МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**НЕЙРОННЫЕ СЕТИ**

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ**

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Бочкарева Матвея Сергеевича

|  |  |  |
| --- | --- | --- |
| Научный руководитель  Преподаватель, аспирант | \_\_\_\_\_\_\_\_\_\_ | А. С. Никитина |
|  | подпись, дата |  |

Саратов 2024

**СОДЕРЖАНИЕ**

[1 Создание ориентированного графа 3](#_Toc186155712)

[2 Создание функции по графу 5](#_Toc186155713)

[3 Вычисление значения функции в графе 6](#_Toc186155714)

[4 Построение многослойной нейронной сети 7](#_Toc186155715)

[5 Реализация метода обратного распространения ошибки для многослойной НС 9](#_Toc186155716)

[ПРИЛОЖЕНИЕ А 12](#_Toc186155717)

[ПРИЛОЖЕНИЕ Б 14](#_Toc186155718)

[ПРИЛОЖЕНИЕ В 16](#_Toc186155719)

[ПРИЛОЖЕНИЕ Г 20](#_Toc186155720)

[ПРИЛОЖЕНИЕ Д 22](#_Toc186155721)

# **1 Создание ориентированного графа**

**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:**

1. Ориентированный граф с именованными вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла) – XML-файл.
2. 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>

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



Рисунок 1 – Пример запуска программы

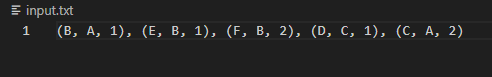


Рисунок 2 – Пример содержимого файла input.txt

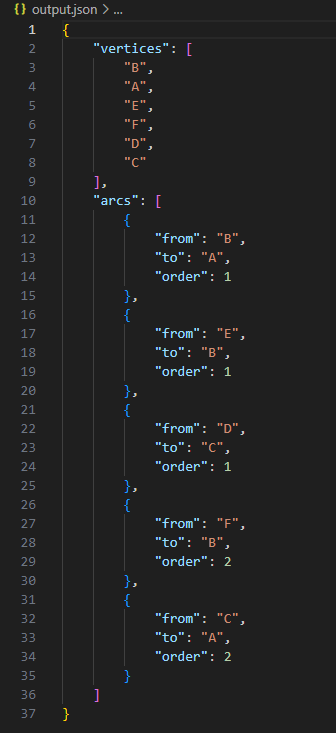


Рисунок 3 – Результат работы программы в файле output.json

Листинг программы представлен в Приложении А.

# **2 Создание функции по графу**

**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.



Рисунок 4 – Пример запуска программы

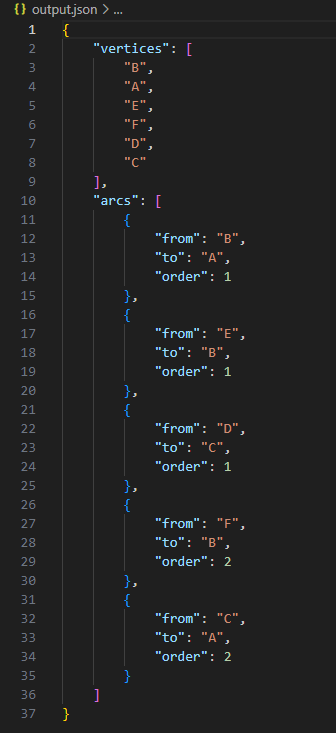


Рисунок 5 – Пример содержимого входного файла output.json



Рисунок 6 – Результат работы программы в файле output.txt

Листинг программы представлен в Приложении Б.

# **3 Вычисление значения функции в графе**

**На входе:**

1. Текстовый файл с описанием графа в виде списка дуг (смотри задание 1).
2. Текстовый файл соответствий арифметических операций именам вершин:

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

Допустимы следующие символы операций:

– cумма значений,

– произведение значений,

– экспонирование входного значения,

– любая числовая константа.

**Выход:** значение функции, построенной по графу и файлу .

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

******

Рисунок 7 – Пример запуска программы

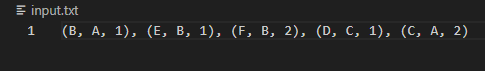


Рисунок 8 – Пример содержимого входного input.txt

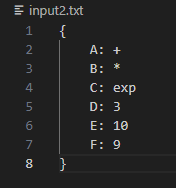


Рисунок 9 – Пример содержимого входного файла input2.txt



Рисунок 10 – Результат работы программы в файле output.txt

Листинг программы представлен в Приложении В.

# **4 Построение многослойной нейронной сети**

**Описание задачи:**

**Вход:**

1. Текстовый файл с набором матриц весов межнейронных связей:

1. Текстовый файл с входным вектором в формате:

**Выход:**

1. Сериализованная многослойная нейронная сеть (в формате или ) с полносвязной межслойной структурой.
2. Файл с выходным вектором – результатом вычислений НС в формате:
3. Сообщение об ошибке, если в формате входного вектора или файла описания НС допущена ошибка.

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

******

Рисунок 11 – Пример запуска программы

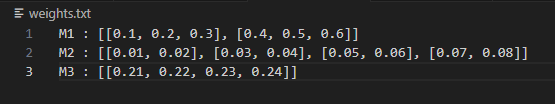


Рисунок 12 – Пример содержимого файла weights.txt

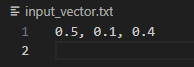


Рисунок 13 – Пример содержимого файла input\_vector.txt

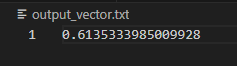


Рисунок 14 – Результат работы программы в файле output\_vector.txt

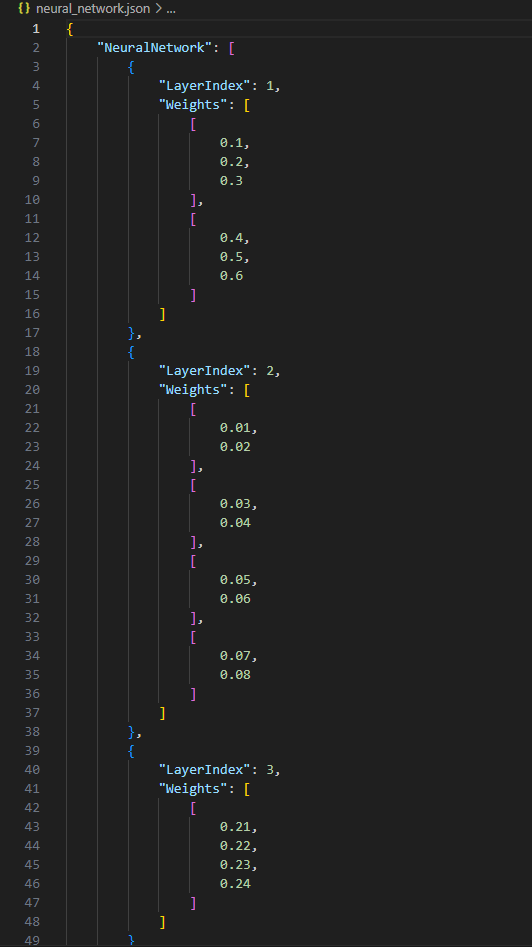


Рисунок 16 – Результат работы програмы в файле neural\_network.json

Листинг программы представлен в Приложении Г.

# **5 Реализация метода обратного распространения ошибки для многослойной НС**

**На входе:**

1. Текстовый файл с набором матриц весов межнейронных связей:
2. Текстовый файл с обучающей выборкой в формате:
3. Число итераций обучения, , .

**На выходе:**

Текстовый файл с историей итераций обучения методом обратного распространения ошибки:

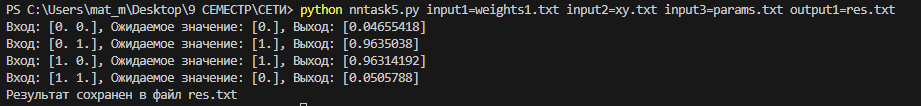


Рисунок 17 – Пример запуска программы

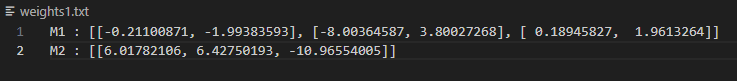


Рисунок 18 – Пример содержимого файла weights1.txt

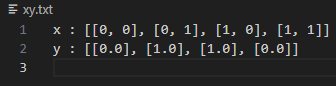


Рисунок 19 – Пример содержимого файла xy.txt

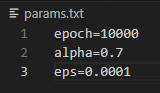


Рисунок 20 – Пример содержимого файла params.txt

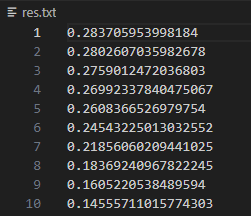


Рисунок 21 – Результат работы программы в начале файла res.txt

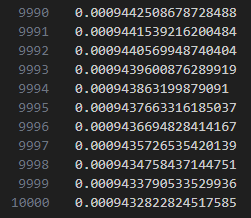


Рисунок 22 – Результат работы программы в конце файла res.txt

Листинг программы представлен в Приложении Д.

# **ПРИЛОЖЕНИЕ А**

**Листинг программы**

import sys

import json

from collections import OrderedDict, defaultdict

def parse\_args():

    args = sys.argv[1:]

    input\_file1 = None

    input\_file2 = None

    output\_file1 = None

    output\_file2 = None

    for arg in args:

        if arg.startswith('input1='):

            input\_file1 = arg[7:]

        elif arg.startswith('input2='):

            input\_file2 = arg[7:]

        elif arg.startswith('output1='):

            output\_file1 = arg[8:]

        elif arg.startswith('output2='):

            output\_file2 = arg[8:]

    if not input\_file1:

        input\_file1 = 'input.txt'

    if not output\_file1:

        output\_file1 = 'output.json'

    return input\_file1, input\_file2, output\_file1, output\_file2

def read\_input\_file(input\_file):

    try:

        with open(input\_file, 'r', encoding='utf-8') as f:

            lines = [line.strip() for line in f.readlines()]

            return lines

    except FileNotFoundError:

        print(f"Ошибка: файл '{input\_file}' не найден.")

        sys.exit(1)

def parse\_graph(lines):

    graph = {

        'vertices': OrderedDict(),

        'arcs': []

    }

    seen\_arcs = set()

    incoming\_orders = defaultdict(set)

    for i, line in enumerate(lines):

        line = line.strip()

        if not line:

            continue

        try:

            arcs\_str = line.split('), (')

            for arc\_str in arcs\_str:

                arc\_str = arc\_str.strip('()')

                a, b, n = arc\_str.split(', ')

                a = a.strip()

                b = b.strip()

                n = int(n.strip())

                arc\_key = (a, b)

                if arc\_key in seen\_arcs:

                    print(f"Ошибка: повторяющаяся дуга ({a}, {b}, {n}) на строке {i+1}.")

                    sys.exit(1)

                seen\_arcs.add(arc\_key)

                if n in incoming\_orders[b]:

                    print(f"Ошибка: в вершину '{b}' входит несколько дуг с порядком {n} на строке {i+1}.")

                    sys.exit(1)

                incoming\_orders[b].add(n)

                if a not in graph['vertices']:

                    graph['vertices'][a] = True

                if b not in graph['vertices']:

                    graph['vertices'][b] = True

                graph['arcs'].append({'from': a, 'to': b, 'order': n})

        except ValueError:

            print(f"Ошибка: неверный формат строки {i+1}: '{line}'")

            sys.exit(1)

    return graph

def write\_json(graph, output\_file):

    graph['arcs'] = sorted(graph['arcs'], key=lambda arc: arc['order'])

    graph\_json = {

        'vertices': list(graph['vertices'].keys()),

        'arcs': graph['arcs']

    }

    try:

        with open(output\_file, 'w', encoding='utf-8') as f:

            json.dump(graph\_json, f, ensure\_ascii=False, indent=4)

    except IOError:

        print(f"Ошибка: не удалось записать файл '{output\_file}'.")

        sys.exit(1)

def main():

    input\_file1, input\_file2, output\_file1, output\_file2 = parse\_args()

    lines = read\_input\_file(input\_file1)

    graph = parse\_graph(lines)

    write\_json(graph, output\_file1)

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

    main()

# **ПРИЛОЖЕНИЕ Б**

**Листинг программы**

import sys

import json

from collections import defaultdict, deque

def parse\_args():

    args = sys.argv[1:]

    input\_file = None

    output\_file = None

    for arg in args:

        if arg.startswith('input1='):

            input\_file = arg[7:]

        elif arg.startswith('output1='):

            output\_file = arg[8:]

    if not input\_file:

        input\_file = 'output.json'

    if not output\_file:

        output\_file = 'output.txt'

    return input\_file, output\_file

def load\_graph(input\_file):

    try:

        with open(input\_file, 'r', encoding='utf-8') as f:

            graph = json.load(f)

            vertices = set(graph['vertices'])

            arcs = graph['arcs']

            return vertices, arcs

    except FileNotFoundError:

        print(f"Ошибка: файл '{input\_file}' не найден.")

        sys.exit(1)

    except json.JSONDecodeError:

        print(f"Ошибка: файл '{input\_file}' имеет некорректный формат.")

        sys.exit(1)

def cycle(vertices, arcs):

    in\_degree = {v: 0 for v in vertices}

    adjacency\_list = defaultdict(list)

    for arc in arcs:

        from\_vertex, to\_vertex = arc['from'], arc['to']

        adjacency\_list[from\_vertex].append(to\_vertex)

        in\_degree[to\_vertex] += 1

    queue = deque([v for v in vertices if in\_degree[v] == 0])

    sorted\_vertices = []

    while queue:

        vertex = queue.popleft()

        sorted\_vertices.append(vertex)

        for neighbor in adjacency\_list[vertex]:

            in\_degree[neighbor] -= 1

            if in\_degree[neighbor] == 0:

                queue.append(neighbor)

    has\_cycle = len(sorted\_vertices) != len(vertices)

    return has\_cycle, sorted\_vertices, adjacency\_list

def find\_root(vertices, arcs):

    outgoing = {arc['from'] for arc in arcs}

    incoming = {arc['to'] for arc in arcs}

    roots = incoming - outgoing

    if len(roots) != 1:

        print("Ошибка: граф должен иметь ровно одну корневую вершину.")

        sys.exit(1)

    return next(iter(roots))

def output(vertices, arcs):

    adjacency\_list = defaultdict(list)

    for arc in arcs:

        from\_vertex, to\_vertex, order = arc['from'], arc['to'], arc['order']

        adjacency\_list[to\_vertex].append((from\_vertex, order))

    for vertex in adjacency\_list:

        adjacency\_list[vertex].sort(key=lambda x: x[1])

    def build\_subtree(vertex):

        if vertex not in adjacency\_list or not adjacency\_list[vertex]:

            return f"{vertex}()"

        children = [build\_subtree(child[0]) for child in adjacency\_list[vertex]]

        return f"{vertex}({', '.join(children)})"

    root = find\_root(vertices, arcs)

    return build\_subtree(root)

def write\_output(output\_file, function\_representation):

    try:

        with open(output\_file, 'w', encoding='utf-8') as f:

            f.write(function\_representation)

    except IOError:

        print(f"Ошибка: не удалось записать файл '{output\_file}'.")

        sys.exit(1)

def main():

    input\_file, output\_file = parse\_args()

    vertices, arcs = load\_graph(input\_file)

    has\_cycle, \_, \_ = cycle(vertices, arcs)

    if has\_cycle:

        print("Ошибка: граф содержит циклы.")

        sys.exit(1)

    function\_representation = output(vertices, arcs)

    write\_output(output\_file, function\_representation)

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

    main()

# **ПРИЛОЖЕНИЕ В**

**Листинг программы**

import sys

import math

from collections import defaultdict, deque

def parse\_args():

    args = sys.argv[1:]

    input\_file1 = None

    input\_file2 = None

    output\_file = None

    for arg in args:

        if arg.startswith('input1='):

            input\_file1 = arg[7:]

        elif arg.startswith('input2='):

            input\_file2 = arg[7:]

        elif arg.startswith('output='):

            output\_file = arg[7:]

    if not input\_file1:

        input\_file1 = 'input.txt'

    if not input\_file2:

        input\_file2 = 'operations.txt'

    if not output\_file:

        output\_file = 'output.txt'

    return input\_file1, input\_file2, output\_file

def parse\_graph(input\_file):

    try:

        with open(input\_file, 'r', encoding='utf-8') as f:

            lines = f.read().strip()

            arcs = []

            vertices = set()

            for item in lines.split('), ('):

                item = item.strip('()')

                a, b, n = item.split(', ')

                a, b, n = a.strip(), b.strip(), int(n)

                arcs.append({'from': a, 'to': b, 'order': n})

                vertices.add(a)

                vertices.add(b)

            return vertices, arcs

    except FileNotFoundError:

        print(f"Ошибка: файл '{input\_file}' не найден.")

        sys.exit(1)

    except ValueError:

        print(f"Ошибка: неверный формат входного файла '{input\_file}'.")

        sys.exit(1)

def load\_operations(input\_file):

    try:

        with open(input\_file, 'r', encoding='utf-8') as f:

            operations = {}

            lines = f.readlines()

            for line in lines[1:-1]:

                line = line.strip()

                if ':' in line:

                    key, value = map(str.strip, line.split(':', 1))

                    operations[key] = value

                else:

                    raise ValueError(f"Неверный формат строки: {line}")

            return operations

    except FileNotFoundError:

        print(f"Ошибка: файл '{input\_file}' не найден.")

        sys.exit(1)

    except ValueError as e:

        print(f"Ошибка: {e}")

        sys.exit(1)

def detect\_cycles(vertices, arcs):

    in\_degree = {v: 0 for v in vertices}

    adjacency\_list = defaultdict(list)

    for arc in arcs:

        from\_vertex, to\_vertex = arc['from'], arc['to']

        adjacency\_list[from\_vertex].append(to\_vertex)

        in\_degree[to\_vertex] += 1

    queue = deque([v for v in vertices if in\_degree[v] == 0])

    sorted\_vertices = []

    while queue:

        vertex = queue.popleft()

        sorted\_vertices.append(vertex)

        for neighbor in adjacency\_list[vertex]:

            in\_degree[neighbor] -= 1

            if in\_degree[neighbor] == 0:

                queue.append(neighbor)

    has\_cycle = len(sorted\_vertices) != len(vertices)

    return has\_cycle, sorted\_vertices

def find\_sink(vertices, arcs):

    outgoing = {arc['from'] for arc in arcs}

    sink\_candidates = vertices - outgoing

    if len(sink\_candidates) != 1:

        print("Ошибка: граф должен иметь ровно одну стоковую вершину.")

        sys.exit(1)

    return next(iter(sink\_candidates))

def evaluate\_function(vertices, arcs, operations):

    adjacency\_list = defaultdict(list)

    for arc in arcs:

        from\_vertex, to\_vertex, order = arc['from'], arc['to'], arc['order']

        adjacency\_list[to\_vertex].append((from\_vertex, order))

    for vertex in adjacency\_list:

        adjacency\_list[vertex].sort(key=lambda x: x[1])

    computed\_values = {}

    def evaluate(vertex):

        if vertex in computed\_values:

            return computed\_values[vertex]

        if vertex not in adjacency\_list or not adjacency\_list[vertex]:

            operation = operations.get(vertex, None)

            if operation is None:

                print(f"Ошибка: операция для вершины '{vertex}' не найдена.")

                sys.exit(1)

            try:

                value = float(operation) if operation.replace('.', '', 1).isdigit() else None

                computed\_values[vertex] = value

                return value

            except ValueError:

                print(f"Ошибка: некорректная операция '{operation}' для вершины '{vertex}'.")

                sys.exit(1)

        children\_values = [evaluate(child[0]) for child in adjacency\_list[vertex]]

        if None in children\_values:

            print(f"Ошибка: не удалось вычислить значение для вершины '{vertex}'.")

            sys.exit(1)

        operation = operations.get(vertex, None)

        if operation is None:

            print(f"Ошибка: операция для вершины '{vertex}' не найдена.")

            sys.exit(1)

        try:

            if operation == '+':

                result = sum(children\_values)

            elif operation == '\*':

                result = math.prod(children\_values)

            elif operation == 'exp':

                if len(children\_values) != 1:

                    print(f"Ошибка: операция 'exp' ожидает ровно одно входное значение для вершины '{vertex}'.")

                    sys.exit(1)

                result = math.exp(children\_values[0])

            else:

                result = float(operation)

            computed\_values[vertex] = result

            return result

        except Exception as e:

            print(f"Ошибка: некорректная операция '{operation}' для вершины '{vertex}': {e}")

            sys.exit(1)

    root = find\_sink(vertices, arcs)

    return evaluate(root)

def write\_output(output\_file, result):

    try:

        with open(output\_file, 'w', encoding='utf-8') as f:

            f.write(str(result))

    except IOError:

        print(f"Ошибка: не удалось записать файл '{output\_file}'.")

        sys.exit(1)

def main():

    input\_file1, input\_file2, output\_file = parse\_args()

    vertices, arcs = parse\_graph(input\_file1)

    operations = load\_operations(input\_file2)

    has\_cycle, \_ = detect\_cycles(vertices, arcs)

    if has\_cycle:

        print("Ошибка: граф содержит циклы.")

        sys.exit(1)

    result = evaluate\_function(vertices, arcs, operations)

    write\_output(output\_file, result)

    print(f"Результат сохранён в '{output\_file}'")

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

    main()

# **ПРИЛОЖЕНИЕ Г**

**Листинг программы**

import numpy as np

import json

import sys

def parse\_args():

    args = sys.argv[1:]

    input\_file1 = None

    input\_file2 = None

    output\_file1 = None

    output\_file2 = None

    for arg in args:

        if arg.startswith('input1='):

            input\_file1 = arg[7:]

        elif arg.startswith('input2='):

            input\_file2 = arg[7:]

        elif arg.startswith('output1='):

            output\_file1 = arg[8:]

        elif arg.startswith('output2='):

            output\_file2 = arg[8:]

    if not input\_file1:

        input\_file1 = 'input1.txt'

    if not input\_file2:

        input\_file2 = 'input2.txt'

    if not output\_file1:

        output\_file1 = 'output\_vector.txt'

    if not output\_file2:

        output\_file2 = 'neural\_network.json'

    return input\_file1, input\_file2, output\_file1, output\_file2

def read\_matrices(file\_path):

    matrices = []

    try:

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

            data = file.read().split("\n")

            for line in data:

                if line.strip():

                    matrix = eval(line.split(":")[1].strip())

                    matrices.append(np.array(matrix))

    except Exception as e:

        raise ValueError(f"Ошибка при чтении матриц: {e}")

    return matrices

def read\_input\_vector(file\_path):

    try:

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

            data = file.read().strip()

            vector = np.array([float(x) for x in data.split(",")])

    except Exception as e:

        raise ValueError(f"Ошибка при чтении входного вектора: {e}")

    return vector

def network(matrices, input\_vector):

    activations = [input\_vector]

    for i, matrix in enumerate(matrices):

        if activations[-1].shape[0] != matrix.shape[1]:

            raise ValueError(

                f"Несоответствие размеров: матрица {i+1} ожидает вход {matrix.shape[1]}, а получила {activations[-1].shape[0]}"

            )

        z = np.dot(matrix, activations[-1])

        a = 1 / (1 + np.exp(-z))

        activations.append(a)

    return activations

def serialize\_to\_json(matrices, output\_file):

    net\_data = {"NeuralNetwork": []}

    for i, matrix in enumerate(matrices):

        layer\_data = {

            "LayerIndex": i + 1,

            "Weights": matrix.tolist()

        }

        net\_data["NeuralNetwork"].append(layer\_data)

    try:

        with open(output\_file, 'w', encoding='utf-8') as file:

            json.dump(net\_data, file, indent=4, ensure\_ascii=False)

    except Exception as e:

        raise ValueError(f"Ошибка при записи JSON: {e}")

def write\_output\_vector(vector, output\_file):

    try:

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

            file.write(", ".join(map(str, vector)))

    except Exception as e:

        raise ValueError(f"Ошибка при записи выходного вектора: {e}")

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

    weight\_file, input\_file, output\_vector\_file, output\_json = parse\_args()

    try:

        matrices = read\_matrices(weight\_file)

        input\_vector = read\_input\_vector(input\_file)

        activations = network(matrices, input\_vector)

        serialize\_to\_json(matrices, output\_json)

        write\_output\_vector(activations[-1], output\_vector\_file)

        print(f"Результаты сохранены в файлы: {output\_json}, {output\_vector\_file}")

    except ValueError as e:

        print(f"Ошибка: {e}")

# **ПРИЛОЖЕНИЕ Д**

**Листинг программы**

import argparse

import sys

import numpy as np

import re

from math import exp

def sigmoid(x):

    return 1 / (1 + exp(-x))

def forward\_pass(layers, inputs):

    for idx, layer in enumerate(layers):

        layer['inputs'] = inputs if idx == 0 else layers[idx - 1]['outputs']

        for i in range(layer['neurons']):

            weighted\_sum = sum(layer['weights'][i][j] \* layer['inputs'][j] for j in range(layer['inputs\_count']))

            layer['outputs'][i] = sigmoid(weighted\_sum)

            layer['derivatives'][i] = layer['outputs'][i] \* (1 - layer['outputs'][i])

    return layers[-1]['outputs']

def backward\_pass(layers, expected):

    total\_error = 0

    deltas = [np.zeros(layer['neurons']) for layer in layers]

    last\_layer = layers[-1]

    for i in range(len(expected)):

        error = last\_layer['outputs'][i] - expected[i]

        deltas[-1][i] = error \* last\_layer['derivatives'][i]

        total\_error += error \*\* 2 / 2

    for idx in range(len(layers) - 1, 0, -1):

        current\_layer = layers[idx]

        for i in range(layers[idx - 1]['neurons']):

            deltas[idx - 1][i] = sum(current\_layer['weights'][j][i] \* deltas[idx][j] for j in range(current\_layer['neurons'])) \* layers[idx - 1]['derivatives'][i]

    return total\_error, deltas

def update\_weights(layers, deltas, learning\_rate):

    for layer, delta in zip(layers, deltas):

        for i in range(layer['neurons']):

            for j in range(layer['inputs\_count']):

                layer['weights'][i][j] -= learning\_rate \* delta[i] \* layer['inputs'][j]

def train(layers, inputs, outputs, max\_epochs, learning\_rate, error\_threshold):

    history = []

    for epoch in range(max\_epochs):

        total\_error = 0

        for x, y in zip(inputs, outputs):

            forward\_pass(layers, x)

            error, deltas = backward\_pass(layers, y)

            total\_error += error

            update\_weights(layers, deltas, learning\_rate)

        mean\_error = total\_error / len(inputs)

        history.append(mean\_error)

        if mean\_error <= error\_threshold:

            break

    for i in range(len(inputs)):

        print(f'Вход: {inputs[i]}, Ожидаемое значение: {outputs[i]}, Выход: {forward\_pass(layers, inputs[i])}')

    return history

def load\_matrix\_file(filename):

    matrices = {}

    try:

        with open(filename, 'r') as file:

            data = file.read()

        matches = re.findall(r"(\S+)\s\*:\s\*\[\[(.\*?)\]\]", data, re.DOTALL)

        for name, matrix\_str in matches:

            matrix\_str = matrix\_str.replace(" ", "").strip(',')

            rows = matrix\_str.split('],[')

            try:

                matrix\_data = [list(map(float, row.strip('[]').split(','))) for row in rows]

                matrices[name] = np.array(matrix\_data)

            except ValueError as e:

                print(f"Ошибка при обработке матрицы {name}: {matrix\_str}. Детали: {e}")

                sys.exit(1)

    except FileNotFoundError:

        print(f"Ошибка, файл не найден: {filename}")

        sys.exit(1)

    except Exception as e:

         print(f"Ошибки при считывании матрицы с файла '{filename}': {e}")

         sys.exit(1)

    return matrices

def load\_parameters\_file(filename):

    try:

        with open(filename, 'r') as file:

            params = {}

            for line in file:

                if '=' in line:

                    key, value = line.split('=', 1)

                    key = key.strip()

                    value = value.strip()

                    try:

                        params[key] = float(value) if '.' in value else int(value)

                    except ValueError:

                         print(f"Ошибка, неверный формат параметра в файле '{filename}'. Параметр '{key}' имеет значение '{value}'")

                         sys.exit(1)

                else:

                    print(f"Ошибка, неверный формат параметра с файла '{filename}': {line.strip()}")

                    sys.exit(1)

            return params

    except FileNotFoundError:

        print(f"Error: File not found: {filename}")

        sys.exit(1)

    except Exception as e:

        print(f"Error loading parameter file '{filename}': {e}")

        sys.exit(1)

def parse\_arguments():

    parser = argparse.ArgumentParser(description="Neural Network Training Script")

    parser.add\_argument('params', nargs='+', type=lambda param: param.split('='))

    try:

        args = parser.parse\_args()

        return {param [0] : param[1] for param in args.params}

    except SystemExit:

         sys.exit(1)

def main():

    args = parse\_arguments()

    if not args:

      sys.exit(1)

    if not all(key in args for key in ['input1', 'input2', 'input3', 'output1']):

        print("Error: Missing parameters. Please provide input1, input2, input3, and output1.")

        sys.exit(1)

    weights\_file = args['input1']

    data\_file = args['input2']

    params\_file = args['input3']

    output\_file = args['output1']

    try:

       weights = load\_matrix\_file(weights\_file)

       layers = [{'weights': w, 'neurons': w.shape[0], 'inputs\_count': w.shape[1], 'inputs': np.zeros(w.shape[1]), 'outputs': np.zeros(w.shape[0]), 'derivatives': np.zeros(w.shape[0])} for w in weights.values()]

       data = load\_matrix\_file(data\_file)

       inputs, outputs = data['x'], data['y']

    except Exception as e:

       print(f"Error: Unable to load data : {e}")

       sys.exit(1)

    if len(inputs) != len(outputs):

        print("Error: Input and output data dimensions do not match.")

        sys.exit(1)

    try:

       params = load\_parameters\_file(params\_file)

    except Exception as e:

        print(f"Error: Unable to load parameters: {e}")

        sys.exit(1)

    try:

      history = train(layers, inputs, outputs, params['epoch'], params['alpha'], params['eps'])

    except Exception as e:

      print(f"Error during training: {e}")

      sys.exit(1)

    try:

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

            file.write("\n".join(map(str, history)))

        print(f"Результат сохранен в файл {output\_file}")

    except Exception as e:

      print(f"Error writing output to '{output\_file}': {e}")

      sys.exit(1)

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

    main()