

Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №5
По дисциплине: «ОМО»
Тема: «Нелинейные ИНС в задачах регрессии»

Выполнил:
Студент 3-го курса
Группы АС-65
Осовец М. М.
Проверил:
Крощенко А. А.

Брест 2025

Цель работы: научиться выполнять моделирование прогнозирующей нелинейной ИНС

Задание:

1. Выполнить моделирование прогнозирующей нелинейной ИНС. Для генерации обучающих и тестовых данных использовать функцию

$$y = a \cos(bx) + c \sin(dx) .$$

Варианты заданий приведены в следующей таблице:

№ варианта	a	b	c	d	Кол-во входов ИНС	Кол-во НЭ в скрытом слое
1	0.1	0.1	0.05	0.1	6	2
2	0.2	0.2	0.06	0.2	8	3
3	0.3	0.3	0.07	0.3	10	4
4	0.4	0.4	0.08	0.4	6	2
5	0.1	0.5	0.09	0.5	8	3
6	0.2	0.6	0.05	0.6	10	4
7	0.3	0.1	0.06	0.1	6	2
8	0.4	0.2	0.07	0.2	8	3
9	0.1	0.3	0.08	0.3	10	4
10	0.2	0.4	0.09	0.4	6	2
11	0.3	0.5	0.05	0.5	8	3

Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

Вариант 3

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from typing import Tuple
np.random.seed(42)

# Генерация данных
a, b, c, d = 0.3, 0.3, 0.07, 0.3
n_inputs = 10
N_total = 2000
train_frac = 0.7

t = np.linspace(0, 40, N_total)
y = a*np.cos(b*t) + c*np.sin(d*t)

X=[]
Y=[]
for i in range(N_total-n_inputs):
    X.append(y[i:i+n_inputs])
    Y.append(y[i+n_inputs])

X=np.array(X)
Y=np.array(Y)

split=int(len(X)*train_frac)
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]
```

```
print("Форматы:", X_train.shape, Y_train.shape)
```

```
Форматы: (1393, 10) (1393,)
```

```
# Реализация 5: нативная MLP (numpy)
```

```
class NumpyMLP:
```

```
    def __init__(self, n_in, n_hidden, lr=1e-3):
        self.W1 = np.random.randn(n_hidden, n_in) * 0.1
        self.b1 = np.zeros((n_hidden, 1))
        self.W2 = np.random.randn(1, n_hidden) * 0.1
        self.b2 = np.zeros((1, 1))
        self.lr = lr
```

```
    def sigmoid(self, z):
        return 1/(1+np.exp(-z))
```

```
    def sigmoid_deriv(self, s):
        return s*(1-s)
```

```
    def forward(self, x):
        z1 = x.dot(self.W1.T) + self.b1.T
        a1 = self.sigmoid(z1)
        z2 = a1.dot(self.W2.T) + self.b2.T
        out = z2[:,0]
        return out, (x,z1,a1,z2)
```

```
    def compute_loss(self, pred, y):
        return np.mean((pred-y)**2)
```

```
    def backward(self, cache, pred, y):
        x,z1,a1,z2 = cache
        m = x.shape[0]
        dloss_dz2 = (2/m)*(pred-y).reshape(-1,1)
        dW2 = dloss_dz2.T.dot(a1)
        db2 = np.sum(dloss_dz2,axis=0,keepdims=True)
```

```
        da1 = dloss_dz2.dot(self.W2)
        dz1 = da1*self.sigmoid_deriv(a1)
```

```
        dW1 = dz1.T.dot(x)
        db1 = np.sum(dz1,axis=0,keepdims=True).T
```

```
        self.W2 -= self.lr*dW2
        self.b2 -= self.lr*db2.T
        self.W1 -= self.lr*dW1
        self.b1 -= self.lr*db1
```

```
    def fit(self, X, Y, epochs=500, batch_size=32):
        history=[]
        for ep in range(epochs):
            idx=np.random.permutation(len(X))
            Xs,Ys=X[idx],Y[idx]
            for i in range(0,len(X),batch_size):
                xb=xs= Xs[i:i+batch_size]
                yb=Ys[i:i+batch_size]
                pred,cache=self.forward(xb)
                self.backward(cache,pred,yb)
            pred_all,_=self.forward(X)
            loss=self.compute_loss(pred_all,Y)
            history.append(loss)
            if ep%100==0:
                print("Epoch",ep,"Loss:",loss)
```

```

        return history

    def predict(self,X):
        pred,_=self.forward(X)
        return pred

model_np = NumpyMLP(n_inputs,4,lr=1e-3)
history_np = model_np.fit(X_train,Y_train,epochs=500)

pred_train_np=model_np.predict(X_train)
df_train_np=pd.DataFrame({
    "y_true":Y_train,
    "y_pred":pred_train_np,
    "diff":pred_train_np-Y_train
})
df_train_np.head(20)

```

```

Epoch 0 Loss: 0.04443062386734559
Epoch 100 Loss: 0.04199218001037184
Epoch 200 Loss: 0.038779590799793394
Epoch 300 Loss: 0.03401927713711878
Epoch 400 Loss: 0.027398651355764653

```

	y_true	y_pred	diff
0	0.303659	0.127100	-0.176559
1	0.303965	0.127219	-0.176746
2	0.304260	0.127335	-0.176925
3	0.304544	0.127447	-0.177097
4	0.304817	0.127556	-0.177261
5	0.305079	0.127661	-0.177418
6	0.305330	0.127762	-0.177568
7	0.305570	0.127860	-0.177710
8	0.305799	0.127954	-0.177845
9	0.306017	0.128045	-0.177973
10	0.306224	0.128132	-0.178093
11	0.306420	0.128215	-0.178206
12	0.306605	0.128295	-0.178311
13	0.306779	0.128371	-0.178409
14	0.306942	0.128443	-0.178499
15	0.307094	0.128512	-0.178582
16	0.307235	0.128577	-0.178657
17	0.307364	0.128639	-0.178726
18	0.307483	0.128697	-0.178786
19	0.307590	0.128751	-0.178839

```

plt.plot(history_np)
plt.title("Ошибка (MSE) — нативная модель")
plt.show()

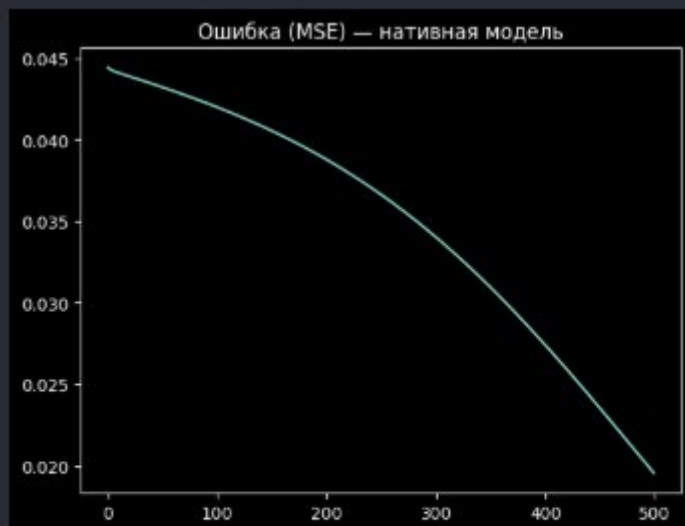
```

```

pred_test_np=model_np.predict(X_test)
df_test_np=pd.DataFrame({
    "y_true":Y_test,
    "y_pred":pred_test_np,
    "diff":pred_test_np-Y_test
})

```

```
df_test_np.head(20)
```



	y_true	y_pred	diff
0	-0.102442	-0.007197	0.095246
1	-0.104184	-0.007800	0.096385
2	-0.105923	-0.008401	0.097522
3	-0.107658	-0.009002	0.098656
4	-0.109388	-0.009601	0.099788
5	-0.111115	-0.010198	0.100917
6	-0.112838	-0.010795	0.102043
7	-0.114557	-0.011390	0.103167
8	-0.116271	-0.011983	0.104288
9	-0.117981	-0.012576	0.105406
10	-0.119688	-0.013166	0.106521
11	-0.121389	-0.013756	0.107634
12	-0.123087	-0.014343	0.108743
13	-0.124780	-0.014930	0.109850
14	-0.126468	-0.015514	0.110954
15	-0.128152	-0.016098	0.112055
16	-0.129832	-0.016679	0.113153
17	-0.131506	-0.017259	0.114247
18	-0.133176	-0.017838	0.115339
19	-0.134841	-0.018414	0.116427

```
# PyTorch модель
import torch
import torch.nn as nn
import torch.optim as optim
```

```
torch.manual_seed(0)
```

```
X_train_t=torch.tensor(X_train,dtype=torch.float32)
Y_train_t=torch.tensor(Y_train,dtype=torch.float32).unsqueeze(1)
X_test_t=torch.tensor(X_test,dtype=torch.float32)
Y_test_t=torch.tensor(Y_test,dtype=torch.float32).unsqueeze(1)
```

```

class TorchMLP(nn.Module):
    def __init__(self, n_in, n_hidden):
        super().__init__()
        self.fc1=nn.Linear(n_in, n_hidden)
        self.act=nn.Sigmoid()
        self.fc2=nn.Linear(n_hidden, 1)
    def forward(self, x):
        return self.fc2(self.act(self.fc1(x)))

model_t=TorchMLP(n_inputs, 4)
opt=optim.Adam(model_t.parameters(), lr=1e-3)
loss_fn=nn.MSELoss()

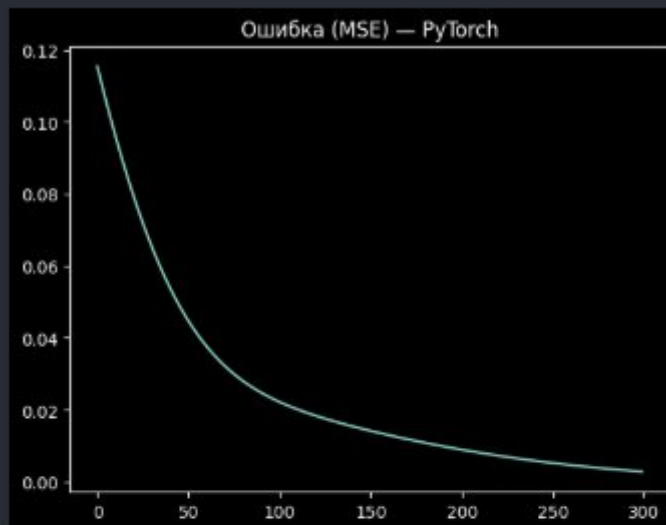
hist_t=[]
for ep in range(300):
    opt.zero_grad()
    out=model_t(X_train_t)
    loss=loss_fn(out, Y_train_t)
    loss.backward()
    opt.step()
    hist_t.append(loss.item())
    if ep%50==0:
        print("Epoch", ep, "Loss:", loss.item())

plt.plot(hist_t)
plt.title("Ошибка (MSE) - PyTorch")
plt.show()

pred_test_t=model_t(X_test_t).detach().numpy().squeeze()
df_test_t=pd.DataFrame({
    "y_true":Y_test,
    "y_pred":pred_test_t,
    "diff":pred_test_t-Y_test
})
df_test_t.head(20)

```

```
Epoch 0 Loss: 0.11538349837064743
Epoch 50 Loss: 0.04469527676701546
Epoch 100 Loss: 0.0221650842577219
Epoch 150 Loss: 0.014133847318589687
Epoch 200 Loss: 0.00890666525810957
Epoch 250 Loss: 0.005193581338971853
```



	y_true	y_pred	diff
0	-0.102442	-0.060263	0.042179
1	-0.104184	-0.061626	0.042558
2	-0.105923	-0.062986	0.042937
3	-0.107658	-0.064343	0.043314
4	-0.109388	-0.065697	0.043691
5	-0.111115	-0.067048	0.044067
6	-0.112838	-0.068395	0.044442
7	-0.114557	-0.069739	0.044817
8	-0.116271	-0.071080	0.045191
9	-0.117981	-0.072418	0.045564
10	-0.119688	-0.073751	0.045936
11	-0.121389	-0.075082	0.046308
12	-0.123087	-0.076409	0.046678
13	-0.124780	-0.077732	0.047048
14	-0.126468	-0.079051	0.047417
15	-0.128152	-0.080367	0.047786
16	-0.129832	-0.081679	0.048153
17	-0.131506	-0.082987	0.048520
18	-0.133176	-0.084291	0.048886
19	-0.134841	-0.085591	0.049251

Вывод: научились выполнять моделирование прогнозирующей нелинейной ИНС