МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №1 на тему:

«Исследование однослойных нейронных сетей на примере моделирования булевых выражений»

Вариант 4

Преподаватель:
Коннова Н.С.

Студент:
Куликова А.В.

Группа:
ИУ8-21М

Цель работы

Исследовать функционирование нейронной сети (НС) на базе нейрона с нелинейной функцией активации и ее обучение по правилу Видроу-Хоффа.

Постановка задачи

Получить модель булевой функции (БФ) на основе однослойной НС (единичный нейрон) с двоичными вхо¬дами x1, x2, x3, x4 ∈ {0,1}, единичным входом смещения x0 = 1, синаптическими весами w0, w1, w2, w3, w4, выходом y ∈ {0,1} и заданной нелинейной функцией активации (ФА) f : R → (0, 1).

Для заданной БФ реализовать обучение НС с использованием:

1) всех комбинаций переменных x1, x2, x3, x4;
2) части возможных комбинаций переменных x1, x2, x3, x4 остальные комбинации являются тестовыми.

№ Варианта	Моделируемая БФ	ФА*
4	$(\overline{x_1} + x_3)x_2 + x_2x_4$	1, 2

* Функции активации:

1) $f(\text{net}) = \begin{cases} 1, & \text{net} \geq 0, \\ 0, & \text{net} < 0; \end{cases}$

2) $f(\text{net}) = \frac{1}{2}\left(\frac{\text{net}}{1+|\text{net}|} + 1\right)$;

Ход работы

Таблица истинности представлена в таблице 1.

Таблица 1 – Таблица истинности

x_1	x_2	x_3	x_4	F
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**0**
1	1	0	0	**1**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**1**
0	0	0	0	**0**
0	0	0	1	**0**

0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**

Был рассмотрена функция активации 1.

Для анализа была запущена программа, где осуществлялся анализ каждого случая с дальнейшей записью наиболее подходящего результата и сохранением графика результата.

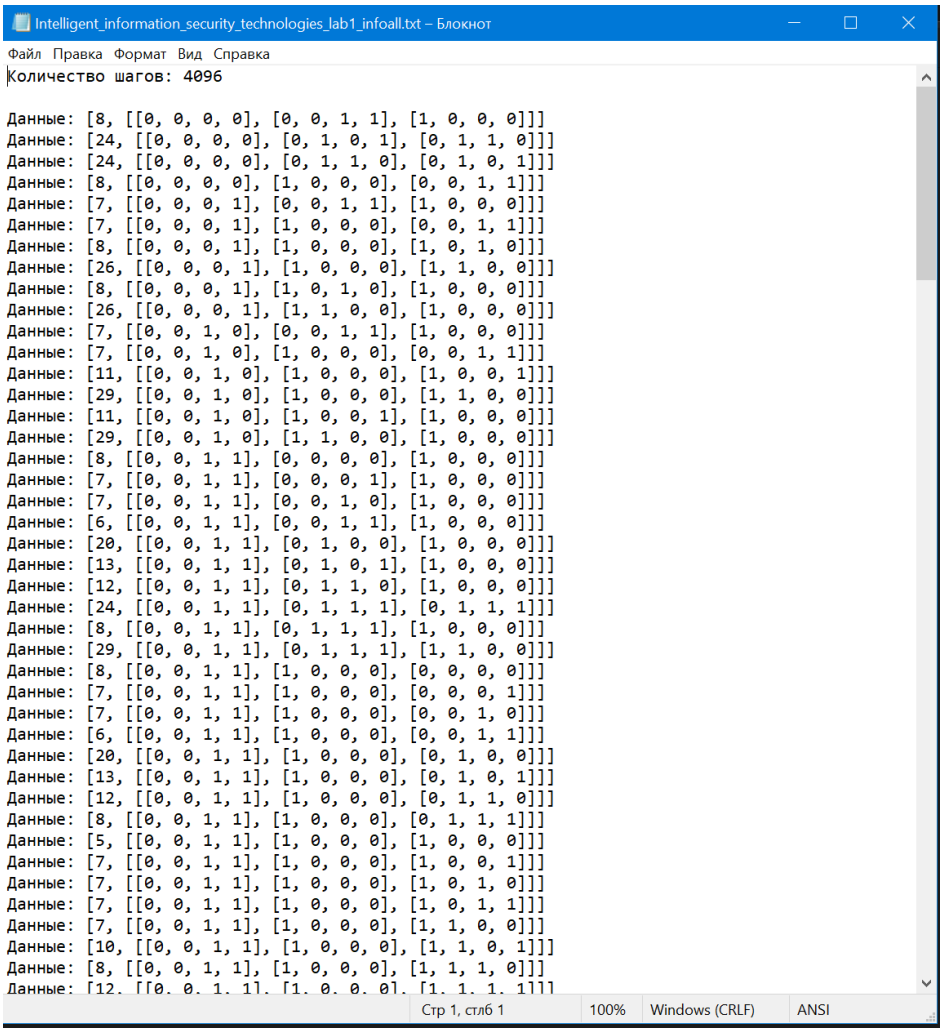

Рисунок 1 – Запись наиболее подходящего результата

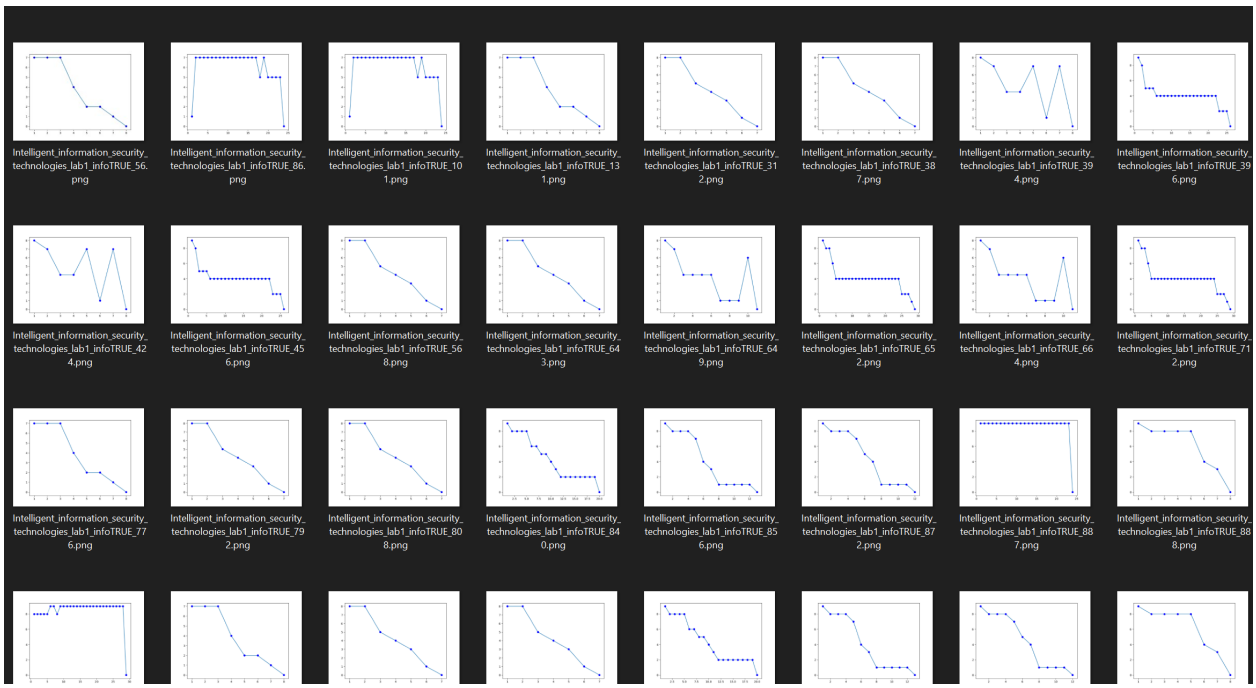

Рисунок 2 – сохранение графиков наилучшего результата

Проанализировав полученный результат сужается, что график суммарной квадратичной ошибки сужается до диапазона: [7-30].

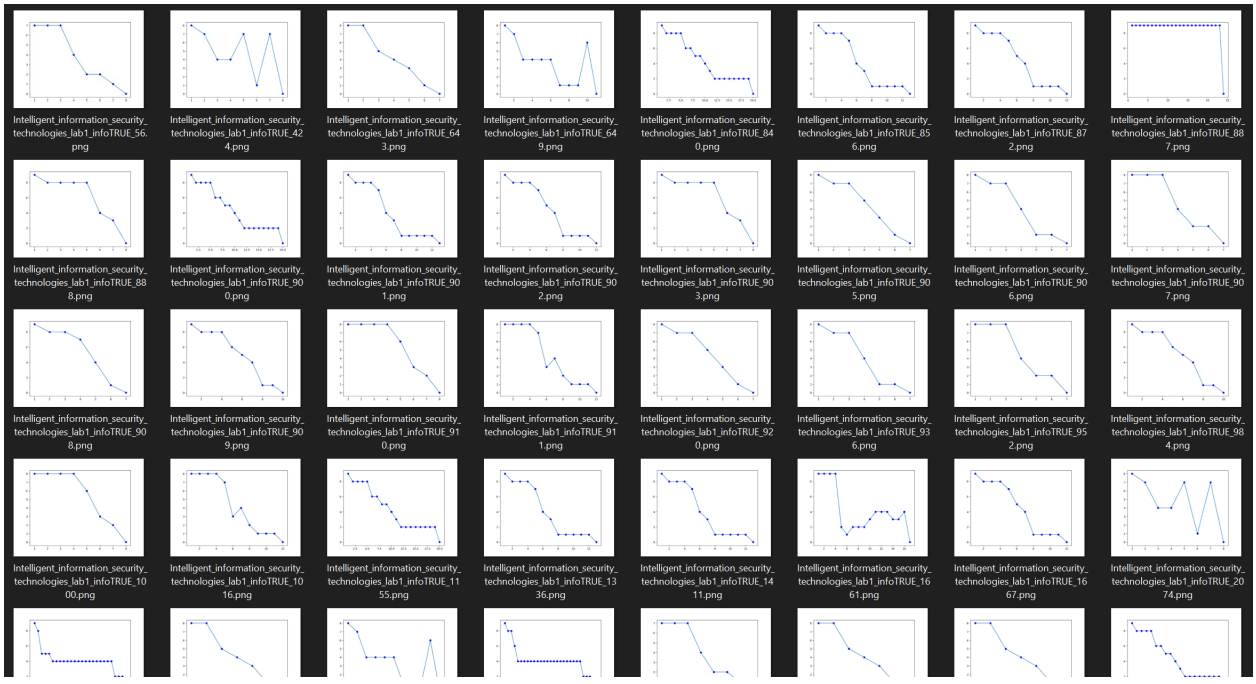

Рисунок 3 – сохранение графиков наилучшего результата

Осуществив анализ полученного результата, были выбраны следующие графики суммарной квадратичной ошибки.

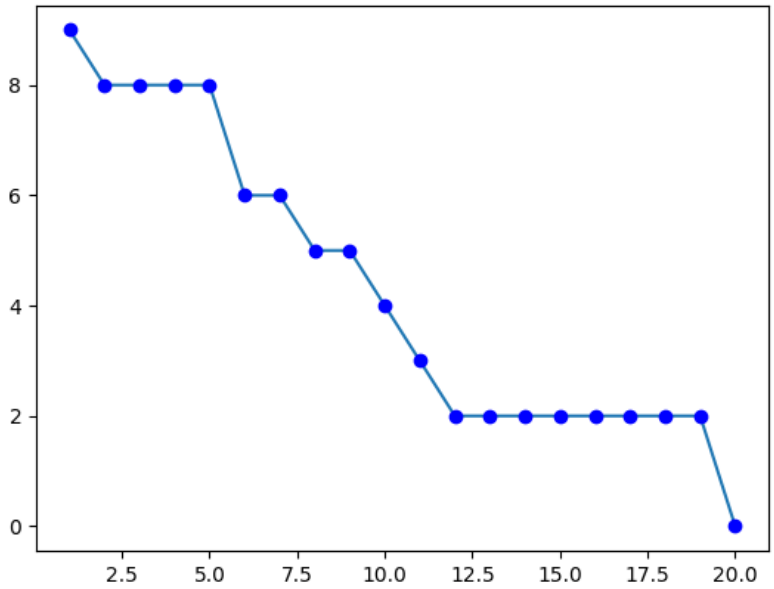

Рисунок 4 - График суммарной квадратичной ошибки для ФА 1

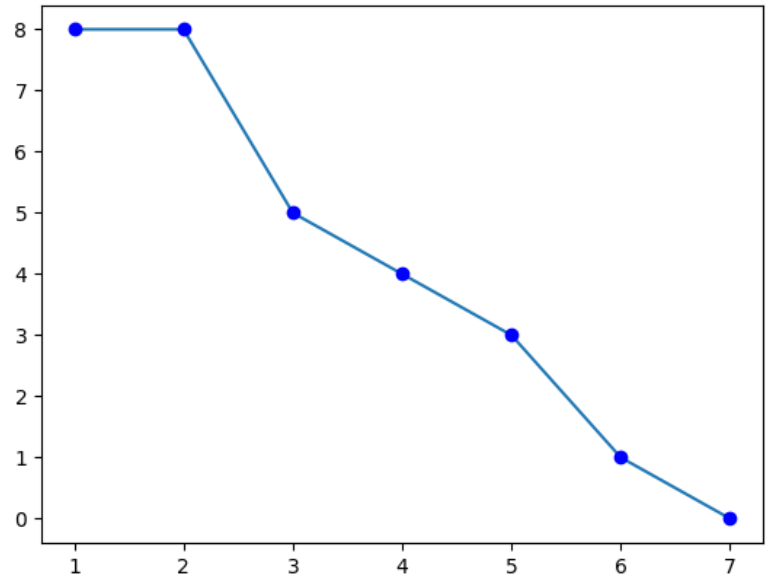

Рисунок 5 - График суммарной квадратичной ошибки для ФА 1

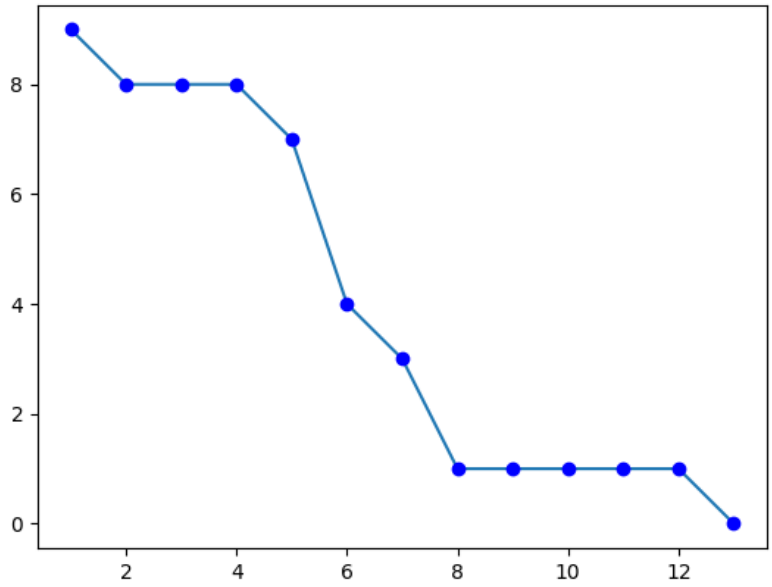

Рисунок 6 - График суммарной квадратичной ошибки для ФА 1

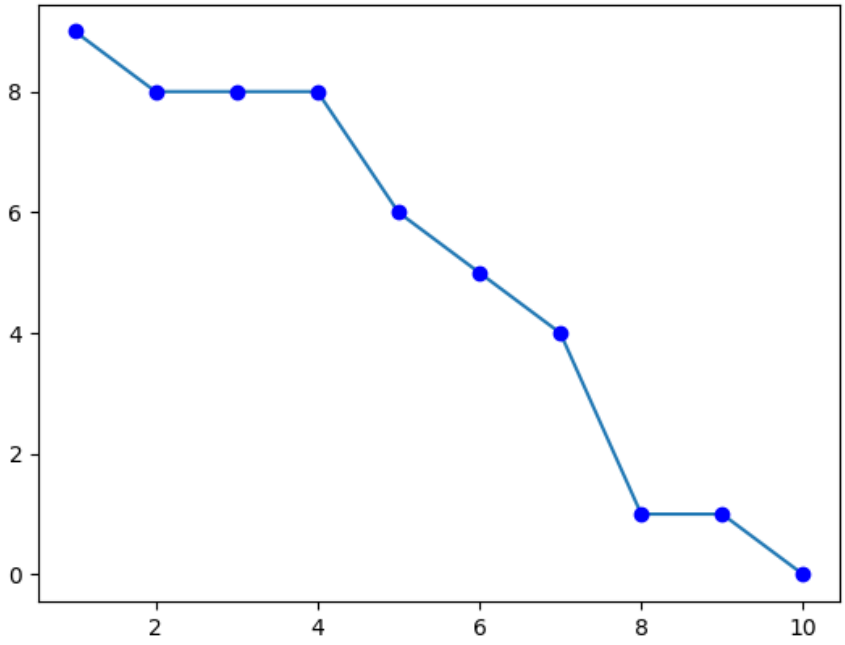

Рисунок 7 - График суммарной квадратичной ошибки для ФА 1

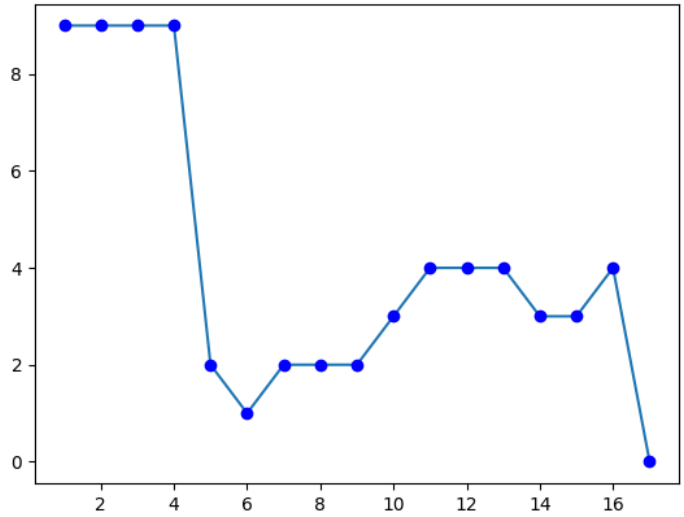

Рисунок 8 - График суммарной квадратичной ошибки для ФА 1

Был рассмотрена функция активации 2.

Для анализа была запущена программа, где осуществлялся анализ каждого случая с дальнейшей записью наиболее подходящего результата и сохранением графика результата.

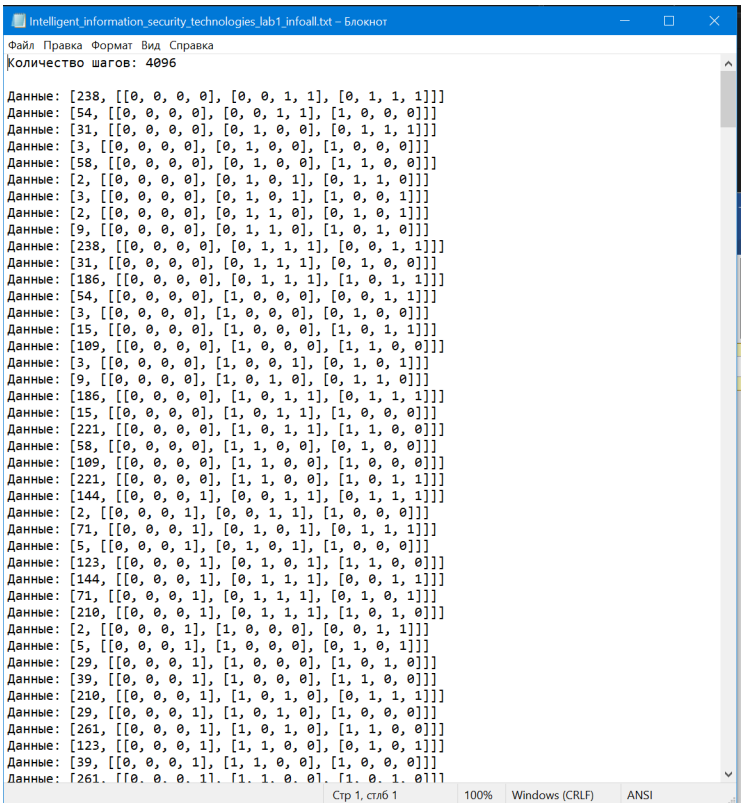

Рисунок 9 – Запись наиболее подходящего результата

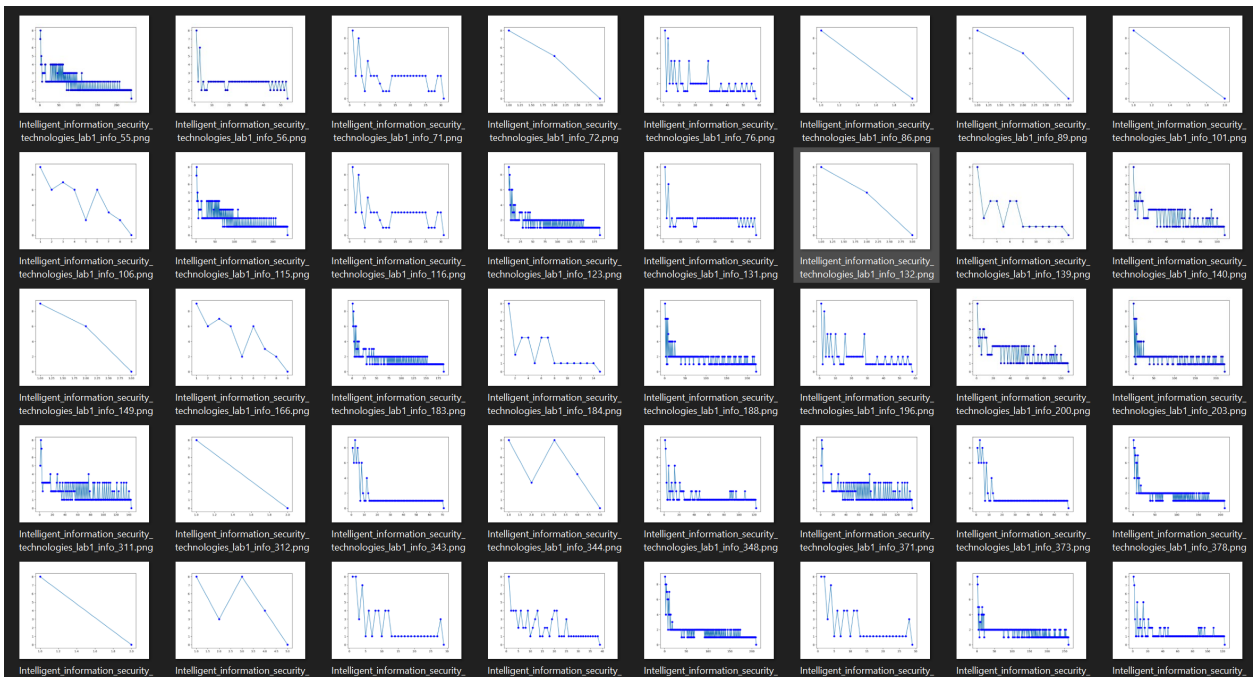

Рисунок 10 – сохранение графиков наилучшего результата

Проанализировав полученный результат сужается, что график суммарной квадратичной ошибки сужается до диапазона: [10-50].

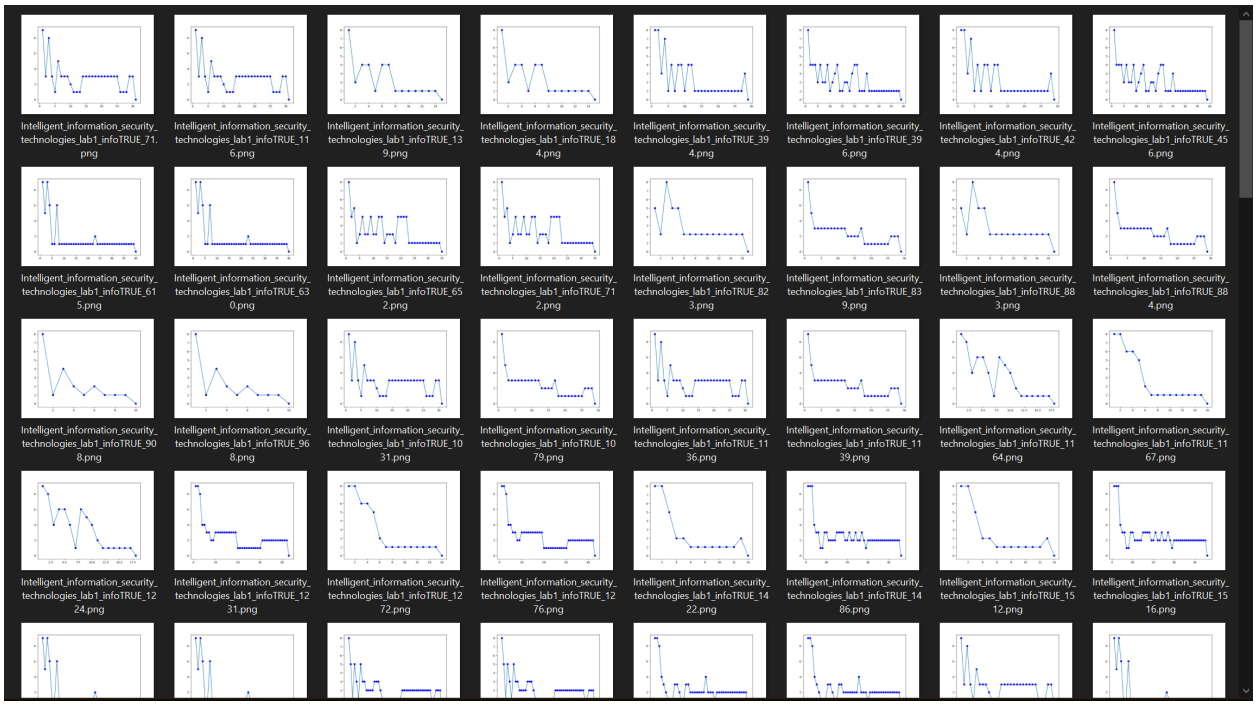

Рисунок 11 – сохранение графиков наилучшего результата

Осуществив анализ полученного результата, были выбраны следующие графики суммарной квадратичной ошибки.

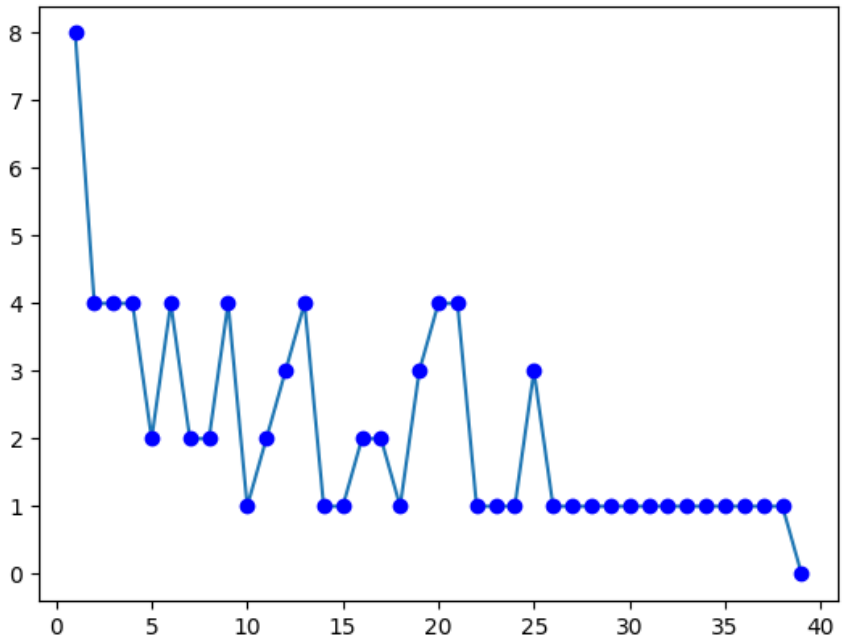

Рисунок 13 – График суммарной квадратичной ошибки для ФА 2

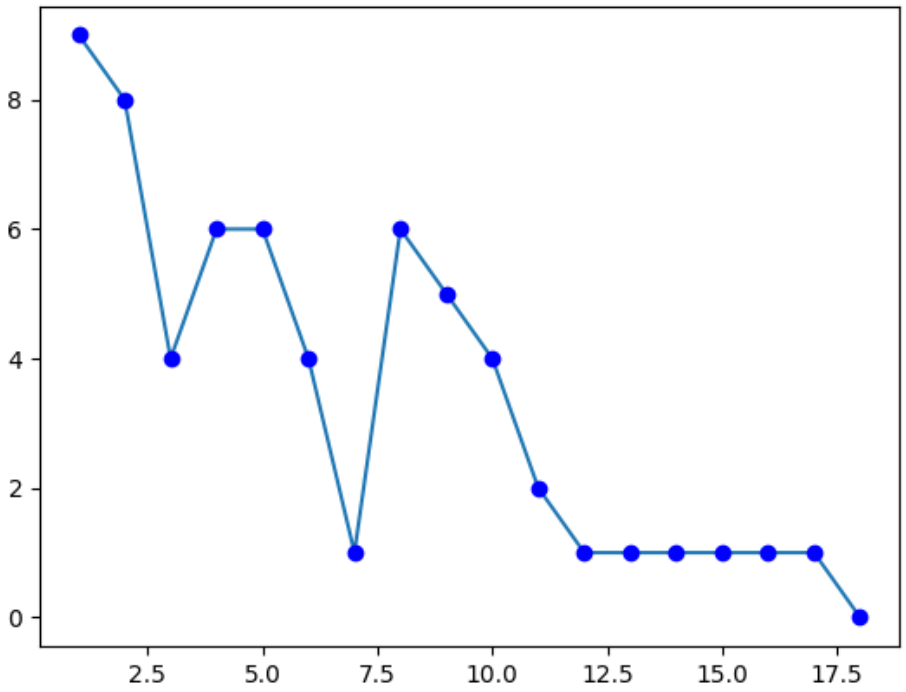

Рисунок 14 – График суммарной квадратичной ошибки для ФА 2

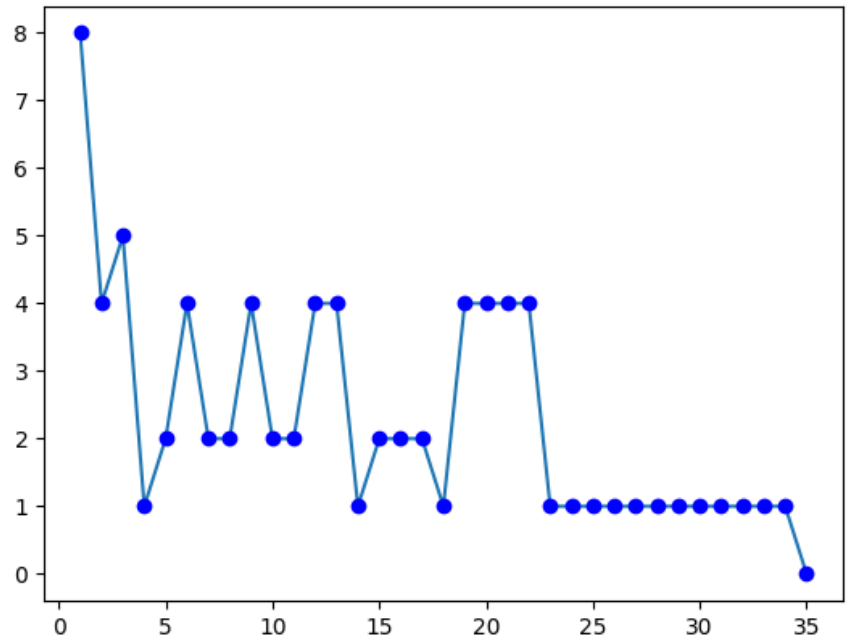

Рисунок 15 – График суммарной квадратичной ошибки для ФА 2

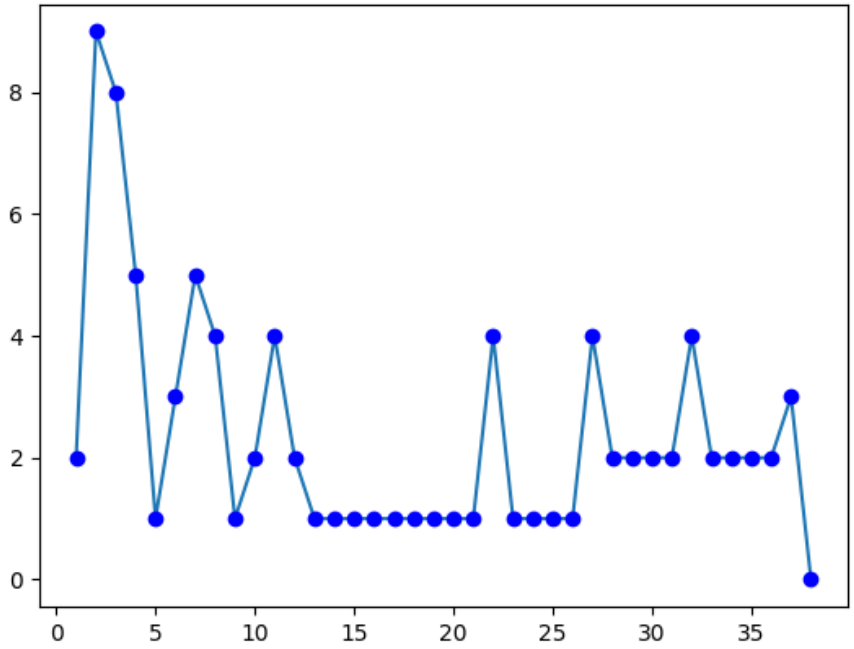

Рисунок 16 – График суммарной квадратичной ошибки для ФА 2

Выводы

В ходе выполнения лабораторной работы было изучено функционирование простейшей нейронной сети на базе нейрона с нелинейной функцией активации и проведено ее обучение по правилу Видроу-Хоффа. В ходе обучения на полных наборах было выявлено, что с использованием ФА 1 понадобилось меньше эпох, чем для обучения с использованием ФА 2.

Контрольные вопросы

1. Дайте определение персептрона и поясните алгоритм его функционирования.

Персептрон — это простая модель искусственного нейрона, разработанная Фрэнком Розенблаттом в 1957 году. Персептрон представляет собой алгоритм машинного обучения для бинарной классификации, который пытается разделить два класса данных с помощью гиперплоскости.

Алгоритм функционирования персептрона:

1. Инициализация весов (weights) и смещения (bias) нулями или случайными значениями.
2. Подача обучающего примера на вход персептрона.
3. Вычисление суммарного взвешенного входа для нейрона: weighted_sum = x1 * w1 + x2 * w2 + ... + xn * wn + bias
4. Применение функции активации (обычно ступенчатой или пороговой функции) к суммарному взвешенному входу, чтобы получить выход нейрона.
5. Сравнение полученного выхода персептрона с ожидаемым значением класса (true class).
6. Если выход нейрона не совпадает с ожидаемым классом, то обновляются веса и смещение с помощью правила обучения персептрона.
7. Повторение шагов 2-6 на протяжении обучающей выборки до тех пор, пока все примеры будут корректно классифицированы или до достижения заданного количества итераций.

Основная идея персептрона заключается в коррекции весов и смещения на основе ошибки классификации, чтобы улучшить его способность разделять классы. В итоге, персептрон может обучаться на данных и делать простые бинарные предсказания.

2. Приведите функции активации НС и их производные.

1. Сигмоидальная функция активации:

Функция: f(x) = 1 / (1 + exp(-x))

Производная: f'(x) = f(x) * (1 - f(x))

2. Гиперболический тангенс (tanh) функция активации:

Функция: f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))

Производная: f'(x) = 1 - f(x)^2

3. ReLU (Rectified Linear Unit) функция активации:

Функция: f(x) = max(0, x)

Производная: f'(x) = 1, если x > 0; 0, если x <= 0

4. Leaky ReLU функция активации:

Функция: f(x) = max(0.01x, x)

Производная: f'(x) = 1.01, если x > 0; 0.01, если x <= 0

5. Softmax функция активации (для многоклассовой классификации):

Функция: f(x_i) = exp(x_i) / sum(exp(x))

Производная: f'(x_i) = f(x_i) * (1 - f(x_i)), где f(x_i) это значение функции Softmax для класса i

6. Слой с линейной функцией активации:

Функция: f(x) = x

Производная: f'(x) = 1

3. Сформулируйте правило обучения Видроу — Хоффа.

Правило обучения Видроу — Хоффа, также известное как алгоритм обратного распространения ошибки, является одним из основных методов обучения нейронных сетей. Оно состоит из следующих шагов:

1. Пряжний (forward) проход: Входные данные подаются на вход нейронной сети, и сигнал проходит через все слои сети до выходного слоя. Для каждого нейрона вычисляется его активация на основе взвешенной суммы входных сигналов и функции активации.
2. Рассчет ошибки: Вычисляется разница между выходом нейронной сети и ожидаемым выходом (заданным целевым значением). Эта ошибка используется для дальнейшего корректировки весов.

3. Обратный (backward) проход: Ошибка распространяется обратно через сеть, начиная с выходного слоя. Для каждого нейрона вычисляется градиент функции потерь по отношению к его весам. Эти градиенты используются для корректировки весов сети.

4. Коррекция весов: Веса нейронной сети обновляются в направлении, противоположном градиенту функции потерь. Этот шаг выполняется с использованием метода градиентного спуска или его вариаций, таких как стохастический градиентный спуск или адам.

Этот процесс повторяется многократно на тренировочном наборе данных для настройки весов сети таким образом, чтобы минимизировать ошибку между прогнозируемыми и целевыми значениями.

Приложение А

Листинг программы:

```
import math
import matplotlib.pyplot as plt
import itertools

import numpy as np
import os

NORM_LEARNING = 0.3

# Создается класс Network, который содержит методы и переменные для обучения
и использования нейронной сети
class Network:
    def __init__(self, synapses: list, RBF: list, norm_learn: int):
        # Инициализация параметров нейронной сети
        self.synapses = synapses  # Весовые коэффициенты
        self.epoch_rate = 0  # Номер эпохи
        self.input_vec = input_vector()  # Входной вектор
        self.RBF = RBF  # RBF (Radial Basis Function)
        self.norm_learn = norm_learn  # Норма обучения
        self.errors = []  # Список для хранения ошибок на каждой эпохе

        # Вывод информации о созданной нейронной сети
        print("Создана нейронная сеть со следующими параметрами: ")
        print("Номер Эпохи: ", self.epoch_rate)
        print("Вектор весов: ", self.synapses)
        print("Выходной вектор: ", self.out_vec())
        print("Суммарная ошибка: ", self.err())
        print("Норма ообучения: ", self.norm_learn)

    def func_activation(self, net):
        if net >= 0:
            return 1
        else:
            return 0

    def func_activation_2(self, net):
        # return 1/2 * (net / (1 + abs(net)) + 1)
        if net >= 0:
            return 1 if net > 0 else 0
        else:
            # return 1 if 0.5 * (1/2 * (net / (1 + abs(net)) + 1)) >= 0.5
else 0
            return 0.5 * (1/2 * (net / (1 + abs(net)) + 1))


    # вычисляется радиально-базисная функция для входных данных
    def find_fi(self, inputs: list):
        fi = []
        for i in range(len(self.RBF)):
            buf = 0
            for j in range(1, 5):
                buf += (inputs[j] - self.RBF[i][j - 1]) ** 2
            fi.append(math.exp(-buf))
        return fi

    # выполняет итерацию нейронной сети для входных данных
    def itertion(self, inputs: list):  # итерация нейронной сети по входным
данным
        net = 0
```

```
        fi = self.find_fi(inputs)
        for i in range(len(fi)):
            net += self.synapses[i + 1] * fi[i]
        # return self.func_activation(net + self.synapses[0])
        return self.func_activation_2(net + self.synapses[0])

    # выполняет одну эпоху обучения нейронной сети
    def epoch_1(self, out=True):
        for i in range(len(self.input_vec)):
            self.step_learning_1((self.input_vec[i]))
        self.epoch_rate += 1
        # if out:
            # print()
            # print("Номер Эпохи: ", self.epoch_rate)
            # print("Вектор весов: ", [round(elem, 2) for elem in
self.synapses])
            # print("Выходной вектор: ", self.out_vec())
            # print("Суммарная ошибка: ", self.err())

        self.errors.append(self.err())

    # выполняет шаг обучения нейронной сети
    def step_learning_1(self, inputs: list):
        buf = function(inputs) - self.itertion(inputs)
        net = 0
        fi = self.find_fi(inputs)
        for i in range(len(self.synapses) - 1):
            net += self.synapses[i + 1] * fi[i]
        net += self.synapses[0]

        for i in range(len(self.RBF)):
            self.synapses[i + 1] += self.norm_learn * buf * fi[i]
        self.synapses[0] += self.norm_learn * buf
        return 1

    # для сброса сети, вывода результата, подсчета ошибки и поиска
оптимального
    # подмножества входных данных соответственно
    def reset_net(self):
        self.synapses = [0, 0, 0, 0]
        self.epoch_rate = 0
        self.error_count = []
        self.error_count.append(self.err())

    # для сброса сети, вывода результата, подсчета ошибки и поиска
оптимального
    # подмножества входных данных соответственно
    def out_vec(self):
        buf = []
        input_vec = input_vector()
        for i in range(len(input_vec)):
            if self.itertion(input_vec[i]) >= 0.5:
                buf.append(1)
            else:
                buf.append(0)
        return buf

    def err(self):
        err = 0
        input_vec = input_vector()
        for i in range(len(input_vec)):
            if self.out_vec()[i] != function(input_vec[i]): err += 1
        return err
```

```
# определена логическая функция, которая вычисляется на входных данных
def function(x: list):
    # (!x1 + x3) * x2 + x2 * x4
    return int(((not x[1] or x[3]) and x[2]) or (x[2] and x[4]))

# создается список всех возможных комбинаций входных значений для заданной
логической функции
def input_vector():
    out = []
    for i in range(16):
        out.append([])
        buf = i
        for j in range(4):
            out[i].insert(0, buf % 2)
            buf = int(buf / 2)
        out[i].insert(0, 1)
    return out


def main():
    X = [0]*16
    for i in range(16):
        X[i] = [0]*4
    X = ([0, 0, 0, 0],
         [0, 0, 0, 1],
         [0, 0, 1, 0],
         [0, 0, 1, 1],
         [0, 1, 0, 0],
         [0, 1, 0, 1],
         [0, 1, 1, 0],
         [0, 1, 1, 1],
         [1, 0, 0, 0],
         [1, 0, 0, 1],
         [1, 0, 1, 0],
         [1, 0, 1, 1],
         [1, 1, 0, 0],
         [1, 1, 0, 1],
         [1, 1, 1, 0],
         [1, 1, 1, 1])

    Net_true_array = []
    Net_true_array_size_start = 0
    Net_true_array_size_end = 0

    for i1 in range(16):
        for i2 in range(16):
            for i3 in range(16):
                Net_true_array_size_end += 1

    for i1 in range(16):
        for i2 in range(16):
            for i3 in range(16):
                my_file =
open("result\_txt\Intelligent_information_security_technologies_lab1_info_%s.
txt" % (Net_true_array_size_start), "w")
                RFB_NETS =[X[i1], X[i2], X[i3]]

                print("Шаг %s/%s" % (Net_true_array_size_start,
Net_true_array_size_end))
                print(RFB_NETS)

                Net = Network([0, 0, 0, 0], RFB_NETS, NORM_LEARNING)
                i = 0
```

```
                while Net.err() > 0 and Net.epoch_rate <= 1000:
                    Net.epoch_1()
                    i += 1
                if Net.epoch_rate < 1000:
                    print("Результат записи: успешно")
                    Net_true_array.append([Net.epoch_rate, RFB_NETS])
                    my_file.write("Шаг: %s\nДанные (Net.epoch_rate,
RFB_NETS): %s, %s" % (Net_true_array_size_start, Net.epoch_rate, RFB_NETS))
                    my_file.close()
                else:
                    print("Результат записи: ошибка")
                    my_file.close()

os.remove("result\_txt\Intelligent_information_security_technologies_lab1_inf
o_%s.txt" % (Net_true_array_size_start))

                if Net.epoch_rate < 1000:
                    x = [i for i in range(1, len(Net.errors) + 1)]
                    y = Net.errors
                    plt.plot(x, y)
                    plt.plot(x, y, 'bo')

plt.savefig('result\_png\Intelligent_information_security_technologies_lab1_i
nfo_%s.png' % (Net_true_array_size_start))
                    plt.close()

                if Net.epoch_rate < 1000 and Net.epoch_rate >= 7:
                    x = [i for i in range(1, len(Net.errors) + 1)]
                    y = Net.errors
                    plt.plot(x, y)
                    plt.plot(x, y, 'bo')

plt.savefig('result\_true_png\Intelligent_information_security_technologies_l
ab1_infoTRUE_%s.png' % (Net_true_array_size_start))
                    plt.close()

                Net_true_array_size_start += 1

    my_file =
open("result\Intelligent_information_security_technologies_lab1_infoall.txt",
"w")
    my_file.write("Количество шагов: %s\n" % (Net_true_array_size_end))
    for Net_true_array_i in range(len(Net_true_array)):
        print(Net_true_array[Net_true_array_i])
        my_file.write("\nДанные: %s" % (Net_true_array[Net_true_array_i]))

    my_file.close()


if __name__ == "__main__":
    main()
```