Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №4

“ Алгоритм обратного распространения ошибки”

Выполнил: Студент 1 курса

Факультета Цифровых

Промышленных Технологий

группы 20221 Хохлов Д.Р.

Преподаватель:

Кафедра киберфизических систем

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc183382922)

[1 Теоретическая часть 4](#_Toc183382923)

[2 Практическая часть 6](#_Toc183382924)

[2.1 Постановка задачи 6](#_Toc183382925)

[2.2 Таблица истинности 6](#_Toc183382926)

[2.3 Проектирование нейронной сети 7](#_Toc183382927)

[2.4 Реализация программы 8](#_Toc183382928)

[2.5 Результаты обучения 9](#_Toc183382929)

[2.6 Тестирование сети 9](#_Toc183382930)

[Приложение А 11](#_Toc183382931)

[Листинг – Установка и импорт библиотек 11](#_Toc183382932)

[Листинг – класс “Layer” 11](#_Toc183382933)

[Листинг – класс “Activation” с функциями активации 11](#_Toc183382934)

[Листинг – класс “NN” с алгоритмом обратного распространения 12](#_Toc183382935)

[Листинг – Обучение нейросети на данных 13](#_Toc183382936)

[Листинг – Тестирование нейросети и вывод таблицы истинности 13](#_Toc183382937)

# Введение

Цель работы: проектирование нейросети с использованием алгоритма обратного распространения ошибки

Задача: доработать программу из второй лабораторной работы, необходимо сделать теперь её автоматическое обучение с помощью алгоритма обратного распространения ошибки.

# 1 Теоретическая часть

Алгоритм обратного распространения (backpropagation) (Рисунок 1) — это метод, используемый для обучения нейронных сетей. Он минимизирует функцию потерь, изменяя веса сети с помощью вычисления градиента.

Данный алгоритм состоит из нескольких этапов:

1. **Прямое распространение (Forward)**: входные данные проходят через сеть, вычисляется выход.
2. **Вычисление ошибки**: сравнивается выход сети с истинным значением с помощью функции потерь, например:
   * Среднеквадратичная ошибка (MSE):
   * Кросс-энтропия:
3. **Обратное распространение (Backward)**:
   * Используя правило цепочки, вычисляется градиент ошибки по каждому весу и смещению.
   * Градиенты передаются от выходного слоя к входному.
4. **Обновление параметров**: веса корректируются с учетом градиента.

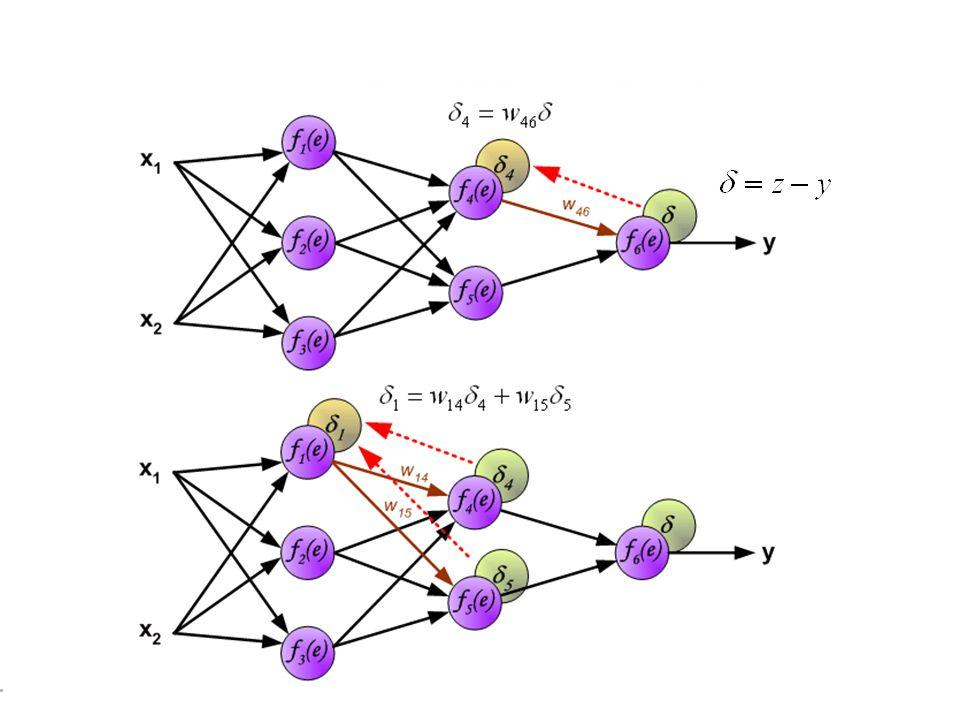


Рисунок 1 – Работа обратного распространение ошибки

Градиентный спуск (Рисунок 2) — это оптимизационный алгоритм, используемый для минимизации функции потерь путем итеративного изменения параметров модели.

Существует множество видов градиентного спуска, вот несколько из них:

1. **Пакетный градиентный спуск** (Batch Gradient Descent): использует все данные для вычисления среднего градиента.
2. **Стохастический градиентный спуск** (SGD): обновляет веса после обработки каждого образца.
3. **Мини-пакетный градиентный спуск** (Mini-batch Gradient Descent): использует небольшие группы данных (мини-батчи), комбинируя преимущества первых двух методов.

В приведённых выше примерах, существует проблема локального минимума (ну т.е. когда минизация ошибки достигается в каком-то определённом участке, а не на всём графике и из-за этого обучение застревает), однако в новых видах градиентного спуска, такие как Adam, данная проблема решена.

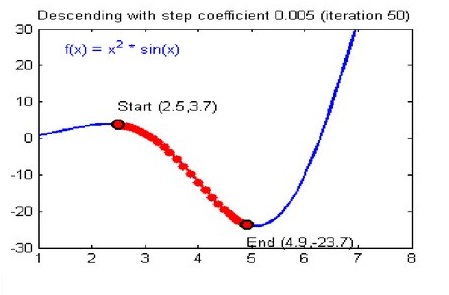


Рисунок 2 – Пример градиентного спуска

# 2 Практическая часть

## 2.1 Постановка задачи

Задача данной лабораторной работы заключается в доработке Лабораторной работы №2, где использовалась нейросеть на основе персептрона Розенблатта. В данной лабораторной работе нужно использовать алгоритм обратного распространения ошибки (backpropagation).

Для выполнения задачи необходимо:

1. Составить таблицу истинности для соответствующей схемы.
2. Разработать архитектуру нейронной сети, состоящую из входного слоя, нескольких скрытых слоёв и выходного слоя
3. Реализовать алгоритм обратного распространения ошибки для обучения нейронной сети
4. Провести проверку работы сети на тестовых данных

## 2.2 Таблица истинности

Таблица истинности реализуется на основании схемы (Рисунок 3), соответствующей варианту задания, была составлена таблица истинности, которая представляет собой список всех возможных входных состояний системы и соответствующих им выходных состояний. Эта таблица является основой для дальнейшего построения нейронной сети.

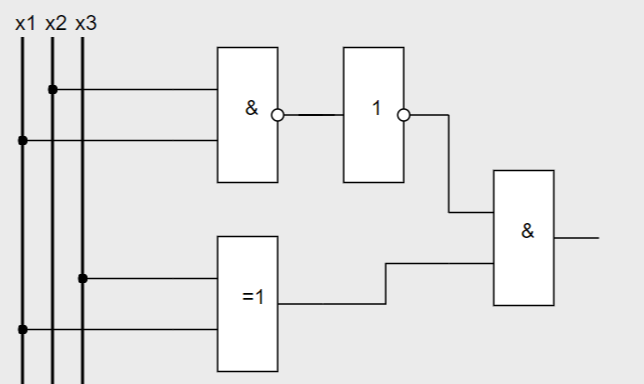


Рисунок 3 - Логическая схема

В данном случае таблица истинности будет иметь такой вид:

Таблица 1 – Таблица истинности

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x1** | **x2** | **x3** | **¬(x1∧x2)** | **¬(¬(x1∧x2))** | **x1⊕x2** | **¬(x1∧x2)∧x1⊕x2** |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |

В таблице истинности (Таблица 1) нам нужны 3 входа и последний выход, все остальные операции нейросеть обучится выполнять самостоятельно с помощью алгоритма обратного распространения.

## 2.3 Проектирование нейронной сети

Нейросеть, которую необходимо спроектировать состоит из нескольких входных нейронов (3 нейрона), нескольких скрытых слоёв (2 скрытых слоя) в которых по несколько нейронов (по 4 нейрона на каждый скрытый слой) и выходной нейрон (1 нейрон)

В переходах между слоями должны использоваться функции активации, в данной задаче используется 2 вида функций активации:

* Гиперболический тангенс для входного и скрытых слоёв
* Сигмоидальная функция активации для выходного слоя

Таким образом, такая нейросеть имеет вид:

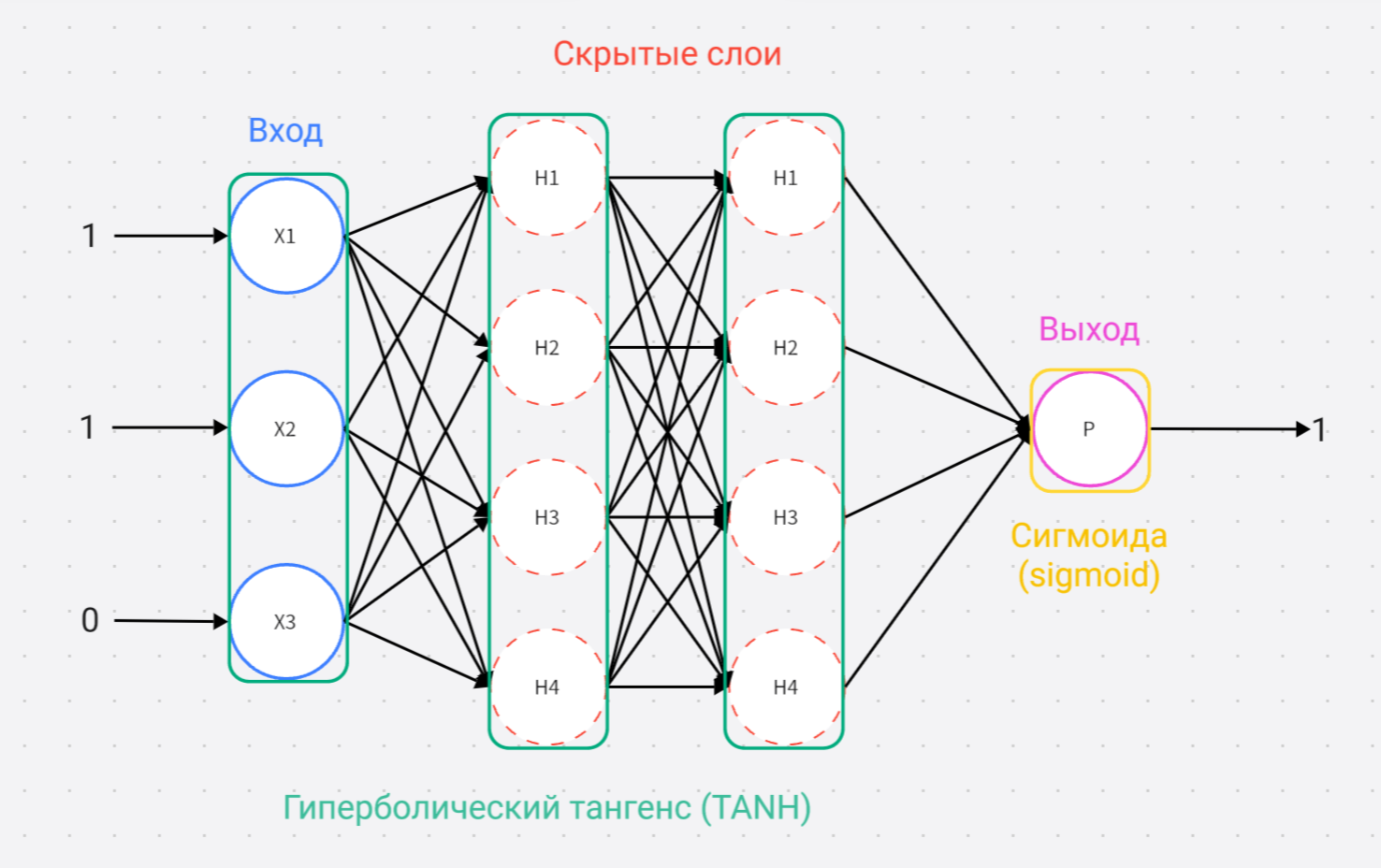


Рисунок 4 – Схема нейросети

## 2.4 Реализация программы

Программа для реализации нейронной сети была написана с использованием библиотеки NumPy, для линейной алгебры, библиотеки PrettyTable, для вывода таблицы истинности.

Нейросеть обучается при помощи алгоритма обратного распространения ошибки (backpropagation)

Нейросеть изначально берёт случайные веса, и в итоге её начальные параметры выглядят так:

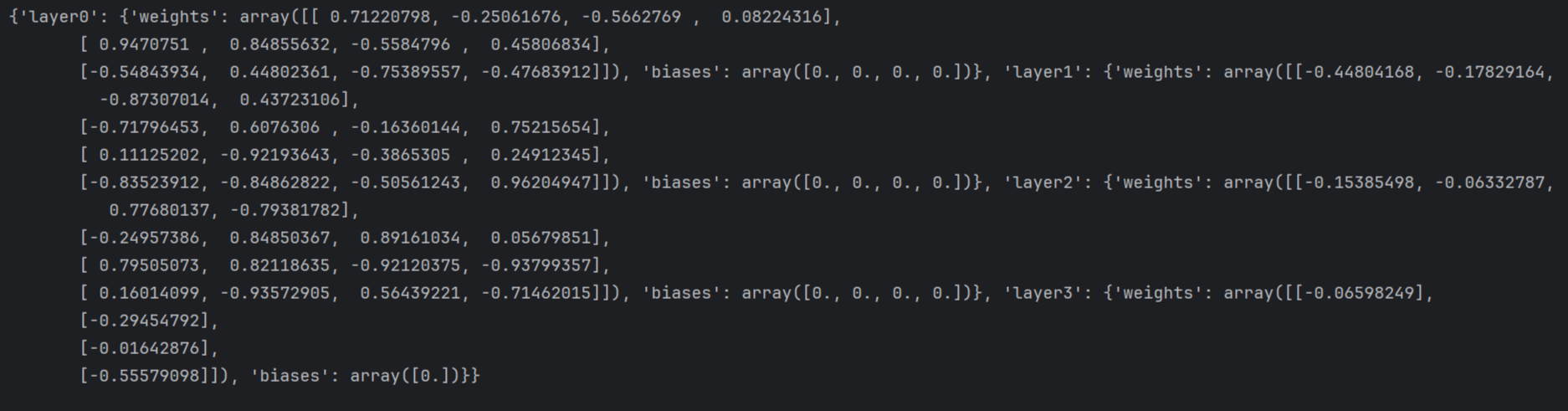


Рисунок 5 – Начальные параметры нейросети

Обучение нейросети на данных взятых из таблицы истинности, составляет ~ 65мс

После обучения параметры нейросети изменились, и она обучилась выполнять данную ей задачу (Рисунок 6)

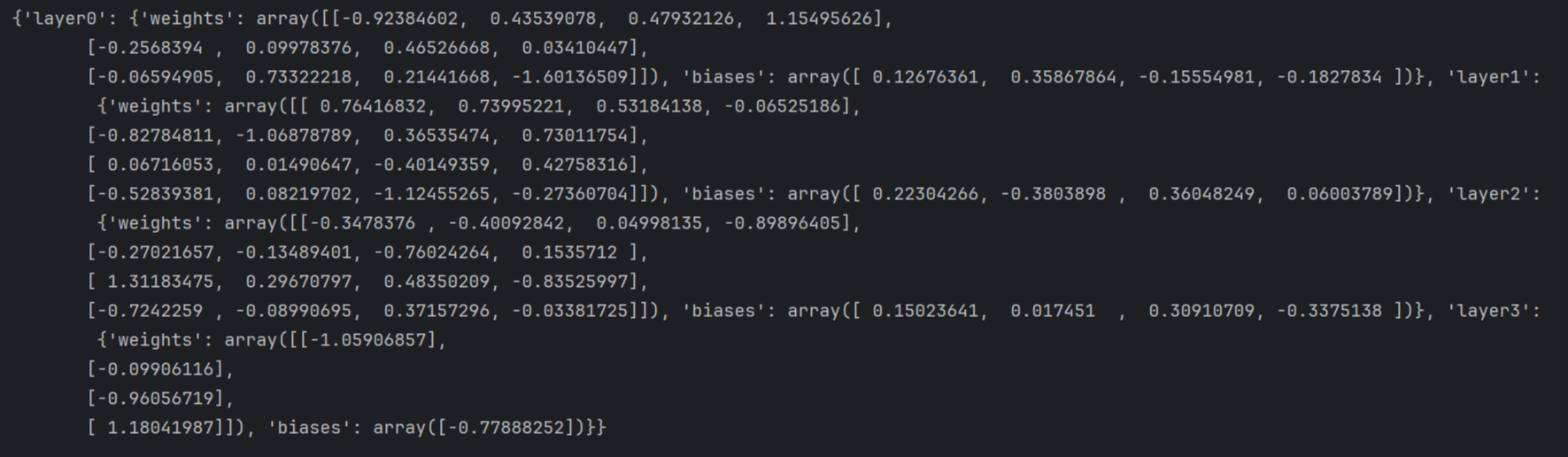


Рисунок 6 – Параметры нейросети после обучения

## 2.5 Результаты обучения

В процессе обучения параметры нейросети изменялись. По мере уменьшения ошибок сеть постепенно обучалась правильно воспроизводить таблицу истинности.

## 2.6 Тестирование сети

После завершения обучения была проведена проверка работы нейронной сети на тестовых данных . Для этого использовались все возможные комбинации входных данных из таблицы истинности. Программа выводила полученные выходные значения и сравнивала их с ожидаемыми результатами. В результате тестирования была построена итоговая таблица истинности (Рисунок 7), которая подтвердила корректную работу сети.

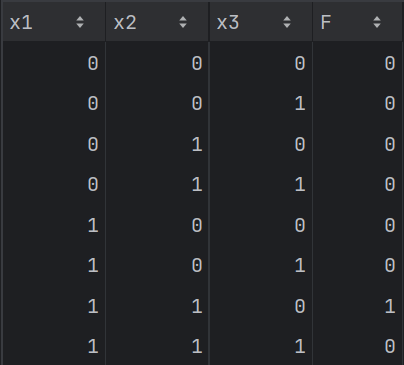


Рисунок 7 – Таблица истинности от нейросети (Готовый результат)

Нейросеть вывела только входные и выходные данные и от полной формулы из таблицы истинности, так как при таком алгоритме обучения мы не можем посмотреть каждое действие, которое выполнила нейронная сеть.

# Приложение А

**Листинг программы**

## Листинг – Установка и импорт библиотек

#~~Установка библиотек~~#

!pip install numpy  
!pip install prettytable

#~~Импортирование библиотек~~#  
import random  
  
import numpy as np  
  
from prettytable import PrettyTable

## Листинг – класс “Layer”

class Layer:  
 def \_\_init\_\_(self, input\_size, output\_size, activation\_func):  
 self.activate, self.derivative = activation\_func() #функция активации и его производная   
 self.weights = np.random.uniform(-1.0,1.0,size=(input\_size, output\_size))  
 self.biases = np.zeros(output\_size)   
 def \_\_call\_\_(self):  
 return {  
 'weights':self.weights,   
 'biases':self.biases  
 }

## Листинг – класс “Activation” с функциями активации

class Activation:  
 @staticmethod  
 def SIGMOID():  
 def activation(x):  
 return 1 / (1 + np.exp(-x))  
 def derivative(x):  
 return x \* (1 - x)  
 return activation, derivative  
   
 @staticmethod  
 def TANH():  
 def activation(x):  
 return np.tanh(x)  
 def derivative(x):  
 return 1 - x \*\* 2  
 return activation, derivative

## Листинг – класс “NN” с алгоритмом обратного распространения

class NN:  
 def \_\_init\_\_(self, layers):  
 self.layers = layers  
 self.params = self.\_\_params()  
   
 def \_\_params(self):  
 params = {}  
 for i, layer in enumerate(layers):  
 params[f'layer{i}'] = {"weights":layer.weights,  
 "biases":layer.biases}  
 return params  
 def \_\_forward(self, x):  
 for layer in self.layers:  
 z = np.dot(x, layer.weights) + layer.biases  
 x = layer.activate(z)  
 layer.output = np.array([x])  
 return x  
   
 def \_\_backward(self, x, y, lr):  
 error = -(y - self.layers[-1].output)  
 for layer in reversed(self.layers):  
 delta = error \* layer.derivative(layer.output)  
 prev\_output = np.array([x]) if layer == self.layers[0] else self.layers[self.layers.index(layer) - 1].output  
   
 layer.weights -= lr \* np.dot(prev\_output.T, delta)  
 layer.biases -= lr \* np.sum(delta, axis=0)  
 error = np.dot(delta, layer.weights.T)  
  
 def train(self, input\_data, output\_data, learning\_rate=0.01):  
 index\_list = list(range(input\_data.shape[0]))  
 all\_correct = False  
 while not all\_correct:  
 all\_correct = True  
 random.shuffle(index\_list)  
 for i in index\_list:   
 self.\_\_forward(input\_data[i])  
 self.\_\_backward(input\_data[i], output\_data[i][0], learning\_rate)  
 for i in range(input\_data.shape[0]):   
 p\_out = self.\_\_forward(input\_data[i])  
 all\_correct = False if(((output\_data[i] < 0.5) and (p\_out >= 0.5)) or ((output\_data[i] >= 0.5) and (p\_out < 0.5))) else all\_correct   
   
 def \_\_call\_\_(self, input\_data):  
 output = []  
 for i in range(input\_data.shape[0]):  
 output.append(1 if self.\_\_forward(input\_data[i]) >= 0.5 else 0)  
 return output

## Листинг – Обучение нейросети на данных

LEARNING\_RATE = 0.1  
#~~Тренировочные данные~~#  
train\_input = np.array([[0, 0, 0],  
 [0, 0, 1],  
 [0, 1, 0],  
 [0, 1, 1],  
 [1, 0, 0],  
 [1, 0, 1],  
 [1, 1, 0],  
 [1, 1, 1]])  
train\_output = np.array([[0,0,0,0,0,0,1,0]]).T  
#%%  
#~~Структура нейросети~~#  
layers = (  
 Layer(train\_input.shape[1], 4, Activation.TANH),  
 Layer(4, 4, Activation.TANH),  
 Layer(4, 4, Activation.TANH),  
 Layer(4, 1, Activation.SIGMOID),  
)  
#%%  
model = NN(layers) #создание объекта нейросети  
print(model.params) #вывод параметров  
#%%  
#~~Обучение нейросети~~#  
model.train(train\_input, train\_output, learning\_rate=LEARNING\_RATE)   
#%%  
print(model.params) #вывод параметров

## Листинг – Тестирование нейросети и вывод таблицы истинности

pt = PrettyTable(['x1','x2','x3','F'])  
for i in range(train\_data.shape[0]):  
 pt.add\_row([train\_data[i][0], train\_data[i][1],train\_data[i][2], model(train\_data)[i]])  
print(pt)