

# Перцептрон. Многослойные нейронные сети

# Содержание

<b>1</b>	<b>Однослойный перцептрон</b>	<b>2</b>
1.1	Введение . . . . .	2
1.2	Схема искусственного нейрона . . . . .	3
1.3	Построение архитектуры модели классификатора . . . . .	4
1.4	Аналитическое построение модели . . . . .	6
1.5	Градиентный и стохастический градиентный спуски . . . . .	8
1.6	Функции активации: начало . . . . .	13
<b>2</b>	<b>Многослойный перцептрон</b>	<b>16</b>
2.1	Многослойные нейронные сети . . . . .	16
2.2	Backpropagation: начало . . . . .	18
2.3	Функции активации: продолжение . . . . .	21
<b>3</b>	<b>Заключение</b>	<b>26</b>

# 1 Однослойный перцептрон

## 1.1 Введение

Здравствуйте, уважаемые слушатели. В этой лекции мы переходим к самой сути – к непосредственно построению многослойных нейронных сетей. Можно смело утверждать, что данная лекция является ключевой для понимания всего того, что происходит в мире нейросетей. Мы расскажем о том, что такое нейронная сеть, объясним базовые принципы построения многослойных нейронных сетей, покажем кроющуюся за этим математику и, конечно же, расскажем, как на практике происходит обучение. Для начала, однако, обсудим, насколько нейроны в сети связаны с нейронами в головном мозге человека, а также стоит ли между этими понятиями ставить знаки равенства или хотя бы проводить параллели.

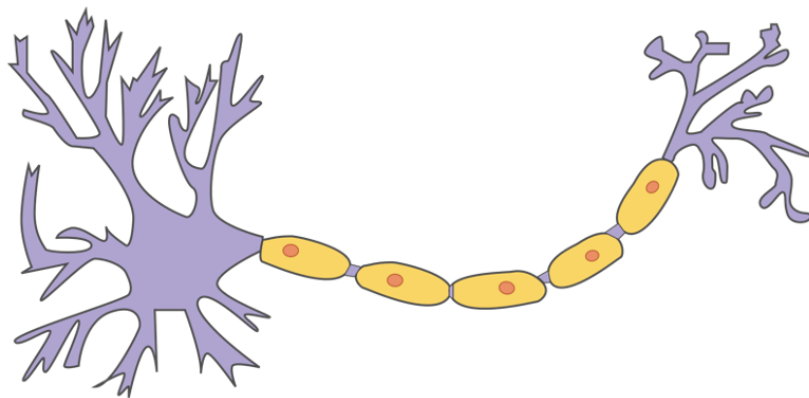


Рис. 1: Типичная структура нейрона

На самом деле, конечно, нейронная (или, как еще часто подчеркивают, искусственная нейронная) сеть имеет мало отношения к нейронам в человеческом мозге. В то же время вдохновение на создание архитектуры нейронной сети все-таки пришло из биологии. Если опустить сложные биологические детали, то настоящий нейрон – это некоторая клетка (пример такого рода клетки см. на рисунке 1), у которой есть множество входов, в которые приходят некоторые импульсы от других похожих клеток; эти импульсы каким-то образом обрабатываются, и после обработки нейрон выдает некоторый импульс на выходе, который, в свою очередь, идет на вход уже другим клеткам, и так далее.

Мы же в лекции будем рассматривать достаточно простое и грубое, однако весьма эффективное математическое приближение описанной модели –

искусственный нейрон (перцептрон).

## 1.2 Схема искусственного нейрона

Итак, давайте обсудим, из чего же состоит наш искусственный нейрон (или, что то же самое, перцептрон или юнит), рисунок 2. У искусственного нейрона, как и у настоящего, есть входы. Допустим, что их  $n$  штук. На каждый вход подается некоторое число  $x_1, x_2, \dots, x_n$ , соответственно. Каждому входу нейрона сопоставляется некоторый вес  $w_1, w_2, \dots, w_n$ , кроме того добавляется вес  $w_0$ , отвечающий за сдвиг. Тем самым, общее количество весов оказывается равным  $(n + 1)$ . Нейрон берет числа на входе, поочередно умножает их на веса и складывает, вычисляя значение следующего выражения:

$$w_1x_1 + w_2x_2 + \dots + w_nx_n - w_0.$$

После этого вычисляется некоторая функция  $\varphi$  (так называемая функция активации, часто – нелинейная) от полученного на предыдущем шаге значения:

$$\varphi(w_1x_1 + w_2x_2 + \dots + w_nx_n - w_0) = \varphi\left(\sum_{i=1}^n w_ix_i - w_0\right).$$

Полученное в результате этих манипуляций число и является выходным сиг-

### Входы

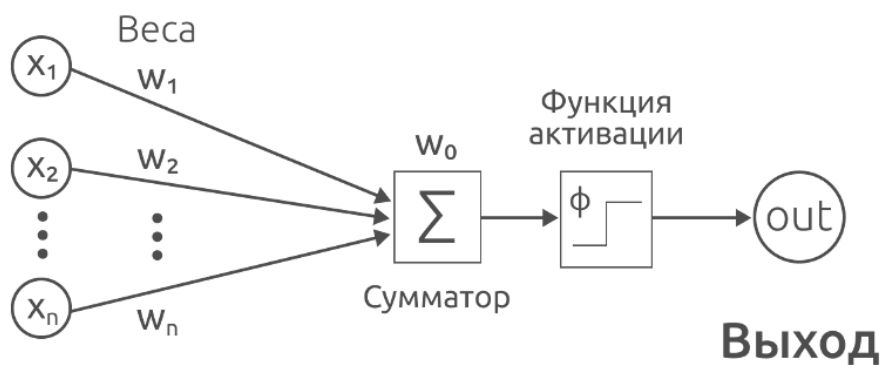


Рис. 2: Схема искусственного нейрона

налом нейрона, который может пойти на вход множеству других нейронов; то, куда выходной сигнал отправится, каким нейронам он достанется и определяет архитектуру нейронной сети.

**Замечание 1.2.1** Полезно отметить, что часто схему нейрона рисуют следующим образом, рисунок 3. В этой схеме искусственным образом добавлен еще один вход, значение на котором всегда равно  $-1$ . Тогда блок  $\Sigma$

в схеме просто-напросто вычисляет сумму произведений значений на входе на соответствующие входам веса. Понятно, что обе схемы совершенно равноправны.

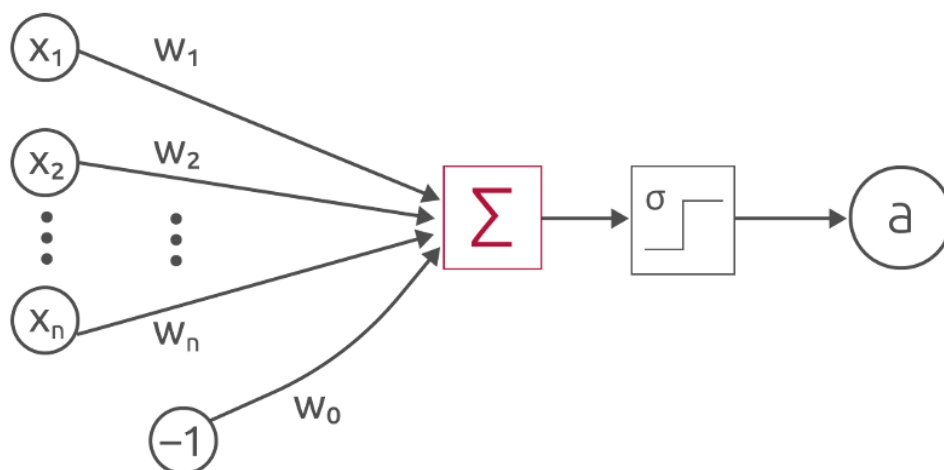


Рис. 3: Схема искусственного нейрона с искусственным входом

Отметим отдельно, что в процессе вычисления веса  $w_i$  не меняются; определение весов – это и есть задача, которая решается при обучении модели, заранее веса, конечно же, неизвестны. Детальнее мы это обсудим несколько позже.

### 1.3 Построение архитектуры модели классификатора

Оказывается, даже такая, на первый взгляд бесхитростная модель, уже может помочь нам при решении разного рода задач. Для демонстрации этого, построим модель классификации на основе набора данных MNIST, использующую, как не трудно догадаться, 10 нейронов. Понятно, что эта модель обобщается и на другие данные, с другим числом классов, но мы будем придерживаться конкретного применения.

**Замечание 1.3.1** Напомним, что упомянутый набор данных представляет из себя матричное представление черно-белых изображений рукописных цифр размера  $8 \times 8$ . Пример изображения и его матричного представления (значение в матрице отвечает интенсивности соответствующего пикселя на изображении) можно видеть на рисунке 4.

Преобразуем исходные данные, чтобы подавать на вход наших нейронов числа, а не векторы. Для этого «распрявим» изображения, записав все интен-

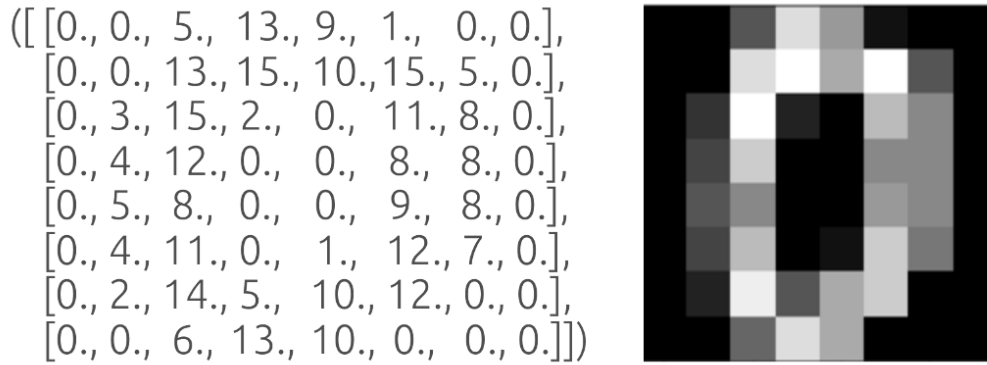


Рис. 4: Пример данных MNIST

сивности в строчку. Тем самым, мы естественным образом установили количество входов у наших нейронов – их 64. Для изображения, озвученного выше, набор признаков получается таким, рисунок 5. Итак, как же постро-

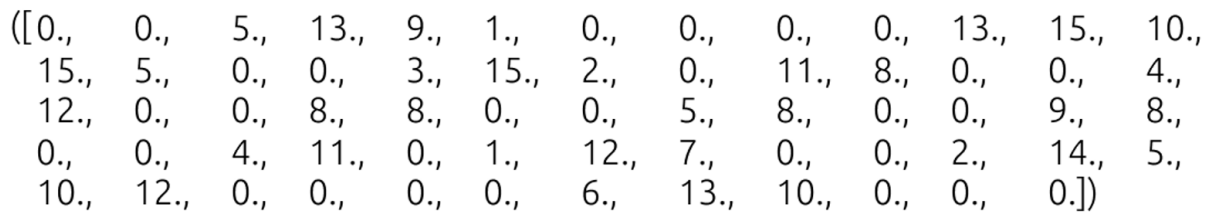


Рис. 5: Пример данных MNIST

ить архитектуру для классификации этого набора данных? Например, так. Каждому классу  $0, 1, \dots, 9$  сопоставим свой нейрон со своими весами. Итого, каждый нейрон получает 65 весов  $w_{i0}, w_{i1}, \dots, w_{i64}$ ,  $i \in \{0, 1, \dots, 9\}$ . Схема условно может быть изображены следующим образом, рисунок 6.

На ней видно, что нулевой нейрон отвечает за цифру 0, первый за цифру 1, и так далее. А что означает слово «отвечает»? Означает оно следующее: мы хотим натренировать модель таким образом (то есть найти такие веса всех нейронов), чтобы на данных, отвечающих цифре  $i$ ,  $i$ -ый нейрон выдавал максимальное значение среди всех остальных,  $i \in \{0, 1, \dots, 9\}$ . Тот нейрон, который выдает это наибольшее значение, часто называют активизирующимся нейроном. Итого, у нас получилась система из 10 нейронов, каждый из которых обладает 65 весами (то есть всего  $65 \cdot 10 = 650$  параметров модели), и мы хотим ее натренировать так, чтобы на новом входном данном активировался правильный нейрон.

**Замечание 1.3.2** Понятно, что таким образом построенная нейронная сеть эквивалентна линейному классификатору, ведь на самом деле мы

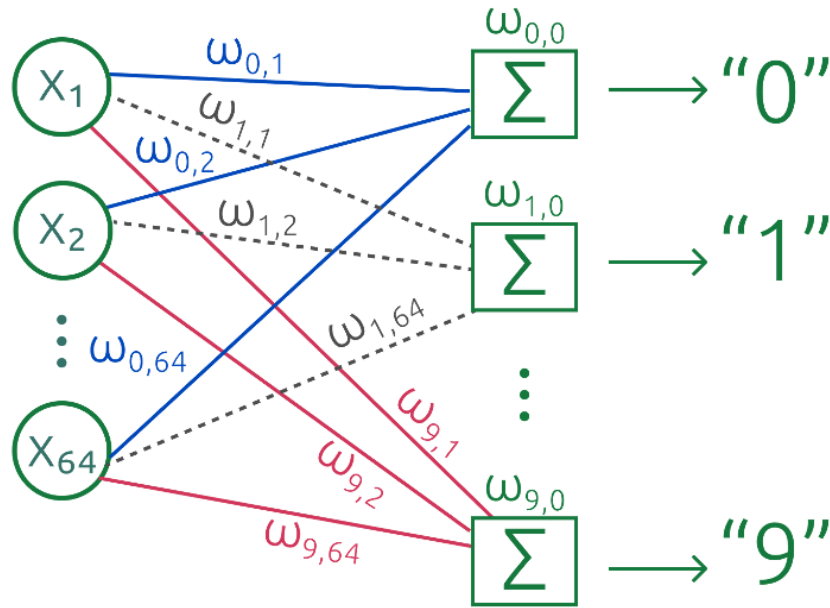


Рис. 6: Пример данных MNIST

ищем 10 гиперплоскостей вида

$$w_{i1}x_1 + w_{i2}x_2 + \dots + w_{i64}x_{64} - w_{i0} = 0, \quad i \in \{0, 1, \dots, 9\},$$

разделяющих наше 64-мерное пространство на какие-то части. Дальше, в зависимости от того, с какой стороны от гиперплоскости оказывается новое наблюдение, а также в зависимости от расстояния до ближайших гиперплоскостей, делается вывод о назначении наблюдению того или иного класса.

Теперь давайте поймем, каким образом мы будем искать веса, ведь, как уже отмечалось, без конкретных весов наша модель представляет лишь теоретический, а вовсе не практический интерес.

## 1.4 Аналитическое построение модели

Итак, давайте составим целевую функцию, минимизируя которую мы сможем найти интересующие нас коэффициенты. Для этого сопоставим выходу каждого нейрона следующее значение (softmax):

$$P(\text{class} = j | x_1, x_1, \dots, x_{64}) = \frac{\exp(w_{j1}x_1 + w_{j2}x_2 + \dots + w_{j64}x_{64} - w_{j0})}{\sum_{i=0}^9 \exp(w_{i1}x_1 + w_{i2}x_2 + \dots + w_{i64}x_{64} - w_{i0})},$$

$j \in \{0, \dots, 9\}$ . Иными словами, для вычисления вероятности отнесения объекта к классу  $j$ , мы вычисляем экспоненту от выхода  $j$ -ого нейрона и делим

это значение на сумму экспонент, вычисленных от выходов каждого из десяти нейронов.

**Замечание 1.4.1** *Отдельно поговорим, почему мы используем softmax. Легко понять, что получающиеся в результате значения образуют вероятностное распределение, которое позволяет сопоставить классифицируемому объекту вероятность отнесения его нашей моделью к каждому конкретному классу. Тем самым, рассматриваемый подход нормализует выходные значения нейронов относительно всех остальных и выдает безразмерную оценку его принадлежности соответствующему классу.*

Ну а дальше все стандартно – используем метод максимального правдоподобия. Имея тренировочный набор данных объема  $N$ :

$$x_i = (x_1^i, x_2^i, \dots, x_{64}^i), \quad y_i \in \{0, 1, \dots, 9\}, \quad i \in \{1, 2, \dots, N\},$$

мы максимизируем функцию

$$\text{likelihood} = \prod_{i=1}^N P(\text{class} = y_i | x_1^i, x_2^i, \dots, x_{64}^i),$$

то есть максимизируем вероятность того, что тренировочные объекты относятся к своим истинным классам. Введенная функция зависит от 650 переменных-весов, которые нам и надо определить. Естественно, чтобы не рассматривать произведение, обычно максимизируемую функцию логарифмируют, тем самым задача сводится к максимизации функции

$$\ln(\text{likelihood}) = \sum_{i=1}^N \ln P(\text{class} = y_i | x_1^i, x_2^i, \dots, x_{64}^i).$$

И даже больше, так как обычно (по умолчанию, в том числе библиотеками) решается задача минимизации функции, то задачу максимизации сводят к задаче минимизации и минимизируют функцию **logloss**:

$$\text{logloss} = -\ln(\text{likelihood}) = -\sum_{i=1}^N \ln P(\text{class} = y_i | x_1^i, x_2^i, \dots, x_{64}^i).$$

Понятно, что наша модель по сути переросла в модель многоклассовой логистической регрессии.

**Замечание 1.4.2** *По большому счету, подход softmax приближения вероятностей, использованный нами – эмпирический. В теории нейронных сетей невозможно провести какого-то математического вывода удачности или, наоборот, неудачности данного подхода. На практике предложенный подход softmax используется достаточно часто.*



Отметим и еще одно важное практическое замечание.

**Замечание 1.4.3** Обычно на практике минимизируют не написанную функцию  $\text{logloss}$ , а регуляризованную. Регуляризацию проводят путем добавления функции, зависящей только от параметров модели, а не от тренировочной выборки, например, следующим образом:

$$L = - \sum_{i=1}^N \ln P(\text{class} = y_i | x_1^i, x_2^i, \dots, x_{64}^i) + \lambda R(\omega),$$

где

$$R(\omega) = \sum_{i=0}^9 \sum_{j=0}^{64} \omega_{ij}^2, \quad \text{либо} \quad R(\omega) = \sum_{i=0}^9 \sum_{j=0}^{64} |\omega_{ij}|,$$

либо их комбинация. На самом деле написанное – не что иное как квадрат  $l_2$ -нормы вектора всех весов и  $l_1$ -норма вектора всех весов, соответственно. Регуляризация используется, как обычно, чтобы препятствовать переобучению (если говорить про числа – препятствовать необоснованному росту модулей коэффициентов), и чтобы модель на данных вела себя более плавно, регулярно, рисунок 7. В нашем случае регуляризатор выполняет и еще одну функцию. Дело в том, что предложенная модель оказывается неоднозначной: например, если все  $w_{i0}$  увеличить на одно и то же число, введенные ранее вероятности не поменяются. Введение регуляризатора решает эту проблему. Параметр  $\lambda$  является гиперпараметром модели, его правильный подбор позволяет найти тот самый баланс между точностью и интерпретируемостью модели. Подбор параметра осуществляется стандартным способом, например, при помощи валидации.

Пример обучения написанной нейронной сети можно найти в приложенном блокноте.

## 1.5 Градиентный и стохастический градиентный спуски

Теперь, построив минимизируемую функцию, обсудим некоторые численные методы минимизации: методы градиентного и стохастического градиентного спуска. Оба метода являются итеративными и используют простую механическую идею: находясь в какой-то точке, лежащей на графике функции, для наиболее качественного уменьшения значения функции двигаться нужно в том направлении, в котором функция максимально быстро убывает. А что это за направление? Из курса анализа известно, что для дифференцируемой функции направлением наискорейшего роста является направление, сонаправленное с градиентом. Значит, направление противоположное

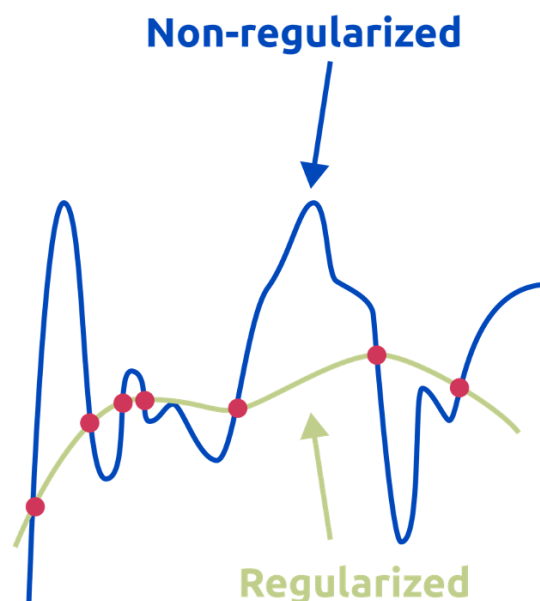


Рис. 7: Модель, обученная с регуляризацией и без нее

градиенту (сонаправленное с антиградиентом) как раз-таки и является направлением наискорейшего убывания функции. На всякий случай напомним полезное для практики определения.

**Определение 1.5.1** Пусть  $f : D \rightarrow \mathbb{R}$ ,  $D \subset \mathbb{R}^k$ . Градиентом функции  $f$  по переменным  $x_1, x_2, \dots, x_k$ ,  $x = (x_1, x_2, \dots, x_k)$  в точке  $x^0 = (x_1^0, x_2^0, \dots, x_k^0)$  называется вектор

$$\nabla_x f(x^0) = \left( \frac{\partial f(x^0)}{\partial x_1}, \frac{\partial f(x^0)}{\partial x_2}, \dots, \frac{\partial f(x^0)}{\partial x_k} \right),$$

если все написанные частные производные существуют.

Итого, алгоритм градиентного спуска для функции  $f$  может быть описан следующим образом.



Рис. 8: Спуск

---

#### Алгоритм 1 Метод градиентного спуска

---

**Вход:** Функция  $f$ , начальное приближение  $x_0$ , точность  $\varepsilon$ , максимальное количество итераций  $N$ , стратегия выбора  $\alpha$ .

- 1: **Для**  $k \in \{1, 2, \dots, N\}$  **Выполнять**
- 2:     Вычислить  $f'(x^0) = \nabla_x f(x^0)$ .
- 3:     Выбрать  $\alpha$  согласно стратегии.
- 4:      $x := x^0 - \alpha f'(x^0)$ .
- 5:     **Если** Выполнен критерий останова **тогда**
- 6:         Возвратить  $x, f(x)$ .
- 7:     **Конец условия**
- 8:      $x^0 := x$
- 9: **Конец цикла**

**Выход:** Точка минимума  $x_{\min}$  и значение в ней  $f(x_{\min})$ .

---

Возможная траектория градиентного спуска показана на рисунке 9. Следя в том числе за плоской картинкой (серыми линиями построены линии уровня нарисованной поверхности), видно, что в точках на линиях уровня траектория минимизации идет перпендикулярно к ним (перпендикулярно к касательным к этим линиям, построенным в этих точках). Это – еще одно свойство градиента.

На рисунке 10 приведено еще несколько траекторий градиентного спуска. В левой части рисунка четко видно, что стартовая точка оказалась весьма неудачно инициализирована – требуется много итераций для нахождения точки минимума. Все потому, что признаки не отмасштабированы, линии уровня минимизируемой функции слишком вытянуты. На правом же рисун-

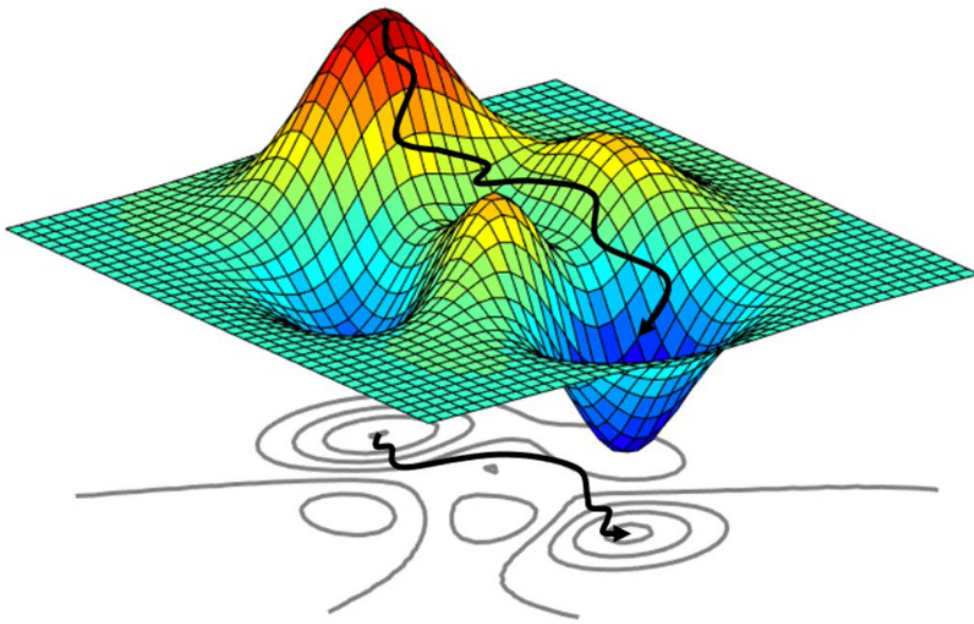


Рис. 9: Траектория градиентного спуска

ке видно, что любая точка старта будет одинаково удачной, так как линии уровня – практически окружности. Этот пример приводит еще один довод в пользу нормализации переменных перед обучением алгоритмов в машинном обучении.

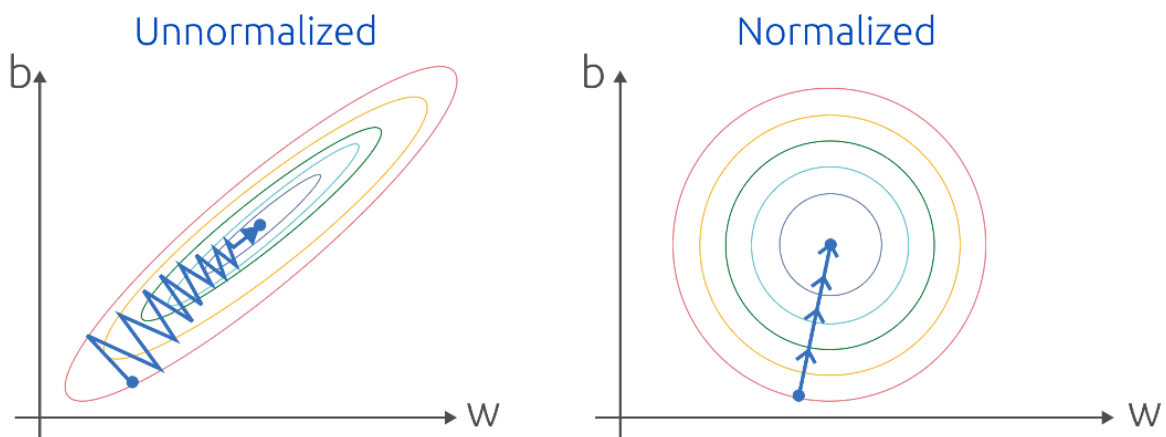


Рис. 10: Траектория градиентного спуска

Отметим также, что стратегия выбора положительного параметра  $\alpha$ , или, грубо говоря, шага метода спуска – это отдельная наука. Часто для этого применяют так называемые правила Армихо и Вульфа, однако мы на этом подробно останавливаться не будем. Наконец, критерий остановки тоже мо-

жет меняться в зависимости от задачи: можно останавливаться, если новая точка незначительно отличается от старой – отслеживать так называемую невязку по точке

$$\|x^k - x^{k+1}\|^2 < \varepsilon \|x^0\|^2,$$

можно останавливаться, если значение функции стало меняться мало – отслеживать так называемую невязку по функции

$$\|f(x^k) - f(x^{k+1})\|^2 < \varepsilon \|f(x^0)\|^2,$$

а можно – отслеживать невязку по градиенту из соображений, что в точке экстремума градиент равен нулю:

$$\|\nabla f(x^{k+1})\|^2 < \varepsilon \|\nabla f(x^0)\|^2.$$

Все это подробно обсуждается в курсах по методам оптимизации. При обучении нейронных сетей обычно смотрят на точность. Если точность перестает увеличиваться, то процесс оптимизации прекращают.

**Замечание 1.5.1** *Естественно, возникает вопрос, а как вычислить или хотя бы приблизить градиент? Ведь не зная градиента, не получится применить и метод спуска; еще раз прокрутите в голове алгоритм градиентного спуска, он явно на каждом шаге использует градиент для вычисления следующего приближения точки минимума. Вариантов подсчета градиента существует достаточно много. Первый из них – вычислить градиент аналитически. Периодически это возможно, но не исключена ошибка как при вычислении, так и при записи формулы в код программы, поэтому второй предлагаемый вариант, с одной стороны, позволяет провести численную проверку аналитической формулы, а с другой – дает альтернативный способ приближения градиента функции – приближение градиента конечными разностями:*

$$\frac{\partial f(x^0)}{\partial x_i} \approx \frac{f(x^0 + \varepsilon e_i) - f(x^0 - \varepsilon e_i)}{2\varepsilon}, \quad \varepsilon > 0,$$

где  $e_i$  – направляющий орт  $i$ -ой координатной оси, обычно

$$e_i = (0, \dots, 1, \dots, 0), \quad i \in \{1, 2, \dots, k\},$$

где 1 стоит на  $i$ -ой позиции, на остальных позициях стоят 0. Такой метод, с одной стороны, чрезвычайно прост в реализации, но с другой стороны крайне неэффективен с точки зрения вычислительной сложности, ведь вычисление (двух значений функции) нужно проводить для **каждого веса** каждого нейрона, причем на **каждом шаге** процесса оптимизации.

*В конце данной лекции мы обсудим намного более эффективный метод вычисления градиента. Здесь же отметим последнее, но важное замечание: с точки зрения математики, чем меньше параметр  $\varepsilon$ , тем лучше написанное приближение аппроксимирует истинное значение частной производной. На компьютере не стоит брать значения  $\varepsilon$  меньшие, чем  $10^{-7}$ ,  $10^{-8}$  в виду ограничений по вычислительной точности.*

Итак, все бы хорошо, но очевидная проблема, с которой сталкивается метод градиентного спуска, частично уже была озвучена: в случае, когда объем тренировочного набора данных велик, вычисление градиента функции по большому количеству параметров становится очень трудоемкой задачей, а метод градиентного спуска сходится (то есть находит оптимальное значение весов) за весьма долгое время. Идея **стохастического градиентного спуска** достаточно проста: давайте искать градиент функции потерь, построенной не по всему набору тренировочных данных, а лишь по маленькому, случайно выбранному на этом шаге, поднабору (batch, minibatch), и делать шаг в градиентном спуске в направлении, противоположном к направлению найденного градиента. Количество необходимых вычислений оказывается сильно меньше, шаг метода делается быстрее, и именно поэтому встроенные в библиотеки оптимизаторы градиентному спуску предпочитают стохастический градиентный спуск.

**Замечание 1.5.2** *Интересно заметить, что, в отличие от градиентного спуска, нет математического обоснования сходимости стохастического градиентного спуска, так как нет гарантии уменьшения значения функции на каждом следующем шаге (может не повезти с направлением градиента, ведь функция потерь строится лишь по части тренировочной выборки). В то же время, особенно в начале работы алгоритма, достаточно большие шансы на то, что метод покажет себя хорошо. Находясь далеко от оптимума у метода есть достаточно большая свобода в выборе направления **приближения** к точке оптимума. Конечно, чем ближе мы подходим к точке оптимума, тем точнее нужно проводить вычисления.*

Итак, мы приходим к важной общей схеме обучения. Сначала набор данных делится на некоторые батчи, затем запускается стохастический градиентный спуск (градиентный спуск на батчах), и, как только все батчи использованы, говорят, что завершается одна **эпоха** обучения.

## 1.6 Функции активации: начало

В начале лекции мы подробно обсудили способ решения задачи классификации, используя достаточно простую архитектуру нейронной сети. Наверное, вы понимаете, что один из ключевых моментов при построении сети

— это выбор функции активации  $\varphi$ . В частности, выбор функции активации позволяет переключаться, скажем, между задачей классификации и регрессии. Обсудим способы выбора функции активации детальнее. Итак, как и прежде, будем рассматривать один нейрон, описываемый схемой 11. Здесь

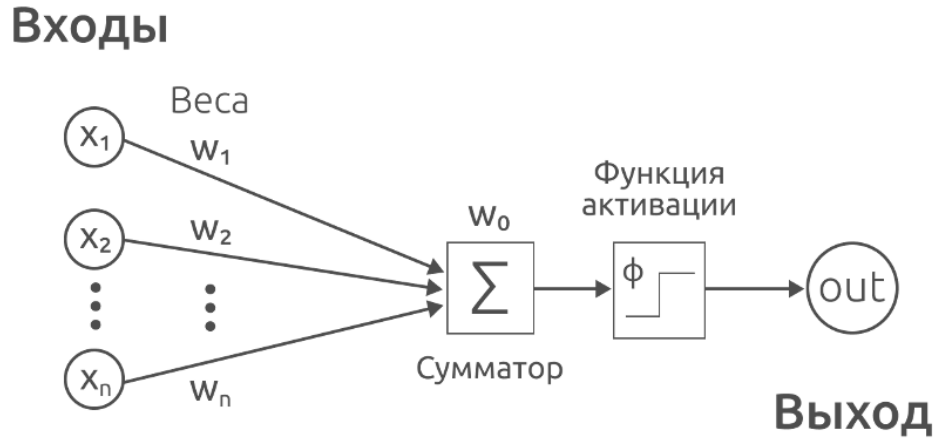


Рис. 11: Схема искусственного нейрона

же приведем набор популярных функций активации и посмотрим, что есть что.

Итак, первая функция (рисунок 12) — это функция единичного скачка. Использование ее в качестве функции активации превращает нашу модель в модель двухклассовой линейной классификации, ведь выходной сигнал нейрона равен 1, если

$$\sum_{i=1}^n w_i x_i - w_0 > 0,$$

то есть если наблюдение находится «над» гиперплоскостью

$$\sum_{i=1}^n w_i x_i - w_0 = 0,$$

ноль, если «под» и 0.5, если «на». Вторая функция имеет тот же смысл, разве что выходные сигналы «названы» несколько иначе.

Третья функция позволяет нам прогнозировать непрерывную переменную и, тем самым, решать задачу регрессии. Понятно, что в качестве функции ошибки может выступать, например, аналог MSE (метод наименьших квадратов):

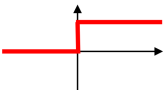
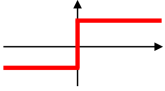
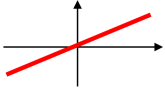

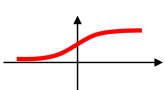
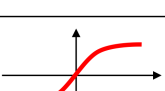


$$L = \sum_{j=1}^N \left( \sum_{i=1}^n w_i x_i^j - w_0 - y_j \right)^2,$$

где  $N$  – объем тренировочной выборки, а

$$x_i = (x_1^i, x_2^i, \dots, x_n^i), \quad y_i \in \mathbb{R}, \quad i \in \{1, 2, \dots, N\}$$

– объекты тренировочной выборки.

Следующие три функции, в некотором смысле, пытаются сгладить единичную ступеньку из первого и второго пунктов. Например, четвертая функция позволяет моделировать двухклассовую классификацию методом опорных векторов, пятая – двухклассовую классификацию с помощью логистической регрессии. Остальные функции весьма полезны в качестве промежуточных функций активации в многослойных нейронных сетях, о которых мы сейчас и поговорим.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016  
(<http://sebastianraschka.com>)

Рис. 12: Функции активации



## 2 Многослойный перцептрон

### 2.1 Многослойные нейронные сети

Итак, мы рассмотрели способ построения простейшей (на самом деле – однослойной) нейронной сети, которая, при нашем выборе функции активации, таутологична линейному классификатору. Однослойной эта сеть называется потому, что сигналы передаются нейронам, каждый из которых сразу выдает выходной сигнал. Легко догадаться, что всю эту конструкцию можно несколько утяжелить, рисунок 13.

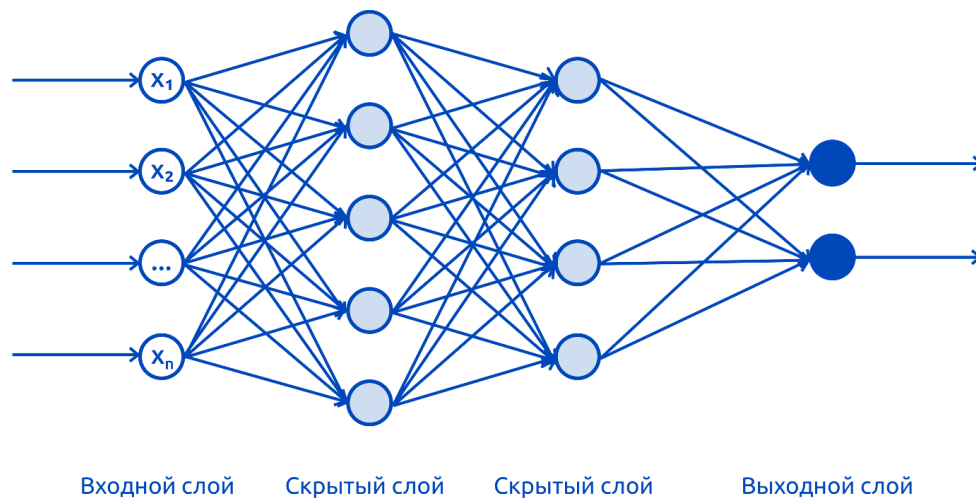


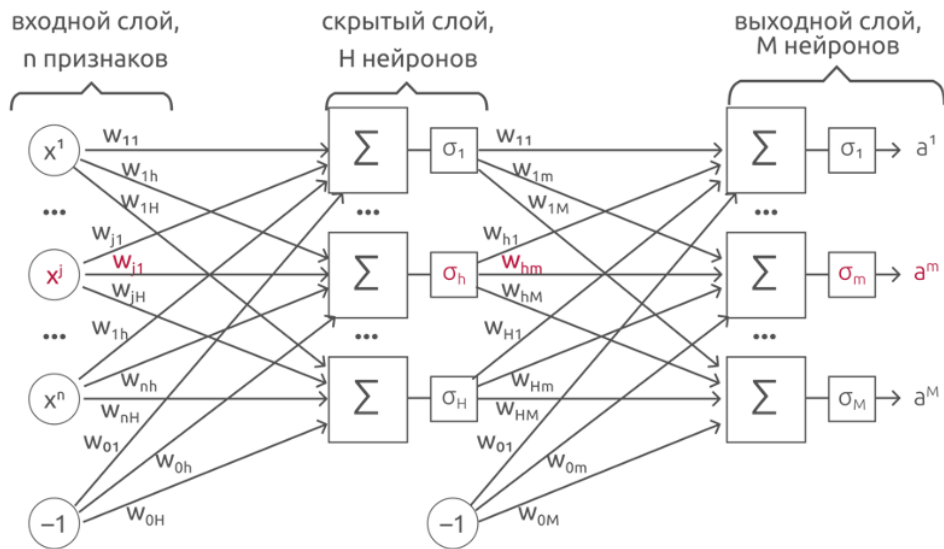
Рис. 13: Многослойный перцептрон

Что мы имеем? Первым идет так называемый **входной слой** – это слой нейронов, которые просто выдают исходные сигналы  $x_1, x_2, \dots, x_n$  и распределяют их по нейронам дальше. По сути, входной слой – это просто наши входы, которые мы обсуждали ранее.

Далее, как уже было сказано, эти входы передаются следующему слою нейронов (так называемому **скрытому слою**), каждый нейрон которого, конечно, имеет свои веса и свою функцию активации. Нейроны этого слоя, отрабатывая, выдают сигналы, которые распределяются по следующему слою (снова скрытому), и так далее вплоть до последнего, **выходного слоя**, набор сигналов которого мы считаем финальным. Рисунок 14 показывает подробную иллюстрацию всех этапов расчетов в двухслойной нейронной сети.

**Замечание 2.1.1** Отметим, что функции активации в скрытых слоях берутся нелинейными, ведь иначе нейрон в скрытом слое оказывается лишним, так как композиция линейных функций – это линейная функция. Есть множество способов выбрать нелинейную функцию активации, но

$Y = \mathbb{R}^M$ , два слоя.



Вектор параметров модели  $w = (w_{jh}, w_{hm}) \in \mathbb{R}^{Hn + H + Mn + M}$

Рис. 14: Двухслойная нейронная сеть

она должна удовлетворять нескольким свойствам – быть, как уже сказано, нелинейной, иметь везде (или почти везде) производную для применения методов спуска и быть вычислительно эффективной. Часто в качестве функции активации в скрытых слоях используется ReLU. Архитектура нейронной сети, то есть количество слоев, количество нейронов на каждом слое, функции активации – это гиперпараметры, которые отдаются на откуп исследователю. Впрочем, есть некоторые практические эвристики. Подробнее про функции активации мы поговорим чуть позже в этой лекции.

Обучение нейронной сети происходит точно таким же образом, как это было и ранее: в зависимости от значений сигналов выходного слоя мы минимизируем некоторую функцию потерь, аналитическое выражение которой, конечно, зависит от задачи, а структура которой с ростом количества слоев и нейронов в каждом слое становится все менее обзримой.

**Замечание 2.1.2** Отметим отдельно, что минимизация происходит при помощи градиентного спуска и его аналогов точно таким же образом, как и ранее, однако теперь функция потерь представляет из себя сложную композицию нелинейных функций, а значит подсчет градиента становится еще более трудоемкой задачей.

Естественно, возникает вопрос: а как эффективно вычислять градиент у такой функции?

## 2.2 Backpropagation: начало

Поясним идею вычисления производной сложной функции компьютерными средствами. Возьмем для примера функцию

$$f(x, y, z) = 1 + 2^{xy+4z}$$

и вычислим ее градиент при  $x = -2$ ,  $y = 2$ ,  $z = 2$ . В данном случае это не сложно сделать аналитически, результат будет таков:

$$\frac{\partial f}{\partial x} = 2^{xy+4z} y \ln 2 \rightarrow 32 \ln 2,$$

$$\frac{\partial f}{\partial y} = 2^{xy+4z} x \ln 2 \rightarrow -32 \ln 2,$$

$$\frac{\partial f}{\partial z} = 2^{xy+4z} 4 \ln 2 \rightarrow 64 \ln 2.$$

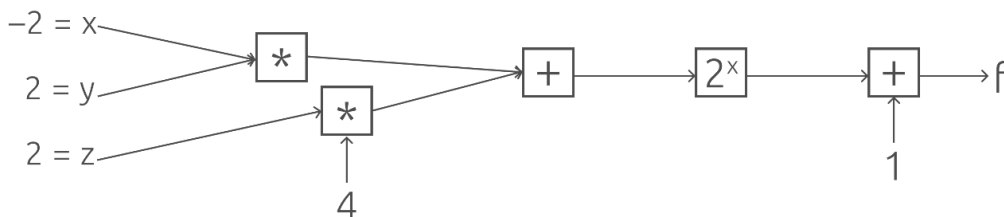


Рис. 15: Граф вычислений

Для организации вычислений на компьютере полезно составить граф вычислений, который составляется на основе приоритета арифметических операций. Граф выглядит следующим образом, рисунок 15. Итак, сначала перемножаются  $x$  и  $y$ , затем  $z$  умножается на 4, результаты этих операций складываются, становятся степенью двойки, и затем финально прибавляется единица.

Подпишем над линиями связи значения, передающиеся в следующий блок, рисунок 16. Отследите внимательно все операции. Видно, что значение функции в предложенной точке равно 17. Давайте теперь обсудим, как вычислить градиент данной функции. Для этого мы будем использовать формулу производной сложной функции: если рассматривается функция  $f(g(x))$ , то

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}.$$

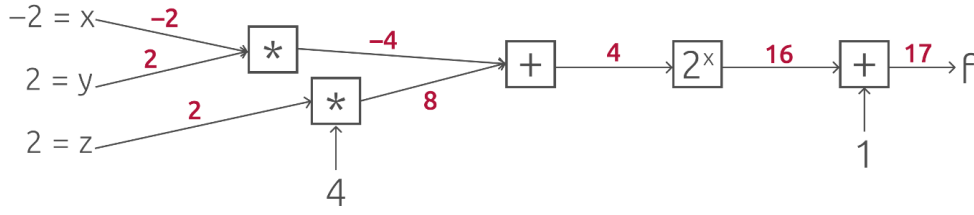


Рис. 16: Граф вычислений со значениями

Теперь опишем алгоритм вычисления градиента на основе графа вычислений. Для удобства обозначим все блоки заглавными латинскими буквами, рисунок 17. Будем идти справа налево, постепенно усложняя нашу производную. Сначала вычислим производную нашей функции по  $E$ . Так как  $f = E$ , то

$$\frac{\partial f}{\partial E} = 1,$$

запишем это. Далее вычислим производную по  $D$ :

$$\frac{\partial f}{\partial D} = \frac{\partial f}{\partial E} \frac{\partial E}{\partial D} = |E = 1 + D| = 1,$$

ведь первый сомножитель мы уже знаем. Идем дальше, ищем производную по  $C$ :

$$\frac{\partial f}{\partial C} = \frac{\partial f}{\partial D} \frac{\partial D}{\partial C} = |D = 2^C| = 2^C \ln 2 \rightarrow 2^4 \ln 2.$$

Пойдем сначала по нижней ветке и найдем производную по  $B$ :

$$\frac{\partial f}{\partial B} = \frac{\partial f}{\partial C} \frac{\partial C}{\partial B} = |C = A + B| = 2^4 \ln 2.$$

Наконец, находим производную по  $z$ :

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial B} \frac{\partial B}{\partial z} = |B = 4z| = 2^6 \ln 2.$$

Теперь возвращаемся к верхней ветке. Сначала ищем производную по  $A$ :

$$\frac{\partial f}{\partial A} = \frac{\partial f}{\partial C} \frac{\partial C}{\partial A} = |C = A + B| = 2^4 \ln 2.$$

Ну и осталось найти

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial A} \frac{\partial A}{\partial x} = |A = xy| = y2^4 \ln 2 \rightarrow 2^5 \ln 2$$

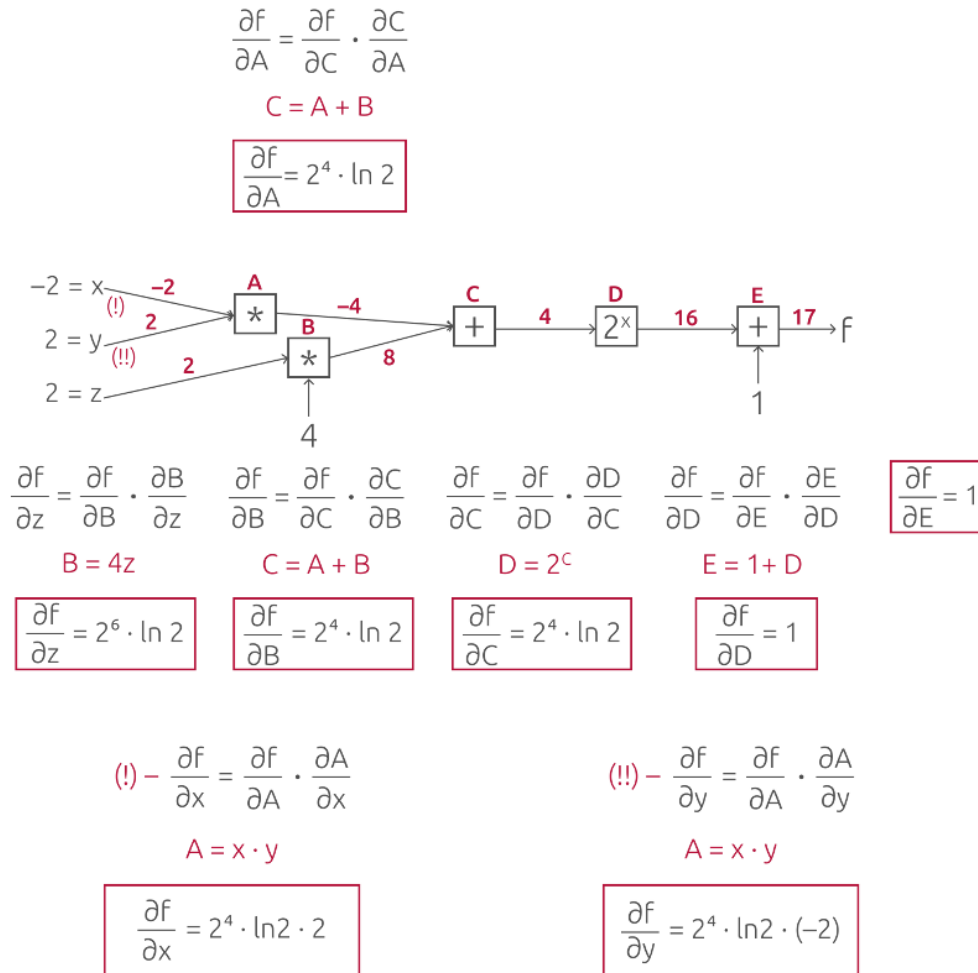


Рис. 17: Вычисление производной

и

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial A} \frac{\partial A}{\partial y} = |A = xy| = x2^4 \ln 2 \rightarrow -2^5 \ln 2.$$

Итак, интересующий нас градиент найден.

**Замечание 2.2.1** Обратим особое внимание, что во время вычислений мы не выписываем градиент полностью, мы вычисляем его локально, в каждом узле, шаг за шагом, проходя всего один раз по графу вычислений.

А как это применяется к нейронным сетям? Очень просто. Посмотрите на схему нейронной сети для обсуждаемой ранее модели, скажем, с тремя слоями, 18. Регуляризованная функция потерь будет иметь следующий вид:

$$L(w, \lambda) = - \sum_{i=1}^N \ln P(\text{class} = y_i | x_i) + \lambda \|w\|_2^2,$$

где  $\|w\|_2^2$  – квадрат  $l_2$  нормы вектора весов. Аналитическое выражение для вероятности под логарифмом нас сейчас не интересует, нам нужно разобраться с идеологией вычислений. Построим по функции граф вычислений, рисунок 19. Имея граф вычислений, все делается по описанной нами схеме.

**Замечание 2.2.2** Конечно, при работе с нейронными сетями, все умножения на вектор весов удобнее переписывать в матричной терминологии. В этом случае производные вычисляются несколько другим способом, но вся идеология остается прежней. Сейчас мы на этом останавливаться не будем

По такой схеме и происходит батч градиентный спуск: данные подаются для тренировки, иногда перемешиваются и зацикливаются, и все это происходит до тех пор, пока мы не решим, что сеть обучена. Как решать, когда остановиться – это отдельная тема, к которой мы вернемся позже.

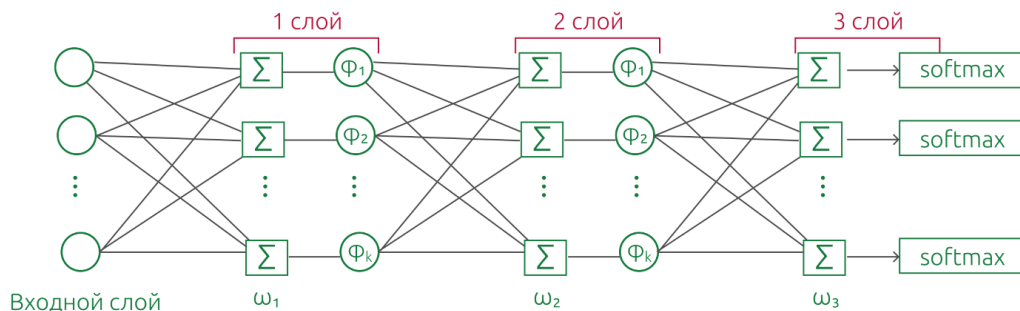


Рис. 18: Схема сети

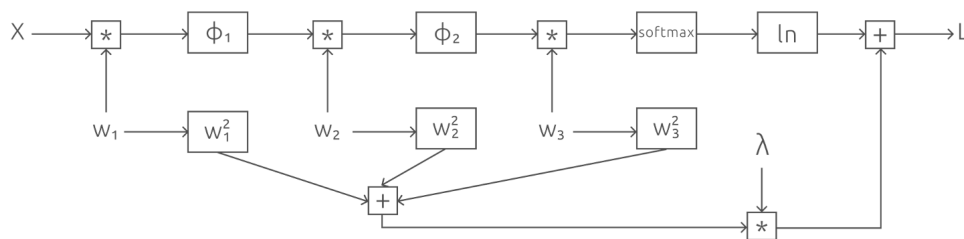


Рис. 19: Граф лосса

## 2.3 Функции активации: продолжение

Как вы, наверное, уже поняли, при конструировании нейронной сети одним из важнейших параметров является наличие корректно выбранной функции активации. В этом фрагменте мы с вами ответим сразу на несколько вопросов: зачем нужны функции активации, на что они влияют, каким условиям они должны удовлетворять и какие функции активации существуют в принципе.

Для ответа на первый вопрос начнем с того, что вспомним, что делает нейрон. Нейрон считает взвешенную сумму входов, добавляет смещение и выдает полученный результат  $Y$  согласно следующему соотношению:

$$Y = \sum_{i=1}^n w_i x_i - w_0.$$

С одной стороны, несомненным плюсом является то, что написанная операция очень простая. В этой простоте кроется и обратная сторона медали – операция-то линейная. Таким образом, даже комбинируя множество нейронов и слоев, не используя функции активации, вся наша искусственная нейронная сеть математически будет являться линейной комбинацией входных данных. Это значит, что все слои и нейроны можно «схлопнуть» в одну большую линейную комбинацию параметров. Такая сеть (без функций активации) может аппроксимировать лишь линейные зависимости в данных, что является существенным ограничением и нам, конечно, не подходит. Впрочем, об этом мы упоминали и ранее.

Функции активации – это элементы, которые вставляются между слоями нейронной сети. Они получают на вход результат нейрона и преобразовывают его согласно некоей математической функции. На данную функцию мы с вами будем накладывать несколько условий:

- **Нелинейность.** Так как задача нейронной сети – выявлять нелинейные зависимости в данных, нам необходимо внести обучаемую нелинейность в вычисления внутри сети. Почти всегда функции активации являются нелинейными для удовлетворения вышеуказанного условия.
- **Наличие производной (почти везде).** Мы с вами уже обсудили метод градиентного спуска, который позволяет обучать нейронную сеть, явно используя вычисление градиента. Однако, для того, чтобы градиент можно было вычислить, функция активации, так как она тоже участвует в вычислениях, должна обладать производной. Математически требуется, чтобы функция была везде дифференцируема, однако в реальности допускается наличие одной или нескольких точек, где производная не определена. Все дело в том, что в реальных вычислениях вероятность получения конкретного, наперед заданного значения, равна нулю.

Еще одним фактором, который стоит учитывать при выборе функции активации, является область значений данной функции. В некоторых задачах, например, классификации, мы хотим, чтобы выход алгоритма находился в некоторых границах – к примеру, внутри отрезка  $[-1, 1]$ . Этого можно добиться,

выбрав функцию активации с соответствующей областью значений или даже модифицировав ее, добившись необходимых вам характеристик. Кроме того, область значений функции напрямую влияет на величину градиента: если функция может принимать очень большие значения или ее график является «достаточно крутым», то абсолютные значения выходов и градиентов соответственно могут оказаться достаточно большими для того, чтобы в процессе обучения возникали проблемы – такие ситуации называются «взрывающимся градиентом». Подробнее это будет обсуждаться в дальнейшем. Давайте рассмотрим несколько примеров функций активации и выясним их плюсы и минусы.

**Сигмоида.** Данная функция активации является одной из самых часто используемых функций активации в нейронных сетях, ее аналитическое выражение таково:

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

а график представлен на рисунке 20. Она имеет ограниченную область зна-

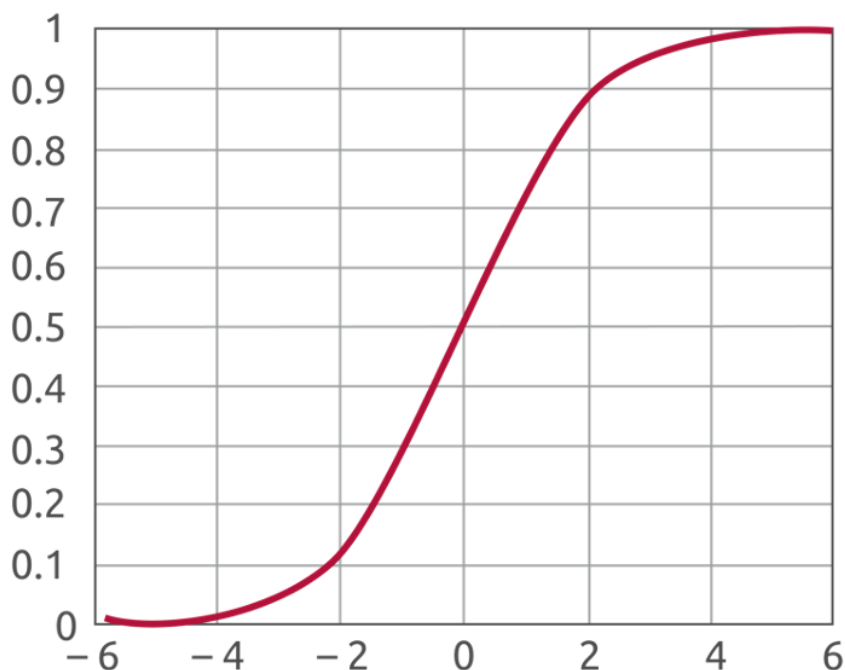


Рис. 20: Сигмоида

чений (ее значения зажаты в промежутке  $(0, 1)$ ) и достаточно плавное изменение, что снижает вероятность «взрыва градиентов»; она нелинейна, ее производная везде определена и является достаточно простой для вычислений:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$



что положительно сказывается на времени тренировки сети; область ее значений делает ее частым выбором в качестве функции активации последнего слоя нейронной сети при решении задачи бинарной классификации, где вам необходимо оценить вероятность принадлежности объекта одному из двух классов (и, автоматически, другому). Из недостатков стоит отметить затухание градиента ближе к границам области значений; на этом вопросе мы подробно остановимся ближе к концу курса.

**Гиперболический тангенс.** Гиперболический тангенс, аналитическое выражение для которого таково:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

является скорректированной функцией сигмоиды, поэтому обладает похожими характеристиками, кроме отличающейся области значений функции и скорости роста: гиперболический тангенс гораздо «круче» растет и убывает. Его производная тоже легко вычисляется:

$$\tanh' x(x) = 1 - \tanh^2(x),$$

а график представлен на рисунке 21.



Рис. 21: Гиперболический тангенс

Давайте теперь двигаться к современности. Наверное, главная идея, которая преобразила архитектуру сегодняшних нейронных сетей – это функция ReLU.

**ReLU (Rectified linear units).** Данная функция активации является кусочно-заданной функцией, ее аналитическое выражение таково:

$$\text{ReLU}(x) = \begin{cases} 0, & x < 0, \\ x, & x \geq 0 \end{cases} = \max(0, x).$$

Несмотря на кажущуюся линейность функции, она не является таковой и подходит для обучения сетей. Конечно, производная данной функции не определена в точке 0, однако в связи с тем, что абсолютное равенство выхода нейрона нулю встречается очень редко, данный факт можно проигнорировать или же принудительно использовать фиксированную константу в этой точке. Зато производная этой функции совершенно элементарна и вычислительно проста, что является несомненным плюсом для численных методов оптимизации:

$$\text{ReLU}'(x) = \begin{cases} 1, & x > 0, \\ a, & x = 0, \\ 0, & x < 0 \end{cases}.$$

В частности поэтому данная функция активации достаточно часто используется в больших сетях. Ее использование позволяет сократить время тренировки модели, сохраняя возможность работы с нелинейными зависимостями. Однако, при отрицательных значениях аргумента, производная данной функции равна нулю, что, в свою очередь, обнулит любой градиент для данного нейрона. Эта проблема известна как проблема «умирающего ReLU» – данный нейрон уже никак не будет участвовать в процессе обучения, и с каждым таким нейроном сеть теряет теоретическую глубину.

**LeakyReLU.** Одним из способов решения проблемы умирающего ReLU является использование LeakyReLU:

$$\text{LeakyReLU} = \begin{cases} x, & x \geq 0, \\ ax, & x < 0 \end{cases}, \quad a > 0$$

и обычно значение  $a$  весьма небольшое. Данная функция обладает всеми свойствами ReLU, кроме нулевого градиента в отрицательной области, что исключает проблему умирающих градиентов. Данная функция активации будет более стабильной и позволит более полно использовать запоминающую способность сети. Тем не менее, как и у ReLU, недостатком данной функции также является то, что область значений функции не ограничена и, теоретически, выход нейрона может достигать огромных значений, внося проблемы в вычисление градиентов и работу сети.

**Linear.** Последней рассматриваемой функцией активации является линейная функция. Эта функция – специфический случай, она не используется между слоями, так как является линейной. Однако, достаточно часто ее называют как функцию активации последнего слоя, чтобы подчеркнуть, что выход слоя используется как он есть. Это часто необходимо в задаче регрессии, если областью значений целевого признака является вся числовая ось, и модель должна теоретически уметь выдавать любой ответ из этой области

значений. Однако, повторим еще раз, данная функция не является нелинейной и не стоит ее использовать между слоями, так как связываемые ею слои математически станут эквивалентными одному слою.

Существуют и другие функции активации, мы коснулись лишь самых основных и часто используемых; остальные лучше изучать в практических задачах.

### 3 Заключение

Итак, в этой лекции мы познакомились с основами построения простейших нейронных сетей. Можно сказать, что вся дальнейшая теория нейронных сетей – это усовершенствование идей, изложенных в данной лекции. Именно поэтому очень важно осознать те основы, тот базис, о котором мы говорили сегодня. В заключение, подведем некоторые итоги:

1. Искусственный нейрон, по своей сути, решает задачу линейной классификации или регрессии в зависимости от выбранной функции активации.
2. Нейронная сеть – суперпозиция нейронов с нелинейными функциями активации.
3. Backpropagation – алгоритм быстрого дифференцирования композиции функций, позволяет обучать сети почти любой конфигурации.
4. Функция активации – чуть ли не важнейшая характеристика нейронной сети, требующая отдельного внимания и качественного участия исследователя.

Удачи и до новых встреч!