Отчет по лабораторной работе №1 по курсу "Нейроинформатика".

Выполнил Пищик Е.С. М8О-406Б-19.

Цель работы.

Исследование свойств персептрона Розенблатта и его применение для решения задачи распознавания образов.

Файл run.py.

Импортируем нужные библиотеки, класс Perceptron_Pytorch из файла model.py.

In [ ]:

```
import numpy as np
import torch
from torch import nn
from model import Perceptron_Pytorch
from tqdm import tqdm
import matplotlib.pyplot as plt
import os
```

Создаем класс Pipeline, где происходит инициализация гиперпараметров, датасета, модели, запуск цикла обучения, предсказание по тестовому образцу, отрисовка результатов.

In [ ]:

```
class Pipeline:
    def __init__(self, num_classes,
                 epochs=1000,
                 lr=0.01,
                 wd=0.0001,
                 mode='pytorch',
                 save_path='checkpoints/',
                 save_every=500,
                 save_weights=True,
                 logs_path='logs/'):
        self.epochs = epochs
        self.lr = lr
        self.wd = wd
        self.mode = mode
        self.save_path = save_path
        self.logs_path = logs_path
        self.save_every = save_every
        self.save_weights = save_weights
        self.state_dicts = []
        self.num_classes = num_classes

        if not os.path.exists(save_path):
            os.makedirs(save_path, mode=0o777)

        if not os.path.exists(logs_path):
            os.makedirs(logs_path, mode=0o777)

        self.init_dataset()
        self.init_model()

        self.train()
        self.predict()

        self.plot()

    def init_weights(self, m):
        if isinstance(m, nn.Linear):
            nn.init.xavier_uniform_(m.weight)
            m.bias.data.fill_(0.0)

    def init_dataset(self):
        if self.num_classes == 2:
            self.train_samples = torch.FloatTensor(np.array([[1.9, 0.8],
                                                             [3.1, 3.7],
                                                             [4.8, -5.0],
                                                             [3.6, 1.1],
                                                             [4.8, 0.2],
                                                             [-0.3, 3.0]]))

            self.train_labels = torch.FloatTensor(
                np.array([1.0, 1.0, 0.0, 0.0, 0.0, 1.0]))

            self.test_samples = torch.FloatTensor(np.array([[1.8, 0.8],
                                                            [3.3, 3.6],
                                                            [4.9, -5.2],
                                                            [3.7, 1.1],
                                                            [4.8, 0.2],
                                                            [-0.4, 3.0]]))
        elif self.num_classes == 4:
```

```
            self.train_samples = torch.FloatTensor(np.array([[-4, 1.5],
                                                             [-0.1, 2.7],
                                                             [2.1, 4.0],
                                                             [3.9, -1.7],
                                                             [1.9, -3.1],
                                                             [-4.7, 2.4],
                                                             [0.0, -0.3],
                                                             [4.0, 1.0]]))

            self.train_labels = torch.FloatTensor(np.array([[0.0, 0.0],
                                                            [0.0, 1.0],
                                                            [0.0, 1.0],
                                                            [1.0, 1.0],
                                                            [1.0, 1.0],
                                                            [0.0, 0.0],
                                                            [0.0, 1.0],
                                                            [1.0, 1.0]]))

            self.test_samples = torch.FloatTensor(np.array([[-4, 1.5],
                                                            [-0.1, 2.7],
                                                            [2.1, 4.0],
                                                            [3.9, -1.7],
                                                            [1.9, -3.1],
                                                            [-4.7, 2.4],
                                                            [0.0, -0.3],
                                                            [4.0, 1.0]]))

    def init_model(self):
        if self.mode == 'pytorch':
            if self.num_classes == 2:
                self.model = Perceptron_Pytorch(inp=2, outp=1)
            elif self.num_classes == 4:
                self.model = Perceptron_Pytorch(inp=2, outp=2)

            self.model.apply(self.init_weights)

            self.optimizer = torch.optim.Adam(self.model.parameters(),
                                              lr=self.lr,
                                              weight_decay=self.wd)
            self.loss_fn = nn.MSELoss()

    def train(self):
        tqdm_iter = tqdm(range(self.epochs))

        for epoch in tqdm_iter:
            pred = self.model(self.train_samples)

            if self.num_classes == 2:
                labels = self.train_labels.unsqueeze(1)
            elif self.num_classes == 4:
                labels = self.train_labels

            loss = self.loss_fn(pred, labels)

            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

            if self.save_weights:
                if epoch % self.save_every == 0 or epoch == self.epochs - 1:
                    self.save(epoch)
```

```
            tqdm_iter.set_postfix(
                {'epoch:': f'{epoch + 1}/{self.epochs}', 'loss:': loss.item()})
            tqdm_iter.refresh()

    def predict(self):
        predict_samples = self.model(self.test_samples)
        print('samples:', ["{:.2f} : {:.2f}".format(float(x[0]), float(
            x[1])) for x in self.test_samples.detach().numpy()])

        if self.num_classes == 2:
            print('predictions:', ["{:.2f}".format(float(x))
                  for x in predict_samples.detach().numpy()])
        elif self.num_classes == 4:
            preds = predict_samples.detach().numpy()
            print('predictions:', ["{:.2f} : {:.2f}".format(
                float(x[0]), float(x[1])) for x in preds])

    def save(self, epoch):
        state_dict = self.model.state_dict()
        self.state_dicts.append(state_dict)
        torch.save(state_dict, self.save_path +
                   f'epoch-{epoch}-nc-{self.num_classes}.pth')

    def plot(self):
        keys = self.state_dicts[-1].keys()

        for key in keys:
            if 'weight' in key:
                if self.num_classes == 2:
                    weights = self.state_dicts[-1][key][0].detach().numpy()
                elif self.num_classes == 4:
                    weights = self.state_dicts[-1][key].detach().numpy()
            elif 'bias' in key:
                if self.num_classes == 2:
                    bias = self.state_dicts[-1][key][0].detach().numpy()
                elif self.num_classes == 4:
                    bias = self.state_dicts[-1][key].detach().numpy()

        samples = self.train_samples.detach().numpy()
        x1_val, x2_val = [pair[0] for pair in samples], [
            pair[1] for pair in self.train_samples.detach().numpy()]

        if self.num_classes == 2:
            cls_val = [el for el in self.train_labels.detach().numpy()]

            plt.scatter(x1_val, x2_val, c=cls_val, s=20)

            ymin, ymax = plt.ylim()

            k = -weights[0] / weights[1]
            xx = np.linspace(ymin, ymax)
            yy = k * xx - bias / weights[1]

            plt.plot(xx, yy)
            plt.savefig(self.logs_path + f'result-nc-{self.num_classes}.png')
        elif self.num_classes == 4:
            fig = plt.figure()

            labels = self.train_labels.detach().numpy()
            cls1_val, cls2_val = [pair[0] for pair in self.train_labels.detach(
```

```
            ).numpy()], [pair[1] for pair in labels]

            plt.scatter(x1_val, x2_val, c=cls1_val, s=20)

            ymin, ymax = plt.ylim()

            k = -weights[0][0] / weights[0][1]
            xx = np.linspace(ymin, ymax)
            yy = k * xx - bias[0] / weights[0][1]

            plt.plot(xx, yy)
            plt.savefig(self.logs_path +
                        f'result-nc-{self.num_classes}-img-0.png')

            plt.cla()
            plt.clf()
            plt.close(fig)

            fig = plt.figure()

            plt.scatter(x1_val, x2_val, c=cls2_val, s=20)

            ymin, ymax = plt.ylim()

            k = -weights[1][0] / weights[1][1]
            xx = np.linspace(ymin, ymax)
            yy = k * xx - bias[1] / weights[1][1]

            plt.plot(xx, yy)
            plt.savefig(self.logs_path +
                        f'result-nc-{self.num_classes}-img-1.png')

            plt.cla()
            plt.clf()
            plt.close(fig)


def main():
    Pipeline(num_classes=2)
    Pipeline(num_classes=4)


if __name__ == '__main__':
    main()
```

Файл model.py.

Модель - однослойный перцептрон, написанный при помощи библиотеки PyTorch.

In [ ]:

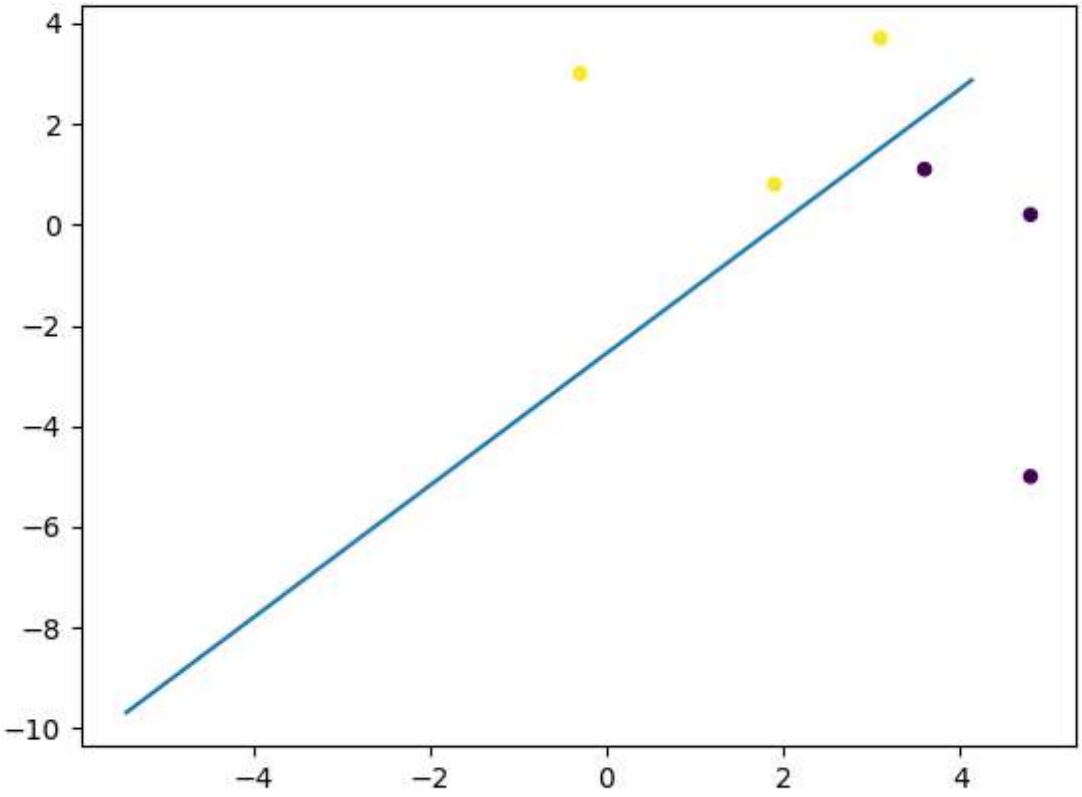
```
from torch import nn


class Perceptron_Pytorch(nn.Module):
    def __init__(self, inp, outp):
        super().__init__()
        self.fc = nn.Linear(inp, outp)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.fc(x))
```

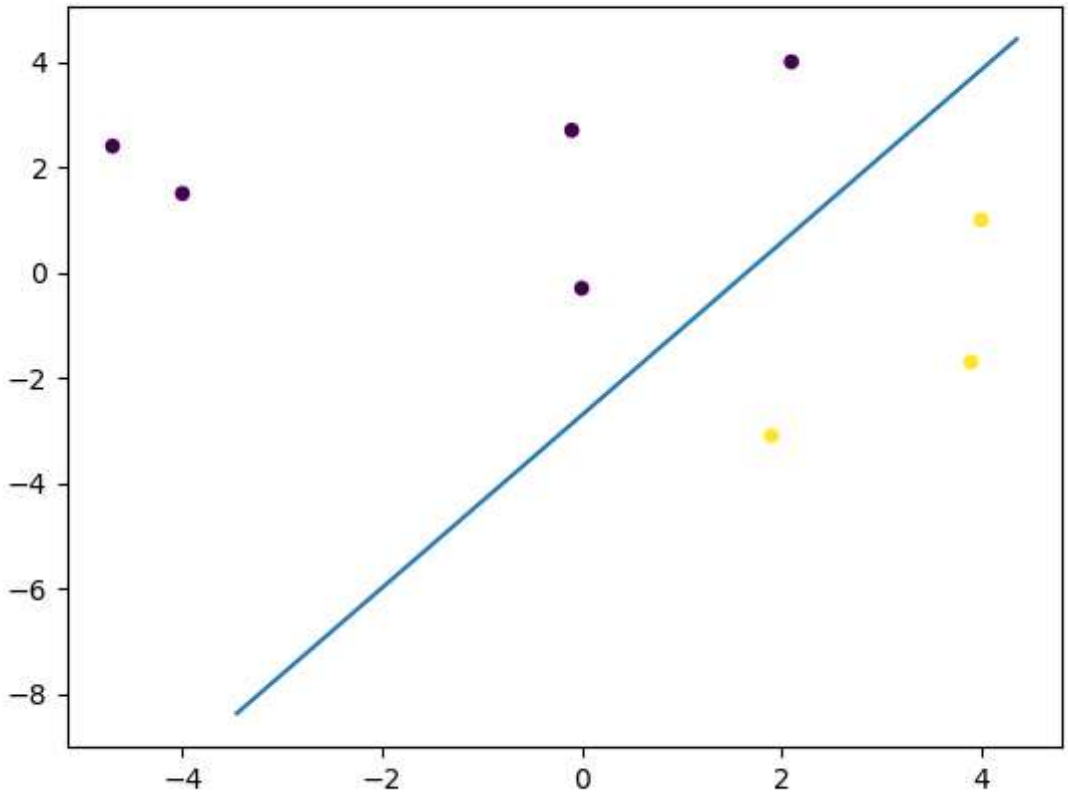
Результаты.

Классификация на два класса с использованием одного выходного нейрона и функции ошибки MSE.

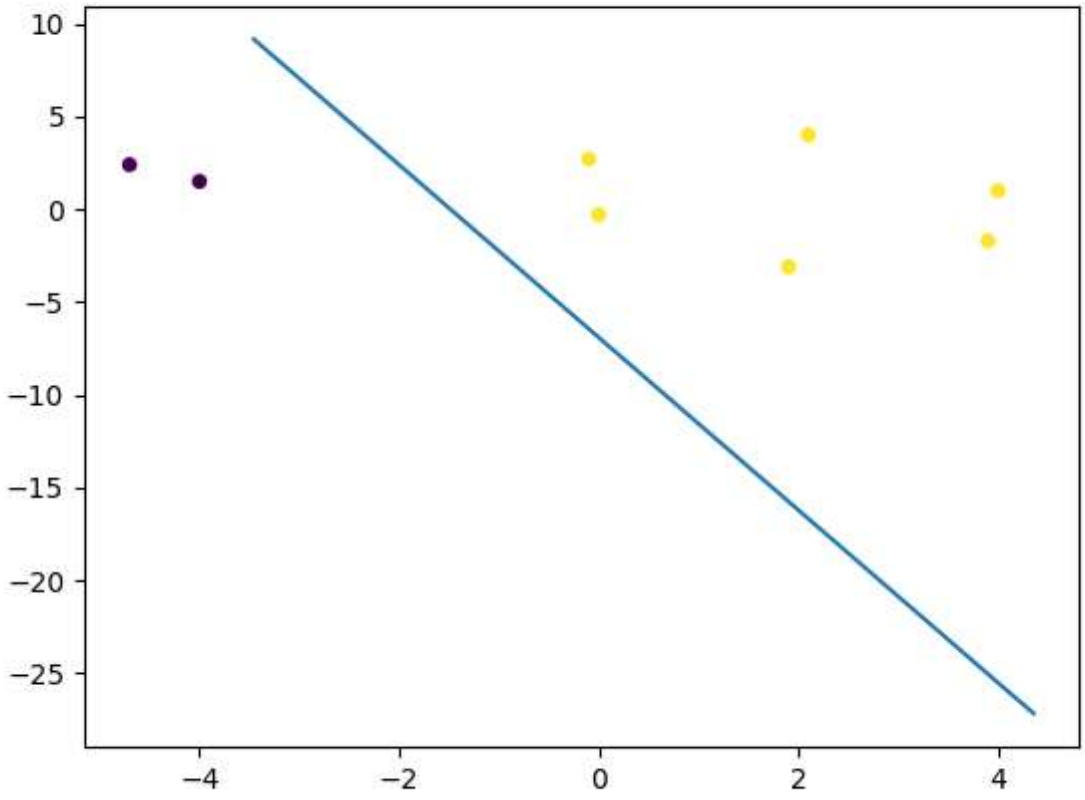


Классификация на четыре класса с использованием двух выходных нейронов и функции ошибки MSE.

Первый выходной нейрон.



Второй выходной нейрон.



Выводы.

В данной лабораторной работе мы научились работать с моделью "Персептрон", создавать модель, задавать параметры обучения, выбирать функцию ошибки, собирать данные в датасет, решили задачу классификации объектов.