**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине: «Введение в нейронные сети»

на тему: **«**Контролируемое обучение нейронных сетей**»**

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.

Принял: преподаватель-стажёр

Дашкевич Д.А.

Гомель 2021

**Цель работы**: разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием простейших нейронных сетей.

**Задание:**

В качестве параметров программа должна принимать:

– файл, содержащий;

– ссылки на файлы с обучающей выборкой;

– ссылки на файлы с тестируемыми образцами;

– ссылку на файл с результатами проверки образцов (принадлежит / не

принадлежит).

Если происходит запуск без параметров, то запускается программа с графическим интерфейсом, ссылки на обучающую выборку берутся из конфигурационного файла аналогичной структуры, как и файл параметров командной строки.

Архитектуру нейронной сети выбрать самостоятельно. На входном слое не должно быть менее 25 нейронов и не более 36. Количество скрытых слоёв не должно быть менее 1 и более 3-х. Изображения использовать бинаризованные.

В качестве первичного алгоритма обучения использовать алгоритм, указанный в таблице, согласно варианта. Дополнить разработанное программное обеспечение методом обратного распространения ошибки (*backpropagation*). Сравнить с помощью графиков качество и скорость обучения одной и той же сети различными методами.

В отчёт представить архитектуру нейронной сети и графики, отражающие сравнительный анализ скорости и качества обучения на одних и тех же данных разными методами.

На рисунке 1 представлена схема нейронной сети.

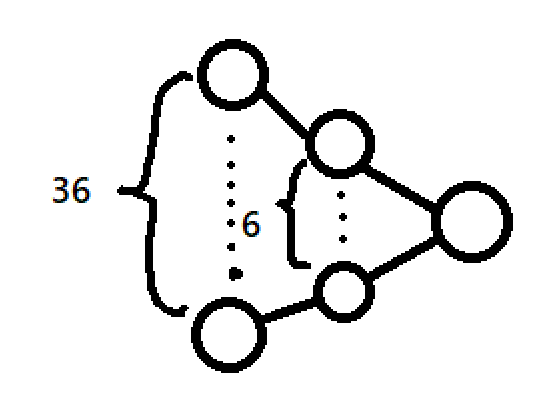


Рисунок 1 – Схема нейронной сети

Распознаваемые классы: стоящий и идущий человек.

Правила Хебба и метод обратного распространения ошибки.

**Ход работы**

На рисунке 2 представлен один из человек выборки.

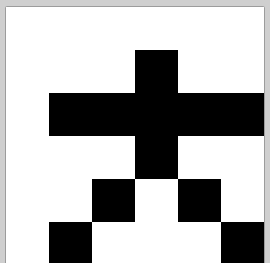


Рисунок 2 – Пример человека выборки

На рисунке 3 представлен графический интерфейс программного обеспечения.

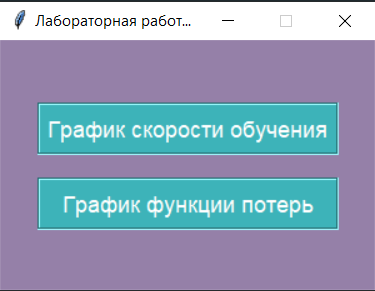


Рисунок 3 – Графический интерфейс программного обеспечения

На рисунке 4 представлен пример конфигурационного файла.

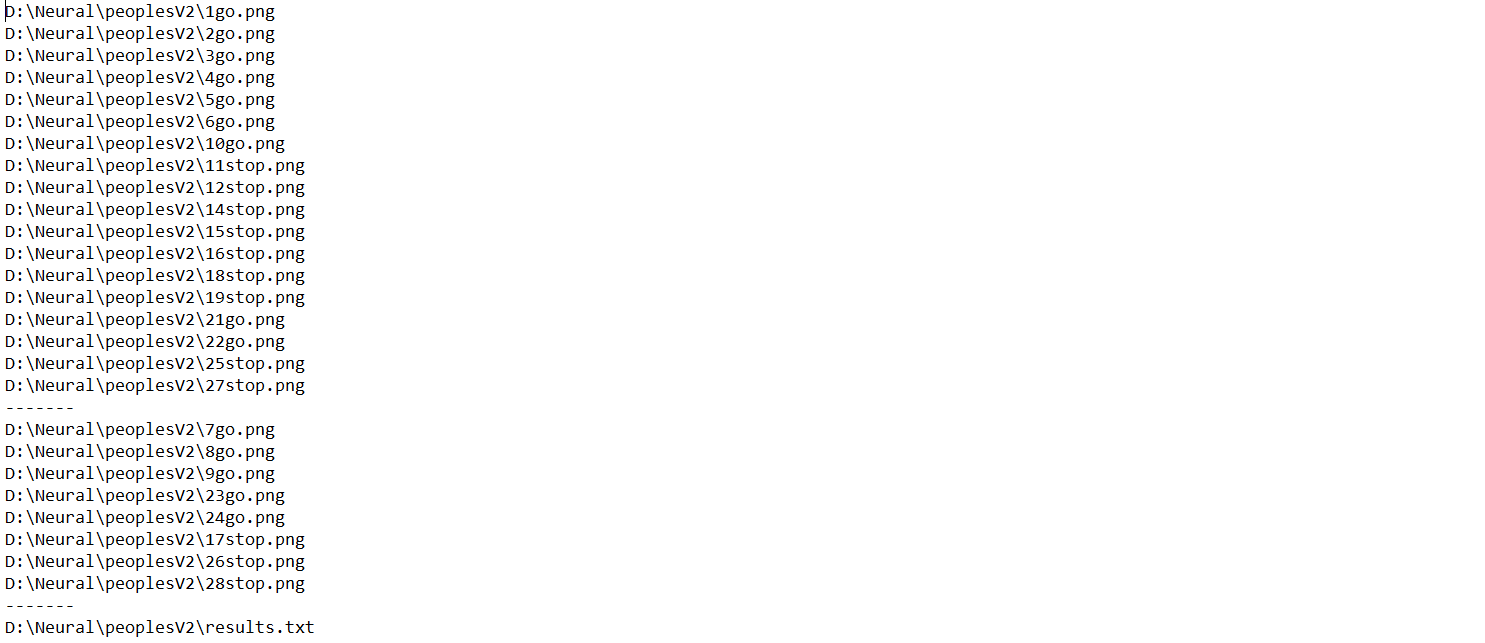


Рисунок 4 – Пример конфигурационного файла

На рисунке 5 представлен пример файла с результатами проверки тестируемых образцов.

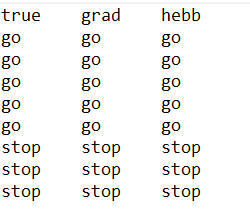


Рисунок 5 – Пример файла с результатами проверки тестируемых образцов

На рисунке 6 представлен график сравнения скорости обучения методами Хебба и обратного распространения ошибки.

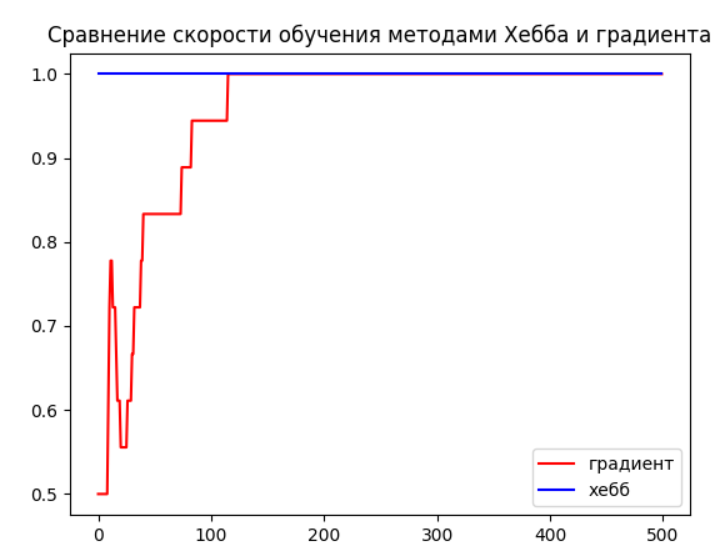


Рисунок 6 – График сравнения скорости обучения

На рисунке 7 представлен график сравнения качества обучения методами Хебба и обратного распространения ошибки.

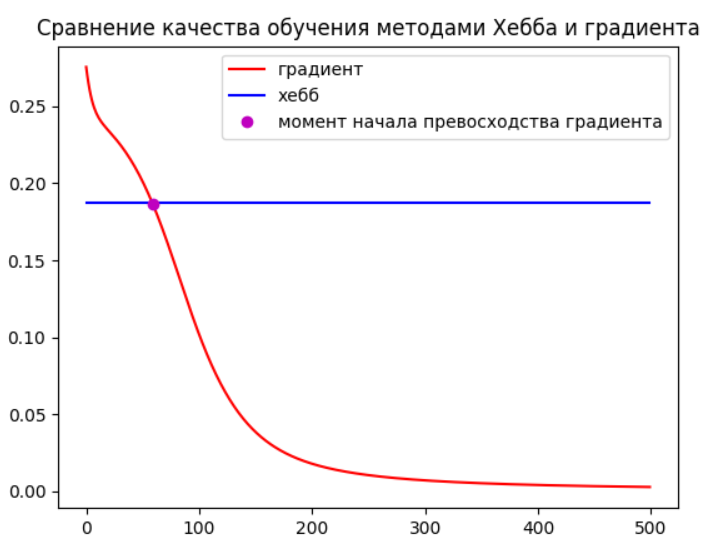


Рисунок 7 – График сравнения качества обучения

Время прохождения одной эпохи у метода Хебба оказалось выше, чем у метода обратного распространения ошибки.

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы выбрана и обучена нейронная сеть методами Хебба и обратного распространения ошибки. Обучения методом обратного распространения ошибки требует гораздо большее количество эпох, но в то же время обучение данным методом является качественнее, чем методом Хебба.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

import numpy as np

import random

import cv2

import os

from tkinter import \*

from tkinter import filedialog

import matplotlib.pyplot as plt

def main():

mainWindow = Tk()

mainWindow.resizable(width=False, height=False)

mainWindow.title("Лабораторная работа 2")

screenWidth = mainWindow.winfo\_screenwidth()

screenHeight = mainWindow.winfo\_screenheight()

x = (screenWidth -600)/ 2

y = (screenHeight -500)/ 2

mainWindow.geometry('%dx%d+%d+%d' % (600, 500, x, y))

mainWindow.configure(background="#3db3b9")

#Создание меню

xCoordinate = 600/2-230/2

buttonColor="#c22323"

buttonWidth="21"

button1 = Button(text="Выбрать файл и вычислить", command=lambda:openFile(False),bg=buttonColor,fg="white",width=buttonWidth, height="1",font=("MS Sans Serif", 14),relief="groove",activebackground="#dc3f3f")

button1.place(x=xCoordinate, y=90)

button2 = Button(text="График скорости обучения", command=lambda:createGraphic(),bg=buttonColor,fg="white",width=buttonWidth, height="1",font=("MS Sans Serif", 14),relief="groove",activebackground="#dc3f3f")

button2.place(x=xCoordinate, y=150)

button3 = Button(text="График функции потерь", command=lambda:createLossGraphic(),bg=buttonColor,fg="white",width=buttonWidth, height="1",font=("MS Sans Serif", 14),relief="groove",activebackground="#dc3f3f")

button3.place(x=xCoordinate, y=210)

openFile(True)

mainWindow.mainloop()

def createGraphic():

global isNew

global hebbInfo

global gradInfo

global trainData

global train\_y\_trues

if isNew:

network = GradientNeuralNetwork()

gradInfo = []

network.train(trainData, train\_y\_trues, gradInfo, 500)

network = HebbNeuralNetwork()

hebbInfo = []

network.train(trainData, train\_y\_trues, hebbInfo, 500)

isNew = False

fig, ax = plt.subplots()

ax.plot(gradInfo[0],gradInfo[1],'r', label = 'градиент')

ax.plot(hebbInfo[0],hebbInfo[1],'b', label = 'хебб')

ax.legend()

plt.title('Сравнение скорости обучения методами Хебба и градиента')

plt.show()

def createLossGraphic():

global gradLossInfo

global hebbLossInfo

global isNew

global hebbInfo

global gradInfo

global trainData

global train\_y\_trues

if isNew:

network = GradientNeuralNetwork()

gradInfo = []

network.train(trainData, train\_y\_trues, gradInfo, 500)

network = HebbNeuralNetwork()

hebbInfo = []

network.train(trainData, train\_y\_trues, hebbInfo, 500)

isNew = False

fig, ax = plt.subplots()

ax.plot(gradLossInfo[0],gradLossInfo[1],'r', label = 'градиент')

ax.plot(hebbLossInfo[0],hebbLossInfo[1],'b', label = 'хебб')

cross = [[],[]]

for i in range(len(gradLossInfo[0])):

print(abs(gradLossInfo[1][i]-hebbLossInfo[1][i]))

if abs(gradLossInfo[1][i]-hebbLossInfo[1][i]) < 0.001:

cross[0].append(gradLossInfo[0][i])

cross[1].append(gradLossInfo[1][i])

plt.plot(cross[0][len(cross[0])-1], cross[1][len(cross[1])-1], 'mo',label = 'момент начала превосходства градиента')

ax.legend()

plt.title('Сравнение качества обучения методами Хебба и градиента')

plt.show()

def openFile(isDefault):

global isNew

global trainData

global train\_y\_trues

if isDefault==True:

path=os.path.dirname(os.path.abspath(\_\_file\_\_))+'\\config.txt'

else:

path = filedialog.askopenfilename(filetypes=(("Текстовый файл", "\*.txt"),("Текстовый файл", "\*.txt")))

if path!='':

images = []

lines = open(path).readlines()

files = []

j = 0

for i in range(3):

files.append([])

while j < len(lines) and lines[j].\_\_contains\_\_("-----") == False:

files[i].append(lines[j].replace("\n",""))

j += 1

j += 1

train\_y\_trues = []

for i in range(len(files[0])):

if files[0][i].\_\_contains\_\_(".png"):

images.append(binarize(cv2.imread(files[0][i],cv2.IMREAD\_GRAYSCALE)))

if files[0][i].\_\_contains\_\_("go"):

train\_y\_trues.append(1)

else:

train\_y\_trues.append(0)

print(train\_y\_trues)

trainData = np.array(images)

test\_y\_trues = []

images = []

for i in range(len(files[1])):

if files[1][i].\_\_contains\_\_(".png"):

images.append(binarize(cv2.imread(files[1][i],cv2.IMREAD\_GRAYSCALE)))

if files[1][i].\_\_contains\_\_("go"):

test\_y\_trues.append(1)

else:

test\_y\_trues.append(0)

testData = np.array(images)

gradNetwork = GradientNeuralNetwork()

gradNetwork.train(trainData, train\_y\_trues, None, 2000)

hebbNetwork = HebbNeuralNetwork()

hebbNetwork.train(trainData, train\_y\_trues, None, 2000)

test(gradNetwork,hebbNetwork, testData, test\_y\_trues, files[2][0])

isNew = True

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def derivSigmoid(x):

fx = sigmoid(x)

return fx \* (1 - fx)

def test(grad,hebb, images,y\_true,resultPath):

grad\_y\_pred = []

for i in range(len(images)):

hiddenNeurons = []

for k in range(6):

hiddenSum = 0

for j in range(k\*6,(k+1)\*6):

hiddenSum += grad.firstWeights[j] \* images[i][j]

hiddenNeurons.append(sigmoid(hiddenSum))

sumOutput = 0

for z in range(6):

sumOutput += grad.secondWeights[z] \* hiddenNeurons[z]

output = sigmoid(sumOutput)

grad\_y\_pred.append(output)

hebb\_y\_pred = []

for i in range(len(images)):

hiddenNeurons = []

for k in range(6):

hiddenSum = 0

for j in range(k\*6,(k+1)\*6):

hiddenSum += hebb.firstWeights[j] \* images[i][j]

hiddenNeurons.append(hebbSigmoid(hiddenSum))

sumOutput = 0

for z in range(6):

sumOutput += hebb.secondWeights[z] \* hiddenNeurons[z]

output = hebbSigmoid(sumOutput)

hebb\_y\_pred.append(output)

f = open(resultPath, 'w')

resultStr = "true grad hebb\n"

for i in range(len(y\_true)):

if y\_true[i] == 1:

resultStr += '%-8s' % "go"

else:

resultStr += '%-8s' % "stop "

if round(grad\_y\_pred[i]) == 1:

resultStr += '%-8s' % "go"

else:

resultStr += '%-8s' % "stop"

if hebb\_y\_pred[i] == 1:

resultStr += '%-8s' % "go"

else:

resultStr += '%-8s' % "stop"

resultStr += "\n"

f.write(resultStr)

f.close()

def mseLoss(y\_true, y\_pred):

sum = 0.0

for i in range(len(y\_true)):

sum += (y\_true[i] - y\_pred[i]) \*\* 2

return sum/len(y\_true)

class GradientNeuralNetwork:

def \_\_init\_\_(self):

# Веса

self.firstWeights = [np.random.normal()]\*36

self.secondWeights = [np.random.normal()]\*6

def feedforward(self, x):

hiddenNeurons = []

for i in range(6):

hiddenSum = 0

for j in range(i\*6,(i+1)\*6):

hiddenSum += self.firstWeights[j] \* x[j]

hiddenNeurons.append(sigmoid(hiddenSum))

sumOutput = 0

for i in range(6):

sumOutput += self.secondWeights[i] \* hiddenNeurons[i]

output = sigmoid(sumOutput)

return output

def train(self, data, all\_y\_trues, infoData, epochs):

global gradLossInfo

gradLossInfo = []

gradLossInfo.append([])

gradLossInfo.append([])

learn\_rate = 0.1

if infoData != None:

infoData.append([])

infoData.append([])

for epoch in range(epochs):

for x, y\_true in zip(data, all\_y\_trues):

sumHiddens = []

sumSumHiddens = []

for i in range(6):

sumH = 0

for j in range(i\*6,(i+1)\*6):

sumH += self.firstWeights[j]\*x[j]

sumSumHiddens.append(sumH)

sumHiddens.append(sigmoid(sumH))

sumO = 0

for i in range(6):

sumO += self.secondWeights[i]\*sumHiddens[i]

o1 = sigmoid(sumO)

y\_pred = o1

d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)

d\_ypred\_d\_wH = []

for i in range(6):

d\_ypred\_d\_wH.append(sumHiddens[i]\*derivSigmoid(sumO))

d\_ypred\_d\_hH = []

for i in range(6):

d\_ypred\_d\_hH.append(self.secondWeights[i]\*derivSigmoid(sumO))

d\_h1\_d\_w = []

currentH = 0

for i in range(36):

d\_h1\_d\_w.append(x[i]\*derivSigmoid(sumSumHiddens[currentH]))

if i !=0 and i%6==0:

currentH += 1

# Обновление весов

currentH = 0

for i in range(36):

self.firstWeights[i] -= learn\_rate \* d\_L\_d\_ypred \* d\_h1\_d\_w[i] \* d\_ypred\_d\_hH[currentH]

if i !=0 and i%6==0:

currentH += 1

for i in range(6):

self.secondWeights[i] -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_wH[i]

y\_pred = []

for x, y\_true in zip(data, all\_y\_trues):

y\_pred.append(round(self.feedforward(x)))

correctCount = 0

for i in range(len(y\_pred)):

if y\_pred[i] == all\_y\_trues[i]:

correctCount+=1

if infoData != None:

infoData[0].append(epoch)

infoData[1].append(correctCount/len(y\_pred))

y\_preds = np.apply\_along\_axis(self.feedforward, 1, data)

loss = mseLoss(all\_y\_trues, y\_preds)

gradLossInfo[0].append(epoch)

gradLossInfo[1].append(loss)

print("Epoch %d loss: %.3f" % (epoch, loss))

def hebbSigmoid(x):

y = 1 / (1 + np.exp(-x))

if y > 0.5:

return 1

else:

return 0

def findDeltaW(x, y):

if x\*y != 0:

return random.random()

if x==0:

return 0

return -1\*random.random()

class HebbNeuralNetwork:

def \_\_init\_\_(self):

self.firstWeights = [0.0]\*36

self.secondWeights = [0.0]\*6

def feedforward(self, x, isSigmoid):

hiddenNeurons = []

for i in range(6):

hiddenSum = 0

for j in range(i\*6,(i+1)\*6):

hiddenSum += self.firstWeights[j] \* x[j]

hiddenNeurons.append(hebbSigmoid(hiddenSum))

sumOutput = 0

for i in range(6):

sumOutput += self.secondWeights[i] \* hiddenNeurons[i]

if isSigmoid:

return hebbSigmoid(sumOutput)

else:

return 1 / (1 + np.exp(-sumOutput))

def train(self, data, all\_y\_trues, infoData, epochs):

global hebbLossInfo

hebbLossInfo = []

hebbLossInfo.append([])

hebbLossInfo.append([])

if infoData != None:

infoData.append([])

infoData.append([])

for epoch in range(epochs):

isEnd = False

eps = 1

for x, y\_true in zip(data, all\_y\_trues):

hiddenNeurons = []

for i in range(6):

hiddenSum = 0

for j in range(i\*6,(i+1)\*6):

hiddenSum += self.firstWeights[j] \* x[j]

hiddenNeurons.append(hebbSigmoid(hiddenSum))

sumOutput = 0

for i in range(6):

sumOutput += self.secondWeights[i] \* hiddenNeurons[i]

output = hebbSigmoid(sumOutput)

if output != y\_true:

for i in range(len(self.firstWeights)):

self.firstWeights[i] += findDeltaW(x[i], y\_true)

for i in range(len(self.secondWeights)):

self.secondWeights[i] += findDeltaW(hiddenNeurons[i], y\_true)

y\_pred = []

for x, y\_true in zip(data, all\_y\_trues):

y\_pred.append(self.feedforward(x, True))

correctCount = 0

for i in range(len(y\_pred)):

if y\_pred[i] == all\_y\_trues[i]:

correctCount+=1

if correctCount/len(y\_pred) >= eps and infoData == None:

isEnd = True

break

if epoch % 100 == 0 and infoData == None:

eps -= 0.01

y\_pred = []

for x, y\_true in zip(data, all\_y\_trues):

y\_pred.append(self.feedforward(x, True))

y\_loss = []

for x, y\_true in zip(data, all\_y\_trues):

y\_loss.append(self.feedforward(x, False))

correctCount = 0

for i in range(len(y\_pred)):

if y\_pred[i] == all\_y\_trues[i]:

correctCount+=1

if infoData != None:

infoData[0].append(epoch)

infoData[1].append(correctCount/len(y\_pred))

hebbLossInfo[0].append(epoch)

hebbLossInfo[1].append(mseLoss(all\_y\_trues, y\_loss))

if epoch % 100 == 0:

print("ep:"+str(epoch))

if isEnd == True:

print("epochs number:"+str(epoch))

break

def binarize(img):

vectorImg = []

for i in range(len(img)):

for j in range(len(img[0])):

if img[i][j] > 0:

vectorImg.append(0)

else:

vectorImg.append(1)

return vectorImg

gradLossInfo = []

hebbLossInfo = []

hebbInfo = []

gradInfo = []

isNew = True

trainData = []

train\_y\_trues = []

main()