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 № 3
по дисциплине «Интеллектуальные технологии информационной безопасности»

Тема: «Исследование нейронных сетей с радиальными базисными функциями (RBF) 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 НС с радиальными базисными функциями (RBF) 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 с помощью RBF нейронов. Функционирование:

$$\varphi_j(X) = \exp\left(-\sum_{i=1}^{4}(x_i - c_{ji})^2\right),\ j = \overline{1,J}$$

Функция активации:

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

Коррекция весов происходит по правилу Видроу-Хоффа:

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

$$\Delta v_j^l = \eta\delta\varphi_j^l(X)$$

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

x_4	x_3	x_2	x_1	y
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

Находим J как $min(J_0, J_1)$ для данной функции $J = 3$

Соответственно C:

$$C^1 = (0,1,0,0)$$

$$C^2 = (0,1,0,1)$$

$$C^3 = (0,1,1,1)$$

В самом начале весовые коэффициенты нулевые:

$$v_0^0 = v_1^0 = v_2^0 = v_3^0 = 0$$

Обучение на полной выборке:

Результаты работы программы для полной выборки представлены в Таблице 2. График суммарной ошибки представлен на рисунке 1.

Таблица 2. Результаты обучения на полной выборке

k	W	y	E
0	0.1199, 0.1395, 0.2153, 0.0000	10000110	4
1	0.2397, 0.2791, 0.4306, 0.0000	10000110	4
2	0.3190, 0.3082, 0.3459, -0.300	10000111	3
3	0.3190, 0.3082, 0.3459, -0.300	00001101	0

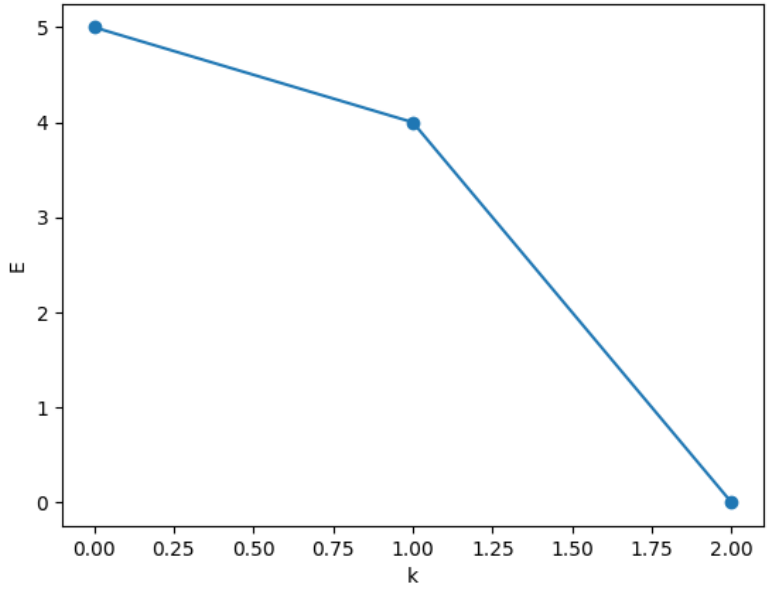

Рисунок 1 – График суммарной ошибки нейронной сети на полной выборке

Обучение на минимальной выборке:

Минимальная выборка для заданной функции была взята из лабораторной работы №1. Выборка представлена на таблице 3.

Таблица 3. 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

Результаты работы программы для минимальной выборки представлены в Таблице 4. График суммарной ошибки представлен на рисунке 2.

Таблица 4. Результаты обучения на минимальной выборке

k	W	y	E
0	0.1199, 0.1395, 0.2153, 0.0000	10000110	4
1	0.2397, 0.2791, 0.4306, 0.0000	10000110	4
2	0.3190, 0.3082, 0.3459, -0.300	10000111	3
3	0.3190, 0.3082, 0.3459, -0.300	00001101	0

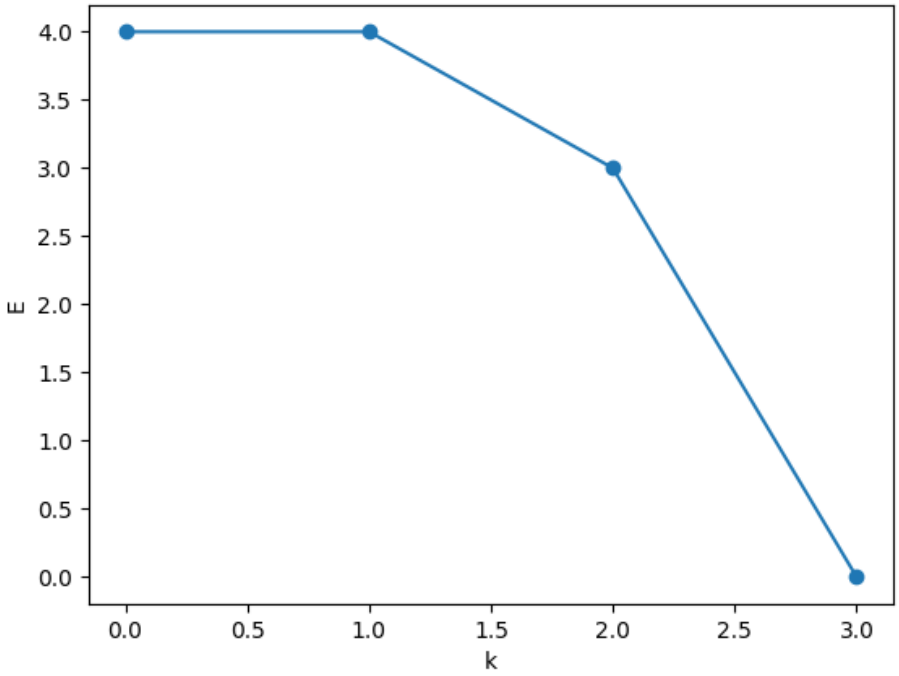

Рисунок 2 – График суммарной ошибки нейронной сети на минимальной выборке

Код программы приведен в Приложении А.

4. Выводы

В ходе выполнения лабораторной работы было произведено обучение нейронной сети с помощью радиальными базисных функций (RBF) и правила Видроу –Хоффа.

Этот метод оказался более эффективный, чем простое обучение с помощью пороговой функции. Количество эпох, которое понадобилось для обучения, оказалось намного меньше, чем в случае с простым обучением.

Приложение А. Исходный код программы

Файл main.py

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


class NeuralNetwork:
    def __init__(self, eta, bf, x, c):
        self.__eta = eta
        self.__net = 0
        self.__bf = bf
        self.__errors = []
        self.__x = x
        self.__c = c
        self.__n = len(x[0])
        self.__J = 3
        self.__weight = [0] * (self.__J + 1)

    def countPhi(self, j, x):
        result = 0
        for i in range(0, self.__n):
            result += (x[i] - self.__c[j][i]) ** 2
        return math.exp(-1 * result)

    def countNet(self, x):
        self.__net = 0
        for j in range(self.__J):
            self.__net += self.countPhi(j, x) * self.__weight[j]
        self.__net += self.__weight[-1]

    def weightCorrect(self, sigma, x):
        for j in range(self.__J + 1):
            self.__weight[j] = self.__weight[j] + self.countDeltaWeight(sigma,
j, x)

    def countDeltaWeight(self, sigma, j, x):
        if j == self.__J:
            return sigma * self.__eta
        return sigma * self.__eta * self.countPhi(j, x)


    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)):

                self.countNet(self.__x[i])
                result = self.__net >= 0.0
```

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

                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 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 = [
        [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],
    ]

    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],
```

```
    ]

    c = [
        [0, 1, 0, 0],
        [0, 1, 0, 1],
        [0, 1, 1, 1],
    ]

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

    print("Task Selected", '\n')
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")
    nw2 = NeuralNetwork(0.3, bFuncSelected, xSelected, c)
    nw2.study()
    printGraph(nw2.getErrors())
    print("+", "-" * 3, "+", "-" * 39, "+", "-" * 18, "+", "-" * 3, "+")


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