МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Автоматизация схемотехнического проектирования»

Тема: НЕЙРОННЫЕ СЕТИ

Студент гр. 1302	_____	Новиков Г.В.
Студентка гр. 1302	_____	Романова О.В.
Студентка гр. 1302	_____	Марзаева В.И.
Преподаватель	_____	Боброва Ю.О.

Санкт-Петербург

2025

Цель работы

Создание простейшей нейронной сети на Python без использования специализированных библиотек.

Основные теоретические положения

Простейшая нейронная сеть состоит из слоя искусственных нейронов (ИН). ИН по своей сути представляет из себя алгоритм вычисления выхода из входных данных по известной формуле, с использованием набора весовых коэффициентов w и смещений b.

На вход нейрона поступают сигналы xi, каждый умножается на соответствующий коэффициент wi и суммируется. Полученная взвешенная сумма поступает на функцию активации, результат вычисления которой и является выходом нейрона.

Простейшей функцией активации является функция единичного скачка, принимающая только два значения – 0, если net ниже порогового значения, 1 если превышает. Самой распространенной функцией активации является сигмоида, с которой вы уже знакомы. На выходе нейрон отдает значение от 0 до 1. Существует множество других видов функций активации, с которыми вы познакомитесь в рамках курса.

Нейронные сети состоят из набора нейронов, сгруппированных по слоям. В однослойных нейронных сетях сигналы с входного слоя сразу подаются на выходной слой. Он производит необходимые вычисления, результаты которых сразу подаются на выходы.

Входы не считаются за входной слой сети и обозначены кружками. Справа (квадраты) расположены нейроны основного слоя. Под обучением нейронной сети понимается поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется с минимальной ошибкой в нужный нам выходной.

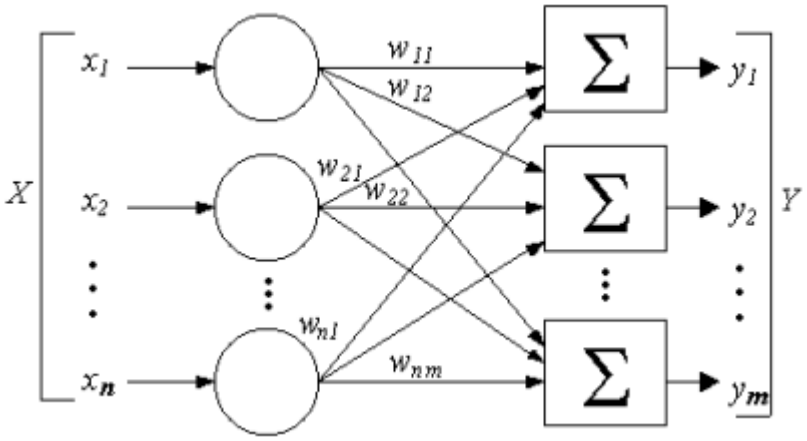

Рис. 1. Структура нейронной сети

Ход работы

1. *Полный код программы:*

lab4.py:

data_generator.py:

```
import numpy as np

def norm_dataset(mu,sigma,N):
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]

    col = len(mu0)                          # количество столбцов-признаков – длина массива средних

    class0 = np.random.normal(mu0[0], sigma0[0], [N, 1])      # инициализируем первый столбец (в Python нумерация от 0)
    class1 = np.random.normal(mu1[0], sigma1[0], [N, 1])
    for i in range(1, col):
```

```
        v0 = np.random.normal(mu0[i], sigma0[i], [N, 1])
        class0 = np.hstack((class0, v0))

        v1 = np.random.normal(mu1[i], sigma1[i], [N, 1])
        class1 = np.hstack((class1, v1))

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()                    # ravel позволяет сделать массив плоским – одномерным,
размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
    arr = np.arange(2*N)                    # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1



def nonlinear_dataset_13(cen0, cen1, radii0, radii1, N):
    col = len(cen0)
    theta = 2 * np.pi * np.random.rand(N)
    theta = theta[:, np.newaxis]

    class0 = np.empty((N, col))
    class1 = np.empty((N, col))

    r = radii0[0] + np.random.rand(N)
    r = r[:, np.newaxis]
```

```
    class0[:, 0] = (r * np.sin(theta) + cen0[0]).flatten()

    r = radii1[0] + np.random.rand(N)
    r = r[:, np.newaxis]
    class1[:, 0] = (r * np.sin(theta) + cen1[0]).flatten()

    for i in range(1, col):
        r = radii0[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class0[:, i] = (r * np.cos(theta) + cen0[i]).flatten()

        r = radii1[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class1[:, i] = (r * np.cos(theta) + cen1[i]).flatten()

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()                    # ravel позволяет сделать массив плоским – одномерным,
размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
    arr = np.arange(2*N)                               # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1


def generate_dataset_A(N: int):
```

```
    mu0 = [0, 2, 3]
    mu1 = [3, 5, 1]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_B(N: int):
    mu0 = [3, 4, 3]
    mu1 = [3, 5, 2]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_C(N: int):
    cen0 = [0, 0, 0]
    cen1 = [0, 0, 0]
    radii0 = [6, 1, 2]
    radii1 = [2, 6, 1]

    return nonlinear_dataset_13(cen0, cen1, radii0, radii1, N)
```

2. *Пояснения к коду:*

   lab4.py:

   Импорт библиотек:

```
import numpy as np
import matplotlib.pyplot as plt

from NeuralNetwork import NeuralNetwork
import data_generator as dg
```

Генерация данных:

```
N = 1000

X, Y, class0, class1 = dg.generate_dataset_A(N)
Y = np.reshape(Y, [2000, 1])
```

Инициализация нейросети:

```
n_neuro = 4
NN = NeuralNetwork(X, Y, n_neuro) # инициализируем сетку на наших данных
```

Обучение:

```
# Accuracy and loss while training
N_epoch = 70
accuracy = np.zeros(N_epoch)
loss = np.zeros(N_epoch)
for i in range(N_epoch):
    pred_proba = NN.feedforward()
    pred = pred_proba >= 0.5
    accuracy[i] = sum(pred == Y) / len(pred)
    loss[i] = np.mean(np.square(Y - NN.feedforward()))

    print("Iteration #" + str(i) + ":")
    print("Accuracy: " + str(accuracy[i]))
    print("Loss: " + str(loss[i]))
    print()
    NN.train(X, Y) # обучение сети
```

Сохранение весов в переменную:

```
weigths = [NN.weights1, NN.weights2]
```

Вывод графиков:

```
plt.plot(loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss")
plt.savefig("loss.png")
plt.show()

plt.plot(accuracy)
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Accuracy")
plt.savefig("accuracy.png")
plt.show()
```

Определение оптимального числа нейронов:

```
# Accuracy when changing neurons number
max_neuro_n = 20
accuracy = np.zeros(max_neuro_n)
loss = np.zeros(max_neuro_n)
for i in range(1, max_neuro_n):
    NN_1 = NeuralNetwork(X, Y, i)
    for j in range(N_epoch):
        NN_1.train(X, Y) # обучение сети

    pred_proba = NN_1.feedforward()
    pred = pred_proba >= 0.5
    accuracy[i] = sum(pred == Y) / len(pred)
```

Вывод графиков:

```
plt.plot(accuracy)
plt.xlabel("Number of neurons")
plt.ylabel("Accuracy")
plt.title("Accuracy")
plt.savefig("accuracy_neuron_n.png")
plt.show()
```

Точность:

```
pred_proba = NN.feedforward()
pred = pred_proba >= 0.5

acc = sum(pred == Y) / len(pred)
print("Accuracy:", acc[0])
```

Файл data_generator.py:

Функция norm_dataset:

Генерирует данные для двух классов на основе нормального распределения.

```
def norm_dataset(mu,sigma,N):
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]

    col = len(mu0)                          # количество столбцов-признаков – длина массива средних

    class0 = np.random.normal(mu0[0], sigma0[0], [N, 1])    # инициализируем первый столбец (в Python нумерация от 0)
    class1 = np.random.normal(mu1[0], sigma1[0], [N, 1])
    for i in range(1, col):
        v0 = np.random.normal(mu0[i], sigma0[i], [N, 1])
        class0 = np.hstack((class0, v0))

        v1 = np.random.normal(mu1[i], sigma1[i], [N, 1])
        class1 = np.hstack((class1, v1))

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()             # ravel позволяет сделать массив плоским – одномерным, размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
```

```
    arr = np.arange(2*N)                    # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

Функция nonlinear_dataset_13:

Генерирует нелинейные данные для двух классов на основе полярных координат.

```
def nonlinear_dataset_13(cen0, cen1, radii0, radii1, N):
    col = len(cen0)
    theta = 2 * np.pi * np.random.rand(N)
    theta = theta[:, np.newaxis]

    class0 = np.empty((N, col))
    class1 = np.empty((N, col))

    r = radii0[0] + np.random.rand(N)
    r = r[:, np.newaxis]
    class0[:, 0] = (r * np.sin(theta) + cen0[0]).flatten()

    r = radii1[0] + np.random.rand(N)
    r = r[:, np.newaxis]
    class1[:, 0] = (r * np.sin(theta) + cen1[0]).flatten()

    for i in range(1, col):
        r = radii0[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class0[:, i] = (r * np.cos(theta) + cen0[i]).flatten()

        r = radii1[i] + np.random.rand(N)
        r = r[:, np.newaxis]
        class1[:, i] = (r * np.cos(theta) + cen1[i]).flatten()

    Y1 = np.ones((N, 1), dtype=bool)
    Y0 = np.zeros((N, 1), dtype=bool)
```

```
    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()                # ravel позволяет сделать массив плоским – одномерным, размера (N,)

    # перемешиваем данные
    rng = np.random.default_rng()
    arr = np.arange(2*N)                           # индексы для перемешивания
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

Функции generate_dataset_A, generate_dataset_B, generate_dataset_C:

Вызывают norm_dataset или nonlinear_dataset_13 с конкретными параметрами для генерации данных. generate_dataset_A и generate_dataset_B генерируют линейно разделимые данные. generate_dataset_C генерирует нелинейно разделимые данные.

```
def generate_dataset_A(N: int):
    mu0 = [0, 2, 3]
    mu1 = [3, 5, 1]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_B(N: int):
    mu0 = [3, 4, 3]
    mu1 = [3, 5, 2]
    sigma0 = [2, 1, 2]
    sigma1 = [1, 2, 1]

    mu = [mu0, mu1]
```

```
    sigma = [sigma0, sigma1]
    return norm_dataset(mu, sigma, N)


def generate_dataset_C(N: int):
    cen0 = [0, 0, 0]
    cen1 = [0, 0, 0]
    radii0 = [6, 1, 2]
    radii1 = [2, 6, 1]

    return nonlinear_dataset_13(cen0, cen1, radii0, radii1, N)
```

Файл NeuralNetwork.py:

```
import numpy as np
```

Сигмоида:

```
def sigmoid(Z):
    return 1 / (1 + np.exp(-Z))
```

Производная сигмоиды:

```
def sigmoid_derivative(p):
    return p * (1 - p)
```

Нейросеть:

```
class NeuralNetwork:
    def __init__(self, x, y, n_neuro):
        self.input = x
        n_inp = self.input.shape[1] # кол-во входов
        # инициализация весов рандомными значениями
        self.weights1 = np.random.rand(n_inp, n_neuro)
        self.weights2 = np.random.rand(n_neuro, 1)
        self.y = y
        self.output = np.zeros(y.shape)
```

Вычисление выхода:

```
    def feedforward(self):
```

```
        # выходы слоёв вычисляются по сигмоиде
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2
```

Коррекция весов:

```
    def backprop(self):
        # здесь происходит коррекция весов по известному вам из курса алгоритму
        d_weights2 = np.dot(self.layer1.T, 2*(self.y - self.output)*sigmoid_derivative(self.output))
        d_weights1 = np.dot(
            self.input.T,
            np.dot(
                2*(self.y - self.output) * sigmoid_derivative(self.output),
                self.weights2.T
                ) * sigmoid_derivative(self.layer1)
            )

        # обновляем веса
        self.weights1 += d_weights1
        self.weights2 += d_weights2
```

Метод для тренировки модели:

```
    def train(self, X, Y):
        # весь процесс обучения прост – высчитываем выход с помощью прямого распространения, а после обновляем веса
        self.output = self.feedforward()
        self.backprop()
```

Результаты

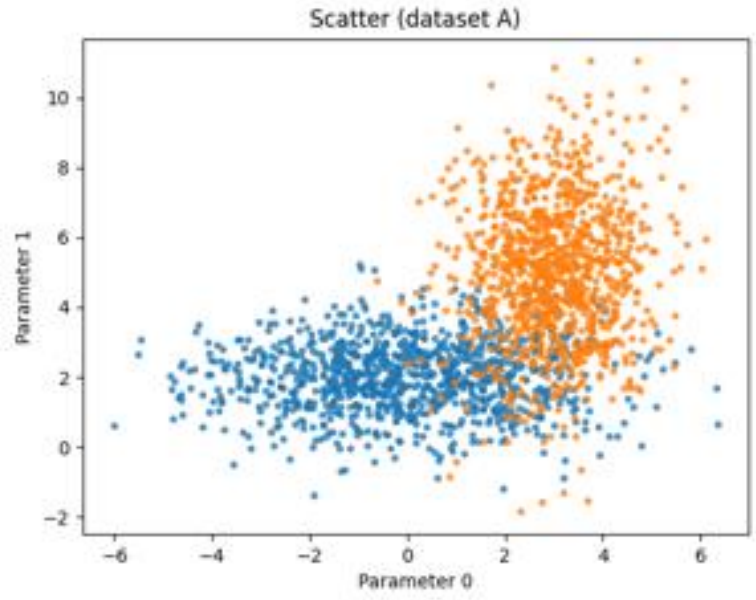

Рис. 2. Данные

Значения нейронов на каждом из слоев после обучения (4 нейрона, 70 эпох):

Слой 1:

-0.60414383	-1.66752863	0.25110639	-50.00588016
-16.94582562	-23.92644896	-24.42988905	-20.21921527
-12.56353456	-34.45795571	-17.84041849	59.99816461

Слой 2:

-172.96448616

-144.81368034

-168.65499539

-170.92733681

Точность (при обучении) варьируется при каждом запуске в диапазоне от 0.55 до 0.92.

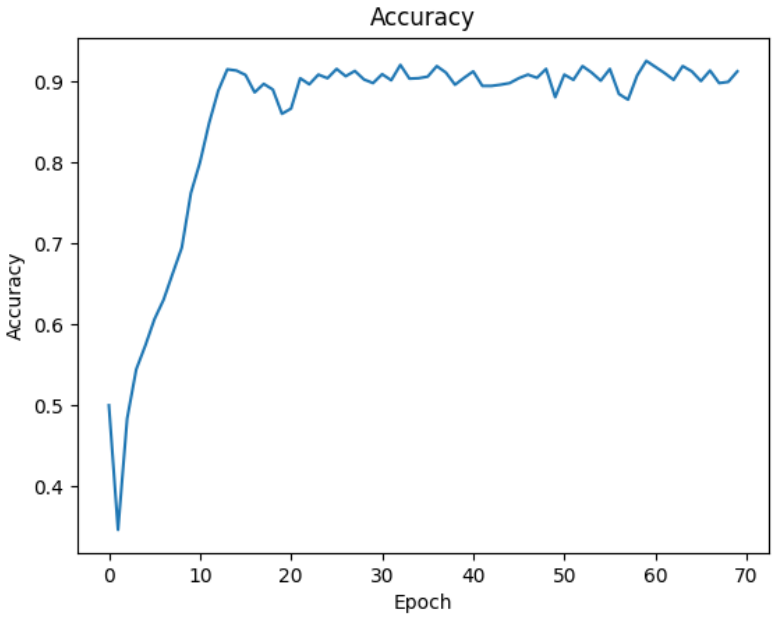

Рис. 3. Точность при обучении

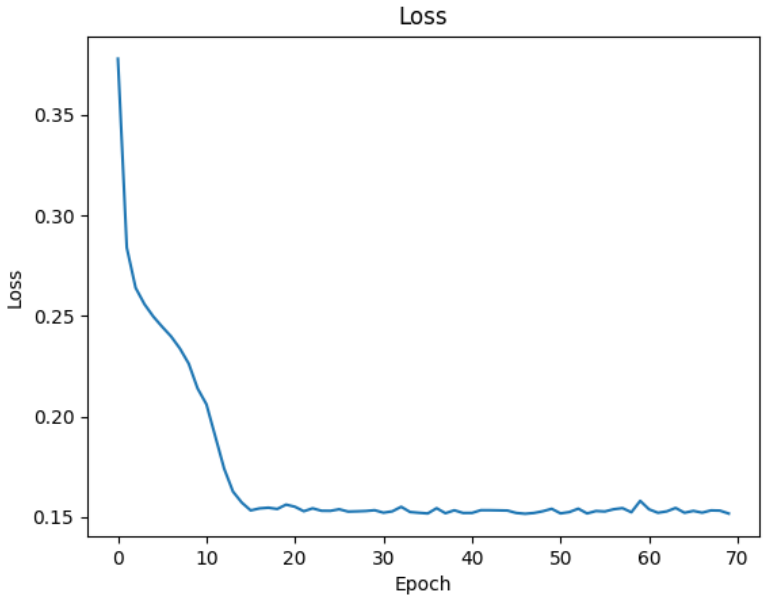

Рис. 4. Потеря при обучении

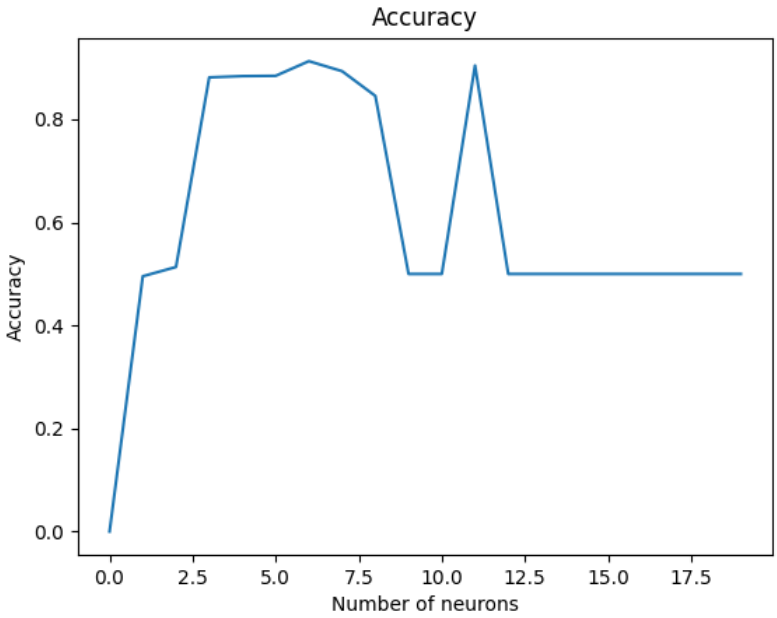

Рис. 5. Зависимость итоговой точности от числа нейронов

Оптимальное число эпох – 15 и больше. Оптимальное число нейронов – от 3 до 9.

Выводы

В ходе выполнения лабораторной работы была создана простейшая нейронная сеть на Python без использования специализированных библиотек, таких как TensorFlow или PyTorch. Были изучены основные этапы работы нейронной сети, включая прямое и обратное распространение, а также методы оценки качества модели.