

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Разработка систем поддержки принятий решений»**

Студент группы: ИКБО-14-20 Вежновец Ф.Ю. *(Фамилия студента)*

Руководитель самостоятельной работы Холмогоров В.В.

*(Фамилия преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc138003573)

[1 АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ 5](#_Toc138003574)

[1.1 Постановка задачи 5](#_Toc138003575)

[1.2 Описание алгоритма 5](#_Toc138003576)

[1.3 Результат работы программы 7](#_Toc138003577)

[ЗАКЛЮЧЕНИЕ 9](#_Toc138003578)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 10](#_Toc138003579)

[ПРИЛОЖЕНИЯ 11](#_Toc138003580)

ВВЕДЕНИЕ

Среди различных структур нейронных сетей (НС) одной из наиболее известных является многослойная структура, в которой каждый нейрон произвольного слоя связан со всеми аксонами нейронов предыдущего слоя или, в случае первого слоя, со всеми входами НС. Такие НС называются полносвязными. Когда в сети только один слой, алгоритм ее обучения с учителем довольно очевиден, так как правильные выходные состояния нейронов единственного слоя заведомо известны и подстройка синаптических связей идет в направлении, минимизирующем ошибку на выходе сети. По этому принципу строится, например, алгоритм обучения однослойного персептрона. В многослойных же сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, не известны, и двух- или более слойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС.

Для обучения многослойной сети в 1986 г. Руммельхартом и Хинтоном был предложен алгоритм обратного распространения ошибок (error back propagation). Многочисленные публикации о промышленных применениях многослойных сетей с этим алгоритмом обучения подтвердили его принципиальную работоспособность на практике.

Основная идея обратного распространения состоит в том, как получить оценку ошибки для нейронов скрытых слоев. Заметим, что известные ошибки, делаемые нейронами выходного слоя, возникают вследствие неизвестных пока ошибок нейронов скрытых слоев. Чем больше значение синоптической связи между нейроном скрытого слоя и выходным нейроном, тем сильнее ошибка первого влияет на ошибку второго. Следовательно, оценку ошибки элементов скрытых слоев можно получить, как взвешенную сумму ошибок последующих слоев. При обучении информация распространяется от низших слоев иерархии к высшим, а оценки ошибок, делаемые сетью - в обратном направлении, что и отражено в названии метода.

Таким образом, входные сигналы двигаются в прямом направлении, в результате чего мы получаем выходной сигнал, из которого мы получаем значение ошибки. Величина ошибки двигается в обратном направлении, в результате происходит корректировка весовых коэффициентов связей сети.

1. АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

1.1 Постановка задачи

Реализовать алгоритм обратного распространения ошибки в многослойной нейронной сети, для классификации цифр.

* 1. Описание алгоритма

Общая структура алгоритма аналогична, алгоритму обучения Розенблатта (дельта-правило) с усложнением формул подстройки весов. В качестве активационной функции в многослойных персептронах, как правило, используется сигмоидальная активационная функция, в частности логистическая:

𝒇(𝑼) = 𝟏⁄(𝟏 + 𝒆−𝒙) или 𝑶𝑼𝑻 = 𝟏⁄(𝟏 + 𝒆−𝑵𝑬𝑻)

Вспомним, что производная этой функции равна:

𝑶𝑼𝑻/ = OUT(1 – OUT)

Рассмотрим алгоритм обратного распространения ошибки, который требует выполнения следующих операций:

Шаг 1. Выбрать очередную обучающую пару из обучающего множества; подать входной вектор на вход сети. Берется пример входного сигнала с соответствующим правильным значением выхода.

Шаг 2. Вычислить выход сети. При этом нейроны последовательно от слоя к слою функционируют по следующим формуле.

Операции, выполняемые шагами 1 и 2, сходны с теми, которые выполняются при функционировании уже обученной сети, – подается входной вектор и

вычисляется получающийся выход. Вычисления выполняются послойно. На шаги 1 и 2 можно смотреть как на «движение вперед», так как сигнал распространяется по сети от входа к выходу

Шаг 3. Рассчитывается прямое распространение ошибки через сеть. Берется, функционал суммарной квадратичной ошибки сети для одного входного образа имеет вид:

𝐸 = 0,5 ∑(𝑑𝑗 − 𝑦𝑗)2

𝑛

где dj – целевое значение входного сигнала; yj – реальное значение входного сигнала.

Шаг 4. Начиная с выходов, выполняется обратное движение через ячейки выходного и промежуточного слоя, при этом программа рассчитывает значения:

Для выходной ячейки

✿𝟎 = 𝒚𝒋(𝟏 − 𝒚𝒋)(𝒅𝒋 − 𝒚𝒋) или ✿𝟎 = 𝑶𝑼𝑻(𝟏 − 𝑶𝑼𝑻)(𝑶𝑼𝑻∗ − 𝑶𝑼𝑻);

где 𝑂𝑈𝑇∗ - целевое значение.

𝑂𝑈𝑇 – реальное значение

𝑂𝑈𝑇(1 − 𝑂𝑈𝑇) – производная от сигмоида.

Выход нейрона слоя, вычитаемый из целевого значения, дает сигнал ошибки. Он умножается на производную сжимающей функции 𝑂𝑈𝑇(1 − 𝑂𝑈𝑇), вычисленную для этого нейрона, давая, таким образом, величину обратной ошибки в узле.

Для скрытых ячеек

𝜹𝒋 = 𝑶𝑼𝑻(𝟏 − 𝑶𝑼𝑻) ∑𝒌 𝑚𝒌𝒋 ✿𝒌

𝜹𝒋 – ошибка элемента с индексом j;

k – индекс, соответствующий слою, который посылает ошибку «обратно»;

∑𝒌 𝑚𝒌𝒋 ✿𝒌 – обозначает все ячейки, связанные со скрытым слоем, 𝑚𝒊𝒋 – заданный вектор веса в скрытом слое, ✿𝒌 – обратная ошибка от слоя, который ее посылает.

Шаг 5. Рассчитываем величину, на которую необходимо изменить значения весовых коэффициентов. При этом используется дельта-праило.

Тогда для весов соединений между скрытым слоем и выходом

𝒘∗ = 𝒘𝒊𝒋 + 𝑎 𝜹𝟎𝒇(𝑼𝒊)

где i – номер нейрона в выходном слое; j – номер нейрона в скрытом слое; α – коэффициент обучения; 𝒇(𝑼𝒊)– входной сигнал; 𝜹𝟎 – ошибка обучения на выходе.

Для весов соединений между скрытым слоем и входом

* 1. Результат работы программы

Архитектура сети состоит из 15 входных нейронов, 5 нейронов в скрытом слое и 1 выходной нейрон (рисунок 1).

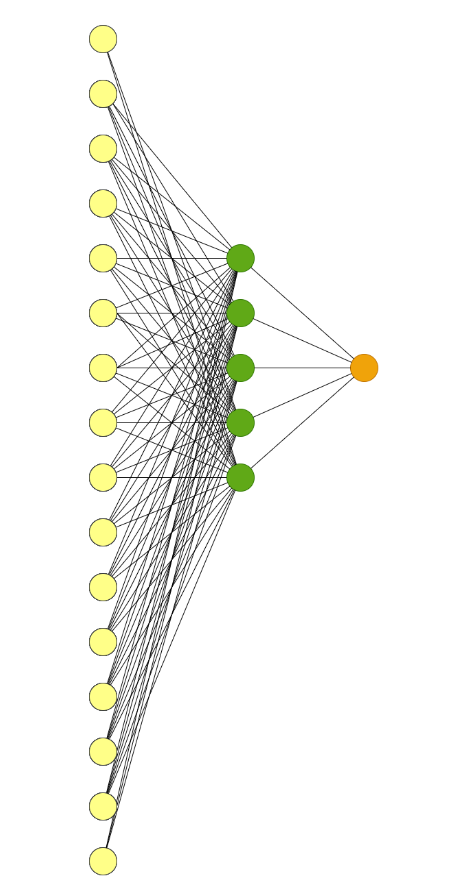


Рисунок 1 – Архитектура нейронной сети

Будем распознавать цифру 1 (скобочка). В качестве функции активацию на всех слоях будем использовать сигмойду.

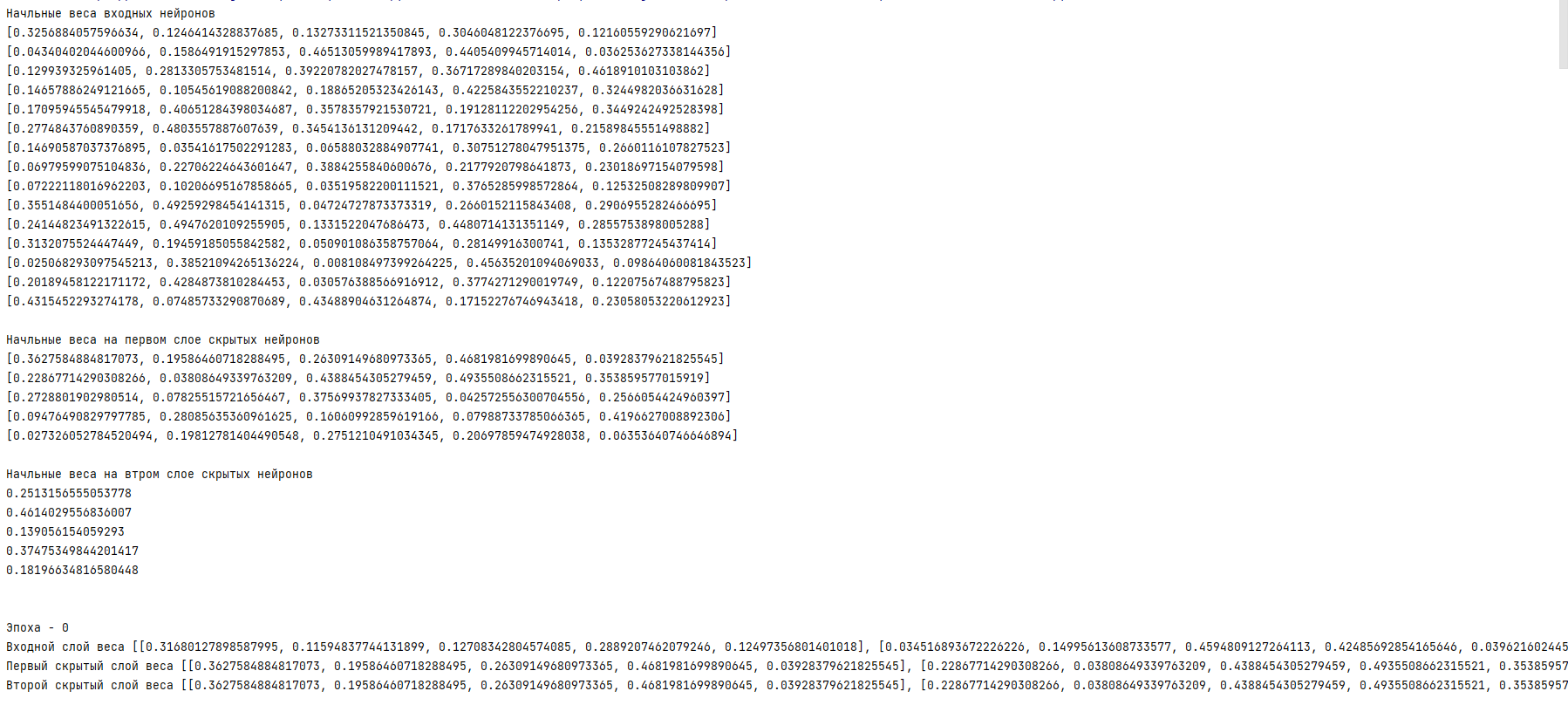


Рисунок 2 – Результат работы программы

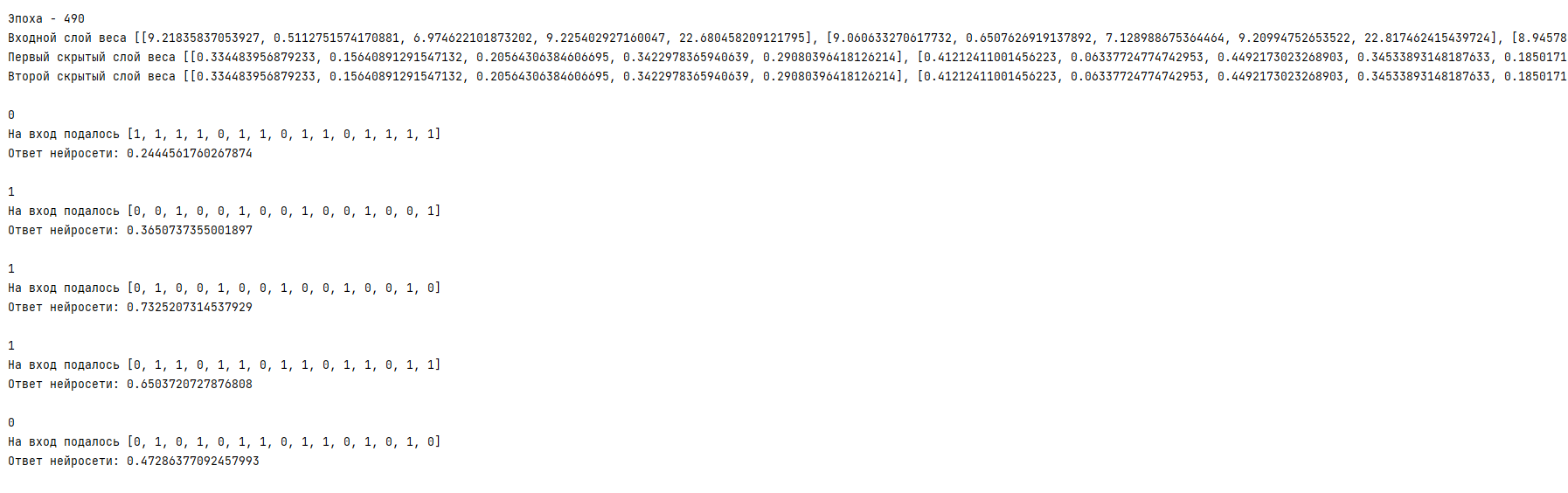


Рисунок 3 – Результат работы программы

Нейронная сеть правильно классифицирует единицу и обучается при помощи обратного распространения ошибки.

ЗАКЛЮЧЕНИЕ

В данной практике был реализована многослойная нейронная сеть, в качестве оптимизатора был реализован алгоритм обратного распространения ошибки, на языке высокого уровня python.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Линейное программирование: практикум [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин, Е. В. Бражникова, О. В. Платонова. — М.: МИРЭА, 2017. — Электрон. опт. диск (ISO).
2. Методы оптимизации: гибридные генетические алгоритмы [Электронный ресурс]: учебно-метод. пособие / А. Б. Сорокин. — М.: МИРЭА, 2016. — Электрон. опт. диск (ISO).

ПРИЛОЖЕНИЯ

Приложение А – Листинг кода для нейронной сети

Приложение А

Листинг кода для нейронной сети

Листинг А.1 – Используемые библиотеки

import math

import random

Листинг А.2 – Функция main

if \_\_name\_\_ == '\_\_main\_\_':

input\_neurons\_weight = get\_start\_weight()

first\_hidden\_neurons\_weight = get\_first\_hidden\_weight()

second\_hidden\_neurons\_weight = get\_second\_hidden\_weight()

data, y = get\_data\_and\_y()

multi\_layered\_thing = Neural(data, y, input\_neurons\_weight, first\_hidden\_neurons\_weight,

second\_hidden\_neurons\_weight,

learning\_step=0.1, iteration=500)

multi\_layered\_thing.train()

new\_data = [1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]

print('0')

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]

multi\_layered\_thing.predict(new\_data)

print('1')

new\_data = [0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1]

multi\_layered\_thing.predict(new\_data)

print('0')

new\_data = [0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0]

multi\_layered\_thing.predict(new\_data)

Листинг А.3 – Функция генерации начальных весов

def get\_start\_weight(): # генерируем веса для 15 входных нейронов

input\_neurons\_weight = []

for j in range(15):

weight = []

for i in range(5):

weight.append(random.uniform(0, 0.5))

input\_neurons\_weight.append(weight)

print('Начльные веса входных нейронов')

for i in input\_neurons\_weight:

print(i)

print()

return input\_neurons\_weight

Листинг А.4 – Функция генерации данных

def get\_data\_and\_y():

data = [[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],

[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0],

[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]]

y = [[0], [1], [0], [1], [0], [1]]

return data, y

Листинг А.5 – Функция сигмойды

def singmoid(input\_value):

return 1 / (1 + math.e \*\* (-input\_value))

Листинг А.6 – Функция производной сигмойды

def grad\_singmoid(input\_value):

return input\_value \* (1 - input\_value)

Листинг А.7 – Класс нейросети

class Neural:

def \_\_init\_\_(self, data, y, input\_neurons\_weight, first\_hidden\_neurons\_weight, second\_hidden\_neurons\_weight,

learning\_step, iteration):

self.data = data

self.y = y

self.input\_neurons\_weight = input\_neurons\_weight

self.first\_hidden\_neurons\_weight = first\_hidden\_neurons\_weight

self.second\_hidden\_neurons\_weight = second\_hidden\_neurons\_weight

self.learning\_step = learning\_step

self.iteration = iteration

Листинг А.8 –Мметод обчуния

def train(self):

epoch = 0

while epoch != self.iteration:

self.direct\_distribution()

if epoch % 10 == 0:

print()

print(f'Эпоха - {epoch}')

print(f'Входной слой веса {self.input\_neurons\_weight}')

print(f'Первый скрытый слой веса {self.first\_hidden\_neurons\_weight}')

print(f'Второй скрытый слой веса {self.first\_hidden\_neurons\_weight}')

print()

epoch += 1

Листинг А.9 – Обартное распространение ошибки

def direct\_distribution(self):

for d, dd in enumerate(self.data): # проход по данным

values\_on\_input\_second\_layer = [0] \* len(self.second\_hidden\_neurons\_weight)

values\_on\_out\_layer = 0

for i, ii in enumerate(self.first\_hidden\_neurons\_weight):

for j, jj in enumerate(self.first\_hidden\_neurons\_weight):

values\_on\_input\_second\_layer[j] += dd[j] \* self.first\_hidden\_neurons\_weight[j][i]

values\_on\_input\_second\_layer = [singmoid(i) for i in values\_on\_input\_second\_layer]

# print(f'Значения после второго слоя - {values\_on\_input\_second\_layer}')

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

values\_on\_out\_layer += values\_on\_input\_second\_layer[j] \* self.second\_hidden\_neurons\_weight[j]

values\_on\_out\_layer = singmoid(values\_on\_out\_layer)

# print(f'Значения на выходном слое слое - {values\_on\_out\_layer}')

# print()

# обратное распрастранение ошибки

# расчёт ошибки на всех уровнях

answer\_error = self.y[d][0] - values\_on\_out\_layer

# print('Ошибки сети', answer\_error)

second\_layer\_error = [0] \* len(values\_on\_input\_second\_layer)

# first\_layer\_error = [0] \* len(values\_on\_input\_first\_layer)

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

second\_layer\_error[j] += answer\_error \* self.second\_hidden\_neurons\_weight[j]

# print('Ошибки на втором скрытом слое', second\_layer\_error)

# обновляем веса

for i, ii in enumerate(self.input\_neurons\_weight[0]):

for j, jj in enumerate(self.input\_neurons\_weight):

self.input\_neurons\_weight[j][i] += \

second\_layer\_error[i] \* grad\_singmoid(values\_on\_input\_second\_layer[i]) \* \

data[d][i] \* self.learning\_step

# print(f'Новые веса для первого слоя')

# for i in input\_neurons\_weight:

# print(i)

# print()

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

self.second\_hidden\_neurons\_weight[j] += \

answer\_error \* grad\_singmoid(values\_on\_out\_layer) \* \

values\_on\_input\_second\_layer[j] \* self.learning\_step

Листинг А.10 – Предсказение нейросети

def predict(self, new\_data):

values\_on\_input\_second\_layer = [0] \* len(self.second\_hidden\_neurons\_weight)

values\_on\_out\_layer = 0

for i, ii in enumerate(self.first\_hidden\_neurons\_weight):

for j, jj in enumerate(self.first\_hidden\_neurons\_weight):

values\_on\_input\_second\_layer[j] += new\_data[j] \* self.first\_hidden\_neurons\_weight[j][i]

values\_on\_input\_second\_layer = [singmoid(i) for i in values\_on\_input\_second\_layer]

for j, jj in enumerate(self.second\_hidden\_neurons\_weight):

values\_on\_out\_layer += values\_on\_input\_second\_layer[j] \* self.second\_hidden\_neurons\_weight[j]

values\_on\_out\_layer = singmoid(values\_on\_out\_layer)

print(f'На вход подалось {new\_data}')

print(f'Ответ нейросети: {values\_on\_out\_layer}')

print()