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»

на тему «Нейронные сети»

Студенты гр. 1301 Семейкин С.А.

Гальченко М.А

Преподаватель Боброва Ю.О.

Санкт-Петербург

2025

Цель:

Создание простейшей нейронной сети на Python без использования специализированных библиотек.

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 (рис. 1). Полученная взвешенная сумма поступает на функцию активации, результат вычисления которой и является выходом нейрона.

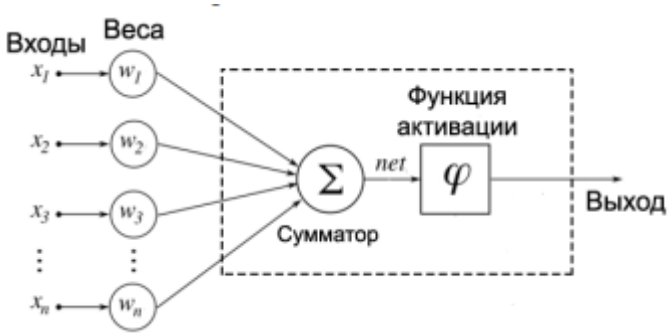

Рис. 1 – Структура нейрона

Используемые библиотеки: numpy, matplotlib

Ход работы

1. Создайте новый скрипт. Импортируйте numpy.

2. Для начала создадим ряд функций для основных вычислений, происходящих внутри сети. Активационную функцию используем сигмоиду.

3. Для расчетов также потребуется производная сигмоиды

4. Сама нейронная сеть будет представлять из себя класс с набором методов – инициализация (init), прямое распространение (feedforward), обратное распространение (backprop) и обучение (train).

5. Чтобы запустить созданную сеть сгенерируйте выборку, используя ранее написанный генератор данных. Единственное, надо будет изменить размер массива меток Y и из плоского сделать его одномерным. Т.к. в этой работе делается самый простой вариант НС, делить выборку на обучение и тест не будем.

6. В цикле обучите сеть на достаточном количестве эпох

7. Для получения предсказания необходимо вызвать метод feedforward. Заметьте, что он выдает не номер класса, а значение от 0 до 1 pred = NN.feedforward()

8. Рассчитайте итоговую точность полученной НС. *(доп. задание «со звездочкой») Конечно, оценивать точность на обучающей выборке – плохая практика, так что вы можете доработать класс NN и включить в него метод test для получения предсказания на тестовых данных.

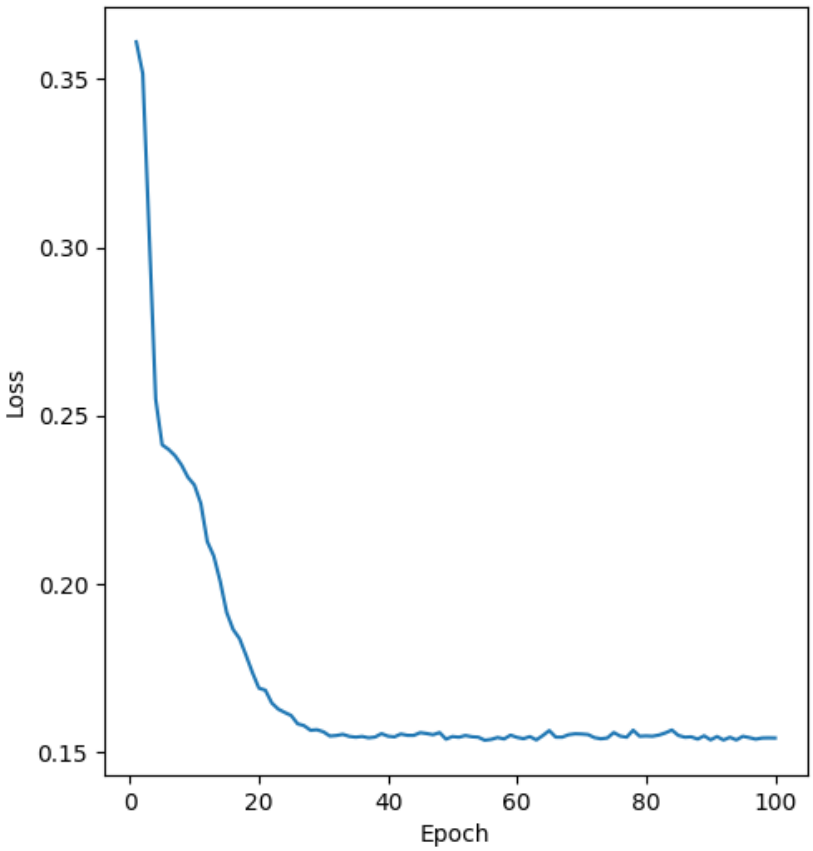

Рис. 2 – График потерь на количестве эпох

9. Вычисляя на каждой эпохе обучения значение точности и среднеквадратичной потери, постройте графики их зависимости от номера эпохи.

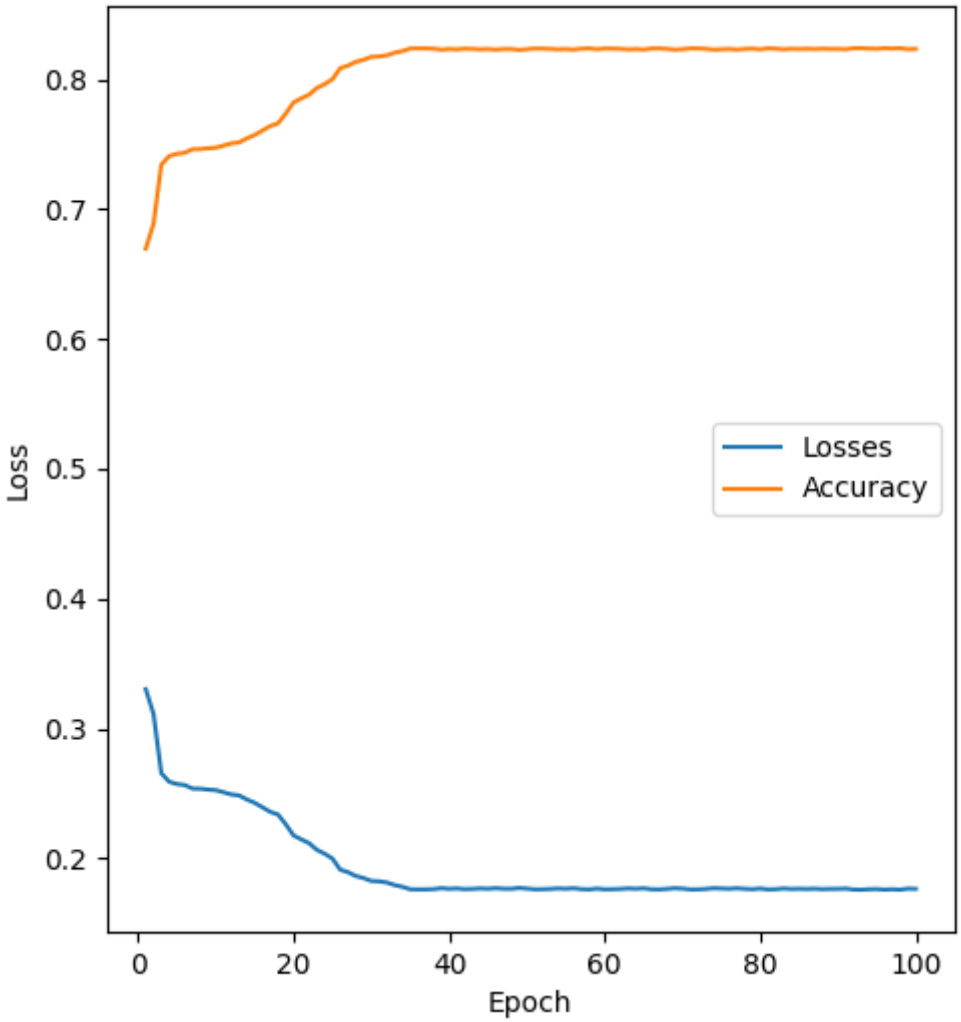

Рис. 3 – Графики зависимости от номера эпохи.

10. Проанализировав графики, подберите оптимальное число нейронов и эпох обучения.

По графику видим, что он уходит вниз примерно на отрезке от 0 до 35. Соответственно, считаем, что итоговая точность полученной нейронной сети находится в точке 35.

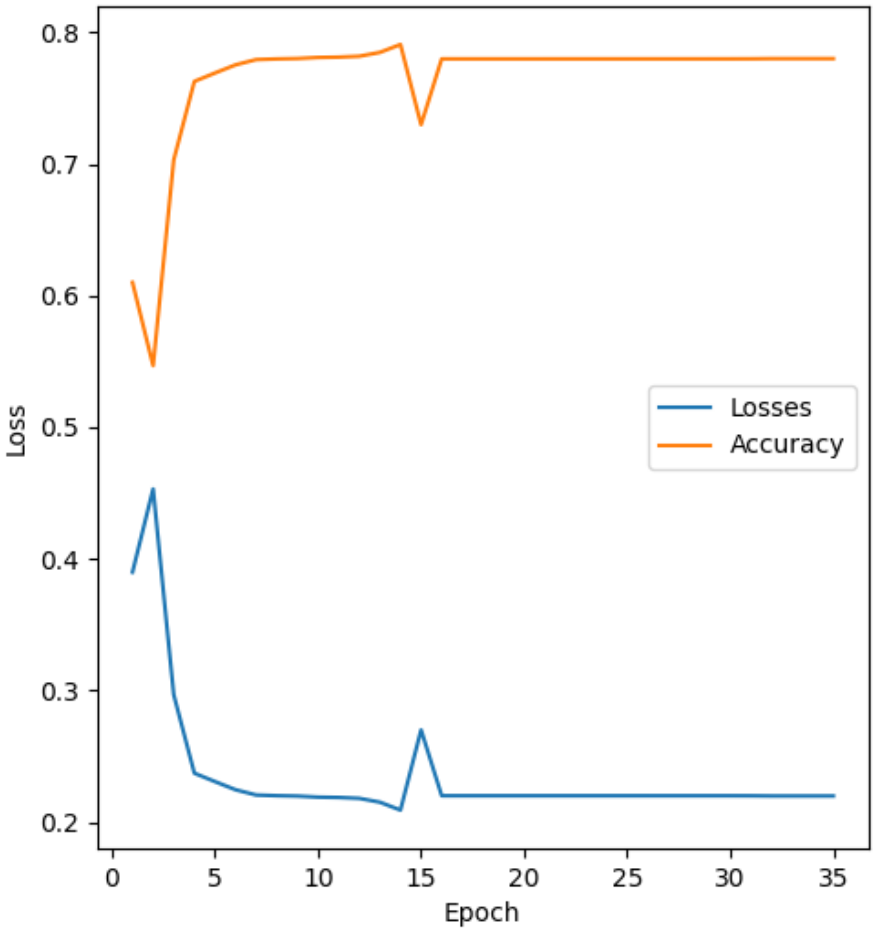

Рис. 4 – Графики зависимости от номера эпохи.

Как мы видим при ограничении эпох до 35, наилучший результат достигается около 10-13 эпохи. Именно 10 эпох дают наилучший результат.

11. Вынесите в отдельную переменную значения весов нейронов на каждом из слоев.

self.weights_history.append((self.weights1, self.weights2))

Вывод:

По результатам лабораторной работы по созданию простейшей нейронной сети на Python без использования специализированных библиотек были выполнены следующие задания:

- Рассчитана итоговая точность полученной нейронной сети.
- Реализовано дополнительное задание "со звездочкой" - добавлен метод test для получения предсказаний на тестовых данных.
- Построены графики зависимости значения точности и среднеквадратичной потери от номера эпохи во время обучения нейронной сети.
- Проведен анализ графиков для определения оптимального числа эпох обучения.
- Значения весов нейронов на каждом из слоев вынесены в отдельную переменную.

Важным этапом в обучении нейронных сетей является определение оптимального числа нейронов и эпох обучения, что поможет улучшить производительность нейронной сети.

Листинг:

```
import numpy as np
from matplotlib import pyplot as plt
from lab1.data_generator import DataGenerator


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


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


class NeuralNetwork:

    def __init__(self, x, y):
        self.input = x
        n_inp = self.input.shape[1]  # кол-во входов
        n_neuro = 6  # число нейронов на главном слое
        # инициализация весов рандомными значениями
        self.weights1 = np.random.rand(n_inp, n_neuro)
        self.weights2 = np.random.rand(n_neuro, 1)
        self.weights_history = []
        self.output = np.zeros(y.shape)

    def feedforward(self, x):
        # выходы слоёв вычисляются по сигмоиде
        self.layer1 = sigmoid(np.dot(x, self.weights1))
        self.layer2 = sigmoid(np.dot(self.layer1, self.weights2))
        return self.layer2

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

        # обновляем веса
        self.weights1 += d_weights1
        self.weights2 += d_weights2
        self.weights_history.append((self.weights1, self.weights2))

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

    def test(self, X):
        print(self.weights1)
        print(self.weights2)

        output = self.feedforward(X)
        return output


mu0 = [0, 2, 3]  # параметры выборки даны для примера, задайте свои и можно
```

```
выбрать более 3х столбцов
mu1 = [3, 5, 1]
sigma0 = [2, 1, 2]
sigma1 = [1, 2, 1]
N = 1000  # число объектов класса
col = len(mu0)  # количество столбцов-признаков

mu = (mu0, mu1)
sigma = (sigma0, sigma1)
X, Y, class0, class1 = DataGenerator.norm_dataset(mu, sigma, N)

train_size = 0.7
trainCount = round(train_size * N * 2)  # *2 потому что было 2 класса
Xtrain = X[0:trainCount]
Xtest = X[trainCount:N * 2 + 1]
Ytrain = Y[0:trainCount]
Ytest = Y[trainCount:N * 2 + 1]
Ytrain = np.reshape(Ytrain, [trainCount, 1])
NN = NeuralNetwork(Xtrain, Ytrain)  # инициализируем сетку на наших данных
N_epoch = 35
losses = []

for i in range(N_epoch):
    # print("for iteration # " + str(i))
    res = NN.train(Xtrain, Ytrain)
    loss = np.mean(np.square(Ytrain - res))
    losses.append(loss)
    print("Loss:" + str(loss))

# print(NN.weights_history)
# pred = NN.feedforward(Xtrain)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
x_c = NN.test(Xtest)
print(((x_c >= 0.5).flatten() == Ytest).sum() / Ytest.size)
# print(x_c.shape)
# print(x_c)
# print(Ytest.shape)
# print(Ytest)
# print((x_c == Ytest).sum() / Ytest.size)
# print(losses)
plt.plot(range(1, N_epoch + 1), losses)
plt.plot(range(1, N_epoch + 1), [1 - e for e in losses])
plt.legend(["Losses", "Accuracy"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```