

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

студента 5 курса 531 группы

факультета компьютерных наук и информационных технологий

Дусалиева Тахира Ахатовича

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель
Слеповичев

И.И.

подпись, дата

Саратов 2015

СОДЕРЖАНИЕ

Задание 1. Создание ориентированного графа	3
Задание 2. Создание функции по графу.....	8
Задание 3. Вычисление значения функции на графе	10
Задание 4. Построение многослойной нейронной сети	13
Задание 5. Реализация метода обратного распространения ошибки для многослойной НС.....	16
ПРИЛОЖЕНИЕ А	20
ПРИЛОЖЕНИЕ Б.....	26
ПРИЛОЖЕНИЕ В	33
ПРИЛОЖЕНИЕ Г	41
ПРИЛОЖЕНИЕ Д	45

Задание 1. Создание ориентированного графа

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

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

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

- а) 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.

Описание работы программы

Шаг 1. Открывается файл с текстовым описанием графа, после чего выбираются все данные (дуги), что соответствуют следующему формату (v_i, v_j, n) при этом игнорируются все пробелы, все текстовые символы между дугами. Достигается это с помощью регулярных выражений.

Шаг 2. Полученные данные форматируются, создаются экземпляры класса Vertex для v_i и v_j , где описывается количество дуг, которые в них заходят и исходят. Создаётся экземпляр класса Arc для вершин v_i и v_j и n , при этом увеличивая соответствующие счётчики исходящих и заходящих дуг в вершинах. Если n не соответствует количеству заходящих дуг в вершине v_j , то соответствующая запись заносится в файл с ошибками, а последующая сериализация графа прерывается.

Шаг 3. Создаётся экземпляр класса Graph с сортированным списком вершин Vertex и списком дуг Arc, с тем же порядком, что указан в файле. По этому графу проводится сериализация в xml файл.

Примеры исполнения

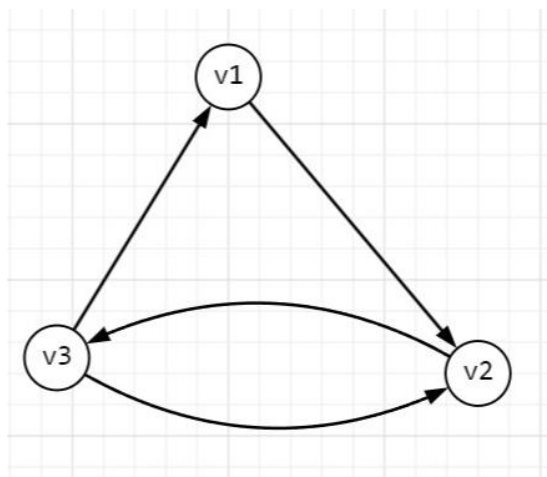


Рисунок 1 – Ориентированный граф 1

Вид этого графа в сериализованном виде:

```
<?xml version="1.0" encoding="utf-8"?>
<graph>
  <vertex>v1</vertex>
  <vertex>v2</vertex>
  <vertex>v3</vertex>
  <arc>
    <from>v1</from>
    <to>v2</to>
    <order>2</order>
```

```

</arc>
<arc>
  <from>v3</from>
  <to>v2</to>
  <order>1</order>
</arc>
<arc>
  <from>v2</from>
  <to>v3</to>
  <order>1</order>
</arc>
<arc>
  <from>v3</from>
  <to>v1</to>
  <order>1</order>
</arc>
</graph>

```

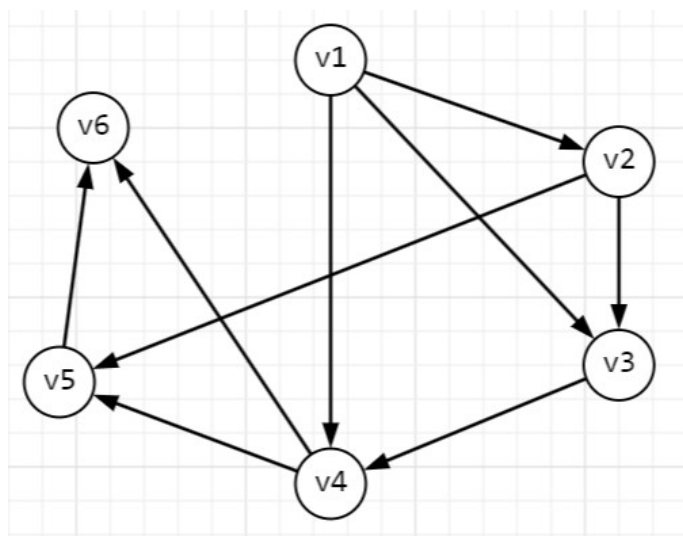


Рисунок 2 – Ориентированный граф 2
Вид этого графа в сериализованном виде:

```

<?xml version="1.0" encoding="utf-8"?>
<graph>
  <vertex>v1</vertex>
  <vertex>v2</vertex>
  <vertex>v3</vertex>
  <vertex>v4</vertex>
  <vertex>v5</vertex>
  <vertex>v6</vertex>
  <arc>
    <from>v1</from>
    <to>v2</to>
    <order>1</order>
  </arc>
  <arc>
    <from>v2</from>
    <to>v3</to>
    <order>1</order>

```

```

</arc>
<arc>
  <from>v1</from>
  <to>v3</to>
  <order>2</order>
</arc>
<arc>
  <from>v3</from>
  <to>v4</to>
  <order>1</order>
</arc>
<arc>
  <from>v1</from>
  <to>v4</to>
  <order>2</order>
</arc>
<arc>
  <from>v4</from>
  <to>v5</to>
  <order>1</order>
</arc>
<arc>
  <from>v4</from>
  <to>v6</to>
  <order>1</order>
</arc>
<arc>
  <from>v5</from>
  <to>v6</to>
  <order>2</order>
</arc>
<arc>
  <from>v2</from>
  <to>v5</to>
  <order>2</order>
</arc>
</graph>

```

Вид графов в программе::

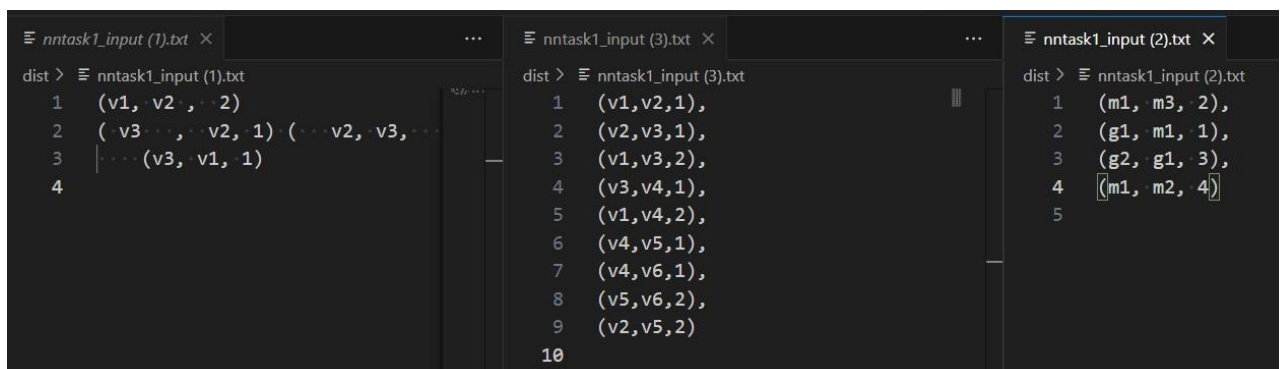


Рисунок 3 – Входные файлы задания 1

```

F:\Projects\Vcourse\Slepovichev\dist>nntask1.exe input="nntask1_input (1).txt" output="nntask1_output (1).xml"
Попытка завершилась успешно

F:\Projects\Vcourse\Slepovichev\dist>nntask1.exe input="nntask1_input (3).txt" output="nntask1_output (3).xml"
Попытка завершилась успешно

F:\Projects\Vcourse\Slepovichev\dist>nntask1.exe input="nntask1_input (2).txt" output="nntask1_output (2).xml"
Попытка завершилась с ошибками

```

Рисунок 4 – Исполнение программы задания 1

```

nntask1_output (1).xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <graph>
3   <vertex>v1</vertex>
4   <vertex>v2</vertex>
5   <vertex>v3</vertex>
6   <arc>
7     <from>v1</from>
8     <to>v2</to>
9     <order>2</order>
10  </arc>
11  <arc>
12    <from>v3</from>
13    <to>v2</to>
14    <order>1</order>
15  </arc>
16  <arc>
17    <from>v2</from>
18    <to>v3</to>
19    <order>1</order>
20  </arc>
21  <arc>
22    <from>v3</from>
23    <to>v1</to>
24    <order>1</order>
25  </arc>
26 </graph>
27 </xml>

nntask1_output (3).xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <graph>
3   <vertex>v1</vertex>
4   <vertex>v2</vertex>
5   <vertex>v3</vertex>
6   <vertex>v4</vertex>
7   <vertex>v5</vertex>
8   <vertex>v6</vertex>
9   <arc>
10    <from>v1</from>
11    <to>v2</to>
12    <order>1</order>
13  </arc>
14  <arc>
15    <from>v2</from>
16    <to>v3</to>
17    <order>1</order>
18  </arc>
19  <arc>
20    <from>v1</from>
21    <to>v3</to>
22    <order>2</order>
23  </arc>
24 </graph>
25 </xml>

errors.txt
1 2024-11-16 12:26:20.004451 : Shapes (3,3) and (2,2) not aligned. 3 (dim 1) != 2 (dim 0)
2
3 2024-11-16 12:28:49.740180 : Слой W1 не имеет связности со слоем W2
4
5 F:\Projects\Vcourse\Slepovichev\dist\nntask1_input (1).txt : F:\Projects\Vcourse\Slepovichev\dist\nntask1_output (
6
7 FileExistsError: Файл F:\Projects\Vcourse\Slepovichev\dist\nntask1_output (3).xml уже существует
8
9 F:\Projects\Vcourse\Slepovichev\dist\nntask1_input (2).txt : F:\Projects\Vcourse\Slepovichev\dist\nntask1_output (
10
11 OrderError: Порядок дуги (m1, m3, 2) указан неверно

```

Рисунок 5 – Результат работы программы задания 1

Исходный код программы указан в приложении А.

Задание 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:

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

Способ проверки результата:

- а) выгрузка в текстовый файл результата преобразования графа в имя функции.
- б) сообщение о наличии циклов в графе, если они присутствуют.

Описание работы программы

Шаг 1. Распаковывается файл с графом в формате xml и создается экземпляр объекта Graph, состоящий из подклассов Arc и Vertex описывающие дуги и вершины соответственно.

Шаг 2. Строится префиксная запись по следующим шагам:

Шаг 2.1. Graph проверяется на присутствие циклов с помощью метода поиска в глубину (DFS). Если циклы присутствуют, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 2.2. Идёт поиск корневой вершины, то есть той вершины, у которой нет заходящих в неё дуг. Если такой вершины нет, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 2.3. Строится префиксная запись.

Шаг 3. Префиксная запись записывается в файл.

Примеры исполнения

Для ориентированного графа 1 из рисунка 1 префиксной записи не существует, поскольку присутствуют цикл.

Для ориентированного графа 2 из рисунка 2 префиксная запись будет выглядеть так.

$$v1 \left(v2 \left(v3 \left(v4 \left(v5 \left(v6 \right), v6 \right) \right), v5 \left(v6 \right) \right), v3 \left(v4 \left(v5 \left(v6 \right), v6 \right) \right), v4 \left(v5 \left(v6 \right), v6 \right) \right)$$

Вид из программы:

```

dist > nntask1_output (1).xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <graph>
3 <vertex>v1</vertex>
4 <vertex>v2</vertex>
5 <vertex>v3</vertex>
6 <arc>
7 <from>v1</from>
8 <to>v2</to>
9 <order>2</order>
10 </arc>
11 <arc>
12 <from>v3</from>
13 <to>v2</to>
14 <order>1</order>
15 </arc>
16 <arc>
17 <from>v2</from>
18 <to>v3</to>
19 <order>1</order>
20 </arc>
21 <arc>
22 <from>v3</from>
23 <to>v1</to>
24 <order>1</order>
25 </arc>
26 </graph>
27

dist > nntask1_output (3).xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <graph>
3 <vertex>v1</vertex>
4 <vertex>v2</vertex>
5 <vertex>v3</vertex>
6 <vertex>v4</vertex>
7 <vertex>v5</vertex>
8 <vertex>v6</vertex>
9 <arc>
10 <from>v1</from>
11 <to>v2</to>
12 <order>1</order>
13 </arc>
14 <arc>
15 <from>v2</from>
16 <to>v3</to>
17 <order>1</order>
18 </arc>
19 <arc>
20 <from>v1</from>
21 <to>v3</to>
22 <order>2</order>
23 </arc>
24 <arc>
25 <from>v3</from>
26 <to>v4</to>
27 <order>1</order>
28 </arc>
29 <arc>
30 <from>v1</from>
31 <to>v4</to>
32 <order>2</order>

```

Рисунок 6 – Входные данные задания 2

```

F:\Projects\Vcourse\Slepovichev\dist>nntask2.exe input="nntask1_output (1).xml" output="nntask2_output (1).txt"
Попытка завершилась с ошибками

F:\Projects\Vcourse\Slepovichev\dist>nntask2.exe input="nntask1_output (3).xml" output="nntask2_output (3).txt"
Попытка завершилась успешно

```

Рисунок 7 – Исполнение программы задания 2

```

errors_txt
1 F:\Projects\Vcourse\Slepovichev\dist\nntask1_output (3).xml : F:\Projects\Vcourse\Slepovichev\dist\output.txt
2 FileExistsError: Файл F:\Projects\Vcourse\Slepovichev\dist\output.txt уже существует
3 F:\Projects\Vcourse\Slepovichev\dist\nntask1_output (1).xml : F:\Projects\Vcourse\Slepovichev\dist\nntask2_output
4 HasCyclesError: В графе присутствуют циклы
5

nntask2_output (3).txt
1 v1(v2(v3(v4(v5(v6), v6)), v5(v6)), v3(v4(v5(v6), v6)), v4(v5(v6), v6))

```

Рисунок 8 – Результат работы программы задания 2

Исходный код программы указан в приложении Б.

Задание 3. Вычисление значения функции на графе

На входе:

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

{
 a_1 : операция₁
 a_2 : операция₂
 ...
 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.

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

Способ проверки результата: результат вычисления, выведенный в файл.

Описание работы программы

Шаг 1. Распаковывается файл с графом в формате xml и создается экземпляр объекта Graph, состоящий из подклассов Arc и Vertex описывающие дуги и вершины соответственно.

Шаг 2. Распаковывается файл с операциями и вершинами, которым эти операции присвоены. В классе Graph соответствующим Vertex присваиваются операции.

Шаг 3. Строится префиксная запись по следующим шагам:

Шаг 3.1. Graph проверяется на присутствие циклов с помощью метода поиска в глубину (DFS). Если циклы присутствуют, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 3.2. Идёт поиск корневой вершины, то есть той вершины, у которой нет заходящих в неё дуг. Если такой вершины нет, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 3.3. Строится префиксная запись, где вместо вершин присутствуют их операции.

Шаг 4. Вычисляется значение префиксной записи с операциями и записывается в файл.

Пример исполнения

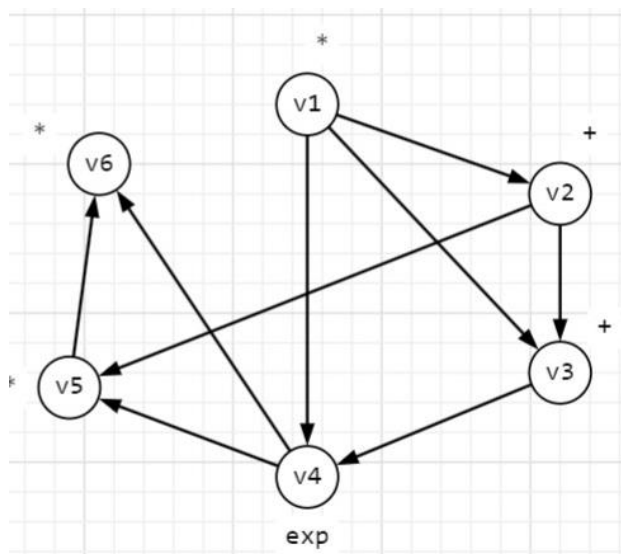
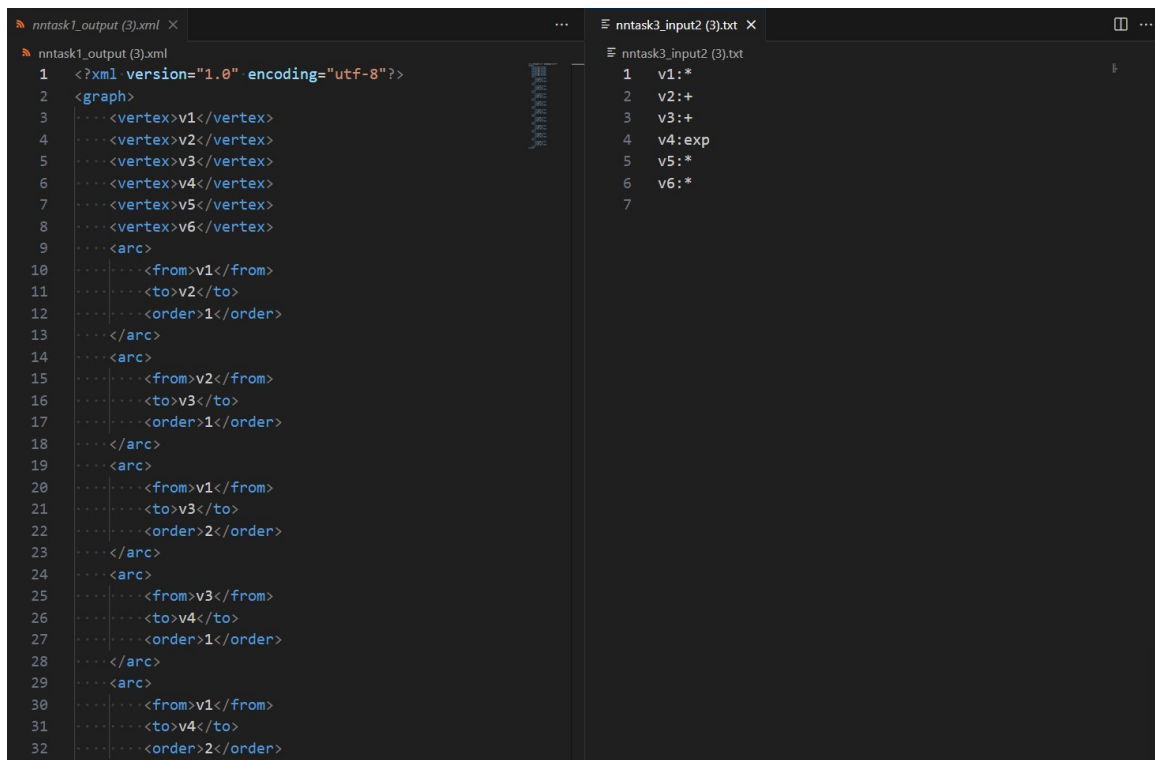


Рисунок 9 – Ориентированный граф 2 с определёнными на нём операциями

Для графа из рисунка 9 вычисление значения функции на этом графе будет выглядеть следующим образом:

$$\begin{aligned}
 &v1 \left(v2 \left(v3 \left(v4 \left(v5 \left(v6 \right), v6 \right) \right), v5 \left(v6 \right) \right), v3 \left(v4 \left(v5 \left(v6 \right), v6 \right) \right), v4 \left(v5 \left(v6 \right), v6 \right) \right) \right. \\
 &\quad \left. * \left(+ \left(+ \left(\exp \left(* \left(* \right), * \right) \right), * \left(* \right) \right), + \left(\exp \left(* \left(* \right), * \right) \right), \exp \left(* \left(* \right), * \right) \right) \right) = \\
 &\quad = 258.56367397895224
 \end{aligned}$$

Вид из программы:



```
<?xml version="1.0" encoding="utf-8"?>
<graph>
  <vertex>v1</vertex>
  <vertex>v2</vertex>
  <vertex>v3</vertex>
  <vertex>v4</vertex>
  <vertex>v5</vertex>
  <vertex>v6</vertex>
  <arc>
    <from>v1</from>
    <to>v2</to>
    <order>1</order>
  </arc>
  <arc>
    <from>v2</from>
    <to>v3</to>
    <order>1</order>
  </arc>
  <arc>
    <from>v1</from>
    <to>v3</to>
    <order>2</order>
  </arc>
  <arc>
    <from>v3</from>
    <to>v4</to>
    <order>1</order>
  </arc>
  <arc>
    <from>v1</from>
    <to>v4</to>
    <order>2</order>
  </arc>
</graph>

v1:*
v2:+
v3:+
v4:exp
v5:*
v6:*
```

Рисунок 10 – Входные данные из задания 3

```
F:\Projects\Vcourse\Slepovichev\dist>nntask3.exe input1="nntask1_output (3).xml" input2="nntask3_input2 (3).txt" output
"nntask3_output (3).txt"
Попытка завершилась успешно
```

Рисунок 11 – Исполнение программы из задания 3



```
nntask3_output (3).txt
1  *((+((exp(*(*), *)), *(*)), +(exp(*(*), *)), exp(*(*), *))) = 258.56367397895224
```

Рисунок 12 – Результат работы программы из задания 3

Исходный код программы указан в приложении В.

Задание 4. Построение многослойной нейронной сети

На входе:

а) Текстовый файл с набором матриц весов межнейронных связей:

$M1: [M1[1,1], M1[1,2], \dots, M1[1,n]], \dots, [M1[t, 1], M1[t, 2], \dots, M1[t, n]]$

$M2: [M2[1,1], M2[1,2], \dots, M2[1,n]], \dots, [M2[t, 1], M2[t, 2], \dots, M2[t, n]]$

...

$Mr: [Mr[1,1], Mr[1,2], \dots, Mr[1,n]], \dots, [Mr[t, 1], Mr[t, 2], \dots, Mr[t, n]]$

б) Текстовый файл с входным вектором в формате:

$x1, x2, \dots, xn.$

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:

$y1, y2, \dots, yn.$

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

Описание работы программы

Шаг 1. Открывается файл с несериализованной НС, после чего данные разделяются на слои. Создаётся экземпляр класса Network с именнованными слоями $W_i, i = \overline{1, p}$, если слои в соответствии с порядком в текстовом файле и имеют связности друг другом, то соответствующая запись записывается в файл с ошибками и дальнейший процесс вычисления выходного вектора прерывается. Сериализованная НС записывается в файл.

Шаг 2. Открывается файл с входным вектором $\overline{X_1}$.

Шаг 3. Последовательно вычисляются выходные векторы $\overline{Y_i} = W_i * \overline{X_i}$. Входной вектор $\overline{X_{i+1}} = \overline{Y_i^*}$, где $\overline{Y_i^*}$ это выходной вектор $\overline{Y_i}$ к которому была применена процедура $\text{sigmoid}(y) = \frac{1}{1+e^{-y}}$.

Шаг 4. Последний выходной вектор $\overline{Y_p^*}$ записывается в файл.

Пример исполнения

Возьмём НС вида:

$$W_1 = \begin{bmatrix} [0,2; 0,3; 0,9; 0,4] \\ [0,7; 0,3; 0,1; 0,5] \\ [0,6; 0,1; 0,2; 0,9] \\ [0,3; 0,3; 0,3; 0,3] \\ [0,0; 0,5; 1,0; 0,6] \end{bmatrix};$$
$$W_2 = \begin{bmatrix} [0,6; 0,7; 0,8; 0,9; 1,0] \\ [0,301; 0,302; 0,303; 0,304; 0,305] \\ [0,1; 0,2; 0,3; 0,2; 0,1] \end{bmatrix},$$

и входной вектор:

$$\overline{X}_1 = [5,81; 3,71; 6,21; 9,2].$$

Тогда вычисление выходного вектора будет выглядеть следующим образом:

$$\overline{Y}_1 = W_1 * \overline{X}_1 = [11,544; 10,401; 13,379; 7,479; 13,585];$$

$$\overline{Y}_1^* = \overline{X}_2 = [0,99999031; 0,9999696; 0,99999845; 0,9994355; 0,99999874];$$

$$\overline{Y}_2 = W_2 * \overline{X}_2 = [3,99946235; 1,51481544; 0,89987946];$$

$$\overline{Y}_2^* = [0,98200429; 0,81977376; 0,71092473].$$

Выходной вектор равен:

$$\overline{Y}_2^* = [0,98200429; 0,81977376; 0,71092473].$$

Вид в программе:

```
nttask4_input_1.txt x
dist > nntask4_input_1.txt
1 [[0.2, 0.3, 0.9, 0.4], [0.7, 0.3, 0.1, 0.5], [0.6, 0.1, 0.2, 0.9], [0.3, 0.3, 0.3, 0.3], [0.0, 0.5, 1.0, 0.6]]
2 [[0.6, 0.7, 0.8, 0.9, 1.0], [0.301, 0.302, 0.303, 0.304, 0.305], [0.1, 0.2, 0.3, 0.2, 0.1]]
3

nttask4_input_2.txt x
dist > nntask4_input_2.txt
1 [5.81, 3.71, 6.21, 9.2]
2
```

Рисунок 13 – Входные данные из задания 4

```

F:\Projects\Vcourse\Slepovichev\dist>nntask4.exe input1=nntask4_input_1.txt input2=nntask4_input_2.txt output=nntask4_output.json
-----
W1:
[[0.2 0.3 0.9 0.4]
 [0.7 0.3 0.1 0.5]
 [0.6 0.1 0.2 0.9]
 [0.3 0.3 0.3 0.3]
 [0. 0.5 1. 0.6]]
*
X:[5.81 3.71 6.21 9.2 ]
||
Y:[11.544 10.401 13.379 7.479 13.585]
Применяется процедура сигмоида на Y
Y:[0.99999031 0.9999696 0.99999845 0.9994355 0.99999874]
-----
W2:
[[0.6 0.7 0.8 0.9 1. ]
 [0.301 0.302 0.303 0.304 0.305]
 [0.1 0.2 0.3 0.2 0.1 ]]
*
X:[0.99999031 0.9999696 0.99999845 0.9994355 0.99999874]
||
Y:[3.99946235 1.51481544 0.89987946]
Применяется процедура сигмоида на Y
Y:[0.98200429 0.81977376 0.71092473]
-----
{'Y': [0.9820042912747702, 0.8197737585134958, 0.7109247309146316]}

```

Рисунок 14 – Исполнение программы из задания 4

The screenshot shows a code editor with three tabs: nntask4_input_1.json, nntask4_input_2.json, and nntask4_output.json. The first tab contains a JSON object with weights W1 and W2. The second tab contains a JSON object with input X. The third tab contains a JSON object with the output Y.

```

nntask4_input_1.json
dist > {} nntask4_input_1.json > ...
1 {
2   ... "W1": [[0.2, 0.3, 0.9, 0.4], [0.7, 0.3, 0.1, 0.5], [0.6, 0.1, 0.2, 0.9], [0.3, 0.3, 0.3, 0.3], [0.0, 0.5, 1.0,
3   ... "W2": [[0.6, 0.7, 0.8, 0.9, 1.0], [0.301, 0.302, 0.303, 0.304, 0.305], [0.1, 0.2, 0.3, 0.2, 0.1]]
4 }
5

nntask4_input_2.json
dist > {} nntask4_input_2.json > ...
1 {
2   ... "X": [5.81, 3.71, 6.21, 9.2]
3 }
4

nntask4_output.json
dist > {} nntask4_output.json > ...
1 [{"Y": [0.9820042912747702, 0.8197737585134958, 0.7109247309146316]}]

```

Рисунок 15 – Результат работы программы из задания 4
Исходный код программы указан в приложении Г.

Задание 5. Реализация метода обратного распространения ошибки для многослойной НС

На входе:

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

б) Текстовый файл с обучающей выборкой:

$$\begin{aligned} [x_{1,1}, x_{1,2}, \dots, x_{1,n}] &\rightarrow [y_{11}, y_{12}, \dots, y_{1m}] \\ &\dots \\ [x_{k,1}, x_{k,2}, \dots, x_{k,n}] &\rightarrow [y_{k,1}, y_{k,2}, \dots, y_{k,m}]. \end{aligned}$$

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

в) Текстовый файл с параметрами обучения (количество итераций, скорость обучения, погрешность)

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

1 : Ошибка1
2 : Ошибка2
...
N : ОшибкаN

Описание работы программы

Шаг 1. Распаковываются файлы с сериализованной НС, обучающей выборкой и параметрами.

Шаг 2. На вход НС подаётся обучающий пример (один из входных векторов обучающей выборки). НС вычисляет для каждого слоя выходной вектор \overline{Y}_i и производную $\partial \overline{Y}_i = \overline{Y}_i^* * (\overline{1} - \overline{Y}_i^*)$, а также дельты (разницы) для каждого слоя по формуле $\overline{\delta}_{i-1} = W_i^T * \overline{\delta}_i * \partial \overline{Y}_i$, где W_i^T – транспонированная матрица W_i , причём для последнего k -го слоя формула другая $\overline{\delta}_k = (\overline{Y}_k^* - \overline{Y}_k') * \partial \overline{Y}_k$, где \overline{Y}_k' – ожидаемый вектор.

Шаг 3. Сравнивается полученный выход с ожидаемым (из обучающей выборки) и вычисляется ошибка по формуле $E = \frac{1}{2} \sum (\bar{Y}_i^* - \bar{Y}_i')^2$, а также

Шаг 4. Распространить ошибку вверх по сети последовательно умножая ошибку на дельты слоев.

Шаг 5. Изменить веса в слоях, для уменьшения ошибок по следующей формуле $W_i = W_i + \alpha * \bar{\delta}_i * \bar{X}_i$, где α – коэффициент характеризующий скорость обучения.

Шаг 6. Положить $iter = iter - 1$, где $iter$ – итерация тренировки. Если $iter = 0$ закончить тренировку и записать список всех ошибок в файл, иначе перейти на шаг 1.

Пример исполнения

Допустим необходимо, чтобы НС могла определять число, загоревшееся на экране из таблицы 1.

Таблица 1 – Экран с пронумерованными пикселями

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28

Например, для числа 8 экран должен гореть как показано в таблице 2.


```
nttask5_output.txt X
dist > nntask5_output.txt
24975 24975: 0.00011751701970557878
24976 24976: 6.46972025895414e-05
24977 24977: 3.0849875211903624e-05
24978 24978: 0.0001118595663130473
24979 24979: 6.743762266144821e-05
24980 24980: 2.6471307944160095e-05
24981 24981: 0.00010238191875868187
24982 24982: 6.797636348748923e-05
24983 24983: 2.217651322802399e-05
24984 24984: 9.048666058845254e-05
24985 24985: 6.684960625526277e-05
24986 24986: 1.837315123951176e-05
24987 24987: 7.748654703117362e-05
24988 24988: 6.464548650207256e-05
24989 24989: 1.5408454249011912e-05
24990 24990: 6.441199935542634e-05
24991 24991: 6.187459112613533e-05
24992 24992: 1.3560259297290158e-05
24993 24993: 5.1983300517211925e-05
24994 24994: 5.891826244421216e-05
24995 24995: 1.3060181030640692e-05
24996 24996: 4.067980592538754e-05
24997 24997: 5.60322005806906e-05
24998 24998: 1.4126590735086953e-05
24999 24999: 3.08394242244784e-05
25000 25000: 5.3374824708575825e-05
25001
```

Рисунок 18 – Результат работы программы из задания 5
Результат окончательный оказался весьма близким к истинным.

```
X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.], out: [0.00987635]
X: [0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1.], Y: [0.1], out: [0.10086522]
X: [0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1.], Y: [0.2], out: [0.20167746]
X: [0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0.], Y: [0.3], out: [0.30199688]
X: [0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 0. 0. 0. 1. 0.], Y: [0.4], out: [0.40344592]
X: [1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0.], Y: [0.5], out: [0.48448955]
X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1.], Y: [0.6], out: [0.62387609]
X: [1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0.], Y: [0.7], out: [0.68975164]
X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1.], Y: [0.8], out: [0.79577815]
X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1.], Y: [0.9], out: [0.89587261]
it: 25000, error: 5.3374824708575825e-05
```

Исходный код программы указан в приложении Д.

ПРИЛОЖЕНИЕ А

Исходный код программы nntask1.exe.

Листинг 1 – Скрипт main.py

```
import sys
import argparse
from serialization import *
import os

path_to_dir = os.path.dirname(sys.executable)
# path_to_dir, _ = os.path.split(os.path.abspath(__file__))

def format_file_path(file):
    head, tail = os.path.split(file)
    if head:
        return file
    else:
        return os.path.join(path_to_dir, tail)

def get_graph(input_file, output_file):
    formatted_input_file = format_file_path(input_file)
    formatted_output_file = format_file_path(output_file)
    formatted_errors_file = format_file_path(f'errors.txt')
    try:
        serialization(input_file=formatted_input_file,
output_file=formatted_output_file)
        return 1
    except Exception as e:
        with open(formatted_errors_file, 'a') as f:
            f.write(f'{formatted_input_file}      :
{formatted_output_file}\n{e}\n')
        return -1

def parse_value(value : str):
    parts = value.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неверный формат введенных
данных')
    return parts[0], parts[1]

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_value)
    args = parser.parse_args()
    params = dict(args.params)
    input = params.get('input')
    output = params.get('output')

    error_file = format_file_path('errors.txt')
    r = get_graph(input, output)
    dct = {-1 : 'Попытка завершилась с ошибками',
```

```

        1 : 'Попытка завершилась успешно'
    }
    print(dct[r])

```

Листинг 2 – Скрипт serialization.py

```

import xml.etree.ElementTree as ET
import xml.dom.minidom as minidom
import re
import os.path
from utils import *

__all__ = ['serialization']

def get_data(file):
    try:
        with open(file, 'r') as f:
            all_data = f.read()
    except:
        raise FileNotFoundError(f'FileNotFoundError: Файл {file} не найден')

    match = re.findall(r"\\(\\s*?\\w+\\d+\\s*?,\\s*?\\w+\\d+\\s*?,\\s*?\\d+\\s*?)\\",
re.sub(r'\\s*', ' ', all_data))
    if not match:
        raise ValueError(f'ValueError: В файле {file} не указаны или неверно заданы данные')
    graph = Graph()
    sorted_match = sorted(match, key = lambda x:
int(x.split(',')[2].strip(' ')))
    arcs = []
    for raw_data in sorted_match:
        str_vertex1, str_vertex2 = re.findall(r"\\b\\w+\\d+", raw_data)
        vertex1 = graph.get_vertex(str_vertex1)
        if vertex1 is None:
            vertex1 = Vertex(str_vertex1)
            graph.add_vertex(vertex1)
        vertex2 = graph.get_vertex(str_vertex2)
        if vertex2 is None:
            vertex2 = Vertex(str_vertex2)
            graph.add_vertex(vertex2)
        arcs.append(Arc(vertex1, vertex2, int(re.search(r'\\b\\d+',
raw_data)[0])))
    for raw_data in match:
        str_vertex1, str_vertex2 = re.findall(r"\\b\\w+\\d+", raw_data)
        vertex1 = graph.get_vertex(str_vertex1)
        vertex2 = graph.get_vertex(str_vertex2)
        for arc in arcs:
            if arc.arc_from == vertex1 and arc.arc_to == vertex2 and
arc.order == int(re.search(r'\\b\\d+', raw_data)[0]):
                graph.add_arc(arc)
                break

```

```

    return graph

def pretty_print(xml_elem):
    rough_string = ET.tostring(xml_elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent='    ', encoding='utf-8')

class FileExtensionError(Exception):
    pass

def serialization(input_file, output_file):
    if os.path.exists(output_file):
        raise FileExistsError(f'FileExistsError: Файл {output_file}
уже существует')
    _, extension = os.path.splitext(output_file)
    if extension != '.xml':
        raise FileExtensionError(f'FileExtensionError: Выходной файл
должен быть формата XML-документа')

    graph = get_data(input_file)
    # vertexes, arcs = get_data(input_file)
    data_xml = ET.Element('graph')
    for v in sorted(graph.vertexes, key=lambda x: x.name):
        vertex_xml = ET.SubElement(data_xml, 'vertex')
        vertex_xml.text = v.name
    for arc in graph.arcs:
        arc_xml = ET.SubElement(data_xml, 'arc')
        arc_xml_from = ET.SubElement(arc_xml, 'from')
        arc_xml_from.text = arc.arc_from.name
        arc_xml_to = ET.SubElement(arc_xml, 'to')
        arc_xml_to.text = arc.arc_to.name
        arc_xml_order = ET.SubElement(arc_xml, 'order')
        arc_xml_order.text = str(arc.order)

    str_xml = pretty_print(data_xml)
    with open(output_file, 'wb') as f:
        f.write(str_xml)

```

Листинг 3 – Скрипт utils.py

```

class Vertex:
    def __init__(self, name : str):
        self.name = name
        self.arc_to = 0
        self.arc_from = 0

    def __str__(self):
        return f'Vertex({self.name})'

    def __repr__(self):
        return f'Vertex({self.name})'

class OrderError(Exception):

```

```

pass

class Arc:
    def __init__(self, vertex1 : Vertex, vertex2 : Vertex, order :
int):
        self.arc_from = vertex1
        self.arc_to = vertex2
        self.order = order
        self.arc_from.arc_from += 1
        self.arc_to.arc_to += 1
        if self.order != self.arc_to.arc_to:
            raise OrderError(f'OrderError: Порядок дуги
({self.arc_from.name}, {self.arc_to.name}, {self.order}) указан
неверно')

    def __str__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def __repr__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def _get_vertexes(self):
        return self.arc_from, self.arc_to

class QuadMatrixError(Exception):
    pass

class QuadMatrix:
    def __init__(self, args : list):
        if len(args) != len(args[0]):
            raise QuadMatrixError('QuadMatrixError: На вход попал
список из которого невозможно создать квадратную матрицу')
        self.args = args

    def __len__(self):
        return len(self.args)

    def __getitem__(self, index : int):
        return self.args[index]

    def __setitem__(self, index : int, value):
        self.args[index] = value

    def __str__(self):
        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y),
x)), self.args))

    def __repr__(self):
        return f'Matrix({self.args})'

    def transpose(self):
        self.args = [list(row) for row in zip(*self.args)]

```

```

        return self

class GraphHasNotVertex(Exception):
    pass

class Graph:
    def __init__(self, vertexes : list = None, arcs : list = None):
        if vertexes is not None:
            self.vertexes = set(self.from_list(vertexes, Vertex))
        else:
            self.vertexes = set()
        if arcs is not None:
            self.arcs = self.from_list(arcs, Arc)
        else:
            self.arcs = list()

    def from_list(self, li : list, obj):
        if all(map(lambda x: isinstance(x, obj), li)):
            return li

    def has_vertex(self, other : str):
        if other in map(lambda x: x.name, self.vertexes):
            return True
        return False

    def get_vertex(self, other : str):
        if self.has_vertex(other):
            for vertex in self.vertexes:
                if vertex.name == other:
                    return vertex

    def add_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.add(other)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def pop_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.discard(other)
            remove_arc_indexes = []
            for index, arc in enumerate(self.arcs):
                vertex1, vertex2 = arc._get_vertexes()
                if vertex1 == other or vertex2 == other:
                    remove_arc_indexes.append(index)
            map(lambda x: self.arcs.pop(x), remove_arc_indexes)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def add_arc(self, other):

```



```

        if isinstance(other, Arc):
            vertex1, vertex2 = other._get_vertexes()
            if vertex1 not in self.vertexes and vertex2 not in
self.vertexes:
                raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины
{vertex1} или {vertex2} нет в Graph')
            self.arcs.append(other)
        else:
            raise ValueError('ValueError: На вход попал не объект
Arc')

    def pop_arc(self, index : int = -1):
        self.arcs.pop(index)

    def __str__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
                'arcs' : self.arcs}
        return f'Graph({dct})'

    def __repr__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
                'arcs' : self.arcs}
        return f'Graph({dct})'

    def get_matrix_adjacency(self):
        n = len(self.vertexes)
        self.adj = QuadMatrix([[0 for _ in range(n)] for _ in
range(n)])
        dct = dict(map(lambda x: (x[1], x[0]),
enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:
            self.adj[dct[arc.arc_from]][dct[arc.arc_to]] += 1
        return self

```

ПРИЛОЖЕНИЕ Б

Исходный код программы nntask2.exe.

Листинг 1 – Скрипт main.py

```
import sys
import re
import os
from cycles import prefix_function
import argparse

path_to_dir = os.path.dirname(sys.executable)
# path_to_dir, _ = os.path.split(os.path.abspath(__file__))

def format_file_path(file):
    head, tail = os.path.split(file)
    if head:
        return file
    else:
        return os.path.join(path_to_dir, tail)

def parse_value(value : str):
    parts = value.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неверный формат введённых данных')
    return parts[0], parts[1]

def get_prefix_notation(input_file, output_file):
    formatted_input_file = format_file_path(input_file)
    formatted_output_file = format_file_path(output_file)
    formatted_errors_file = format_file_path('errors.txt')
    try:
        prefix_function(formatted_input_file, formatted_output_file)
        return 1
    except Exception as e:
        with open(formatted_errors_file, 'a') as f:
            f.write(f'{formatted_input_file} : {formatted_output_file}\n{e}\n')
        return -1

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_value)
    args = parser.parse_args()
    params = dict(args.params)
    input = params.get('input')
    output = params.get('output')

    error_file = format_file_path('errors.txt')
    r = get_prefix_notation(input, output)
    dct = {-1 : 'Попытка завершилась с ошибками',
```

```

        1 : 'Попытка завершилась успешно'
    }
    print(dct[r])

```

Листинг 2 – Скрипт cycles.py

```

import xml.etree.ElementTree as ET
from utils import Graph, Arc, Vertex
import os.path

class FileExtensionError(Exception):
    pass

def unpack(input_file):
    _, extension = os.path.splitext(input_file)
    if extension != '.xml':
        raise FileExtensionError(f'FileExtensionError: Входящий файл
должен быть формата XML-документа')
    try:
        with open(input_file, 'r') as f:
            xml_data = f.read()
    except:
        raise FileNotFoundError(f"FileNotFoundError: Файл
{input_file} не был найден")
    tree = ET.fromstring(xml_data)
    graph = Graph()
    for vertex in tree.findall('vertex'):
        graph.add_vertex(Vertex(vertex.text))

    for arc in sorted(tree.findall('arc'), key = lambda x:
int(x.findall('order')[0].text)):
        from_vertex = graph.get_vertex(arc.findall('from')[0].text)
        to_vertex = graph.get_vertex(arc.findall('to')[0].text)
        order = int(arc.findall('order')[0].text)
        graph.add_arc(Arc(from_vertex, to_vertex, order))
    return graph

class HasCyclesError(Exception):
    pass

class RootVertexNotFoundError(Exception):
    pass

def prefix_function(input_file, output_file):
    graph = unpack(input_file)
    has_cycles = graph.cycles()
    if has_cycles:
        raise HasCyclesError(f'HasCyclesError: В графе присутствуют
циклы')

    root_vertex = None
    for vertex in graph.vertexes:
        if vertex.arc_to == 0:

```

```

        root_vertex = vertex

    if root_vertex is None:
        raise RootVertexNotFoundError(f'RootVertexNotFoundError: В
графе отсутствуют вершины без входящих в них дуг')

    prefix_notation = build_prefix_notation(graph, root_vertex)
    if os.path.exists(output_file):
        raise FileExistsError(f"FileExistsError: Файл {output_file}
уже существует")
    with open(output_file, 'w') as f:
        f.write(strfprefix(prefix_notation))

def build_prefix_notation(graph : Graph, root_vertex : Vertex):
    result = {root_vertex : {}}
    for arc in graph.arcs:
        if arc.arc_from == root_vertex:
            result[root_vertex].update(build_prefix_notation(graph,
arc.arc_to))
    return result

def strfprefix(prefix_notation : dict):
    result = []
    for key, value in prefix_notation.items():
        result.append(key.name)
        if value == {}:
            continue
        result[-1] += f'({strfprefix(value)})'
    return ', '.join(result)

```

Листинг 3 – Скрипт utils.py

```

class Vertex:
    def __init__(self, name : str):
        self.name = name
        self.arc_to = 0
        self.arc_from = 0

    def __str__(self):
        return f'Vertex({self.name})'

    def __repr__(self):
        return f'Vertex({self.name})'

class OrderError(Exception):
    pass

class Arc:
    def __init__(self, vertex1 : Vertex, vertex2 : Vertex, order :
int):
        self.arc_from = vertex1
        self.arc_to = vertex2
        self.order = order

```

```

        self.arc_from.arc_from += 1
        self.arc_to.arc_to += 1
        if self.order != self.arc_to.arc_to:
            raise OrderError(f'OrderError: Порядок дуги
({self.arc_from.name}, {self.arc_to.name}, {self.order}) указан
неверно')

    def __str__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def __repr__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def _get_vertexes(self):
        return self.arc_from, self.arc_to

    def __getitem__(self, value : Vertex):
        if value == self.arc_from:
            return self.arc_to

class QuadMatrixError(Exception):
    pass

class QuadMatrix:
    def __init__(self, args : list):
        if len(args) != len(args[0]):
            raise QuadMatrixError('QuadMatrixError: На вход попал
список из которого невозможно создать квадратную матрицу')
        self.args = args

    def __len__(self):
        return len(self.args)

    def __getitem__(self, index : int):
        return self.args[index]

    def __setitem__(self, index : int, value):
        self.args[index] = value

    def __str__(self):
        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y),
x)), self.args))

    def __repr__(self):
        return f'Matrix({self.args})'

    def transpose(self):
        self.args = [list(row) for row in zip(*self.args)]
        return self

class GraphHasNotVertex(Exception):
    pass

```

```

class Graph:
    def __init__(self, vertexes : list = None, arcs : list = None):
        if vertexes is not None:
            self.vertexes = set(self.from_list(vertexes, Vertex))
        else:
            self.vertexes = set()
        if arcs is not None:
            self.arcs = self.from_list(arcs, Arc)
        else:
            self.arcs = list()

    def from_list(self, li : list, obj):
        if all(map(lambda x: isinstance(x, obj), li)):
            return li

    def has_vertex(self, other : str):
        if other in map(lambda x: x.name, self.vertexes):
            return True
        return False

    def get_vertex(self, other : str):
        if self.has_vertex(other):
            for vertex in self.vertexes:
                if vertex.name == other:
                    return vertex

    def add_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.add(other)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def pop_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.discard(other)
            remove_arc_indexes = []
            for index, arc in enumerate(self.arcs):
                vertex1, vertex2 = arc._get_vertexes()
                if vertex1 == other or vertex2 == other:
                    remove_arc_indexes.append(index)
            map(lambda x: self.arcs.pop(x), remove_arc_indexes)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def add_arc(self, other):
        if isinstance(other, Arc):
            vertex1, vertex2 = other._get_vertexes()
            if vertex1 not in self.vertexes and vertex2 not in
self.vertexes:

```

```

        raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины
{vertex1} или {vertex2} нет в Graph')
        self.arcs.append(other)
    else:
        raise ValueError('ValueError: На вход попал не объект
Arc')

    def pop_arc(self, index : int = -1):
        self.arcs.pop(index)

    def __str__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
               'arcs' : self.arcs}
        return f'Graph({dct})'

    def __repr__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
               'arcs' : self.arcs}
        return f'Graph({dct})'

    def get_matrix_adjacency(self):
        n = len(self.vertexes)
        self.adj = QuadMatrix([[0 for _ in range(n)] for _ in
range(n)])
        dct = dict(map(lambda x: (x[1], x[0]),
enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:
            self.adj[dct[arc.arc_from]][dct[arc.arc_to]] += 1
        return self

    def cycles(self):
        def dfs(vertex : Vertex, visited : set, stack : set):
            visited.add(vertex)
            stack.add(vertex)
            for arc in self.arcs:
                if arc.arc_from == vertex and arc.arc_to not in
visited:
                    if dfs(arc.arc_to, visited, stack):
                        return True
                elif arc.arc_from == vertex and arc.arc_to in stack:
                    return True
            stack.remove(vertex)
            return False

        visited = set()
        for vertex in self.vertexes:
            if vertex not in visited:
                if dfs(vertex, visited, set()):
                    return True

```

```
return False
```


ПРИЛОЖЕНИЕ В

Исходный код программы nntask3.exe.

Листинг 1 – Скрипт main.py

```
import sys
import os
from calculate import prefix_function
import argparse

path_to_dir = os.path.dirname(sys.executable)
# path_to_dir, _ = os.path.split(os.path.abspath(__file__))

def format_file_path(file):
    head, tail = os.path.split(file)
    if head:
        return file
    else:
        return os.path.join(path_to_dir, tail)

def get_prefix_notation(input_file, input_oper_file, output_file):
    formatted_input_file = format_file_path(input_file)
    formatted_input_oper_file = format_file_path(input_oper_file)
    formatted_output_file = format_file_path(output_file)
    formatted_errors_file = format_file_path('errors_.txt')
    try:
        prefix_function(formatted_input_file,
formatted_input_oper_file, formatted_output_file)
        return 1
    except Exception as e:
        with open(formatted_errors_file, 'a') as f:
            f.write(f'{formatted_input_file}      :
{formatted_output_file}\n{e}\n')
        return 0

def parse_value(value : str):
    parts = value.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неверный формат введенных
данных')
    return parts[0], parts[1]

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_value)
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    input2 = params.get('input2')
    output = params.get('output')

    error_file = format_file_path('errors.txt')
```

```

r = get_prefix_notation(input1, input2, output)
dct = {-1 : 'Попытка завершилась с ошибками',
       1 : 'Попытка завершилась успешно'
       }
print(dct[r])

```

Листинг 2 – Скрипт calculate.py

```

import xml.etree.ElementTree as ET
from utils import Graph, Arc, Vertex
import os.path
import re
from math import exp

class FileExtensionError(Exception):
    pass

_NUMBER_MATCH = re.compile(
    r"-?\b\d+\.\d*\|\.\d+\b"
)

_INT_MATCH = re.compile(
    r"-?\b\d+"
)

def unpack(input_file):
    _, extension = os.path.splitext(input_file)
    if extension != '.xml':
        raise FileExtensionError(f'FileExtensionError: Входящий файл
должен быть формата XML-документа')
    try:
        with open(input_file, 'r') as f:
            xml_data = f.read()
    except:
        raise FileNotFoundError(f"FileNotFoundError: Файл
{input_file} не был найден")
    tree = ET.fromstring(xml_data)
    graph = Graph()
    for vertex in tree.findall('vertex'):
        graph.add_vertex(Vertex(vertex.text))

    for arc in sorted(tree.findall('arc'), key = lambda x:
int(x.findall('order')[0].text)):
        from_vertex = graph.get_vertex(arc.findall('from')[0].text)
        to_vertex = graph.get_vertex(arc.findall('to')[0].text)
        order = int(arc.findall('order')[0].text)
        graph.add_arc(Arc(from_vertex, to_vertex, order))
    return graph

def summarize(*args):
    res = 0
    for ch in args:
        res += ch

```

```

    return res

def multiplie(*args):
    res = 1
    for ch in args:
        res *= ch
    return res

def my_exp(*args):
    res = []
    for ch in args:
        res.append(exp(ch))
    return res

def get_number(s):
    if _NUMBER_MATCH.match(s):
        if _INT_MATCH.match(s):
            return int(s)
        else:
            return float(s)

def get_operations(input_file, graph : Graph):
    try:
        with open(input_file, 'r') as f:
            operations_data = f.read()
    except:
        raise FileNotFoundError(f"FileNotFoundError: Файл
{input_file} не был найден")
    match = re.findall(r'(\b\w+\d+)\s*:\s*([+*]|exp|(-?\b\d+\.\d*|\.\d+\b))',
operations_data)
    if not match:
        raise ValueError(f'ValueError: В файле {input_file} не
указаны или неверно заданы данные')
    operations_dct = {
        '+' : summarize,
        '*' : multiplie,
        'exp' : my_exp,
        'NUMBER' : get_number
    }
    for raw_data in match:
        vertex = raw_data[0]
        v = graph.get_vertex(vertex)
        if v is None:
            raise ValueError(f'ValueError: Вершины {vertex} не
существует в графе')
        operation = raw_data[1]
        if operation in operations_dct.keys():
            v.init_operation((operation, operations_dct[operation]))
        else:
            v.init_value(operations_dct['NUMBER'](operation))
    return graph

```

```

class HasCyclesError(Exception):
    pass

class RootVertexNotFoundError(Exception):
    pass

def prefix_function(input_graph_file, input_oper_file,
output_file):
    graph = unpack(input_graph_file)
    has_cycles = graph.cycles()
    if has_cycles:
        raise HasCyclesError(f'HasCyclesError: В графе присутствуют
циклы')

    root_vertex = None
    for vertex in graph.vertexes:
        if vertex.arc_to == 0:
            root_vertex = vertex

    if root_vertex is None:
        raise RootVertexNotFoundError(f'RootVertexNotFoundError: В
графе отсутствуют вершины без входящих в них дуг')

    graph = get_operations(input_oper_file, graph)

    prefix_notation = build_prefix_notation(graph, root_vertex)
    str_oper_prefix_notation = strfprefix(prefix_notation)
    result = calculate_prefix(prefix_notation, root_vertex)
    if os.path.exists(output_file):
        raise FileExistsError(f"FileExistsError: Файл {output_file}
уже существует")
    with open(output_file, 'w') as f:
        f.write(str_oper_prefix_notation + ' = ' + str(result))

def calculate_prefix(prefix_notation, root_vertex : Vertex):
    result = None
    if root_vertex.value is not None:
        result = root_vertex.value
    else:
        tmp = []
        for key in prefix_notation[root_vertex].keys():
            x = calculate_prefix(prefix_notation[root_vertex], key)
            if isinstance(x, list):
                tmp.extend(x)
            else:
                tmp.append(x)
        result = root_vertex.operation[1](*tmp)
    return result

def build_prefix_notation(graph : Graph, root_vertex : Vertex):
    result = {root_vertex : {}}
```

```

        for arc in graph.arcs:
            if arc.arc_from == root_vertex:
                result[root_vertex].update(build_prefix_notation(graph,
arc.arc_to))
        return result

def strfprefix(prefix_notation : dict):
    result = []
    for key, value in prefix_notation.items():
        if key.operation is not None:
            result.append(key.operation[0])
        else:
            result.append(str(key.value))
        if value == {}:
            continue
        result[-1] += f'({strfprefix(value)})'
    return ', '.join(result)

```

Листинг 3 – Скрипт utils.py

```

class Vertex:
    def __init__(self, name : str):
        self.name = name
        self.arc_to = 0
        self.arc_from = 0
        self.operation = None
        self.value = None

    def __str__(self):
        return f'Vertex({self.name})'

    def __repr__(self):
        return f'Vertex({self.name})'

    def init_operation(self, operation):
        self.operation = operation

    def init_value(self, value):
        self.value = value

class OrderError(Exception):
    pass

class Arc:
    def __init__(self, vertex1 : Vertex, vertex2 : Vertex, order :
int):
        self.arc_from = vertex1
        self.arc_to = vertex2
        self.order = order
        self.arc_from.arc_from += 1
        self.arc_to.arc_to += 1
        if self.order != self.arc_to.arc_to:

```

```

        raise OrderError(f'OrderError: Порядок дуги
({self.arc_from.name}, {self.arc_to.name}, {self.order}) указан
неверно')

    def __str__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def __repr__(self):
        return f'Arc(from: {self.arc_from}, to: {self.arc_to})'

    def _get_vertexes(self):
        return self.arc_from, self.arc_to

    def __getitem__(self, value : Vertex):
        if value == self.arc_from:
            return self.arc_to

class QuadMatrixError(Exception):
    pass

class QuadMatrix:
    def __init__(self, args : list):
        if len(args) != len(args[0]):
            raise QuadMatrixError('QuadMatrixError: На вход попал
список из которого невозможно создать квадратную матрицу')
        self.args = args

    def __len__(self):
        return len(self.args)

    def __getitem__(self, index : int):
        return self.args[index]

    def __setitem__(self, index : int, value):
        self.args[index] = value

    def __str__(self):
        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y),
x))), self.args))

    def __repr__(self):
        return f'Matrix({self.args})'

    def transpose(self):
        self.args = [list(row) for row in zip(*self.args)]
        return self

class GraphHasNotVertex(Exception):
    pass

class Graph:
    def __init__(self, vertexes : list = None, arcs : list = None):

```

```

    if vertexes is not None:
        self.vertexes = set(self.from_list(vertexes, Vertex))
    else:
        self.vertexes = set()
    if arcs is not None:
        self.arcs = self.from_list(arcs, Arc)
    else:
        self.arcs = list()

    def from_list(self, li : list, obj):
        if all(map(lambda x: isinstance(x, obj), li)):
            return li

    def has_vertex(self, other : str):
        if other in map(lambda x: x.name, self.vertexes):
            return True
        return False

    def get_vertex(self, other : str):
        if self.has_vertex(other):
            for vertex in self.vertexes:
                if vertex.name == other:
                    return vertex

    def add_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.add(other)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def pop_vertex(self, other):
        if isinstance(other, Vertex):
            self.vertexes.discard(other)
            remove_arc_indexes = []
            for index, arc in enumerate(self.arcs):
                vertex1, vertex2 = arc._get_vertexes()
                if vertex1 == other or vertex2 == other:
                    remove_arc_indexes.append(index)
            map(lambda x: self.arcs.pop(x), remove_arc_indexes)
        else:
            raise ValueError('ValueError: На вход попал объект
отличный от Vertex')

    def add_arc(self, other):
        if isinstance(other, Arc):
            vertex1, vertex2 = other._get_vertexes()
            if vertex1 not in self.vertexes and vertex2 not in
self.vertexes:
                raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины
{vertex1} или {vertex2} нет в Graph')
            self.arcs.append(other)

```

```

        else:
            raise ValueError('ValueError: На вход попал не объект
Arc')

    def pop_arc(self, index : int = -1):
        self.arcs.pop(index)

    def __str__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
                'arcs' : self.arcs}
        return f'Graph({dct})'

    def __repr__(self):
        dct = {'vertexes' : sorted(self.vertexes, key=lambda x:
x.name),
                'arcs' : self.arcs}
        return f'Graph({dct})'

    def get_matrix_adjacency(self):
        n = len(self.vertexes)
        self.adj = QuadMatrix([[0 for _ in range(n)] for _ in
range(n)])
        dct = dict(map(lambda x: (x[1], x[0]),
enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:
            self.adj[dct[arc.arc_from]][dct[arc.arc_to]] += 1
        return self

    def cycles(self):
        def dfs(vertex : Vertex, visited : set, stack : set):
            visited.add(vertex)
            stack.add(vertex)
            for arc in self.arcs:
                if arc.arc_from == vertex and arc.arc_to not in
visited:
                    if dfs(arc.arc_to, visited, stack):
                        return True
                elif arc.arc_from == vertex and arc.arc_to in stack:
                    return True
            stack.remove(vertex)
            return False

        visited = set()
        for vertex in self.vertexes:
            if vertex not in visited:
                if dfs(vertex, visited, set()):
                    return True
        return False

```


ПРИЛОЖЕНИЕ Г

Исходный код программы nntask4.exe.

Листинг 1 – Скрипт main.py

```
import sys
import os.path
from calculateN import *
from serializationN import *
import argparse
from datetime import datetime

path_to_dir = os.path.dirname(sys.executable)
# path_to_dir, _ = os.path.split(os.path.abspath(__file__))

def format_file_path(file):
    head, tail = os.path.split(file)
    if head:
        return file
    else:
        return os.path.join(path_to_dir, tail)

def parse_value(value : str):
    parts = value.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неверный формат введенных данных')
    return parts[0], parts[1]

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_value)
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    input2 = params.get('input2')
    output = params.get('output')

    error_file = format_file_path('errors.txt')
    try:
        Name_network = serialization_network(format_file_path(input1))
        Name_entry = serialization_entry(format_file_path(input2))
        Network(Name_network, Name_entry, format_file_path(output))
    except Exception as e:
        print(e)
        with open(error_file, 'a') as f:
            f.write(f'{datetime.now()} : {e}\n')
```

Листинг 2 – Скрипт serializationN.py

```
import json
```

```

import re
import os.path

def serialization_network(input_file):
    try:
        with open(input_file, 'r') as f:
            data = f.read()
    except:
        raise FileNotFoundError(f'Файл {input_file} не найден')

    match = re.findall(r'\[(\.[*\])\]', data)
    if match[0] is None:
        raise ValueError(f'Файл {input_file} не содержит данных')

    k = 1
    Network = dict()
    for raw_data in match:
        W = re.findall(r'\[[^\]]*\]', raw_data)
        new_W = list()
        for layer in W:
            numbers = re.findall(r'(-?\b\d+\.?\d*|\.\d+\b)', layer)
            new_W.append(list(map(lambda x: float(x), numbers)))
        k_len = len(str(k))
        layer_num = str(k)
        if k_len < len(match) // 10:
            layer_num = '0' * (len(match) // 10 - k_len) + str(k)
        Network[f'W{layer_num}'] = new_W
        k += 1

    filename, _ = os.path.splitext(input_file)
    output_file = filename + '.json'
    i = 1
    while os.path.exists(output_file):
        output_file = filename + f' ({i})' + '.json'
        i += 1

    with open(output_file, 'w') as f:
        json.dump(Network, f)

    return output_file

def serialization_entry(input_file):
    try:
        with open(input_file, 'r') as f:
            data = f.read()
    except:
        raise FileNotFoundError(f'Файл {input_file} не найден')

    match = re.findall(r'(-?\b\d+\.?\d*|\.\d+\b)', data)
    if match[0] is None:
        raise ValueError(f'Файл {input_file} не содержит данных')

```

```

Entry = {'X' : list(map(lambda x: float(x), match))}

filename, _ = os.path.splitext(input_file)
output_file = filename + '.json'
i = 1
while os.path.exists(output_file):
    output_file = filename + f' ({i})' + '.json'
    i += 1

with open(output_file, 'w') as f:
    json.dump(Entry, f)

return output_file

```

Листинг 3 – Скрипт calculateN.py

```

import numpy as np
import json
from math import exp

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

class Layer:
    def __init__(self, name, W):
        self.name = name
        self.W = np.array(W)
        shape = self.W.shape
        self.n = shape[0]
        self.m = shape[1]
        self.X = np.zeros(self.m)
        self.Y = np.zeros(self.n)

    def set_x(self, X):
        self.X = np.array(X)
        assert(self.X.shape[0] != self.n)

    def calculate(self):
        print(f'{self.name}:\n{self.W}')
        print(('t' * (self.m // 2)) + '*')
        print('X:' + str(self.X))
        Y = np.dot(self.W, self.X)
        print(('t' * (self.m // 2)) + '||')
        print('Y:' + str(Y))
        print('Применяется процедура сигмоида на Y')
        for i in range(self.n):
            Y[i] = sigmoid(Y[i])
        print('Y:' + str(Y))
        self.Y = Y
        return self.Y

class LayerShapeError(Exception):
    pass

```

```

class Network:
    def __init__(self, input_layers, input_entry, output):
        weights = load(input_layers)
        self.layers = []
        sorted_layers = sorted(weights.keys())
        for w in sorted_layers:
            self.layers.append(Layer(w, weights[w]))
        for w1, w2 in zip(self.layers[:-1], self.layers[1:]):
            if w1.n != w2.m:
                raise LayerShapeError(f'Слой {w1.name} не имеет
связности со слоем {w2.name}')
        X = load(input_entry)
        self.layers[0].X = np.array(X['X'])
        dump(output, {'Y' : list(self.calculate())})

    def __len__(self):
        return len(self.layers)

    def calculate(self):
        k = len(self)
        for i in range(k):
            print('-----')
            l = self.layers[i]
            new_x = l.calculate()
            if i < k - 1:
                self.layers[i + 1].X = new_x
            print('-----')
        return self.layers[-1].Y

    def load(file):
        with open(file, 'r') as f:
            return json.load(f)

    def dump(file, data):
        with open(file, 'w') as f:
            json.dump(data, f)
        print(data)

```

ПРИЛОЖЕНИЕ Д

Исходный код программы nntask5.exe.

Листинг 1 – Скрипт main.py

```
import sys
import os.path
from calculateN import *
import argparse
from datetime import datetime

path_to_dir = os.path.dirname(sys.executable)
# path_to_dir, _ = os.path.split(os.path.abspath(__file__))

def format_file_path(file):
    head, tail = os.path.split(file)
    if head:
        return file
    else:
        return os.path.join(path_to_dir, tail)

def parse_value(value : str):
    parts = value.split('=')
    if len(parts) != 2:
        raise argparse.ArgumentTypeError('Неверный формат введенных данных')
    return parts[0], parts[1]

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('params', nargs='+', type=parse_value)
    args = parser.parse_args()
    params = dict(args.params)
    input1 = params.get('input1')
    input2 = params.get('input2')
    input3 = params.get('input3')
    output = params.get('output')
    error_file = format_file_path('errors.txt')

    try:
        network = Network(format_file_path(input1),
                           format_file_path(input2),
                           format_file_path(input3),
                           format_file_path(output))
        result = network.train()
        with open(format_file_path(output), 'w') as f:
            f.write(result)
    except Exception as e:
        print(e)
        with open(error_file, 'a') as f:
            f.write(f'{datetime.now()} : {e}\n')
```

Листинг 2 – Скрипт calculateN.py

```

import numpy as np
import json
from math import exp

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

class Layer:
    def __init__(self, name, W):
        self.name = name
        self.W = np.array(W)
        shape = self.W.shape
        self.n = shape[0]
        self.m = shape[1]
        self.X = np.zeros(self.m)
        self.Y = np.zeros(self.n)
        self.diff = np.zeros(self.n)

    def calculate(self):
        for i in range(self.n):
            y = 0
            for j in range(self.m):
                y += self.W[i][j] * self.X[j]

            self.Y[i] = sigmoid(y)
            self.diff[i] = self.Y[i] * (1 - self.Y[i])
        return self.Y

class LayerShapeError(Exception):
    pass

class Network:
    def __init__(self, input_Network, input_Train, input_Params,
output):
        weights = load(input_Network)
        self.layers = []
        sorted_layers = sorted(weights.keys())
        for w in sorted_layers:
            self.layers.append(Layer(w, weights[w]))

        self.deltas = [[] * len(self)]
        XY = load(input_Train)

        self.X = np.array(XY['X'])
        self.Y = np.array(XY['Y'])

        train_params = load(input_Params)

        self.iters = int(train_params['iters'])
        self.alpha = float(train_params['alpha'])
        self.eps = float(train_params['eps'])

```

```

def __len__(self):
    return len(self.layers)

def forward(self, input):
    k = len(self)
    for i in range(k):
        l = self.layers[i]
        if i == 0:
            l.X = input
        new_x = l.calculate()
        if i < k - 1:
            self.layers[i + 1].X = new_x
    return self.layers[-1].Y

def backward(self, output):
    error = 0
    last_layer = self.layers[-1]
    out_len = len(output)
    self.deltas[-1] = np.zeros(out_len)
    for i in range(out_len):
        e = last_layer.Y[i] - output[i]
        self.deltas[-1][i] = e * last_layer.diff[i]
        error += e * e / 2
    k = len(self)
    for i in range(k - 1, 0, -1):
        layer_i = self.layers[i]
        self.deltas[i - 1] = np.zeros(layer_i.m)
        for a in range(layer_i.m):
            for b in range(layer_i.n):
                self.deltas[i - 1][a] += layer_i.W[b][a] *
self.deltas[i][b]
                self.deltas[i - 1][a] *= self.layers[i - 1].diff[a]
    return error

def update(self):
    for i in range(len(self)):
        layer = self.layers[i]
        for a in range(layer.n):
            for b in range(layer.m):
                layer.W[a][b] -= self.alpha * self.deltas[i][a]
* layer.X[b]

def train(self):
    it = 0
    error = 1
    result = ''
    while it < self.iters and error > self.eps:
        it += 1
        errors = []
        for i in range(len(self.X)):
            out = 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: {out}')
        error = np.mean(np.array(errors))
        # print(f'it: {it}, error: {error}')
        result += f'{it}: {error}\n'
        print('-----')
    # for i in range(len(self.X)):
    #     print(f'X: {self.X[i]}, Y: {self.Y[i]}, out:
{self.forward(self.X[i])}')
    return result

def load(file):
    with open(file, 'r') as f:
        return json.load(f)

def dump(file, data):
    with open(file, 'w') as f:
        json.dump(data, f)
    print(data)

```