МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

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

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Лабораторная работа №4 на тему:

«Интеллектуальные технологии информационной безопасности» на тему «Изучение алгоритма обратного распространения ошибки (метод Back Propagation)»

Вариант 4

Преподаватель:
Коннова Н.С.

Студент:
Куликова А.В.

Группа:
ИУ8-21М

Цель работы

Исследовать функционирование многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки (англ. Back Propagation – BP).

Постановка задачи

На примере МНС архитектуры $N-J-M$ реализовать ее обучение методом BP, проведя настройку весов нейронов скрытого $(w_{ij}^{(1)}(k), i=\overline{0,N}, j=\overline{1,J})$ и выходного $(w_{jm}^{(2)}(k), j=\overline{0,J}, m=\overline{1,M})$ слоев, где индексы $i,j=0$ соответствуют нейронам смещения; $k=1,2,\ldots$ – номер эпохи обучения.

№ варианта	Архитектура	x	$10t$
4	1 – 2 – 1	(1 3)	1

Ход работы

Архитектура МНС: 1 – 2 – 1 ($N=1,\ J=2,\ M=1$).

Пусть требуется обучить МНС на восстановление по входному вектору

$$x=[1,3]$$

целевого вектора

$$t=[0.1]$$

с погрешностью не более $\varepsilon=1\times10^{-3}$.

Исходные веса принимаются случайным образом:

Инициализация весов:

- ***Веса выходного слоя:***

[0.64979638],

[0.38528792],

[0.97424727]

- ***Веса скрытого слоя:***

[0.66864863, 0.97180816],

[0.24059453, 0.69485529]

В таблице 1 приведены результаты обучения МНС методом ВР.

Таблица 1 – Обучение МНС методом ВР

Номер эпохи к	**Веса скрытого слоя**	**Веса выходного слоя**	**Выходной вектор y**	**Суммарная ошибка E(к)**
0	0.6686 0.9718 0.2406 0.6949	0.6498 0.3853 0.9742	0.4220 0.6752	0.6592
1	0.5363 0.8153 0.1132 0.5442	0.6446 0.3801 0.9587	0.3118 0.5843	0.5286
2	0.4407 0.6558 0.0214 0.3911	0.6408 0.3763 0.9474	0.2266 0.4741	0.3950
3	0.3807 0.5108 -0.0362 0.2521	0.6385 0.3740 0.9403	0.1713 0.3593	0.2689
4	0.3461 0.3979 -0.0693 0.1439	0.6374 0.3728 0.9369	0.1390 0.2616	0.1662
5	0.3270 0.3226 -0.0876 0.0719	0.6370 0.3725 0.9357	0.1210 0.1932	0.0956
6	0.3166 0.2778 -0.0975 0.0289	0.6369 0.3724 0.9354	0.1112 0.1515	0.0527
7	0.3111 0.2526 0.1028 0.0048	0.6369 0.3723 0.9354	0.1060 0.1279	0.0285
8	0.3081 0.2389 -0.1056 -0.0083	0.6369 0.3724 0.9354	0.1032 0.1149	0.0153
9	0.3066 0.2315 -0.1071 -0.0154	0.6369 0.3724 0.9355	0.1017 0.1080	0.0081
10	0.3057 0.2275 -0.1079 -0.0191	0.6369 0.3724	0.1009 0.1042	0.0043

Номер эпохи к	Веса скрытого слоя	Веса выходного слоя	Выходной вектор y	Суммарная ошибка E(к)
		0.9355		
11	0.3053 0.2254 -0.1083 -0.0211	0.6369 0.3724 0.9355	0.1005 0.1023	0.0023
12	0.3050 0.2243 -0.1085 -0.0222	0.6369 0.3724 0.9355	0.1003 0.1012	0.0012
13	0.3049 0.2237 -0.1087 -0.0228	0.6369 0.3724 0.9355	0.1001 0.1006	0.0006

График зависимости ошибки от номера эпохи представлен на рисунке 1.

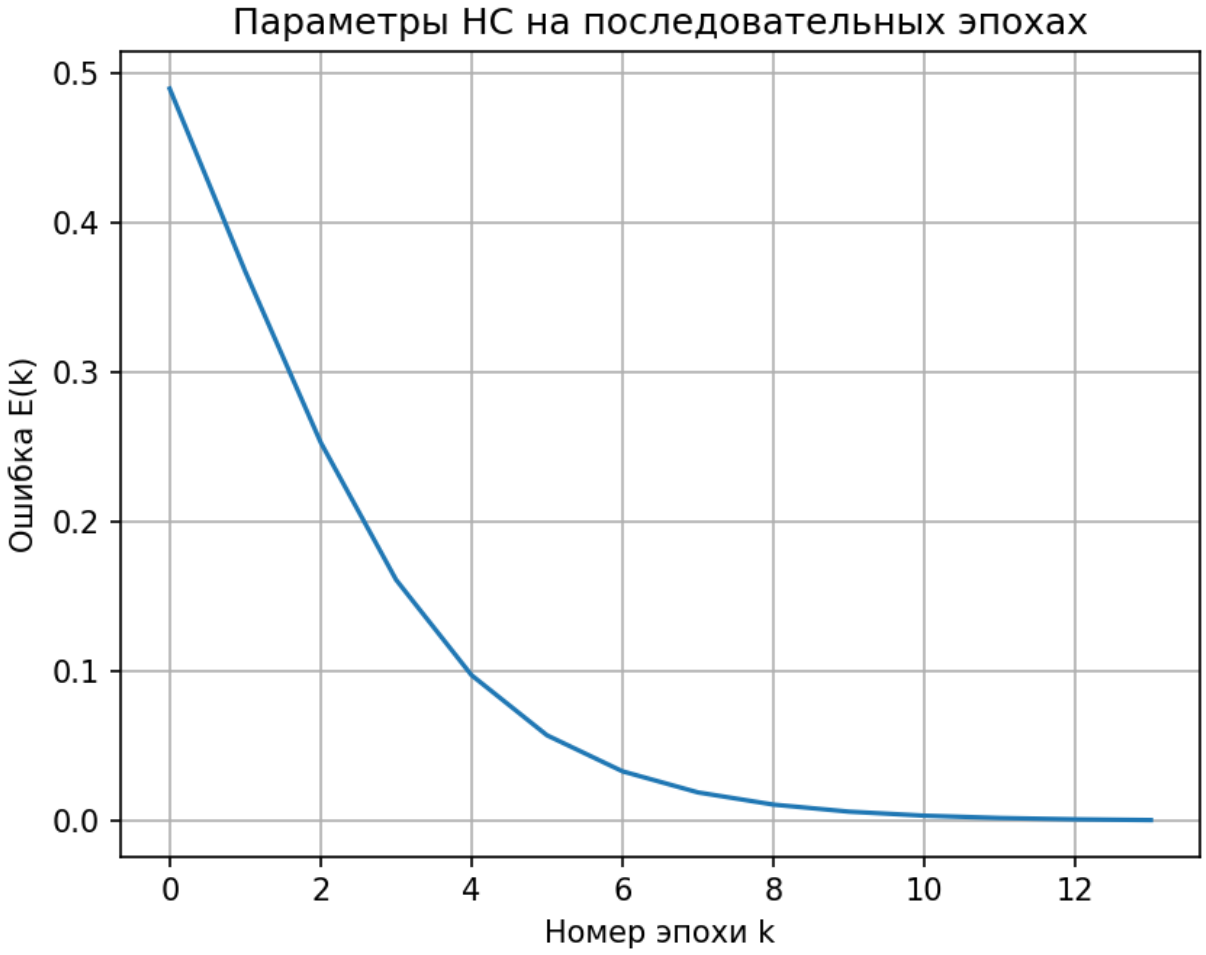

Рисунок 1 – График зависимости ошибки от номера эпохи

Выводы:

В ходе выполнения настоящей лабораторной работы было исследовано функционирование многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки (англ. Back Propagation).

Также был простроен график зависимости ошибки от номера эпохи.

Контрольные вопросы

1. Дайте определение МНС и объясните ее принципиальное отличие от однослойной с точки зрения нелинейной классификации.

Многослойные нейронные сети (МНС) – это тип нейронных сетей, которые состоят из нескольких слоёв нейронов, связанных между собой. В таких сетях входные данные проходят через несколько слоёв нейронов, каждый из которых выполняет свою функцию.

Однослойные нейронные сети состоят из одного слоя нейронов. Они могут выполнять только самые простые задачи, такие как линейное разделение данных.

Принципиальное отличие МНС от однослойной с точки зрения нелинейной классификации заключается в том, что многослойные сети способны выполнять более сложные задачи, требующие нелинейной обработки данных. Это достигается за счёт добавления дополнительных слоёв нейронов и связей между ними.

В многослойных сетях каждый слой нейронов может выполнять свою функцию, например, выделение признаков, классификация или регрессия. Это позволяет МНС быть более гибкими и эффективными в решении сложных задач, таких как распознавание образов, прогнозирование временных рядов и т. д.

2. В чем заключается основная идея метода обратного распространения ошибки?

Основная идея метода обратного распространения ошибки заключается в последовательном вычислении частных производных функции ошибки по всем весам и корректировке весов с целью минимизации этой ошибки.

Этот метод используется для обучения многослойных нейронных сетей. Он позволяет корректировать веса связей между нейронами таким образом, чтобы минимизировать ошибку между реальными и ожидаемыми выходными значениями.

Обучение происходит в два этапа:

Проход вперёд — входные данные проходят через все слои сети, и вычисляется выходное значение.

Проход назад — вычисляется ошибка между реальным и ожидаемым выходными значениями, и эта ошибка распространяется обратно через сеть, чтобы скорректировать веса связей.

Процесс повторяется до тех пор, пока ошибка не достигнет приемлемого уровня.

3. Сформулируйте теорему Колмогорова об аппроксимации.

Теорема Колмогорова об аппроксимации утверждает, что любая непрерывная функция на компактном множестве может быть равномерно аппроксимирована с любой заданной точностью полиномами. То есть для любой непрерывной функции $f(x)$ на компактном множестве $[a, b]$ и для любого эпсилон > 0 существует полином $P(x)$, такой что $|f(x) - P(x)| <$ эпсилон для всех x из $[a, b]$.

ПРИЛОЖЕНИЕ А

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

from tabulate import tabulate


# Функция активации
def activation(net):
    return (1 - np.exp(-net)) / (1 + np.exp(-net))

# Производная функции активации
def derivative_activation_function(net):
    return (1 - activation(net)**2)/2

class Layer:
    def __init__(self, neurons):
        self.neurons = neurons

class NN:
    def __init__(self, input_size, layers, epsilon=0.001, norma=1):
        self.input_size = input_size  # Размер входных данных
        self.layers = layers          # Количество нейронов в слое
        self.epsilon = epsilon        # Коэффициент для инициализации весов
        self.norma = norma            # Норма для инициализации весов

        # Инициализация весов
        self.weights = self.init_weights()
        # print(f"Инициализация весов: {self.weights}")

        # Инициализация Промежуточные значения
        self.nets = None  # сумматоры
        self.outs = None  # выходы

    def init_weights(self):
        weights = []
        prev_size = self.input_size  # size предыдущий слой = size входные
данные
        for layer in self.layers:
            weights.append(np.random.rand(prev_size + 1, layer.neurons))
            prev_size = layer.neurons  # Обновляем размер предыдущего слоя
для следующего нейрона
            print(f"Инициализация весов: {weights}")
        return weights

    # Обратное распространение ошибки для обновления весов (error: ошибка на
выходе сети)
    def backpropagation_func(self, error):
        derivative = derivative_activation_function(self.nets[-1])
        delta = [derivative * error]  # дельта для последнего слоя

        # Обратное распространение ошибки по всем слоям сети
        for i, nets in enumerate(reversed(self.nets[:-1]), 1):
            derivative = derivative_activation_function(nets)
            delta.append(derivative * np.dot(self.weights[-i][1:], delta[-
1]))  # дельта для текущего слоя

        # Обновление весов по дельтам
        for i, weights in enumerate(reversed(self.weights)):
            weights[0] += self.norma * delta[i]  # Обновление веса 0 индекса
```

```
            weights[1:] += self.norma * np.outer(self.outs[-i-2], delta[i])
# Обновление

    # Предсказание выхода сети для входных данных
    def network_output_prediction_input_data(self, x_input):
        nets, outs = [], [x_input]
        out = x_input  # первый слой = входное данное

        for layer, weights in zip(self.layers, self.weights):
            net = np.dot(out, weights[1:]) + weights[0]
            nets.append(net)
            out = activation(net)
            outs.append(out)

        self.nets, self.outs = nets, outs  # Сохраняем для дальше
        return out  #  предсказание

    # Обучение нейронной сети
    def lerning_func(self,
            x_train, # обучающие данные
            t_train # целевые значения
            ):

        k = -1
        error_mse = np.inf  # ошибка MSE
        error_mse_arr = []

        while error_mse > self.epsilon:
            k += 1
            y = self.network_output_prediction_input_data(x_train)  #
предсказанные значения сети
            error = t_train - y
            error_mse = np.sqrt(np.sum(error**2))  # MSE
            error_mse_arr.append(error_mse)

            data = []
            data.append(["Номер эпохи", "Ошибка E(k)", "Выходной вектор y"])
            data.append([k, error_mse.round(4), y.round(4)])

            for i, w in enumerate(self.weights, 1):
                if i != 1:
                    data.append([f"Веса скрытого слоя", "\n".join(map(str,
w.round(4))), ""])
                else:
                    data.append([f"Веса выходного слоя", "\n".join(map(str,
w.round(4))), ""])
            print(tabulate(data, tablefmt="plain", headers="firstrow"))
            print("\n")
            if error_mse > self.epsilon:
                self.backpropagation_func(error)  # обновление. обратное
распространение ошибки

        print(f"t: {t_train}")

        plt.plot(range(len(error_mse_arr)), error_mse_arr)
        plt.title('Параметры НС на последовательных эпохах'),
plt.xlabel('Номер эпохи k'), plt.ylabel('Ошибка E(k)')
        plt.grid(), plt.show()

x = [3]
t = [0.1]

'''
Архитектура 1-2-1 : нейронная сеть состоит из трех слоев
```

```
1. Входной слой с одним нейроном: нейрон принимает входные данные
2. Скрытый слой с двумя нейронами: вычисление внутренних представлений данных
3. Выходной слой с одним нейроном: Этот нейрон принимает выходные данные от
скрытого слоя и генерирует окончательный результат
'''

# (1-2-1) два слоя с количеством нейронов 2 и 1

nn = NN(input_size=len(x), layers=[Layer(neurons=4), Layer(neurons=1)])
nn.lerning_func(x_train=x, t_train=t)
```