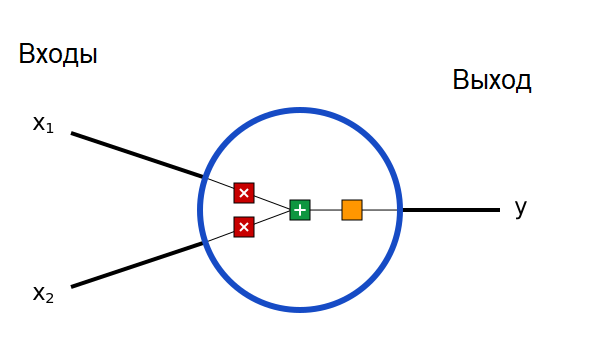
Пишем нейросеть на Python с нуля

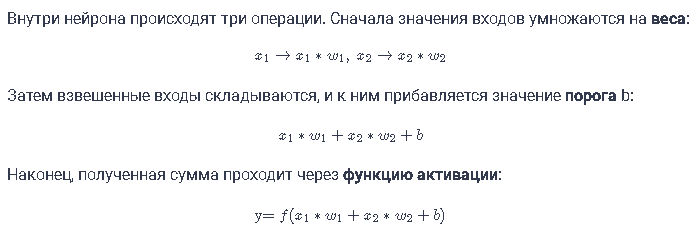
Термин "нейронные сети" сейчас можно услышать из каждого утюга, и многие верят, будто это что-то очень сложное. На самом деле нейронные сети совсем **не такие сложные**, как может показаться! Мы разберемся, как они работают, реализовав одну сеть с нуля на Python.

Эта статья предназначена для **полных новичков**, не имеющих **никакого**опыта в машинном обучении. Поехали!

1. Составные элементы: нейроны

Прежде всего нам придется обсудить нейроны, базовые элементы нейронной сети. Нейрон принимает несколько **входов**, выполняет над ними кое-какие **математические операции**, а потом выдает один **выход**. Вот как выглядит нейрон с двумя входами:

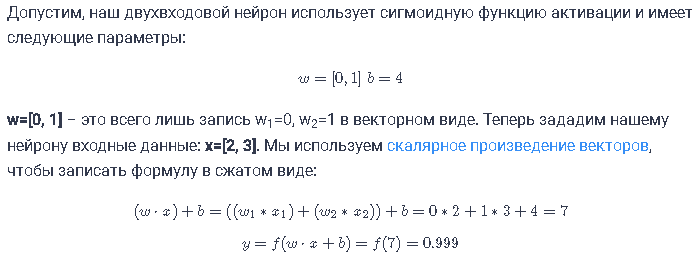


Функция активации преобразует неограниченные значения входов в выход, имеющий ясную и предсказуемую форму. Одна из часто используемых функций активации – **[сигмоида](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BC%D0%BE%D0%B8%D0%B4%D0%B0" \t "_blank)**:

Сигмоида

Сигмоида выдает результаты в интервале **(0, 1)**. Можно представить, что она «упаковывает» интервал от минус бесконечности до плюс бесконечности в (0, 1): большие отрицательные числа превращаются в числа, близкие к **0**, а большие положительные – к **1**.

Простой пример

Наш нейрон выдал 0.999 при входах x=[2, 3]. Вот и все! Процесс передачи значений входов дальше, чтобы получить выход, называется [**прямой связью (feed forward)**](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C_%D1%81_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%D1%8E).

Пишем код для нейрона

Настало время написать свой нейрон! Мы используем [NumPy](http://www.numpy.org/" \t "_blank), популярную и мощную расчетную библиотеку для Python, которая поможет нам с вычислениями:

import numpy as np

def sigmoid(x):

# Наша функция активации: f(x) = 1 / (1 + e^(-x))

return 1 / (1 + np.exp(-x))

class Neuron:

def \_\_init\_\_(self, weights, bias):

self.weights = weights

self.bias = bias

def feedforward(self, inputs):

# Умножаем входы на веса, прибавляем порог, затем используем функцию активации

total = np.dot(self.weights, inputs) + self.bias

return sigmoid(total)

weights = np.array([0, 1]) # w1 = 0, w2 = 1

bias = 4 # b = 4

n = Neuron(weights, bias)

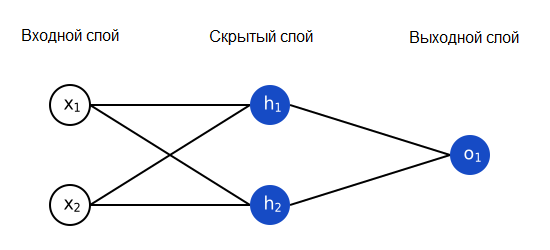
x = np.array([2, 3]) # x1 = 2, x2 = 3

print(n.feedforward(x)) # 0.9990889488055994

Узнаете эти числа? Это тот самый пример, который мы только что рассчитали! И мы получили тот же результат – **0.999**.

2. Собираем нейронную сеть из нейронов

Нейронная сеть – это всего лишь несколько нейронов, соединенных вместе. Вот как может выглядеть простая нейронная сеть:



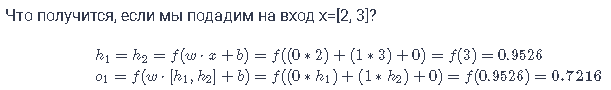
У этой сети два входа, скрытый слой с двумя нейронами (h1 и h2) и выходной слой с одним нейроном (o1). Обратите внимание, что входы для o1 – это выходы из h1 и h2. Именно это создает из нейронов сеть.

Замечание

Скрытый слой – это любой слой между входным (первым) слоем сети и выходным (последним). Скрытых слоев может быть много!

Пример: прямая связь

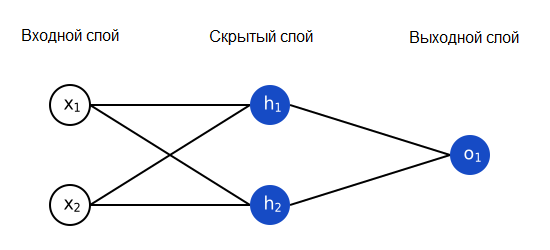
Давайте используем сеть, изображенную выше, и будем считать, что все нейроны имеют одинаковые веса w=[0, 1], одинаковые пороговые значения b=0, и одинаковую функцию активации – сигмоиду. Пусть h1, h2 и o1 обозначают выходные значения соответствующих нейронов.

Если подать на вход нашей нейронной сети **x=[2, 3]**, на выходе получится **0.7216**. Достаточно просто, не правда ли?

Нейронная сеть может иметь **любое количество слоев**, и в этих слоях может быть **любое количество нейронов**. Основная идея остается той же: передавайте входные данные по нейронам сети, пока не получите выходные значения. Для простоты мы будем использовать сеть, показанную выше, до конца статьи.

Пишем код нейронной сети

Давайте реализуем прямую связь для нашей нейронной сети. Напомним, как она выглядит:



import numpy as np

# ... вставьте сюда код из предыдущего раздела

class OurNeuralNetwork:

'''

Нейронная сеть с:

- 2 входами

- скрытым слоем с 2 нейронами (h1, h2)

- выходным слоем с 1 нейроном (o1)

Все нейроны имеют одинаковые веса и пороги:

- w = [0, 1]

- b = 0

'''

def \_\_init\_\_(self):

weights = np.array([0, 1])

bias = 0

# Используем класс Neuron из предыдущего раздела

self.h1 = Neuron(weights, bias)

self.h2 = Neuron(weights, bias)

self.o1 = Neuron(weights, bias)

def feedforward(self, x):

out\_h1 = self.h1.feedforward(x)

out\_h2 = self.h2.feedforward(x)

# Входы для o1 - это выходы h1 и h2

out\_o1 = self.o1.feedforward(np.array([out\_h1, out\_h2]))

return out\_o1

network = OurNeuralNetwork()

x = np.array([2, 3])

print(network.feedforward(x)) # 0.7216325609518421

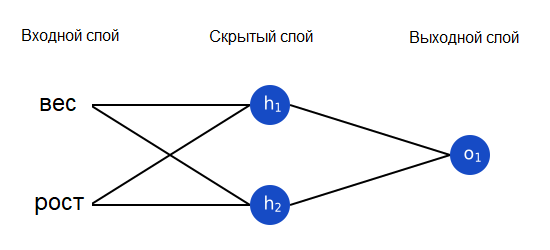
Мы снова получили 0.7216! Похоже, наша сеть работает.

3. Обучаем нейронную сеть (часть 1)

Допустим, у нас есть следующие измерения:

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Вес (в фунтах)** | **Рост (в дюймах)** | **Пол** |
| Алиса | 133 (54.4 кг) | 65 (165,1 см) | Ж |
| Боб | 160 (65,44 кг) | 72 (183 см) | М |
| Чарли | 152 (62.2 кг) | 70 (178 см) | М |
| Диана | 120 (49 кг) | 60 (152 см) | Ж |

Давайте обучим нашу нейронную сеть предсказывать пол человека по его росту и весу.



Мы будем представлять мужской пол как 0, женский – как 1, а также сдвинем данные, чтобы их было проще использовать:

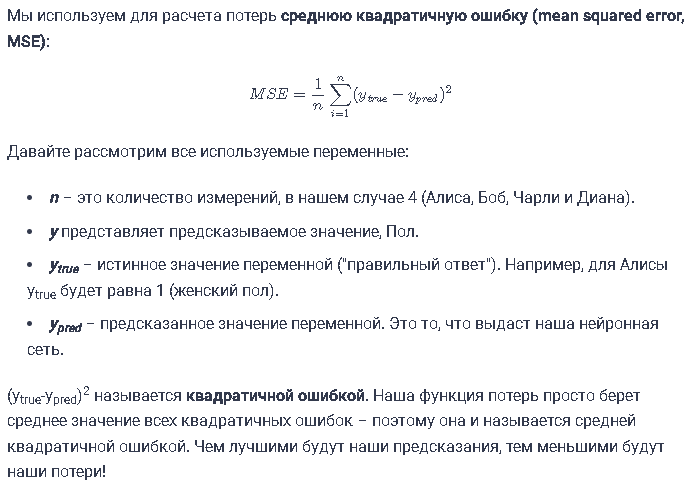
|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Вес (минус 135)** | **Рост (минус 66)** | **Пол** |
| Алиса | -2 | -1 | 1 |
| Боб | 25 | 6 | 0 |
| Чарли | 17 | 4 | 0 |
| Диана | -15 | -6 | 1 |

Замечание

Я выбрал величину сдвигов (135 и 66), чтобы числа выглядели попроще. Обычно сдвигают на среднее значение.

Потери

Прежде чем обучать нашу нейронную сеть, нам нужно как-то измерить, насколько "хорошо" она работает, чтобы она смогла работать "лучше". Это измерение и есть [**потери (loss)**](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C).

Лучшие предсказания = меньшие потери.

**Обучение нейронной сети = минимизация ее потерь**.

Пример расчета потерь

Предположим, что наша сеть всегда возвращает 0 – иными словами, она уверена, что все люди мужчины. Насколько велики будут наши потери?

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **ytrue** | **ypred** | **(ytrue-ypred)2** |
| Алиса | 1 | 0 | 1 |
| Боб | 0 | 0 | 0 |
| Чарли | 0 | 0 | 0 |
| Диана | 1 | 0 | 1 |



Пишем функцию средней квадратичной ошибки

Вот небольшой кусок кода, который рассчитает наши потери. Если вы не понимаете, почему он работает, прочитайте в [руководстве NumPy](https://numpy.org/doc/stable/user/quickstart.html#basic-operations) про операции с массивами.

import numpy as np

def mse\_loss(y\_true, y\_pred):

# y\_true и y\_pred - массивы numpy одинаковой длины.

return ((y\_true - y\_pred) \*\* 2).mean()

y\_true = np.array([1, 0, 0, 1])

y\_pred = np.array([0, 0, 0, 0])

print(mse\_loss(y\_true, y\_pred)) # 0.5

Отлично. Идем дальше!

4. Обучаем нейронную сеть (часть 2)

Теперь у нас есть четкая цель: **минимизировать потери** нейронной сети. Мы знаем, что можем изменять веса и пороги нейронов, чтобы изменить ее предсказания, но как нам делать это таким образом, чтобы минимизировать потери?

Внимание: математика!

Этот раздел использует частные производные по нескольким переменным. Если вы плохо знакомы с дифференциальным исчислением, можете просто пропускать математические формулы.

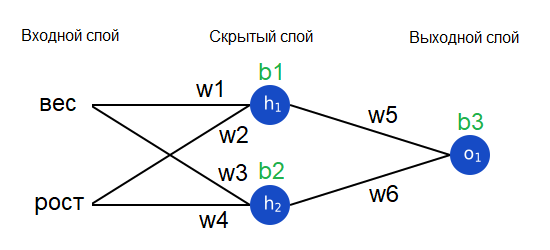
Для простоты давайте представим, что в нашем наборе данных только одна Алиса.

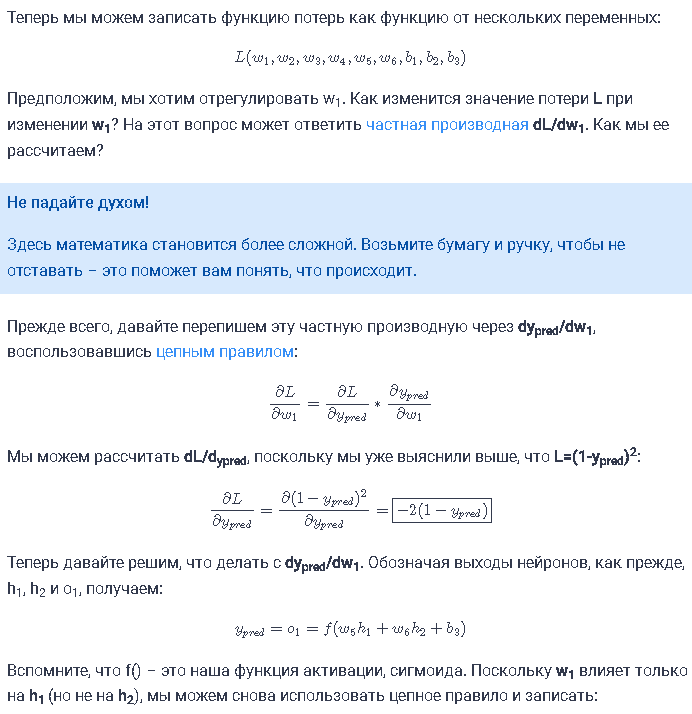
|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Вес (минус 135)** | **Рост (минус 66)** | **Пол** |
| Алиса | -2 | -1 | 1 |

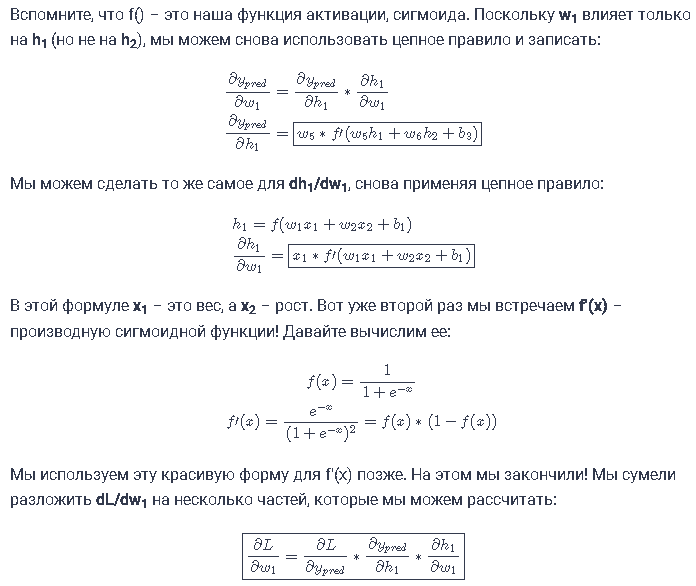
Тогда средняя квадратичная ошибка будет квадратичной ошибкой только для Алисы:



Другой метод – это рассматривать функцию потерь как функцию от весов и порогов. Давайте отметим все веса и пороги нашей нейронной сети:





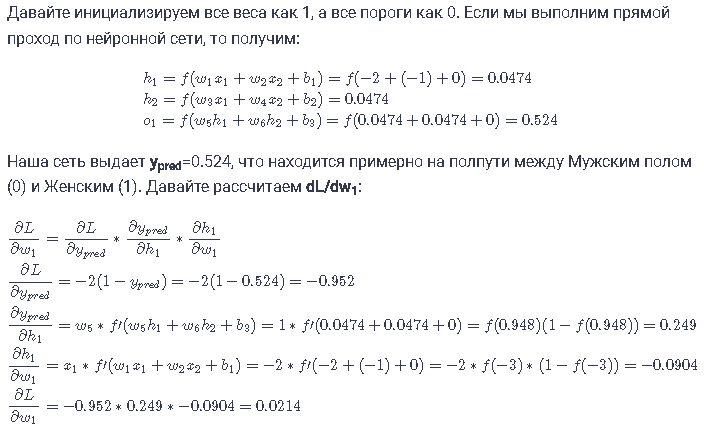
Такой метод расчета частных производных "от конца к началу" называется [**методом обратного распространения (backpropagation)**](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B8).

Уффф. Здесь было очень много символов, так что не страшно, если вы пока не все понимаете. Давайте покажем, как это работает, на практическом примере!

Пример. Считаем частную производную

Мы по-прежнему считаем, что наш набор данных состоит из одной Алисы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Вес (минус 135)** | **Рост (минус 66)** | **Пол** |
| Алиса | -2 | -1 | 1 |

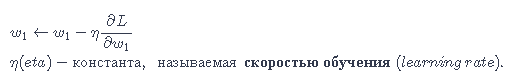
Напоминаем:

Ранее мы получили формулу для производной сигмоиды **f'(x)=f(x)(1-f(x))**

Вот и все! Результат говорит нам, что при увеличении **w1,** функция ошибки *чуть-чуть* повышается.

Обучение: стохастический градиентный спуск

Теперь у нас есть все нужные инструменты для обучения нейронной сети! Мы используем алгоритм оптимизации под названием [**стохастический градиентный спуск (stochastic gradient descent)**](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D1%85%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA), который определит, как мы будем изменять наши веса и пороги для минимизации потерь. Фактически, он заключается в следующей формуле обновления:

Скорость обучения определяет, как быстро наша сеть учится. Все, что мы делаем – это вычитаем **eta\*dL/dw1** из **w1**:

* Если **dL/dw1** положительна, **w1** уменьшится, что уменьшит **L**.
* Если **dL/dw1** отрицательна, **w1** увеличится, что также уменьшит **L**.

Если мы сделаем то же самое для каждого веса и порога в сети, потери будут постепенно уменьшаться, и наша сеть будет выдавать более точные результаты.

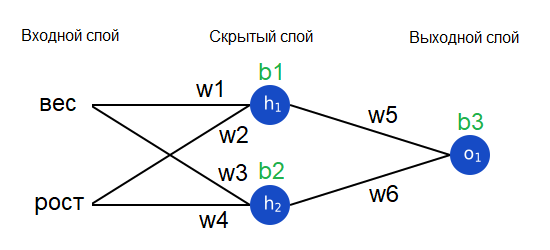
Процесс обучения сети будет выглядеть примерно так:

1. Выбираем **одно**наблюдение из набора данных. Именно то, что мы работаем только с одним наблюдением, делает наш градиентный спуск *стохастическим*.
2. Считаем все частные производные функции потерь по всем весам и порогам (**dL/dw1**, **dL/dw2**и т.д.)
3. Используем формулу обновления, чтобы обновить значения каждого веса и порога.
4. Снова переходим к шагу 1.

Пишем код всей нейронной сети

*Наконец* настало время реализовать всю нейронную сеть.

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Вес (минус 135)** | **Рост (минус 66)** | **Пол** |
| Алиса | -2 | -1 | 1 |
| Боб | 25 | 6 | 0 |
| Чарли | 17 | 4 | 0 |
| Диана | -15 | -6 |  |



import numpy as np

def sigmoid(x):

# Сигмоидная функция активации: f(x) = 1 / (1 + e^(-x))

return 1 / (1 + np.exp(-x))

def deriv\_sigmoid(x):

# Производная сигмоиды: f'(x) = f(x) \* (1 - f(x))

fx = sigmoid(x)

return fx \* (1 - fx)

def mse\_loss(y\_true, y\_pred):

# y\_true и y\_pred - массивы numpy одинаковой длины.

return ((y\_true - y\_pred) \*\* 2).mean()

class OurNeuralNetwork:

'''

Нейронная сеть с:

- 2 входами

- скрытым слоем с 2 нейронами (h1, h2)

- выходной слой с 1 нейроном (o1)

\*\*\* DISCLAIMER \*\*\*:

Следующий код простой и обучающий, но НЕ оптимальный.

Код реальных нейронных сетей совсем на него не похож. НЕ копируйте его!

Изучайте и запускайте его, чтобы понять, как работает эта нейронная сеть.

'''

def \_\_init\_\_(self):

# Веса

self.w1 = np.random.normal()

self.w2 = np.random.normal()

self.w3 = np.random.normal()

self.w4 = np.random.normal()

self.w5 = np.random.normal()

self.w6 = np.random.normal()

# Пороги

self.b1 = np.random.normal()

self.b2 = np.random.normal()

self.b3 = np.random.normal()

def feedforward(self, x):

# x is a numpy array with 2 elements.

h1 = sigmoid(self.w1 \* x[0] + self.w2 \* x[1] + self.b1)

h2 = sigmoid(self.w3 \* x[0] + self.w4 \* x[1] + self.b2)

o1 = sigmoid(self.w5 \* h1 + self.w6 \* h2 + self.b3)

return o1

def train(self, data, all\_y\_trues):

'''

- data - массив numpy (n x 2) numpy, n = к-во наблюдений в наборе.

- all\_y\_trues - массив numpy с n элементами.

Элементы all\_y\_trues соответствуют наблюдениям в data.

'''

learn\_rate = 0.1

epochs = 1000 # сколько раз пройти по всему набору данных

for epoch in range(epochs):

for x, y\_true in zip(data, all\_y\_trues):

# --- Прямой проход (эти значения нам понадобятся позже)

sum\_h1 = self.w1 \* x[0] + self.w2 \* x[1] + self.b1

h1 = sigmoid(sum\_h1)

sum\_h2 = self.w3 \* x[0] + self.w4 \* x[1] + self.b2

h2 = sigmoid(sum\_h2)

sum\_o1 = self.w5 \* h1 + self.w6 \* h2 + self.b3

o1 = sigmoid(sum\_o1)

y\_pred = o1

# --- Считаем частные производные.

# --- Имена: d\_L\_d\_w1 = "частная производная L по w1"

d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)

# Нейрон o1

d\_ypred\_d\_w5 = h1 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_w6 = h2 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_b3 = deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_h1 = self.w5 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_h2 = self.w6 \* deriv\_sigmoid(sum\_o1)

# Нейрон h1

d\_h1\_d\_w1 = x[0] \* deriv\_sigmoid(sum\_h1)

d\_h1\_d\_w2 = x[1] \* deriv\_sigmoid(sum\_h1)

d\_h1\_d\_b1 = deriv\_sigmoid(sum\_h1)

# Нейрон h2

d\_h2\_d\_w3 = x[0] \* deriv\_sigmoid(sum\_h2)

d\_h2\_d\_w4 = x[1] \* deriv\_sigmoid(sum\_h2)

d\_h2\_d\_b2 = deriv\_sigmoid(sum\_h2)

# --- Обновляем веса и пороги

# Нейрон h1

self.w1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w1

self.w2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w2

self.b1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_b1

# Нейрон h2

self.w3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w3

self.w4 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w4

self.b2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_b2

# Нейрон o1

self.w5 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w5

self.w6 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w6

self.b3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_b3

# --- Считаем полные потери в конце каждой эпохи

if epoch % 10 == 0:

y\_preds = np.apply\_along\_axis(self.feedforward, 1, data)

loss = mse\_loss(all\_y\_trues, y\_preds)

print("Epoch %d loss: %.3f" % (epoch, loss))

# Определим набор данных

data = np.array([

[-2, -1], # Алиса

[25, 6], # Боб

[17, 4], # Чарли

[-15, -6], # Диана

])

all\_y\_trues = np.array([

1, # Алиса

0, # Боб

0, # Чарли

1, # Диана

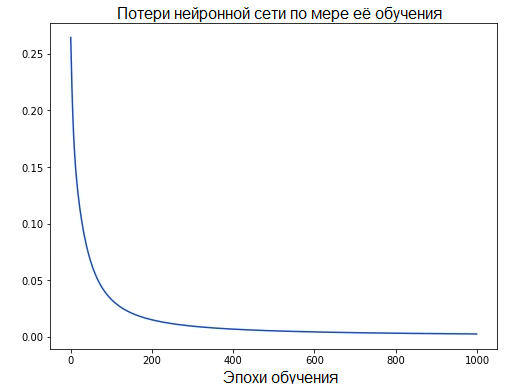
])

# Обучаем нашу нейронную сеть!

network = OurNeuralNetwork()

network.train(data, all\_y\_trues)

По мере обучения сети ее потери постепенно уменьшаются:



Теперь мы можем использовать нашу сеть для предсказания пола:

# Делаем пару предсказаний

emily = np.array([-7, -3]) # 128 фунтов (52.35 кг), 63 дюйма (160 см)

frank = np.array([20, 2]) # 155 pounds (63.4 кг), 68 inches (173 см)

print("Эмили: %.3f" % network.feedforward(emily)) # 0.951 - Ж

print("Фрэнк: %.3f" % network.feedforward(frank)) # 0.039 - М

Что теперь?

Вы сделали это! Давайте перечислим все, что мы с вами сделали:

* Определили **нейроны**, составные элементы нейронных сетей.
* Использовали **сигмоидную функцию активации** для наших нейронов.
* Увидели, что нейронные сети – это всего лишь несколько нейронов, соединенных друг с другом.
* Создали набор данных, в котором Вес и Рост были входными данными (или **признаками**), а Пол – выходным (или **меткой**).
* Узнали о **функции потерь** и **средней квадратичной ошибке (MSE)**.
* Поняли, что обучение нейронной сети – это всего лишь минимизация ее потерь.
* Использовали **метод обратного распространения (backpropagation)** для расчета частных производных.
* Использовали **стохастический градиентный спуск (SGD)** для обучения нашей сети.

Перед вами – множество путей, на которых вас ждет масса нового и интересного:

* Экспериментируйте с большими и лучшими нейронными сетями, используя подходящие библиотеки вроде [Tensorflow](https://www.tensorflow.org/" \t "_blank), [Keras](https://keras.io/" \t "_blank)и [PyTorch](https://pytorch.org/" \t "_blank).
* [Создайте свою первую нейронную сеть с помощью Keras](https://victorzhou.com/blog/keras-neural-network-tutorial/).
* Прочитайте остальные статьи из серии "[Нейронные сети с нуля](https://victorzhou.com/series/neural-networks-from-scratch/)".
* Исследуйте [другие функции активации](https://keras.io/activations/), кроме сигмоиды, например, [Softmax](https://victorzhou.com/blog/softmax/" \t "_blank).
* Исследуйте [другие оптимизаторы](https://keras.io/optimizers/), кроме стохастического градиентного спуска.