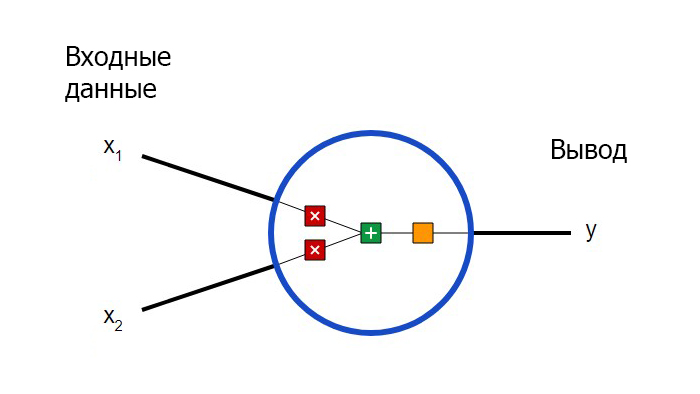
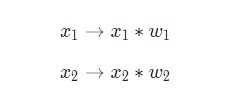
**Пример реализации нейронной сети на Python**

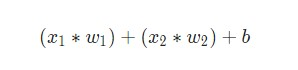
Нейрон с двумя входами имеет следующую структуру:



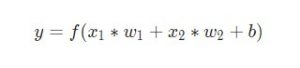
Во-первых, каждый вход умножается на вес (на схеме обозначен красным):



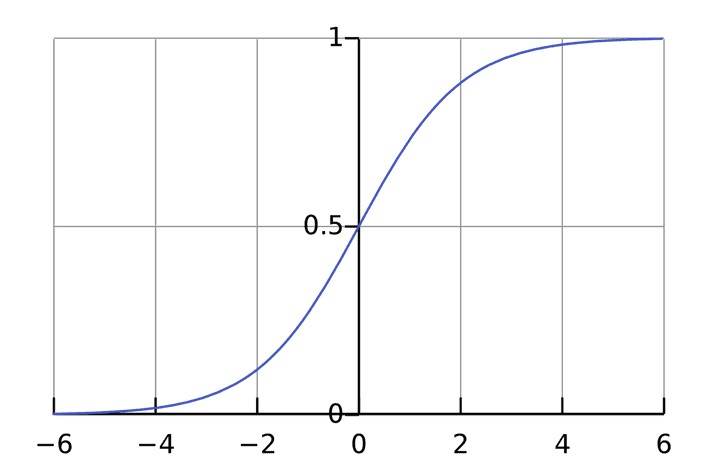
Затем все взвешенные входы складываются вместе со смещением b (на схеме обозначен зеленым):



Наконец, сумма передается через функцию активации (на схеме обозначена **желтым**):

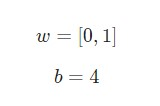


Функция активации используется для преобразования несвязанных входных данных в выходной сигнал. Как правило, в качестве функции активации берется функция [сигмоида](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BC%D0%BE%D0%B8%D0%B4%D0%B0" \t "_blank):

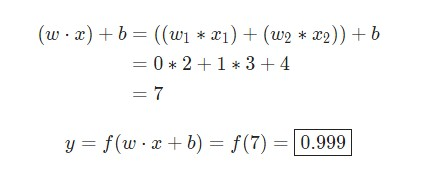


Функция сигмоида выполняет отображение значения из диапазона (−∞, +∞) в значение из диапазона (0, 1). Большие отрицательные числа преобразуются в ~0, а большие положительные числа в ~1.

Допустим



w = [0,1] — это просто один из способов написания w1 = 0, w2 = 1 в векторной форме. Присвоим нейрону вход со значением x = [2, 3]. Для более компактного представления будет использовано скалярное произведение.



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

Ниже приводится реализация нейрона на языке Python. Для этого потребуется использовать NumPy. Это мощная вычислительная библиотека Python, которая реализует математические операции:

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):

        # 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

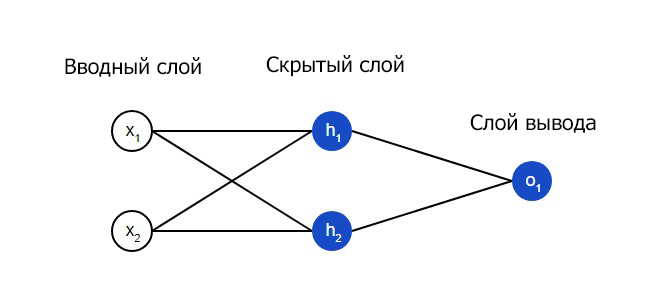
bias = 4  # b = 4

n = Neuron(weights, bias)

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

print(n.feedforward(x))  **# 0.999**

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

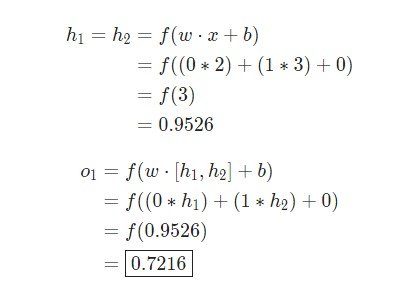


На входном слое сети два входа – x1 и x2. На скрытом слое два нейтрона — h1 и h2. На слое вывода находится один нейрон – о1. Обратите внимание на то, что входные данные для о1 являются результатами вывода h1 и h2.

**Пример прямого распространения FeedForward**

**И**спользуем 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 h1, h2 и o1 сами отметят результаты вывода представленных ими нейронов.

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



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

Следующий код реализует механизм прямого распространения feedforward для рассмотренного примера.

import numpy as np

# Здесь код из предыдущего раздела

 сlass 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 о1 являются выводами 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))  **# 0.7216**

Получили результат, как в примере выше.

## Пример обучения нейронной сети. Минимизация ошибки обучения.

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 |

Необходимо обучить нейронную сеть таким образом, чтобы она предсказывала пол заданного человека в зависимости от его веса и роста.

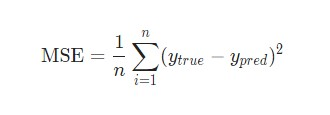
## 

## 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.

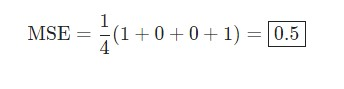
Задача обучения нейронной сети сводится к минимизации ошибки обучения. В данном случае будет использоваться среднеквадратическая ошибка обучения:



* 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;
* ytrue – истинное значение переменной, то есть так называемый правильный ответ. Например, для Alice значение ytrue будет 1, то есть Female;
* ypred – предполагаемое значение переменной. Это результат вывода сети.

Допустим наша сеть всегда выдает 0. Другими словами, она уверена, что все люди — Мужчины. Какой будет ошибка обучения?

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя/Name** | **ytrue** | **ypred** | **(ytrue — ypred)2** |
| Alice | 1 | 0 | 1 |
| Bob | 0 | 0 | 0 |
| Charlie | 0 | 0 | 0 |
| Diana | 1 | 0 | 1 |



Код, вычисляющий MSE:

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**

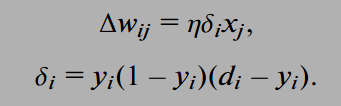
При возникновении сложностей с пониманием работы кода стоит ознакомиться с реализацией **операций с массивами в** NumPy.

Для минимизации ошибки обучения будем использовать метод градиентного спуска.

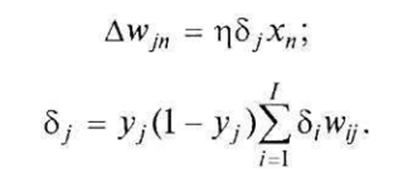
В соответствии с этим методом, для выходного нейрона веса синаптических связей и смещения (с учетом использования сигмоидной активационной функции) вычисляются по формуле



где



Для нейронов скрытого слоя веса синаптических связей, в соответствии с алгоритмом обратного распространения ошибки, вычисляются по формуле:



В нашем случае один выходной нейрон, поэтому



Ниже приводится программа, реализующая нейронную сеть из примера.

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)

"""

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

# количество циклов во всём наборе данных

epochs = 1000

for epoch in range(epochs):

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

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)

# для нейрона o1 обновляем веса и смещение в соответствии с методом градиентного

#спуска

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

# --- для нейронов h1 и h2 обновляем веса и смещения в соответствии с методом

# обратного распространения ошибки

# 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

# --- Подсчитываем общую ошибку обучения 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 нейронную сеть

network = OurNeuralNetwork()

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

# 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