Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 1
по дисциплине «Интеллектуальные технологии информационной безопасности»

Тема: «Исследование однослойных неронных сетей на примере моделирования булевых выражений»

Вариант 3

Выполнил: Горбачев А.А.,
студент группы ИУ8-63

Проверил: Волосова Н.К.,
преподаватель каф. ИУ8

г. Москва,
2021 г.

1. Цель работы

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

2. Условия

Условия согласно варианту 3:

Булевая функция от 4ех переменных:

$$F(x_1, x_2, x_3, x_4) = x_1 + \overline{x_2} + \overline{(x_3 + x_4)}$$

Функция активации 1:

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

Функция активации 2:

$$f(net) = \frac{1}{2}(\tanh(net) + 1)$$

Норма обучения $\eta = 0.5$

3. Аналитическая часть

Алгоритм функционирования НС с пороговой ФА имеет вид

$$net = \sum_{i=1}^{4} w_i x_i + w_0$$

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

Где net – сетевой(комбинированный) вход, а y – реальный выход НС.

Алгоритм функционирования НС с логической ФА выглядит следующим образом:

$$net = \sum_{i=1}^{4} w_i x_i + w_0$$

$$out = f(net)$$

$$y(out) = \begin{cases} 1, & out \geq 0.5 \\ 0, & net < 0.5 \end{cases}$$

Где out – сетевой (недискретизированный) выход НС

Для необученной НС ее реальный выход y в общем случае отличается от целевого выхода t, представляющего собой значения заданной БФ нескольких переменных

$F(x_1, x_2, x_3, x_4): \{0,1\}^4 \rightarrow \{0,1\}$, т. е. имеется хотя бы один набор сигналов (x1 ,x2 ,x3 ,x4), для которого ошибка $\delta = t - y \neq 0$

Правило Видроу – Хоффа (дельта правило):

$$w_i^{l+1} = w_i^l + \Delta w_i^l$$

$$\Delta w_i^l = \eta \delta^l \frac{df(net)}{d\ net}\ x_i^l$$

На каждой эпохе k суммарная квадратичная ошибка E(k) равна расстоянию Хемминга между векторами целевого и реального выходов по всем входным векторам x1, x2, x3, x4

4. Ход работы

Получим нейросетевую модель булевой функции (таблица 1)

$$F(x_1, x_2, x_3, x_4) = x_1 + \overline{x_2} + \overline{(x_3 + x_4)}$$

Таблица 1. Таблица истинности БФ

F	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1
X4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
X3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
X2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
X1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

На начальном шаге $l = 0$ (эпоха $k = 0$) весовые коэффициенты беруться в виде:

$$w_0^0 = w_1^0 = w_2^0 = w_3^0 = w_4^0 = 0$$

Используя ФА 1. Динамика НС представлена в таблице 2, график суммарной ошибки приведен на рисунке 2.

Таблица 2. Параметры НС на послежовательных эпохах (Пороговая ФА)

EPOCH	FUNCTION	WEIGHTS	E
0	1111110001111011	0.3000, 0.6000, 0.0000, 0.0000, 0.0000	3
1	1111110011111111	0.0000, 0.6000, -0.300, 0.0000, -0.300	1
2	1011110011111011	0.3000, 0.9000, -0.300, 0.0000, 0.0000	3
3	1111110011111011	0.3000, 1.2000, -0.300, 0.0000, 0.0000	2
4	1111110011111111	0.0000, 1.2000, -0.600, 0.0000, -0.300	1
5	1011010011111111	0.3000, 1.2000, -0.600, 0.0000, -0.300	3
6	1111010011111111	0.3000, 1.2000, -0.600, 0.0000, -0.600	2
7	1011101011111111	0.3000, 1.2000, -0.900, -0.300, -0.300	2
8	1110010011111111	0.6000, 1.2000, -0.900, 0.0000, -0.300	3
9	1111010011111111	0.6000, 1.2000, -0.900, 0.0000, -0.600	2
10	1011101011111110	0.9000, 1.5000, -0.900, 0.0000, 0.0000	3
11	1111110011111111	0.6000, 1.5000, -1.200, 0.0000, -0.300	1
12	1111001011111111	0.6000, 1.5000, -1.200, -0.300, -0.300	2
13	1110010011111111	0.9000, 1.5000, -1.200, 0.0000, -0.300	3
14	1111010011111111	0.9000, 1.5000, -1.200, 0.0000, -0.600	2
15	1111001011111111	0.9000, 1.5000, -1.200, -0.300, -0.600	2
16	1110101011111111	0.9000, 1.5000, -1.500, -0.300, -0.300	2
17	1111000011111111	1.2000, 1.5000, -1.200, -0.300, -0.300	1
18	1111100011111111	1.2000, 1.5000, -1.200, -0.300, -0.300	0

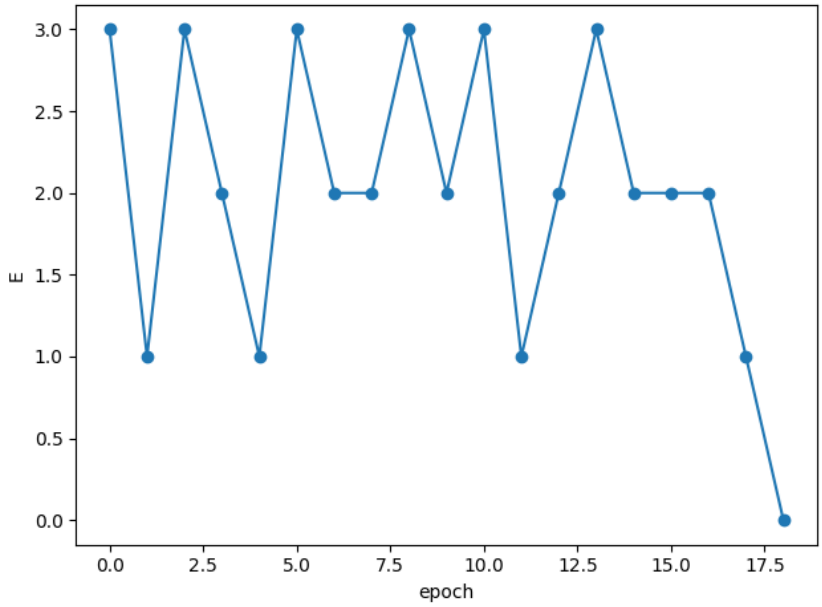

Рисунок 2 – График суммарной ошибки НС по эпохам обучения (пороговая ФА)

Используя логическую ФА и считая $\frac{d\,f(net)}{d\,net} = \frac{1}{\cosh(2net)+1}$ получим результаты приведенные в таблице 3 и на рисунке 3.

Таблица 3. Параметры НС на последовательных эпохах (Логическая ФА)

EPOCH	FUNCTION	WEIGHTS	E
0	1111110000111111	0.1467, 0.2967, -0.150, 0.0000, -0.000	3
1	1111011011111011	0.1571, 0.4446, -0.140, -0.150, 0.0103	4
2	1111110011111110	0.1548, 0.5922, -0.142, -0.002, 0.0081	2
3	1111110011111111	0.0049, 0.5922, -0.292, -0.002, -0.142	1
4	1011010011111111	0.1530, 0.5922, -0.291, -0.002, -0.141	3
5	1111010011111111	0.1502, 0.5922, -0.294, -0.002, -0.291	2
6	1011101011111110	0.2973, 0.7422, -0.294, -0.002, 0.0062	3
7	1111110011111111	0.1473, 0.7422, -0.444, -0.002, -0.144	1
8	1111000011111111	0.2848, 0.7422, -0.306, -0.002, -0.144	1
9	1111010011111111	0.2875, 0.7422, -0.304, -0.002, -0.291	2
10	1011101011111111	0.2900, 0.7422, -0.451, -0.150, -0.141	2
11	1110011011111111	0.3026, 0.7422, -0.588, -0.150, -0.128	4
12	1111000011111111	0.4410, 0.7422, -0.450, -0.150, -0.128	1
13	1111010011111111	0.4449, 0.7422, -0.446, -0.150, -0.275	2
14	1111010011111111	0.4450, 0.7422, -0.446, -0.150, -0.425	2
15	1110101011111111	0.4456, 0.7422, -0.593, -0.149, -0.277	2
16	1111000011111111	0.5924, 0.7422, -0.446, -0.149, -0.277	1
17	1111100011111111	0.5924, 0.7422, -0.446, -0.149, -0.277	0

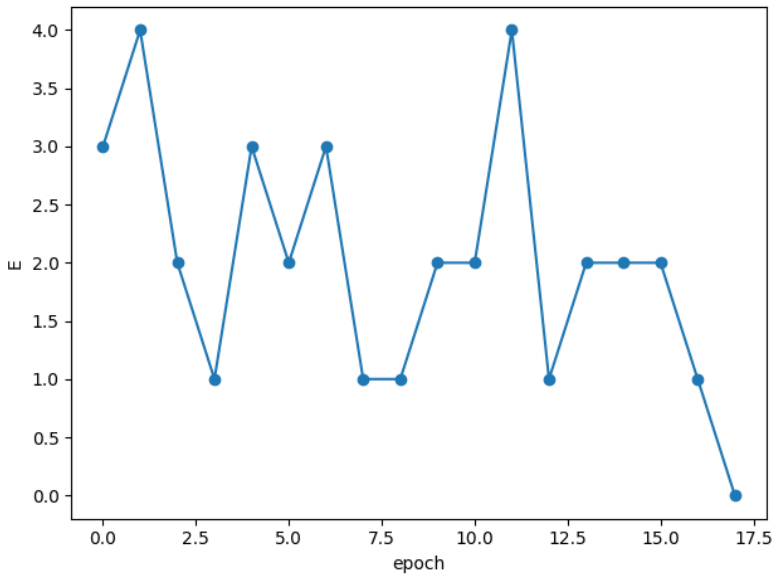

Рисунок 3 – График суммарной ошибки НС по эпохам обучения (логическая ФА)

Уменьшим размер выборки до 5 наборов

$x^1 = (0,0,1,1)$ $x^2 = (0,1,0,0)$ $x^3 = (0,1,0,1)$ $x^4 = (0,1,1,1)$

$x^5 = (0,1,1,0)$ $x^6 = (1,0,0,0)$ $x^7 = (1,0,1,0)$ $x^8 = (1,1,0,1)$

Результаты приведены в таблице 4 и на рисунке 4

Таблица 3. Параметры НС на последовательных эпохах (Логическая ФА) с уменьшенной выборкой

EPOCH	FUNCTION	WEIGHTS	E
0	11100010	0.1431, 0.2931, -0.004, 0.0000, -0.004	3
1	11100110	0.1458, 0.4430, -0.001, 0.0000, -0.001	2
2	11100111	-0.001, 0.4430, -0.148, 0.0000, -0.148	1
3	00110110	0.1559, 0.5908, -0.137, -0.003, -0.141	5
4	11001111	0.0059, 0.5908, -0.287, -0.153, -0.141	1
5	00100111	0.1444, 0.5908, -0.287, -0.015, -0.149	3
6	01100111	0.1443, 0.5908, -0.437, 0.1351, -0.149	2
7	10001111	0.1342, 0.5908, -0.447, -0.013, -0.149	2
8	00100111	0.2826, 0.5908, -0.449, 0.1370, -0.147	3
9	10010111	0.2806, 0.5908, -0.451, -0.011, -0.295	2
10	00101110	0.4353, 0.7408, -0.446, -0.009, -0.142	5
11	10100111	0.4385, 0.7408, -0.443, -0.009, -0.289	2
12	10100111	0.4385, 0.7408, -0.443, -0.009, -0.439	2
13	01001111	0.4502, 0.7408, -0.581, 0.0028, -0.289	2
14	10001111	0.4518, 0.7408, -0.580, -0.143, -0.289	2
15	10001111	0.4494, 0.7408, -0.582, -0.293, -0.289	2
16	01000111	0.5968, 0.7408, -0.582, -0.146, -0.142	1
17	11000111	0.5968, 0.7408, -0.582, -0.146, -0.142	0

1111100011111111

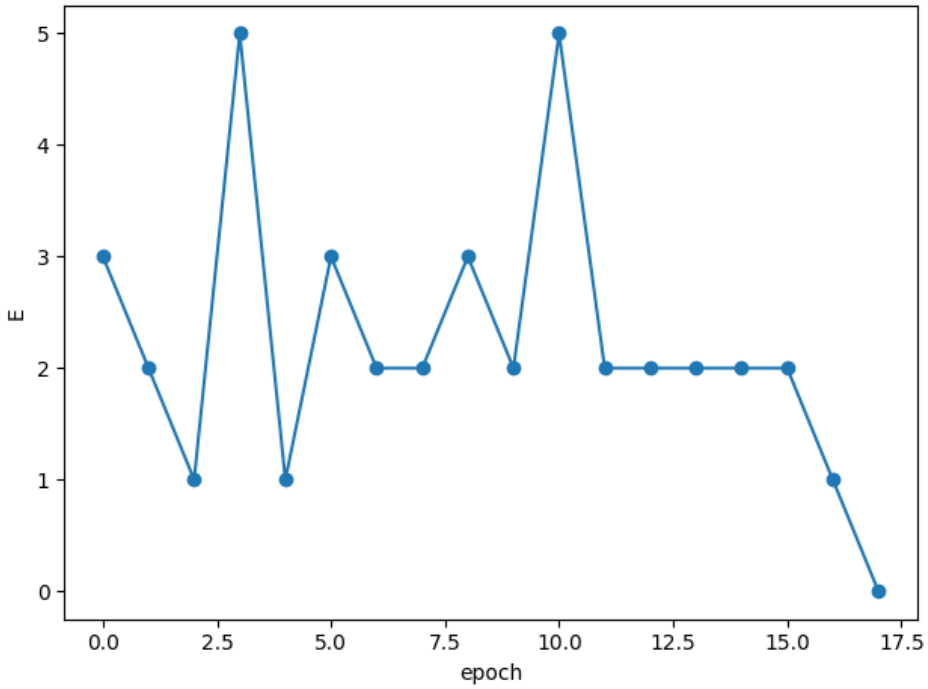

Рисунок 4 – График суммарной ошибки НС по эпохам обучения (логическая ФА) с уменьшенной выборкой

Код программы приведен в Приложении А.

5. Выводы

В ходе выполнения лабораторной работы, мною было исследованно функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации. Также она была обучена по правилу Видроу-Хоффа. Результаты совпали с ожидаемыми, что говорит о корректности работы программы.

Приложение А. Исходный код программы

Файл main.py

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


class NeuralNetwork:
    def __init__(self, type, n, func):
        self.__type = type
        self.__n = n
        self.__net = 0
        self.__weights = [0, 0, 0, 0, 0]
        self.__func = func
        self.__errors = []

    def function_one(self):
        return 1 if self.__net >= 0 else 0

    def function_two(self):
        return 0.5 * (np.tanh(self.__net) + 1)

    def function_two_dxdy(self):
        return 1 / (np.cosh(2 * self.__net) + 1)

    def net_calculator(self, x):
        self.__net =  self.__weights[0] + self.__weights[1] * x[1] +
self.__weights[2] * x[2] + self.__weights[3] * x[3] + self.__weights[4] * x[4]

    def weights_correction(self, sigma, x, dfdnet):
        for i in range(5):
            self.__weights[i] = self.__weights[i] + self.__n * sigma * dfdnet *
x[i]

    def study(self):
        print("EPOCH".rjust(5), "FUNCTION".rjust(18), "WEIGHTS".rjust(39),
"E".rjust(3))
        print("_" * 5, " ", "_" * 16, "_" * 39, " __")
        epoch = 0
        while True:
            error = 0
            predicted_y = False
            y = ""
            for i in range(16):
                x = [1, math.floor(i // 8) % 2, math.floor(i // 4) % 2,
math.floor(i // 2) % 2, math.floor(i // 1) % 2]
                if self.  type == 1:
                    self.net_calculator(x)
                    predicted_y = True if self.function_one() == 1 else False
                elif self.__type == 2:
                    self.net_calculator(x)
                    predicted_y = True if self.function_two() >= 0.5 else False
                if predicted_y is not self.__func[i]:
                    error += 1
                y += str(int(predicted_y))
                dfdnet = 1
                tn = int(self.__func[i])
                yn = int(predicted_y)
                sigm = tn - yn
```

```
                self.net_calculator(x)
                if self.__type == 2:
                    dfdnet = self.function_two_dxdy()
                self.weights_correction(sigm, x, dfdnet)
            w_string = ', '.join([str("%.3f" % it) if it < 0 else str("%.4f" %
it) for it in self.__weights])
            self.__errors.append([epoch, error])
            print(str(epoch).rjust(5), str(y).rjust(18),
str(w_string).rjust(39), str(error).rjust(3))
            if error == 0:
                break
            epoch += 1

    def study_selected(self):
        SELECTED = [
            [1, 0, 0, 1, 1],
            [1, 0, 1, 0, 0],
            [1, 0, 1, 0, 1],
            [1, 0, 1, 1, 1],
            [1, 0, 1, 1, 0],
            [1, 1, 0, 0, 0],
            [1, 1, 0, 1, 0],
            [1, 1, 1, 0, 1],
        ]
        SELECTED_FUNC = [ True,True, False, False, False, True, True, True]
        print("EPOCH".rjust(5), "FUNCTION".rjust(18), "WEIGHTS".rjust(39),
"E".rjust(3))
        print("_" * 5, " ", "_" * 16, "_" * 39, " __")
        epoch = 0
        while True:
            error = 0
            predicted_y = False
            y = ""
            for i in range(len(SELECTED)):
                if self.__type == 1:
                    self.net_calculator(SELECTED[i])
                    predicted_y = True if self.function_one() == 1 else False
                elif self.__type == 2:
                    self.net_calculator(SELECTED[i])
                    predicted_y = True if self.function_two() >= 0.5 else False
                if predicted_y is not SELECTED_FUNC[i]:
                    error += 1
                y += str(int(predicted_y))
                dfdnet = 1
                tn = int(SELECTED_FUNC[i])
                yn = int(predicted_y)
                sigm = tn - yn
                self.net_calculator(SELECTED[i])
                if self.__type == 2:
                    dfdnet = self.function_two_dxdy()
                self.weights_correction(sigm, SELECTED[i], dfdnet)
            w_string = ', '.join([str("%.3f" % it) if it < 0 else str("%.4f" %
it) for it in self.__weights])
            self.__errors.append([epoch, error])
            print(str(epoch).rjust(5), str(y).rjust(18),
str(w_string).rjust(39), str(error).rjust(3))
            if error == 0:
                break
            epoch += 1

        predicted = []
        for i in range(16):
```

```
            x = [1, math.floor(i // 8) % 2, math.floor(i // 4) % 2, math.floor(i
// 2) % 2, math.floor(i // 1) % 2]
            if self.__type == 1:
                self.net_calculator(x)
                predicted.append(True if self.function_one() == 1 else False)
            elif self.__type == 2:
                self.net_calculator(x)
                predicted.append(True if self.function_two() >= 0.5 else False)
        y_string = ''.join(['1' if it else '0' for it in predicted])
        print(y_string)

    def printgraph(self):
        err = np.array(self.__errors)
        x, y = err.T
        plt.ylabel('E')
        plt.xlabel('epoch')
        plt.scatter(x, y)
        plt.plot(x, y)
        plt.show()

    def reset(self, type, n, func):
        self.__type = type
        self.__n = n
        self.__net = 0
        self.__weights = [0, 0, 0, 0, 0]
        self.__func = func
        self.__errors = []

def main():
    FUNCTION = [True, True, True, True, True, False, False, False, True, True,
True, True, True, True, True, True]
    nw = NeuralNetwork(1, 0.3, FUNCTION)
    print('-' * 30, 'FA 1st type', '-' * 30)
    print()
    nw.study()
    nw.printgraph()
    print()
    print()
    print('-' * 30, 'FA 2nd type', '-' * 30)
    print()
    nw.reset(2, 0.3, FUNCTION)
    nw.study()
    nw.printgraph()
    print()
    print()
    print('-' * 30, 'FA 2nd type', '-' * 30)
    print()
    nw.reset(2, 0.3, FUNCTION)
    nw.study_selected()
    nw.printgraph()
    print()
    print()


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