте книгу “Великие противостояния в науке” (ИД “Вильямс”, 2007), где рассказывается о драме, разыгравшейся между Лейбницем и Ньютона, каждый из которых приписывал открытие дифференциального исчисления себе!



Готфрид Лейбниц



Сэр Исаак Ньютона

Прямая линия

Чтобы настроиться на работу, начнем для разминки с очень простого сценария.

Вновь представьте себе автомобиль, движущийся с постоянной скоростью 30 миль в час. Не больше и не меньше, а ровно 30 миль в час.

В приведенной ниже таблице указана скорость автомобиля в различные моменты времени с интервалом в полминуты.

Время (мин.)	Скорость (миль/ч)
0,0	30
0,5	30
1,0	30
1,5	30
2,0	30
2,5	30
3,0	30

Следующий график представляет значения скорости в соответствующие моменты времени.



Как видите, скорость не изменяется с течением времени, и поэтому точки выстраиваются в прямую линию. Эта линия не идет ни вверх (увеличение скорости), ни вниз (уменьшение скорости), а остается на уровне 30 миль в час.

В данном случае математическое выражение для скорости, которую мы обозначим как s , имеет следующий вид:

$$s = 30$$

Если бы кто-то спросил нас о том, как изменяется скорость со временем, мы бы ответили, что она не изменяется. Скорость ее изменения равна нулю. Иными словами, скорость не зависит от времени. Зависимость в данном случае нулевая.

Мы только что выполнили одну из операций дифференциального исчисления! Я не шучу!

Дифференциальное исчисление сводится к нахождению изменения одной величины в результате изменения другой. В данном случае нас интересует, как скорость изменяется со временем.

Вышеизложенное можно записать в следующей математической форме:

$$\frac{\delta s}{\delta t} = 0$$

Что собой представляют эти символы? Считайте, что они означают “как скорость изменяется с изменением времени”, или “как s зависит от t ”.

Таким образом, это выражение является компактной математической записью утверждения о том, что скорость не изменяется со временем. Другими словами, изменение времени не влияет на скорость. Зависимость скорости от времени нулевая. Именно это означает нуль в выражении. Обе величины полностью независимы. Ладно, ладно — мы все уже поняли!

В действительности эту независимость можно легко заметить, если вновь взглянуть на выражение для скорости $s = 30$. В нем вообще нет даже намека на время, т.е. в нем отсутствует любое проявление символа t . Поэтому, чтобы сказать, что $\partial s / \partial t = 0$, нам не потребуется никакое дифференциальное исчисление. Говоря языком математиков, “это следует из самого вида выражения”.

Выражения вида $\partial s / \partial t$, определяющие степень изменения одной величины при изменении другой, называют производными. Для наших целей знание этого термина не является обязательным, но он вам может встретиться где-нибудь еще.

А теперь посмотрим, что произойдет, если надавить на педаль акселератора. Поехали!

Наклонная прямая линия

Представьте себе все тот же автомобиль, движущийся со скоростью 30 миль в час. Вы надавили на педаль акселератора, и автомобиль набирает скорость. Вы удерживаете педаль нажатой, смотрите на показания спидометра и записываете их через каждые 30 секунд.

По прошествии 30 секунд скорость автомобиля составила 35 миль в час. Через минуту она возрастает до 40 миль в час. Через 90 секунд автомобиль ускоряется до 45 миль в час, а после двух минут автомобиль разгоняется до 50 миль в час. С каждой минутой скорость автомобиля увеличивается на 10 миль в час.

Эта информация сведена в представленной ниже таблице.

Время (мин.)	Скорость (миль/ч)
0,0	30
0,5	35
1,0	40
1,5	45
2,0	50
2,5	55
3,0	60

Визуализируем эти данные.



Как видите, увеличение скорости движения автомобиля с 30 до 60 миль в час происходит с **постоянной скоростью изменения**. Она действительно постоянна, поскольку приращение скорости за каждые полминуты остается одним и тем же, что приводит к **прямолинейному** графику скорости.

А что собой представляет выражение для скорости? В нулевой момент времени скорость равна 30 миль в час. Далее мы добавляем по 10 миль в час за каждую минуту. Таким образом, искомое выражение должно иметь следующий вид:

$$\text{скорость} = 30 + (10 * \text{время})$$

Перепишем его, используя символические обозначения:

$$s = 30 + 10t$$

В этом выражении имеется константа 30. В нем также есть член **(10 * время)**, который каждую минуту увеличивает скорость на 10 миль в час. Вы быстро сообразите, что 10 — это **угловой коэффициент линии**, график которой мы построили. Вспомните, что общая форма уравнения прямой линии имеет вид $y = ax + b$, где a — угловой коэффициент, или **наклон**, линии.

А как будет выглядеть выражение, описывающее изменение скорости во времени? Мы уже говорили, что с каждой минутой скорость увеличивается на 10 миль в час.

$$\frac{\delta s}{\delta t} = 10$$

Это выражение говорит о том, что между скоростью движения автомобиля и временем существует зависимость. На это указывает тот факт, что $\partial s / \partial t$ не равно нулю.

Вспомните, что для прямой линии, описываемой уравнением $y = ax + b$, наклон равен a , и поэтому наклон для прямой линии $s = 30 + 10t$ должен быть равным 10.

Отличная работа! Мы уже успели охватить довольно много базовых элементов дифференциального исчисления, и это оказалось совсем несложно.

А теперь надавим на акселератор еще сильнее!

Кривая линия

Представьте, что я начинаю поездку в автомобиле, нажав педаль акселератора почти до упора и удерживая ее в этом положении. Совершенно очевидно, что начальная скорость равна нулю, поскольку в первый момент автомобиль вообще не двигался.

Поскольку педаль акселератора нажата почти до упора, скорость движения будет увеличиваться не равномерно, а быстрее. Это означает, что скорость автомобиля уже не будет возрастать только на 10 миль в час каждую минуту. Само ежеминутное приращение скорости будет с каждой минутой увеличиваться, если педаль акселератора по-прежнему удерживается нажатой.

Предположим, что скорость автомобиля принимает в каждую минуту значения, приведенные в следующей таблице.

Время (мин.)	Скорость (миль/ч)
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64

Внимательно присмотревшись к этим данным, вы заметите, что выбранные мною значения скорости представляют собой время в минутах, возведенное в квадрат, т.е. скорость в момент времени 2 равна $2^2=4$, в момент времени 3 – $3^2=9$, в момент времени 4 – $4^2=16$ и т.д.

Записать соответствующее математическое выражение не составляет труда:

$$s = t^2$$

Да, я отдаю себе отчет в том, что пример со скоростью автомобиля довольно искусственный, но он позволяет наглядно продемонстрировать, как мы можем использовать дифференциальное исчисление.

Представим табличные данные в виде графика, который позволит нам лучше понять характер изменения скорости автомобиля во времени.



Как видите, со временем скорость автомобиля растет все интенсивнее. График уже не является прямой линией. Нетрудно сделать вывод, что вскоре скорость должна достигнуть очень больших значений. На 20-й минуте она должна была бы составить 400 миль в час, а на 100-й — целых 10000 миль в час!

Возникает интересный вопрос: как быстро изменяется скорость с течением времени? Другими словами, каково приращение скорости в каждый момент времени?

Это не то же самое, что спросить: а какова фактическая скорость в каждый момент времени? Ответ на этот вопрос нам уже известен, поскольку для него у нас есть соответствующее выражение: $s = t^2$.

Мы же спрашиваем следующее: какова **скорость изменения скорости** автомобиля в каждый момент времени? Но что вообще это означает в нашем примере, в котором график оказался криволинейным?

Если вновь обратиться к двум предыдущим примерам, то в них скорость изменения скорости определялась наклоном графика зависимости скорости от времени. Когда автомобиль двигался с постоянной скоростью 30 миль в час, его скорость не изменялась, и поэтому скорость ее изменения была равна 0. Когда автомобиль равномерно набирал скорость, скорость ее изменения составляла 10 миль в час за минуту. Данный показатель имел одно и то же значение в любой момент времени. Он был равен 10 миль в час на второй, четвертой и даже на сотой минуте.

Можем ли мы применить те же рассуждения к криволинейному графику? Можем, но с этого момента нам следует немного сбавить темп, чтобы не спеша обсудить этот вопрос.

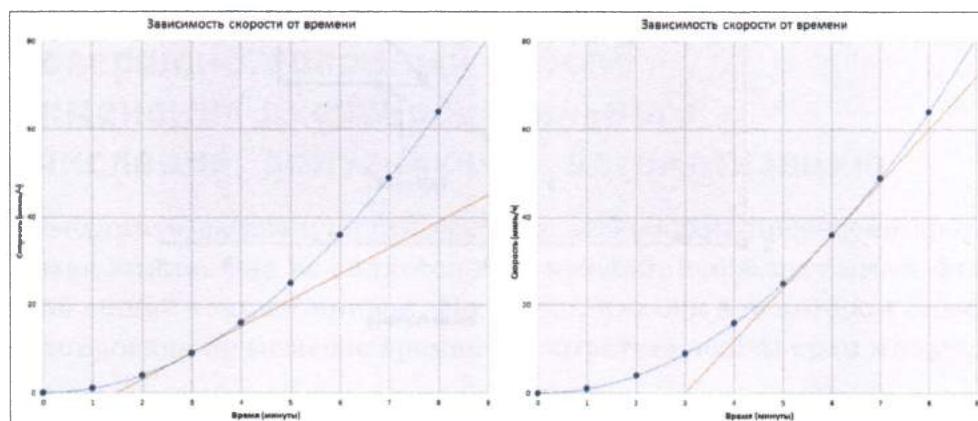
Применение дифференциального исчисления вручную

Присмотримся повнимательнее к тому, что происходит в конце третьей минуты движения.

По прошествии трех минут с момента начала движения ($t=3$) скорость (s) составит 9 миль в час. Сравним это с тем, что будет в конце шестой минуты движения. В этот момент скорость составит 36 миль в час, но мы знаем, что после этого автомобиль будет двигаться еще быстрее.

Мы также знаем, что в любой момент времени вслед за шестой минутой скорость будет увеличиваться быстрее, чем в эквивалентный момент времени, следующий за третьей минутой. Существует реальное различие в том, что происходит в моменты времени, соответствующие трем и шести минутам движения.

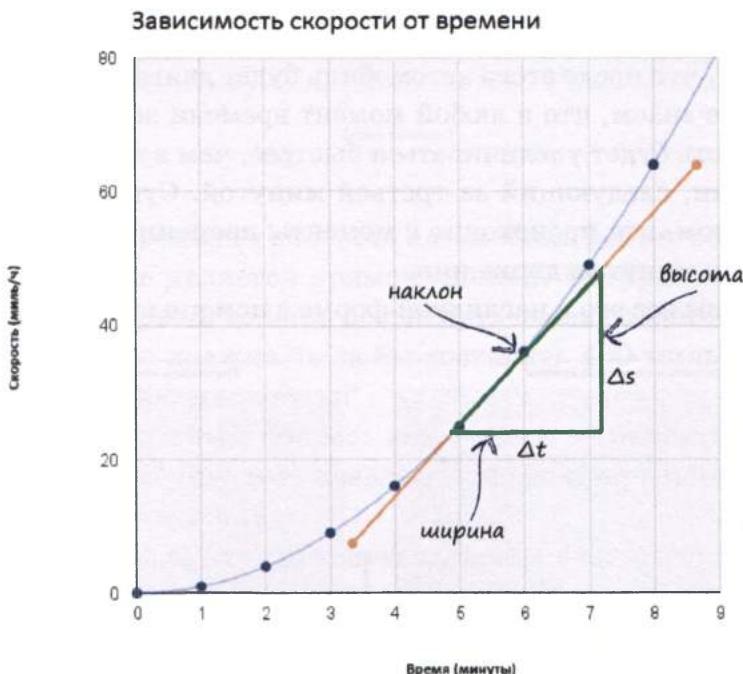
Представим все это в наглядной форме с помощью графика.



Вы видите, что в момент времени 6 минут наклон кривой круче, чем в момент времени 3 минуты. Оба наклона представляют иско-мую скорость изменения скорости движения. Очень важно, чтобы вы это поняли, потому повторим мысль в следующей формулировке: скорость изменения кривой в любой точке определяется ее наклоном в этой точке.

Но как измерить наклон линии, которая искривлена? С прямыми линиями все просто, а вот как быть с кривыми? Мы можем попытаться оценить наклон, прочертив прямую линию, так называемую касательную, которая лишь касается кривой (имеет с ней только одну общую точку) таким образом, чтобы иметь тот же наклон, что и кривая в данной точке. Именно так в действительности люди и поступали, пока не были изобретены другие способы.

Давайте испытаем этот приближенный простой способ хотя бы для того, чтобы лучше понять, к чему он приводит. На следующей иллюстрации представлен график скорости с касательной к кривой в точке, соответствующей шести минутам движения.



Из школьного курса математики нам известно, что для определения наклона, или углового коэффициента, следует разделить приращение вертикальной координаты на приращение горизонтальной координаты. На диаграмме приращение по вертикали (скорость) обозначено как Δs , а приращение по горизонтали (время) — как Δt . Символ Δ (читается “дельта”) просто означает небольшое изменение. Поэтому Δt — это небольшое изменение t .

Наклон определяется отношением $\Delta s / \Delta t$. Мы можем выбрать для определения наклона любой треугольник и измерить длину его катетов с помощью линейки. В выбранном мною треугольнике приращение Δs оказалось равным 9,6, а приращение Δt — 0,8. Это дает следующую величину наклона.

скорость изменения = наклон в данной точке

$$\begin{aligned} &= \frac{\Delta s}{\Delta t} \\ &= 9,6 / 0,8 \\ &= 12,0 \end{aligned}$$

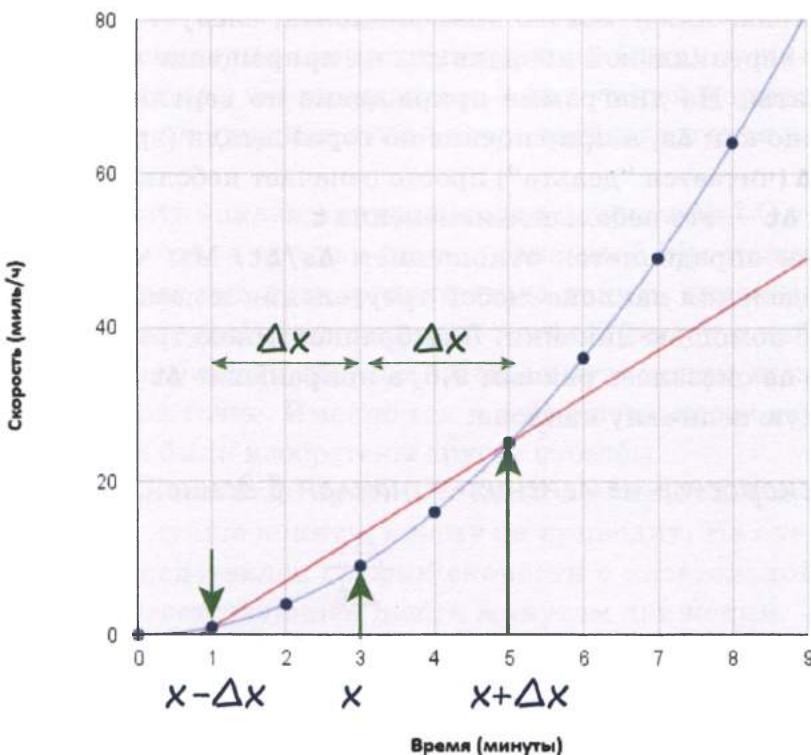
Мы получили очень важный результат! Скорость изменения скорости в момент времени 6 минут составляет 12,0 миль в час за минуту.

Нетрудно заметить, что от способа, основанного на проведении касательной вручную и выполнении измерений с помощью линейки, вряд ли можно ожидать высокой точности. Поэтому мы должны обратиться к более совершенным методам.

Усовершенствованный способ применения дифференциального исчисления, допускающий автоматизацию

Взгляните на следующий график, на котором проведена другая прямая линия. Она не является касательной, поскольку имеет более одной общей точки с кривой. Но видно, что она в некотором смысле центрирована на моменте времени, соответствующем трем минутам.

Зависимость скорости от времени



Связь этой прямой с моментом времени 3 минуты действительно существует. Для ее проведения были выбраны моменты времени до и после интересующего нас момента времени $t=3$. В данном случае были выбраны точки, отстоящие от точки $t=3$ на две минуты в большую и меньшую стороны, т.е. этим моментам времени соответствуют точки $t=1$ и $t=5$.

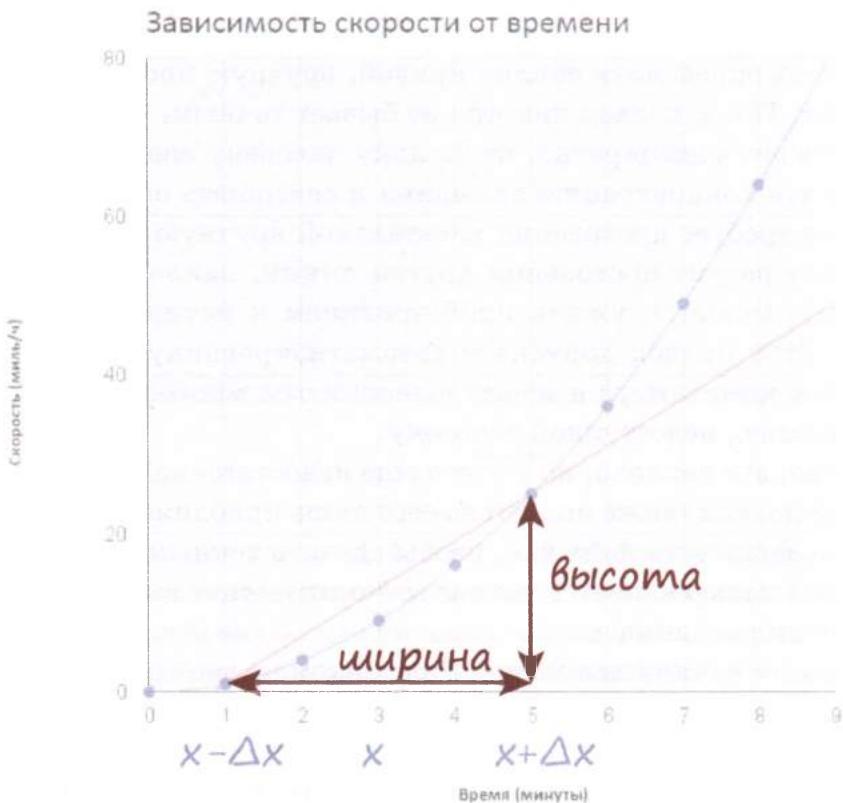
Используя математические обозначения, можно сказать, что Δx равно 2 минуты. Тогда выбранным нам точкам соответствуют координаты $x - \Delta x$ и $x + \Delta x$. Вспомните, что символ Δ означает “небольшое изменение”, поэтому Δx — это небольшое изменение x .

Зачем мы это делаем? Потерпите минутку — очень скоро все прояснится.

Если мы возьмем значения скорости в моменты времени $x - \Delta x$ и $x + \Delta x$ и проведем через соответствующие две точки прямую линию, то ее наклон будет примерно равен наклону касательной в средней точке x .

Вернитесь к предыдущей иллюстрации и взгляните на эту прямую линию. Несомненно, ее наклон не совпадает в точности с наклоном истинной касательной в точке x , но мы исправим этот недостаток.

Вычислим наклон этой прямой. Мы используем прежний подход и рассчитаем наклон как отношение смещения точки по вертикали к смещению по горизонтали. Следующая иллюстрация проясняет, что в данном случае представляют собой эти смещения.



Смещение по вертикали — это разность между значениями скорости в точках $x + \Delta x$ и $x - \Delta x$, соответствующих пяти минутам и одной минуте движения. Эти скорости нам известны: $5^2 = 25$ и $1^2 = 1$, поэтому разность составляет 24. Смещение по горизонтали — это просто расстояние между точками $x + \Delta x$ и $x - \Delta x$, т.е. $5 - 1 = 4$. Следовательно, имеем:

$$\text{наклон} = \frac{\text{высота}}{\text{ширина}}$$

$$= 24 / 4$$

$$= 6$$

Таким образом, наклон прямой линии, являющейся приближением к касательной в точке $t=3$ минуты, составляет 6 миль в час за минуту.

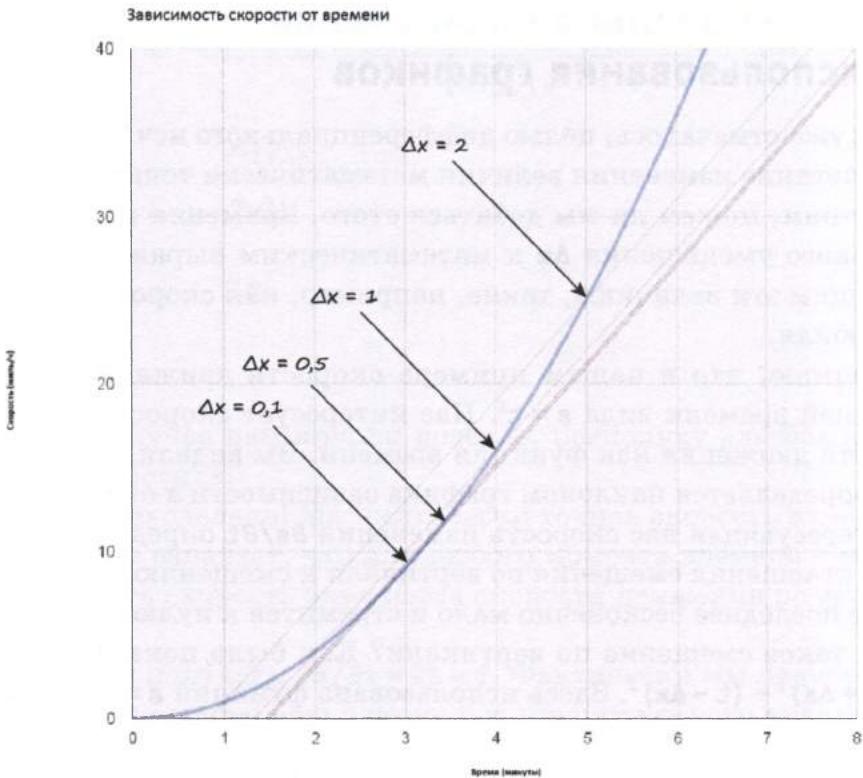
Сделаем паузу и осмыслим полученные результаты. Сначала мы попытались определить наклон кривой, вручную проведя касательную к ней. Такой подход никогда не бывает точным, и мы не можем повторять его многократно, поскольку человеку свойственно уставать, терять концентрацию внимания и совершать ошибки. Второй подход не требует проведения касательной вручную и вместо этого предлагает рецепт построения другой линии, наклон которой, по-видимому, может служить приближением к истинному наклону кривой. Этот подход допускает автоматизированную реализацию с помощью компьютера и может выполняться многократно с огромной скоростью, недоступной человеку.

Конечно, это неплохо, но и этого еще недостаточно!

Второй подход также является всего лишь приближенным. Можно ли его усовершенствовать так, чтобы сделать точным? В конце концов, нашей целью является точное математическое описание скорости изменения величин.

Вот здесь и начинается магия! Я познакомлю вас с одним из самых элегантных инструментов, разработанных математиками.

Что случится, если мы уменьшим величину смещения. Иными словами, что произойдет, если уменьшить Δx ? Следующая иллюстрация демонстрирует несколько приближений в виде наклонных прямых, соответствующих уменьшению Δx .



Мы провели линии для $\Delta x=2$, $\Delta x=1$, $\Delta x=0,5$ и $\Delta x=0,1$. Вы видите, что эти линии постепенно приближаются к интересующей нас точке $x=3$. Нетрудно сообразить, что по мере уменьшения Δx прямые линии будут все более и более приближаться к истинной касательной.

При бесконечно малой величине Δx линия приблизится к истинной касательной на бесконечно малое расстояние. Это очень круто!

Идея постепенного улучшения первоначального приближенного решения путем уменьшения отклонений необычайно плодотворна. Она позволяет математикам решать задачи, непосредственное решение которых наталкивается на значительные трудности. При таком подходе к решению приближаются постепенно, словно крадучись, вместо того чтобы атаковать его прямо в лоб!

Дифференциальное исчисление без использования графиков

Как уже отмечалось, целью дифференциального исчисления является описание изменения величин математически точным способом. Посмотрим, можем ли мы добиться этого, применяя идею последовательного уменьшения Δx к математическим выражениям, определяющим эти величины, такие, например, как скорость движения автомобиля.

Напомню, что в нашем примере скорость движения является функцией времени вида $s = t^2$. Нас интересует скорость изменения скорости движения как функция времени. Вы видели, что эта величина определяется наклоном графика зависимости s от t .

Интересующая нас скорость изменения $\frac{\partial s}{\partial t}$ определяется величиной отношения смещения по вертикали к смещению по горизонтали, где последнее бесконечно мало и стремится к нулю.

Что такое смещение по вертикали? Как было показано раньше, это $(t + \Delta x)^2 - (t - \Delta x)^2$. Здесь использована функция $s = t^2$, где t принимает значения, немного меньшие и немного большие, чем в интересующей нас точке. Это “немного” равно Δx .

Что такое смещение по горизонтали? Как вы видели раньше, это просто расстояние между точками $(t + \Delta x)$ и $(t - \Delta x)$, которое равно $2\Delta x$.

Мы уже почти у цели:

$$\begin{aligned}\frac{\delta s}{\delta t} &= \frac{\text{высота}}{\text{ширина}} \\ &= \frac{(t + \Delta x)^2 - (t - \Delta x)^2}{2\Delta x}\end{aligned}$$

Раскроем и упростим это выражение:

$$\begin{aligned}\frac{\delta s}{\delta t} &= \frac{t^2 + \Delta x^2 + 2t\Delta x - t^2 - \Delta x^2 + 2t\Delta x}{2\Delta x} \\ &= \frac{4t\Delta x}{2\Delta x} \\ \frac{\delta s}{\delta t} &= 2t\end{aligned}$$

В данном случае нам крупно повезло, поскольку алгебра необычайно все упростила.

Итак, мы это сделали! Математически точная скорость изменения $\frac{\partial s}{\partial t} = 2t$. Это означает, что для любого момента времени t мы можем вычислить скорость изменения скорости движения по формуле $\frac{\partial s}{\partial t} = 2t$.

При $t=3$ мы получаем $\frac{\partial s}{\partial t} = 2t = 6$. Фактически мы подтвердили этот результат еще раньше с помощью приближенного метода. Для $t=6$ получаем $\frac{\partial s}{\partial t} = 2t = 12$, что также согласуется с найденным ранее результатом.

А что насчет ста минут? В этом случае $\frac{\partial s}{\partial t} = 2t = 200$ миль в час за минуту. Это означает, что спустя сто минут от начала движения машина будет ускоряться со скоростью 200 миль в час за минуту.

Сделаем небольшую паузу и немного поразмышляем над величием и красотой того, что нам удалось сделать. Мы получили математическое выражение, с помощью которого можем точно определить скорость изменения скорости движения автомобиля в любой момент времени. При этом, в полном соответствии с нашими рассуждениями в начале приложения, мы видим, что эти изменения s действительно зависят от времени.

Нам повезло, что алгебра все упростила, но в силу простоты выражения $s = t^2$ нам не представилась возможность проверить на практике идею устремления Δx к нулю. В связи с этим полезно рассмотреть другой пример, в котором скорость движения автомобиля описывается несколько более сложной формулой:

$$s = t^2 + 2t$$

$$\frac{\delta s}{\delta t} = \frac{\text{высота}}{\text{ширина}}$$

А что теперь означает вертикальное смещение? Это разность между скоростью s , рассчитанной в момент времени $t + \Delta x$, и скоростью s , рассчитанной в момент времени $t - \Delta x$. Найдем эту разность путем несложных вычислений: $(t + \Delta x)^2 + 2(t + \Delta x) - (t - \Delta x)^2 - 2(t - \Delta x)$.

А что насчет горизонтального смещения? Это просто расстояние между точками $(t + \Delta x)$ и $(t - \Delta x)$, которое по-прежнему равно $2\Delta x$:

$$\frac{\delta s}{\delta t} = \frac{(t + \Delta x)^2 + 2(t + \Delta x) - (t - \Delta x)^2 - 2(t - \Delta x)}{2\Delta x}$$

Раскроем и упростим это выражение:

$$\frac{\delta s}{\delta t} = \frac{t^2 + \Delta x^2 + 2t\Delta x + 2t + 2\Delta x - t^2 - \Delta x^2 + 2t\Delta x - 2t + 2\Delta x}{2\Delta x}$$

$$= \frac{4t\Delta x + 4\Delta x}{2\Delta x}$$

$$\frac{\delta s}{\delta t} = 2t + 2$$

Это замечательный результат! К сожалению, алгебра и в этот раз немного перестаралась, упростив нам работу. Но этот пример не был напрасным, поскольку здесь уже вырисовывается закономерность, к которой мы еще вернемся.

Попробуем рассмотреть еще один, чуть более сложный, пример. Предположим, что скорость движения автомобиля описывается кубической функцией времени:

$$s = t^3$$

$$\frac{\delta s}{\delta t} = \frac{\text{высота}}{\text{ширина}}$$

$$\frac{\delta s}{\delta t} = \frac{(t + \Delta x)^3 - (t - \Delta x)^3}{2\Delta x}$$

Раскроем и упростим это выражение:

$$\frac{\delta s}{\delta t} = \frac{t^3 + 3t^2\Delta x + 3t\Delta x^2 + \Delta x^3 - t^3 + 3t^2\Delta x - 3t\Delta x^2 + \Delta x^3}{2\Delta x}$$

$$= \frac{6t^2\Delta x + 2\Delta x^3}{2\Delta x}$$

$$\frac{\delta s}{\delta t} = 3t^2 + \Delta x^2$$

Это уже намного интереснее! Мы получили результат, содержащий Δx , тогда как раньше эти члены взаимно сокращались.

Вспомните, что корректное значение наклона получается лишь в том случае, если величина Δx уменьшается, становясь бесконечно малой.

А сейчас смотрите! Что произойдет с величиной Δx в выражении $\frac{\delta s}{\delta t} = 3t^2 + \Delta x^2$, если она становится все меньшей и меньшей? Она исчезнет! Если для вас это неожиданность, то представьте, что величина Δx очень мала. Напрягитесь и представьте, что она еще меньше. Потом представьте, что на самом деле она еще меньше... И этот процесс мысленного уменьшения Δx вы могли бы продолжать до бесконечности, устремляя ее к нулю. Поэтому давайте проявим решимость и сразу же перейдем к нулю без лишней сути.

В результате мы получаем искомый математически точный ответ:

$$\frac{\delta s}{\delta t} = 3t^2$$

Это фантастический результат, и на этот раз он был получен с использованием мощного математического инструмента, что оказалось совсем несложным.

Закономерности

Конечно, это весьма интересное занятие — находить производные, используя приращения наподобие Δx , и смотреть, что произойдет, если делать их все меньшими и меньшими. Но зачастую можно получить результат, не выполняя всю эту работу.

Посмотрите на приведенные ниже формулы и постарайтесь увидеть в них закономерность.

$$s = t^2 \quad \xrightarrow{\hspace{1cm}} \quad \frac{\delta s}{\delta t} = 2t$$

$$s = t^2 + 2t \quad \xrightarrow{\hspace{1cm}} \quad \frac{\delta s}{\delta t} = 2t + 2$$

$$s = t^3 \quad \xrightarrow{\hspace{1cm}} \quad \frac{\delta s}{\delta t} = 3t^2$$

Вы видите, что производная функции t представляет собой ту же функцию, но с понижением на единицу каждой степени t . Поэтому t^4 превращается в t^3 , а t^7 превратилось бы в t^6 и т.д. Это очень просто! А если вы вспомните, что t — это t^1 , то в производной это превращается в t^0 , т.е. в 1.

Свободные постоянные члены, такие как 3, 4 или 5, просто исчезают. Постоянные переменные, не являющиеся коэффициентами, такие как a , b или c , также исчезают, поскольку скорость их изменения также нулевая. Именно поэтому их называют константами.

Но погодите, ведь t^2 превращается в $2t$, а не просто в t , а t^3 превращается в $3t^2$, а не просто в t^2 . Это общее правило: степень переменной, прежде чем уменьшиться на единицу, становится коэффициентом.

Поэтому 5 в $2t^5$ используется в качестве дополнительного коэффициента перед уменьшением степени на единицу: $5 \cdot 2t^4 = 10t^4$.

Приведенная ниже формула суммирует все, что было сказано о дифференцировании степеней, в виде следующего правила:

$$y = ax^n \quad \xrightarrow{\hspace{10em}} \quad \frac{\delta y}{\delta n} = nax^{n-1}$$

Испытаем эту формулу на дополнительных примерах только ради того, чтобы набить руку в использовании этого нового приема:

$$s = t^5 \quad \xrightarrow{\hspace{10em}} \quad \frac{\delta s}{\delta t} = 5t^4$$

$$s = 6t^6 + 9t + 4 \quad \xrightarrow{\hspace{10em}} \quad \frac{\delta s}{\delta t} = 36t^5 + 9$$

$$s = t^3 + c \quad \xrightarrow{\hspace{10em}} \quad \frac{\delta s}{\delta t} = 3t^2$$

Это правило пригодится вам во многих случаях, а зачастую ничего другого вам и не потребуется. Верно и то, что правило применимо только к полиномам, т.е. к выражениям, состоящим из переменных в различных степенях, как, например, выражение $y = ax^3 + bx^2 + cx + d$, но не к функциям вида $\sin(x)$ или $\cos(x)$. Это не является существенным недостатком, поскольку в огромном количестве случаев вам вполне хватит правила дифференцирования степеней.

Однако для нейронных сетей нам понадобится еще один инструмент, о котором сейчас пойдет речь.

Функции функций

Представьте, что в функции

$$f = y^2$$

переменная y сама является функцией:

$$y = x^3 + x$$

При желании можно переписать эту формулу в виде $f = (x^3 + x)^2$.

Как f изменяется с изменением y ? То есть что собой представляет производная $\partial f / \partial y$? Получить ответ на этот вопрос не составляет труда, поскольку для этого достаточно применить только что полученное нами правило дифференцирования степенных выражений, поэтому $\partial f / \partial y = 2y$.

Но возникает более интересный вопрос: как изменяется f при изменении x ? Ну хорошо, мы могли бы раскрыть выражение $f = (x^3 + x)^2$ и применить уже знакомый подход. Только ни в коем случае не считайте наивно, что производная от $(x^3 + x)^2$ — это $2(x^3 + x)$.

Если бы мы проделали множество подобных вычислений прежним трудоемким способом, предполагающим устремление приращений к нулю в результирующих выражениях, то рано или поздно мы подметили бы еще одну закономерность. Я сразу же дам вам готовый рецепт.

Вот как выглядит новая закономерность.

$$\frac{\delta f}{\delta x} = \frac{\delta f}{\delta y} \cdot \frac{\delta y}{\delta x}$$

Это очень мощный результат, который называется **цепным правилом**.

В соответствии с этим правилом нахождение производной в подобных случаях осуществляется поэтапно. Может оказаться так, что для нахождения производной $\frac{\partial f}{\partial x}$ проще найти производные $\frac{\partial f}{\partial y}$ и $\frac{\partial y}{\partial x}$. Если последние две производные действительно вычисляются очень просто, то с помощью этого приема удается находить производные, определить которые другими способами практически невозможно. Цепное правило позволяет разбивать трудные задачи на более легкие.

Рассмотрим следующий пример и применим к нему цепное правило:

$$f = y^2 \quad \text{и} \quad y = x^3 + x$$

$$\frac{\delta f}{\delta x} = \frac{\delta f}{\delta y} \cdot \frac{\delta y}{\delta x}$$

Мы разбили задачу на две простые части. Первая часть дает $(\frac{\partial f}{\partial y}) = 2y$, вторая — $(\frac{\partial y}{\partial x}) = 3x^2 + 1$. Объединяя эти части с помощью цепного правила, получаем

$$\frac{\delta f}{\delta x} = (2y) * (3x^2 + 1)$$

Мы знаем, что $y = x^3 + x$, поэтому можем получить выражение, содержащее только x :

$$\frac{\delta f}{\delta x} = (2(x^3 + x)) * (3x^2 + 1)$$

$$\frac{\delta f}{\delta x} = (2x^3 + 2x)(3x^2 + 1)$$

Магия!

Возможно, вас так и подмывает спросить: а почему бы не представить f в виде функции, зависящей только от x , и применить простое правило дифференцирования степеней к результирующему полиному? Мы могли бы это сделать, но тогда я не продемонстрировал бы вам, как работает цепное правило, которое позволяет разгрызать более твердые орешки.

Рассмотрим еще один пример, на этот раз последний, который демонстрирует, как обращаться с переменными, не зависящими от других переменных.

Предположим, имеется функция

$$f = 2xy + 3x^2z + 4z$$

В ней переменные x , y и z не зависят одна от другой. Что мы подразумеваем под независимостью переменных? Под этим подразумевается, что каждая из переменных x , y и z может принимать любые значения, какими бы ни были значения остальных переменных — их изменения на нее не влияют. В предыдущем примере это было не так, поскольку значение y определялось значением выражения $x^3 + x$, а значит, переменная y зависела от x .

Что такое $\frac{\partial f}{\partial x}$? Рассмотрим каждый член длинного полинома по отдельности. Первый член — это $2xy$, поэтому его производная равна $2y$. Почему так просто? Да потому, что y не зависит от x . Когда мы интересуемся величиной $\frac{\partial f}{\partial x}$, нас интересует, как изменяется f при изменении x . Если переменная y не зависит от x , то с ней можно обращаться как с константой. На ее месте могло бы быть любое другое число, например 2, 3 или 10.

Идем дальше. Следующий член выражения — $3x^2z$. Применяя правило понижения степеней, получаем $2 \cdot 3xz$ или $6xz$. Мы рассматриваем z как обычную константу, значением которой может быть 2, 4 или 100, поскольку x и z не зависят друг от друга. Изменение z не влияет на x .

Последний член, $4z$, вообще не содержит x . Поэтому он полностью исчезает, так как мы рассматриваем его как постоянное число, которым, например, могло бы быть 2 или 4.

Вот как выглядит окончательный ответ:

$$\frac{\delta f}{\delta x} = 2y + 6xz$$

В этом примере была важна возможность уверенно игнорировать переменные, о которых известно, что они являются независимыми. Это значительно упрощает дифференцирование довольно сложных выражений, в чем часто возникает необходимость в ходе анализа нейронных сетей.

Вы освоили дифференциальное исчисление!

Если вам удалось одолеть материал, изложенный в этом приложении, примите мои поздравления!

Я постарался донести до вас, в чем состоит суть дифференциального исчисления и как оно возникло на основе постепенного улучшения приближенных решений. Всегда пытайтесь применять описанные в данном приложении методы, если другие способы решения задачи не приводят к успеху.

Используя метод дифференцирования выражений путем понижения степеней переменных и применения цепного правила для нахождения производных сложных функций, вы сможете многое узнать о природе и механизмах функционирования нейронных сетей.

Желаю вам максимально эффективно использовать этот мощный инструмент, которым вы теперь владеете!