Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №5

“ Работа с датасетом”

Выполнил: Студент 1 курса

Факультета Цифровых

Промышленных Технологий

группы 20221 Хохлов Д.Р.

Преподаватель:

Кафедра киберфизических систем

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

**СОДЕРЖАНИЕ**

[Введение 2](#_Toc186202974)

[1.Практическая часть 3](#_Toc186202975)

[1.1 Постановка задачи 3](#_Toc186202976)

[1.2 Задача по выводу цифр из датасета по номеру варианта 3](#_Toc186202977)

[1.3 Результаты работы программы по выводу цифр 3](#_Toc186202978)

[1.4 Задача обучения нейронной сети на определение последней цифры варианта с помощью датасета MNIST 4](#_Toc186202979)

[1.5 Результаты обучения нейронной сети 4](#_Toc186202980)

[Приложение А 6](#_Toc186202981)

# Введение

Цель работы

Целью данной лабораторной работы является приобретение навыков работы с датасетами, извлечение данных и обучение нейронных сетей с использованием алгоритма обратного распространения ошибки.

Задача

Задача данной работы включает анализ кода, представленного на лекции, заполнение комментариев и написание программы, выводящей пять вариантов номера группы в виде цифр из датасета MNIST. Также необходимо реализовать обучение нейронной сети для распознавания последней цифры номера группы без использования специализированных библиотек.

# 1.Практическая часть

## 1.1 Постановка задачи

Задача данной лабораторной работы заключается в нескольких поставленных задачах

## 1.2 Задача по выводу цифр из датасета по номеру варианта

Для выполнения задачи вывода цифры варианта из датасета MNIST необходимо:

1. Импортировать или скачать датасет MNIST.
2. Отделить массив с изображениями от массива с подписями.
3. Отфильтровать необходимые цифры (в данном случае 1 и 8) из датасета MNIST и получить их индексы из массива с подписями.
4. Выбрать из массива с изображениями нужные изображения по ранее полученным индексам.
5. Изменить размер массива изображения с 784х1 на 28х28.
6. С использованием библиотеки Matplotlib вывести цифры на экран.

## 1.3 Результаты работы программы по выводу цифр

После реализации программы получены несколько изображений, на которых чётко видно число варианта (Рисунок 1).

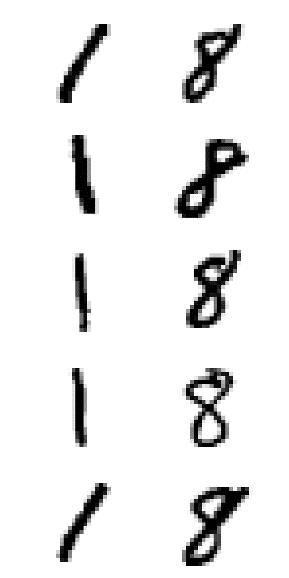


Рисунок 1 – Результат работы программы

по выводу номера варианта

## 1.4 Задача обучения нейронной сети на определение последней цифры варианта с помощью датасета MNIST

Для реализации обучения нейронной сети в данной задаче используются методы, описанные в Практике №4, где был реализован алгоритм обратного распространения ошибки и процесса обучения. Для выполнения задачи требуется:

1. Импортировать датасет MNIST.
2. Отделить массив с изображениями от массива с подписями.
3. Заменить все цифры, кроме требуемой, на нули, а требуемую цифру — на 1.
4. Обучить нейронную сеть на данных. Значения каждого пикселя в изображении являются входными значениями, а выходные значения — 0 или 1.
5. Вывести полученные результаты на экран.

Также необходимо изменить функцию обучения: вместо бесконечного цикла задать определённое количество эпох (эпохи — это количество итераций обучения).

## 1.5 Результаты обучения нейронной сети

Результатом обучения является нейронная сеть, правильно определяющая цифры варианта. После завершения последней эпохи обучения получены данные функции потерь для обучающей и тестовой выборок. (Рисунок 2)

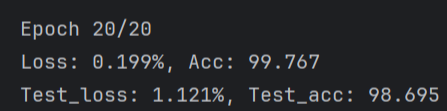


Рисунок 2 – Данные из функции потерь

Через функцию получения параметров каждого слоя можно просмотреть данные весов каждого слоя. (Рисунок 3)

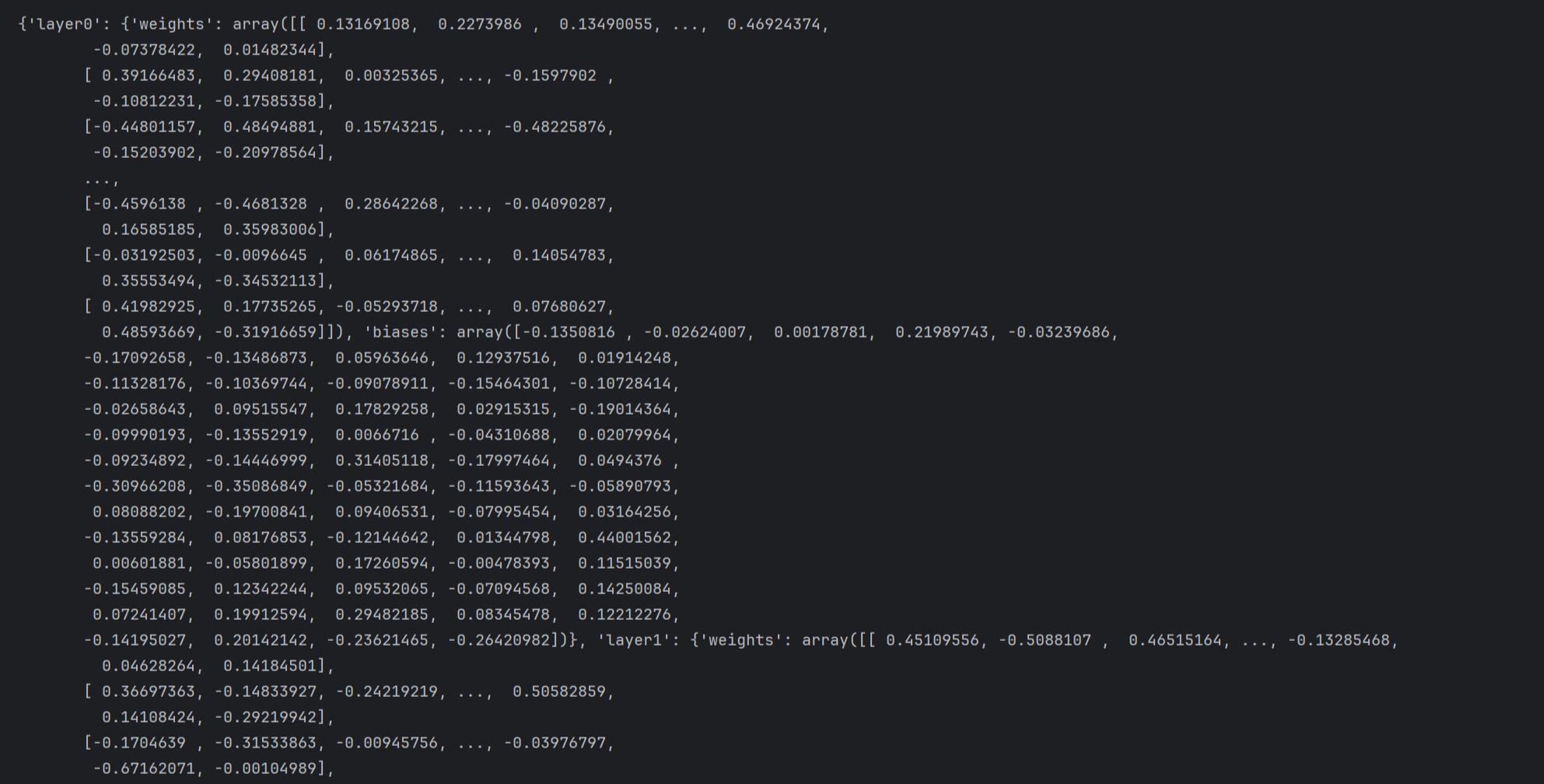
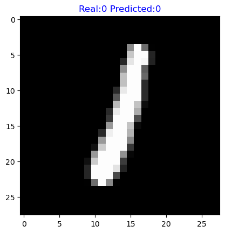
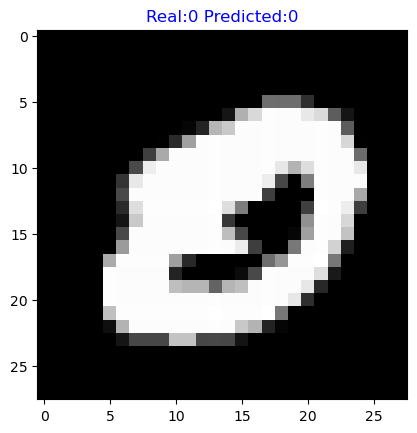
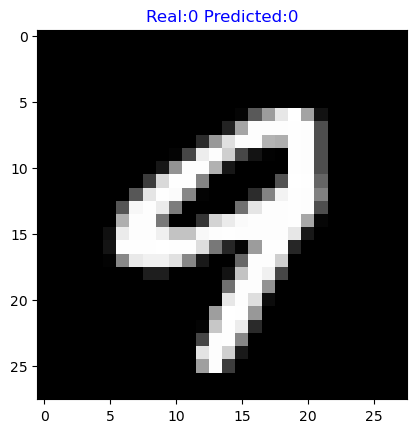
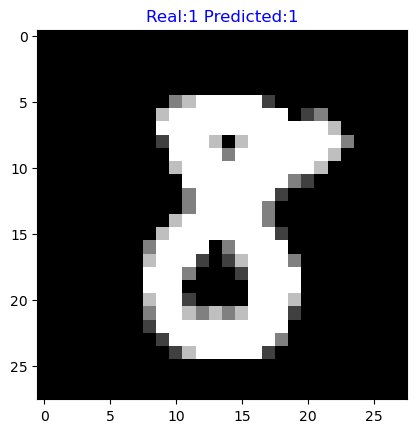
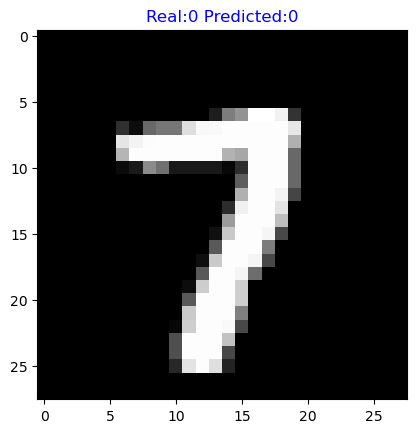


Рисунок 3 – Веса после обучения

После достижения удовлетворительных показателей функции потерь и оптимального подбора слоёв и нейронов осуществляется проверка нейронной сети. Берутся последние 10 цифр из набора MNIST для тестирования (Рисунок 2).

**Важно**: для тестирования используются те данные, на которых нейронная сеть не обучалась.



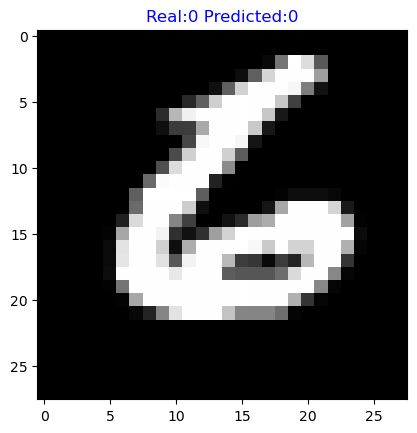
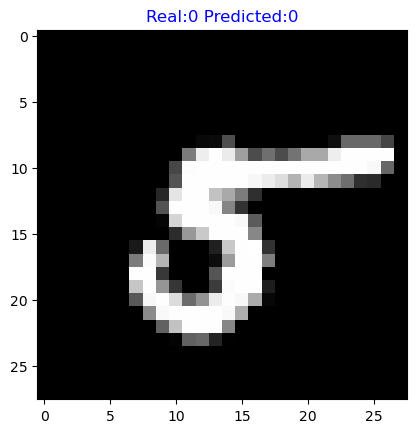
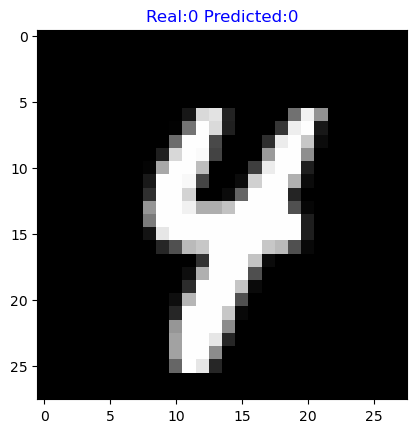
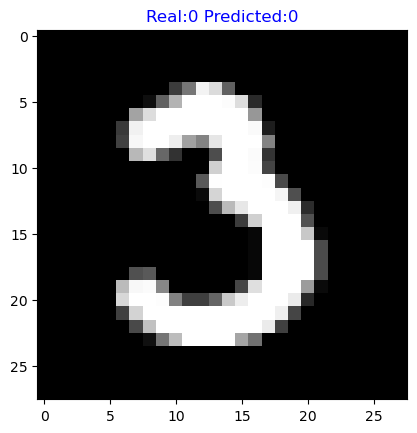
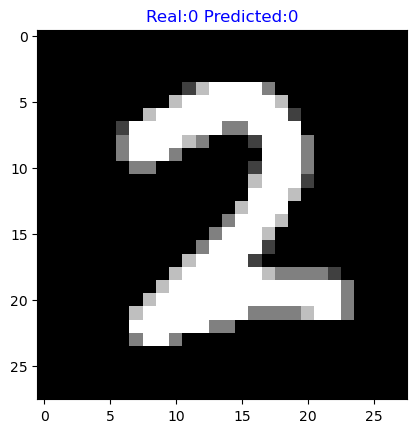


Рисунок 4 – Проверка работоспособности нейросети

Приложение А

**Листинг программы**

Листинг файла «print\_num.py» для задачи с комментированием кода

from sklearn.datasets import fetch\_openml  
import numpy as np  
import os  
%matplotlib inline  
import matplotlib as mpl  
import matplotlib.pyplot as plt  
#%%  
# Рисуем симпатичные рисунки  
mpl.rc('axes', labelsize=14)  
mpl.rc('xtick', labelsize=12)  
mpl.rc('ytick', labelsize=12)  
#%%  
# Указываем куда сохранять рисунки  
PROJECT\_ROOT\_DIR = "."  
CHAPTER\_ID = "classification"  
IMAGES\_PATH = os.path.join(PROJECT\_ROOT\_DIR, "images", CHAPTER\_ID)  
os.makedirs(IMAGES\_PATH, exist\_ok=True)  
#%%  
#Сохраняем картинки с цифрами  
def save\_fig(fig\_id, tight\_layout=True, fig\_extension="png", resolution=300):  
 path = os.path.join(IMAGES\_PATH, fig\_id + "." + fig\_extension)  
 print("Saving figure", fig\_id)  
 if tight\_layout:  
 plt.tight\_layout()  
 plt.savefig(path, format=fig\_extension, dpi=resolution)  
#%%  
#Выводим одну картинку с цифрой  
def plot\_digit(data):  
 image = data.reshape(28, 28)  
 plt.imshow(image, cmap = mpl.cm.binary,  
 interpolation="nearest")  
 plt.axis("off")  
#%%  
# Выводим много цифр…  
def plot\_digits(instances, images\_per\_row=10, \*\*options):  
 size = 28  
 images\_per\_row = min(len(instances), images\_per\_row)  
  
# Эквивалентно n\_rows = ceil(len(instances) / images\_per\_row)  
 n\_rows = (len(instances) - 1) // images\_per\_row + 1  
 # Добавляем пустое изображение, чтобы заполнить конец таблицы  
 n\_empty = n\_rows \* images\_per\_row - len(instances)  
 padded\_instances = np.concatenate([instances, np.zeros((n\_empty,  
size \* size))], axis=0)  
 # Изменяем массив так, чтобы он представлял сетку изображений 28×28  
 image\_grid = padded\_instances.reshape((n\_rows, images\_per\_row,  
size, size))  
 # Комбинируем оси 0 и 2 (вертик ось сетки и гориз ось  
 # и оси 1 и 3 (гориз оси). Сначала перемещаем оси так,  
 # чтобы они стояли друг за другом. Используем transpose()  
 # Только после этого можно изменить форму массива  
 big\_image = image\_grid.transpose(0, 2, 1, 3).reshape(n\_rows \* size,  
images\_per\_row \* size)  
 # Теперь мы получили большую картинку, которую не стыдно показать:  
 plt.imshow(big\_image, cmap = mpl.cm.binary, \*\*options)  
 plt.axis("off")  
  
#Извлекаем датасет "Mnist" из хранилища OpenML  
mnist = fetch\_openml('mnist\_784', version=1, as\_frame=False)  
mnist.keys()  
  
#Распаковываем картинки(массив с цветом каждого пикселя) и подписи к ним  
X, y = mnist["data"], mnist["target"]  
#Выводим размер массива с картинками  
X.shape  
#%%  
#Выводим размер массива с подписями  
y.shape  
#%%  
some\_digit = X[0] #Берём из массива с картинками первую картинку  
some\_digit\_image = some\_digit.reshape(28, 28) #меняем размер массива с выбранной картинкой (Из (756,1) в (28,28)) для дальнейшего удобного вывода  
plt.imshow(some\_digit\_image, cmap=mpl.cm.binary) #Создаём "график" с выбранной картинкой и делаем её в формате   
plt.axis("off") #Выключаем отображение осей  
#%%  
save\_fig("some\_digit\_plot") #Сохраняем выбранную картинку  
plt.show() #Выводим выбранную картинку

**Листинг файла «my\_num.py» для вывода номера варианта**

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib as mpl  
from sklearn.datasets import fetch\_openml  
  
  
mnist = fetch\_openml('mnist\_784', version=1, as\_frame=False)  
X, y = mnist["data"], mnist["target"]  
y = y.astype(np.uint8)  
  
  
ones\_indices = np.where(y == 1)[0]  
eights\_indices = np.where(y == 8)[0]  
  
  
  
def plot\_18(one\_idx, eight\_idx, position):  
 plt.subplot(5, 2, position \* 2 - 1)  
 plt.imshow(X[one\_idx].reshape(28, 28), cmap=mpl.cm.binary)  
 plt.axis('off')  
  
 plt.subplot(5, 2, position \* 2)  
 plt.imshow(X[eight\_idx].reshape(28, 28), cmap=mpl.cm.binary)  
 plt.axis('off')  
  
  
  
plt.figure(figsize=(10, 20))  
for i in range(5):  
 plot\_18(ones\_indices[i], eights\_indices[i], i + 1)  
  
plt.tight\_layout()  
plt.show()

**Листинг файла «nn.py» c обучением нейросети**

#!pip install numpy  
#!pip install prettytable  
#!pip install scikit-learn  
#%%  
import random  
from abc import ABC, abstractmethod  
  
import numpy as np  
  
from sklearn.datasets import fetch\_openml  
import os  
  
import matplotlib.pyplot as plt  
%matplotlib inline  
#%%  
class Activation:  
 @staticmethod  
 def SIGMOID():  
 def activation(x):  
 return 1 / (1 + np.exp(-x))  
 def derivative(x):  
 return x \* (1 - x)  
 return activation, derivative  
   
 @staticmethod  
 def TANH():  
 def activation(x):  
 return np.tanh(x)  
 def derivative(x):  
 return 1 - x \*\* 2  
 return activation, derivative  
  
#%%  
class Layer:  
 def \_\_init\_\_(self, input\_size, output\_size, activation\_func):  
 self.activate, self.derivative = activation\_func()  
 self.weights = np.random.uniform(-0.5,0.5,size=(input\_size, output\_size))  
 self.biases = np.zeros(output\_size)   
 def \_\_call\_\_(self):  
 return {  
 'weights':self.weights,   
 'biases':self.biases  
 }  
#%%  
class NN(ABC):  
 def \_\_init\_\_(self, layers):  
 self.layers = layers  
 self.params = self.\_\_params()  
   
 def \_\_params(self):  
 params = {}  
 for i, layer in enumerate(layers):  
 params[f'layer{i}'] = {"weights":layer.weights,  
 "biases":layer.biases}  
 return params  
 def \_forward(self, x):  
 for layer in self.layers:  
 z = np.dot(x, layer.weights) + layer.biases  
 x = layer.activate(z)  
 layer.output = np.array([x])  
 return x  
   
 def \_backward(self, x, y, lr):  
 error = -(y - self.layers[-1].output)  
 for layer in reversed(self.layers):  
 delta = error \* layer.derivative(layer.output)  
 prev\_output = np.array([x]) if layer == self.layers[0] else self.layers[self.layers.index(layer) - 1].output  
   
 layer.weights -= lr \* np.dot(prev\_output.T, delta)  
 layer.biases -= lr \* np.sum(delta, axis=0)  
 error = np.dot(delta, layer.weights.T)  
   
 @abstractmethod  
 def train(self):  
 pass  
   
   
   
 def \_\_call\_\_(self, input\_data):  
 output = []  
 for i in range(input\_data.shape[0]):  
 output.append(1 if self.\_forward(input\_data[i]) >= 0.5 else 0)  
 return output  
   
#%%  
class MnistNN(NN):  
 def \_\_init\_\_(self, layers):  
 super().\_\_init\_\_(layers)  
   
 def \_\_MSE(self, output, p\_out, error, acc):  
 error += np.sum((output - p\_out)\*\*2) #MSE  
 acc += 0 if(((output < 0.5) and (p\_out >= 0.5)) or ((output >= 0.5) and (p\_out < 0.5))) else 1  
 return error, acc  
   
 def train(self, images, labels, epochs, learning\_rate=0.01):  
   
 index\_list = list(range(40000))  
 for i in range(epochs):  
 random.shuffle(index\_list)  
 e\_loss, e\_acc, test\_loss, test\_acc = 0.0, 0, 0.0, 0  
   
 for j in index\_list:   
   
 p\_out = self.\_forward(images[j])  
 e\_loss, e\_acc = self.\_\_MSE(labels[j],p\_out, e\_loss, e\_acc)  
 self.\_backward(images[j], labels[j], learning\_rate)  
   
 for image, label in zip(images[40000:60000], labels[40000:60000]):  
 p\_out = self.\_forward(image)  
 test\_loss, test\_acc = self.\_\_MSE(label, p\_out, test\_loss, test\_acc)  
   
 print(f"Epoch {i+1}/{epochs}\n"  
 f"Loss: {round(e\_loss/40000.0 \* 100, 3)}%, Acc: {round(e\_acc/40000.0 \* 100, 3)}\n"  
 f"Test\_loss: {round(test\_loss/20000.0 \* 100, 3)}%, Test\_acc: {round(test\_acc/20000.0 \* 100, 3)}\n")  
  
#%%  
mnist = fetch\_openml('mnist\_784', version=1, as\_frame=False)   
mnist.keys()   
mnist["target"]  
#%%  
train\_images = mnist["data"].astype('float32') / 255  
train\_labels = mnist["target"].reshape(-1, 1).astype('int32')  
#%%  
LEARNING\_RATE = 0.1  
EPOCH = 20  
#%%  
for i in range(len(train\_labels)):  
 train\_labels[i][0] = 1 if train\_labels[i][0] == 8 else 0  
train\_labels  
#%%  
#~~Структура нейросети~~#  
layers = (  
 Layer(784, 64, Activation.TANH),  
 Layer(64,32,Activation.TANH),  
 Layer(32, 1, Activation.SIGMOID),  
)  
#%%  
model = MnistNN(layers) #создание объекта нейросети  
print(model.params['layer0']['weights']) #вывод параметров  
#%%  
#~~Обучение нейросети~~#  
model.train(train\_images,train\_labels, EPOCH, learning\_rate=LEARNING\_RATE)   
#%%  
print(model.params)  
#%%  
def RGB2URGB(img):  
 gray = lambda rgb: np.dot(rgb[..., :3], [0.299, 0.587, 0.114])  
 return 1-gray(img).astype("float32") / 255  
#%%  
PATH = "C:/Users/madro/Documents/GitHub/Khokhlov\_Dimitrii\_20121\_23/AI/P.5/data/test/"  
#%%  
def show(images, labels):  
 for img, label in zip(images, labels):  
 plt.imshow(np.array([img]).reshape(28,28), cmap = 'grey')  
 predict = model(np.array([img]))[0]  
 color = "red" if predict != label[0] else "blue"  
 plt.title(f"Real:{label[0]} Predicted:{predict}", color=color)  
 plt.show()  
#%%  
train\_labels  
#%%  
test\_img = train\_images[-10:]  
test\_labels = train\_labels[-10:]  
show(test\_img, test\_labels)