****

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 №2

Создание простейшей нейронной сети  
«Глубокое обучение самостоятельно запрограммированной нейросети»

Студент: Соколова А.Д.

Группа: 6301-020302D

Проверил:

профессор Тюгашев А.А.

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

Самара  
2025

СОДЕРЖАНИЕ

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc195471373)

[1 Исходный текст программы 4](#_Toc195471374)

[2 Протокол исполнения 6](#_Toc195471375)

[ЗАКЛЮЧЕНИЕ 7](#_Toc195471376)

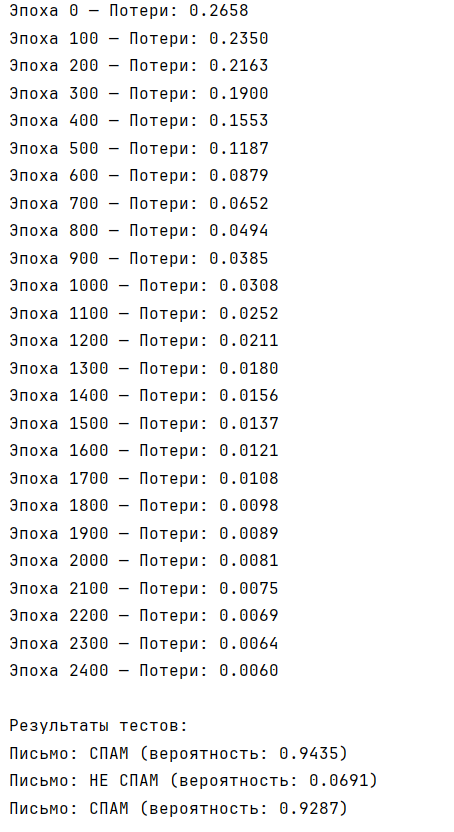
# ПОСТАНОВКА ЗАДАЧИ

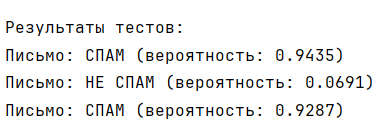
Целью данной работы является построение и обучение простой нейронной сети для решения задачи бинарной классификации — определения, является ли электронное письмо спамом, на основе двух признаков: количества слов и количества ссылок в письме. В рамках поставленной задачи необходимо:

1. реализовать модель нейрона с сигмоидной функцией активации;
2. вручную реализовать процесс прямого и обратного распространения ошибки (backpropagation);
3. обучить модель на нормализованном наборе данных с использованием стохастического градиентного спуска (SGD);
4. провести тестирование обученной нейронной сети на новых примерах.
5. Исходный текст программы

import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
def deriv\_sigmoid(x):  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
def mse\_loss(y\_true, y\_pred):  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
class SpamDetector:  
 def \_\_init\_\_(self):  
 self.w1 = np.random.normal()  
 self.w2 = np.random.normal()  
 self.w3 = np.random.normal()  
 self.w4 = np.random.normal()  
 self.w5 = np.random.normal()  
 self.w6 = np.random.normal()  
 self.b1 = np.random.normal()  
 self.b2 = np.random.normal()  
 self.b3 = np.random.normal()  
  
 def feedforward(self, x):  
 h1 = sigmoid(self.w1 \* x[0] + self.w2 \* x[1] + self.b1)  
 h2 = sigmoid(self.w3 \* x[0] + self.w4 \* x[1] + self.b2)  
 o1 = sigmoid(self.w5 \* h1 + self.w6 \* h2 + self.b3)  
 return o1  
  
 def train(self, data, all\_y\_trues):  
 learn\_rate = 0.01   
 epochs = 5000   
  
 for epoch in range(epochs):  
 for x, y\_true in zip(data, all\_y\_trues):  
   
 sum\_h1 = self.w1 \* x[0] + self.w2 \* x[1] + self.b1  
 h1 = sigmoid(sum\_h1)  
   
 sum\_h2 = self.w3 \* x[0] + self.w4 \* x[1] + self.b2  
 h2 = sigmoid(sum\_h2)  
   
 sum\_o1 = self.w5 \* h1 + self.w6 \* h2 + self.b3  
 o1 = sigmoid(sum\_o1)  
 y\_pred = o1  
  
   
 d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)  
  
   
 d\_ypred\_d\_w5 = h1 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_w6 = h2 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_b3 = deriv\_sigmoid(sum\_o1)  
  
   
 d\_ypred\_d\_h1 = self.w5 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_h2 = self.w6 \* deriv\_sigmoid(sum\_o1)  
  
   
 d\_h1\_d\_w1 = x[0] \* deriv\_sigmoid(sum\_h1)  
 d\_h1\_d\_w2 = x[1] \* deriv\_sigmoid(sum\_h1)  
 d\_h1\_d\_b1 = deriv\_sigmoid(sum\_h1)  
  
   
 d\_h2\_d\_w3 = x[0] \* deriv\_sigmoid(sum\_h2)  
 d\_h2\_d\_w4 = x[1] \* deriv\_sigmoid(sum\_h2)  
 d\_h2\_d\_b2 = deriv\_sigmoid(sum\_h2)  
  
 self.w1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w1  
 self.w2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w2  
 self.b1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_b1  
  
 self.w3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w3  
 self.w4 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w4  
 self.b2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_b2  
  
 self.w5 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w5  
 self.w6 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w6  
 self.b3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_b3  
  
  
 if epoch % 100 == 0:  
 y\_preds = np.apply\_along\_axis(self.feedforward, 1, data)  
 loss = mse\_loss(all\_y\_trues, y\_preds)  
 print(f"Эпоха {epoch} — Потери: {loss:.4f}")  
  
*# данные для обучения*data = np.array([  
 [10, 500], [1, 2000], [5, 300], [3, 1500], [8, 400],  
 [12, 350], [0, 1800], [7, 400], [2, 1700], [9, 300],  
 [4, 1600], [6, 250], [11, 200], [1, 1900], [5, 500]  
])  
all\_y\_trues = np.array([1,0,1,0,1,1,0,1,0,1,0,1,1,0,1])  
  
*# нормализация данных*scaler = MinMaxScaler()  
data\_normalized = scaler.fit\_transform(data)  
  
*# обучение*network = SpamDetector()  
network.train(data\_normalized, all\_y\_trues)  
  
*# тестирование*test\_emails = [  
 np.array([8, 400]),   
 np.array([2, 1800]),   
 np.array([6, 350])  
]  
test\_emails\_normalized = scaler.transform(test\_emails)  
  
print("\nРезультаты тестов:")  
for email in test\_emails\_normalized:  
 prob = network.feedforward(email)  
 print(f"Письмо: {'СПАМ' if prob > 0.5 else 'НЕ СПАМ'} (вероятность: {prob:.4f})")

1. Протокол исполнения

  
Рисунок 1 – Эпохи и потери в каждой из них

  
Рисунок 2 – Вывод результатов тестов

# ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы были изучены принципы построения и обучения нейронных сетей. Была реализована модель формального нейрона с сигмоидной функцией активации на языке Python с использованием библиотеки NumPy.

Нейронная сеть обучалась на наборе данных, где входными признаками были количество слов и количество ссылок в письме, а целью — предсказание вероятности того, что письмо является спамом.