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 нейронных сетей на примере моделирования булевых выражений»

Вариант 1

Выполнил: Антипов И.С.,
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. Условие

Задана булева функция от 4 переменных. Функция представлена вектором своих значений:

$$y = (0000000011010000)$$

Необходимо обучить нейронную сеть сначала с помощью пороговой функции активации:

$$y = \begin{cases} 1, net \geq 0, \\ 0, net < 0; \end{cases}$$

Далее необходимо обучить нейронную сеть с помощью логистической функции активации:

$$y = \frac{1}{2}\left(\frac{net}{1 + |net|} + 1\right)$$

3. Ход работы

Вначале необходимо построить таблицу истинности для исходной булевой функции:

Таблица 1. Таблица истинности заданной булевой функции

x_4	x_3	x_2	x_1	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0

0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

В самом начале весовые коэффициенты нулевые:

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

Коррекция весов происходит по правилу Видроу-Хоффа:

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

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

Обучение с помощью пороговой функции активации:

Используя пороговую функции активации, заданную в условии, обучим нейронную сеть. В таблице 2 представлен результат обучения. На рисунке 1 представлен график суммарной ошибки.

Таблица 2. Результаты обучения с помощью пороговой ФА

k	W	y	E
0	-0.600, -0.600, 0.0000, 0.0000, 0.0000	1000011010000100	6
1	-0.300, -0.600, 0.3000, 0.0000, 0.3000	0000011000000000	3
2	-0.600, -0.900, 0.3000, 0.0000, 0.0000	0100011000000100	5
3	-0.300, -0.900, 0.6000, 0.0000, 0.3000	0000011000000000	3
4	-0.600, -0.900, 0.6000, 0.0000, 0.3000	0100111000000000	3
5	-0.600, -0.900, 0.6000, 0.0000, 0.6000	0000111000000000	2
6	-0.900, -1.200, 0.6000, 0.0000, 0.3000	0100011000000100	5
7	-0.600, -1.200, 0.9000, 0.0000, 0.6000	0000011000000000	3
8	-0.900, -1.200, 0.9000, 0.0000, 0.6000	0100111000000000	3
9	-0.900, -1.200, 0.9000, 0.0000, 0.9000	0000111000000000	2
10	-1.200, -1.500, 0.9000, 0.0000, 0.6000	0100011000000100	5
11	-0.900, -1.500, 1.2000, 0.0000, 0.9000	0000011000000000	3
12	-1.200, -1.500, 1.2000, 0.0000, 0.9000	0100111000000000	3
13	-1.200, -1.500, 1.2000, 0.0000, 1.2000	0000111000000000	2
14	-1.500, -1.500, 1.2000, -0.300, 0.9000	0100011100000000	3
15	-1.200, -1.500, 1.5000, -0.300, 1.2000	0000011000000000	3
16	-1.500, -1.500, 1.5000, -0.300, 0.9000	0100110100000000	1
17	-1.500, -1.500, 1.5000, -0.300, 0.9000	0000110100000000	0

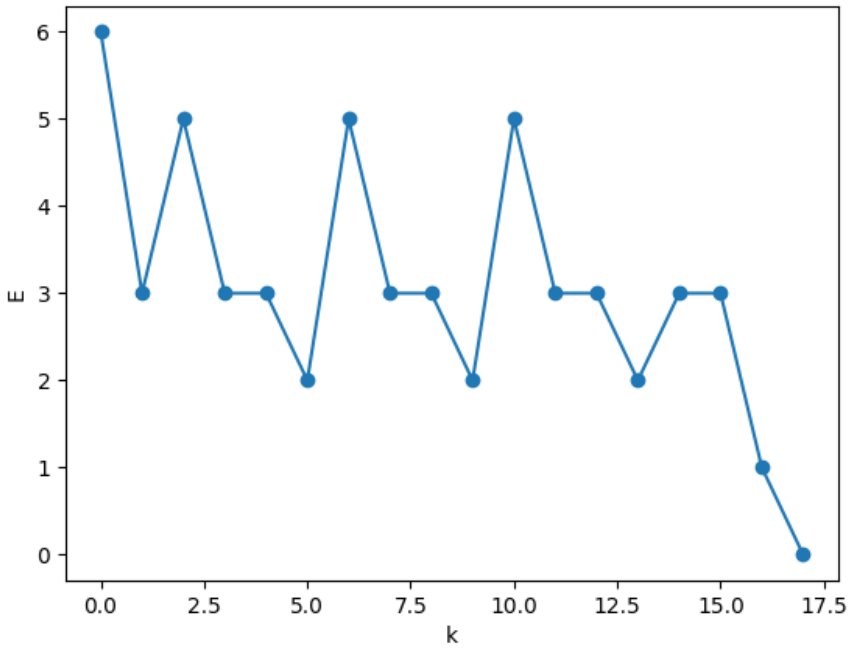

Рисунок 1 – График суммарной ошибки нейронной сети (пороговая ФА)

Обучение с помощью логистической функции активации:

Используя логистическую функции активации, заданную в условии, обучим нейронную сеть. В таблице 3 представлен результат обучения. На рисунке 2 представлен график суммарной ошибки.

Таблица 3. Результаты обучения с помощью логистической ФА

k	W	y	E
0	-0.062, -0.072, 0.0121, 0.0074, 0.0063	1000011010000100	6
1	-0.050, -0.108, 0.0244, 0.0145, 0.0121	0000011000000100	4
2	-0.041, -0.145, 0.0329, 0.0205, 0.0141	0000011000000100	4
3	-0.035, -0.145, 0.0393, -0.011, 0.0141	0000011100000000	2
4	-0.035, -0.145, 0.0393, -0.011, 0.0141	0000110100000000	0

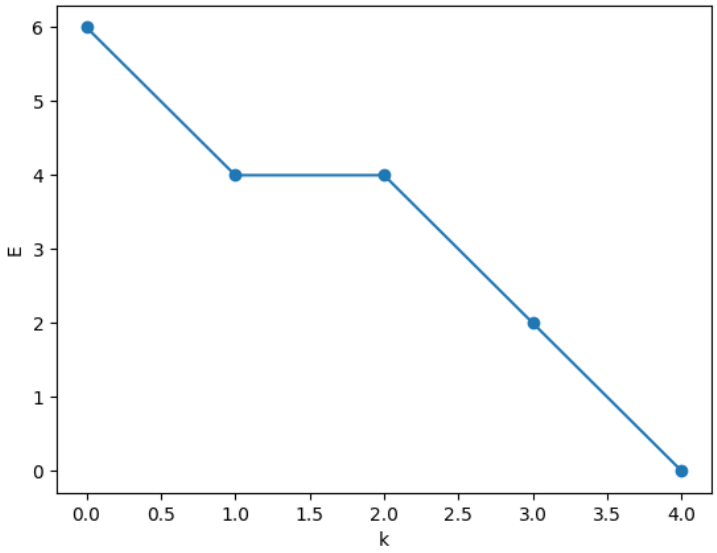

Рисунок 2 – График суммарной ошибки нейронной сети (логистическая ФА)

Уменьшение выборки:

Последовательно уменьшим размер обучающей выборки, пока нейронная сеть все еще способна к обучению. В случае заданного варианта это выборка (см. Таблицу 4).

Таблица 4. Наборы обучающей выборки

x_4	x_3	x_2	x_1	y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1

Используя логистическую функции активации, заданную в условии, и набор обучающей выборки обучим нейронную сеть. В таблице 5 представлен результат обучения. На рисунке 3 представлен график суммарной ошибки.

Таблица 5. Результаты обучения с помощью логистической ФА и обучающей выборки

k	W	y	E
0	0.0102, 0.0000, 0.0477, 0.0074, 0.0419	10000110	4
1	-0.018, 0.0000, 0.0928, 0.0139, 0.0448	11000110	5
2	-0.047, 0.0000, 0.0993, 0.0204, 0.0496	01001110	3
3	-0.077, 0.0000, 0.1060, 0.0271, 0.0539	01001110	3
4	-0.071, 0.0000, 0.1501, -0.004, 0.0561	00010110	4
5	-0.103, 0.0000, 0.1177, -0.037, 0.0561	00001111	1
6	-0.103, 0.0000, 0.1177, -0.037, 0.0561	00001101	0

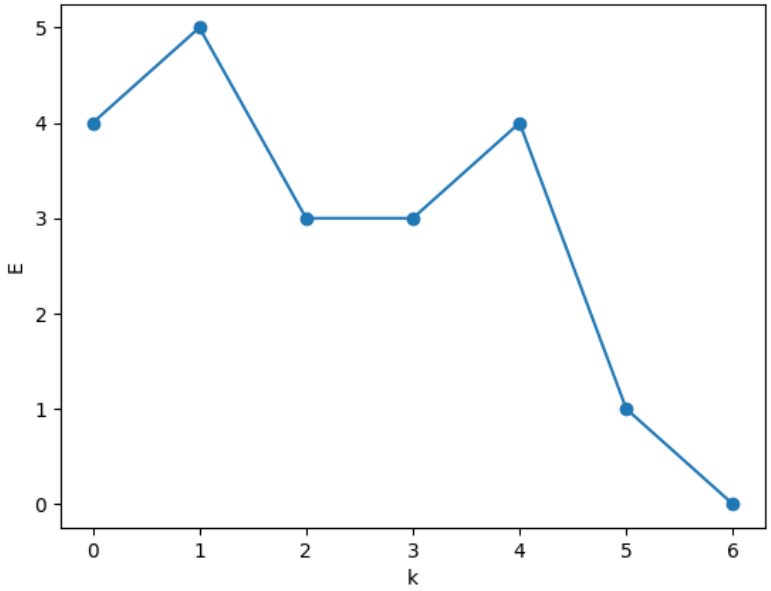

Рисунок 3 – График суммарной ошибки нейронной сети (логистическая ФА и обучающая выборка)

Код программы приведен в Приложении А.

4. Выводы

В ходе выполнения лабораторной работы было произведено обучение нейронной сети 3 разными способами: с помощью пороговой ФА, с помощью логистической ФА и с помощью логистической ФА и уменьшенной обучающей выборки.

Во всех 3 случаях удалось обучить нейронную сеть, но в случае логистической ФА для этого понадобилось минимальное количество эпох.

Приложение А. Исходный код программы

Файл main.py

```
import math
import matplotlib.pyplot as plot
import numpy


class NeuralNetwork:
    def __init__(self, isThres, eta, bf, x):
        self.__isThres = isThres
        self.__eta = eta
        self.__net = 0
        self.__bf = bf
        self.__weight = [0, 0, 0, 0, 0]
        self.__errors = []
        self.__x = x

    def countNet(self, x):
        self.__net = 0
        for i in range(5):
            self.__net += x[i] * self.__weight[i]

    def funcThres(self):
        return self.__net

    def funcLogistic(self):
        return 0.5 * (self.__net / (1 + abs(self.__net)) + 1)

    def weightCorrect(self, sigma, x):
        for i in range(5):
            self.__weight[i] = self.__weight[i] + self.countDeltaweight(sigma, x[i])

    def countDeltaweight(self, sigma, x_i):
        result = sigma * self.__eta * x_i
        if not self.__isThres:
            result *= self.countDerivative()
        return result

    def countDerivative(self):
        result = 0.5 * (1 - abs(self.funcLogistic())) ** 2
        return result

    def getErrors(self):
        return self.__errors

    def study(self):
        print("|", "k".ljust(3), "|", "W".ljust(39), "|", "y".ljust(18), "|", "E".ljust(3),
              "|")
        print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")
        done = False
        epoch = 0
        while not done:
            error = 0
            y = ""
            for i in range(len(self.__x)):
                x = [1, math.floor(i // 8) % 2, math.floor(i // 4) % 2, math.floor(i // 2) % 2,
                     math.floor(i // 1) % 2]
                if self.__isThres:
                    self.countNet(x)
                    result = self.funcThres() >= 0.0
                else:
                    self.countNet(x)
```

```
                    result = 0.5 <= self.funcLogistic()

                if result is not self.__bf[i]:
                    error += 1
                    self.weightCorrect(int(self.__bf[i]) - int(result), x)

                y += str(int(result))

            w_string = ', '.join([str("%.3f" % it) if it < 0 else str("%.4f" % it) for it in
                    self.__weight])
            self.__errors.append([epoch, error])
            print("|", str(epoch).ljust(3), "|",  str(w_string).ljust(39), "|",
                    str(y).ljust(18), "|",
                  str(error).ljust(3), "|")
            if error == 0:
                done = True
            epoch += 1

def printGraph(errors):
    err = numpy.array(errors)
    x, y = err.T
    plot.ylabel('E')
    plot.xlabel('k')
    plot.scatter(x, y)
    plot.plot(x, y)
    plot.show()

def generateX():
    x = []
    for i in range(16):
        x.append([1, math.floor(i // 8) % 2, math.floor(i // 4) % 2, math.floor(i // 2) % 2,
             math.floor(i // 1) % 2])
    return x

def start():
    bFunc = [False, False, False, False, True, True, False, True, False,
             False, False, False, False, False, False, False]

    bFuncSelected = [False, False, False, False, True, True, False, True]

    xSelected = [
        [1, 0, 0, 0, 0],
        [1, 0, 0, 0, 1],
        [1, 0, 0, 1, 0],
        [1, 0, 0, 1, 1],
        [1, 0, 1, 0, 0],
        [1, 0, 1, 0, 1],
        [1, 0, 1, 1, 0],
        [1, 0, 1, 1, 1],
    ]

    print("Task 1", '\n')
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")
    nw1 = NeuralNetwork(True, 0.3, bFunc, generateX())
    nw1.study()
    printGraph(nw1.getErrors())
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")

    print("Task 2", '\n')
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")
    nw2 = NeuralNetwork(False, 0.3, bFunc, generateX())
    nw2.study()
    printGraph(nw2.getErrors())
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")

    print("Task 3", '\n')
```

```
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")
    nw3 = NeuralNetwork(False, 0.3, bFuncSelected, xSelected)
    nw3.study()
    printGraph(nw3.getErrors())
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")

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