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 torch
import torch.nn as nn
import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(42)
torch.manual_seed(42)

a = 0.4
b = 0.6
c = 0.06
d = 0.6

INPUT_SIZE = 10
HIDDEN_SIZE = 4

def generate_series(N=800):
    y = np.zeros(N, dtype=float)
    u = np.random.uniform(-1.0, 1.0, size=N)
    y[0] = 0.0
    y[1] = 0.0
    for k in range(2, N):
        y[k] = (
            a * y[k-1] / (1.0 + y[k-1]**2)
            + b * y[k-2]
            + c * np.sin(u[k-1])
            + d * u[k-2]
        )
    return y

def make_dataset(series, window=INPUT_SIZE):
    X, T = [], []
    for k in range(window, len(series)):
        X.append(series[k-window:k])
        T.append(series[k])
    X = np.array(X, dtype=np.float32)
    T = np.array(T, dtype=np.float32).reshape(-1, 1)
    return X, T
```

```
def smooth(y, window=9):
    kernel = np.ones(window, dtype=float) / float(window)
    return np.convolve(y, kernel, mode="same")

series = generate_series(N=800)
X, T = make_dataset(series, window=INPUT_SIZE)

train_size = int(0.7 * len(X))
X_train, T_train = X[:train_size], T[:train_size]
X_test,  T_test  = X[train_size:], T[train_size:]

X_train_t = torch.from_numpy(X_train)
T_train_t = torch.from_numpy(T_train)
X_test_t  = torch.from_numpy(X_test)
T_test_t  = torch.from_numpy(T_test)

class Net(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        x = self.sigmoid(self.fc1(x))
        return self.fc2(x)

net = Net(INPUT_SIZE, HIDDEN_SIZE)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.01)

epochs = 500
loss_history = []

for _ in range(epochs):
    net.train()
    optimizer.zero_grad()
    outputs = net(X_train_t)
    loss = criterion(outputs, T_train_t)
    loss.backward()
    optimizer.step()
    loss_history.append(float(loss.item()))

net.eval()
with torch.no_grad():
    train_pred = net(X_train_t).numpy().flatten()
    test_pred  = net(X_test_t).numpy().flatten()
```

```
train_true = T_train.flatten()
test_true  = T_test.flatten()

train_residuals = train_pred - train_true
test_residuals  = test_pred - test_true

plt.figure(figsize=(16, 8))
plt.plot(smooth(train_true), label="True (train)", linewidth=3)
plt.plot(smooth(train_pred), label="Pred (train)", linewidth=3)
plt.title("Training: true vs prediction")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 8))
plt.plot(smooth(test_true), label="True (test)", linewidth=3)
plt.plot(smooth(test_pred), label="Pred (test)", linewidth=3)
plt.title("Test: true vs prediction")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 6))
plt.plot(smooth(np.array(loss_history), 15), linewidth=3)
plt.title("Training loss (MSE)")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 6))
plt.plot(train_residuals, linewidth=1.5)
plt.title("Residuals (train)")
plt.xlabel("Index")
plt.ylabel("Error")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 6))
```

```
plt.plot(test_residuals, linewidth=1.5)
plt.title("Residuals (test)")
plt.xlabel("Index")
plt.ylabel("Error")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

train_table = pd.DataFrame({
    "True": train_true,
    "Pred": train_pred,
    "Error": train_residuals
})

test_table = pd.DataFrame({
    "True": test_true,
    "Pred": test_pred,
    "Error": test_residuals
})

print("TRAIN first 10 rows:")
print(train_table.head(10))
print("\nTEST first 10 rows:")
print(test_table.head(10))
```

```
First 10 rows: TRAIN
       True      Pred     Error
0 -0.133021 -0.197469 -0.064447
1  0.257501  0.097939 -0.159562
2 -0.510069 -0.016433  0.493636
3  0.593506 -0.160969 -0.754475
4  0.235805 -0.108270 -0.344075
5  0.064608  0.509825  0.445217
6 -0.250095  0.158031  0.408126
7 -0.458194 -0.185390  0.272804
8 -0.533473 -0.371697  0.161776
9 -0.419464 -0.466286 -0.046822

First 10 rows: TEST
       True      Pred     Error
0 -0.063387  0.378151  0.441537
1 -0.173156  0.229158  0.402315
2 -0.705489 -0.216867  0.488622
3 -0.460874 -0.386722  0.074152
4 -0.480276 -0.498972 -0.018697
5 -0.569418 -0.389353  0.180065
6 -0.491807 -0.466577  0.025230
7 -0.032999 -0.503037 -0.470037
8 -0.488684 -0.294611  0.194073
9 -0.128562 -0.243905 -0.115343
```

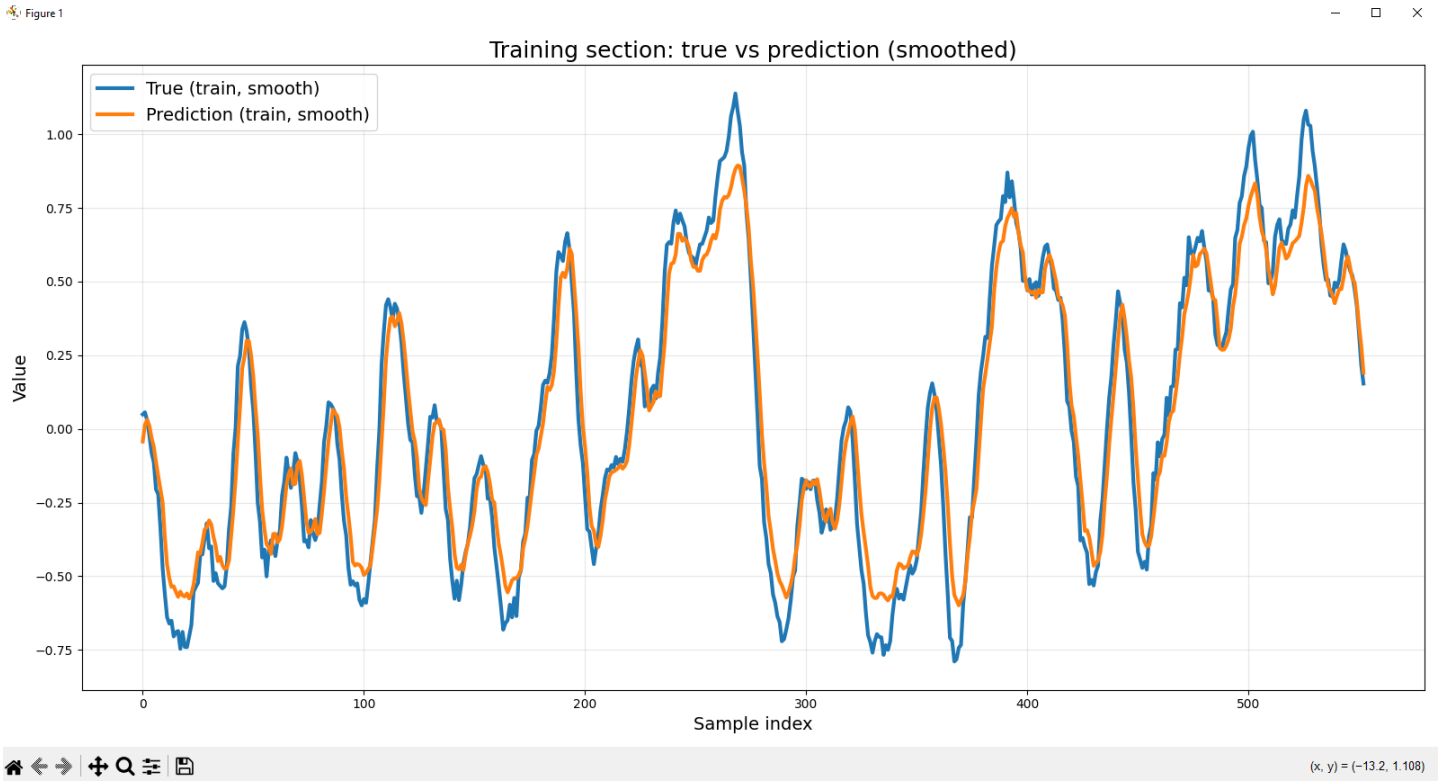


Figure 1
Training section: true vs prediction (smoothed)
True (train, smooth)
Prediction (train, smooth)
Value
Sample index
(x, y) = (−13.2, 1.108)

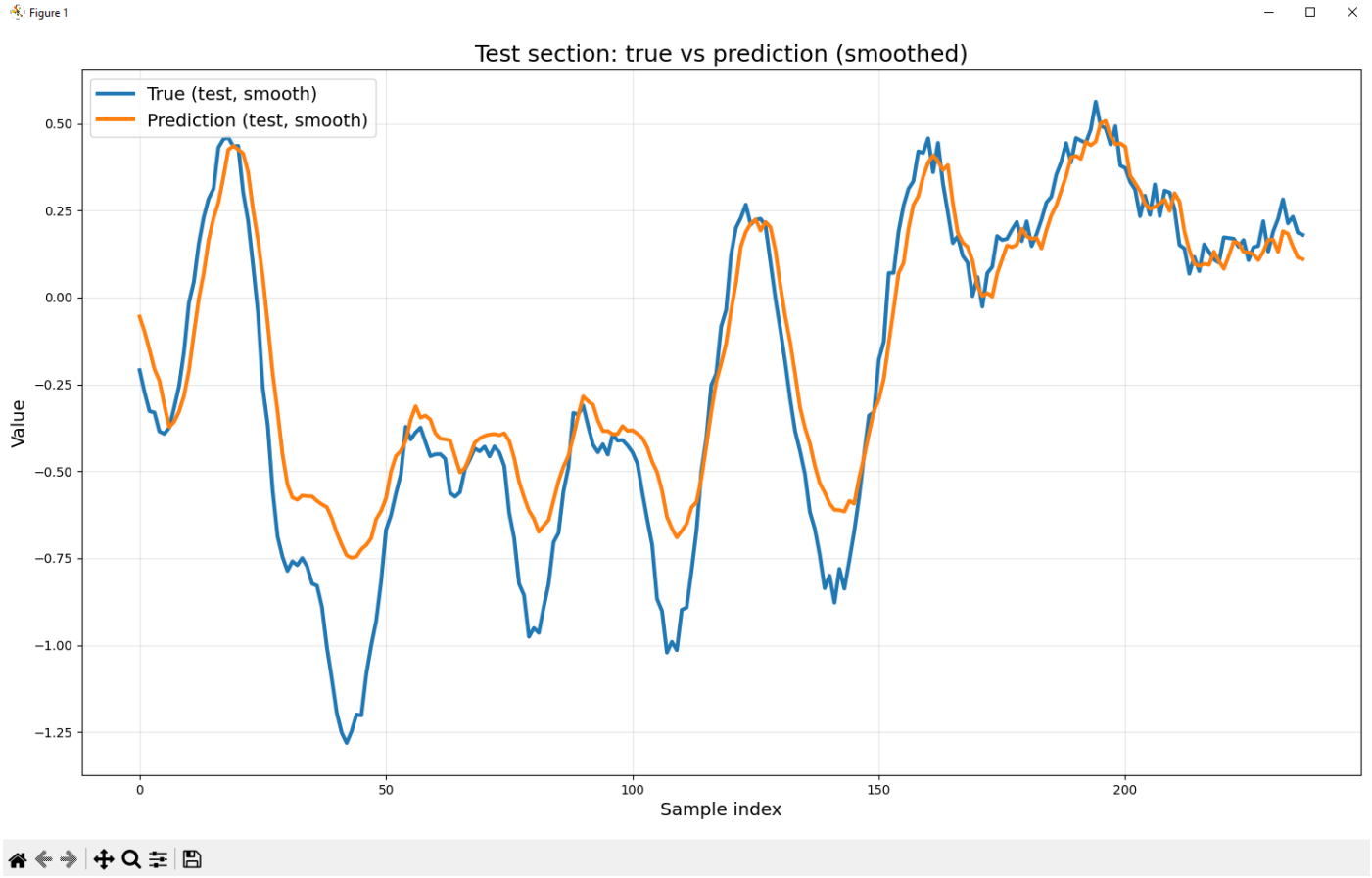


Figure 1
Test section: true vs prediction (smoothed)
True (test, smooth)
Prediction (test, smooth)
Value
Sample index

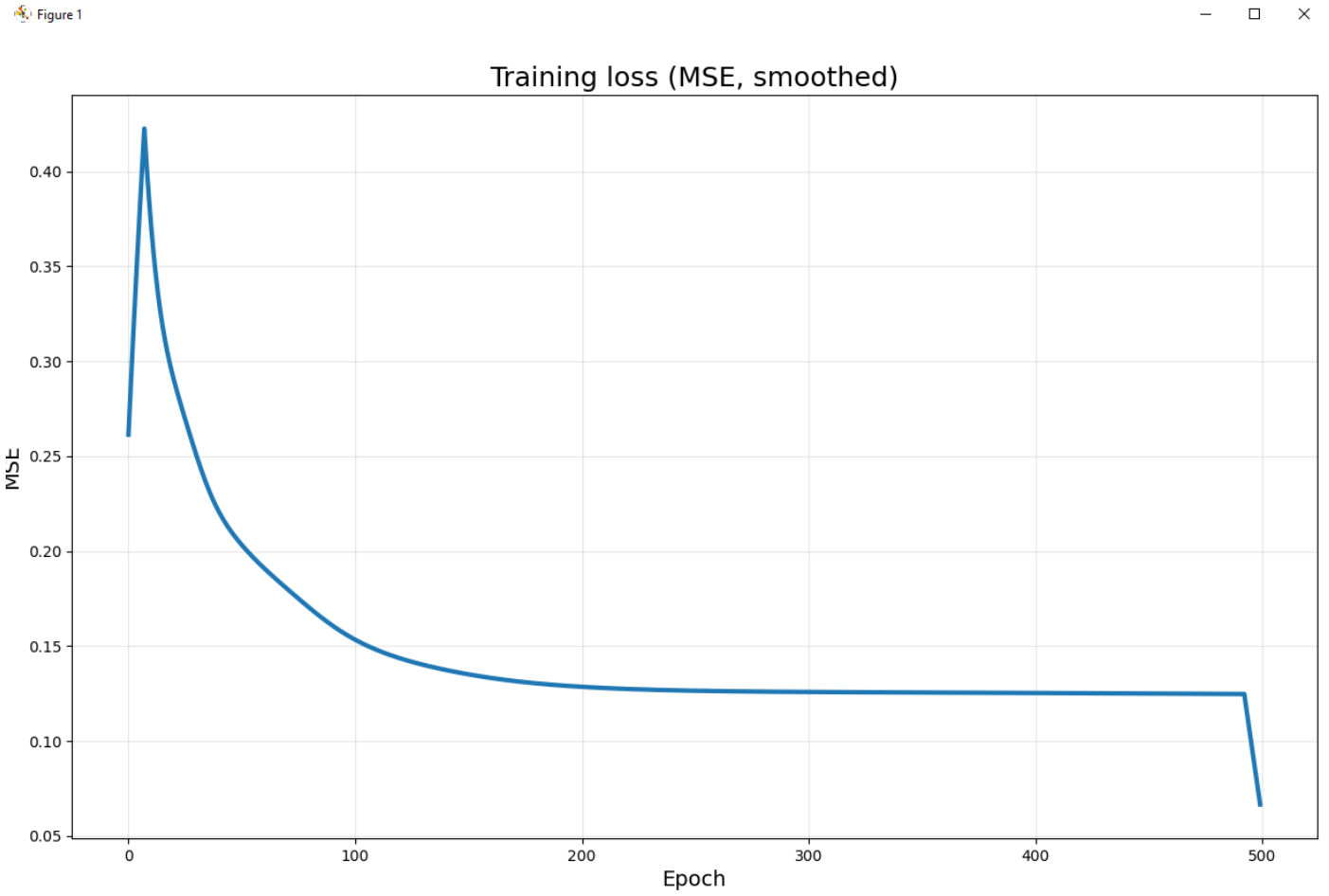


Figure 1
Training loss (MSE, smoothed)
MSE
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
100
200
300
400
500
Epoch

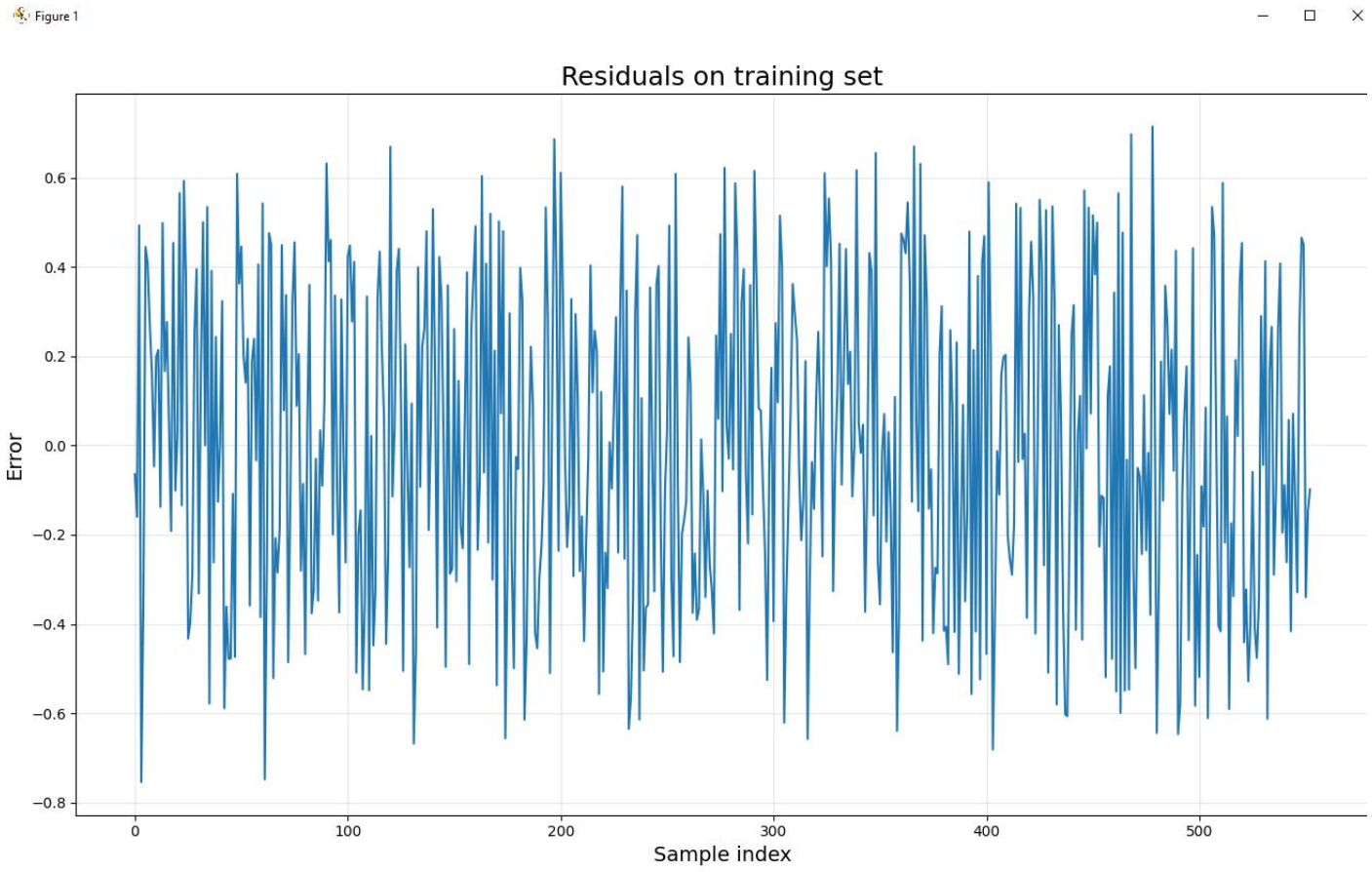
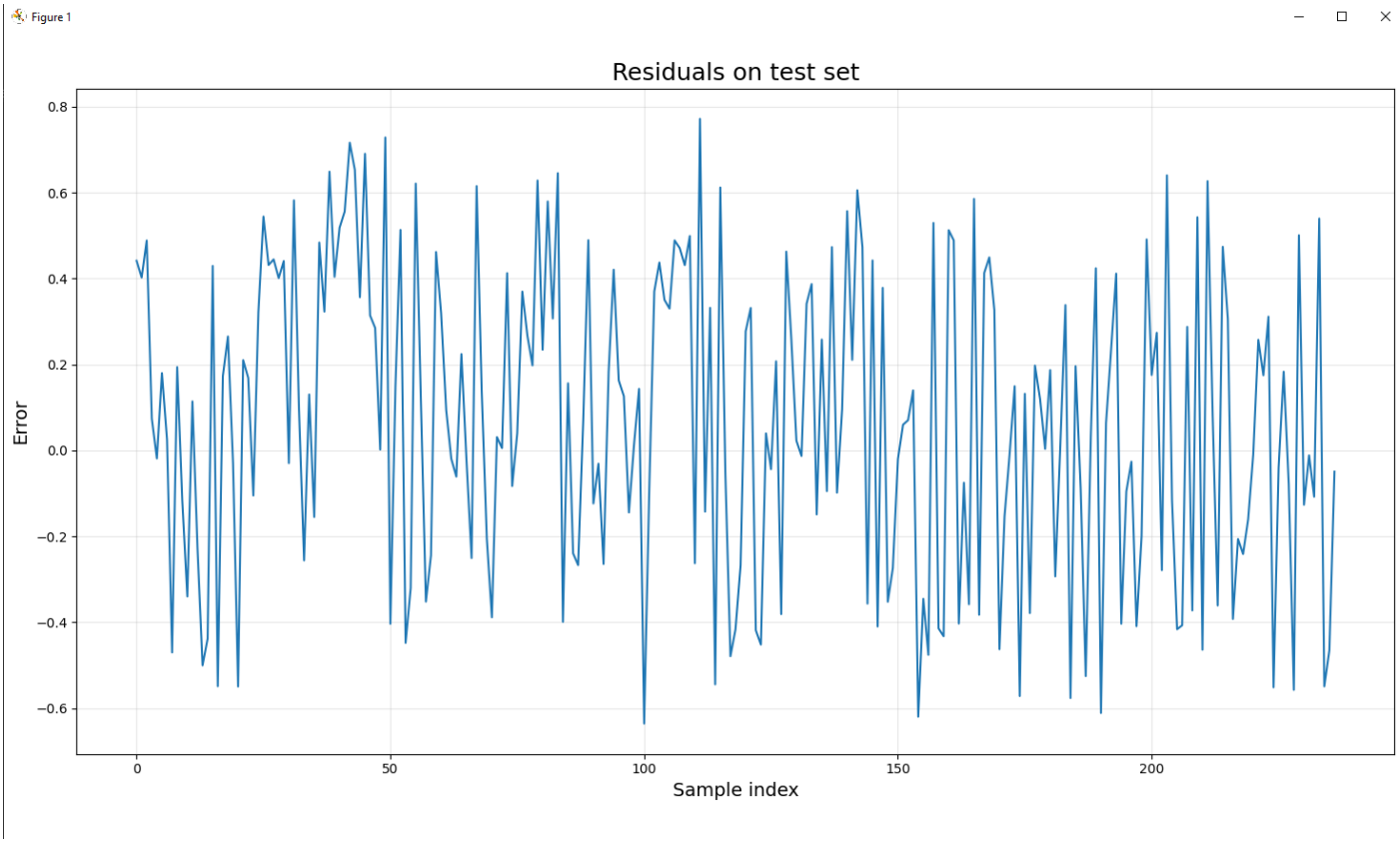


Figure 1
Residuals on training set
Error
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
100
200
300
400
500
Sample index



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