Кимбрис Владимир Владимирович, группа 20-1

Лабораторная работа № 1

**Цель работы**

**Задание 1.1**

Осуществить обучение нейрона выполнению функции логического «ИЛИ».

2. Осуществить попытку обучения одного нейрона выполнению функции исключающего «ИЛИ» и посмотреть на результат.

3. Построить и обучить модель для решения функции исключающего «ИЛИ» (XOR) (используйте adam оптимизатор для более быстрого обучения).

4. Построить график активационной функции заданного вида (по вариантам, по номеру в списке).

**Задание 1.2**

Создать и обучить нейронную сеть для распознавания рукописных цифр набора данных MNIST

**Задание**

**1.1**

1. Осуществить обучение нейрона выполнению функции логического «ИЛИ».

2. Осуществить попытку обучения одного нейрона выполнению функции исключающего «ИЛИ» и посмотреть на результат.

3. Построить и обучить модель для решения функции исключающего «ИЛИ» (XOR) (используйте adam оптимизатор для более быстрого обучения).

4. Построить график активационной функции заданного вида (по вариантам, по номеру в списке).

**1.2**

1. Попробуйте протестировать обученную нейронную сеть на ваших личных цифрах нарисованных, например, в Paint (на чёрном фоне, белым цветом).
2. Попробуйте подать сети цифру, немного сдвинутую вправо или влево от центра, сможет ли сеть распознать вашу цифру?
3. Попробуйте подать сети перевёрнутую цифру, сможет ли сеть распознать вашу цифру?

**Код программы / ответ на вопрос и т.п.**

**Задание 1.1**

# Подключение необходимых библиотек

import numpy as np

import matplotlib.pyplot as plt

import torch

import torch.nn as nn

import torch.nn.functional as F

from torch.optim import Adam

from tqdm import tqdm

# Функция для отрисовки графиков

def plot\_decision\_boundary(model, x, y):

  # Задаем цвета для точек в зависимости от их класса

  colors = ["red" if y[i] == 0 else "blue" for i in range(4)]

    # Рисуем точки на графике

  plt.scatter(x[:, 0], x[:, 1], c=colors)

    # Задаем диапазон значений для осей x и y

  x\_min, x\_max = x[:, 0].min() - 0.1, x[:, 0].max() + 0.1

  y\_min, y\_max = x[:, 1].min() - 0.1, x[:, 1].max() + 0.1

    # Создаем сетку из 100 точек по каждой оси

  xx, yy = torch.meshgrid(torch.linspace(x\_min, x\_max, 100), torch.linspace(y\_min, y\_max, 100))

    # Вычисляем выход модели для каждой точки сетки

  zz = model(torch.cat([xx.reshape(-1, 1), yy.reshape(-1, 1)], dim=1)).reshape(100, 100)

    # Рисуем контурную карту выхода модели

  plt.contourf(xx.detach().numpy(), yy.detach().numpy(), zz.detach().numpy(), alpha=0.7, cmap=plt.cm.RdYlBu)

    # Показываем график

  plt.show()

# Создание обучающей выборки для обучения нейронных сетей

x = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32)

y\_or = torch.tensor([[0], [1], [1], [1]], dtype=torch.float32) # OR

y\_xor = torch.tensor([[0], [1], [1], [0]], dtype=torch.float32) # XOR

#Однослойная нейронная сеть

class model\_OR(nn.Module):

    def \_\_init\_\_(self):

        super(model\_OR, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 1)

    def forward(self, x):

        # Преобразуем входной тензор в одномерный вектор

        x = F.sigmoid(self.fc1(x))

        return x

#Однослойная нейронная сеть

class model\_XOR\_wrong(nn.Module):

    def \_\_init\_\_(self):

        super(model\_XOR\_wrong, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 1)

    def forward(self, x):

        # Преобразуем входной тензор в одномерный вектор

        x = F.sigmoid(self.fc1(x))

        return x

#Многослойная нейронная сеть с функцией активации нейронная сеть

class model\_XOR\_right(nn.Module):

    def \_\_init\_\_(self):

        super(model\_XOR\_right, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 2)

        self.fc2 = nn.Linear(2, 1)

    def forward(self, x):

        # Преобразуем входной тензор в одномерный вектор

        x = F.sigmoid(self.fc1(x))

        x = F.sigmoid(self.fc2(x))

        return x

# Параметры модели

model\_or = model\_OR()

optimizer = Adam(params=model\_or.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

outputs = model\_or(x)

loss = loss\_func(outputs, y\_or)

loss.backward()

losses.append(loss.item())

optimizer.step()

optimizer.zero\_grad()

# Параметры модели

model\_xor1 = model\_XOR1()

optimizer\_xor1 = Adam(params=model\_xor1.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

outputs = model\_xor1(x)

loss = loss\_func(outputs, y\_xor)

loss.backward()

losses.append(loss.item())

optimizer\_xor1.step()

optimizer\_xor1.zero\_grad()

# Параметры модели

model\_xor2 = model\_XOR2()

optimizer\_xor2 = Adam(params=model\_xor2.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

outputs = model\_xor2(x)

loss = loss\_func(outputs, y\_xor)

loss.backward()

losses.append(loss.item())

optimizer\_xor2.step()

optimizer\_xor2.zero\_grad()

# Функции активации

def sigmoid(input):

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

def tanh(input):

return (np.exp(2\*input) - 1) / (np.exp(2\*input) + 1)

def relu(input):

return (max(0, input))

**Задание 2.2**

import torch

import torch.nn as nn

from torch.optim import Adam

from torch.utils.data import Dataset, DataLoader

from torchvision import datasets

from torchvision import transforms

import matplotlib.pyplot as plt

import torch.nn.functional as F

from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print(f"Using {device} device")

Using cuda device

# Скачивание и нормализация датасета

transform = transforms.Compose([transforms.ToTensor(),

                                transforms.Normalize((0.5,), (0.5,))])

train\_ds = datasets.MNIST(

    root="MNIST\_data",

    train=True,

    download=True,

    transform=transform,

)

test\_ds = datasets.MNIST(

    root="MNIST\_data",

    train=False,

    download=True,

    transform=transform,

)

# Создание и обучение нейронной сети

class NN(nn.Module):

    def \_\_init\_\_(self):

        super(NN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(28\*28, 256)

        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):

        # Преобразуем входной тензор в одномерный вектор

        x = x.view(-1, 28 \* 28)

        x = F.relu(self.fc1(x))

        x = F.softmax(self.fc2(x), dim=1)

        return x

model = NN()

model.to(device)

optimizer = Adam(params=model.parameters(), lr=1e-3)

loss\_func = nn.CrossEntropyLoss()

num\_iter = 10

losses = []

for i in tqdm(range(num\_iter)):

    for data, label in train\_dataloader:

        data = data.to(device)

        label = label.to(device)

        outputs = model(data)

        loss = loss\_func(outputs, label)

        loss.backward()

        losses.append(loss.item())

        optimizer.step()

        optimizer.zero\_grad()

**Результаты выполнения**

**Задание 1.1**

print("OR")

print("x\ty\toutput")

for i in range(4):

print(f"{x[i].detach().numpy()}\t{y\_or[i].detach().numpy()}\t{round(model\_or(x[i]).item())}")

OR

x y output

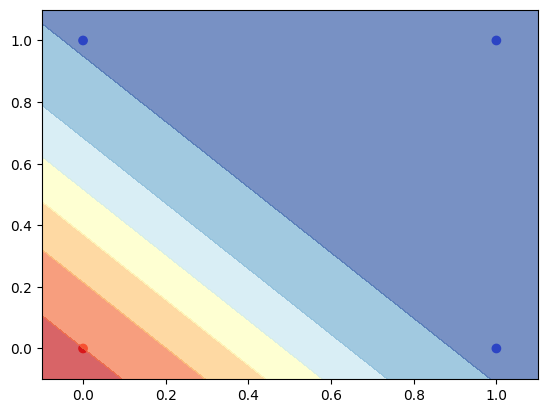
[0. 0.] [0.] 0

[0. 1.] [1.] 1

[1. 0.] [1.] 1

[1. 1.] [1.] 1

plot\_decision\_boundary(model\_or, x, y\_or)



print("XOR")

print("x\ty\toutput")

for i in range(4):

print(f"{x[i].detach().numpy()}\t{y\_xor[i].detach().numpy()}\t{round(model\_xor1(x[i]).item())}")

XOR

x y output

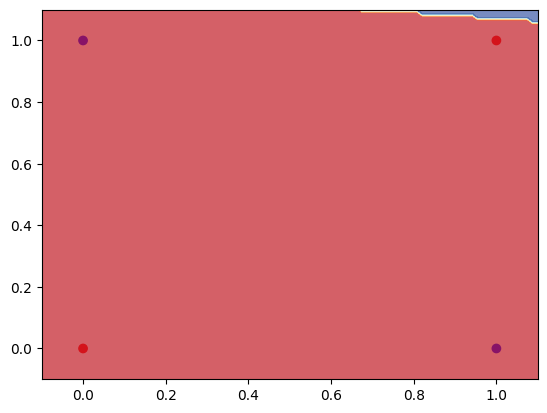
[0. 0.] [0.] 0

[0. 1.] [1.] 0

[1. 0.] [1.] 0

[1. 1.] [0.] 0

plot\_decision\_boundary(model\_xor1, x, y\_xor)



print("XOR")

print("x\ty\toutput")

for i in range(4):

print(f"{x[i].detach().numpy()}\t{y\_xor[i].detach().numpy()}\t{round(model\_xor2(x[i]).item())}")

XOR

x y output

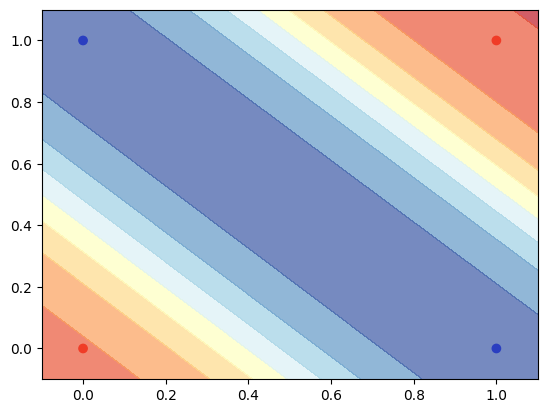
[0. 0.] [0.] 0

[0. 1.] [1.] 1

[1. 0.] [1.] 1

[1. 1.] [0.] 0

plot\_decision\_boundary(model\_xor2, x, y\_xor)



# Вывод на экран графиков функций активации

act\_x = np.linspace(-10, 10, 101)

sigmoid\_y = [sigmoid(x) for x in act\_x]

tanh\_y = [tanh(x) for x in act\_x]

relu\_y = [relu(x) for x in act\_x]

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.plot(act\_x, sigmoid\_y)

plt.title("Sigmoid")

plt.subplot(1, 3, 2)

plt.plot(act\_x, tanh\_y)

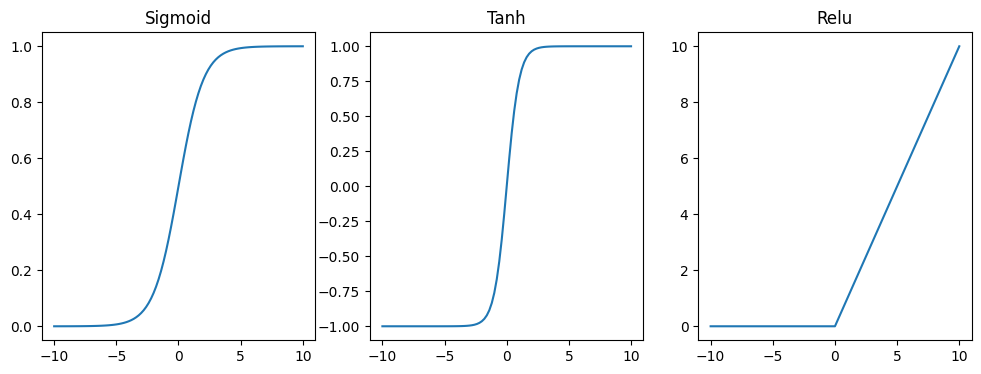
plt.title("Tanh")

plt.subplot(1, 3, 3)

plt.plot(act\_x, relu\_y)

plt.title("Relu")

plt.show()



**Задание 1.2**

# Model test

test\_features, test\_labels = next(iter(test\_dataloader))

img = test\_features[0].squeeze()

label = test\_labels[0]

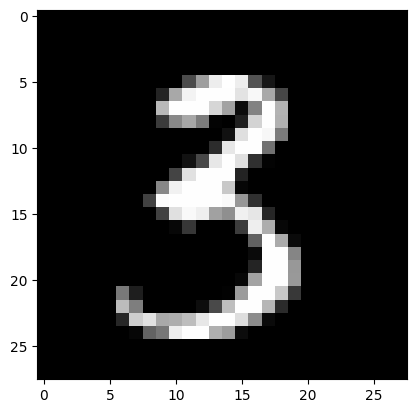
pred = model(test\_features[0].to(device)).argmax()

plt.imshow(img, cmap="gray")

plt.show()

print(f"Label: {label}")

print(f"Pred: {pred}")



Label: 3

Pred: 3

# Accuracy

correct = 0

total = len(test\_dataloader)

for data, label in test\_dataloader:

    data = data.to(device)

    output = model(data).argmax().cpu()

    if (output == label):

        correct += 1

print('Accuracy of the network on the test images: %d %%' % (100 \* correct / total))

Accuracy of the network on the test images: 96 %

# Open your image from file

from PIL import Image

img\_tensors = []

for i in range(1, 4):

    path = f"Image{i}.jpg"

    img = Image.open(path)

    img\_tensors.append(transforms.ToTensor()(img))

for i in range(3):

    print(f"Predicted value = {model(img\_tensors[i][0].to(device)).argmax()}")

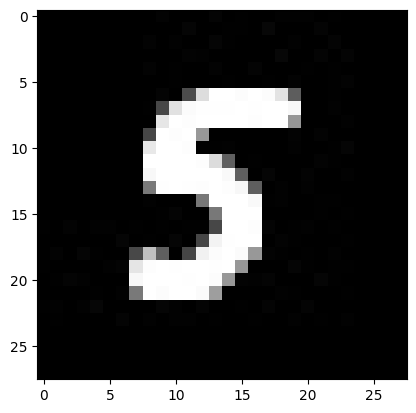
    print(f"Your image:")

    plt.imshow(img\_tensors[i][0].detach().numpy(), cmap="gray")

    plt.show()

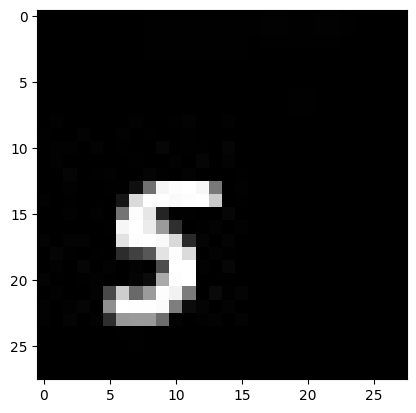
Predicted value = 5

Your image:



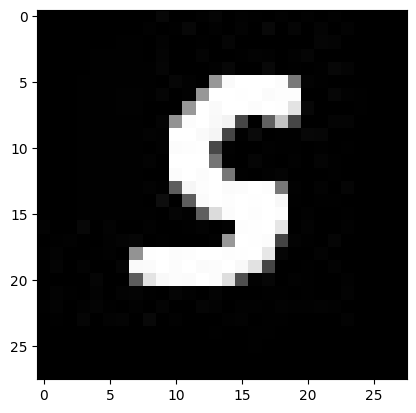
Predicted value = 2

Your image:



Predicted value = 2

Your image:



**Выводы**

В ходе выполнения работы была построена модель нейрона для решения логических операций, а также была решена проблема исключающего ИЛИ.  
Была построена модель нейросети для распознавания рукописных цифр. В качестве обучающей и тестовой выборки использовался набор данных MNIST. Точность полученной модели составила 96%. Также мы убедились, что данная модель не способна распознавать цифры, которые написаны в углу изображения, либо перевёрнуты, так как данная модель распознаёт не отдельные признаки, а изображение целиком.