МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра АПУ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Теория автоматического управления. Часть 2. Нелинейные системы»
Тема: Нейросетевая аппроксимация СХ НЭ

Студент гр. 2392 _____ Жук Ф.П.

Преподаватель _____ Имаев Д.Х.

Санкт-Петербург
2025

Цель работы.

Изучение методов аппроксимации статических характеристик нелинейных элементов с использованием искусственных нейронных сетей на примере газодинамических характеристик центробежного компрессора.

Обработка результатов эксперимента.

В ходе работы, была разработана нейросетевая модель для решения задачи подбора газодинамических характеристик центробежного компрессора. Для разработки нейросетевой модели была использована библиотека torch.

Начальной задачей выступала подготовка данных для обучения модели. В ходе данной подготовки был разработан класс Dataset (рис. 1).

```
class Dataset(torch.utils.data.Dataset):
    def __init__(self, inp):
        super(Dataset, self).__init__()
        self.data = list(zip(map(float, inp[0].split()), map(float, inp[1].split())))
        self.data = torch.Tensor(self.data)
        self.lable = list(map(float, inp[2].split()))
        self.lable = torch.Tensor(self.lable)

    def __getitem__(self, index):
        data = self.data[index]
        lable = self.lable[index]
        return data, lable

    def __len__(self):
        return len(self.lable)
```

Рис. 1 – класс Dataset

Он наследуеться от класса torch.utils.data.Dataset. В методе __init__ представлена инициализация класса. Метод __getitem__ возвращает элемент набора дюанных по индексу. Метод __len__ возвращвет длинну нгабора данных.

Далее данные были разбиты на тренировочную, тестовую и валидационую выборки.

```
train, val, test = torch.utils.data.random_split(data, [61, 5, 5])
```

Далее создаем загрузчики данных (Dataloader)

```
dataloader_train = torch.utils.data.DataLoader(train, 1, shuffle=True)
dataloader_val = torch.utils.data.DataLoader(val, 1, shuffle=True)
```

dataloader_test = torch.utils.data.DataLoader(test, 1, shuffle=True)

В качестве модели была выбрана многослойная нейронная сеть (рис. 2).

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.pipe = nn.Sequential(
            nn.Linear(2, 10),
            nn.ReLU(),
            nn.Linear(10, 30),
            nn.ReLU(),
            nn.Linear(30, 10),
            nn.ReLU(),
            nn.Linear(10 , 1)
        )


    def forward(self, x):
        x = self.pipe(x)
        return x
```

Рис. 2 – Класс многослойной нейронной сети

Данная модель состоит из 4 слоев, на вход принимает 2 параметра, на выходе выдаёт 1. В методе __init__ мы инициализируем модель и её функциональные элементы. Метод forward представляет прямое распространение в нейронной сети.

Также был реализована функция для оценки среднего отклонения на заданной выборке (рис. 3).

```
def test(model, dataloader):
    score = []
    for data in tqdm(dataloader):
        inp, lable = data
        inp, lable = inp.to(device), lable.to(device)

        out = model(inp)

        score.append(abs((lable-out).item()))
    score = np.array(score)
    return score.mean()
```

Рис. 3 – Функция теста

Данная функция принимает 2 аргумента: модель и набор данных.

Далее была реализованная функция для тренировки модели (рис. 4).

```python
def train(model, valloader, trainloader, optimizer, criterion, epoch):

    loss_stats = {
        "loss_func":[],
        "valid_loss":[]
    }

    for ep in range(epoch):
        model.train()
        running_loss = 0
        for data in (bar := tqdm(trainloader)):
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            bar.set_description(f'epoch: {ep}\tloss: {running_loss:.3F}')

        loss_stats["loss_func"].append(running_loss)

        model.eval()

        loss_stats["valid_loss"].append(test(model, valloader))
    return loss_stats
```

Рис. 4 – Функция тренировки

Она принимает следующие аргументыы:

1. Модель для обучения
2. Валидационный загрузчик данных
3. Тестовый загрузчик данных
4. Оптимизатор
5. Функция ощибки

6. Количество эпох

В качестве оптимизатора был выбран Adam, так как он является эффективным и универсальным. В качестве функции ошибки выбрана MSELoss, так как его будет достаточно для решения поставленной задачи. Модель обучалась 100 эпох.

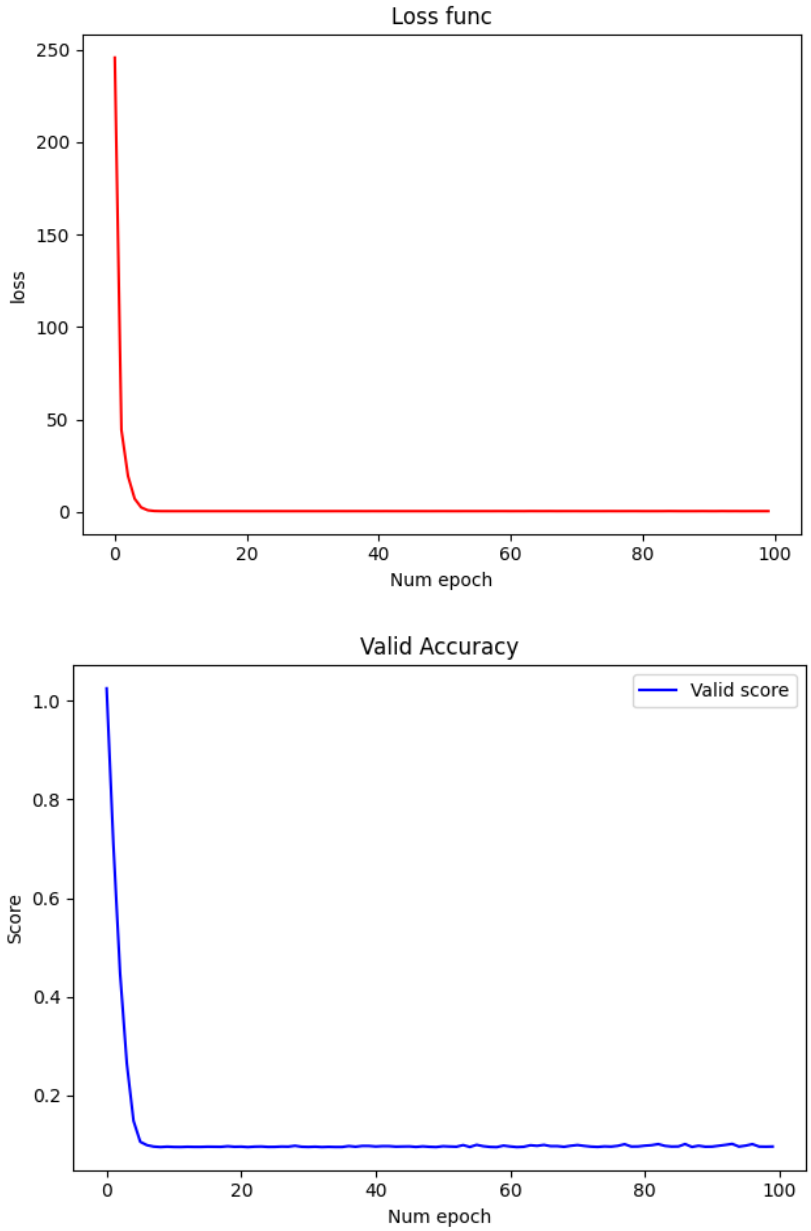

Рис. 5 – Результаты обучения

В ходе обученя были полученны два граффика (рис. 5).

Результатом на тестовой выборке стал показатель 0.1.

Граффик сравнения результатов представлен на рисунке 6.

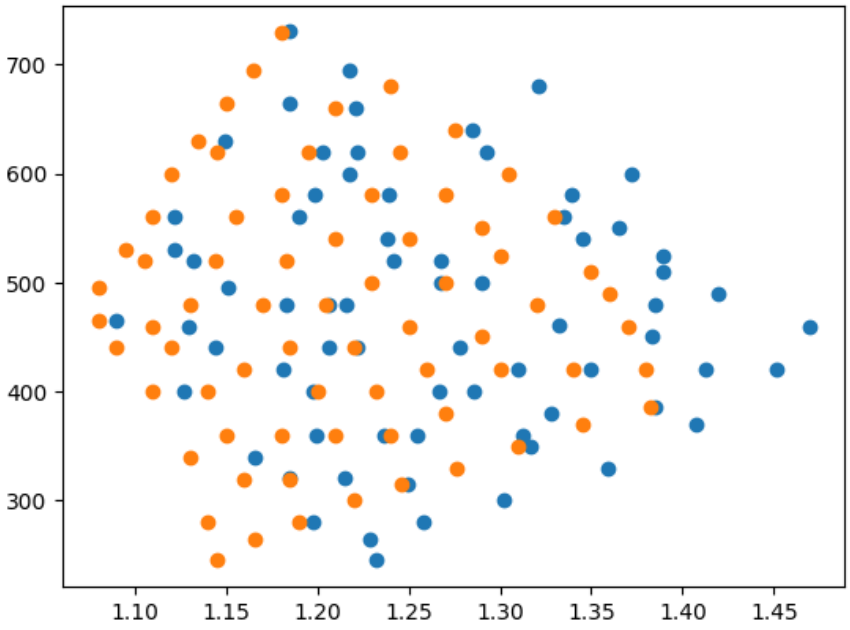

Рис. 6 – Сравнения

Оранжевые точки это данных и дата сета, а синие это полученные моделью.

Выводы.

Результаты обучения нейросетевой модели показали малую эффективность из-за недостаточного объёма данных. Малый набор обучающих примеров (71 точка) привёл к переобучению, что выразилось в высокой ошибке на тестовой выборке. Модель демонстрирует адекватную аппроксимацию только в узких диапазонах параметров, но ошибается вне этих зон. Для улучшения результатов требуется расширение датасета.