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 № 4
по дисциплине «Интеллектуальные технологии информационной безопасности»

Тема: «Изучение алгоритма обратного распространения ошибки (метод Back Propagation)»

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 многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки.

2. Условие

Задана архитектура многослойной нейронной сети:

$$1 - 2 - 1$$

Входной вектор:

$$x = (1, 2)$$

Выходной вектор:

$$t = (0.4)$$

Необходимо обучить нейронную сеть при помощи метода обратного распространения ошибки.

Функция активации:

$$f(net) = \frac{1 - \exp(-net)}{1 + \exp(-net)} \in (-1,1)$$

Её производная:

$$\frac{df(net)}{dnet} = \frac{1}{2}[1 - f^2(net)]$$

Начальные веса:

$$w_{ij}^1 = w_{jm}^2 = 0$$

Значение суммарной среднеквадратичной ошибки:

$$\varepsilon = 0.001$$

3. Ход работы

Первый этап обучения – по заданному входному сигналу рассчитать значения нейронов на скрытом слое, а далее на выходном слое.

1) $x_i^1 \equiv x_i, i = \overline{0, N}$

$$2)\ net_j^1(k) = \sum_{i=1}^{N} w_{ij}^1(k) x_i^1 + w_{0j}^1(k), \qquad j = \overline{1,J}$$

$$3)\ x_j^2(k) \equiv out_j^1(k) = f\left[net_j^1(k)\right], \qquad j = \overline{1,J}$$

$$4)\ net_m^2(k) = \sum_{j=1}^{J} w_{jm}^2(k) x_j^2(k) + w_{0m}^2(k), \qquad m = \overline{1,M}$$

$$5)\ y_m(k) \equiv out_m^2(k) = f[net_m^2(k)], \qquad m = \overline{1,M}$$

Второй этап обучения – оценка ошибки нейронов выходного слоя, а после этого нейронов скрытого слоя.

$$1)\ \delta_m(k) \equiv \delta_m^2(k) = \frac{df[net_m^2(k)]}{dnet_j^1(k)}[t_m - y_m(k)], \qquad m = \overline{1,M}$$

$$2)\ \delta_j^1(k) = \frac{df[net_j^1(k)]}{dnet_j^1(k)} \sum_{m=1}^{M} w_{jm}^2(k)\delta_m(k), \qquad j = \overline{1,J}$$

Третий этап – настройка весов выходного и скрытого слоев.

$$w_{ij}^1(k+1) = w_{ij}^1(k) + \Delta w_{ij}^1(k), \qquad \Delta w_{ij}^1(k) = \eta x_i \delta_j^1(k)$$

$$|w_{jm}^2(k+1) = w_{jm}^2(k) + \Delta w_{jm}^2(k), \qquad \Delta w_{jm}^2(k) = \eta x_j^2 \delta_m^2(k)$$

Обучение происходит до тех пор, пока суммарная среднеквадратичная ошибка не превысит некоторого заданного значения.

$$E(k) = \sqrt{\sum_{j=1}^{M} \left[t_j - y_j(k)\right]^2} \le \varepsilon$$

Результаты обучения приведены в таблице 1. График зависимости суммарной квадратичной ошибки от количества эпох приведен на рисунке 1.

Таблица 1. Результаты обучения

k	y	hidden weight	output weight	E
1	0.000	[0.0000, 0.0000] [0.0000, 0.0000]	[0.1000, 0.0000, 0.0000]	0.400
2	0.050	[0.0012, 0.0012] [0.0000, 0.0000]	[0.1873, 0.0000, 0.0000]	0.350
3	0.093	[0.0056, 0.0056] [0.0000, 0.0000]	[0.2633, 0.0001, 0.0000]	0.307
4	0.131	[0.0142, 0.0142] [0.0000, 0.0000]	[0.3294, 0.0005, 0.0000]	0.269
5	0.163	[0.0277, 0.0277] [0.0000, 0.0000]	[0.3870, 0.0013, 0.0000]	0.237
6	0.191	[0.0462, 0.0462] [0.0001, 0.0001]	[0.4373, 0.0027, 0.0000]	0.209
7	0.215	[0.0696, 0.0696] [0.0002, 0.0002]	[0.4814, 0.0047, 0.0000]	0.185
8	0.236	[0.0979, 0.0979] [0.0005, 0.0005]	[0.5200, 0.0074, 0.0000]	0.164
9	0.255	[0.1307, 0.1307] [0.0010, 0.0010]	[0.5540, 0.0107, 0.0000]	0.145
10	0.271	[0.1676, 0.1676] [0.0017, 0.0017]	[0.5839, 0.0146, 0.0001]	0.129
11	0.285	[0.2081, 0.2081] [0.0027, 0.0027]	[0.6103, 0.0190, 0.0001]	0.115
12	0.298	[0.2516, 0.2516] [0.0042, 0.0042]	[0.6336, 0.0238, 0.0002]	0.102
13	0.309	[0.2976, 0.2976] [0.0060, 0.0060]	[0.6541, 0.0288, 0.0003]	0.091
14	0.320	[0.3455, 0.3455] [0.0083, 0.0083]	[0.6722, 0.0340, 0.0004]	0.080
15	0.329	[0.3947, 0.3947] [0.0111, 0.0111]	[0.6880, 0.0393, 0.0005]	0.071
16	0.338	[0.4446, 0.4446] [0.0144, 0.0144]	[0.7018, 0.0445, 0.0006]	0.062
17	0.345	[0.4946, 0.4946] [0.0183, 0.0183]	[0.7138, 0.0495, 0.0008]	0.055
18	0.352	[0.5443, 0.5443] [0.0226, 0.0226]	[0.7242, 0.0543, 0.0010]	0.048
19	0.359	[0.5933, 0.5933] [0.0275, 0.0275]	[0.7332, 0.0587, 0.0012]	0.041
20	0.365	[0.6412, 0.6412] [0.0328, 0.0328]	[0.7409, 0.0628, 0.0014]	0.035
21	0.370	[0.6878, 0.6878] [0.0386, 0.0386]	[0.7474, 0.0665, 0.0016]	0.030
22	0.374	[0.7328, 0.7328] [0.0448, 0.0448]	[0.7529, 0.0698, 0.0018]	0.026
23	0.378	[0.7763, 0.7763] [0.0514, 0.0514]	[0.7575, 0.0726, 0.0021]	0.022
24	0.382	[0.8180, 0.8180] [0.0583, 0.0583]	[0.7613, 0.0751, 0.0023]	0.018
25	0.385	[0.8580, 0.8580] [0.0656, 0.0656]	[0.7645, 0.0773, 0.0024]	0.015
26	0.388	[0.8963, 0.8963] [0.0730, 0.0730]	[0.7671, 0.0791, 0.0026]	0.012
27	0.390	[0.9329, 0.9329] [0.0807, 0.0807]	[0.7692, 0.0806, 0.0028]	0.010
28	0.392	[0.9679, 0.9679] [0.0885, 0.0885]	[0.7709, 0.0818, 0.0029]	0.008
29	0.394	[1.0014, 1.0014] [0.0965, 0.0965]	[0.7722, 0.0828, 0.0030]	0.006
30	0.395	[1.0333, 1.0333] [0.1046, 0.1046]	[0.7732, 0.0836, 0.0031]	0.005
31	0.396	[1.0639, 1.0639] [0.1128, 0.1128]	[0.7740, 0.0842, 0.0032]	0.004
32	0.397	[1.0931, 1.0931] [0.1211, 0.1211]	[0.7746, 0.0847, 0.0033]	0.003
33	0.398	[1.1211, 1.1211] [0.1294, 0.1294]	[0.7750, 0.0850, 0.0033]	0.002
34	0.399	[1.1479, 1.1479] [0.1377, 0.1377]	[0.7753, 0.0852, 0.0033]	0.001
35	0.399	[1.1737, 1.1737] [0.1461, 0.1461]	[0.7755, 0.0854, 0.0034]	0.001

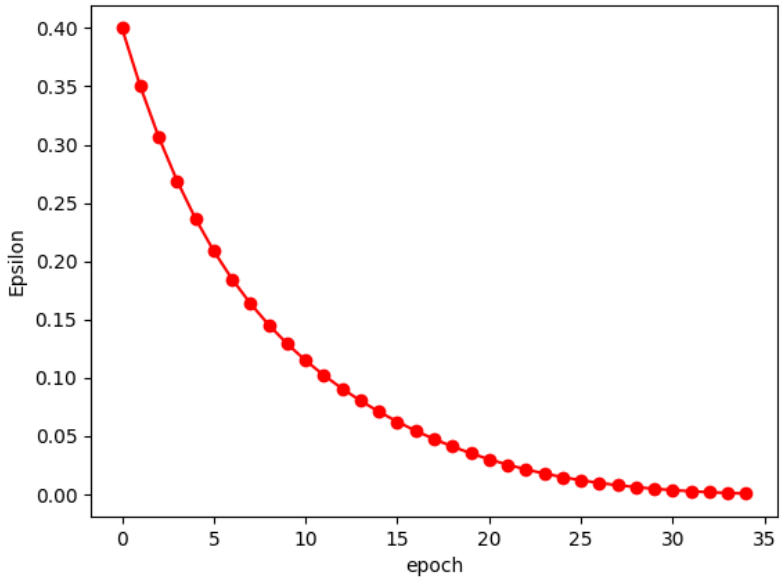

Рисунок 1 – График суммарной ошибки нейронной сети на полной выборке

Код программы приведен в Приложении А.

4. Выводы

В ходе выполнения лабораторной работы было произведено обучение многослойной нейронной сети (МНС) прямого распространения с помощью метода обратного распространения ошибок.

Нейронная сеть смогла обучиться и выдать значения, которые отличаются от оригинала меньше заданной величины.

Приложение А. Исходный код программы

Файл main.py

```
import math
import matplotlib.pyplot as plot


class NeuralNetwork:
    def __init__(self, eta, x, t, N, J, M, target_epsilon):
        self.__eta = eta
        self.__x = x
        self.__t = t
        self.__N = N
        self.__J = J
        self.__M = M
        self.__epoch = 0
        self.__target_epsilon = target_epsilon
        self.__epsilon = 1

        self.__first_w = [[0] * (N + 1) for _ in range(J)]
        self.__first_net = [0] * J
        self.__first_out = [0] * J
        self.__first_error = [0] * J

        self.__second_w = [[0] * (J + 1) for _ in range(M)]
        self.__second_net = [0] * M
        self.__second_out = [0] * M
        self.__second_error = [0] * M

        self.__all_epsilon = []

    def count_first_net(self, i):
        self.__first_net[i] = self.__first_w[i][0]
        for j in range(1, self.__N + 1):
            self.__first_net[i] += self.__first_w[i][j] * self.__x[j - 1]

    def count_second_net(self, i):
        self.__second_net[i] = self.__second_w[i][0]
        for j in range(1, self.__J + 1):
            self.__second_net[i] += self.__second_w[i][j] * self.__first_out[j -
1]

    def count_f(self, net):
        return (1 - math.exp((-1) * net)) / (1 + math.exp((-1) * net))

    def count_df(self, net):
        return 0.5 * (1 - (self.count_f(net) ** 2))

    def count_epsilon(self):
        self.__epsilon = 0
        for i in range(len(self.__t)):
            self.__epsilon += (self.__t[i] - self.__second_out[i]) ** 2
        self.__epsilon = math.sqrt(self.__epsilon)

    def count_sum(self, j):
        result = 0
        for i in range(self.__M):
            result += self.__second_w[i][j] * self.__second_out[i]
        return result
```

```
    def get_epsilon(self):
        return self.__all_epsilon

    def get_epoch(self):
        return self.__epoch

    def study(self):
        done = False

        while not done:
            for i in range(self.__J):
                self.count_first_net(i)
                self.__first_out[i] = self.count_f(self.__first_net[i])

            for i in range(self.__M):
                self.count_second_net(i)
                self.__second_out[i] = self.count_f(self.__second_net[i])

            for i in range(self.__M):
                df = self.count_df(self.__second_net[i])
                self.__second_error[i] = df * (self.__t[i] -
self.__second_out[i])

            for i in range(self.__J):
                df = self.count_df(self.__first_net[i])
                self.__first_error[i] = df * self.count_sum(i)

            for i in range(self.__J):
                self.__first_w[i][0] += self.__eta * self.__first_error[i]
                for j in range(self.__N):
                    self.__first_w[i][j + 1] += self.__eta * self.__x[j] *
self.__first_error[i]

            for i in range(self.__M):
                self.__second_w[i][0] += self.__eta * self.__second_error[i]
                for j in range(self.__J):
                    self.__second_w[i][j + 1] += self.__eta *
self.__first_out[j] * self.__second_error[i]

            self.__epoch += 1
            self.count_epsilon()
            self.__all_epsilon.append(self.__epsilon)

            f_w_string = ""
            for i in range(self.__J):
                f_w_string += "["
                f_w_string += ', '.join([str("%.3f" % it) if it < 0 else
str("%.4f" % it) for it in self.__first_w[i]])
                f_w_string += "] "

            s_w_string = ""
            for i in range(self.__M):
                s_w_string += "["
                s_w_string += ', '.join([str("%.3f" % it) if it < 0 else
str("%.4f" % it) for it in self.__second_w[i]])
                s_w_string += "] "

            print("|", str(self.__epoch).ljust(2),
                  "|", str(', '.join(str("%.3f" % it) for it in
self.__second_out)).ljust(5),
                  "|", str(f_w_string).ljust(34),
                  "|", str(s_w_string).ljust(25),
```

```
                "|", str("%.3f" % self.__epsilon).ljust(5), "|")
            done = self.__epsilon <= self.__target_epsilon

        print("+", "-" * 2, "+", "-" * 5, "+", "-" * 34, "+", "-" * 25, "+", "-"
* 5, "+")


def print_graph(errors, epoch):
    ep = [[i] for i in range(epoch)]
    plot.ylabel('Epsilon')
    plot.xlabel('epoch')
    plot.plot(ep, errors, "ro-")
    plot.show()


def start():
    print("Task Full", '\n')
    print("+", "-" * 2, "+", "-" * 5, "+", "-" * 34, "+", "-" * 25, "+",  "-" *
5, "+")
    print("|", "k".ljust(2),  "|", "y".ljust(5), "|", "hidden weight".ljust(34),
"|", "output weight".ljust(25),
    "|",  "E".ljust(5), "|")
    print("+", "-" * 2, "+", "-" * 5, "+", "-" * 34, "+", "-" * 25, "+",  "-" *
5, "+")

    nw = NeuralNetwork(0.5, [1, 2], [0.4], 1, 2, 1, 0.001)
    nw.study()
    print_graph(nw.get_epsilon(), nw.get_epoch())


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