Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**НЕЙРОННЫЕ СЕТИ**

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ**

студентки 5 курса 531 группы

факультета компьютерных наук и информационных технологий

*Змеевой Вероники Александровны*

фамилия, имя, отчество

Преподаватель

аспирант \_\_\_\_\_\_\_\_\_\_\_\_\_ А. С. Никитина

подпись, дата

Саратов 2024

**СОДЕРЖАНИЕ**

[1. Создание ориентированного графа 3](#_Toc183794991)

[1.1 Описание задания 1 3](#_Toc183794992)

[1.2 Тестирование задания 1 4](#_Toc183794993)

[2. Создание функции по графу 7](#_Toc183794994)

[2.1 Описание задания 2 7](#_Toc183794995)

[2.2 Тестирование задания 2 9](#_Toc183794996)

[3. Вычисление значения функции на графе 14](#_Toc183794997)

[3.1 Описание задания 3 14](#_Toc183794998)

[3.2 Тестирование задания 3 15](#_Toc183794999)

[4. Построение многослойной нейронной сети 19](#_Toc183795000)

[4.1 Описание задания 4 19](#_Toc183795001)

[4.2 Тестирование задания 4 19](#_Toc183795002)

[5. Построение многослойной нейронной сети 23](#_Toc183795003)

[5.1 Описание задания 5 23](#_Toc183795004)

[5.2 Тестирование задания 5 24](#_Toc183795005)

[ПРИЛОЖЕНИЯ 28](#_Toc183795006)

[Приложение 1 – задание 1 28](#_Toc183795007)

[Приложение 2 – задание 2 32](#_Toc183795008)

[Приложение 3 – задание 3 34](#_Toc183795009)

[Приложение 4 – задание 4 40](#_Toc183795010)

[Приложение 5 – задание 5 42](#_Toc183795011)

# **1. Создание ориентированного графа**

## **1.1 Описание задания 1**

**На входе:** текстовый файл с описанием графа в виде списка дуг:

где - начальная вершина дуги , - конечная вершина дуги , - порядковый номер дуги в списке всех заходящих в вершину дуг.

**На выходе:**

а) Ориентированный граф с именованными вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла).

б) Сообщение об ошибке в формате файла, если ошибка присутствует.

**Способ проверки результата:**

1. Сериализованная структура графа в формате XML или JSON.

**Пример:**

<graph>

    <vertex>v1</vertex>

    <vertex>v2</vertex>

    <vertex>v3</vertex>

    <arc>

        <from>v1</from>

        <to>v3</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v2</from>

        <to>v3</to>

        <order>2</order>

    </arc>

</graph>

б) Сообщение об ошибке с указанием номера строки с ошибкой во входном файле.

## **1.2 Тестирование задания 1**

**Пример входного файла:**

(v1, v3, 1), (v2, v3, 2)

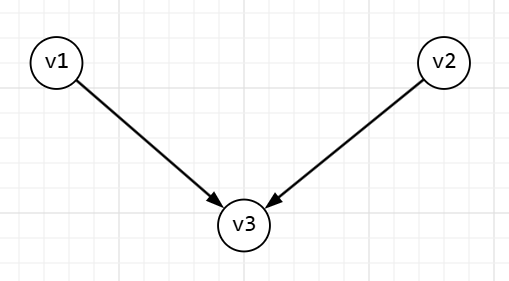


Рисунок 1 – граф по описанию

**Пример работы программы:**

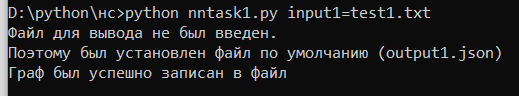


Рисунок 2 – запуск программы в консоли

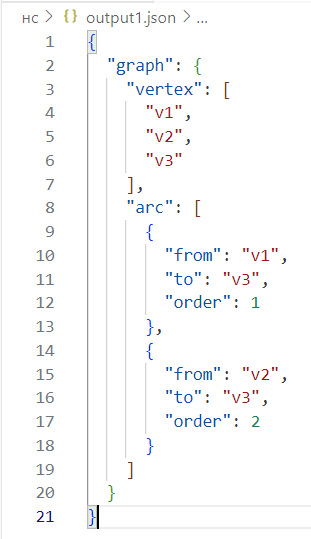


Рисунок 3 – результат работы программы

**Пример входного файла:**

(v1, v2, 1),(v3, v4, 1),(v2, v5, 1),(v5, v6, 1),(v5, v7, 1),(v4, v8, 1),(v4, v9, 1), (v9, v6, 2)

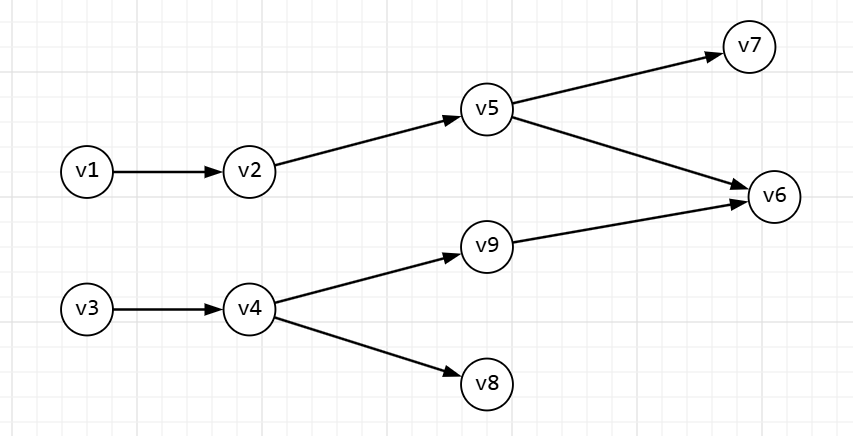


Рисунок 4 – граф по описанию

**Пример работы программы:**

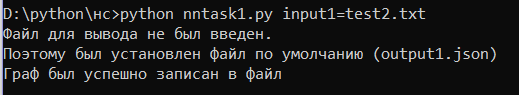


Рисунок 5 – запуск программы в консоли

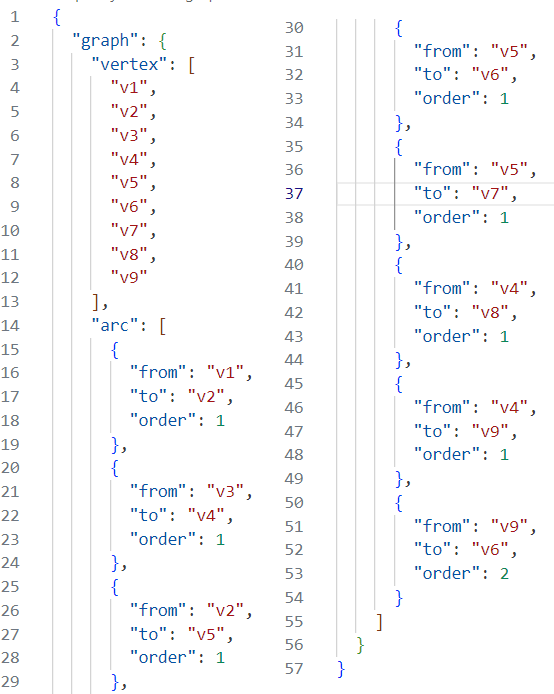


Рисунок 6 – результат работы программы

**Пример входного файла:**

(v1, v2, 1),(v1, v2, 1)

**Пример работы программы:**

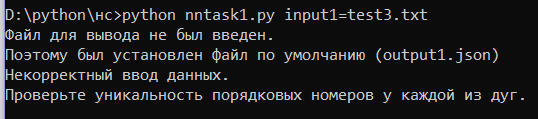


Рисунок 7 – запуск программы в консоли

**Пример входного файла:**

(v1, v2, 1),(v1, v2, a)

**Пример работы программы:**

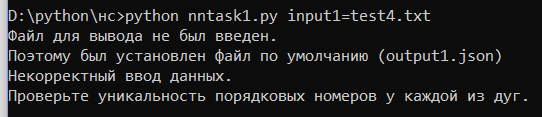


Рисунок 8 – запуск программы в консоли

**Пример входного файла:**

(a, b, 1),(c, d, 1),(b, e, 1),(d, e, 2),(e, f, 1)

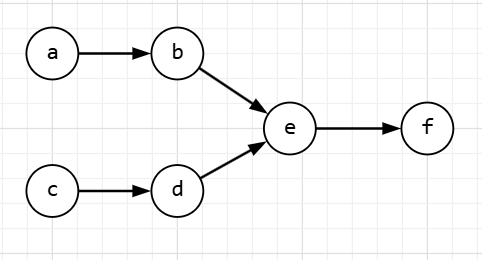


Рисунок 9 – граф по описанию

**Пример работы программы:**

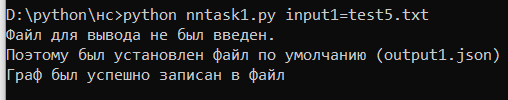


Рисунок 10 – запуск программы в консоли



Рисунок 11 – результат работы программы

**Пример входного файла:**

(a, b, 1),(c, d, 1),(b, e, 1),(d, e, 1),(e, f, 1)

**Пример работы программы:**

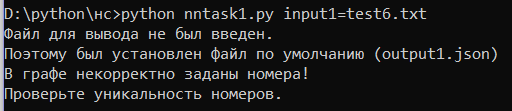


Рисунок 12 – запуск программы в консоли

# **2. Создание функции по графу**

## **2.1 Описание задания 2**

**На входе:** ориентированный граф с именованными вершинами как описано в задании 1.

**На выходе:** линейное представление функции, реализуемой графом в префиксной скобочной записи:

Способ проверки результата:

1. выгрузка в текстовый файл результата преобразования графа в имя функции.
2. сообщение о наличии циклов в графе, если они присутствуют.

## **2.2 Тестирование задания 2**

**Пример входного файла:**



Рисунок 13 – входной файл

**Пример работы программы:**

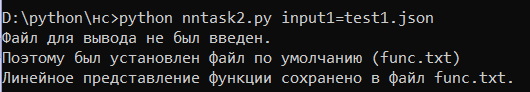


Рисунок 14 – запуск программы в консоли

Линейное представление функции: v3(v1(), v2())

**Пример входного файла:**

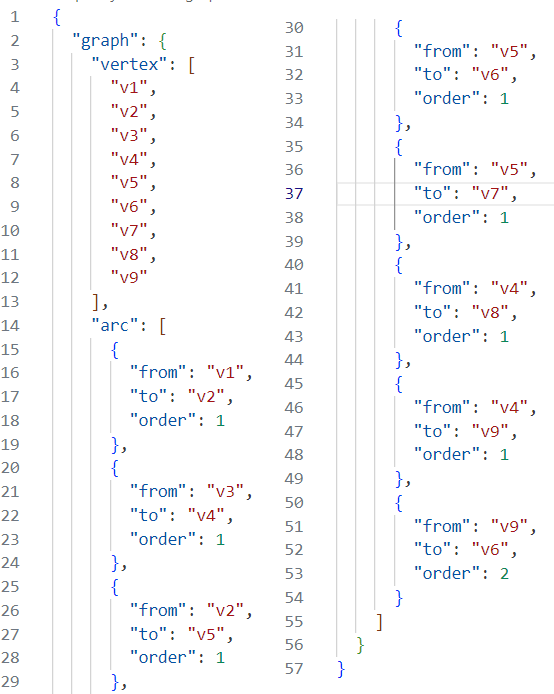


Рисунок 15 – входной файл

**Пример работы программы:**

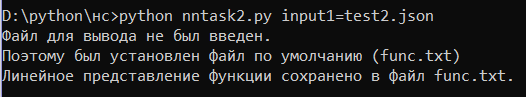


Рисунок 16 – запуск программы в консоли

Линейное представление функции: v6(v5(v2(v1())), v9(v4(v3()))), v7(v5(v2(v1()))), v8(v4(v3()))

**Пример входного файла:**

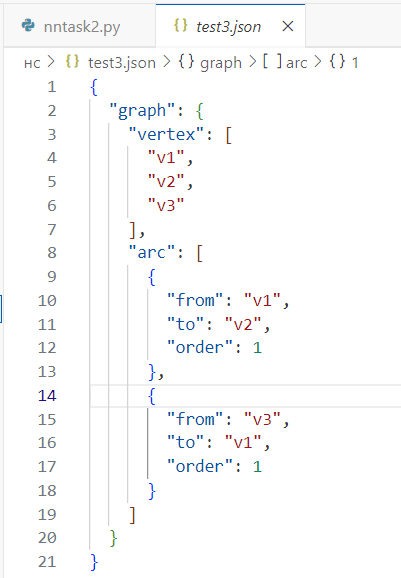


Рисунок 17 – входной файл

**Пример работы программы:**

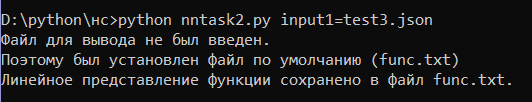


Рисунок 18 – запуск программы в консоли

Линейное представление функции: v2(v1(v3()))

**Пример входного файла:**



Рисунок 19 – входной файл

**Пример работы программы:**

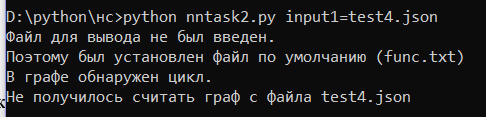


Рисунок 20 – запуск программы в консоли

**Пример входного файла:**

****

Рисунок 21 – входной файл

**Пример работы программы:**

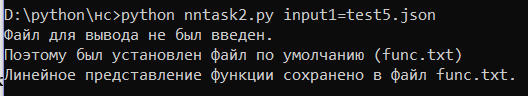


Рисунок 22 – запуск программы в консоли

Линейное представление функции: G(D(A(), A()), F(E(B(), C())))

**Пример входного файла:**



Рисунок 23 – входной файл

**Пример работы программы:**

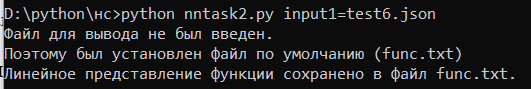


Рисунок 24 – запуск программы в консоли

Линейное представление функции: v6(v5(v4(v1(), v2()), v3()))

# **3. Вычисление значения функции на графе**

## **3.1 Описание задания 3**

**На входе:**

а) Текстовый файл с описанием графа в виде списка дуг (смотри задание 1).

б) Текстовый файл соответствий арифметических операций именам вершин:

где – имя -й вершины, - символ операции, соответствующий вершине .

Допустимы следующие символы операций:

**+** – cумма значений,

**\*** – произведение значений,

***exp*** – экспонирование входного значения,

***число*** – любая числовая константа.

**На выходе:** значение функции, построенной по графу а) и файлу б).

**Способ проверки результата:** результат вычисления, выведенный в файл.

## **3.2 Тестирование задания 3**

**Пример входного файла:**

(v1, v3, 1), (v2, v3, 2)

{

    "v1" : 512,

    "v2" : 2,

    "v3" : "\*"

}



Рисунок 25 – json представление входного файла

Линейное представление функции: v3(v1(), v2())

**Пример работы программы:**

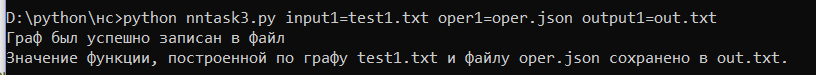


Рисунок 26 – запуск программы в консоли

Вычисленное значение: 1024

**Пример входного файла:**

(v1, v4, 1), (v2, v4, 2), (v2, v5, 1), (v3, v5, 2), (v4, v6, 1), (v5, v6, 2), (v6, v7, 1)

{

    "v1" : 3,

    "v2" : 2,

    "v3" : 5,

    "v4" : "\*",

    "v5" : "+",

    "v6" : "+",

    "v7" : "exp"

}



Рисунок 27 – json представление входного файла

Линейное представление функции: v7(v6(v4(v1(), v2()), v5(v2(), v3())))

**Пример работы программы:**

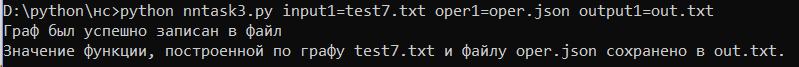


Рисунок 28 – запуск программы в консоли

Вычисленное значение: 442413.3920089205

**Пример входного файла:**

(v1, v2, 1), (v1, v2, 2), (v2, v6, 1), (v3, v5, 1), (v4, v5, 2), (v6, v7, 1), (v5, v7, 2)

{

    "v1" : 1,

    "v2" : "+",

    "v3" : 5,

    "v4" : 12,

    "v5" : "\*",

    "v6" : "exp",

    "v7" : "+"

}



Рисунок 27 – json представление входного файла

Линейное представление функции: v7(v6(v2(v1(), v1())), v5(v3(), v4()))

**Пример работы программы:**

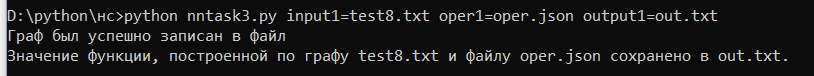


Рисунок 28 – запуск программы в консоли

Вычисленное значение: 67.38905609893065

**Пример входного файла:**

(v1,v4,1),(v2,v4,2),(v4,v5,1),(v3,v5,2),(v5,v6,1)

{

    "v1" : 1,

    "v2" : 10,

    "v3" : 5,

    "v4" : "+",

    "v5" : "\*",

    "v6" : "exp"

}



Рисунок 29 – json представление входного файла

Линейное представление функции: v6(v5(v4(v1(), v2()), v3()))

**Пример работы программы:**

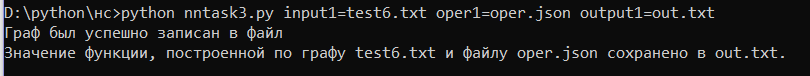


Рисунок 30 – запуск программы в консоли

Вычисленное значение: 7.694785265142018e+23

# **4. Построение многослойной нейронной сети**

## **4.1 Описание задания 4**

**На входе:**

а) Текстовый файл с набором матриц весов межнейронных связей:

М1 : [M1[1,1], M1[1,2],..., M1[1,n]], ..., [M1[m,1], M1[m,2],...,M1[m,n]]

М2 : [M2[1,1], M2[1,2],..., M2[1,n]], ..., [M2[m,1], M2[m,2],...,M2[m,n]]

...

Мp : [Mp[1,1], Mp[1,2],..., Mp[1,n]], ..., [Mp[m,1], Mp[m,2],...,Mp[m,n]]

б) Текстовый файл с входным вектором в формате:

**На выходе:**

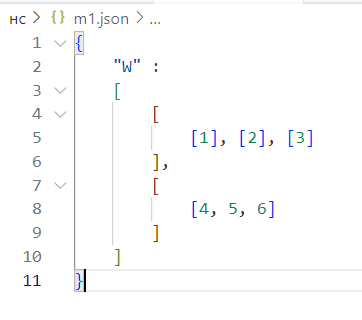
а) Сериализованная многослойная нейронная сеть (в формате XML или JSON) с полносвязной межслойной структурой.

Файл с выходным вектором – результатом вычислений НС в формате:

в) Сообщение об ошибке, если в формате входного вектора или файла описания НС допущена ошибка.

## **4.2 Тестирование задания 4**

**Пример входного файла:**



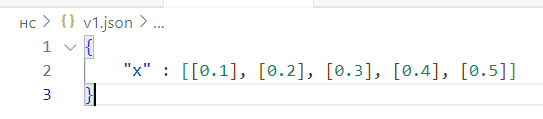
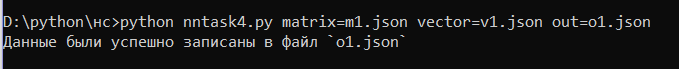


Рисунок 31 – json представления входных файлов матрицы и вектора

**Пример работы программы:**



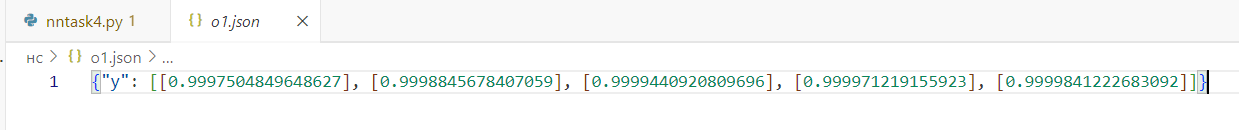
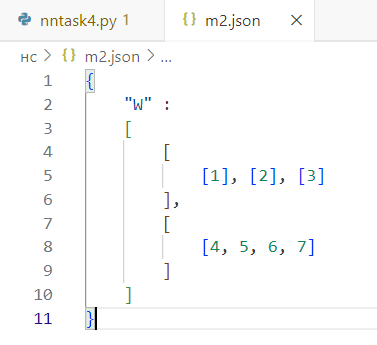


Рисунок 32 – запуск программы в консоли и результирующий вектор

**Пример входного файла:**



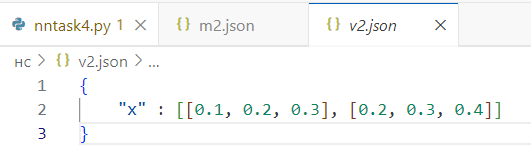


Рисунок 33 – json представления входных файлов матрицы и вектора

**Пример работы программы:**

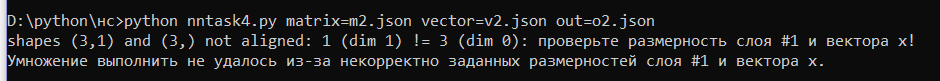
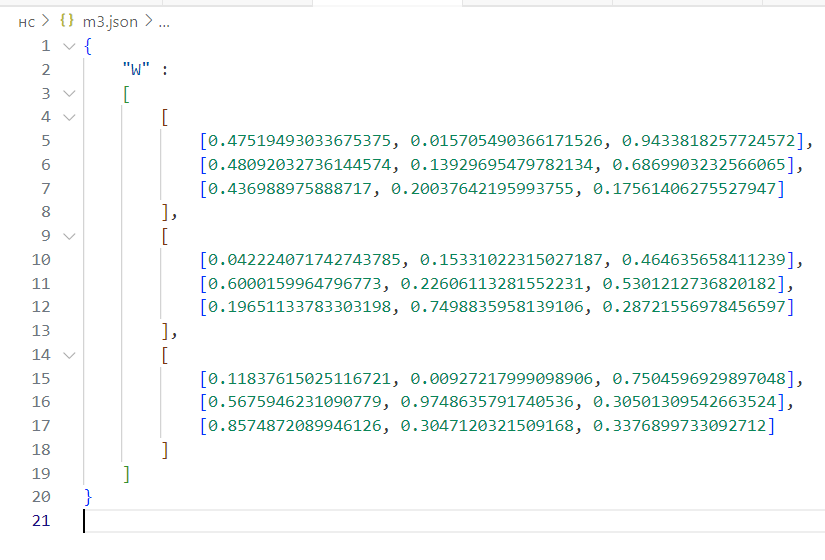


Рисунок 34 – запуск программы в консоли

**Пример входного файла:**



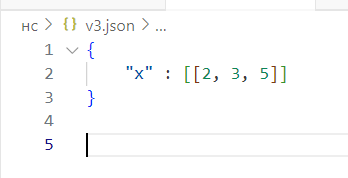
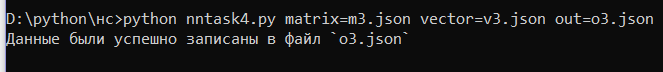


Рисунок 35 – json представления входных файлов матрицы и вектора

**Пример работы программы:**



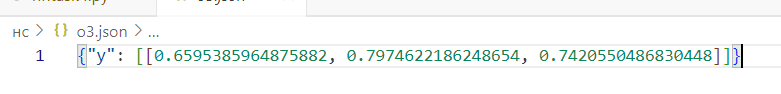


Рисунок 36 – запуск программы в консоли и результирующий вектор

# **5. Построение многослойной нейронной сети**

## **5.1 Описание задания 5**

**На входе:**

а) Текстовый файл с описанием НС (формат см. в задании 4).

б) Текстовый файл с обучающей выборкой:

[x11, x12, ..., x1n] -> [y11, y12, ..., y1m]

...

[xk1, xk2, ..., xkn] -> [yk1, yk2, ..., ykm]

Формат описания входного вектора *x* и выходного вектора *y* соответствует формату из задания 4.

в) Число итераций обучения (в строке параметров).

**На выходе:**

Текстовый файл с историей N итераций обучения методом обратного распространения ошибки:

1 : Ошибка1

2 : Ошибка2

...

N : ОшибкаN

## **5.2 Тестирование задания 5**

**Пример входного файла:**

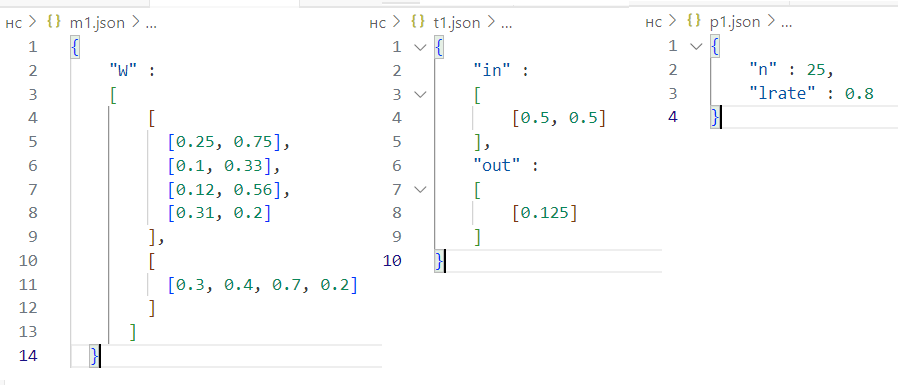
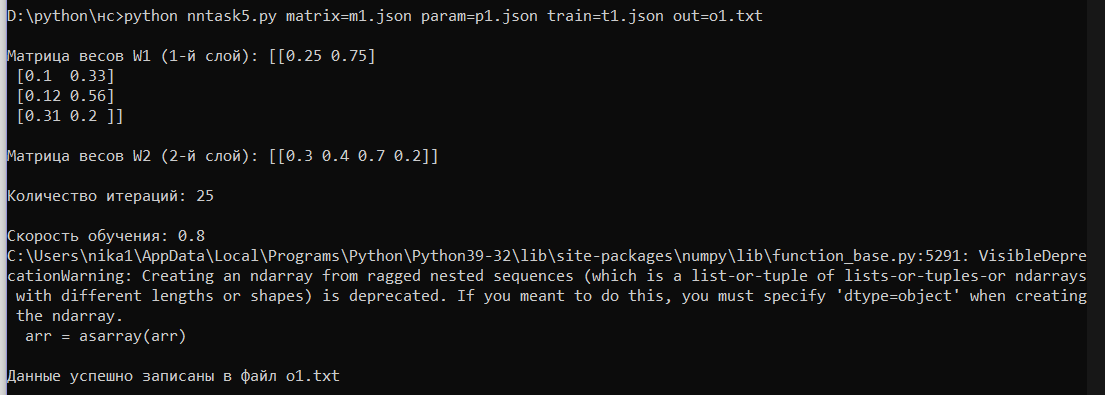


Рисунок 37 – json представления входных файлов матрицы и вектора

**Пример работы программы:**



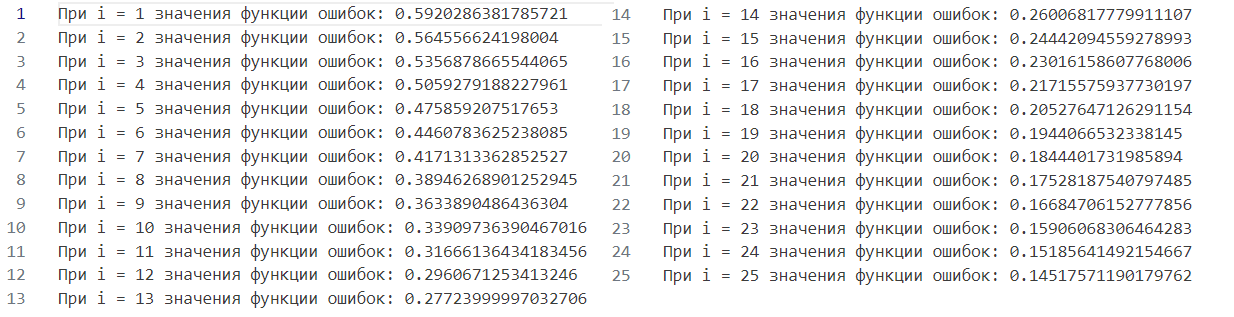


Рисунок 38 – запуск программы в консоли и результирующий файл

**Пример входного файла:**

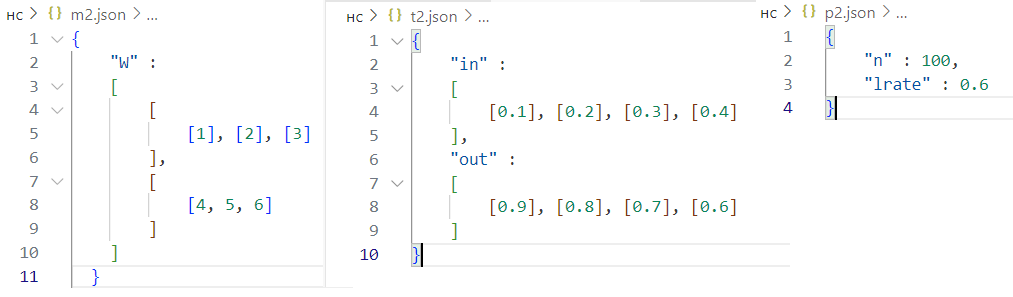
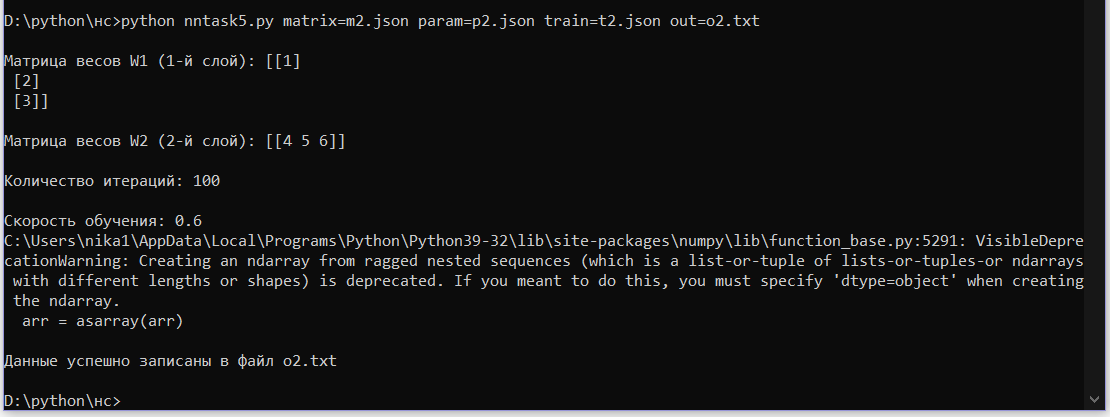


Рисунок 39 – json представления входных файлов матрицы и вектора

**Пример работы программы:**



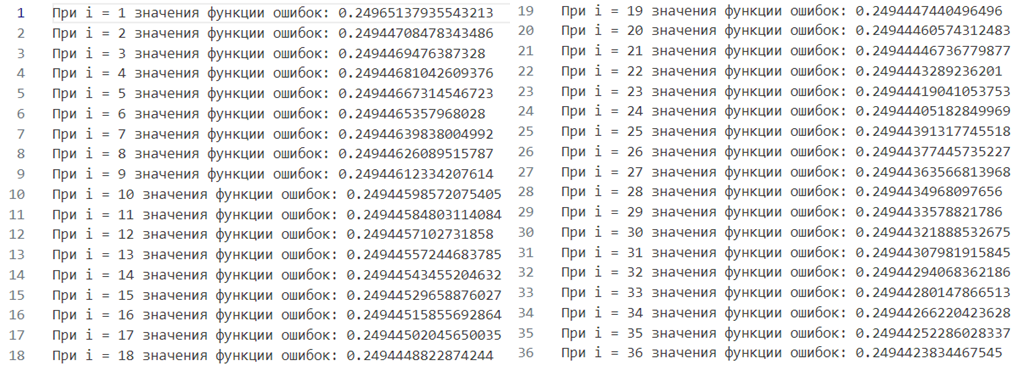
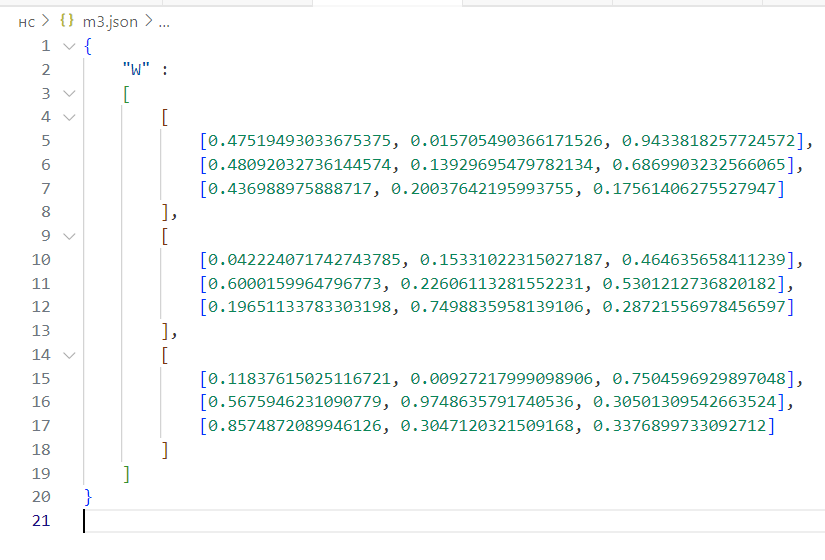


Рисунок 40 – запуск программы в консоли и результирующий файл

**Пример входного файла:**



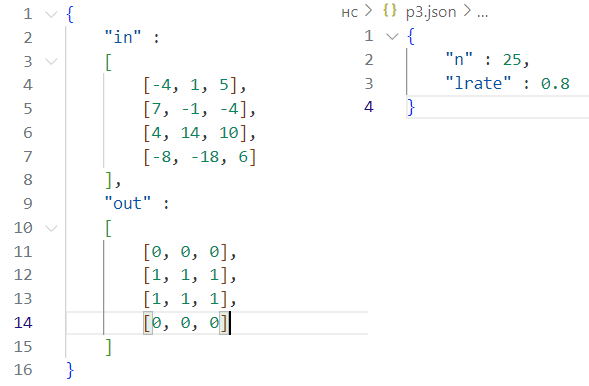
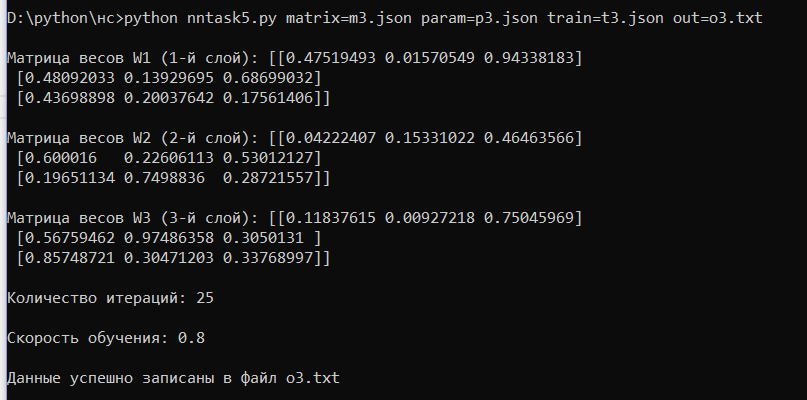


Рисунок 41 – json представления входных файлов матрицы и вектора

**Пример работы программы:**



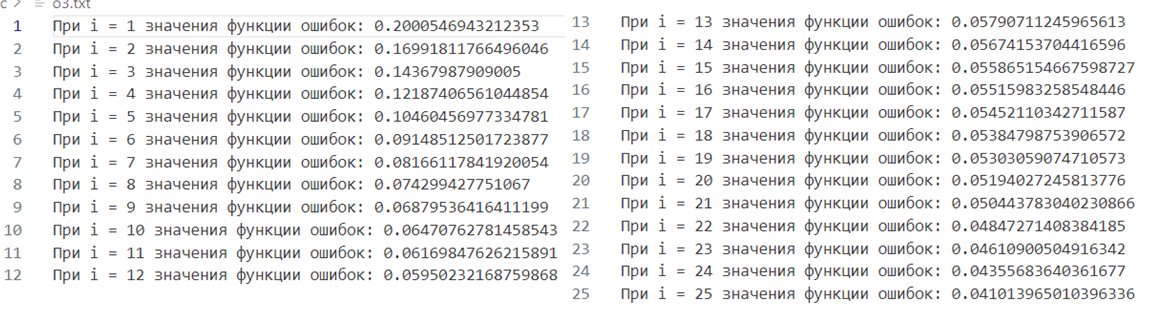


Рисунок 42 – запуск программы в консоли и результирующий файл

# **ПРИЛОЖЕНИЯ**

## Приложение 1 – задание 1

import sys,re

import json

class Graph:

    def \_\_init\_\_(self) -> None:

        self.vertex = set() # множество для хранения всех вершин графа

        self.adjc = {} # словарь для хранения списка дуг, исходящих из каждой вершины

        self.check = {} # словарь для хранения списка дуг, входящих в каждую вершину

    def graph\_construction(self, data : list) -> None:

        for el in data:

            # Добавление вершин в множество vertex

            self.vertex.add(el[0])

            self.vertex.add(el[1])

            # Добавление входящей дуги в словарь check

            if el[1] not in self.check:

                self.check[el[1]] = []

            self.check[el[1]].append((el[0], int(el[2])))

            # Добавление исходящей дуги в словарь adjc

            if el[0] not in self.adjc:

                self.adjc[el[0]] = []

            self.adjc[el[0]].append((el[1], int(el[2])))

        # Сортировка списка вершин

        self.vertex = list(self.vertex)

        self.vertex.sort()

        # Проверка уникальности номеров входящих дуг для каждой вершины

        for \_, vs in self.check.items(): # Проход по каждой вершине в словаре check

            n = len(vs)

            tmp = [''] \* n

            for el, k in vs:

                t = (k - 1) % n

                if tmp[t] == el:

                    tmp[t + 1] = el

                else:

                    tmp[t] = el

            test = set()

            for el in tmp:

                test.add(el)

                if el == '':

                    self.vertex = [-1]

                    print('В графе некорректно заданы номера!\nПроверьте уникальность номеров.')

                    return

class GraphCreation:

    def \_\_init\_\_(self, input, output) -> None:

        self.\_in = input

        self.\_out = output

        self.data = None

        self.graph = None

        self.g = Graph()

    def check\_params(self) -> bool:

        if self.\_in is None:

            return False

        if self.\_out is None:

            self.\_out = f"output.json"

            print(f"Файл для вывода не был введен.\n"

                  f"Поэтому был установлен файл по умолчанию ({self.\_out})")

        return True

    def read\_from\_file(self) -> bool:

        try:

            f = open(self.\_in, 'r', encoding='utf-8')

            self.data = f.read()

            f.close()

        except:

            print(f"Файла `{self.\_in}` не существует. "

                  "Проверьте корректность имени файла.")

            return False

        return True

    def data\_parser(self) -> None:

        rawdata = self.data

        self.data = []

        cnt = 0

        tmp = []

        for el in re.split(r"\W+", rawdata):

            if el.isalnum():

                cnt += 1

                tmp.append(el)

                if cnt == 3:

                    if not tmp[2].isdigit():

                        self.data = []

                        return

                    cnt = 0

                    self.data.append(tmp)

                    tmp = []

    def check\_data(self) -> bool:

        if self.data == []:

            return False

        arcs\_cnt = {}

        arcs = {}

        for el in self.data:

            s = f'{el[0]}-{el[1]}'

            if s not in arcs and s not in arcs\_cnt:

                arcs\_cnt[s] = 0

                arcs[s] = set()

            arcs\_cnt[s] += 1

            arcs[s].add(el[2])

        for k in arcs.keys():

            if len(arcs[k]) != arcs\_cnt[k]:

                return False

        return True

    def get\_graph(self) -> None:

        self.g.graph\_construction(self.data)

        arc = []

        for el in self.g.adjc.keys():

            for tpl in self.g.adjc[el]:

                arc.append({ "from" : el

                           , "to" : tpl[0]

                           , "order" : tpl[1]})

        self.graph = {"graph" : {"vertex" : self.g.vertex, "arc" : arc}}

    def write\_to\_file(self) -> bool:

        try:

            with open(self.\_out, 'w') as f:

                f.write(json.dumps(self.graph, indent=2))

        except:

            print(f"Не удалось записать в файл `{self.\_out}`.\n")

            return False

        return True

    def graph\_creation(self) -> None:

        if self.check\_params():

            if self.read\_from\_file():

                self.data\_parser()

                if not self.check\_data():

                    print('Некорректный ввод данных.\n'

                          'Проверьте уникальность порядковых номеров '

                          'у каждой из дуг.')

                    return

                self.get\_graph()

                if self.g.vertex == [-1]:

                    return

                if self.write\_to\_file():

                    print(f"Граф был успешно записан в файл")

def args\_parser(argv : list) -> None:

    global graphs

    in1, in2, out1, out2 = None, None, None, None

    for el in argv:

        if "input1=" in el:

            in1 = el[el.find("=") + 1:]

        elif "input2=" in el:

            in2 = el[el.find("=") + 1:]

        elif "output1=" in el:

            out1 = el[el.find("=") + 1:]

        elif "output2=" in el:

            out2 = el[el.find("=") + 1:]

    if (in1 is None) and (in2 is None):

        print("Для корректной работы программы необходимо"

              " добавить в качестве аргументов названия файлов"

              " для ввода и вывода.\n"

              "Пример: input1=in.txt output1=out.json\n")

        return False

    if in1 is not None:

        if out1 is None:

            out1 = f"output1.json"

            print(f"Файл для вывода не был введен.\n"

                f"Поэтому был установлен файл по умолчанию ({out1})")

        graphs.append(GraphCreation(in1, out1))

    if in2 is not None:

        if out2 is None:

            out2 = f"output2.json"

            print(f"Файл для вывода не был введен.\n"

                f"Поэтому был установлен файл по умолчанию ({out2})")

        graphs.append(GraphCreation(in2, out2))

    return True

def main() -> None:

    args\_parser(sys.argv)

    for g in graphs:

        g.graph\_creation()

if \_\_name\_\_ == "\_\_main\_\_":

    graphs = []

    main()

## Приложение 2 – задание 2

import json

import sys,re

def args\_parser(argv : list) -> None:

    global graphs

    in1, out1 = None, None

    for el in argv:

        if "input1=" in el:

            in1 = el[el.find("=") + 1:]

        if "output1=" in el:

            out1 = el[el.find("=") + 1:]

    if (in1 is None):

        print("Для корректной работы программы необходимо"

              " добавить в качестве аргументов названия файлов"

              " для ввода и вывода.\n"

              "Пример: input1=in.json output1=out.txt\n")

        return False

    if in1 is not None:

        if out1 is None:

            out1 = f"func.txt"

            print(f"Файл для вывода не был введен.\n"

                f"Поэтому был установлен файл по умолчанию ({out1})")

        return in1, out1

class Node:

    def \_\_init\_\_(self, node, parents=None, child=None) -> None:

        self.node = node

        self.from\_nodes = [] if parents is None else parents

        self.to\_nodes = [] if child is None else child

def get\_graph\_from\_json(json\_graph):

    with open(json\_graph, "r") as f:

        data = json.load(f)

    graph\_data = data['graph']

    graph = {}

    nodes = {}

    for vertex in graph\_data['vertex']:

        graph[vertex] = []

        nodes[vertex] = Node(vertex)

    for arc in graph\_data['arc']:

        from\_vertex = arc['from']

        to\_vertex = arc['to']

        graph[from\_vertex].append(to\_vertex)

        nodes[from\_vertex].to\_nodes.append(to\_vertex)

        nodes[to\_vertex].from\_nodes.append(from\_vertex)

    if cycle\_check(graph):

        print("В графе обнаружен цикл.")

        raise RuntimeError("Цикл в графе.")

    return graph, nodes

def cycle\_check(graph):

    path = set()

    def visit(vertex):

        path.add(vertex)

        for neighbour in graph.get(vertex, ()):

            if neighbour in path or visit(neighbour):

                return True

        path.remove(vertex)

        return False

    return any(visit(v) for v in graph)

def get\_prefix\_func(graph, output):

    def iterates(cur\_node, nodes):

        from\_nodes = [iterates(nodes[p], nodes)

                    for p in cur\_node.from\_nodes]

        return f'{cur\_node.node}({", ".join(from\_nodes)})'

    graph, nodes = graph

    root = None

    result = []

    for vertex in graph.keys():

        if not graph[vertex]:

            root = vertex

            result.append(iterates(nodes[root], nodes))

    with open(output, 'w') as file:

        file.write(", ".join(result))

    print(f"Линейное представление функции сохранено в файл {output}.")

def main():

    try:

        input, output = args\_parser(sys.argv)

    except:

        print("Ошибка чтения аргументов!")

        return 0

    try:

        graph = get\_graph\_from\_json(input)

    except:

        print("Не получилось считать граф с файла", input)

        return 0

    try:

        get\_prefix\_func(graph, output)

    except:

        print("Не получилось составить префиксную фунцкию")

        return 0

if \_\_name\_\_ == "\_\_main\_\_":

    main()

## Приложение 3 – задание 3

import sys,re

import json

import math

import os

def sum(x, y):

    return x + y

def multiply(x, y):

    return x \* y

def exponent(x):

    return math.exp(x)

STRING\_TO\_OPERATION = {

    "+": "sum",

    "\*": "multiply",

    "exp": "exponent"

}

class GraphCreation:

    def \_\_init\_\_(self, input, output) -> None:

        self.\_in = input

        self.\_out = output

        self.data = None

        self.graph = None

        self.g = Graph()

    def check\_params(self) -> bool:

        if self.\_in is None:

            return False

        if self.\_out is None:

            self.\_out = f"output.json"

            print(f"Файл для вывода не был введен.\n"

                  f"Поэтому был установлен файл по умолчанию ({self.\_out})")

        return True

    def read\_from\_file(self) -> bool:

        try:

            f = open(self.\_in, 'r', encoding='utf-8')

            self.data = f.read()

            f.close()

        except:

            print(f"Файла `{self.\_in}` не существует. "

                  "Проверьте корректность имени файла.")

            return False

        return True

    def data\_parser(self) -> None:

        rawdata = self.data

        self.data = []

        cnt = 0

        tmp = []

        for el in re.split(r"\W+", rawdata):

            if el.isalnum():

                cnt += 1

                tmp.append(el)

                if cnt == 3:

                    if not tmp[2].isdigit():

                        self.data = []

                        return

                    cnt = 0

                    self.data.append(tmp)

                    tmp = []

    def check\_data(self) -> bool:

        if self.data == []:

            return False

        arcs\_cnt = {}

        arcs = {}

        for el in self.data:

            s = f'{el[0]}-{el[1]}'

            if s not in arcs and s not in arcs\_cnt:

                arcs\_cnt[s] = 0

                arcs[s] = set()

            arcs\_cnt[s] += 1

            arcs[s].add(el[2])

        for k in arcs.keys():

            if len(arcs[k]) != arcs\_cnt[k]:

                return False

        return True

    def get\_graph(self) -> None:

        self.g.graph\_construction(self.data)

        arc = []

        for el in self.g.adjc.keys():

            for tpl in self.g.adjc[el]:

                arc.append({ "from" : el

                           , "to" : tpl[0]

                           , "order" : tpl[1]})

        self.graph = {"graph" : {"vertex" : self.g.vertex, "arc" : arc}}

    def write\_to\_file(self) -> bool:

        try:

            with open(self.\_out, 'w') as f:

                f.write(json.dumps(self.graph, indent=2))

        except:

            print(f"Не удалось записать в файл `{self.\_out}`.\n")

            return False

        return True

    def graph\_creation(self) -> None:

        if self.check\_params():

            if self.read\_from\_file():

                self.data\_parser()

                if not self.check\_data():

                    print('Некорректный ввод данных.\n'

                          'Проверьте уникальность порядковых номеров '

                          'у каждой из дуг.')

                    return

                self.get\_graph()

                if self.g.vertex == [-1]:

                    return

                if self.write\_to\_file():

                    print(f"Граф был успешно записан в файл")

class Graph:

    def \_\_init\_\_(self) -> None:

        self.vertex = set() # множество для хранения всех вершин графа

        self.adjc = {} # словарь для хранения списка дуг, исходящих из каждой вершины

        self.check = {} # словарь для хранения списка дуг, входящих в каждую вершину

    def graph\_construction(self, data : list) -> None:

        for el in data:

            # Добавление вершин в множество vertex

            self.vertex.add(el[0])

            self.vertex.add(el[1])

            # Добавление входящей дуги в словарь check

            if el[1] not in self.check:

                self.check[el[1]] = []

            self.check[el[1]].append((el[0], int(el[2])))

            # Добавление исходящей дуги в словарь adjc

            if el[0] not in self.adjc:

                self.adjc[el[0]] = []

            self.adjc[el[0]].append((el[1], int(el[2])))

        # Сортировка списка вершин

        self.vertex = list(self.vertex)

        self.vertex.sort()

        # Проверка уникальности номеров входящих дуг для каждой вершины

        for \_, vs in self.check.items(): # Проход по каждой вершине в словаре check

            n = len(vs)

            tmp = [''] \* n

            for el, k in vs:

                t = (k - 1) % n

                if tmp[t] == el:

                    tmp[t + 1] = el

                else:

                    tmp[t] = el

            test = set()

            for el in tmp:

                test.add(el)

                if el == '':

                    self.vertex = [-1]

                    print('В графе некорректно заданы номера!\nПроверьте уникальность номеров.')

                    return

def args\_parser(argv : list) -> None:

    global graphs

    in1, op1, out1 = None, None, None

    for el in argv:

        if "input1=" in el:

            in1 = el[el.find("=") + 1:]

        elif "oper1=" in el:

            op1 = el[el.find("=") + 1:]

        elif "output1" in el:

            out1 = el[el.find("=") + 1:]

    if (in1 is None):

        print("Для корректной работы программы необходимо"

              " добавить в качестве аргументов названия файлов"

              " для ввода и вывода.\n"

              "Пример: input1=in.txt oper1=operations.json output1=out.txt\n")

        return False

    if op1 is not None:

        graphs.append(GraphCreation(in1, 'tmp.json'))

        return in1, op1, out1

    else:

        print('Необходимо также ввести название файла, в котором содержатся операции'

              '\nНапример oper1=op.json')

class Node:

    def \_\_init\_\_(self, node, parents=None, child=None) -> None:

        self.node = node

        self.from\_nodes = [] if parents is None else parents

        self.to\_nodes = [] if child is None else child

def get\_graph\_from\_json(json\_graph):

    with open(json\_graph, "r") as f:

        data = json.load(f)

    graph\_data = data['graph']

    graph = {}

    nodes = {}

    for vertex in graph\_data['vertex']:

        graph[vertex] = []

        nodes[vertex] = Node(vertex)

    for arc in graph\_data['arc']:

        from\_vertex = arc['from']

        to\_vertex = arc['to']

        graph[from\_vertex].append(to\_vertex)

        nodes[from\_vertex].to\_nodes.append(to\_vertex)

        nodes[to\_vertex].from\_nodes.append(from\_vertex)

    if cycle\_check(graph):

        print("В графе обнаружен цикл.")

        raise RuntimeError("Цикл в графе.")

    return graph, nodes

def cycle\_check(graph):

    path = set()

    def visit(vertex):

        path.add(vertex)

        for neighbour in graph.get(vertex, ()):

            if neighbour in path or visit(neighbour):

                return True

        path.remove(vertex)

        return False

    return any(visit(v) for v in graph)

def get\_prefix\_func(graph):

    def iterates(cur\_node, nodes):

        from\_nodes = [iterates(nodes[p], nodes)

                    for p in cur\_node.from\_nodes]

        return f'{cur\_node.node}({", ".join(from\_nodes)})'

    graph, nodes = graph

    root = None

    result = []

    for vertex in graph.keys():

        if not graph[vertex]:

            root = vertex

            result.append(iterates(nodes[root], nodes))

    return(", ".join(result))

def evaluate\_graph(graph\_string, ops):

    graph\_string = graph\_string.replace("()", "")

    for cur\_op in ops:

        operation = None

        if ops[cur\_op] in STRING\_TO\_OPERATION.keys():

            operation = STRING\_TO\_OPERATION[ops[cur\_op]]

        else:

            operation = str(ops[cur\_op])

        graph\_string = graph\_string.replace(cur\_op, operation)

    return eval(graph\_string)

def main():

    try:

        graph\_input, operations\_path, output = args\_parser(sys.argv)

    except:

        print("Ошибка чтения аргументов!")

        return 0

    try:

        for g in graphs:

            g.graph\_creation()

    except:

        print("Не удалось создать граф с помощью входного файла.")

        return 0

    graph = get\_graph\_from\_json("tmp.json")

    os.remove("tmp.json")

    lin\_interpretation = get\_prefix\_func(graph)

    try:

        with open(operations\_path, 'r') as file:

            operations\_dict = file.read()

        operations\_dict = eval(operations\_dict)

        result = evaluate\_graph(lin\_interpretation, operations\_dict)

        with open(output, 'w') as file:

            file.write(str(result))

        print(f"Значение функции, построенной по графу {graph\_input} и файлу" +

            f" {operations\_path} сохранено в {output}.")

    except:

        print("Ошибка сопоставления операций с описанием графа!")

        return 0

if \_\_name\_\_ == "\_\_main\_\_":

    graphs = []

    main()

## Приложение 4 – задание 4

import autograd.numpy as np

import json, sys

class FeedForward:

    def \_\_init\_\_(self, ws) -> None:

        self.ws = ws

        self.n = len(ws)

    def sigmoid(self, x : float) -> float:

            return 1.0 / (1.0 + np.exp(-x))

    def go\_forward(self, x : list) -> (list, list):

        try:

            sum = np.dot(self.ws[0], x)

            y = np.array([self.sigmoid(x) for x in sum])

        except ValueError as e:

            print(f"{e}: проверьте размерность слоя #1 и вектора х!\n"

                  f"Умножение выполнить не удалось из-за некорректно заданных размерностей"

                  f" слоя #1 и вектора x.")

            return [-1]

        for i in range(1, self.n):

            try:

                sum = np.dot(self.ws[i], y)

                y = np.array([self.sigmoid(x) for x in sum])

            except ValueError as e:

                print(f"{e}: проверьте размерность слоя #{i + 1} и вектора y!\n"

                  f"Умножение выполнить не удалось из-за некорректно заданных размерностей"

                  f" слоя #{i + 1} и вектора y.")

                return [-1]

        return y

    def get\_result(self, xs : list):

        ys = []

        for x in xs:

            new\_val = list(self.go\_forward(np.array(x)))

            if new\_val == [-1]:

                ys.clear()

                break

            ys.append(new\_val)

        return ys

def read\_json\_file(name) -> dict:

    try:

        f = open(name)

        data = json.load(f)

        f.close()

    except:

        print(f"Файла `{name}` не существует. "

              "Проверьте корректность имени файла.")

        exit(0)

    return data

def args\_parser(argv : list) -> None:

    in1, in2, out = None, None, None

    for el in argv:

        if "matrix=" in el:

            in1 = el[el.find("=") + 1:]

        elif "vector=" in el:

            in2 = el[el.find("=") + 1:]

        elif "out=" in el:

            out = el[el.find("=") + 1:]

    if (in1 is None) or (in2 is None):

        print("Для корректной работы программы необходимо"

              " добавить в качестве аргументов названия файлов"

              " формата JSON для следующих параметров:\n"

              "matrix= -- файл, где лежат матрицы весов\n"

              "vector= -- файл с входными параметрами\n")

        exit(0)

    return in1, in2, out

def write\_to\_file(data, filename) -> bool:

        try:

            with open(filename, 'w') as f:

                json.dump(data, f)

        except:

            print(f'Ошибка записи данных в файл `{filename}`')

            exit(0)

        else:

            print(f'Данные были успешно записаны в файл `{filename}`')

def main():

    mtrx, vec, out = args\_parser(sys.argv)

    if out is None:

        print("Отсутствует название файла для вывода.\n"

              "Файл для вывода был выбран по умолчанию (output.txt)")

        out = "output.txt"

    WS = read\_json\_file(mtrx)

    XS = read\_json\_file(vec)

    ws = []

    if "W" not in WS.keys():

        print('\nНекорректный ввод данных')

        return

    if "x" not in XS.keys():

        print('\nНекорректный ввод данных')

        return

    for w in WS['W']:

        ws.append(np.array(w))

    c = FeedForward(ws)

    ys = c.get\_result(XS['x'])

    if ys != []:

        write\_to\_file({'y' : ys}, out)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

## Приложение 5 – задание 5

import autograd.numpy as np

import json, sys

class BackPropagation:

    def \_\_init\_\_(self, ws, n, lrate) -> None:

        self.ws = ws

        self.len\_ws = len(ws)

        self.n = n

        self.lrate = lrate

        self.messages = None

    def sigmoid(self, x : float) -> float:

        return 1.0 / (1.0 + np.exp(-x))

    def d\_sigmoid(self, y : float) -> float:

        return y \* (1.0 - y)

    def go\_forward(self, x : list) -> (list, list):

        ys = []

        try:

            sum = np.dot(self.ws[0], x)

            y = np.array([self.sigmoid(x) for x in sum])

            ys.append(y)

        except ValueError as e:

            print(f"{e}: проверьте размерность слоя #1 и вектора х!\n"

                  f"Умножение выполнить не удалось из-за некорректно заданных размерностей"

                  f" слоя #1 и вектора x.")

            exit(0)

        for i in range(1, self.len\_ws):

            try:

                sum = np.dot(self.ws[i], y)

                y = np.array([self.sigmoid(x) for x in sum])

                ys.append(y)

            except ValueError as e:

                print(f"{e}: проверьте размерность слоя #{i + 1} и вектора y!\n"

                  f"Умножение выполнить не удалось из-за некорректно заданных размерностей"

                  f" слоя #{i + 1} и вектора y.")

                exit(0)

        return ys

    def train(self, x\_vec : list, y\_true : list) -> None:

        m = len(x\_vec)

        self.messages = []

        for it in range(1, self.n + 1):

            errors = []

            for k in range(m):

                ys = self.go\_forward(np.array(x\_vec[k]))

                ys = np.insert(ys, 0, np.array(x\_vec[k]), axis=0)

                if len(ys[-1]) != len(y\_true[k]):

                    print(f'\nРазмерность выходного значения алгоритма\n'

                          f'не совпадает с размерностью желаемого выходного состояния {y\_true[k]}')

                    exit(0)

                j = len(ys) - 2

                e = ys[-1] - y\_true[k]

                grad = e \* self.d\_sigmoid(ys[-1])

                self.ws[-1] = self.ws[-1] - self.lrate \* ys[j] \* grad

                for i in range(self.len\_ws - 2, -1, -1):

                    j -= 1

                    grad = self.ws[i + 1] \* grad \* self.d\_sigmoid(ys[j + 1])

                    for t in range(len(self.ws[i])):

                        self.ws[i][t, :] = self.ws[i][t, :] - ys[j] \* grad[:, t] \* self.lrate

                errors.append(sum(e) / len(e))

            self.messages.append(f"При i = {it} значения функции ошибок: {sum(errors) / len(errors)}\n")

def read\_json\_file(name) -> dict:

    try:

        f = open(name)

        data = json.load(f)

        f.close()

    except:

        print(f"Файла `{name}` не существует. "

              "Проверьте корректность имени файла.")

        exit(0)

    return data

def args\_parser(argv : list) -> None:

    in1, in2, in3, out = None, None, None, None

    for el in argv:

        if "matrix=" in el:

            in1 = el[el.find("=") + 1:]

        elif "param=" in el:

            in2 = el[el.find("=") + 1:]

        elif "train=" in el:

            in3 = el[el.find("=") + 1:]

        elif "out=" in el:

            out = el[el.find("=") + 1:]

    if (in1 is None) or (in2 is None) or (in3 is None):

        print("Для корректной работы программы необходимо"

              " добавить в качестве аргументов названия файлов"

              " формата JSON для следующих параметров:\n"

              "matrix= -- файл, где лежат матрицы весов\n"

              "param= -- файл с параметром n (количество итераций)\n"

              "train= -- файл с входными и выходными параметрами\n")

        exit(0)

    return in1, in2, in3, out

def write\_inf(mes : list, filename):

    mes = "".join(mes)

    try:

        f = open(filename, 'w')

        f.write(mes)

        f.close()

    except:

        print(f'Ошибка записи данных в файл {filename}')

        exit(0)

    else:

        print(f'\nДанные успешно записаны в файл {filename}')

def main():

    # elements = []

    mtrx, par, train, out = args\_parser(sys.argv)

    if out is None:

        print("Отсутствует название файла для вывода.\n"

              "Файл для вывода был выбран по умолчанию (output.txt)")

        out = "output.txt"

    mtrx = read\_json\_file(mtrx)

    par = read\_json\_file(par)

    train = read\_json\_file(train)

    if "W" not in mtrx.keys():

        print('\nНекорректный ввод данных!\n'

              'Не удалось считать матрицы весов.')

        return

    if "n" not in par.keys() or "lrate" not in par.keys():

        print('\nНекорректный ввод данных!\n'

              'Не удалось считать параметры')

        return

    if "in" not in train.keys() or "out" not in train.keys():

        print('\nНекорректный ввод данных!\n'

              'Не удалось считать входные/выходные значения')

        return

    ws = []

    for w in mtrx['W']:

        ws.append(np.array(w))

    for i in range(len(ws)):

        print(f"\nМатрица весов W{i + 1} ({i + 1}-й слой): {ws[i]}")

    print(f"\nКоличество итераций: {par['n']}")

    print(f"\nСкорость обучения: {par['lrate']}")

    bp = BackPropagation(ws, par['n'], par['lrate'])

    xs = np.array(train["in"])

    ys = np.array(train["out"])

    bp.train(xs, ys)

    write\_inf(bp.messages, out)

if \_\_name\_\_ == '\_\_main\_\_':

    main()