Лабораторная работа №3
СОЗДАНИЕ НЕЙРОННОЙ СЕТИ С ГЛАДКОЙ ФУНКЦИЕЙ АКТИВАЦИИ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ NumPy

Цель работы:

Теперь перейдем к примеру создания нейронной сети из нескольких нейронов.

Еще раз вспомним, что собой представляют базовые компоненты нейронной сети – *нейроны*. Нейрон принимает входные данные, выполняет с ними определенные математические операции, а затем выдает результат. Нейрон с двумя входными данными выглядит следующим образом:

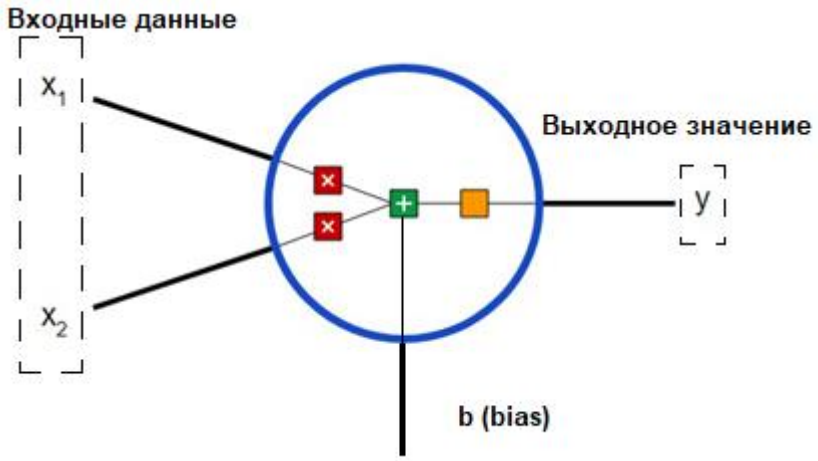


Рисунок 1 – Схема нейрона с двумя входами и смещением (bias)

Здесь происходят три этапа. Во-первых, каждый вход умножается на вес (на схеме обозначены красными квадратами):

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

Затем все взвешенные входы складываются вместе со смещением `b` (на схеме обозначен зеленым квадратом**)**:

$$(x_1 * w_1) + (x_2 * w_2) + b$$

Наконец, сумма передается через функцию активации (на схеме обозначена оранжевым квадратом):

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Для чего теперь в нейрон введен дополнительный вход – смещение b (bias)? По сути, значение смещения позволяет передвинуть взвешенную сумму входов влево или вправо, что может иметь решающее значение для успешного обучения.

Простейший способ понять, что такое смещение: оно каким-то образом похоже на константу b линейной функции $y = ax + b$. Наличие этой константы позволяет перемещать линию вверх и вниз, чтобы лучше соответствовать прогнозу на каких-то данных. Без b линия всегда проходит через начало координат (0, 0), и можете получить плохую подгонку.

Так и смещение в нейроне. Оно позволяет разместить аргумент функции активации нейрона в том месте, где кривая сигмоиды имеет наибольший наклон. Это значит, что значения на выходе нейрона для разных входных данных будут достаточно хорошо отличаться друг от друга, а в целом – сеть будет лучше обучаться.

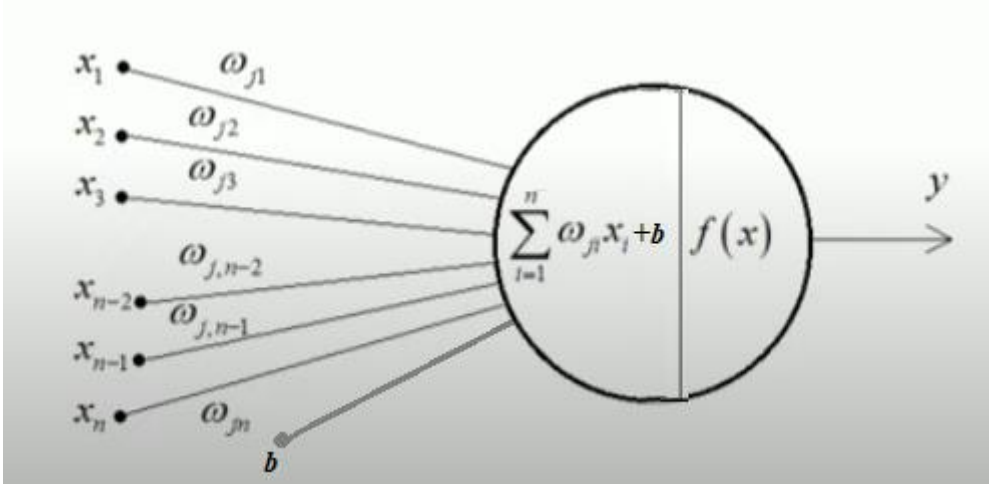


Рисунок 2 – Схема j-го нейрона с n входами и смещением

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

$$w = [0, 1]$$

$$b = 4$$

ω = [0,1] – это просто один из способов написания ω1 = 0, ω2 = 1 в векторной форме. Присвоим нейрону вход со значением x = [2, 3]. Для более компактного представления будет использовано *скалярное произведение*.

$$(w \cdot x) + b = ((w_1 * x_1) + (w_2 * x_2)) + b = 0 * 2 + 1 * 3 + 4 = 7$$

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

С учетом, что входной вектор был таким: x = [2, 3], вывод будет равен 0.999. Такой процесс передачи входных данных для получения вывода называется *прямым распространением*, или *feedforward*.

Создание класса Нейрон

По-прежнему будем использовать библиотеку numpy. Это мощная вычислительная библиотека Python, которая реализует большое число математических операций:

```
import numpy as np
 def sigmoid(x):
   # Функция активации нейрона: 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

       # Функция осуществляющая прямой проход сигнала через нейро
       # Вводные данные о весе, добавление смещения
      # и последующее использование функции активации

   def feedforward(self, inputs):
      total = np.dot(self.weights, inputs) + self.bias
      return sigmoid(total)
# Конец описания класса Neuron

 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))
```

Ответ программы

```
# 0.9990889488055994
```

Пример сбора нейронов в нейросеть

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

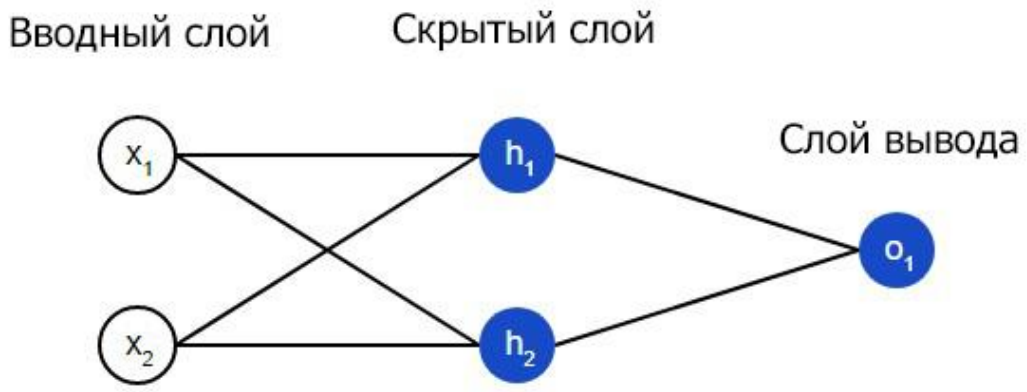


Рисунок 3 – Схема нейронной сети

Во входном слое сети два входа – x_1 и x_2. Обратите внимание, это НЕ нейроны! В следующем слое два нейрона – h_1 и h_2. На выходе схемы находится один нейрон – o_1. Здесь символы выбраны от английских слов hidden – скрытый и output – выходной. Обратите внимание на то, что входные данные для o_1 являются результатами выхода нейронов h_1 и h_2. Таким образом и строится нейросеть.

Скрытым слоем называется любой слой между первым слоем нейронов и последним слоем, которые являются входным и выходным слоями соответственно. Скрытых слоев может быть несколько.

В *полносвязной нейронной сети* каждый нейрон предыдущего слоя связан с каждым нейроном предыдущего слоя. А *прямое распространение сигнала* (feedforward) означает, что входной сигнал через все слои распространяется от входа к выходу.

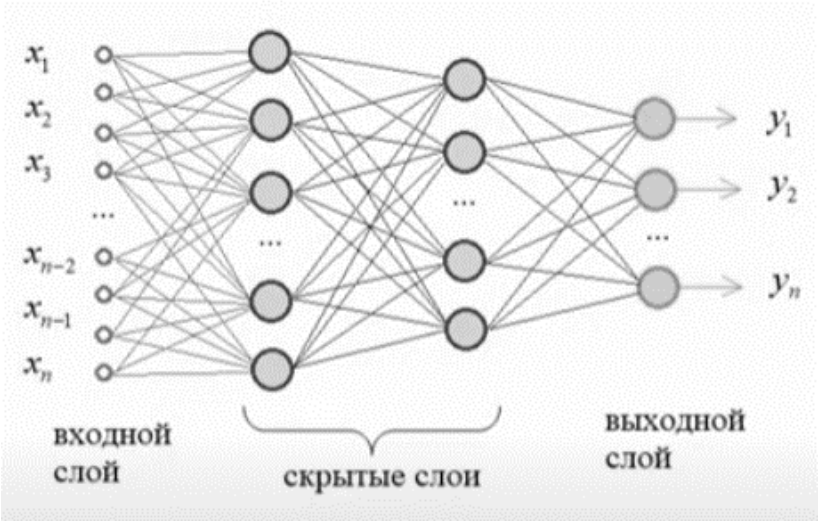


Рисунок 4 – Схема полносвязной нейронной сети с двумя скрытыми слоями

Иногда в литературе по нейронным сетям встречается такая трактовка, в которой входной слой тоже относится к слою нейронов – или это первый слой, или это первый слой скрытых нейронов, хотя это и не так. Обращайте на это внимание. Далее в представленном материале будем первый слой нейронной сети называть «входным» и помнить, что его назначение – всего-навсего представить входной сигнал и передать его на все нейроны первого скрытого слоя. Т.е. нейронов во входном слое нет.

Пример прямого распространения feedforward

Давайте используем представленную выше сеть и пусть все нейроны имеют одинаковый вес w = [0, 1], одинаковое смещение b = 0 и ту же самую функцию активации – сигмоиду.

Что будет на выходе сети, если в качестве ввода будет использовано значение вектора x = [2, 3] (т.е x+=2, x_2=3)?

$$h_1 = h_2 = f(w \cdot x + b) = f((0 * 2) + (1 * 3) + 0) = f(3) = 0.9526$$

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

Итак, результат вывода нейронной сети для входного значения x = [2, 3] составляет 0.7216. Все очень просто.

Нейронная сеть может иметь любое количество слоев с любым количеством нейронов в этих слоях. Суть *feedforward* остается той же: нужно направить входные данные через нейроны в сеть для получения в итоге выходных данных.

Далее рассмотрим, как реализовать прямое распространение *feedforward* в отношении нейронной сети на языке python. Будет использована рассмотренная ранее схема нейронной сети:

```
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)

    # Входы для нейрона o1 являются выходами нейронов 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))
```

Результат работы программы:

```
# 0.7216325609518421
```

Мы вновь получили 0.7216. Ручной и программный расчёты совпали.

Пример тренировки (обучения)нейронной сети — минимизация потерь

Предположим, у нас есть следующая обучающая выборка 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 (обучим) нейронную сеть таким образом, чтобы она предсказывала пол заданного человека в зависимости от его веса и роста.

Закодируем выходной вектор, т.к. нейронная сеть не может работать с символами (у нас – M и F). Мужчины Male будут представлены как 0, а женщины Female как 1. Входные значения будут несколько смещены. Для простоты здесь произведены произвольные смещения на 135 и 66. Хотя обычно для смещения выбираются средние показатели.

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

Ошибки (потери, loss)

Перед обучением нейронной сети потребуется выбрать способ оценки того, насколько хорошо сеть справляется с задачами. Это необходимо оценки того как сеть выполняет поставленную задачу. В данном случае будет использоваться среднеквадратическая ошибка (MSE) потери:

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

,

здесь:

n – число рассматриваемых объектов, которое в нашем случае равно 4. Это Alice, Bob, Charlie и Diana;

y – переменные, которые будут вычислены нейронной сетью (предсказаны - prediction). В нашем 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 ошибки (или функция потери) просто берет среднее значение по всем квадратичным ошибкам. Отсюда и название ошибки. Чем лучше предсказания, тем ниже ошибки. Тогда можно сказать, что:

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 ошибка?

Имя/Name	ytrue	ypred	$(ytrue — ypred)^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

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

Ниже представлен код для подсчета потерь:

```
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))              #  Ответ программы  0.5
```

Тренировка нейронной сети –back propagation (backprop)

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

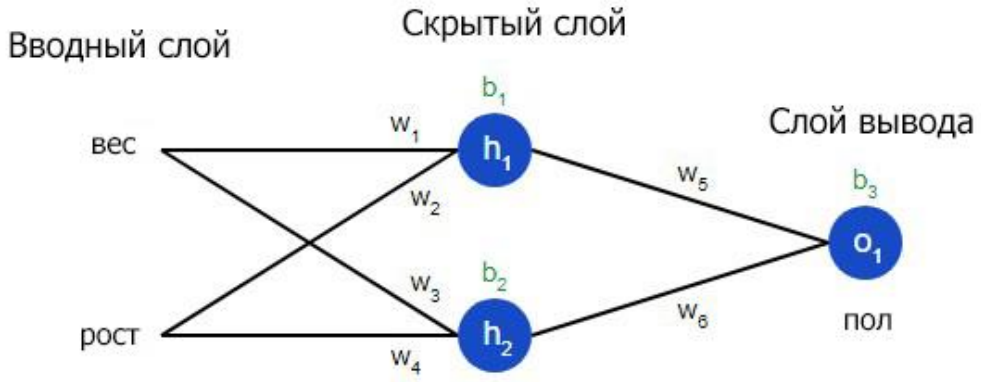
Для простоты давайте представим, что в наборе данных рассматривается только Alice:

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

Потеря среднеквадратической ошибки будет просто квадратической ошибкой для Alice:

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

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



Потерю можно прописать 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 ω_1. В таком случае, как изменится потеря L после внесения поправок в ω_1? На этот вопрос может ответить частная произво $\frac{\partial L}{\partial w_1}$

Как же ее вычислить?

Далее 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, обозначим h_1, h_2, o_1 результатами вывода соответствующих 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 сигмоиды.

Так как ω_1 влияет только на h_1, а не на h_2, можно записать, используя дифференцирования сложной функции:

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

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

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

Это еще одно использование дифференцирования сложной функции.

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

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

В данном случае x_1 – вес, а x_2 – рост. Здесь f′(x) как производная функции сигмоиды 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*.

Для лучшего понимания принципа backprop 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:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1) \ = f(-2 + -1 + 0) \ = 0.0474$$
$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$
$$o_1 = f(w_5 h_1 + w_6 h_2 + b_3) \ = f(0.0474 + 0.0474 + 0) = 0.524$$

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

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

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

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

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

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

Получен результат 0.0214!!! Результат говорит о том, что если мы собираемся увеличить ω_1, то 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:

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

где η является константой под названием *оценка обучения*, что контролирует скорость обучения. Иногда её сразу наывают скоростью обучения.

Все что мы делаем, так это вычитаем $\frac{\partial L}{\partial w_1}$ из ω_1:

Если $\frac{\partial L}{\partial w_1}$ положительная, ω_1 уменьшится, что приведет к уменьшению L.

Если $\frac{\partial L}{\partial w_1}$ отрицательная, ω_1 увеличится, что приведет к уменьшению L.

Если мы применим это для каждого веса и смещения всех нейронов сети, потеря будет постепенно снижаться, а показатели качества работы сети сильно улучшатся.

Процесс тренировки будет выглядеть следующим образом:

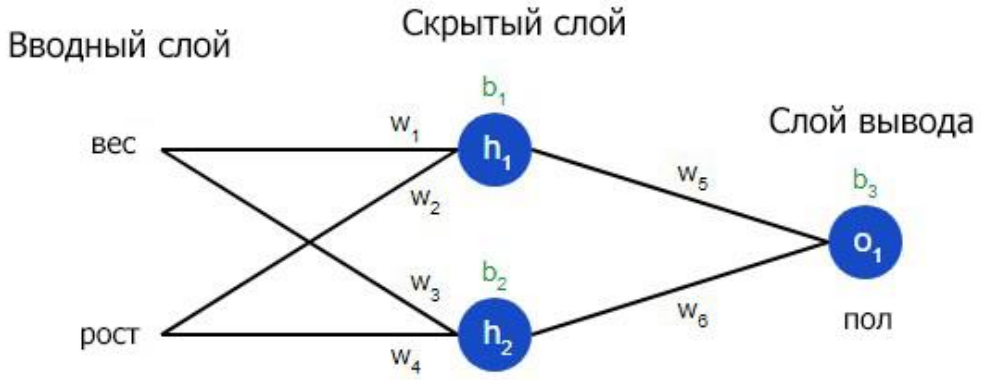
1. Выбираем один пункт из нашего набора данных. Это то, что делает его стохастическим градиентным спуском. Мы обрабатываем только один пункт за раз.
2. Подсчитываем все частные производные потери по весу или смещению. Это может быть Формулы нейронной сети, Формулы нейронной сети и так далее.
3. Используем уравнение обновления для обновления каждого веса и смещения.
4. Возвращаемся к первому пункту.

Давайте посмотрим, как это работает в программном коде.

Задача предсказания пола может быть отнесена к задаче двухклассовой (бинарной) классификации.

Создание нейронной сети на Python

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




```
  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 для практических
   задач.
    Вместо этого прочитайте и запустите его, чтобы понять, как работает эта сеть.
    """
    def __init__(self):
```

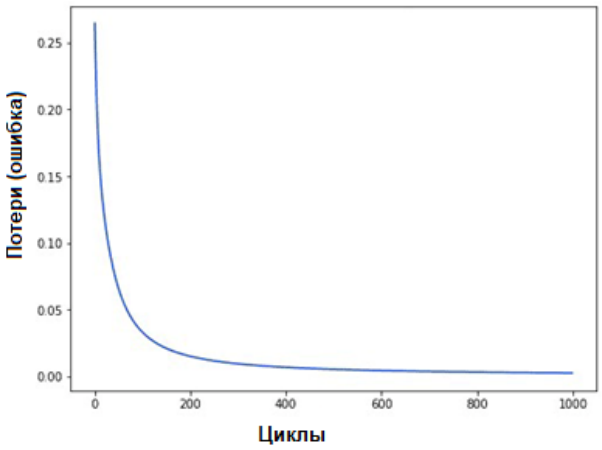
```
    # Вес
    self.w1 = np.random.normal()
    self.w2 = np.random.normal()
    self.w3 = np.random.normal()
    self.w4 = np.random.normal()
    self.w5 = np.random.normal()
    self.w6 = np.random.normal()

    # 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)
```

Можно поэкспериментировать с этим кодом самостоятельно.

Потери или ошибка сети (ошибка обучения) постоянно уменьшаются по мере того, как учится нейронная сеть:



Теперь можно использовать нейронную сеть для предсказания пола:

```
# Делаем предсказания с помощью обученной сети:
emily = np.array([-7, -3])  # 128 фунтов, 63 дюйма, не забываем, что вычитается 135 и 66
frank = np.array([20, 2])  # 155 фунтов, 68 дюймов
print("Emily: %.3f" % network.feedforward(emily))
print("Frank: %.3f" % network.feedforward(frank))
```

Результаты отличные:

```
# 0.951 – F
# 0.039 – M
```

Заключение

Подведем итоги:

- Вы узнали, что такое нейроны, как создать из них нейронную сеть.
- Использовали сигмоидальную функцию активации нейронов.
- Увидели, что по сути нейронные сети – это просто набор нейронов, связанных между собой.
- Создали обучающую выборку, содержащую объекты с параметрами вес и рост в качестве входных данных, а также использовали пол в качестве вывода (или маркера класса).
- Узнали о функциях потерь на основе математической функции среднеквадратической ошибки (MSE).
- Узнали, что тренировка нейронной сети – это минимизация ее потерь.
- Использовали обратное распространение (backprop) для вычисления частных производных.
- Использовали стохастический градиентный спуск (SGD) для тренировки нейронной сети.

Содержание работы

1. Ознакомиться с процессом обучения нейронной сети.
2. Подготовить тренировочный набор с новыми данными, предназначенными для решения бинарной классификации.
3. Реализовать обучение и проверку работы сети. Проанализировать полученный результат.
4. Оформить отчет по лабораторной работе.

Содержание отчета

1. Цель работы.
2. Формулировка задания.
3. Программный код с комментариями.
4. Результат.
5. Выводы по полученным результатам и лабораторной работе.