Лабораторная работа 5

Простой пример работы с нейронами в Python

Предположим, у нас есть **нейрон с двумя входами**, который использует функцию **активации сигмоида** и имеет следующие параметры:

$$w = [0, 1]$$

$$b = 4$$

`w = [0,1]` — это просто один из способов написания $w_1 = 0$, $w_2 = 1$ в векторной форме. Присвоим нейрону вход со значением `x = [2, 3]`. Для более компактного представления будет использовано скалярное произведение.

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

С учетом, что вход был `x = [2, 3]`, вывод будет равен `0.999`. Вот и все. Такой процесс **передачи входных данных** для получения вывода называется прямым распространением, или feedforward.

Создание нейрона с нуля в Python

Практический Python 3 для начинающих

Приступим к имплементации нейрона. Для этого потребуется использовать NumPy. Это мощная вычислительная библиотека Python, которая задействует математические операции:

Python

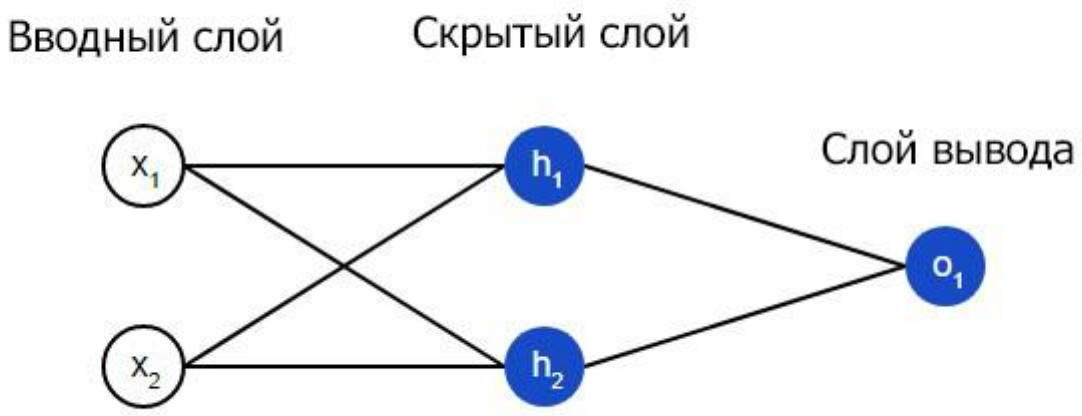
```
1  import numpy as np
2
3
4  def sigmoid(x):
```

```
    # 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):
        # Вводные данные о весе, добавление смещения
        # и последующее использование функции 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 числа? Это тот же пример, который рассматривался ранее. Ответ полученный на этот раз также равен `0.999`.

Пример сбор нейронов в нейросеть

Нейронная сеть по сути представляет собой группу **связанных между собой нейронов**. Простая нейронная сеть выглядит следующим образом:



На вводном слое сети два входа – x_1 и x_2. На скрытом слое два нейтрона — h_1 и h_2. На слое вывода находится один нейрон – o_1. Обратите внимание на то, что входные

данные для o_1 являются результатами вывода h_1 и h_2. Таким образом и **строится нейросеть**.

Скрытым слоем называется любой слой между вводным слоем и слоем вывода, что являются первым и последним слоями соответственно. Скрытых слоев может быть несколько.

Пример прямого распространения FeedForward

Давайте используем продемонстрированную выше сеть и представим, что все **нейроны** имеют одинаковый вес `w = [0, 1]`, одинаковое смещение `b = 0` и ту же самую функцию активации сигмоида. Пусть h_1, h_2 и o_1 сами отметят результаты вывода представленных ими нейронов.

Что случится, если в качестве ввода будет использовано значение `x = [2, 3]`?

$$
\begin{aligned}
h_1 = h_2 &= f(w \cdot x + b) \\
&= f((0 * 2) + (1 * 3) + 0) \\
&= f(3) \\
&= 0.9526
\end{aligned}
$$

$$
\begin{aligned}
o_1 &= f(w \cdot [h_1, h_2] + b) \\
&= f((0 * h_1) + (1 * h_2) + 0) \\
&= f(0.9526) \\
&= \boxed{0.7216}
\end{aligned}
$$

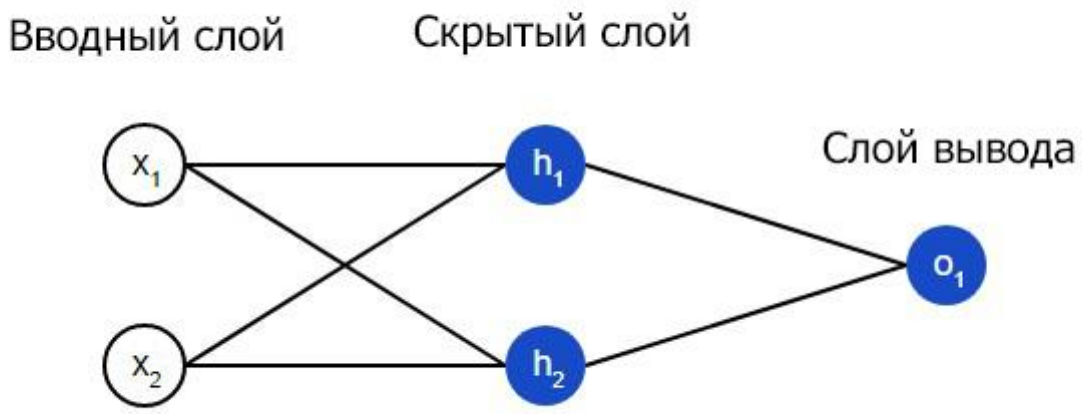
Результат вывода **нейронной сети** для входного значения `x = [2, 3]` составляет `0.7216`.

Нейронная сеть может иметь любое количество слоев с любым количеством нейронов в этих слоях.

Суть остается той же: нужно направить входные данные через нейроны в сеть для получения в итоге выходных данных. Для простоты далее в данной статье будет создан код сети, упомянутая выше.

Создание нейронной сети с прямым распространением FeedForward

Далее будет показано, как реализовать прямое распространение **feedforward** в отношении нейронной сети. В качестве опорной точки будет использована следующая **схема нейронной сети**:



Python

```
import numpy as np

# ... Здесь код из предыдущего раздела


class OurNeuralNetwork:
    """
    Нейронная сеть, у которой:
        - 2 входа
        - 1 скрытый слой с двумя нейронами (h1, h2)
        - слой вывода с одним нейроном (o1)
    У каждого нейрона одинаковые вес и смещение:
        - w = [0, 1]
        - b = 0
    """
    def __init__(self):
        weights = np.array([0, 1])
        bias = 0

        # 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)

        # Вводы для o1 являются выводами h1 и h2
        out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

        return out_o1

```

```
34
35 network = OurNeuralNetwork()
36 x = np.array([2, 3])
37 print(network.feedforward(x))  # 0.7216325609518421
```

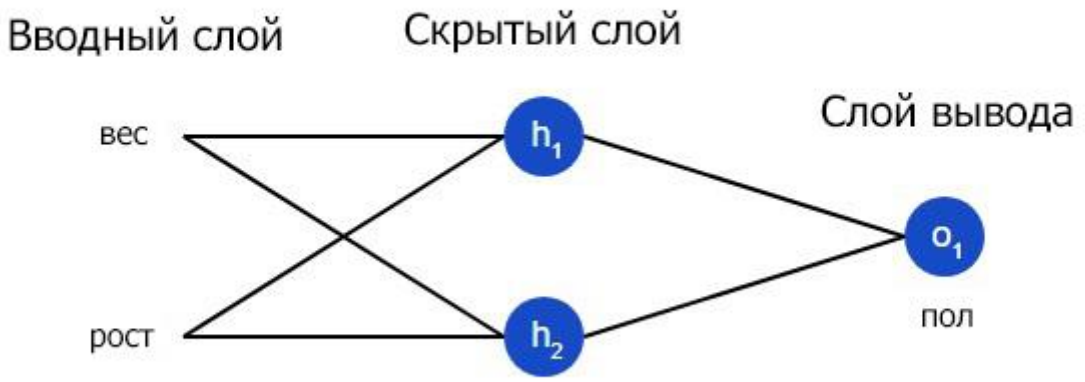
Мы вновь получили `0.7216`. Похоже, все работает.

Пример тренировки нейронной сети — минимизация потерь, Часть 1

Предположим, у нас есть следующие параметры:

Имя/Name	**Вес/Weight (фунты)**	**Рост/Height (дюймы)**	**Пол/Gender**
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Давайте **натренируем нейронную сеть** таким образом, чтобы она предсказывала пол заданного человека в зависимости от его веса и роста.



Мужчины `Male` будут представлены как `0`, а женщины `Female` как `1`. Для простоты представления данные также будут несколько смещены.

Имя/Name	**Вес/Weight (минус 135)**	**Рост/Height (минус 66)**	**Пол/Gender**
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

Для оптимизации здесь произведены произвольные смещения `135` и `66`. Однако, обычно для смещения выбираются средние показатели.

Потери

Перед **тренировкой нейронной сети** потребуется выбрать способ оценки того, насколько хорошо сеть справляется с задачами. Это необходимо для ее последующих попыток выполнять поставленную задачу лучше. Таков принцип потери.

В данном случае будет использоваться **среднеквадратическая ошибка (MSE)** потери:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Давайте разберемся:

- `n` – число рассматриваемых объектов, которое в данном случае равно 4. Это `Alice`, `Bob`, `Charlie` и `Diana`;
- `y` – переменные, которые будут предсказаны. В данном случае это пол человека;
- y_{true} – истинное значение переменной, то есть так называемый правильный ответ. Например, для `Alice` значение y_{true} будет `1`, то есть `Female`;
- y_{pred} – предполагаемое значение переменной. Это результат вывода сети.

$(y_{true} - y_{pred})^2$ называют **квадратичной ошибкой (MSE)**. Здесь функция потери просто берет среднее значение по всем квадратичным ошибкам. Отсюда и название ошибки. Чем лучше предсказания, тем ниже потери.

Лучшие предсказания = Меньшие потери.

Тренировка нейронной сети = стремление к минимизации ее потерь.

Пример подсчета потерь в тренировки нейронной сети

Скажем, наша сеть всегда выдает `0`. Другими словами, она уверена, что все люди — Мужчины. Какой будет потеря?

Имя/Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0

Charlie	0	0	0
Diana	1	0	1

$$\text{MSE} = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

Python код среднеквадратической ошибки (MSE)

Ниже представлен код для подсчета потерь:

Python

```
import numpy as np


def mse_loss(y_true, y_pred):
    # y_true и y_pred являются массивами numpy с одинаковой длиной
    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
```

При возникновении сложностей с пониманием работы кода стоит ознакомиться с quickstart в NumPy для **операций с массивами**.

Тренировка нейронной сети — многовариантные исчисления, Часть 2

Текущая цель понятна – это **минимизация потерь нейронной сети**. Теперь стало ясно, что повлиять на предсказания сети можно при помощи изменения ее веса и смещения. Однако, как **минимизировать потери**?

В этом разделе будут затронуты **многовариантные исчисления**. Если вы не знакомы с данной темой, фрагменты с математическими вычислениями можно пропускать.

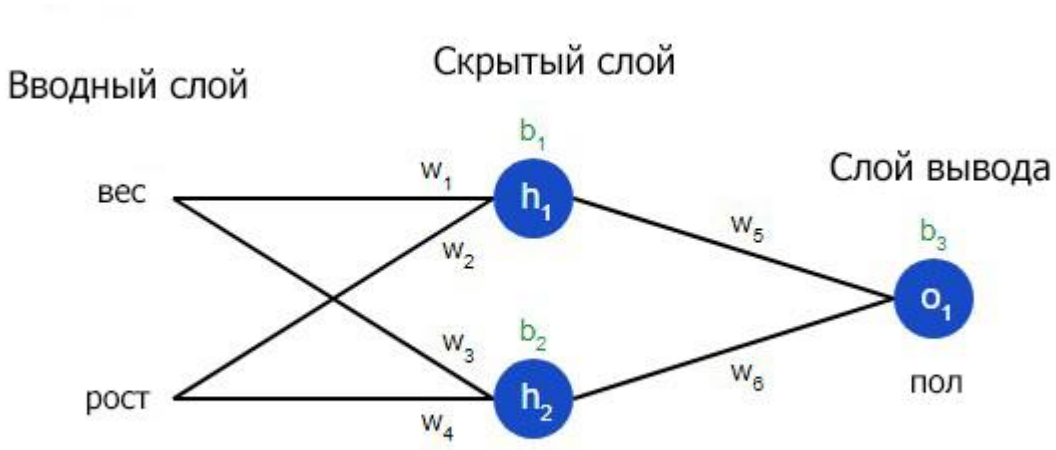
Для простоты давайте представим, что в наборе данных рассматривается только `Alice`:

Имя/Name	Вес/Weight (минус 135)	Рост/Height (минус 66)	Пол/Gender
Alice	-2	-1	1

Затем потеря среднеквадратической ошибки будет просто квадратической ошибкой для `Alice`:

$$\begin{aligned} \text{MSE} &= \frac{1}{1}\sum_{i=1}^{1}(y_{true} - y_{pred})^2 \\ &= (y_{true} - y_{pred})^2 \\ &= (1 - y_{pred})^2 \end{aligned}$$

Еще один способ понимания потери – представление ее как функции веса и смещения. Давайте обозначим каждый вес и смещение в рассматриваемой сети:



Затем можно прописать потерю как многовариантную функцию:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Представим, что нам нужно немного отредактировать w_1. В таком случае, как изменится потеря L после внесения поправок в w_1?

На этот вопрос может ответить частная производная $\frac{\partial L}{\partial w_1}$. Как же ее вычислить?

Здесь математические вычисления будут намного сложнее. С первой попытки вникнуть будет непросто, но отчаиваться не стоит. Возьмите блокнот и ручку – лучше делать заметки, они помогут в будущем.

Для начала, давайте перепишем частную производную в контексте $\frac{\partial y_{pred}}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}$$

Данные вычисления возможны благодаря дифференцированию сложной функции.

Подсчитать $\frac{\partial L}{\partial y_{pred}}$ можно благодаря вычисленной выше $L = (1 - y_{pred})^2$:

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = \boxed{-2(1 - y_{pred})}$$

Теперь, давайте определим, что делать с $\frac{\partial y_{pred}}{\partial w_1}$. Как и ранее, позволим h_1, h_2, o_1 стать результатами вывода нейронов, которые они представляют. Дальнейшие вычисления:

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

Как было указано ранее, здесь f является функцией активации сигмоида.

Так как w_1 влияет только на h_1, а не на h_2, можно записать:

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

Использование дифференцирования сложной функции.

Те же самые действия проводятся для $\frac{\partial h_1}{\partial w_1}$:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$$

$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b_1)}$$

Еще одно использование дифференцирования сложной функции.

В данном случае x_1 — вес, а x_2 — рост. Здесь `f'(x)` как производная функции сигмоида встречается во второй раз. Попробуем вывести ее:

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

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) * (1 - f(x))$$

Функция `f'(x)` в таком виде будет использована несколько позже.

Вот и все. Теперь $\frac{\partial L}{\partial w_1}$ разбита на несколько частей, которые будут оптимальны для подсчета:

$$\boxed{\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}$$

Эта система подсчета частных производных при работе в обратном порядке известна, как **метод обратного распространения ошибки**, или **backprop**.

У нас накопилось довольно много формул, в которых легко запутаться. Для лучшего понимания принципа их работы рассмотрим следующий пример.

Пример подсчета частных производных

В данном примере также будет задействована только `Alice`:

Имя/Name	Вес/Weight (минус 135)	Рост/Height (минус 66)	Пол/Gender
Alice	-2	-1	1

Здесь вес будет представлен как `1`, а смещение как `0`. Если выполним прямое распространение (feedforward) через сеть, получим:

$$\begin{aligned} h_1 &= f(w_1 x_1 + w_2 x_2 + b_1) \\ &= f(-2 + -1 + 0) \\ &= 0.0474 \end{aligned}$$

$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$

$$\begin{aligned} o_1 &= f(w_5 h_1 + w_6 h_2 + b_3) \\ &= f(0.0474 + 0.0474 + 0) \\ &= 0.524 \end{aligned}$$

Выдачи **нейронной сети** $y_{pred} = 0.524$. Это дает нам слабое представление о том, рассматривается мужчина `Male (0)`, или женщина `Female (1)`. Давайте подсчитаем $\frac{\partial L}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\begin{aligned}\frac{\partial L}{\partial y_{pred}} &= -2(1 - y_{pred}) \\ &= -2(1 - 0.524) \\ &= -0.952\end{aligned}$$

$$\begin{aligned}\frac{\partial y_{pred}}{\partial h_1} &= w_5 * f'(w_5 h_1 + w_6 h_2 + b_3) \\ &= 1 * f'(0.0474 + 0.0474 + 0) \\ &= f(0.0948) * (1 - f(0.0948)) \\ &= 0.249\end{aligned}$$

$$\begin{aligned}\frac{\partial h_1}{\partial w_1} &= x_1 * f'(w_1 x_1 + w_2 x_2 + b_1) \\ &= -2 * f'(-2 + -1 + 0) \\ &= -2 * f(-3) * (1 - f(-3)) \\ &= -0.0904\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial w_1} &= -0.952 * 0.249 * -0.0904 \\ &= \boxed{0.0214}\end{aligned}$$

Напоминание: мы вывели `f '(x) = f (x) * (1 - f (x))` ранее для нашей функции **активации сигмоида**.

У нас получилось! Результат говорит о том, что если мы собираемся увеличить w_1, L немного увеличивается в результате.

Тренировка нейронной сети: Стохастический градиентный спуск

У нас есть все необходимые инструменты для **тренировки нейронной сети**. Мы используем алгоритм оптимизации под названием стохастический градиентный спуск (SGD), который говорит нам, как именно поменять вес и смещения для **минимизации потерь**. По сути, это отражается в следующем уравнении:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η является константой под названием **оценка обучения**, что контролирует скорость обучения. Все что мы делаем, так это вычитаем $\frac{\partial L}{\partial w_1}$ из w_1:

- Если $\frac{\partial L}{\partial w_1}$ положительная, w_1 уменьшится, что приведет к уменьшению L.
- Если $\frac{\partial L}{\partial w_1}$ отрицательная, w_1 увеличится, что приведет к уменьшению L.

Если мы применим это на каждый вес и смещение в сети, **потеря будет постепенно снижаться**, а показатели сети сильно улучшатся.

Наш процесс тренировки будет выглядеть следующим образом:

1. Выбираем один пункт из нашего набора данных. Это то, что делает его **стохастическим градиентным спуском**. Мы обрабатываем только один пункт за раз;
2. Подсчитываем все частные производные потери по весу или смещению. Это может быть $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$ и так далее;
3. Используем уравнение обновления для обновления каждого веса и смещения;
4. Возвращаемся к первому пункту.

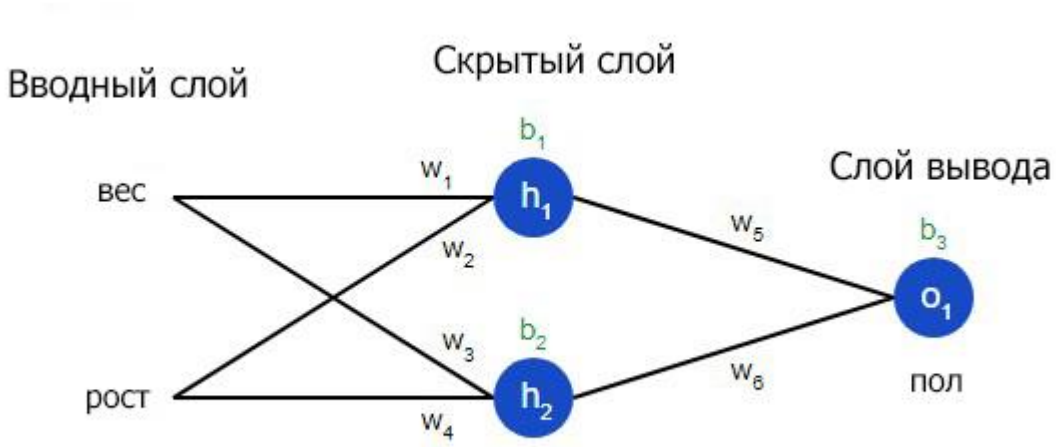
Давайте посмотрим, как это работает на практике.

Создание нейронной сети с нуля на Python

Наконец, мы реализуем **готовую нейронную сеть**:

Имя/Name	**Вес/Weight (минус 135)**	**Рост/Height (минус 66)**	**Пол/Gender**
Alice	-2	-1	1
Bob	25	6	0

Charlie	17	4	0
Diana	-15	-6	1



Python

```
import numpy as np


def sigmoid(x):
    # Функция активации sigmoid:: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))


def deriv_sigmoid(x):
    # Производная от sigmoid: 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 являются массивами numpy с одинаковой длиной
    return ((y_true - y_pred) ** 2).mean()


class OurNeuralNetwork:
    """
    Нейронная сеть, у которой:
        - 2 входа
        - скрытый слой с двумя нейронами (h1, h2)
```

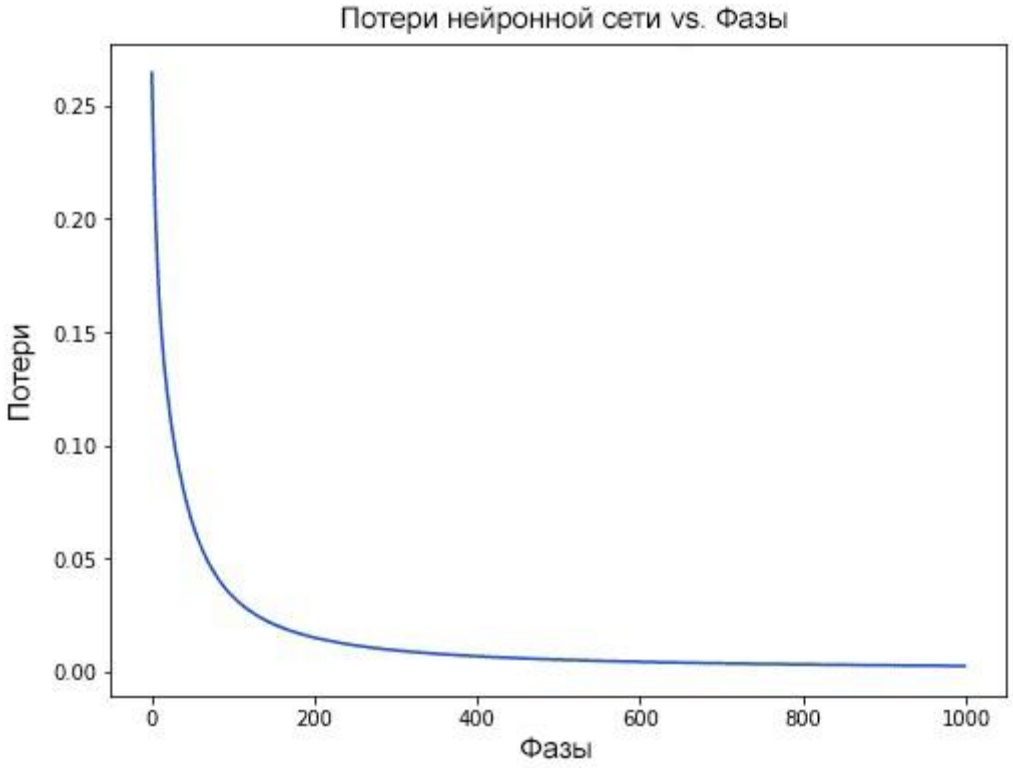
```
    - слой вывода с одним нейроном (o1)

*** ВАЖНО ***:
Код ниже написан как простой, образовательный. НЕ оптимальный.
Настоящий код нейронной сети выглядит не так. НЕ ИСПОЛЬЗУЙТЕ этот код.
Вместо этого, прочитайте/запустите его, чтобы понять, как работает эта сеть.
"""
def __init__(self):
    # Вес
    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()

    # Смещения
    self.b1 = np.random.normal()
    self.b2 = np.random.normal()
    self.b3 = np.random.normal()

def feedforward(self, x):
    # x является массивом numpy с двумя элементами
    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 is a (n x 2) numpy array, n = # of samples in the dataset.
    - all_y_trues is a numpy array with n elements.
      Elements in all_y_trues correspond to those in data.
    """
    learn_rate = 0.1
    epochs = 1000 # количество циклов во всём наборе данных

    for epoch in range(epochs):
        for x, y_true in zip(data, all_y_trues):
            # --- Выполняем обратную связь (нам понадобятся эти значения в
дальнейшем)
            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

            # --- Подсчет частных производных
```

```
            # --- Наименование: d_L_d_w1 представляет "частично L / частично 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 вес и смещения
            # 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

        # --- Подсчитываем общую потерю в конце каждой фазы
        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))


# Определение набора данных
data = np.array([
    [-2, -1],    # Alice
    [25, 6],     # Bob
    [17, 4],     # Charlie
    [-15, -6], # Diana
])

```

```
all_y_trues = np.array([
    1, # Alice
    0, # Bob
    0, # Charlie
    1, # Diana
])

# Тренируем нашу нейронную сеть!
network = OurNeuralNetwork()
network.train(data, all_y_trues)
```

.

Наши потери постоянно уменьшаются по мере того, как учится нейронная сеть:



Теперь мы можем использовать нейронную сеть для предсказания полов:

Python

```
# Делаем предсказания
emily = np.array([-7, -3])  # 128 фунтов, 63 дюйма
frank = np.array([20, 2])  # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily))  # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank))  # 0.039 - M
```

Задание.

1. Реализовать на языке Python нейронную сеть предсказывания полов на основе роста и веса, обучить ее, вычислить среднеквадратичную ошибку, а также проверить работоспособность сети на тестовой выборке https://www.kaggle.com/mustafaali96/weight-height

В качестве среды разработки для языка Python можно использовать любую IDE, доступимо пользоваться онлайновыми средами (например, jupiter).

Для сдачи лабораторной работы необходим исходный файл нейронной сети (или весь проект в виде zip архива) и текстовый отчет, содержащий программный код, а также скриншоты предсказаний на основе тестовой выборки.

2. Используя датасет с Kaggle, ссылка на который дана выше, построить систему предсказания пола на основе роста и веса в Orange. Настроить параметры с целью максимизации точности предсказаний. Добавить в отчет скриншоты с результатами.

Сравнить полученные результаты с нейронной сетью, написанной в соответствии с п.1 задания. Выводы по результатам сравнения привести в отчете.