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
Ярома А.И.
Проверил:
Крощенко А.А.

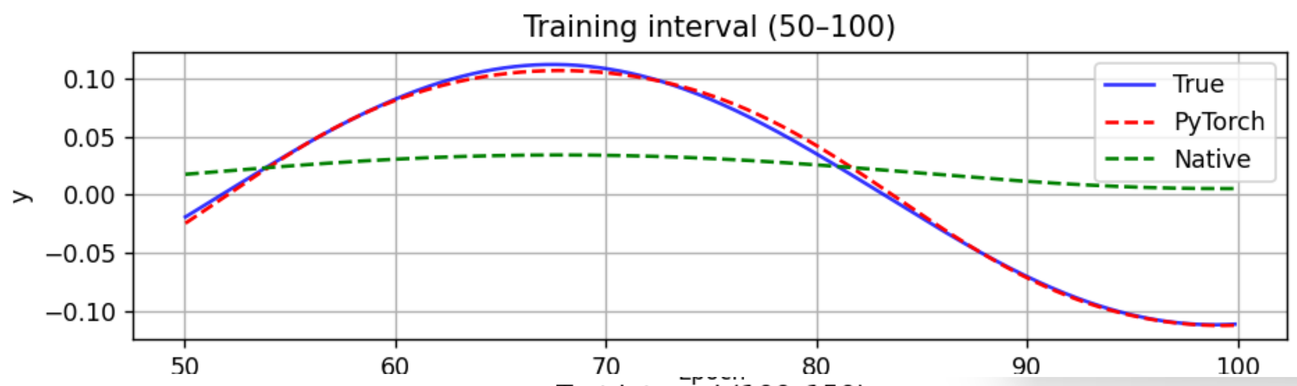
Брест 2025

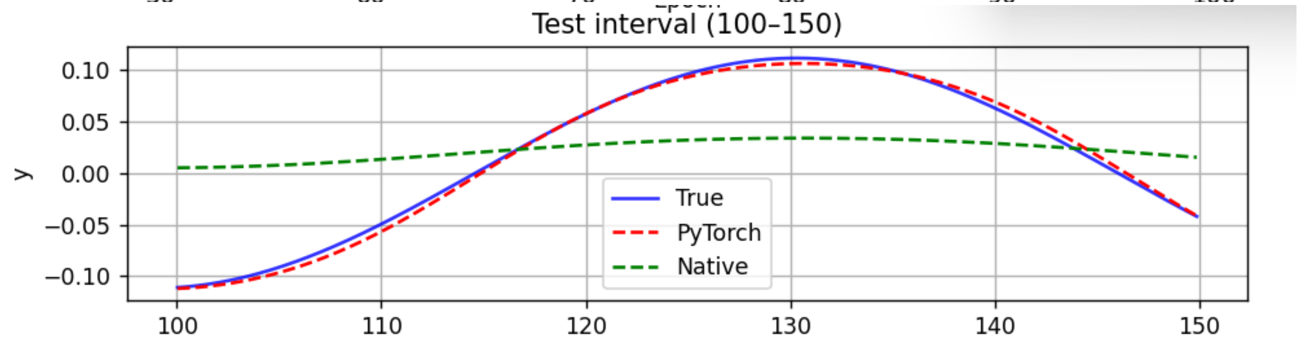
Цель: изучить нелинейные ИНС в задачах регрессии.

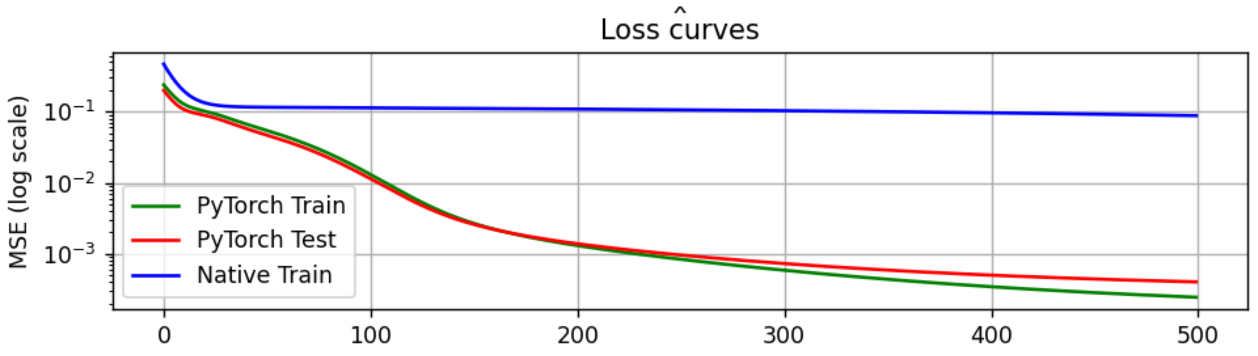
Вариант 12

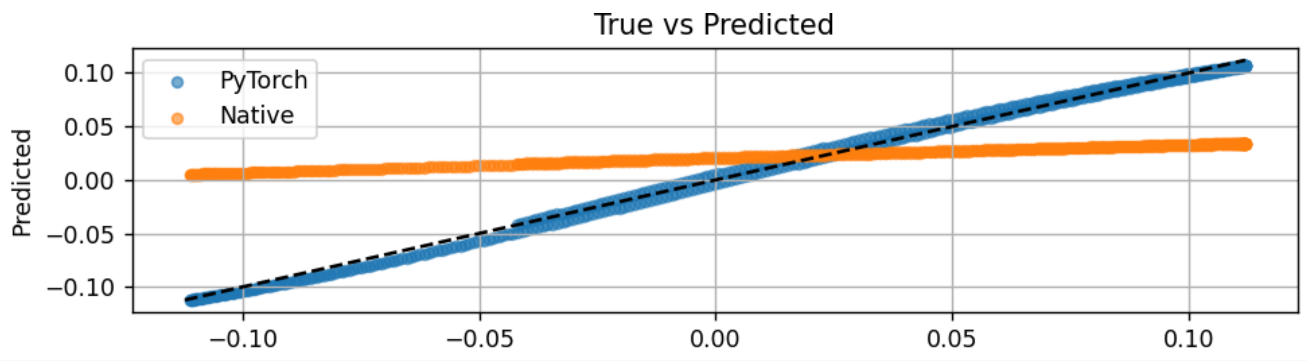
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler

# --------------------------
# ПАРАМЕТРЫ
# --------------------------

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

# --------------------------
# ФУНКЦИЯ ДЛЯ ГЕНЕРАЦИИ ДАННЫХ
# --------------------------

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)

# --------------------------
# СОЗДАНИЕ ДАТАСЕТА
# --------------------------

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)

# --------------------------
# МАСШТАБИРОВАНИЕ
# --------------------------

scaler_y = MinMaxScaler()

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

X_full, Y_full = 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_full[start_train:end_train]
Y_train = Y_full[start_train:end_train]
X_test  = X_full[start_test:end_test]
Y_test  = Y_full[start_test:end_test]

# --------------------------
# Torch-тензоры для PyTorch
# --------------------------

X_train_t = torch.FloatTensor(X_train)
Y_train_t = torch.FloatTensor(Y_train).reshape(-1, 1)
X_test_t  = torch.FloatTensor(X_test)
Y_test_t  = torch.FloatTensor(Y_test).reshape(-1, 1)

# ============================================================
#                    PYTORCH-МОДЕЛЬ
# ============================================================

class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__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))
        return self.output(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()

    pred = model(X_train_t)
    loss = criterion(pred, Y_train_t)
    loss.backward()
    optimizer.step()

    model.eval()
    with torch.no_grad():
        pred_test = model(X_test_t)
        loss_test = criterion(pred_test, Y_test_t)

    train_losses.append(loss.item())
    test_losses.append(loss_test.item())

    if epoch % 100 == 0:
        print(f"[PyTorch] Epoch {epoch}, Train={loss.item():.6f}, Test={loss_test.item():.6f}")

# Получаем предсказания
model.eval()
with torch.no_grad():
    train_pred_p = model(X_train_t).numpy()
    test_pred_p  = model(X_test_t).numpy()

Y_train_actual = scaler_y.inverse_transform(Y_train_t.numpy())
Y_test_actual  = scaler_y.inverse_transform(Y_test_t.numpy())
train_pred_p_actual = scaler_y.inverse_transform(train_pred_p)
test_pred_p_actual  = scaler_y.inverse_transform(test_pred_p)

# ===============================================================
#                    НАТИВНАЯ (NUMPY) МОДЕЛЬ
# ===============================================================

class NativeNN:
    def __init__(self, input_size, hidden_size, output_size, lr=0.05):
        self.lr = lr
        self.W1 = np.random.randn(input_size, hidden_size) * 0.1
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * 0.1
        self.b2 = np.zeros((1, output_size))

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

    def sigmoid_deriv(self, x):
        return x * (1 - x)

    def forward(self, X):
        self.Z1 = X @ self.W1 + self.b1
        self.A1 = self.sigmoid(self.Z1)
        self.Z2 = self.A1 @ self.W2 + self.b2
        return self.Z2

    def backward(self, X, y, out):
        m = len(y)

        dZ2 = out - y
        dW2 = self.A1.T @ dZ2 / m
        db2 = np.sum(dZ2, axis=0, keepdims=True) / m

        dA1 = dZ2 @ self.W2.T
        dZ1 = dA1 * self.sigmoid_deriv(self.A1)
        dW1 = X.T @ dZ1 / m
        db1 = np.sum(dZ1, axis=0, keepdims=True) / m

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

    def train(self, X, y, epochs):
        losses = []
        for epoch in range(epochs):
            out = self.forward(X)
            loss = np.mean((out - y) ** 2)
            losses.append(loss)

            self.backward(X, y, out)

            if epoch % 100 == 0:
                print(f"[Native] Epoch {epoch}, Loss={loss:.6f}")

        return losses

    def predict(self, X):
        return self.forward(X)

native_model = NativeNN(n_inputs, n_hidden, 1, lr=0.05)

native_losses = native_model.train(X_train, Y_train.reshape(-1, 1), epochs)

native_train_pred = native_model.predict(X_train)
native_test_pred  = native_model.predict(X_test)

native_train_pred_actual = scaler_y.inverse_transform(native_train_pred)
native_test_pred_actual  = scaler_y.inverse_transform(native_test_pred)



# ============================================================
#                     ВИЗУАЛИЗАЦИЯ (ПОЛНАЯ)
# ============================================================

plt.figure(figsize=(20, 25))

# -------------------- 1. TRAINING interval --------------------
plt.subplot(4, 1, 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_pred_p_actual, 'r--', label='PyTorch', linewidth=1.5)
plt.plot(x_train_plot, native_train_pred_actual, 'g--', label='Native', linewidth=1.5)

plt.title("Training interval (50–100)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)

# -------------------- 2. LOSS CURVES --------------------
plt.subplot(4, 1, 2)

plt.plot(train_losses, 'g-', label='PyTorch Train', linewidth=1.5)
plt.plot(test_losses, 'r-', label='PyTorch Test', linewidth=1.5)
plt.plot(native_losses, 'b-', label='Native Train', linewidth=1.5)

plt.yscale("log")
plt.title("Loss curves")
plt.xlabel("Epoch")
plt.ylabel("MSE (log scale)")
plt.grid(True)
plt.legend()

# --------------------- 3. TEST interval -------------------------
plt.subplot(4, 1, 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_pred_p_actual, 'r--', label='PyTorch', linewidth=1.5)
plt.plot(x_test_plot, native_test_pred_actual, 'g--', label='Native', linewidth=1.5)

plt.title("Test interval (100–150)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)

# --------------------- 4. TRUE vs PREDICTED --------------------
plt.subplot(4, 1, 4)

plt.scatter(Y_test_actual, test_pred_p_actual, s=20, alpha=0.6, label="PyTorch")
plt.scatter(Y_test_actual, native_test_pred_actual, s=20, alpha=0.6, label="Native")

y_min = min(Y_test_actual.min(), test_pred_p_actual.min(), native_test_pred_actual.min())
y_max = max(Y_test_actual.max(), test_pred_p_actual.max(), native_test_pred_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.legend()

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









Вывод: построили, обучили и оценили многослойный перцептрон (MLP) для

решения задачи классификациина практике сравнили работу несколько алгоритмов классификации.