Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Приложение нейросетевых алгоритмов**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Цель работы**

Получение знаний и практических навыков построения однослойных нейронных сетей.

**Задание**

Разработать алгоритм и программу решения задачи на языке Python без использования специализированных библиотек.

**Описание решения**

*Задача: нейросеть Кохонена для решения задачи кластеризации данных об успеваемости студентов.*

Для решения данной задачи были использованы библиотеки random и math. Сам код задачи включает в себя несколько классов и функций: normalizing\_fun не относится ни к какому классу, она нормализует входные данные, поскольку они имеют разброс значений (0, 1, 65, 98 как пример). Класс Neuron включает функции \_\_init\_\_ (инициализация объекта со случайными весами), summat (сумматор нейрона вместе с порогом), activation\_fun (функция активации гиперболический тангенс) и gradient\_descent (изменение весов градиентным спуском). Класс Neural\_Network включает в себя функции \_\_init\_\_ (инициализация объекта с пустым список выходов нейронов), mse\_less (среднеквадратичная ошибка), calculate\_gradient (расчёт градиента отдельно для каждого веса для корректировки методом градиентного спуска), learning\_algorithm (алгоритм обучения сети, который заключается в получении выхода каждого нейрона для каждого примера, вычисляется ошибка, вычисляется градиент и происходит корректировка весов) и testing (тестирование работы сети и проверка её точности).

Алгоритм работы программы следующий:

- на вход сети подаётся список списков, который является обучающей выборкой, который нормализуется для корректной обработки;

- в качестве желаемых результатов подаётся список правильных ответов к обучающей выборке;

- устанавливается скорость обучения, создаются нейроны и сама нейронная сеть;

- запускается алгоритм обучения сети, в котором происходит корректировка весов;

- запускается тестирование сети после обучения на стартовых данных.

**Реализация программы на языке Python**

import random as rnd

import math as mt

def normalizing\_fun(*list\_of\_list\_of\_int*):

    """

    Функция, которая нормализует входные данные

    Args: list\_of\_list\_of\_int - список списков int-значений (о как)

    Return: нормализованный массив

    """

    result = [[0 for \_ in range(len(*list\_of\_list\_of\_int*[0]))] for \_ in range(len(*list\_of\_list\_of\_int*))]

    for i in range(len(*list\_of\_list\_of\_int*)):

        for j in range(len(*list\_of\_list\_of\_int*[0])):

            mean = sum(*list\_of\_list\_of\_int*[i]) / len(*list\_of\_list\_of\_int*[0])

*#пробовал не / mean, a - mean (тогда у всех нейронов одинаковый результат)*

            result[i][j] = *list\_of\_list\_of\_int*[i][j] / (mean \* 5)

    return result

class Neuron:

*#веса нейрона*

    weights = []

    def \_\_init\_\_(*self*, *num\_of\_inputs*):

        """

        Функция, которая инициализирует веса нейрону

        Args: num\_of\_inputs - кол-во входных сигналов

        Return: отсутствует

        """

*self*.weights = [rnd.uniform(-1.0, 1.0) for \_ in range(*num\_of\_inputs*)]

    def summat(*self*, *list\_of\_inputs*):

        """

        Функция-сумматор нейрона

        Args: list\_of\_inputs - список входных данных

        Return: выход нейрона

        """

*#прибавляем 1 - это порог*

        return 1 + sum(*self*.weights[i] \* *list\_of\_inputs*[i] for i in range(len(*list\_of\_inputs*)))

    def activation\_func(*self*, *input\_x*):

        """

        Функция активации нейрона (гиперболический тангенс)

        Args: input\_x - входной параметр

        Return: значение функции

        """

*#тут появилась ошибка math range error*

*#return (mt.exp(2 \* input\_x) - 1) / (mt.exp(2 \* input\_x) + 1)*

        return mt.tanh(*input\_x*)

    def gradient\_descent(*self*, *gradient*, *learning\_rate*):

        """

        Функция, которая обучает нейроны (корректирует веса) методом градиентного спуска

        Args: gradient - градиент обратного распространения ошибки; learning\_rate - скорость обучения

        Return: отсутствует

        """

        for i in range(len(*self*.weights)):

*self*.weights[i] -= *gradient*[i] \* *learning\_rate*

class Neural\_Network:

    def \_\_init\_\_(*self*):

        """

        Функция, которая создаёт нейросети пустой массив для хранения выходов нейронов по примерам

        Args: отсутствуют

        Return: отсутствует

        """

*self*.neurons\_outputs = []

    def mse\_less(*self*, *list\_of\_answers*, *neurons\_outputs*):

        """

        Функция, которая считает среднеквадратичную ошибку

        Args: list\_of\_answers - список желаемых ответов; neurons\_outputs - выходы нейронов

        Return: среднеквадратичная ошибка

        """

        error = sum(mt.pow(*list\_of\_answers*[i] - *neurons\_outputs*[i], 2) for i in range(len(*list\_of\_answers*)))

        return error / len(*list\_of\_answers*)

    def calculate\_gradient(*self*, *list\_of\_inputs*, *list\_of\_answers*, *average\_error*):

        """

        Функция, которая рассчитывает градиент методом обратного распространения ошибки

        Args: list\_of\_inputs - список входных параметров; list\_of\_answers - список желаемых выходов; average\_error - средняя ошибка

        Return: градиент

        """

        gradient = []

        for i in range(len(*list\_of\_inputs*[0])):

*#gradient.append((1 / len(list\_of\_answers)) \* list\_of\_inputs[i] \* average\_error)*

*#теперь рассчитываем градиент для каждого веса, а не одинаковый для всех, как раньше*

            gradient.append((1 / len(*list\_of\_answers*)) \* sum(*list\_of\_inputs*[j][i] \* (*list\_of\_answers*[j] - *self*.neurons\_outputs[j]) for j in range(len(*list\_of\_answers*))))

        return gradient

    def learning\_algorithm(*self*, *learning\_rate*, *list\_of\_inputs*, *list\_of\_outputs*, *neurons*):

        """

        Функция, которая реализует алгоритм обучения нейросети

        Args: learning\_rate - скорость обучения; list\_of\_inputs - список входных параметров; list\_of\_outputs - список желаемых выходов; neurons - список нейронов

        Return: отсутствует

        """

        while *learning\_rate* > 0.0:

            for i in range(len(*list\_of\_inputs*)):

*#список ошибок всех нейронов для одного примера*

                errors = []

*#список выходов нейронов для одного примера*

                neurons\_outs = []

                for j in range(len(*neurons*)):

*#выход нейрона*

                    neuron\_out = *neurons*[j].activation\_func(*neurons*[j].summat(*list\_of\_inputs*[i]))

                    neurons\_outs.append(neuron\_out)

*#ошибка нейрона*

                    error = *self*.mse\_less([*list\_of\_outputs*[i][0]], [neuron\_out])

                    errors.append(error)

*#заполняем массив для хранения по примерам*

*self*.neurons\_outputs = neurons\_outs

*#средняя ошибка*

                average\_error = sum(errors) / len(errors)

*#градиент*

                gradient = *self*.calculate\_gradient(*list\_of\_inputs*, *list\_of\_outputs*[i], average\_error)

                for j in range(len(*neurons*)):

*neurons*[j].gradient\_descent(gradient, *learning\_rate*)

                print(*neurons*[0].weights)

                print(errors)

*learning\_rate* -= 0.00005

    def testing(*self*, *neurons*, *list\_of\_inputs*, *list\_of\_outputs*):

        """

        Функция, которая показывает результат работы нейросети

        Args: neurons - список нейронов; list\_of\_inputs - список входных данных; list\_of\_outputs - список желаемых выходов

        """

        for i in range(len(*list\_of\_inputs*)):

*#список выходов нейронов*

            neurons\_outs = []

            for j in range(len(*neurons*)):

                neuron\_out = *neurons*[j].activation\_func(*neurons*[j].summat(*list\_of\_inputs*[i]))

                neurons\_outs.append(neuron\_out)

*#среднее значение выхода*

            mean = sum(neurons\_outs) / len(neurons\_outs)

            if 1.0 < mean < 1.1: mean = 1.75

            elif 0.9 < mean < 1.0: mean = 1.50

            elif 0.8 < mean < 0.9: mean = 1.25

            elif 0.7 < mean < 0.8: mean = 1

            else: mean = 0

            print(f"Выход нейронов: {neurons\_outs}; средний выход: {mean}; желаемый выход: {*list\_of\_outputs*[i][0]}")

*#кол-во нейронов*

neurons\_quantity = 4

*#входные параметры*

inputs\_x = [

    [1, 1, 60, 79, 60, 72, 63],

    [1, 0, 60, 61, 30, 5, 17],

    [0, 0, 60, 61, 30, 66, 58],

    [1, 1, 85, 78, 72, 70, 85],

    [0, 1, 65, 78, 60, 67, 65],

    [0, 1, 60, 78, 77, 81, 60],

    [0, 1, 55, 79, 56, 69, 72],

    [1, 0, 55, 56, 50, 56, 60],

    [1, 0, 55, 60, 21, 64, 50],

    [1, 0, 60, 56, 30, 16, 17],

    [0, 1, 85, 89, 85, 92, 85],

    [0, 1, 60, 88, 76, 66, 60],

    [1, 0, 55, 64, 0, 9, 50],

    [0, 1, 80, 83, 62, 72, 72],

    [1, 0, 55, 10, 3, 8, 50],

    [0, 1, 60, 67, 57, 64, 50],

    [1, 1, 75, 98, 86, 82, 85],

    [0, 1, 85, 85, 81, 85, 72],

    [1, 1, 80, 56, 50, 69, 50],

    [1, 0, 55, 60, 30, 8, 60]

]

print(f"Нормализованные входные параметры: {normalizing\_fun(inputs\_x)}")

outputs\_y = [

    [1], [0], [0], [1.25], [1],

    [1.25], [0], [0], [0], [0],

    [1.75], [1.25], [0], [1.25], [0],

    [0], [1.50], [1.25], [0], [0]

]

*#скорость обучения (при настройке 0.30 веса "взрываются")*

learning\_rate = 0.001

*#список нейронов*

neurons = [Neuron(len(inputs\_x[0])) for \_ in range(neurons\_quantity)]

*#нейросеть*

neural\_network = Neural\_Network()

*#запускаем алгоритм обучения сети*

neural\_network.learning\_algorithm(learning\_rate, normalizing\_fun(inputs\_x), outputs\_y, neurons)

*#проверяем нейросеть после обучения*

print()

print("Нейросеть после обучения")

neural\_network.testing(neurons, normalizing\_fun(inputs\_x), outputs\_y)

**Примеры запусков и выводы программы**



