**Федеральное агентство связи**

**Федеральное государственное образовательное**

**Бюджетное учреждение высшего образования «Санкт-**

**Петербургский государственный университет**

**Телекоммуникаций имени профессора Михаила**

**Александровича Бонч-Бруевича»**

Факультет «Инфокоммуникационных сетей и систем»

Кафедра «Программной инженерии и вычислительной

Техники»

Отчёт №2

По практическим (лабораторным) работам

Дисциплина: Системы искусственного интеллекта

**Вариант 2**

Выполнил студент гр. ИКПИ-84

Василец Павел

Приняла: Белая Т.И.

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Санкт-Петербург**

**2021 г.**

**Постановка задачи**

Требуется построить нейронную сеть, которая производит классификацию точек на координатной плоскости на 4 класса.

|  |  |
| --- | --- |
| ***Кол-во классов*** | ***Координаты точек проверочного множества и номер класса, к которому принадлежит каждая точка*** |
| 4 | [2, 0] – 1; [1, 0] – 2; [0, 1] – 3; |

*Таблица 1 – Исходные данные*

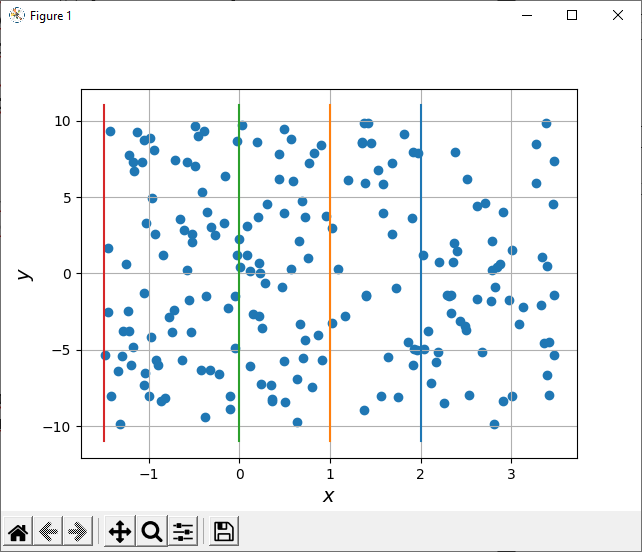
**Ход работы**

Параметры нейронной сети:

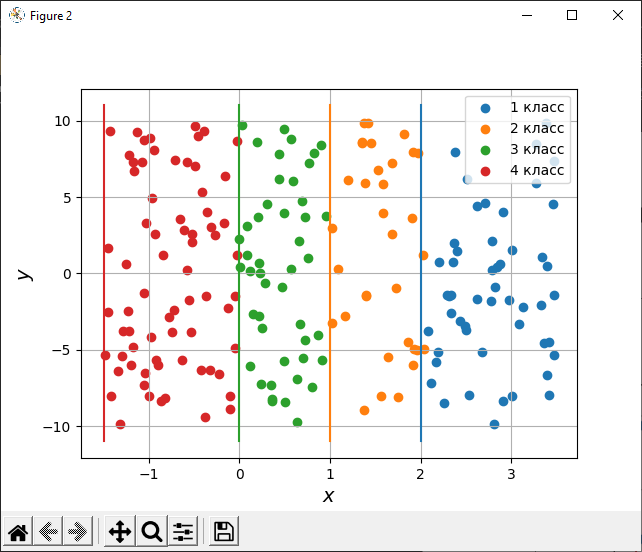
* Входной слой;
* Скрытый слой (10 нейронов)
* Выходной слой;
* В качестве функции потерь используется дискретная перекрестная энтропия;
* В качестве функции активации используется выпрямленный линейный блок (ReLU);
* В качестве оптимизатора используется аппроксимация градиентного спуска (стохастический градиентный спуск);

Наибольше показатели точности были достигнуты при следующих параметрах обучения:

* Скорость обучения: 0.01;
* Размер пакетов: 70 точек;
* Количество эпох (итераций обучения): 7 500;



*Рисунок 1 – Целевые переменные (входные данные)*

**

*Рисунок 2 – Предсказания модели*

Точность предсказаний модели: 0.985

**Вывод**

Поставленная задача была успешно выполнена. Построенная модель нейронной сети справляется с задачей классификации точек на координатной плоскости, условно разбитой на 4 гиперплоскости, с точностью ~0.95%.

Модель нейронной сети была построена с помощью фреймворка PyTorch и высокоуровневого языка программирования Python версии 3.9.4.

**Листинг**

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**import** torch  
  
  
**def** get\_data(batch\_size):  
 data = []  
 targets = np.zeros(batch\_size)  
 **for** batch\_i **in** range(batch\_size):  
 x\_point = [np.random.uniform(-1.5, 3.5), np.random.uniform(-10., 10.)]  
 data.append(x\_point)  
 **if** x\_point[0] >= 2:  
 targets[batch\_i] = 0  
 **elif** 1 <= x\_point[0] < 2:  
 targets[batch\_i] = 1  
 **elif** 0 <= x\_point[0] < 1:  
 targets[batch\_i] = 2  
 **elif** x\_point[0] < 0:  
 targets[batch\_i] = 3  
 **return** torch.FloatTensor(data), torch.LongTensor(targets)  
  
  
**def** get\_x\_y\_from\_tensor(tensor):  
 x\_points = []  
 y\_points = []  
 **for** i **in** range(tensor.size()[0]):  
 x\_points.append(tensor[i][0].item())  
 y\_points.append(tensor[i][1].item())  
 **return** x\_points, y\_points  
  
  
**def** accuracy\_score(preds, targets):  
 preds = preds.data.numpy()  
 preds = np.argmax(preds, axis=1)  
 total = len(targets.data.numpy())  
  
 correct = 0  
 **for** i **in** range(len(targets)):  
 **if** preds[i] == targets[i]:  
 correct += 1  
  
 *# print("Целевые переменные: ", targets.data.numpy())  
 # print("Предсказания модели:", preds)* print(**"Точность: "**, correct / total)  
  
  
**def** point\_classification(test\_data, preds):  
 first\_class\_x\_points = []  
 first\_class\_y\_points = []  
  
 second\_class\_x\_points = []  
 second\_class\_y\_points = []  
  
 third\_class\_x\_points = []  
 third\_class\_y\_points = []  
  
 fourth\_class\_x\_points = []  
 fourth\_class\_y\_points = []  
  
 preds = preds.data.numpy()  
 preds = np.argmax(preds, axis=1)  
  
 **for** i **in** range(test\_data.size()[0]):  
 **if** preds[i] == 0:  
 first\_class\_x\_points.append(test\_data[i][0].item())  
 first\_class\_y\_points.append(test\_data[i][1].item())  
 **elif** preds[i] == 1:  
 second\_class\_x\_points.append(test\_data[i][0].item())  
 second\_class\_y\_points.append(test\_data[i][1].item())  
 **elif** preds[i] == 2:  
 third\_class\_x\_points.append(test\_data[i][0].item())  
 third\_class\_y\_points.append(test\_data[i][1].item())  
 **elif** preds[i] == 3:  
 fourth\_class\_x\_points.append(test\_data[i][0].item())  
 fourth\_class\_y\_points.append(test\_data[i][1].item())  
 **return** first\_class\_x\_points, first\_class\_y\_points, second\_class\_x\_points, second\_class\_y\_points, third\_class\_x\_points, third\_class\_y\_points, fourth\_class\_x\_points, fourth\_class\_y\_points  
  
  
**def** visualize\_data(train\_data, preds):  
 x1\_points, y1\_points, x2\_points, y2\_points, x3\_points, y3\_points, x4\_points, y4\_points = point\_classification(  
 train\_data, preds)  
  
 classes = preds.data.numpy()  
 classes = np.argmax(classes, axis=1)  
  
 x, y = get\_x\_y\_from\_tensor(train\_data)  
  
 plt.figure(1)  
 plt.grid()  
  
 plt.xlabel(**"$x$"**, fontsize=14)  
 plt.ylabel(**"$y$"**, fontsize=14)  
  
 plt.plot([2, 2], [-11, 11])  
 plt.plot([1, 1], [-11, 11])  
 plt.plot([0, 0], [-11, 11])  
 plt.plot([-1.5, -1.5], [-11, 11])  
  
 plt.scatter(x, y)  
  
 plt.figure(2)  
  
 plt.plot([2, 2], [-11, 11])  
 plt.plot([1, 1], [-11, 11])  
 plt.plot([0, 0], [-11, 11])  
 plt.plot([-1.5, -1.5], [-11, 11])  
  
 plt.scatter(x1\_points, y1\_points, label=**"1 класс"**)  
 plt.scatter(x2\_points, y2\_points, label=**"2 класс"**)  
 plt.scatter(x3\_points, y3\_points, label=**"3 класс"**)  
 plt.scatter(x4\_points, y4\_points, label=**"4 класс"**)  
 plt.legend()  
  
 plt.grid()  
 plt.xlabel(**"$x$"**, fontsize=14)  
 plt.ylabel(**"$y$"**, fontsize=14)  
  
 plt.show()  
  
  
batch\_size = 70  
input\_dim = 2  
hidden\_dimension = 10  
output\_dimension = 4  
learning\_rate = 0.01  
epochs = 7500  
  
two\_layer\_net = torch.nn.Sequential(  
 torch.nn.Linear(input\_dim, hidden\_dimension),  
 torch.nn.ReLU(),  
 torch.nn.Linear(hidden\_dimension, output\_dimension)  
)  
  
ce\_loss = torch.nn.CrossEntropyLoss(size\_average=**False**)  
optimizer = torch.optim.SGD(two\_layer\_net.parameters(), lr=learning\_rate)  
print(**"Обучение запущено"**)  
**for** epoch **in** range(epochs):  
 train\_data, train\_targets = get\_data(batch\_size)  
 preds = two\_layer\_net(train\_data)  
  
 loss = ce\_loss(preds, train\_targets)  
 optimizer.zero\_grad()  
  
 *# print(loss.item())* loss.backward()  
 optimizer.step()  
  
 **if** epoch % 500 == 0:  
 print(epoch, **"эпоха"**)  
  
print(**"Обучение окончено"**)  
  
test\_data, test\_targets = get\_data(200)  
check = torch.FloatTensor([[0.99, 10]])  
preds = two\_layer\_net(test\_data)  
  
accuracy\_score(preds, test\_targets)  
  
visualize\_data(test\_data, preds)