Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

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

Кафедра теоретических основ компьютерной безопасности и криптографии

**Нейронные сети**

ОТЧЕТ ПО ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

студента 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Старичкова Павла Александровича

|  |  |  |
| --- | --- | --- |
| Научный руководитель  Доцент | \_\_\_\_\_\_\_\_\_\_ | И. И. Слеповичев |
|  | подпись, дата |  |
|  |  |  |

Саратов 2022

1 Общие требования к выполнению заданий

1. Язык реализации: Python;
2. Каждое задание реализуется отдельной программой. В данном отчете 4 приложения: приложения А, Б, В, Г соответствуют заданиям 1, 2, 3, 4;
3. Программа запускается в консольном режиме с параметрами: «имя входного файла» «имя выходного файла»;
4. Лишние пробелы и символы табуляции во входном файле игнорируются;
5. Кодировка файлов – UTF-8.

**2.1 Создание ориентированного графа.**

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

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

**На выходе:**

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

Сериализованная структура графа в формате XML;

Листинг программы приведен в приложении А.

**Пример работы программы:**



Рисунок 1 – Запуск программы

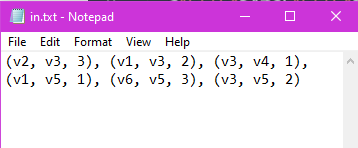


Рисунок 2 – Входной файл

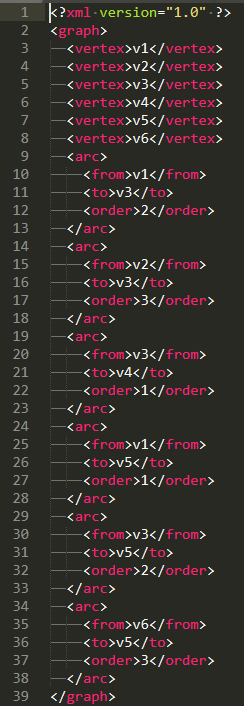


Рисунок 3 – Результат работы

2.2 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:

Листинг программы приведен в приложении Б.

**Пример работы программы:**



Рисунок 4 – Запуск программы

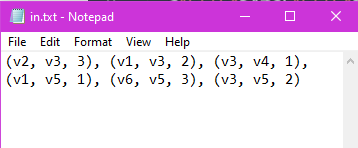


Рисунок 5 – Входной файл

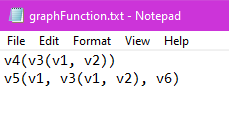


Рисунок 6 – Результат работы

2.3 Вычисление значения функции на графе

**На входе:**

* Текстовый файл с описанием графа в виде списка дуг;
* Текстовый файл соответствий арифметических операций именам вершин:

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

Допустимы следующие символы операций:

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

**На выходе:** значение функции, построенной по заданному графу и файлу операций.

Листинг программы приведен в приложении В.

**Пример работы программы:**



Рисунок 7 – Запуск программы

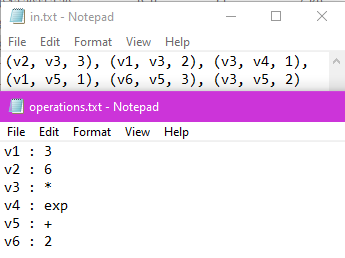


Рисунок 8 – Входные файлы

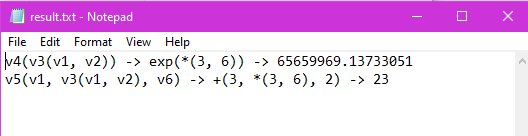


Рисунок 9 – Результат работы

2.4 Построение многослойной нейронной сети

**На входе:**

* Текстовый файл с набором матриц весов межнейронных связей:

…

* Текстовый файл с входным вектором

**На выходе:**

* Сериализованная многослойная нейронная сеть в формате XML с полносвязной межслойной структурой.
* Файл с выходным вектором - результатом вычислений НС.

В качестве функции активации используется рациональная сигмоидальная функция с параметром 1.

Листинг программы приведен в приложении Г.

**Пример работы программы:**

Запуск 1.



Рисунок 10 – Запуск программы

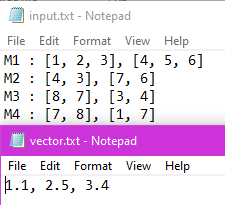


Рисунок 11 – Входные файлы

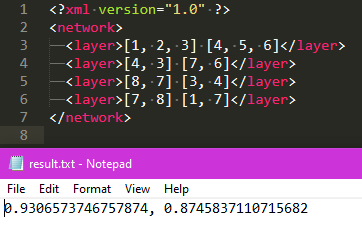


Рисунок 12 – Результат работы

Запуск 2.



Рисунок 13 – Запуск программы

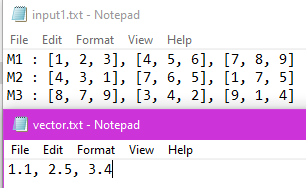


Рисунок 14 – Входные файлы

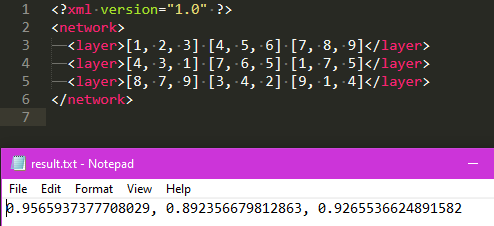


Рисунок 15 – Результат работы

ПРИЛОЖЕНИЕ А

import os

from sys import argv

import xml.etree.ElementTree as ET

import xml.dom.minidom

def parseTextFile(fileName):

if (not os.path.isfile(fileName)):

print('Не удалось открыть файл.')

exit(0)

file = open(fileName, 'r')

text = file.read()

text = text.replace('\t', ' ')

text = text.replace(' ', ' ')

edges = []

for edge in text.split('), '):

edge = edge.lstrip('(').split(', ')

edges.append([edge[0], edge[1], int(edge[2].rstrip(')'))])

return edges

def checkCorrectivity(edges):

stringNum = 0

for edge1 in edges:

stringNum += 1

for edge2 in edges:

if edge1 != edge2:

if edge1[0] == edge2[0] and edge1[1] == edge2[1]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

if edge1[1] == edge2[1] and edge1[2] == edge2[2]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

def getVertexList(edges):

edges.sort(key=lambda i: (i[1], i[2]))

vertexes = []

for edge in edges:

if edge[0] not in vertexes:

vertexes.append(edge[0])

if edge[1] not in vertexes:

vertexes.append(edge[1])

vertexes.sort()

return vertexes

def printGraphToXml(edges, outFileName):

vertexes = getVertexList(edges)

graph = ET.Element('graph')

for vertex in vertexes:

vert = ET.SubElement(graph, 'vertex')

vert.text = vertex

for edge in edges:

arc = ET.SubElement(graph, 'arc')

el1Val = ET.SubElement(arc, 'from')

el1Val.text = edge[0]

el2Val = ET.SubElement(arc, 'to')

el2Val.text = edge[1]

el3Val = ET.SubElement(arc, 'order')

el3Val.text = str(edge[2])

tree = ET.ElementTree(graph)

tree.write(outFileName, encoding='utf-8', xml\_declaration=True)

dom = xml.dom.minidom.parse(outFileName)

pretty\_xml\_as\_string = dom.toprettyxml()

with open(outFileName, 'w', encoding='utf-8') as file:

file.write(pretty\_xml\_as\_string)

if (len(argv) < 3):

print('Не хватает входных аргументов. Требуется входной файл и выходной файл.')

exit(0)

inputFile = argv[1][7::]

outputFile = argv[2][8::]

# Считывание дуг

edges = parseTextFile(inputFile)

# Проверка коррекности введенных дуг

checkCorrectivity(edges)

# Вывод в формате XML

printGraphToXml(edges, outputFile)

ПРИЛОЖЕНИЕ Б

import os

from sys import argv

import xml.etree.ElementTree as ET

import xml.dom.minidom

def parseTextFile(fileName):

if (not os.path.isfile(fileName)):

print('Не удалось открыть файл.')

exit(0)

file = open(fileName, 'r')

text = file.read()

text = text.replace('\t', ' ')

text = text.replace(' ', ' ')

edges = []

for edge in text.split('), '):

edge = edge.lstrip('(').split(', ')

edges.append([edge[0], edge[1], int(edge[2].rstrip(')'))])

return edges

def checkCorrectivity(edges):

stringNum = 0

for edge1 in edges:

stringNum += 1

for edge2 in edges:

if edge1 != edge2:

if edge1[0] == edge2[0] and edge1[1] == edge2[1]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

if edge1[1] == edge2[1] and edge1[2] == edge2[2]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

def getVertexList(edges):

edges.sort(key=lambda i: (i[1], i[2]))

vertexes = []

for edge in edges:

if edge[0] not in vertexes:

vertexes.append(edge[0])

if edge[1] not in vertexes:

vertexes.append(edge[1])

vertexes.sort()

return vertexes

def getAdjList(edges, vertexList):

adjList = {}

for vertex in vertexList:

tmp = []

for edge in edges:

if edge[1] == vertex and edge[0] not in tmp:

tmp.append(edge[0])

adjList[vertex] = tmp

return adjList

def dfs(v, colors, d):

colors[v] = 'grey'

for y in d[v]:

if colors[y] == 'white':

dfs(y, colors, d)

if colors[y] == 'grey':

print('В графе присутствует цикл.')

exit(0)

colors[v] = 'black'

def checkCycles(vertexList, adjList): # Используется обход в глубину

colors = {}

for vertex in vertexList:

colors[vertex] = 'white'

for vertex in vertexList:

dfs(vertex, colors, adjList)

return True

def getCollectorList(edgeList, vertexList):

collectorsList = []

for vertex in vertexList:

flag = True

for edge in edgeList:

if edge[0] == vertex: # Если из вершины выходит дуга, то она не является стоком

flag = False

if flag:

collectorsList.append(vertex)

return collectorsList

def buildGraphFunction(vert, graphFunction):

if len(adjList[vert]) == 0:

return graphFunction

graphFunction += '('

i = 0

for vertex in adjList[vert]:

if i != 0:

graphFunction += ', '

graphFunction += str(vertex)

graphFunction = buildGraphFunction(vertex, graphFunction)

i += 1

graphFunction += ')'

return graphFunction

if (len(argv) < 3):

print('Не хватает входных аргументов. Требуется входной файл и выходной файл.')

exit(0)

inputFile = argv[1][7::]

outputFile = argv[2][8::]

edges = parseTextFile(inputFile)

checkCorrectivity(edges)

vertexList = getVertexList(edges) # Список вершин

adjList = getAdjList(edges, vertexList) # Список смежности

checkCycles(vertexList, adjList) # Проверка на наличие циклов

collectorList = getCollectorList(edges, vertexList) # Список стоков в графе

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

for collector in collectorList: # Для каждого стока выполняется построение функции

graphFunction = str(collector)

graphFunction = buildGraphFunction(collector, graphFunction)

file.write(graphFunction + '\n')

ПРИЛОЖЕНИЕ В

import os

from sys import argv

import math

def parseTextFile(fileName):

if (not os.path.isfile(fileName)):

print('Не удалось открыть файл.')

exit(0)

file = open(fileName, 'r')

text = file.read()

text = text.replace('\t', ' ')

text = text.replace(' ', ' ')

edges = []

for edge in text.split('), '):

edge = edge.lstrip('(').split(', ')

edges.append([edge[0], edge[1], int(edge[2].rstrip(')'))])

return edges

def checkCorrectivity(edges):

stringNum = 0

for edge1 in edges:

stringNum += 1

for edge2 in edges:

if edge1 != edge2:

if edge1[0] == edge2[0] and edge1[1] == edge2[1]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

if edge1[1] == edge2[1] and edge1[2] == edge2[2]:

print('Ошибка формата. Строка: ', stringNum)

exit(0)

def getVertexList(edges):

edges.sort(key=lambda i: (i[1], i[2]))

vertexes = []

for edge in edges:

if edge[0] not in vertexes:

vertexes.append(edge[0])

if edge[1] not in vertexes:

vertexes.append(edge[1])

vertexes.sort()

return vertexes

def getAdjList(edges, vertexList):

adjList = {}

for vertex in vertexList:

tmp = []

for edge in edges:

if edge[1] == vertex and edge[0] not in tmp:

tmp.append(edge[0])

adjList[vertex] = tmp

return adjList

def dfs(v, colors, d):

colors[v] = 'grey'

for y in d[v]:

if colors[y] == 'white':

dfs(y, colors, d)

if colors[y] == 'grey':

print('В графе присутствует цикл.')

exit(0)

colors[v] = 'black'

def checkCycles(vertexList, adjList):

colors = {}

for vertex in vertexList:

colors[vertex] = 'white'

for vertex in vertexList:

dfs(vertex, colors, adjList)

return True

def getCollectorList(edgeList, vertexList):

collectorsList = []

for vertex in vertexList:

flag = True

for edge in edgeList:

if edge[0] == vertex:

flag = False

if flag:

collectorsList.append(vertex)

return collectorsList

def buildGraphFunction(vert, graphFunction):

if len(adjList[vert]) == 0:

return graphFunction

graphFunction += '('

i = 0

for vertex in adjList[vert]:

if i != 0:

graphFunction += ', '

graphFunction += str(vertex)

graphFunction = buildGraphFunction(vertex, graphFunction)

i += 1

graphFunction += ')'

return graphFunction

# Подстановка операций в функцию графа

def substituteOperations(graphFunction, vertexList, opVertDict):

for vertex in vertexList:

graphFunction = graphFunction.replace(vertex, opVertDict[vertex])

return graphFunction

def calculateGraphFunction(graphFunction):

pos1 = graphFunction.rfind('(')

if pos1 == -1:

return graphFunction

pos2 = graphFunction.find(')', pos1)

operation = graphFunction[pos1 - 1 : pos1]

opPos = pos1 - 1

if operation == 'p':

operation = graphFunction[pos1 - 3 : pos1]

opPos = pos1 - 3

values = graphFunction[pos1 + 1 : pos2].split(', ')

if '+' in values or '\*' in values or 'exp' in values:

print('Ошибка вычисления. На месте одной из операций должно быть число.')

return

values = list(map(int, values))

if operation == '+':

newVal = sum(values, 0)

elif operation == '\*':

newVal = math.prod(values)

elif operation == 'exp':

newVal = math.exp(values[0])

graphFunction = graphFunction.replace(graphFunction[opPos : pos2 + 1], str(newVal))

return calculateGraphFunction(graphFunction)

if len(argv) < 4:

print('Не хватает входных аргументов.')

exit(0)

inputFile = argv[1][7::]

operationsInputFile = argv[2][7::]

outputFile = argv[3][8::]

edges = parseTextFile(inputFile)

checkCorrectivity(edges)

vertexList = getVertexList(edges)

adjList = getAdjList(edges, vertexList)

checkCycles(vertexList, adjList)

collectorList = getCollectorList(edges, vertexList)

graphFunctionList = []

for collector in collectorList:

graphFunction = str(collector)

graphFunction = buildGraphFunction(collector, graphFunction)

graphFunctionList.append(graphFunction)

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

operations = file.read().splitlines(False)

# Словарь соответствия вершин и операций

opVertDict = {}

for op in operations:

vert, operation = op.split(':')

vert = vert.strip(' ')

operation = operation.strip(' ')

opVertDict[vert] = operation

# Заменяем вершины на операции и значения в строке функции графа

graphOperationList = []

for fun in graphFunctionList:

graphOperationList.append(substituteOperations(fun, vertexList, opVertDict))

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

for i in range(len(graphOperationList)):

file.write(f'{graphFunctionList[i]} -> {graphOperationList[i]} -> {calculateGraphFunction(graphOperationList[i])}\n')

ПРИЛОЖЕНИЕ Г

import os

from sys import argv

import xml.etree.cElementTree as ET

from xml.dom import minidom

def parseTextFile(fileName):

if (not os.path.isfile(fileName)):

print('Не удалось открыть файл.')

exit(0)

file = open(fileName, 'r')

lines = file.readlines()

matrixList = []

for line in lines:

\_, mat = line.split(':')

mat = mat.strip(' ')

mat = mat.strip('\n')

pos1 = 0

pos2 = 0

tmpMatrix = []

floatFlag = False

while True:

pos1 = mat.find('[', pos2)

pos2 = mat.find(']', pos1)

if pos1 == -1 or pos2 == -1:

break

row = mat[pos1 + 1 : pos2]

if floatFlag or '.' in row:

floatFlag = True

row = list(map(float, row.split(', ')))

else:

row = list(map(int, row.split(', ')))

tmpMatrix.append(row)

matrixList.append(tmpMatrix)

return matrixList

def transmuteVector(vector):

if '.' in vector:

vector = list(map(float, vector.split(', ')))

else:

vector = list(map(int, vector.split(', ')))

return vector

def rationalSigmoidFun(z):

return z / (abs(z) + 1)

def calculate(matrixList, vector):

errIdx = 0

for layer in matrixList:

errIdx += 1

newVector = []

for neuron in layer:

val = 0

if len(vector) < len(neuron):

print(f'Длина входного вектора меньше количества весов нейрона в строке {errIdx} входного файла.')

exit(0)

for i in range(len(vector)): # Вычисление суммы произведения весов нейронов на значения входного вектора

try:

val += neuron[i] \* vector[i]

except IndexError:

print(f'Ошибка. Неверное число весов в строке {errIdx}.')

exit(0)

val = rationalSigmoidFun(val) # Применение функции активации

newVector.append(val)

vector = newVector

return vector

if len(argv) < 5:

print('Не хватает входных аргументов. Треубется 2 входных файла и 2 выходных.')

exit(0)

inputFile = argv[1][7::]

vectorInputFile = argv[2][7::]

outputXML = argv[3][8::]

outputFile = argv[4][8::]

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

vector = file.read()

try:

vector = transmuteVector(vector)

except:

print('Ошибка в формате входного вектора. Преполагаемый формат: x1, x2, x3, x4')

exit(0)

matrixList = parseTextFile(inputFile)

calcResult = calculate(matrixList, vector)

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

file.write(', '.join(list(map(str, calcResult))))

root = ET.Element('network')

for layer in matrixList:

layerVal = ' '.join(list(map(str, layer)))

ET.SubElement(root, 'layer').text = layerVal

dom = minidom.parseString(ET.tostring(root))

tree = dom.toprettyxml(indent='\t')

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

file.write(tree)