**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 группы ИТП-31

Пронуза М.Ю.

Проверил:

Башаримов Ю.С.

Гомель 2024

**Цель работы**: изучить особенности разработки в среде *Google Collab,* используя контролируемое обучение нейронных сетей. Разработать программное обеспечение для решения задачи бинарной классификации.

**Задание:**

Разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием простейших нейронных сетей.

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 варианту. 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.

**Ход выполнения и результаты работы:**

На рисунке 3 представлен результат эпох и точность методом Уидроу-Хоффа.

Final Accuracy (Direct Propagation): 0.9950

Рисунок 3 – Результат работы метода Уидроу-Хоффа

На рисунке 4 представлен результат эпох и потери метода обратного распространения ошибки.

Final Accuracy (Backpropagation): 0.9950

Рисунок 4 – Результат работы метода обратного распространения ошибки

На рисунке 5 представлен результат сравнения работы методов Уидроу-Хоффа и обратного распространения ошибки.

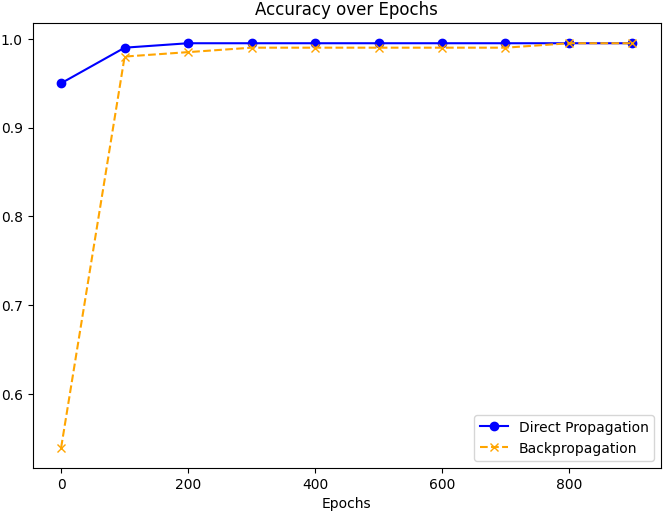


Рисунок 5 – Сравнение результатов

На рисунке 6 представлен результат обучения нейронной сети.

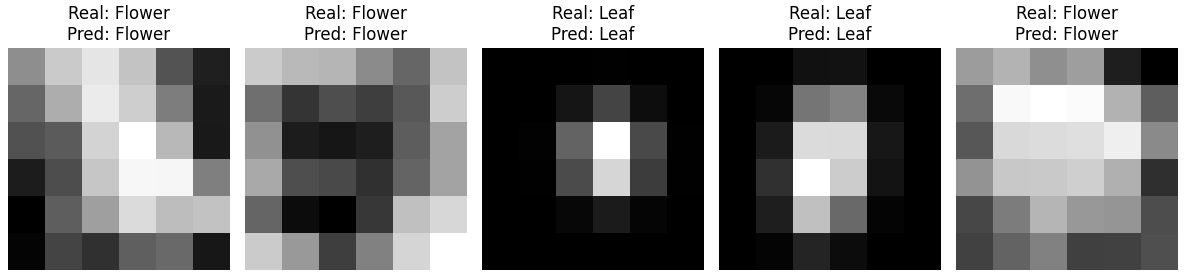


Рисунок 6 – результат обучения нейронной сети

Текст обработки входных данных представлен в приложении А.

Текст программы методом Уидроу-Хоффа представлен в приложении Б.

Текст программы методом обратного распространения ошибки представлен в приложении В.

Текст графиков сравнения представлен в приложении Г.

**Вывод:** в ходе выполнения лабораторной работы изучены особенности разработки в среде *Google Collab*, используя контролируемое обучение нейронных сетей. В результате выполнения лабораторной работы разработана программа для вычисления значений методом Уидроу-Хоффа.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст обработки входных данных**

import requests

import zipfile

import io

import os

from skimage import color, transform

import numpy as np

import matplotlib.pyplot as plt

import tensorflow\_datasets as tfds

url = 'https://archive.ics.uci.edu/static/public/288/leaf.zip'

response = requests.get(url)

if response.status\_code == 200:

with zipfile.ZipFile(io.BytesIO(response.content)) as z:

z.extractall('leaf\_dataset')

leaf\_image\_dir = 'leaf\_dataset/BW'

processed\_leaf\_images = []

target\_size = (6, 6)

max\_images = 100

image\_count = 0

for root, dirs, files in os.walk(leaf\_image\_dir):

for filename in files:

if filename.endswith('.TIFF'):

image\_path = os.path.join(root, filename)

image = plt.imread(image\_path)

if image.shape[2] == 4:

image = image[:, :, :3]

if len(image.shape) == 3:

gray\_image = color.rgb2gray(image)

else:

gray\_image = image

resized\_image = transform.resize(gray\_image, target\_size, anti\_aliasing=True)

processed\_leaf\_images.append(resized\_image)

image\_count += 1

if image\_count >= max\_images:

break

if image\_count >= max\_images:

break

processed\_leaf\_images = np.array(processed\_leaf\_images)

flowers\_dataset, flowers\_info = tfds.load('oxford\_flowers102', split=f'train[:{max\_images}]', with\_info=True)

processed\_flower\_images = []

for example in tfds.as\_numpy(flowers\_dataset):

image = example['image']

gray\_image = color.rgb2gray(image)

resized\_image = transform.resize(gray\_image, target\_size, anti\_aliasing=True)

processed\_flower\_images.append(resized\_image)

processed\_flower\_images = np.array(processed\_flower\_images)

def show\_images(images, n=10):

plt.figure(figsize=(10, 2))

for i in range(min(n, len(images))):

ax = plt.subplot(1, n, i + 1)

plt.imshow(images[i], cmap="gray")

plt.axis("off")

plt.show()

show\_images(processed\_leaf\_images, n=10)

show\_images(processed\_flower\_images, n=10)

print("Количество обработанных изображений листьев:", processed\_leaf\_images.shape)

print("Количество обработанных изображений цветов:", processed\_flower\_images.shape)

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст метода Уидроу-Хоффа**

import numpy as np

import matplotlib.pyplot as plt

normalized\_leaf\_images = (processed\_leaf\_images - processed\_leaf\_images.min()) / (processed\_leaf\_images.max() - processed\_leaf\_images.min())

normalized\_flower\_images = (processed\_flower\_images - processed\_flower\_images.min()) / (processed\_flower\_images.max() - processed\_flower\_images.min())

normalized\_leaf\_images = normalized\_leaf\_images.reshape(len(normalized\_leaf\_images), -1)

normalized\_flower\_images = normalized\_flower\_images.reshape(len(normalized\_flower\_images), -1)

leaf\_labels = np.zeros(normalized\_leaf\_images.shape[0])

flower\_labels = np.ones(normalized\_flower\_images.shape[0])

X = np.vstack((normalized\_leaf\_images, normalized\_flower\_images))

y = np.hstack((leaf\_labels, flower\_labels))

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

y = y[indices]

input\_size = X.shape[1]

output\_size = 1

np.random.seed(42)

weights = np.random.uniform(-0.1, 0.1, input\_size)

bias = np.random.uniform(-0.1, 0.1)

learning\_rate = 0.01

epochs = 1000

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

error\_list = []

accuracy\_list = []

backprop\_error\_list = []

backprop\_accuracy\_list = []

for epoch in range(epochs):

total\_error = 0

for i in range(X.shape[0]):

net\_input = np.dot(X[i], weights) + bias

output = net\_input

error = y[i] - output

total\_error += error \*\* 2

weights += learning\_rate \* error \* X[i]

bias += learning\_rate \* error

predictions = (np.dot(X, weights) + bias > 0.5).astype(int)

accuracy = np.mean(predictions == y)

if epoch % 100 == 0:

print(f"Epoch {epoch}, Total Error: {total\_error:.4f}")

error\_list.append(total\_error)

accuracy\_list.append(accuracy)

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Текст метода обратного распространения ошибки**

weights\_bp = np.random.uniform(-0.1, 0.1, input\_size)

bias\_bp = np.random.uniform(-0.1, 0.1)

for epoch in range(epochs):

total\_error = 0

for i in range(X.shape[0]):

net\_input = np.dot(X[i], weights\_bp) + bias\_bp

output = sigmoid(net\_input)

error = y[i] - output

total\_error += error \*\* 2

output\_derivative = sigmoid\_derivative(output)

weight\_gradient = error \* output\_derivative \* X[i]

bias\_gradient = error \* output\_derivative

weights\_bp += learning\_rate \* weight\_gradient

bias\_bp += learning\_rate \* bias\_gradient

predictions = (sigmoid(np.dot(X, weights\_bp) + bias\_bp) > 0.5).astype(int)

accuracy = np.mean(predictions == y)

if epoch % 100 == 0:

print(f"Epoch {epoch}, Total Error: {total\_error:.4f}")

backprop\_error\_list.append(total\_error)

backprop\_accuracy\_list.append(accuracy)

**ПИЛОЖЕНИЕ Г**

(обязательное)

**Текст графиков сравнения методов**

plt.figure(figsize=(14, 10))

plt.subplot(2, 2, 1)

plt.plot(range(0, epochs, 100), error\_list, marker='o', linestyle='-', color='r', label='Direct Propagation')

plt.plot(range(0, epochs, 100), backprop\_error\_list, marker='x', linestyle='--', color='g', label='Backpropagation')

plt.title('Total Error over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Total Error')

plt.legend()

plt.subplot(2, 2, 2)

plt.plot(range(0, epochs, 100), accuracy\_list, marker='o', linestyle='-', color='b', label='Direct Propagation')

plt.plot(range(0, epochs, 100), backprop\_accuracy\_list, marker='x', linestyle='--', color='orange', label='Backpropagation')

plt.title('Accuracy over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

print(f"Final Accuracy (Direct Propagation): {accuracy\_list[-1]:.4f}")

print(f"Final Accuracy (Backpropagation): {backprop\_accuracy\_list[-1]:.4f}")