МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

НЕЙРОННЫЕ СЕТИ

ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ

студента 5 курса 531 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Проверил
доцент _____ И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1 Создание ориентированного графа

1.1 Описание задачи

1.1.1 Вход

Текстовый файл с описанием графа в виде списка дуг

$$(a_1, b_1, n_1), (a_2, b_2, n_2), \ldots, (a_k, b_k, n_k)$$

где a_i – начальная вершина дуги i, b_i – конечная вершина дуги i, n_i – порядковый номер в списке всех заходящих в вершину b_i дуг.

1.1.2 Выход

1. Ориентированный граф с именованными вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла).
2. Сообщение об ошибке в формате файла, если ошибка присутствует.

1.2 Примеры исполнения

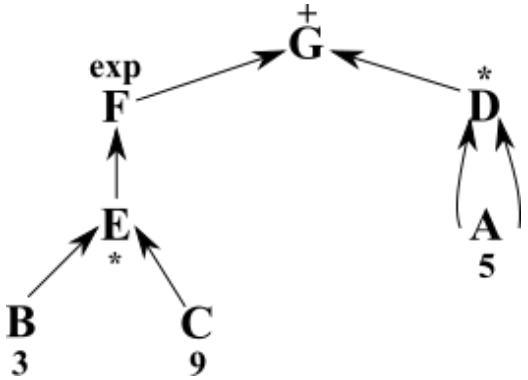

Рисунок 1 – Пример графа

Граф на рисунке 1 задаётся в виде

$$(A, D, 1), (A, D, 2), (B, E, 1), (C, E, 2), (D, G, 1), (E, F, 1), (F, G, 2)$$

В результате программа вернёт сериализованную структуру графа в формате XML

```
<graph>
<vertex>A</vertex>
<vertex>B</vertex>
<vertex>C</vertex>
<vertex>D</vertex>
```

```
  <vertex>E</vertex>
  <vertex>F</vertex>
  <vertex>G</vertex>
  <arc>
    <from>A</from>
    <to>D</to>
    <order>1</order>
  </arc>
  <arc>
    <from>B</from>
    <to>E</to>
    <order>1</order>
  </arc>
  <arc>
    <from>D</from>
    <to>G</to>
    <order>1</order>
  </arc>
  <arc>
    <from>E</from>
    <to>F</to>
    <order>1</order>
  </arc>
  <arc>
    <from>A</from>
    <to>D</to>
    <order>2</order>
  </arc>
  <arc>
    <from>C</from>
    <to>E</to>
    <order>2</order>
  </arc>
  <arc>
    <from>F</from>
    <to>G</to>
    <order>2</order>
  </arc>
</graph>
```

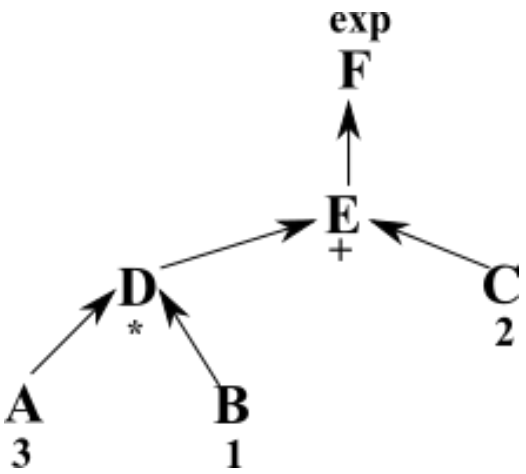

Рисунок 2 – Пример графа

Граф на рисунке 2 задаётся в виде

$$(A, D, 1), (B, D, 2), (D, E, 1), (C, E, 2), (E, F, 1)$$

В результате программа вернёт сериализованную структуру графа в формате XML

```
<graph>
<vertex>A</vertex>
<vertex>B</vertex>
<vertex>C</vertex>
<vertex>D</vertex>
<vertex>E</vertex>
<vertex>F</vertex>
<arc>
  <from>A</from>
  <to>D</to>
  <order>1</order>
</arc>
<arc>
  <from>D</from>
  <to>E</to>
  <order>1</order>
</arc>
<arc>
  <from>E</from>
  <to>F</to>
  <order>1</order>
</arc>
<arc>
  <from>B</from>
  <to>D</to>
  <order>2</order>
```

```
  </arc>
  <arc>
    <from>C</from>
    <to>E</to>
    <order>2</order>
  </arc>
</graph>
```

2 Создание функции по графу

2.1 Описание задачи

2.1.1 Вход

Ориентированный граф с именованными вершинами как описано в задании 1.

2.1.2 Выход

Линейное представление функции, реализуемой графом в префиксной скобочной записи:

$$A_1(B_1(C_1(\dots), \dots, C_m(\dots)), \dots, B_n(\dots))$$

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 Описание работы программы

Циклы в графе обнаруживаются при помощи алгоритма **DFS**.

Далее для каждой вершины сохраняется список её родителей и детей, находится корневая вершина – вершина без детей. Затем от корневой вершины вызывается рекурсивная функция:

```
def make_func(v, vs):
  parents = [make_func(c, vs) for c in v.parents]
  return f'{v.name}({", ".join(parents)})'
```

Эта функция и создаёт функцию по графу в префиксной скобочной записи.

2.3 Примеры исполнения

По графу на рисунке 1 функция будет иметь вид:

$$G(D(A(), A()), F(E(B(), C())))$$

По графу на рисунке 2 функция будет иметь вид:

$$F(E(D(A(), B()), C()))$$

3 Вычисление значения функции на графе

3.1 Описание задачи

3.1.1 Вход

1. Текстовый файл с описанием графа в виде списка дуг.
2. Текстовый файл соответствий арифметических операций именам вершин:
   — a_1: операция 1,
   — ...,
   — a_n: операция n.

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

   Допустимы следующие символы операций:

— \+ – сумма значений,
— * – произведение значений,
— exp – экспонирование входного значения,
— число – любая числовая константа.

3.1.2 Выход

Значение функции, построенной по графу 1) и файлу 2).

Способ проверки результата: результат вычисления, выведенный в файл.

3.2 Описание работы программы

Для каждой вершины определяется либо её числовое значение, либо операция и проверяется, что число аргументов, то есть число родителей соответствует операции. У вершины с числовым значением не должно быть родителей.

Затем от корневой вершины вызывается рекурсивная функция вычисления значения функции графа:

```
def evaluate(v):
    if v.operation is None:
        return v.value

    if v.operation == '+':
        result = 0
        for p in v.parents:
            result += evaluate(p)
        return result
```

```
    if v.operation == '*':
        result = 1
        for p in v.parents:
            result *= evaluate(p)
        return result

    if v.operation == 'exp':
        return exp(evaluate(v.parents[0]))
```

3.3 Примеры исполнения

На рисунках 1 и 2 подписаны заданные операции. Для первого результат будет – 532048240626.7986, а для второго – 1096.6331584284585.

4 Построение многослойной нейронной сети

4.1 Описание задачи

4.1.1 Вход

1. Текстовый файл с набором матриц весов межнейронных связей.
2. Текстовый файл с входным вектором.

4.1.2 Выход

1. Файл с выходным вектором – результатом вычислений НС.
2. Сообщение об ошибке, если в формате входного вектора или файла описания НС допущена ошибка.

4.2 Описание работы программы

Выход каждого нейрона в слое вычисляется следующим образом. Сначала вычисляется неактивированное состояние $Y = W \cdot X$, где W – матрица весовых коэффициентов слоя, X – входной вектор. Затем Y передаётся в функцию активации, в частности, сигмоиду:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Далее активированные значения y_i из Y передаются на следующий слой в качестве входного вектора. Полученные выходные значения на последнем слое и будут результатом вычисления нейронной сети.

4.3 Пример исполнения

На вход подаются матрицы весов:

0.1	0.5	0.9
0.2	0.4	0.8
0.1	0.1	0.1
0.3	0.6	0.9
0.0	0.5	1.0

Таблица 1 – Матрица весов первого слоя

В 1 слое 5 нейронов с 3 входами.

0.1	0.2	0.3	0.4	0.5
0.15	0.25	0.35	0.45	0.55
0.9	0.8	0.7	0.6	0.5

Таблица 2 – Матрица весов второго слоя

Во 2 слое 3 нейрона с 5 входами.

Входной вектор:

$$X = [2.47, 1.2, 9.6]$$

Выходной вектор:

$$Y = [0.8079936459041838, 0.8424440351919794, 0.9662026046584478]$$

5 Реализация метода обратного распространения ошибки для многослойной НС

5.1 Описание задачи

5.1.1 Вход

1. Текстовый файл с описанием НС.
2. Текстовый файл с обучающей выборкой.
3. Текстовый файл с параметрами обучения (скорость обучения, погрешность, колличество итераций).

5.1.2 Выход

1. Текстовый файл с историей итераций обучения методом обратного распространения ошибки.
2. Результирующий вектор, полученный после обучения НС.

5.2 Описание работы программы

В предыдущем задании НС выдаёт «случайные» значения, зависящие от заданных матриц весов и входного вектора. Однако требуется, чтобы ответы сети удовлетворяли поставленной задаче, поэтому её необходимо обучить.

5.2.1 Обратное распространение ошибки

Для обучения будет использоваться обратное распространение ошибки, работающее по следующему алгоритму:

1. Подать на вход сети обучающий пример (один входной вектор).
2. Распространить сигнал по сети вперёд (получить выход сети).
3. Вычислить ошибку (разница получившегося и ожидаемого векторов).
4. Распространить ошибку на предыдущие слои.
5. Обновить весовые коэффициенты для уменьшения ошибки.

Алгоритм останавливается, когда будет достигнута заданная точность или будет исчерпано число итераций.

Для каждого слоя сохраняется его вход x, выход z и производная функции df активации, а так же вектор дельт.

В качестве функции оценки сети $E(W)$ используется среднее квадратичное отклонение

$$E = \frac{1}{2} \cdot \sum (y_{1i} - y_{2i})^2.$$

Чтобы найти значение ошибки E, нужно найти сумму квадратов разности значений вектора, который выдала сеть в качестве ответа, и вектора, который ожидается при обучении.

Также нужно найти дельту для каждого слоя, причём для последнего слоя она будет равна вектору разности полученного и ожидаемого векторов, умноженному (покомпонентно) на вектор значений производных последнего слоя:

$$\delta_{last} = (z_{last} - d) \cdot f'_{last},$$

где z_{last} – выход последнего слоя сети, d – ожидаемый вектор сети, $f`_last$ – вектор значений производной функции активации последнего слоя.

Чтобы найти дельты предыдущих слоёв:

$$\delta_{k-1} = W_k^T \cdot \delta_k \cdot f'_k,$$

где W_k^T – транспонированная матрица весов k-го слоя.

5.2.2 Изменение весов

Для того, чтобы уменьшить ошибку сети нужно изменить весовые коэффициенты каждого слоя. Для этого используется метод градиентного спуска. Градиент по весам равен перемножению входного вектора и вектора дельт. Поэтому, чтобы обновить весовые коэффициенты и уменьшить тем самым ошибку сети нужно вычесть из матрицы весов результат перемножения дельт и входных векторов, умноженный на скорость обучения.

$$W_k = W_k - \alpha \cdot \delta_k \cdot X_k,$$

где W_k – матрица весов k-го слоя, α – скорость обучения, δ_k - дельта k-го слоя, X_k – входной вектор k-го слоя.

5.3 Примеры исполнения

Первый тест представляет собой обучение НС функции xor. В первом слое два нейрона с 2 входами, во втором слое один нейрон с 1 входом. В приложении E можно увидеть входные файлы и выдержку из истории обучения. В результате обучения НС выдала такой результат:

```
X: [0 0], Y: [0.], out: [0.07254825]
X: [0 1], Y: [1.], out: [0.87228817]
```

```
X: [1 0], Y: [1.], out: [0.87094845]
X: [1 1], Y: [0.], out: [0.13987849]
```

Были получены значения, близкие к истинным.

Во втором тесте НС должна угадать, какое число изображено на табло.

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15

Таблица 3 – Матрица с номерами пикселей

Например, если загадано число 4, то будут гореть пиксели 1, 3, 4, 6, 7, 8, 9, 12, 15. В качестве входного вектора задаётся вектор длины 15, где если пиксель с номером $1 \geq i \geq 15$ горит, то на i-м месте стоит 1, иначе 0. То есть число 4 будет иметь вид:

$$[1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1]$$

Поскольку используется сигмоидная функция в качестве функции активации, ответы могут принимать значения от 0 до 1, поэтому эталонными значениями тоже будут числа от 0 до 1. То есть число 4 задаётся в виде 0.4.

В первом слое пять нейронов с 15 входами, во втором три нейрона с 5 входами и на третьем один нейрон с 3 входами.

В результате обучения НС выдала такой результат:

```
X: [1 1 1 1 0 1 1 0 1 1 0 1 1 1 1], Y: [0.], out: [0.00650746]
X: [0 0 1 0 0 1 0 0 1 0 0 1 0 0 1], Y: [0.1], out: [0.10080306]
X: [1 1 1 0 0 1 1 1 1 1 0 0 1 1 1], Y: [0.2], out: [0.2023985]
X: [1 1 1 0 0 1 1 1 1 0 0 1 1 1 1], Y: [0.3], out: [0.30715552]
X: [1 0 1 0 0 1 1 1 1 0 0 1 0 0 1], Y: [0.4], out: [0.40414067]
X: [1 1 1 1 0 0 1 1 1 0 0 1 1 1 1], Y: [0.5], out: [0.49959161]
X: [1 1 1 1 0 0 1 1 1 1 0 1 1 1 1], Y: [0.6], out: [0.59926368]
X: [1 1 1 0 0 1 0 0 1 0 0 1 0 0 1], Y: [0.7], out: [0.69969015]
X: [1 1 1 1 0 1 1 1 1 1 0 1 1 1 1], Y: [0.8], out: [0.80022029]
X: [1 1 1 1 0 1 1 1 1 0 0 1 1 1 1], Y: [0.9], out: [0.90026761]
```

ПРИЛОЖЕНИЕ А

Листинг задания 1

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import argparse
from lxml import etree as et


def parse_param_value(param_value_str):
    parts = param_value_str.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неправильный формат.')
    return parts[0], parts[1]


# TODO: ещё подумать как это красивее сделать
def create_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_param_value,
                        help='input1=arcs1.txt output1=output1.xml')
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    output1 = params.get('output1')
    return input1, output1


def parse_input(input1):
    with open(input1, 'r') as in1:
        str = in1.read()

    str = ''.join(str.split())
    arcs = [tuple(x.split(',')) for x in str[1:-1].split('),(')]
    arcs = [(x[0], x[1], int(x[2])) for x in arcs]
    arcs = sorted(arcs, key=lambda x: x[2])

    visits = dict()
    vs = set()
    for arc in arcs:
        vs.add(arc[0])
        vs.add(arc[1])
        if arc[1] not in visits.keys():
            visits[arc[1]] = [arc[2]]
        else:
            visits[arc[1]].append(arc[2])

    for orders in visits.values():
```

```
46          if list(set(orders)) != orders:
47              raise ValueError('Неправильно заданы порядковые номера заходящих дуг.')
48
49      return arcs, sorted(vs)
50
51
52  def make_xml(arcs, vs, output1):
53      graph = et.Element('graph')
54      for v in vs:
55          et.SubElement(graph, 'vertex').text = v
56      for a in arcs:
57          arc = et.SubElement(graph, 'arc')
58          et.SubElement(arc, 'from').text = a[0]
59          et.SubElement(arc, 'to').text = a[1]
60          et.SubElement(arc, 'order').text = str(a[2])
61
62      tree = et.ElementTree(graph)
63      tree.write(output1, encoding='utf-8', pretty_print=True)
64      print('Граф успешно записан в', output1)
65
66
67  # python3 nntask1.py input1=arcs1.txt output1=digraph1.xml
68  def main():
69      input1, output1 = create_parser()
70      arcs, vs = parse_input(input1)
71      make_xml(arcs, vs, output1)
72
73  main()
```

ПРИЛОЖЕНИЕ Б

Листинг задания 2

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import argparse
from lxml import etree as et


def parse_param_value(param_value_str):
    parts = param_value_str.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неправильный формат.')
    return parts[0], parts[1]


def create_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_param_value,
                        help='input1=arcs1.txt output1=output1.xml')
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    output1 = params.get('output1')
    return input1, output1


class Vertex:
    def __init__(self, v):
        self.name = v
        self.children = []
        self.parents = []


def read_graph(xml_file):
    tree = et.parse(xml_file)
    root = tree.getroot()
    edges, color = dict(), dict()
    vs = dict()

    i = 0
    for elem in root.iter("vertex"):
        v = elem.text
        vs[v] = Vertex(v)
        edges[v] = []
        color[v] = 0
        i += 1

```

```
    for elem in root.iter("arc"):
        v0 = vs[elem[0].text]
        v1 = vs[elem[1].text]
        v0.children.append(v1)
        v1.parents.append(v0)

        edges[elem[0].text].append(elem[1].text)

    dfs(list(edges.keys())[0], edges, color)

    return edges, vs


def dfs(v, edges, color):
    color[v] = 1
    for u in edges[v]:
        if color[u] == 0:
            dfs(u, edges, color)
        if color[u] == 1:
            print('Найден цикл. Работа программы остановлена.')
            exit()
    color[v] = 2


def find_root(edges, vs):
    for v in edges:
        if not edges[v]:
            return vs[v]


def make_func(v, vs):
    parents = [make_func(c, vs) for c in v.parents]
    return f'{v.name}({", ".join(parents)})'


def main():
    input1, output1 = create_parser()
    edges, vs = read_graph(input1)
    root = find_root(edges, vs)
    z = make_func(root, vs)
    with open(output1, 'w') as out:
        out.write(z)
        print(f'Префиксная функция успешно записана в {output1}')


main()
```

ПРИЛОЖЕНИЕ В

Листинг задания 3

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import argparse
from lxml import etree as et
from math import exp


def parse_param_value(param_value_str):
    parts = param_value_str.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неправильный формат.')
    return parts[0], parts[1]


def create_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_param_value)
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    input2 = params.get('input2')
    output1 = params.get('output1')
    return input1, input2, output1


class Vertex:
    def __init__(self, v):
        self.name = v
        self.parents = []
        self.operation = None
        self.value = None


def read_graph(xml_file, ops_file):
    tree = et.parse(xml_file)
    root = tree.getroot()
    edges, color = dict(), dict()
    vs = dict()

    i = 0
    for elem in root.iter("vertex"):
        v = elem.text
        vs[v] = Vertex(v)
        edges[v] = []
        color[v] = 0
```

```
46          i += 1
47
48      for elem in root.iter("arc"):
49          v0 = vs[elem[0].text]
50          v1 = vs[elem[1].text]
51          v1.parents.append(v0)
52
53          edges[elem[0].text].append(elem[1].text)
54
55      dfs(list(edges.keys())[0], edges, color)
56
57      ops = open(ops_file, 'r')
58      while True:
59          line = ops.readline()
60          if not line:
61              break
62          op = line[2:].strip()
63          v = vs[line[0]]
64          if op.isdigit():
65              v.value = int(op)
66              if v.parents:
67                  print(f'{v.name} не должна иметь родителей (аргументов)!')
68                  exit()
69          else:
70              v.operation = op
71              if op == '+' or op == '*':
72                  if len(v.parents) != 2:
73                      print(f'{v.name} должна иметь 2 родителей (аргументов)!')
74                      exit()
75              elif op == 'exp' and len(v.parents) != 1:
76                  print(f'{v.name} должна иметь 1 родителя (аргумент)!')
77                  exit()
78
79      return edges, vs
80
81
82  def dfs(v, edges, color):
83      color[v] = 1
84      for u in edges[v]:
85          if color[u] == 0:
86              dfs(u, edges, color)
87          if color[u] == 1:
88              print('Найден цикл. Работа программы остановлена')
89              exit()
90      color[v] = 2
91
92
93  def find_root(edges, vs):
```

```
    for v in edges:
        if not edges[v]:
            return vs[v]


def evaluate(v):
    if v.operation is None:
        return v.value

    if v.operation == '+':
        result = 0
        for p in v.parents:
            result += evaluate(p)
        return result

    if v.operation == '*':
        result = 1
        for p in v.parents:
            result *= evaluate(p)
        return result

    if v.operation == 'exp':
        return exp(evaluate(v.parents[0]))


def main():
    input1, input2, output1 = create_parser()
    edges, vs = read_graph(input1, input2)
    root = find_root(edges, vs)
    z = evaluate(root)
    with open(output1, 'w') as out:
        print(f'Значение {z} функции записано в {output1}')
        out.write(str(z))

main()
```

ПРИЛОЖЕНИЕ Г

Листинг задания 4

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import argparse
from math import exp
import json
import numpy as np


class Network:
    class Layer:
        def __init__(self, w):
            self.w = np.array(w)
            shape = self.w.shape
            self.n = shape[0]
            self.m = shape[1]
            self.x = None
            self.z = None

    def __init__(self, input1, input2, output1):
        self.layers = []
        weights = read_json(input1)
        for w in weights:
            self.layers.append(Network.Layer(weights[w]))
        self.layers[0].x = np.array(read_json(input2)['x'])
        self.p = len(self.layers)

    def forward(self):
        for k in range(self.p):
            l = self.layers[k]
            print(l.w, l.x)
            y = np.dot(l.w, l.x)
            print('y', y)
            for i in range(l.n):
                y[i] = sigmoid(y[i])
            l.z = y
            if k < self.p - 1:
                self.layers[k + 1].x = y
        return self.layers[-1].z


def sigmoid(x):
    return 1 / (1 + exp(-x))


def parse_param_value(param_value_str):
```

```
46      parts = param_value_str.split('=')
47      if len(parts) != 2:
48          raise argparse.ArgumentTypeError('Неправильный формат.')
49      return parts[0], parts[1]
50
51
52  def create_parser():
53      parser = argparse.ArgumentParser()
54      parser.add_argument('params', nargs='+', type=parse_param_value)
55      args = parser.parse_args()
56      params = dict(args.params)
57      input1 = params.get('input1')
58      input2 = params.get('input2')
59      output1 = params.get('output1')
60      return input1, input2, output1
61
62
63  def write_json(filename, data):
64      with open(filename, 'w') as f:
65          json.dump(data, f)
66      print(f'В {filename} было записано {data}')
67
68
69  def read_json(filename):
70      with open(filename, 'r') as f:
71          data = json.load(f)
72      return data
73
74
75  # python3 nntask4.py input1='W.json' input2='X.json' output1='Y.json'
76  def main():
77      input1, input2, output1 = create_parser()
78      network = Network(input1, input2, output1)
79      write_json(output1, {'y': list(network.forward())})
80
81
82  main()
```

ПРИЛОЖЕНИЕ Д

Листинг задания 5

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-
import argparse
from math import exp
import json
import numpy as np


class Network:
    class Layer:
        def __init__(self, w):
            self.w = np.array(w)
            shape = self.w.shape
            self.n = shape[0]
            self.m = shape[1]
            self.x = np.zeros(self.m)
            self.z = np.zeros(self.n)
            self.df = np.zeros(self.n)

    def __init__(self, input1, input2, input3):
        self.layers = []
        weights = read_json(input1)
        for w in weights:
            self.layers.append(Network.Layer(weights[w]))
        self.p = len(self.layers)

        self.deltas = [[]] * self.p
        self.x = np.array(read_json(input2)['x'])
        self.y = np.array(read_json(input2)['y'])
        self.max_epochs = int(read_json(input3)['iters'])
        self.alpha = float(read_json(input3)['alpha'])
        self.eps = float(read_json(input3)['eps'])

    def forward(self, input):
        for k in range(self.p):
            l = self.layers[k]
            if k == 0:
                l.x = input

            for i in range(l.n):
                y = 0
                for j in range(l.m):
                    y += l.w[i][j] * l.x[j]

                l.z[i] = sigmoid(y)
```

```
                l.df[i] = l.z[i] * (1 - l.z[i])
            if k < self.p - 1:
                self.layers[k + 1].x = l.z

        return self.layers[-1].z

    def backward(self, output):
        error = 0
        last = self.layers[-1]
        olen = len(output)
        self.deltas[-1] = np.zeros(olen)
        for i in range(olen):
            e = last.z[i] - output[i]
            self.deltas[-1][i] = e * last.df[i]
            error += e * e / 2

        for k in range(self.p - 1, 0, -1):
            lk = self.layers[k]
            self.deltas[k - 1] = np.zeros(lk.m)
            for i in range(lk.m):
                for j in range(lk.n):
                    self.deltas[k - 1][i] += lk.w[j][i] * self.deltas[k][j]
                self.deltas[k - 1][i] *= self.layers[k - 1].df[i]

        return error

    def update(self):
        for k in range(self.p):
            lk = self.layers[k]
            for i in range(lk.n):
                for j in range(lk.m):
                    lk.w[i][j] -= self.alpha * self.deltas[k][i] * lk.x[j]

    def train(self):
        ep = 0
        error = 1
        result = ''
        while ep < self.max_epochs and error > self.eps:
            ep += 1
            errors = []
            for i in range(len(self.x)):
                a = self.forward(self.x[i])
                errors.append(self.backward(self.y[i]))
                self.update()
                # print(f'X: {self.x[i]}, Y: {self.y[i]}, out: {a}')
                error = np.mean(np.array(errors))
            result += f'{ep}: {error}\n'

```

```
94          for i in range(len(self.x)):
95              print(f'X: {self.x[i]}, Y: {self.y[i]}, out: {self.forward(self.x[i])}')
96
97          return result
98
99
100 def sigmoid(x):
101     return 1 / (1 + exp(-x))
102
103
104 def parse_param_value(param_value_str):
105     parts = param_value_str.split('=')
106     if len(parts) != 2:
107         raise argparse.ArgumentTypeError('Неправильный формат.')
108     return parts[0], parts[1]
109
110
111 def create_parser():
112     parser = argparse.ArgumentParser()
113     parser.add_argument('params', nargs='+', type=parse_param_value)
114     args = parser.parse_args()
115     params = dict(args.params)
116     input1 = params.get('input1')
117     input2 = params.get('input2')
118     input3 = params.get('input3')
119     output1 = params.get('output1')
120     return input1, input2, input3, output1
121
122
123 def write_json(filename, data):
124     with open(filename, 'w') as f:
125         json.dump(data, f)
126     print(f'В {filename} было записано {data}')
127
128
129 def read_json(filename):
130     with open(filename, 'r') as f:
131         data = json.load(f)
132     return data
133
134
135 # python3 nntask5.py input1='W_xor.json' input2='XY_xor.json' input3='params_xor.json'
    ↪ output1='E_xor.txt'
136 def main():
137     input1, input2, input3, output1 = create_parser()
138     network = Network(input1, input2, input3)
139
140     result = network.train()
```

```
141     with open(output1, 'w', encoding="utf-8") as out:
142         out.write(result)
143     print('История обучения записана в', output1)
144 
145 
146 main()
```

ПРИЛОЖЕНИЕ Е

Тестовые файлы для задания 5

```
{
  "w1": [[0.45, -0.12], [0.78, 0.13]],
  "w2": [[1.5, -2.3]]
}
```

Listing 1: Матрицы весов для обучения xor

```
{
  "x": [[0, 0], [0, 1], [1, 0], [1, 1]],
  "y": [[0.0], [1.0], [1.0], [0.0]]
}
```

Listing 2: Выборка для обучения xor

```
{
  "iters": 10000,
  "alpha": 0.5,
  "eps": 0.000001
}
```

Listing 3: Параметры для обучения xor

```
1: 0.13899320498724652
5000: 0.037662754759293815
10000: 0.008381458044956066
```

Listing 4: Выдержка из истории обучения xor

```
{
"w1": [
        [0.696, 0.282, 0.059, 0.024, 0.275, 0.236, 0.973, 0.378,
        0.016, 0.717, 0.238, 0.202, 0.309, 0.541, 0.456],
        [0.285, 0.433, 0.259, 0.807, 0.294, 0.794, 0.268, 0.205,
        0.247, 0.772, 0.300, 0.575, 0.844, 0.578, 0.938],
        [0.454, 0.824, 0.0794, 0.473, 0.946, 0.378, 0.541, 0.167,
        0.089, 0.019, 0.881, 0.630, 0.083, 0.967, 0.892],
        [0.258, 0.362, 0.31, 0.557, 0.499, 0.967, 0.443, 0.951,
        0.248, 0.454, 0.894, 0.996, 0.418, 0.997, 0.619],
        [0.399, 0.942, 0.245, 0.769, 0.313, 0.611, 0.34, 0.756,
        0.03, 0.065, 0.655, 0.832, 0.399, 0.684, 0.815]
      ],
"w2": [
        [0.231, 0.533, 0.541, 0.409, 0.232],
        [0.479, 0.641, 0.731, 0.159, 0.403],
        [0.525, 0.547, 0.93, 0.698, 0.849]
      ],
"w3": [
        [0.507, 0.838, 0.672]
      ]
}
```

Listing 5: Матрицы весов для обучения угадыванию числа

```
  {
    "x": [
          [1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1],
          [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1],
          [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
          [1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1],
          [1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1],
          [1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1],
          [1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1],
          [1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1],
          [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1],
          [1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1]
         ],
    "y": [[0.0], [0.1], [0.2], [0.3], [0.4], [0.5], [0.6], [0.7], [0.8], [0.9]]
  }
```

Listing 6: Выборка для обучения угадыванию числа

```
{
  "iters": 25000,
  "alpha": 0.75,
  "eps": 0.000001
}
```

Listing 7: Параметры для обучения угадыванию числа

```
1: 0.1137608520762083
5000: 0.0005942319497309006
10000: 0.00030233336286510624
25000: 2.9506600016289843e-05
50000: 9.024999353650563e-06
```

Listing 8: Выдержка из истории обучения угадыванию числа