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-66
Цеван К.А.
Проверил:
Крощенко А.А.

Брест 2025

Цель: изучить нелинейные ИНС в задачах регрессии.

Вариант 12

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

1 0.1 0.1 0.05 0.1 6 2

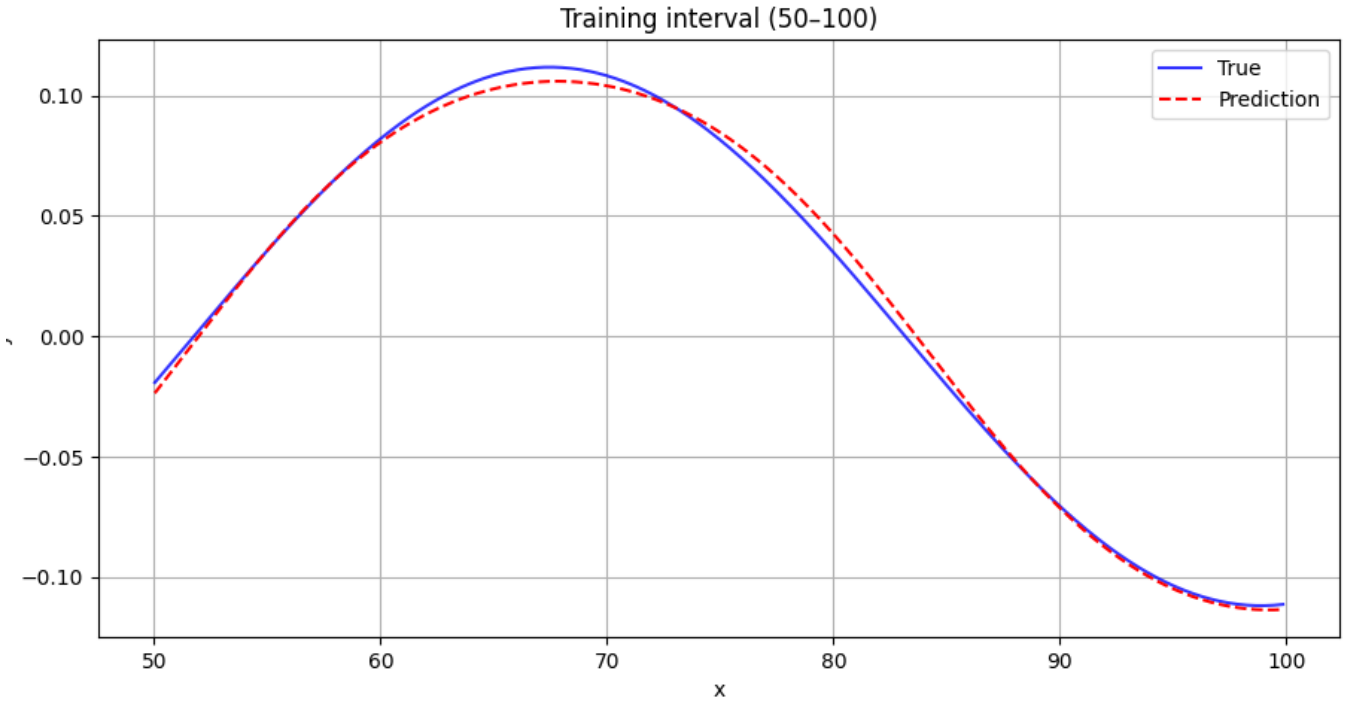
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

a = 0.1
b = 0.1
c = 0.05
d = 0.1
n_inputs = 6
n_hidden = 2

def target_function(x):
    return a * np.cos(b * x) + c * np.sin(d * x)

x = np.linspace(-100, 300, 4000)
y = target_function(x)

def create_dataset(data, n_inputs):
    X, Y = [], []
    for i in range(len(data) - n_inputs):
        X.append(data[i:i + n_inputs])
        Y.append(data[i + n_inputs])
    return np.array(X), np.array(Y)

scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_scaled = scaler_x.fit_transform(x.reshape(-1, 1)).flatten()
y_scaled = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

X, Y = create_dataset(y_scaled, n_inputs)

start_train = np.where(x >= 50)[0][0] - n_inputs
end_train = np.where(x <= 100)[0][-1] - n_inputs
start_test = np.where(x >= 100)[0][0] - n_inputs
end_test = np.where(x <= 150)[0][-1] - n_inputs
```

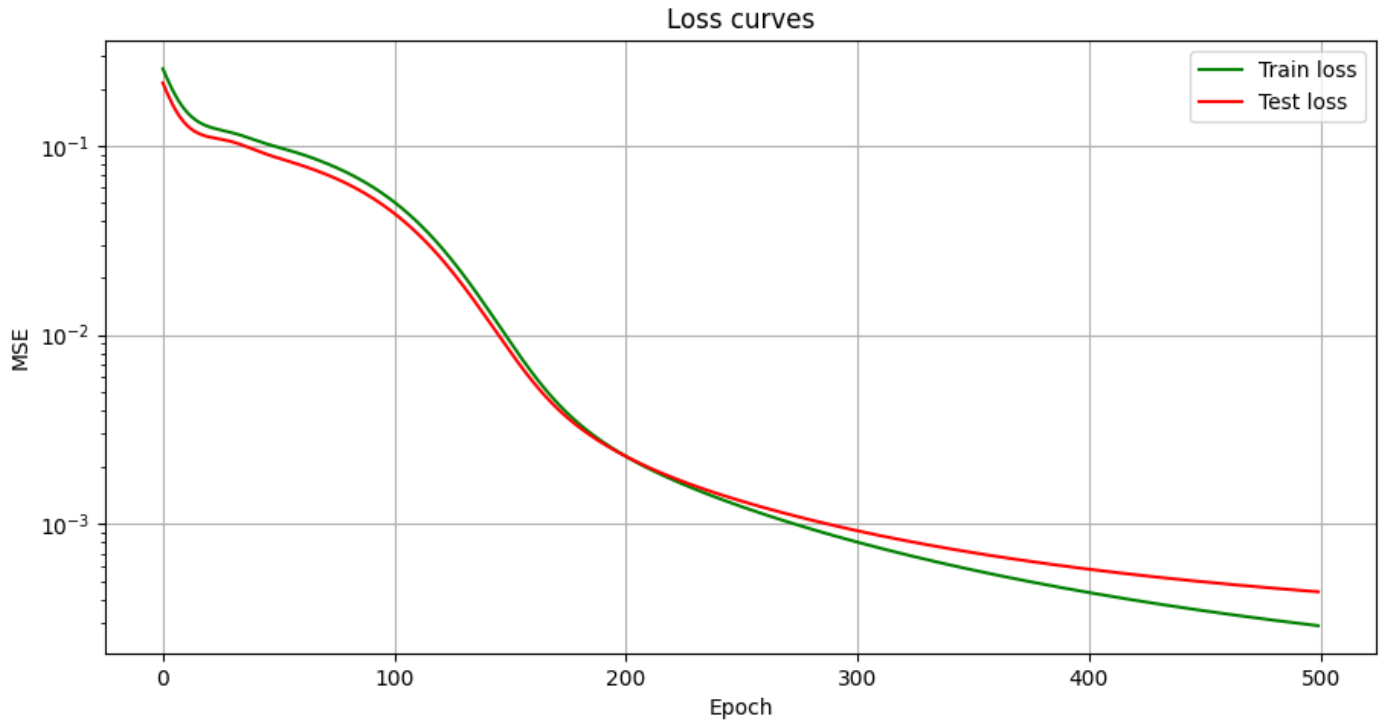
```
X_train, X_test = X[start_train:end_train], X[start_test:end_test]
Y_train, Y_test = Y[start_train:end_train], Y[start_test:end_test]

X_train = torch.FloatTensor(X_train)
Y_train = torch.FloatTensor(Y_train).reshape(-1, 1)
X_test = torch.FloatTensor(X_test)
Y_test = torch.FloatTensor(Y_test).reshape(-1, 1)

class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NeuralNetwork, self).__init__()
        self.hidden = nn.Linear(input_size, hidden_size)
        self.output = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.hidden(x))
        x = self.output(x)
        return x

model = NeuralNetwork(n_inputs, n_hidden, 1)

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

train_losses = []
test_losses = []
epochs = 500

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    train_loss = criterion(outputs, Y_train)
    train_loss.backward()
    optimizer.step()

    model.eval()
    with torch.no_grad():
        test_outputs = model(X_test)
        test_loss = criterion(test_outputs, Y_test)

    train_losses.append(train_loss.item())
    test_losses.append(test_loss.item())

    if epoch % 100 == 0:
```

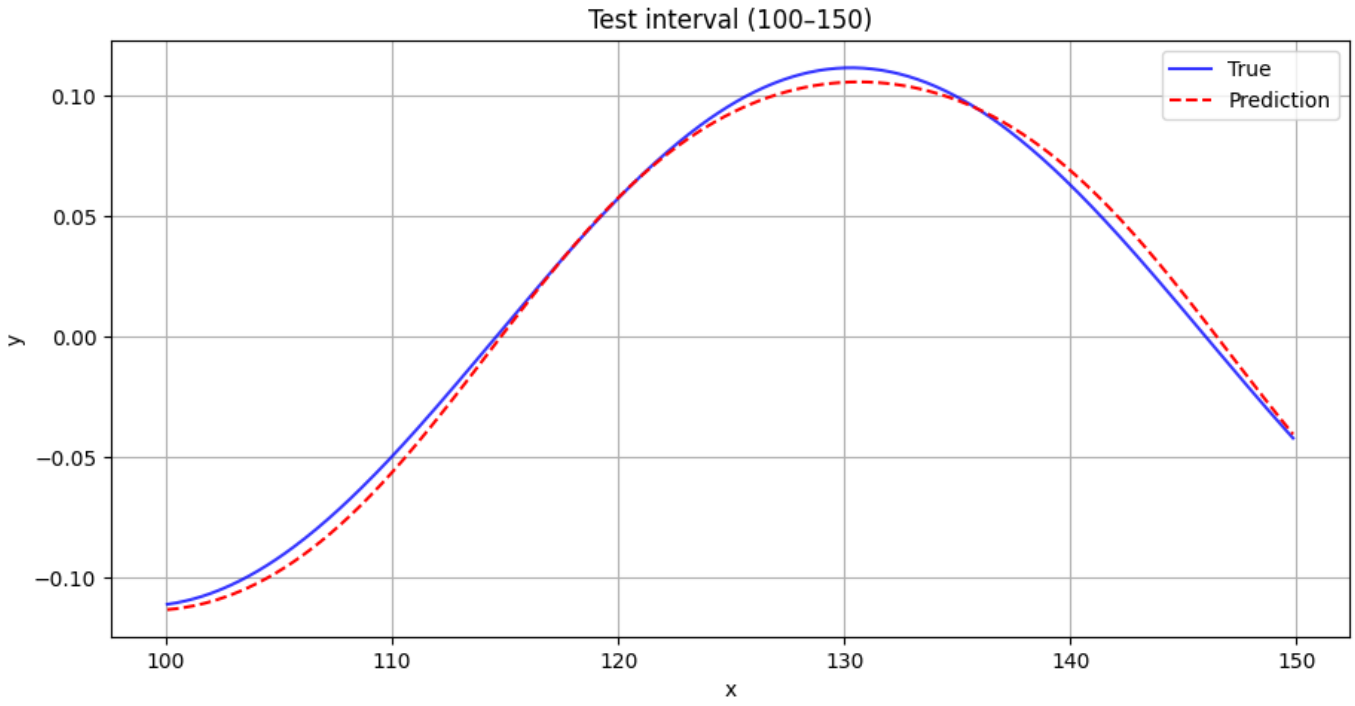
```
    print(f'Epoch [{epoch}/{epochs}], Train Loss: {train_loss.item():.6f}, Test Loss: {test_loss.item():.6f}')

model.eval()
with torch.no_grad():
  train_predictions = model(X_train)
  test_predictions = model(X_test)

Y_train_actual = scaler_y.inverse_transform(Y_train.numpy())
train_predictions_actual = scaler_y.inverse_transform(train_predictions.numpy())
Y_test_actual = scaler_y.inverse_transform(Y_test.numpy())
test_predictions_actual = scaler_y.inverse_transform(test_predictions.numpy())

plt.figure(figsize=(20, 12))

plt.subplot(2, 2, 1)
x_train_plot = x[start_train + n_inputs:end_train + n_inputs]
plt.plot(x_train_plot, Y_train_actual, 'b-', label='True', linewidth=1.5, alpha=0.8)
plt.plot(x_train_plot, train_predictions_actual, 'r--', label='Prediction', linewidth=1.5)
plt.title('Training interval (50–100)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(train_losses, 'g-', label='Train loss', linewidth=1.5)
plt.plot(test_losses, 'r-', label='Test loss', linewidth=1.5)
plt.title('Loss curves')
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.legend()
plt.grid(True)
plt.yscale('log')

plt.subplot(2, 2, 3)
x_test_plot = x[start_test + n_inputs:end_test + n_inputs]
plt.plot(x_test_plot, Y_test_actual, 'b-', label='True', linewidth=1.5, alpha=0.8)
plt.plot(x_test_plot, test_predictions_actual, 'r--', label='Prediction', linewidth=1.5)
plt.title('Test interval (100–150)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.scatter(Y_test_actual, test_predictions_actual, s=20, alpha=0.6)
```

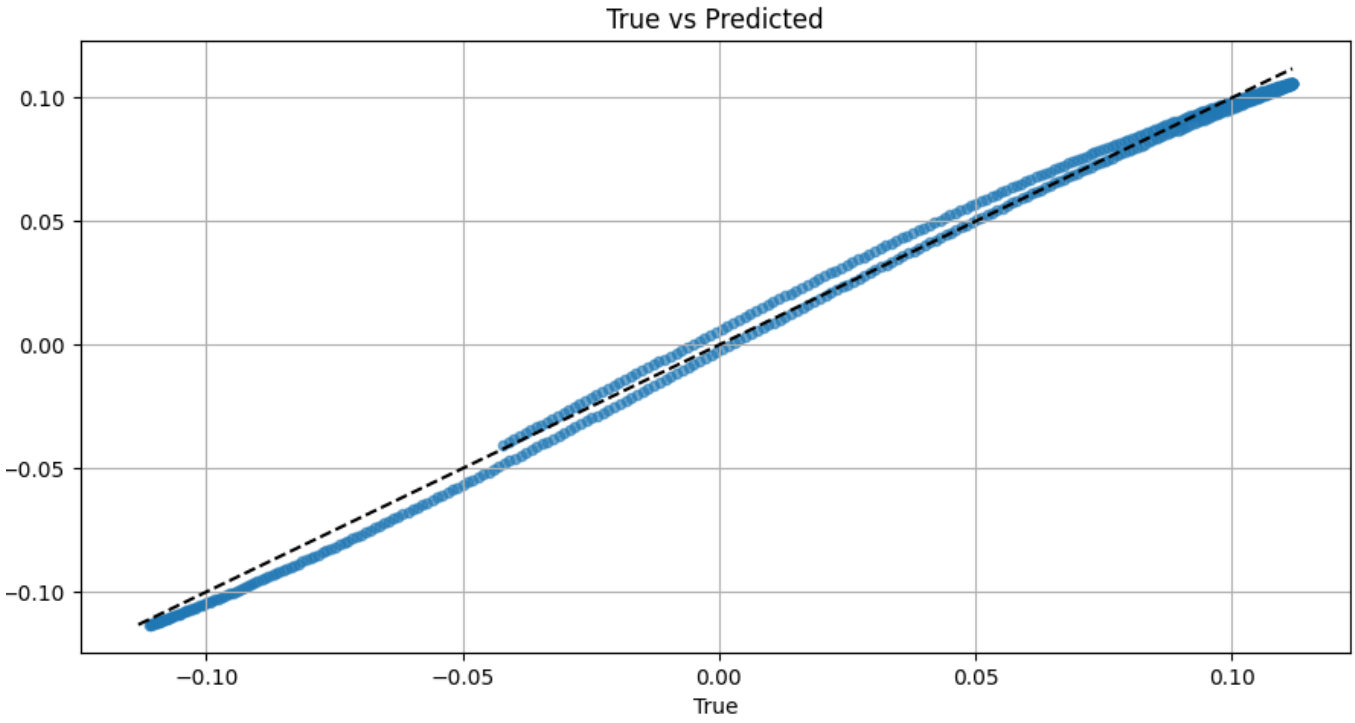
```
y_min = min(Y_test_actual.min(), test_predictions_actual.min())
y_max = max(Y_test_actual.max(), test_predictions_actual.max())
plt.plot([y_min, y_max], [y_min, y_max], 'k--')
plt.title('True vs Predicted')
plt.xlabel('True')
plt.ylabel('Predicted')
plt.grid(True)

plt.tight_layout(pad=3.0)
plt.show()

train_results = pd.DataFrame({
    'True': Y_train_actual.flatten(),
    'Predicted': train_predictions_actual.flatten(),
    'Error': (Y_train_actual.flatten() - train_predictions_actual.flatten())
})

test_results = pd.DataFrame({
    'True': Y_test_actual.flatten(),
    'Predicted': test_predictions_actual.flatten(),
    'Error': (Y_test_actual.flatten() - test_predictions_actual.flatten())
})

print(train_results.head(10))
print(test_results.head(10))
```

```
Epoch [0/500], Train Loss: 0.401234, Test Loss: 0.380093
Epoch [100/500], Train Loss: 0.082466, Test Loss: 0.072372
Epoch [200/500], Train Loss: 0.004531, Test Loss: 0.003995
Epoch [300/500], Train Loss: 0.000759, Test Loss: 0.000962
Epoch [400/500], Train Loss: 0.000414, Test Loss: 0.000632
       True  Predicted     Error
0 -0.019167  -0.027102  0.007935
1 -0.018064  -0.025864  0.007800
2 -0.016960  -0.024621  0.007661
3 -0.015854  -0.023373  0.007519
4 -0.014746  -0.022120  0.007374
5 -0.013637  -0.020863  0.007226
6 -0.012526  -0.019602  0.007076
7 -0.011414  -0.018337  0.006923
8 -0.010301  -0.017068  0.006767
9 -0.009187  -0.015796  0.006609
       True  Predicted     Error
0 -0.111045  -0.109122 -0.001923
1 -0.110909  -0.109057 -0.001852
2 -0.110762  -0.108985 -0.001777
3 -0.110604  -0.108906 -0.001699
4 -0.110436  -0.108818 -0.001617
5 -0.110256  -0.108724 -0.001532
6 -0.110065  -0.108621 -0.001444
7 -0.109863  -0.108511 -0.001352
8 -0.109650  -0.108393 -0.001257
9 -0.109426  -0.108268 -0.001158
```


Training interval (50–100)
True
Prediction
0.10
0.05
0.00
−0.05
−0.10
50
60
70
80
90
100
x


Loss curves
Train loss
Test loss
MSE
10^{-1}
10^{-2}
10^{-3}
0
100
200
300
400
500
Epoch





Вывод: построили, обучили и оценили многослойный перцептрон (MLP) для решения задачи классификациина практике сравнили работу несколько алгоритмов классификации.