****

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

ИНСТИТУТ ИНФОРМАТИКИ И КИБЕРНЕТИКИ

Кафедра программных систем

Дисциплина

Нейронные сети

ОТЧЕТ

по лабораторной работе №1

Создание простейшей нейронной сети «Персептрон»

Студент: Соколова А.Д.

Группа: 6301-020302D

Проверил:

профессор Тюгашев А.А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара  
2025

СОДЕРЖАНИЕ

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc195207704)

[1 Исходный текст программы 4](#_Toc195207705)

[2 Протокол исполнения 6](#_Toc195207706)

[ЗАКЛЮЧЕНИЕ 8](#_Toc195207707)

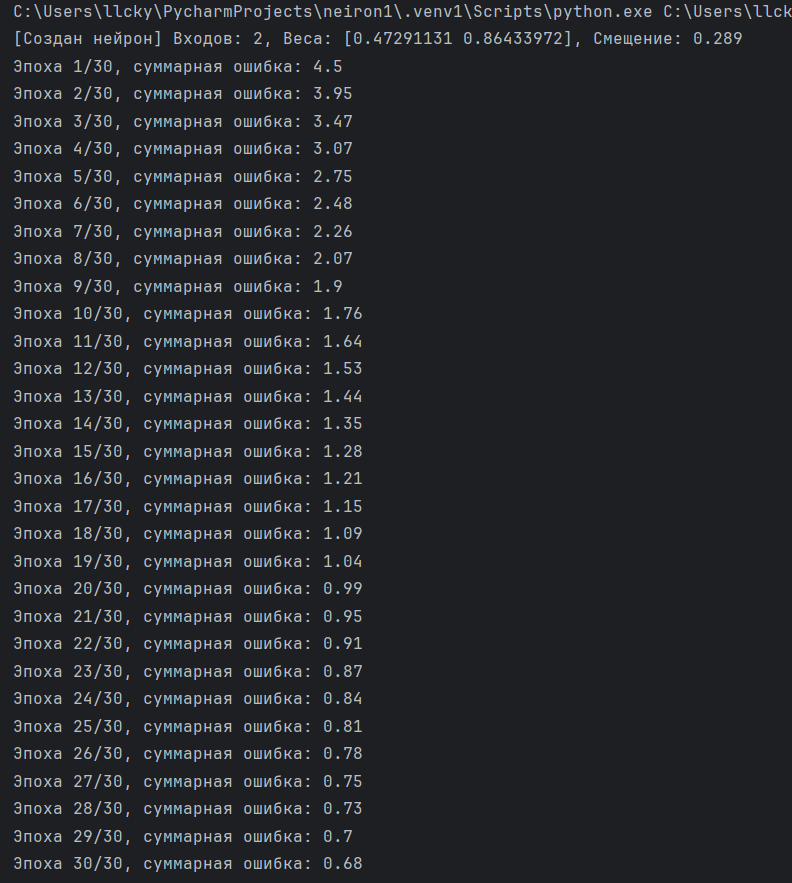
# ПОСТАНОВКА ЗАДАЧИ

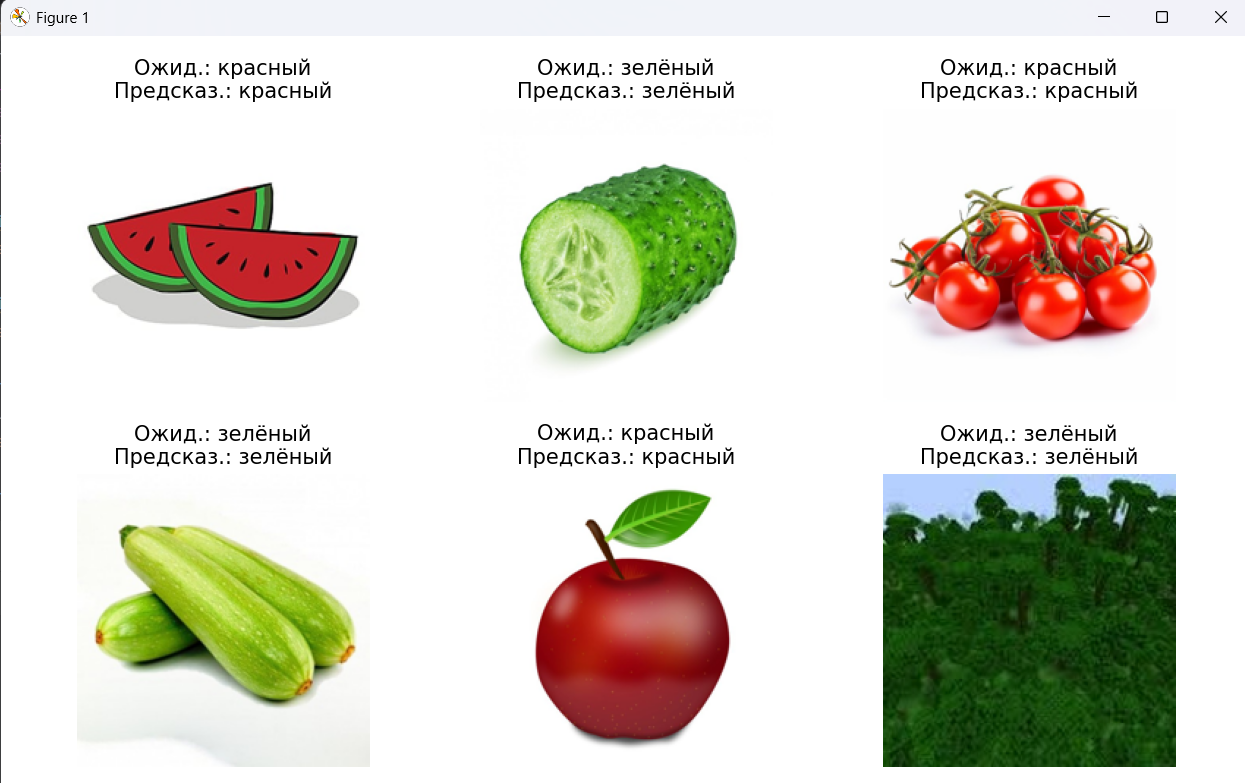
Целью данной работы является изучение основных принципов работы искусственных нейронных сетей на примере реализации формального нейрона и создания простейшей однослойной нейронной сети (персептрона) на его основе. В рамках поставленной задачи необходимо:

1. изучить основы языка Python;
2. написать функцию на Python, имитирующую одиночный формальный нейрон;
3. создать класс «формальный нейрон» на Python, на его основе создать простейшую однослойную нейронную сеть, «персептрон».
4. Исходный текст программы

from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
import random  
  
*# класс "Формальный нейрон"*class Neuron:  
 def \_\_init\_\_(self, input\_size):  
 self.weights = np.random.rand(input\_size)  
 self.bias = random.random()  
 self.learning\_rate = 0.1  
 print(f"[Создан нейрон] Входов: {input\_size}, Веса: {self.weights}, Смещение: {round(self.bias, 3)}")  
  
  
 def sigmoid(self, x):  
 return 1 / (1 + np.exp(-x))  
  
 def sigmoid\_derivative(self, x):  
 return x \* (1 - x)  
  
 def predict(self, inputs):  
 summation = np.dot(inputs, self.weights) + self.bias  
 return 1 if self.sigmoid(summation) >= 0.5 else 0  
  
 def train(self, training\_inputs, labels, epochs=30):  
 for epoch in range(epochs):  
 total\_error = 0  
 for inputs, label in zip(training\_inputs, labels):  
 summation = np.dot(inputs, self.weights) + self.bias  
 output = self.sigmoid(summation)  
 error = label - output  
 total\_error += abs(error)  
 self.weights += self.learning\_rate \* error \* inputs  
 self.bias += self.learning\_rate \* error  
 print(f"Эпоха {epoch + 1}/{epochs}, суммарная ошибка: {round(total\_error, 2)}")  
  
*# функция подготовки изображения*def prepare\_image(path):  
 img = Image.open(path).resize((200, 200))  
 img = np.array(img)[:, :, :3]  
 avg\_r = np.mean(img[:, :, 0]) / 255  
 avg\_g = np.mean(img[:, :, 1]) / 255  
 return img, np.array([avg\_r, avg\_g])  
  
*# генерация тренировочных данных*train\_data = []  
train\_labels = []  
  
*# красные изображения (метка 0)*for \_ in range(5):  
 red\_img = np.zeros((200, 200, 3), dtype=np.uint8)  
 red\_img[:, :, 0] = 255  
 avg\_r = np.mean(red\_img[:, :, 0]) / 255  
 avg\_g = np.mean(red\_img[:, :, 1]) / 255  
 train\_data.append(np.array([avg\_r, avg\_g]))  
 train\_labels.append(0)  
  
*# зелёные изображения (метка 1)*for \_ in range(5):  
 green\_img = np.zeros((200, 200, 3), dtype=np.uint8)  
 green\_img[:, :, 1] = 255  
 avg\_r = np.mean(green\_img[:, :, 0]) / 255  
 avg\_g = np.mean(green\_img[:, :, 1]) / 255  
 train\_data.append(np.array([avg\_r, avg\_g]))  
 train\_labels.append(1)  
  
*# обучение нейрона*perceptron = Neuron(input\_size=2)  
perceptron.train(np.array(train\_data), np.array(train\_labels), epochs=30)  
  
*# проверка на внешних изображениях*image\_paths = [  
 ("red1.jpg", "красный"),  
 ("green1.jpg", "зелёный"),  
 ("red2.jpg", "красный"),  
 ("green2.jpeg", "зелёный"),  
 ("red3.png", "красный"),  
 ("green3.jpeg", "зелёный")  
]  
  
fig, axes = plt.subplots(2, 3, figsize=(10, 6))  
  
for idx, (img\_path, expected\_label) in enumerate(image\_paths):  
 img, features = prepare\_image(img\_path)  
 prediction = perceptron.predict(features)  
 predicted\_label = "красный" if prediction == 0 else "зелёный"  
  
 ax = axes[idx // 3, idx % 3]  
 ax.imshow(img)  
 ax.set\_title(f"Ожид.: {expected\_label}\nПредсказ.: {predicted\_label}")  
 ax.axis('off')  
  
plt.tight\_layout()  
plt.show()

1. Протокол исполнения

  
Рисунок 1 – Эпохи и суммарные ошибки в каждой из них

  
Рисунок **2** –**В**ывод изображений из тестовой выборки с указанием меток

# ЗАКЛЮЧЕНИЕ

В процессе выполнения работы были рассмотрены основы языка Python, а затем с помощью среды Google Colab и языка Python был разработан программный модуль, имитирующий работу формального нейрона, и создан класс персептрона, представляющий собой простейшую однослойную нейронную сеть.

Полученные результаты показали, что даже простейший однослойный персептрон способен решать практические задачи классификации, если правильно подобрать входные данные и провести обучение.