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).

```
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.

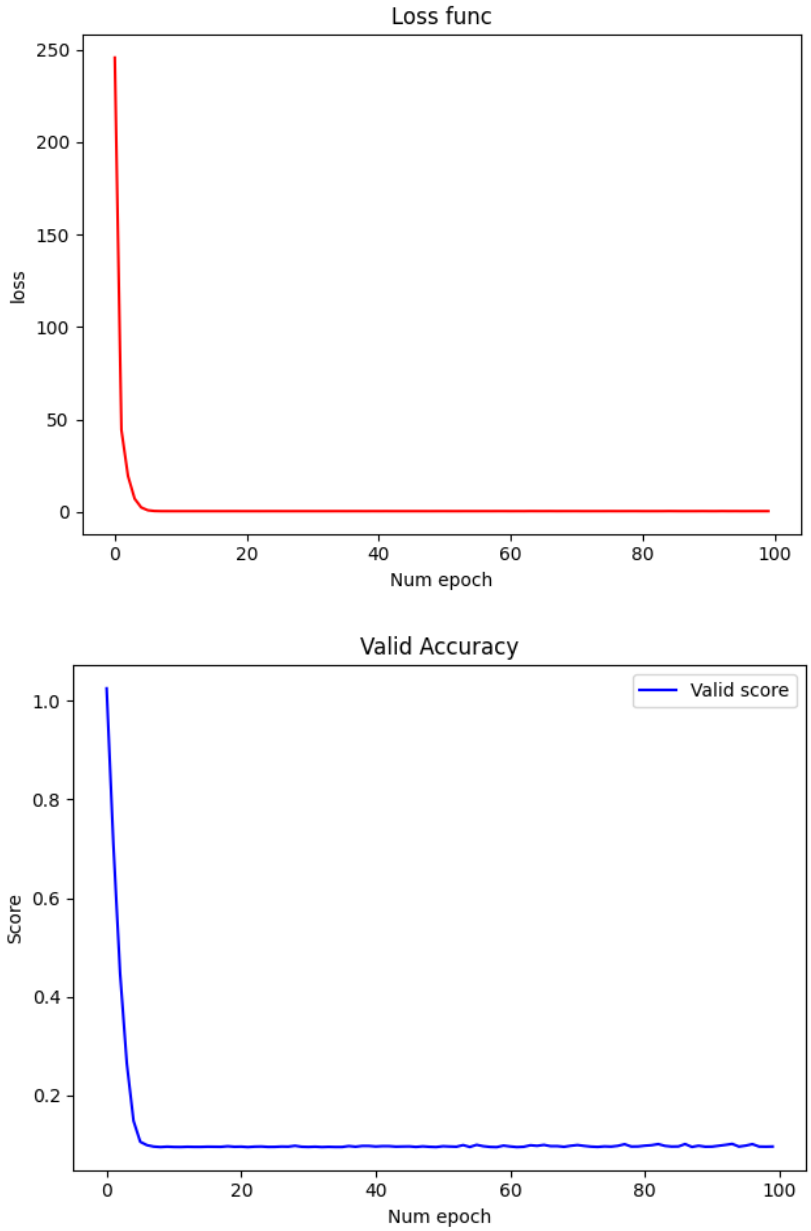

Рис. 5 – Результаты обучения

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

Результатом на тестовой выборке стал показатель 0.1.

Граффик сравнения результатов представлен на рисунке 6.

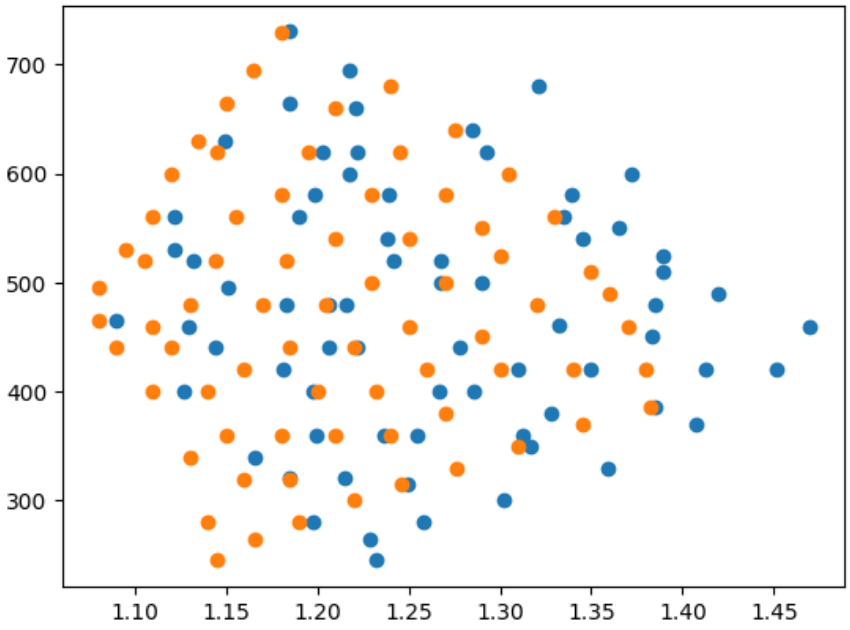

Рис. 6 – Сравнения

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

Также обучим сдледующию модедь (рис. 7).

```
class Net_1(nn.Module):
    def __init__(self):
        super(Net_1, self).__init__()
        self.pipe = nn.Sequential(
            nn.Linear(2, 1),
        )


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

Рис. 7 – Модель

Модель обучалась 25 эпох. И показала не плохие результатов обучения (рис. 8). Итог на тестовой выборке 0.0833, значение ошибки по MSELoss.

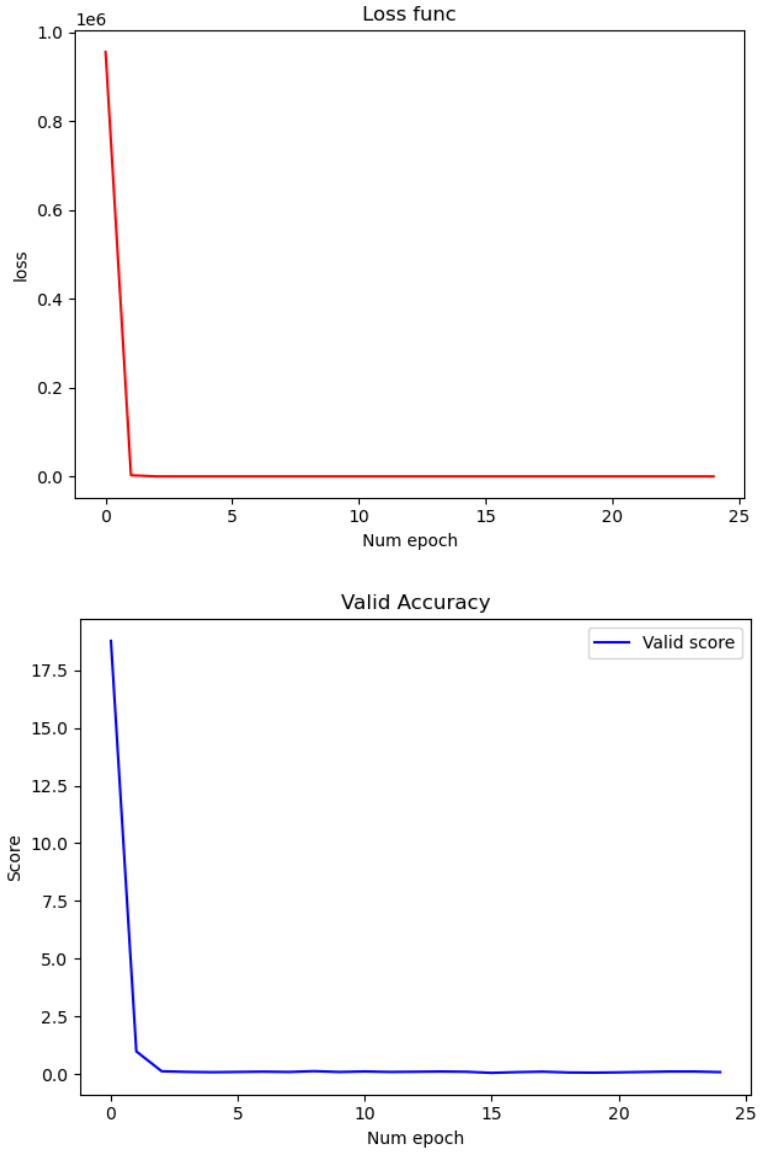

Рис. 8 – Результаты обучения

Граффик сравнения результатов представлен на рисунке 9.

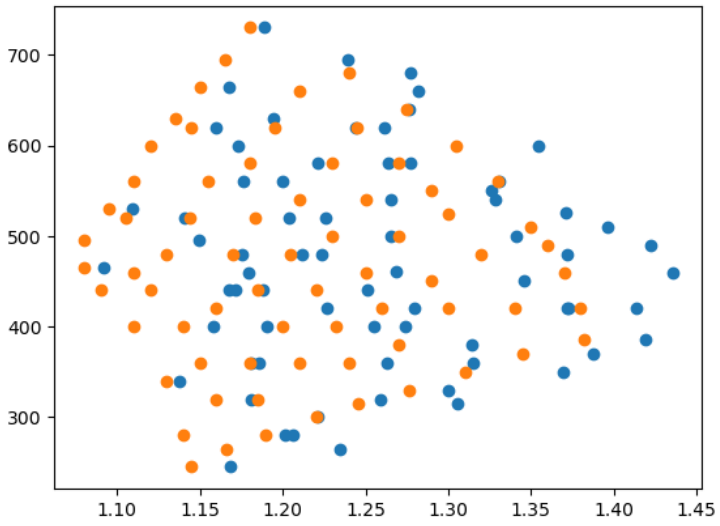

Рис. 9 - Сравнения

Далее была обучена следующая модель (рис. 10)

```
class Net_2(nn.Module):
    def __init__(self):
        super(Net_2, self).__init__()
        self.pipe = nn.Sequential(
            nn.Linear(2, 50),
            nn.ReLU(),
            nn.Linear(50, 1)
        )


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

Рис. 10 – Модель

Модель обучалась 25 эпох. И показала не плохие результатов обучения (рис. 11). Итог на тестовой выборке 0.21, значение ошибки по MSELoss.

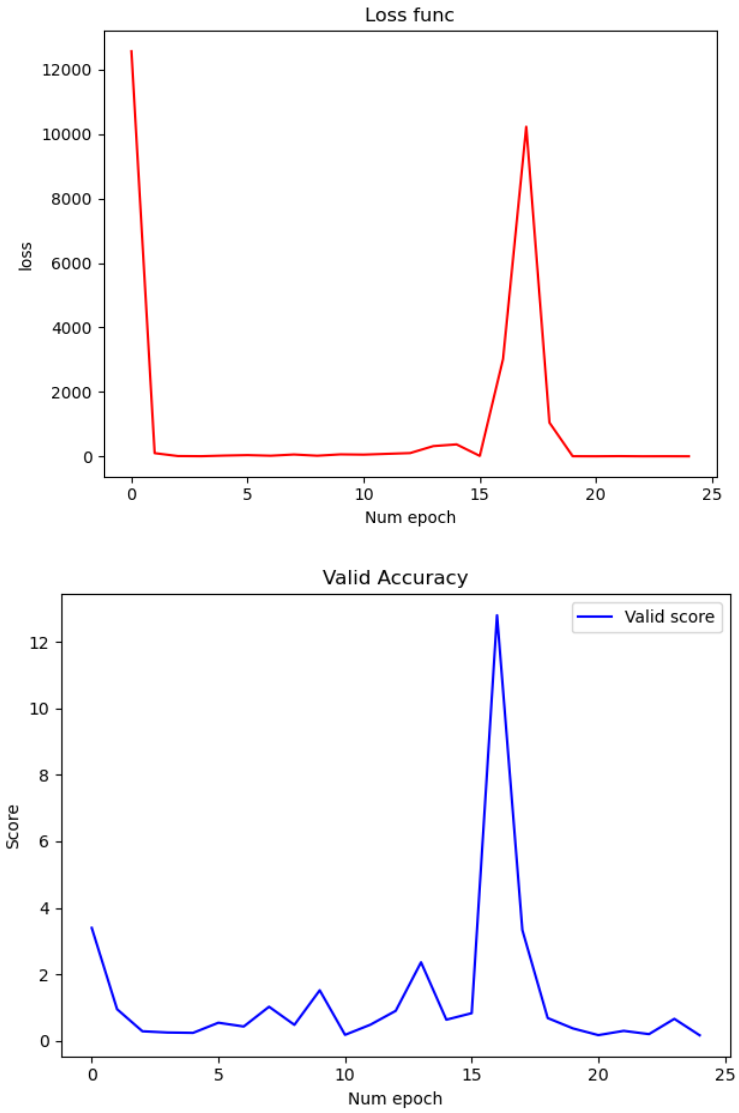

Рис. 11 – Результаты обучения

Граффик сравнения результатов представлен на рисунке 12.

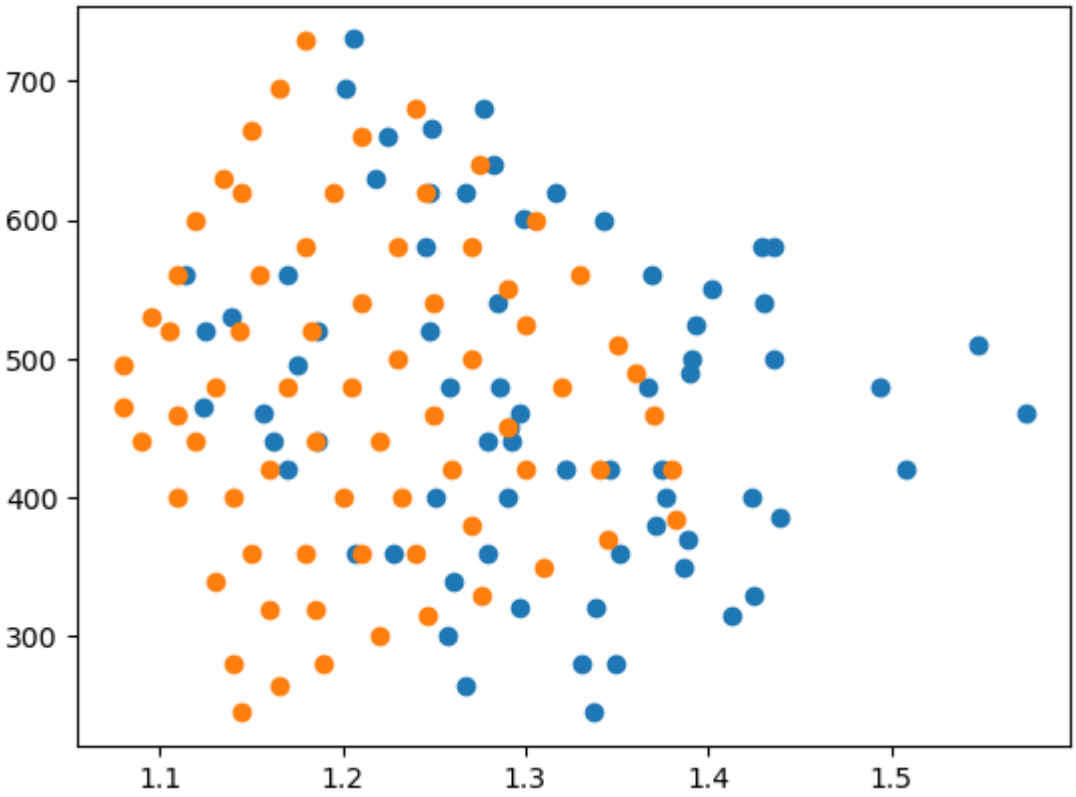

Рис. 12 – Сравнения

Таблица сравнения.

Inp	Out	epoh	loss
1	100	35	0,086598
2	99	37	0,658
3	98	39	0,166689
4	97	35	0,307791
5	96	39	0,11546
6	95	36	0,448204
7	94	36	0,356525
8	93	33	0,52453
9	92	33	0,439483
10	91	30	0,326114
11	90	33	0,426771
12	89	39	0,279333
13	88	35	0,28318
14	87	32	0,315388
15	86	37	0,602992
16	85	37	0,301877
17	84	30	0,315702
18	83	35	0,411531

19	82	30	0,372751
20	81	39	1,0514
21	80	32	0,796629
22	79	38	0,554824
23	78	32	0,317602
24	77	30	0,326181
25	76	31	0,379212
26	75	33	0,267673
27	74	33	0,28669
28	73	39	0,863041
29	72	37	0,264089
30	71	37	0,518287
31	70	36	0,390733
32	69	34	0,284102
33	68	38	0,443014
34	67	34	0,415306
35	66	35	0,662231
36	65	31	0,472882
37	64	38	0,531864
38	63	39	0,627975
39	62	34	0,333471
40	61	33	0,309683
41	60	37	0,285609
42	59	38	0,290118
43	58	34	0,459965
44	57	34	0,269279
45	56	37	0,492928
46	55	35	0,333294
47	54	39	0,316664
48	53	39	0,367993
49	52	36	0,333343
50	51	30	0,444006
51	50	35	0,54968
52	49	38	0,279858
53	48	37	0,292156
54	47	35	0,349992
55	46	38	0,284747
56	45	36	0,30259
57	44	34	0,428471
58	43	34	0,33691
59	42	35	0,307584
60	41	39	0,304415
61	40	37	0,549221
62	39	34	0,358459
63	38	36	0,320629
64	37	31	0,352254

65	36	33	0,281851
66	35	31	0,272728
67	34	32	0,335197
68	33	37	0,320938
69	32	39	0,301592
70	31	33	0,288064
71	30	31	0,287481
72	29	33	0,332253
73	28	35	0,274479
74	27	30	0,335616
75	26	31	0,319622
76	25	33	0,446909
77	24	32	0,349231
78	23	35	0,291298
79	22	32	0,386408
80	21	37	0,30246
81	20	32	0,16969
82	19	38	0,564839
83	18	30	0,278016
84	17	34	0,348181
85	16	34	0,269056
86	15	31	0,278206
87	14	36	0,080301
88	13	35	0,173717
89	12	34	0,34374
90	11	34	0,33308
91	10	33	0,295059
92	9	33	0,2048
93	8	35	0,361607
94	7	34	0,215878
95	6	38	0,083062
96	5	36	0,085968
97	4	38	0,08241
98	3	31	0,16013
99	2	31	0,084575

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.