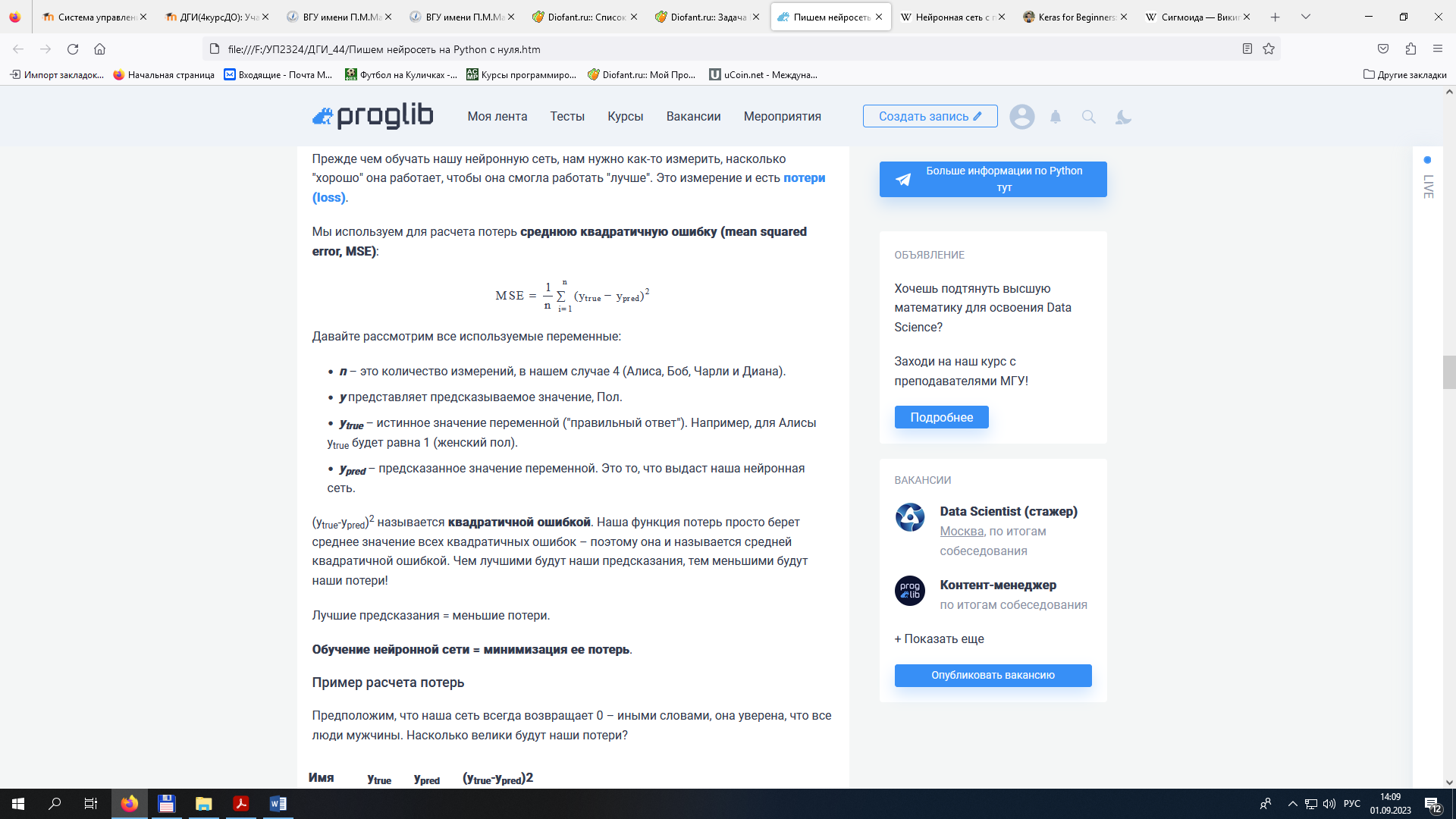
**ЛАБОРАТОРНАЯ РАБОТА** ОБУЧЕНИЕ НЕЙРОСЕТИ

Теоретические сведения

Прежде чем обучать нейронную сеть, нужно измерить, насколько "хорошо" она работает, чтобы она смогла работать "лучше". Это измерение и есть потери (loss - функция потерь — функция, которая в теории статистических решений характеризует потери при неправильном принятии решений).

Используем для расчета потерь среднюю квадратичную ошибку (mean squared error, MSE):



где: n – это количество измерений.

y представляет предсказываемое значение.

ytrue – истинное значение переменной ("правильный ответ").

ypred – предсказанное значение переменной. Это то, что выдаст наша нейронная сеть.

(ytrue-ypred)2 называется квадратичной ошибкой. Наша функция потерь просто берет среднее значение всех квадратичных ошибок – поэтому она и называется средней квадратичной ошибкой. Чем лучшими будут наши предсказания, тем меньшими будут наши потери

Лучшие предсказания = меньшие потери.

Обучение нейронной сети = минимизация ее потерь.

Порядок выполнения работы

Обучим нейронную сеть предсказывать пол человека по его росту и весу.

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 |

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 |

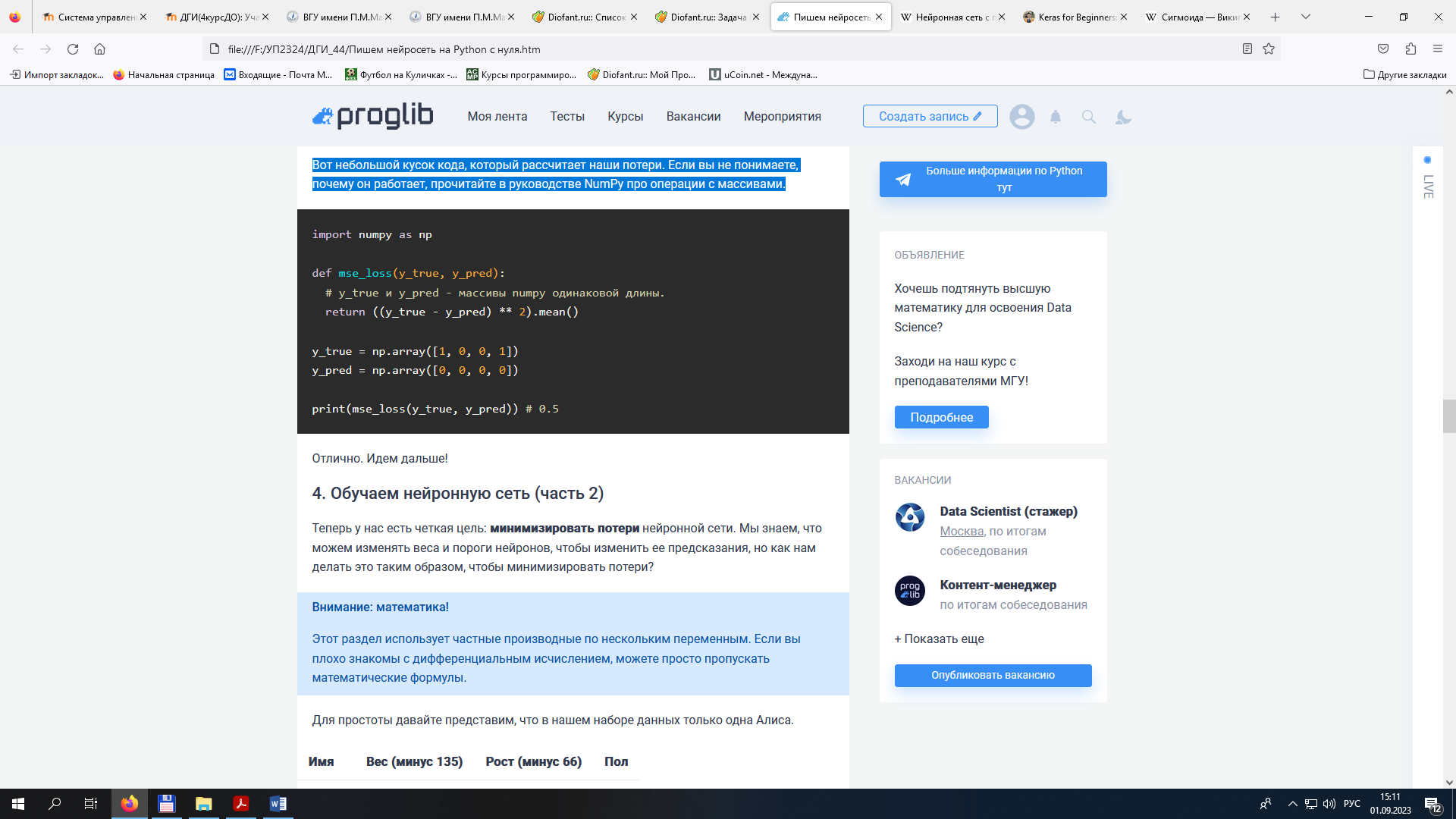
Обычно сдвигают на среднее значение.

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 |

MSE=0.5

Вот часть кода для расчета потерь(почему он работает, прочитайте в [руководстве NumPy](https://numpy.org/doc/stable/user/quickstart.html#basic-operations) про операции с массивами).



Цель: минимизировать потери нейронной сети. Можем изменять веса и пороги нейронов, чтобы изменить ее предсказания, но делать надо это таким образом, чтобы минимизировать потери.

Это позволяют сделать частные производные по нескольким переменным.

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

Имя Вес (минус 135) Рост (минус 66) Пол

Алиса -2 -1 1

Средняя квадратичная ошибка будет квадратичной ошибкой только для Алисы.

Другой метод – это рассматривать функцию потерь как функцию от весов и порогов.

Опуская вывод формул и взятие частных производных имеем:

Используем алгоритм оптимизации под названием стохастический градиентный спуск (stochastic gradient descent), который определит, как мы будем изменять наши веса и пороги для минимизации потерь.

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:

Выбираем одно наблюдение из набора данных. Именно то, что мы работаем только с одним наблюдением, делает наш градиентный спуск стохастическим.

Считаем все частные производные функции потерь по всем весам и порогам (dL/dw1, dL/dw2 и т.д.)

Используем формулу обновления, чтобы обновить значения каждого веса и порога.

Снова переходим к шагу 1.

Имя Вес (минус 135) Рост (минус 66) Пол

Алиса -2 -1 1

Боб 25 6 0

Чарли 17 4 0

Диана -15 -6

Код всей сети:

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)

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