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 ПРАКТИЧЕСКОМУ КУРСУ**

студента 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Минуситова Амиля Куанышкалиевича

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Научный руководитель  д.ф.-м.н.,доцент | \_\_\_\_\_\_\_\_\_\_ | И. И. Слеповичев |
|  | подпись, дата |  |

Саратов 2022

**Задание №1:**

Создать ориентированный граф, по заданным дугам.

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

(v1, u1, n1), (v2, u2, n2), ..., (vk, uk, nk)

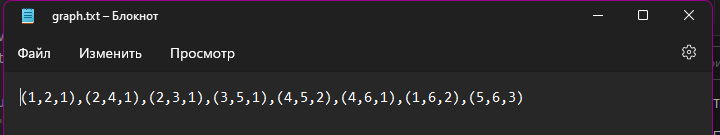
где vi- начальная вершина дуги i, ui- конечная вершина дуги i, ni- порядковый номер дуги в списке всех заходящих в вершину ui дуг.

**На выходе:**

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

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

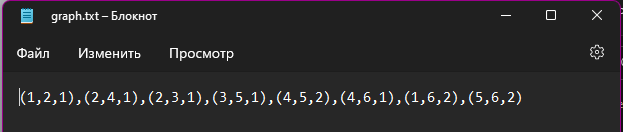
Входной файл

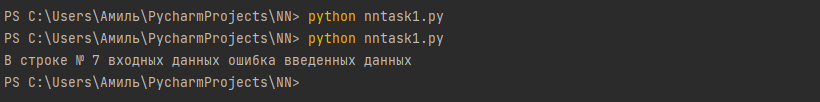


Выходной файл



Изменив в 7 дуге порядковый номер дуги входящей в вершину 6 получаем сообщение об ошибке.





Исходный текст программы:

import xml.etree.ElementTree as xml  
from re import split  
  
  
def write\_file(edges, v):  
 graph = xml.Element("graph")  
 for i in v:  
 vertex = xml.SubElement(graph, "vertex")  
 vertex.text = 'v' + str(i)  
 for i in edges:  
 arc = xml.SubElement(graph, "arc")  
 fromx = xml.SubElement(arc, "from")  
 fromx.text = 'v' + i[0]  
 to = xml.SubElement(arc, "to")  
 to.text = 'v' + i[2]  
 order = xml.SubElement(arc, "order")  
 order.text = i[4]  
 tree = xml.ElementTree(graph)  
 tree.write('graph1.xml')  
  
  
def read\_graph():  
 with open('graph.txt', 'r') as input\_graph:  
 j = 0  
 for line in input\_graph:  
 j += 1  
 edges = split('\),\(', line)  
 n = len(edges)  
 for i in range(n):  
 edges[i] = edges[i].replace('(', '')  
 edges[i] = edges[i].replace(')', '')  
  
 for i in range(n - 1):  
 a = edges[i]  
 # print(edge1)  
 for j in range(1, n):  
 b = edges[j]  
 # print(edge2)  
 if a != b:  
 if a[2] == b[2] and a[4] == b[4]:  
 print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))  
 exit()  
 if a[0] == b[0] and a[2] == b[2]:  
 print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))  
 exit()  
 edges.sort(key=lambda i: (i[0], i[2]))  
 c = edges  
 k = []  
 for i in range(n):  
 k.append(([int(edges[i][0]), int(edges[i][2]), int(edges[i][4])]))  
 edges = k  
 # print(edges)  
 v = []  
 for i in range(n):  
 v.append(edges[i][0])  
 v.append(edges[i][1])  
 v.sort()  
 vertex = []  
 for x in v:  
 if x not in vertex:  
 vertex.append(x)  
 return c, vertex  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 c, vertex = read\_graph()  
 write\_file(c, vertex)

**Задание №2:**

Создать линейное представление функции описывающей граф

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

(v1, u1, n1), (v2, u2, n2), ..., (vk, uk, nk)

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

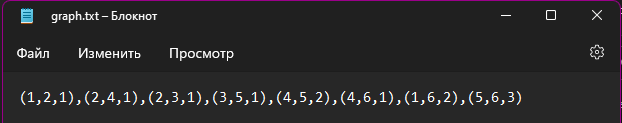
**На выходе:**

а) линейное представление функции, реализуемой графом в префиксной скобочной записи:

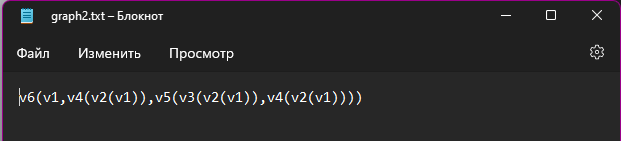
A1(B1(C1(...),..., Cm(...)),..., Bn(...))

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

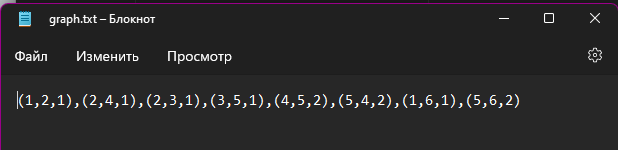
Входной файл

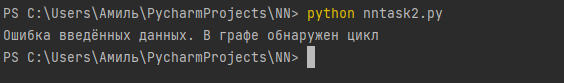
****

Выходной файл

****

Между вершинами 4 и 5 явно зададим цикл. Получаем сообщение о том что в графе имеется цикл.

****

****

Исходный текст программы:

from re import split

def read\_graph():

with open('graph.txt', 'r') as input\_graph:

j = 0

for line in input\_graph:

j += 1

edges = split('\),\(', line)

n = len(edges)

for i in range(n):

edges[i] = edges[i].replace('(', '')

edges[i] = edges[i].replace(')', '')

for i in range(n - 1):

a = edges[i]

# print(edge1)

for j in range(1, n):

b = edges[j]

# print(edge2)

if a != b:

if a[2] == b[2] and a[4] == b[4]:

print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))

exit()

if a[0] == b[0] and a[2] == b[2]:

print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))

exit()

edges.sort(key=lambda i: (i[0], i[2]))

c = edges

k = []

for i in range(n):

k.append(([int(edges[i][0]), int(edges[i][2]), int(edges[i][4])]))

edges = k

# print(edges)

v = []

for i in range(n):

v.append(edges[i][0])

v.append(edges[i][1])

v.sort()

vertex = []

for x in v:

if x not in vertex:

vertex.append(x)

return (edges, vertex)

def matr(edges, vertex):

n = len(vertex)

m = [[0]\*n for i in range(n)]

for a in edges:

m[a[0]-1][a[1]-1] = 1

return m

def stok(matr):

n = len(matr)

crs = []

for r in range(n):

for c in range(n):

if matr[r][c] != 0:

break

else:

crs.append(r + 1)

return crs

def dfs(v, color, d):

color[v] = 'grey'

for y in d[v]:

if color[y] == 'white':

dfs(y, color, d)

if color[y] == 'grey':

print('Ошибка введённых данных. В графе обнаружен цикл')

exit()

color[v] = 'black'

def Function(x, func): # вычисление функции графа

if len(d[x]) == 0:

return func

func += '('

t = False

for v in d[x]:

if t == True:

func += ','

func += 'v' + str(v)

t = True

func = Function(v, func)

func += ')'

return func

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

edges, vertex = read\_graph()

#print(edges)

d = {}

color = {}

for v in vertex:

color[v] = 'white'

tmp = []

for edge in edges:

if edge[1] == v:

tmp.append(edge[0])

d[v] = tmp

for v in vertex:

dfs(v, color, d)

crc = stok(matr(edges, vertex))

f1 = open('graph2.txt', 'w')

for x in crc:

func ='v' + Function(x, str(x))

f1.write(func + '\n')

**Задание №3:**

Вычисление значения функции на графе

**На входе:**

а) Текстовый файл с описанием графа в виде списка дуг.

б) Текстовый файл соответствий арифметических операций именам вершин:

a1 : операция 1

a2 : операция 2

...

an : операция n

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

Допустимы следующие символы операций:

+ – cумма значений,

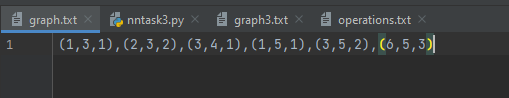
\* – произведение значений,

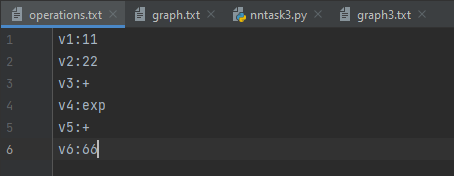
exp – экспонирование входного значения,

число – любая числовая константа.

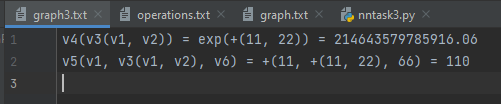
**На выходе:** значение функции, построенной по графу а) и файлу б).

Входные файлы:

а)

б)

Выходной файл:



Исходный текст программы:

from re import split

import math

def read\_graph():

with open('graph.txt', 'r') as input\_graph:

j = 0

for line in input\_graph:

j += 1

edges = split('\),\(', line)

n = len(edges)

for i in range(n):

edges[i] = edges[i].replace('(', '')

edges[i] = edges[i].replace(')', '')

for i in range(n - 1):

a = edges[i]

# print(edge1)

for j in range(1, n):

b = edges[j]

# print(edge2)

if a != b:

if a[2] == b[2] and a[4] == b[4]:

print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))

exit()

if a[0] == b[0] and a[2] == b[2]:

print('В строке № {} входных данных ошибка введенных данных'.format(str(j)))

exit()

edges.sort(key=lambda i: (i[0], i[2]))

c = edges

k = []

for i in range(n):

k.append(([int(edges[i][0]), int(edges[i][2]), int(edges[i][4])]))

edges = k

# print(edges)

v = []

for i in range(n):

v.append(edges[i][0])

v.append(edges[i][1])

v.sort()

vertex = []

for x in v:

if x not in vertex:

vertex.append(x)

return (edges, vertex)

def matr(edges, vertex):

n = len(vertex)

m = [[0]\*n for i in range(n)]

for a in edges:

m[a[0]-1][a[1]-1] = 1

return m

def stok(matr):

n = len(matr)

crs = []

for r in range(n):

for c in range(n):

if matr[r][c] != 0:

break

else:

crs.append(r + 1)

return crs

def dfs(v, color, d):

color[v] = 'grey'

for y in d[v]:

if color[y] == 'white':

dfs(y, color, d)

if color[y] == 'grey':

print('Ошибка введённых данных. В графе обнаружен цикл')

exit()

color[v] = 'black'

def dfsres(v, color, d):

color[v] = 'grey'

for y in d[v]:

if color[y] == 'white':

dfs(y, color, d)

if color[y] == 'grey':

print('Ошибка введённых данных. В графе обнаружен цикл')

exit()

color[v] = 'black'

def Function(x, func): # вычисление функции графа

if len(d[x]) == 0:

return func

func += '('

t = False

for v in d[x]:

if t == True:

func += ', '

func += f'v{v}'

t = True

func = Function(v, func)

func += ')'

return func

def resultfunc(func):

start = 0

while True:

start = func.rfind('(')

if start == -1:

return func

finish = func.find(')', start)

numbers = func[start + 1:finish:]

if func[start - 1] == '\*':

func = func.replace(func[start - 1:finish + 1:], str(numresultfunc(func[start + 1:finish:], '\*')))

if func[start - 1] == '+':

func = func.replace(func[start - 1:finish + 1:], str(numresultfunc(func[start + 1:finish:], '+')))

if func[start - 3:start:] == 'exp':

func = func.replace(func[start - 3:finish + 1:], str(numresultfunc(func[start + 1:finish:], 'exp')))

def numresultfunc(func, operation):

cnt = []

for x in func.split(', '):

cnt.append(int(x))

if operation == 'exp':

return math.exp(cnt[0])

if operation == '+':

result = 0

for x in cnt:

result += x

return result

if operation == '\*':

result = 1

for x in cnt:

result \*= x

return result

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

edges, vertex = read\_graph()

d = {}

color = {}

for v in vertex:

color[v] = 'white'

tmp = []

for edge in edges:

if edge[1] == v:

tmp.append(edge[0])

d[v] = tmp

#print(vertex)

vert = []

for v in vertex:

dfs(v, color, d)

crc = stok(matr(edges, vertex))

#print(crc)

f = open('operations.txt', 'r')

operation = {}

f1 = open('graph3.txt', 'w')

for row in f.read().split('\n'):

tmp = row.split(':')

operation[tmp[0]] = tmp[1]

for x in crc:

func = 'v' + Function(x, str(x))

#print(func)

result = func + ' = '

for x in operation:

func = func.replace(x, operation[x])

#f1.write(func + '\n')

#print(func)

result += func + ' = ' + resultfunc(func)

print(result)

f1.write(result + '\n')

#print(operation)

**Задание №4:** **Построение многослойной нейронной сети**

**На входе:**

А) Текстовый файл с набором матриц весов межнейронных связей:

M1: [M1[1,1], M1[1,2], …, M1[1,n]], …, [M1[m,1], M1[m,2], …, M1[m, n]]

M2: [M2[1,1], M2[1,2], …, M2[1,n]], … , [M2[m,1], M2[m,2], … , M2[m,n]]

…

Mp: [Mp[1,1], Mp[1,2], …, Mp[1,n]], … , [Mp[m,1], Mp[m,2], … , Mp[m,n]]

Б) Текстовый файл с входным вектором

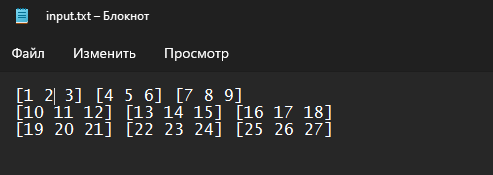
**На выходе:**

А) Сериализованная многослойная нейронная сеть (в формате XML) с полносвязной межслойной структурой.

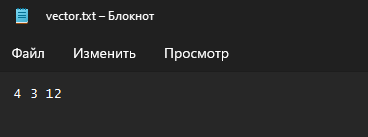
Файл с выходным вектором – результатом вычислений НС.

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

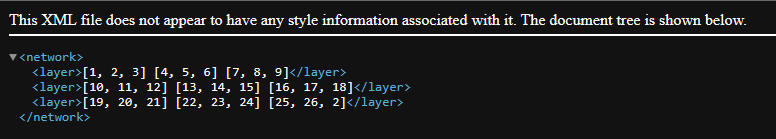
Входной файл с набором матриц весов межнейронных связей



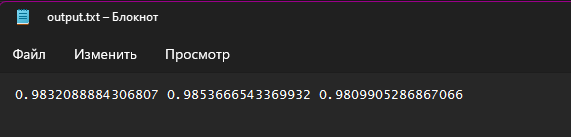
Входной файл с вектором



Сериализованная многослойная нейронная сеть в формате XML с полносвязной межслойной структурой



Файл с выходным вектором



Исходный текст программы:

import sys

import math

import argparse

import xml.etree.cElementTree as ET

from xml.dom import minidom

def read(textfile, input):

vector = open(textfile).read().split(" ")

vector = [int(x) for x in vector]

matrix = []

in\_file = open(input).readlines()

indx = 0

length = len(vector)

for line in in\_file:

indx += 1

line = line.replace("[", " ")

line = line.replace("]", " ")

line = line[1:-2]

line = line.split(" ")

tmp = []

for x in line:

x = x.split(" ")

try:

x = [int(i) for i in x]

tmp.append(x)

except ValueError:

print("Ошибка в строке " + str(indx))

sys.exit(1)

if len(x) != length:

print("Не совпадает число компонент нейронов в слoях " + str(indx - 1) + " - " + str(indx))

sys.exit(1)

length = len(line)

matrix.append([tmp])

return matrix, vector

def evaluate(matrix, vector):

new\_matrix = []

for layer in matrix:

tmp = []

for x in layer:

for neuron in x:

value = 0

for i in range(len(vector)):

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

value = value / (1 + abs(value))

tmp.append(value)

new\_matrix.append(tmp)

vector = tmp

return new\_matrix

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('-i', required=True, help='input.txt')

parser.add\_argument('-v', required=True, help='vector.txt')

parser.add\_argument('-o', required=True, help='output.txt')

args = parser.parse\_args()

matrix, vector = read(args.v, args.i)

new\_matrix = evaluate(matrix, vector)

with open(args.o, 'w') as output:

for x in new\_matrix[-1]:

output.write(str(x) + " ")

root = ET.Element("network")

for layer in matrix:

tmp = ""

for x in layer:

for y in x:

tmp += str(y) + " "

break

tmp = tmp[:-1]

ET.SubElement(root, "layer").text = tmp

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

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

with open("output.xml", 'w') as file:

file.write(tree)

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

main()

**Задание №4:** **Реализация метода обратного распространения ошибки для многослойной НС**

**На входе:**

а) Текстовый файл с набором матриц весов межнейронных связей:

M1: [M1[1,1], M1[1,2], …, M1[1,n]], …, [M1[m,1], M1[m,2], …, M1[m, n]]

M2: [M2[1,1], M2[1,2], …, M2[1,n]], … , [M2[m,1], M2[m,2], … , M2[m,n]]

…

Mp: [Mp[1,1], Mp[1,2], …, Mp[1,n]], … , [Mp[m,1], Mp[m,2], … , Mp[m,n]]

б) Текстовый файл с обучающей выборкой:

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

...

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

в) Число итераций обучения (в строке параметров).

**На выходе:**

Текстовый файл с историей 777 итераций обучения методом обратного распространения ошибки:

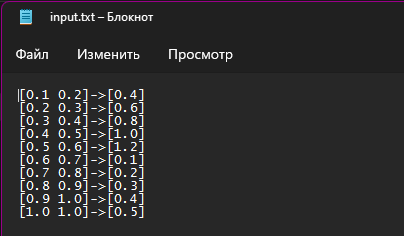
1 : Ошибка1

2 : Ошибка2

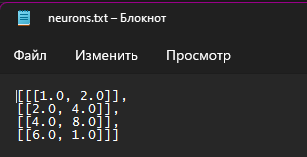
...

N : ОшибкаN

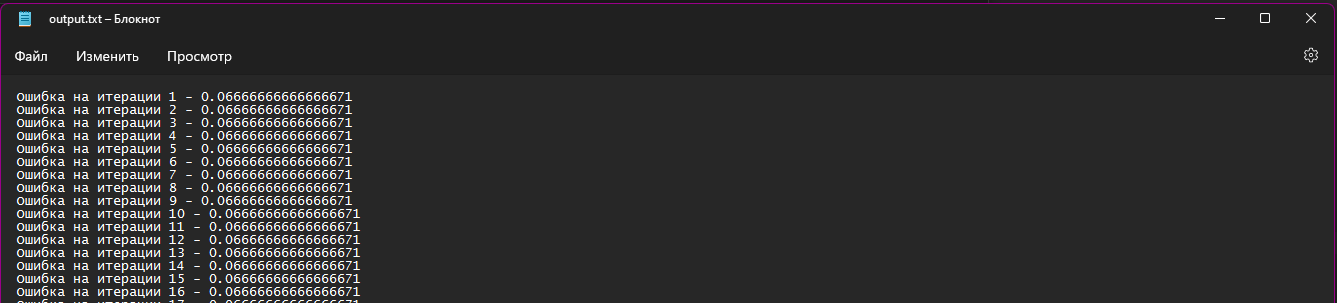
Входной файл с обучающей выборкой

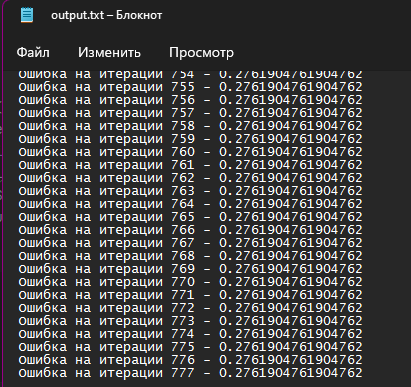


Входной файл с набором матриц весов межнейронных связей



Выходной файл с историей 777 итераций обучения методом обратного распространения ошибок





Исходный текст программы:

import ast

import sys

def read(input\_file, neurons\_file):

try:

input = open(input\_file).readlines()

samples = []

for line in input:

line = line.replace("\n", "")

mass = line.split("->")

ver = []

for neuron in mass:

neuron = neuron[1:-1].split(" ")

ver.append(neuron)

samples.append(ver)

for x in range(len(samples)):

for y in range(len(samples[x])):

for z in range(len(samples[x][y])):

samples[x][y][z] = float("".join(samples[x][y][z]))

neurons = open(neurons\_file).read()

neurons = ast.literal\_eval(neurons)

return samples, neurons

except:

print("Ошибка в входном файле")

sys.exit(1)

class Selection:

def \_\_init\_\_(self, parameters, neurons, samples):

self.parameters = parameters

self.values = neurons

self.samples = samples

self.functions = []

self.weights = []

for i in range(len(neurons)):

self.functions.append([])

self.weights.append([])

def check(self, input):

result = []

result.extend(input)

for j in range(len(self.values)):

resultSize = len(result)

for k in range(len(self.values[j])):

if resultSize != len(self.values[j][k]):

print("Число нейронов не совпадает с длиной массива весов")

sys.exit(1)

summa = 0.0

for i in range(len(self.values[j][k])):

summa += result[i] \* self.values[j][k][i]

self.weights[j].append(summa)

summa = summa / (1 + abs(summa))

self.functions[j].append(summa)

result.append(summa)

if resultSize > 0:

result[:resultSize] = []

return result

def educ\_neuron(self, output):

eps = self.parameters[1]

n = self.parameters[0]

resString = []

step = 3

for sample in range(len(self.samples)):

resString.append([])

for sample in range(step):

for count in range(n + 1):

input = self.samples[sample][0]

expected = self.samples[sample][1]

resOfPerceptron = self.check(input)

if count != 0:

resString[sample].append("Ошибка на итерации" + " " + str(count) + " - " + str(expected[0] - resOfPerceptron[0])+ "\n")

sigma = []

for x in range(len(self.values)):

sigma.append([])

for i in range(len(resOfPerceptron)):

sigma[len(self.values) - 1] = [expected[i] - resOfPerceptron[i]]

for i in range(len(self.values) - 1, 0, -1):

for j in range(len(self.values[i][i - 1])):

sum = 0.0

for z in range(len(sigma[i])):

sum += abs(sigma[i][z]) \* self.values[i][z][j]

sigma[i - 1].append([sum])

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

for j in range(len(self.values[i])):

for weight in range(len(self.values[i][j])):

der = 1 / pow((1 + abs(self.weights[i][j])), 2)

deltaWeight = sigma[i][j] \* der \* self.functions[i-1][weight] \* eps

self.values[i][j][weight] = self.values[i][j][weight] + deltaWeight

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

for line in resString:

for x in line:

file.write(x)

def main():

values, layers = read('input.txt', 'neurons.txt')

n = int(777)

epsil = float(0.1)

x = Selection([n, epsil], [layers[0]], values)

x.educ\_neuron('output.txt')

main()