Ивонин Даниил Александрович, группа 20-1

Лабораторная работа № 1

**Задание 1.1.**

**Цель работы**

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. 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. Создать Dense слой z1 с двумя или более нейронами;
2. Применить к выходным сигналам z1 нелинейную функцию активации act1;
3. Создать выходной Dense слой z2 с одним нейроном, принимающий на вход act1;
4. Применить к выходному сигналу z2 функцию активацию Sigmoid для получения вероятности.

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

# 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 # loop vizualization

# 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

# Нейронная сеть для визуализации проблемы логического ИЛИ

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

# 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()

# Создание модели для решения задачи исключающего ИЛИ

class model\_XOR1(nn.Module):

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

super(model\_XOR1, 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

# 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()

# Нейронная сеть для решения задачи логического ИЛИ

class model\_XOR2(nn.Module):

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

super(model\_XOR2, 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\_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))

**Результаты выполнения**

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

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

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

# 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)

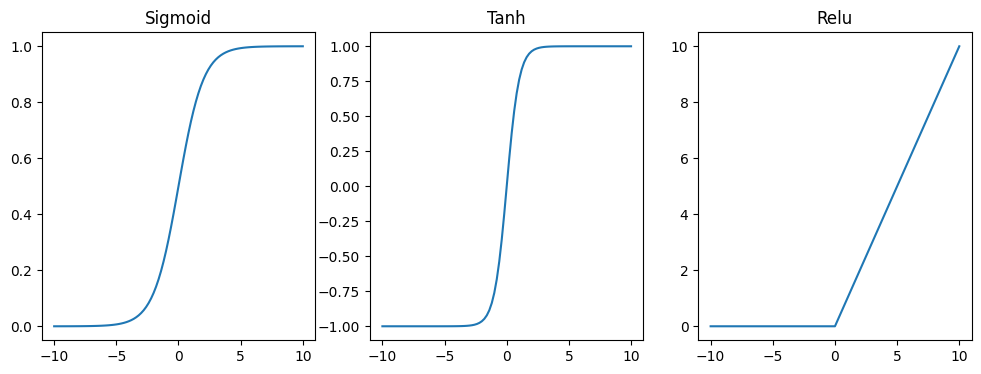
plt.title("Tanh")

plt.subplot(1, 3, 3)

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

plt.title("Relu")

plt.show()



**Выводы**

В результате выполнения работы, я изучил и научился решать проблему логического ИЛИ. Построил и обучил 3 модели нейроннных сетей, а также увидел графики функций активации*.*

**Задание 1.2.**

**Цель работы**

Создать и обучить нейронную сеть для распознавания рукописных цифр.

**Задание**

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.**

**# Подключение необходимых библиотек**

**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 # loop vizualization**

**# Переключение работы нейросети на GPU, если это возможно**

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

**print(f"Using {device} device")**

**# Нормализация изображений и превращение в тензор**

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

**transforms.Normalize((0.1307,), (0.3081,))])**

**# Загрузка обучающей и тестовой выборки**

**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 даталоадеров**

**train\_dataloader = DataLoader(train\_ds, batch\_size=64, shuffle=True, num\_workers=4)**

**test\_dataloader = DataLoader(test\_ds, batch\_size=1, shuffle=True, num\_workers=4)**

**# Визуализация изображений и описаний**

**train\_features, train\_labels = next(iter(train\_dataloader))**

**print(f"Feature batch shape: {train\_features.size()}")**

**print(f"Labels batch shape: {train\_labels.size()}")**

**img = train\_features[0].squeeze()**

**label = train\_labels[0]**

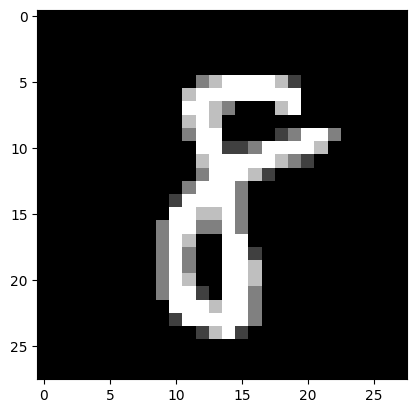
**plt.imshow(img, cmap="gray")**

**plt.show()**

**print(f"Label: {label}")**

**Feature batch shape: torch.Size([64, 1, 28, 28])**

**Labels batch shape: torch.Size([64])**

****

**Label: 8**

**# Модель нейросети для распознавания рукописных цифр**

**class NN(nn.Module):**

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

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

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

**self.fc2 = nn.Linear(256, 128)**

**self.fc3 = nn.Linear(128, 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.relu(self.fc2(x))**

**x = F.softmax(self.fc3(x), dim=1)**

**return x**

**# Объявление модели и её параметров обучения**

**model = NN()**

**model.to(device)**

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

**loss\_func = nn.CrossEntropyLoss()**

**# Процесс обучения модели**

**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()**

**100%|██████████| 10/10 [00:42<00:00, 4.24s/it]**

**Результаты выполнения**

**# Тест модели**

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

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

**label = test\_labels[0]**

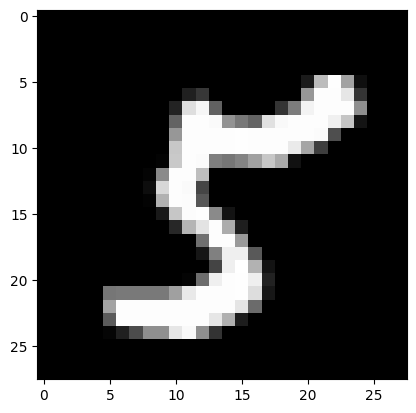
**pred = model(test\_features[0].to(device)).argmax()**

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

**plt.show()**

**print(f"Label: {label}")**

**print(f"Pred: {pred}")**

****

**Label: 5**

**Pred: 5**

**# Подсчёт точности модели**

**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: 97 %**

1. Проверка работы на собственном изображении

**# Проверка работы нейронной сети на собственном изображении**

**from PIL import Image**

**path = "MyImg.jpg"**

**img = Image.open(path)**

**img\_tensor = transforms.ToTensor()(img)**

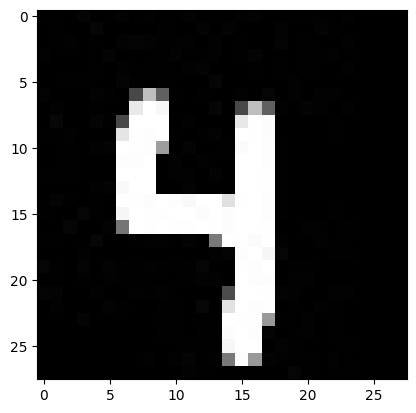
**print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")**

**print("Your image:")**

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

**Predicted value = 4**

**Your image:**

****

1. Проверка работы на собственном изображении, сдвинутом в сторону

**# Проверка работы нейронной сети на сдвинутом изображении**

**from PIL import Image**

**path = "MyImg2.jpg"**

**img = Image.open(path)**

**img\_tensor = transforms.ToTensor()(img)**

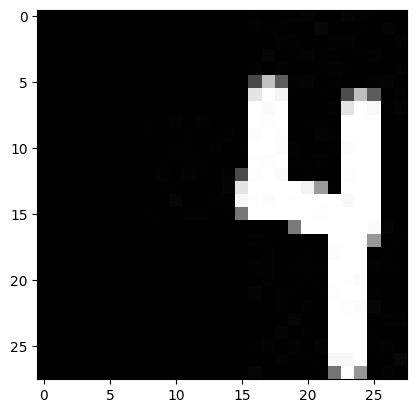
**print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")**

**print("Your image:")**

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

**Predicted value = 7**

**Your image:**

****

1. Проверка работы на собственном перевёрнутом изображении

# Проверка работы нейронной сети на перевёрнутом изображении

from PIL import Image

path = "MyImg3.jpg"

img = Image.open(path)

img\_tensor = transforms.ToTensor()(img)

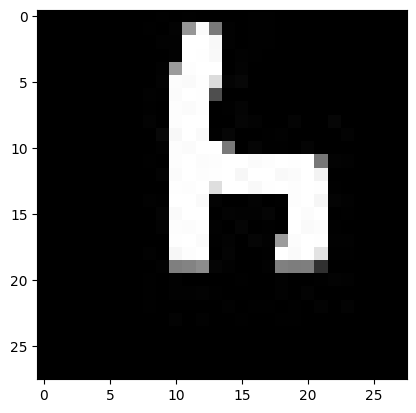
print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")

print("Your image:")

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

Predicted value = 6

Your image:



**Выводы**

В результате выполнения работы, создал и обучил модель нейронной сети, которая способна распознавать рукописные цифры. Точность модели равна 97%. В ходе экспериментов, я убедился, что данная модель не способна распознать сдвинутые или перевёрнутые изображения.