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

Лабораторная работа №6
По дисциплине: «ОМО»
Тема : “Реккурентные нейронные сети”

Выполнил:
Студент 3-го курса
Группы АС-66
Цеван К.А.
Проверил:
Крощенко А.А.

Брест 2025

Цель: изучить реккурентные нейронные сети.

**Вариант 12**

Задание:

1. По вариантам предыдущей лабораторной работы реализовать предложенный вариант рекуррентной нейронной сети. Сравнить полученные результаты с ЛР 5.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0.1 | 0.1 | 0.05 | 0.1 | 6 | 2 | Элмана |

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
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(-100.0, 300.0, 4000)
y = target_function(x)

y_min = y.min()
y_max = y.max()
y_scaled = (y - y_min) / (y_max - y_min)

def create_dataset(series, n_inputs):
    X, T = [], []
    for i in range(len(series) - n_inputs):
        X.append(series[i:i + n_inputs])
        T.append(series[i + n_inputs])
    X = np.array(X, dtype=np.float32)
    T = np.array(T, dtype=np.float32)
    X = X.reshape(-1, n_inputs, 1)
    return X, T

X_all, T_all = create_dataset(y_scaled, n_inputs)

start_train = np.where(x >= 50.0)[0][0] - n_inputs
end_train = np.where(x <= 100.0)[0][-1] - n_inputs
```

```
start_forecast_idx = np.where(x >= 100.0)[0][0]
end_forecast_idx = np.where(x <= 150.0)[0][-1]
n_forecast = end_forecast_idx - start_forecast_idx + 1

X_train = X_all[start_train:end_train]
T_train = T_all[start_train:end_train]

X_train_t = torch.from_numpy(X_train)
T_train_t = torch.from_numpy(T_train.reshape(-1, 1))

class ElmanRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.in_layer = nn.Linear(input_size, hidden_size, bias=True)
        self.rec_layer = nn.Linear(hidden_size, hidden_size, bias=False)
        self.out_layer = nn.Linear(hidden_size, output_size, bias=True)
        self.act = nn.Sigmoid()
        self.hidden_size = hidden_size

    def forward(self, x):
        batch_size, seq_len, _ = x.shape
        h = torch.zeros(batch_size, self.hidden_size, device=x.device)
        for t in range(seq_len):
            x_t = x[:, t, :]
            h = self.act(self.in_layer(x_t) + self.rec_layer(h))
        y = self.out_layer(h)
        return y

model = ElmanRNN(1, n_hidden, 1)

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

epochs = 500
train_losses = []

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_t)
    loss = criterion(outputs, T_train_t)
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())

model.eval()
```

```
with torch.no_grad():
    train_pred_scaled = model(X_train_t).numpy().flatten()

train_true_scaled = T_train
train_true = train_true_scaled * (y_max - y_min) + y_min
train_pred = train_pred_scaled * (y_max - y_min) + y_min

idx_train_targets = np.arange(start_train + n_inputs, end_train + n_inputs)
x_train_plot = x[idx_train_targets]

start_window = y_scaled[start_forecast_idx - n_inputs:start_forecast_idx].astype(np.float32)
forecast_scaled = []

with torch.no_grad():
    window = start_window.copy()
    for k in range(n_forecast):
        x_seq = torch.from_numpy(window.reshape(1, n_inputs, 1))
        y_hat_scaled = model(x_seq).item()
        forecast_scaled.append(y_hat_scaled)
        window = np.roll(window, -1)
        window[-1] = y_hat_scaled

forecast_scaled = np.array(forecast_scaled, dtype=np.float32)
y_forecast = forecast_scaled * (y_max - y_min) + y_min

y_true_forecast = y[start_forecast_idx:end_forecast_idx + 1]
x_forecast = x[start_forecast_idx:end_forecast_idx + 1]

error_forecast = y_forecast - y_true_forecast

plt.figure(figsize=(16, 8))
plt.plot(x_train_plot, train_true, 'b-', label="Train true")
plt.plot(x_train_plot, train_pred, 'r--', label="Train prediction")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(16, 8))
plt.plot(x_forecast, y_true_forecast, 'g-', label="True")
plt.plot(x_forecast, y_forecast, 'r--', label="Forecast")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 5))
```

```
plt.plot(train_losses)
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 5))
plt.plot(x_forecast, error_forecast)
plt.grid(True)
plt.tight_layout()
plt.show()

results_forecast = pd.DataFrame({
    "x": x_forecast,
    "True": y_true_forecast,
    "Forecast": y_forecast,
    "Error": error_forecast
})

print(results_forecast.head(10).to_string(index=False))
print("\nMAE:", np.mean(np.abs(error_forecast)))
print("Max error:", np.max(np.abs(error_forecast)))
```

```
         x      True  Forecast    Error
100.050013 -0.111045 -0.088956 0.022089
100.150038 -0.110909 -0.071020 0.039889
100.250063 -0.110762 -0.055703 0.055059
100.350088 -0.110604 -0.042543 0.068062
100.450113 -0.110436 -0.031038 0.079397
100.550138 -0.110256 -0.020877 0.089379
100.650163 -0.110065 -0.011861 0.098203
100.750188 -0.109863 -0.003861 0.106002
100.850213 -0.109650  0.003220 0.112869
100.950238 -0.109426  0.009455 0.118881

MAE: 0.060279174129419986
Max error: 0.14781393283436556
```

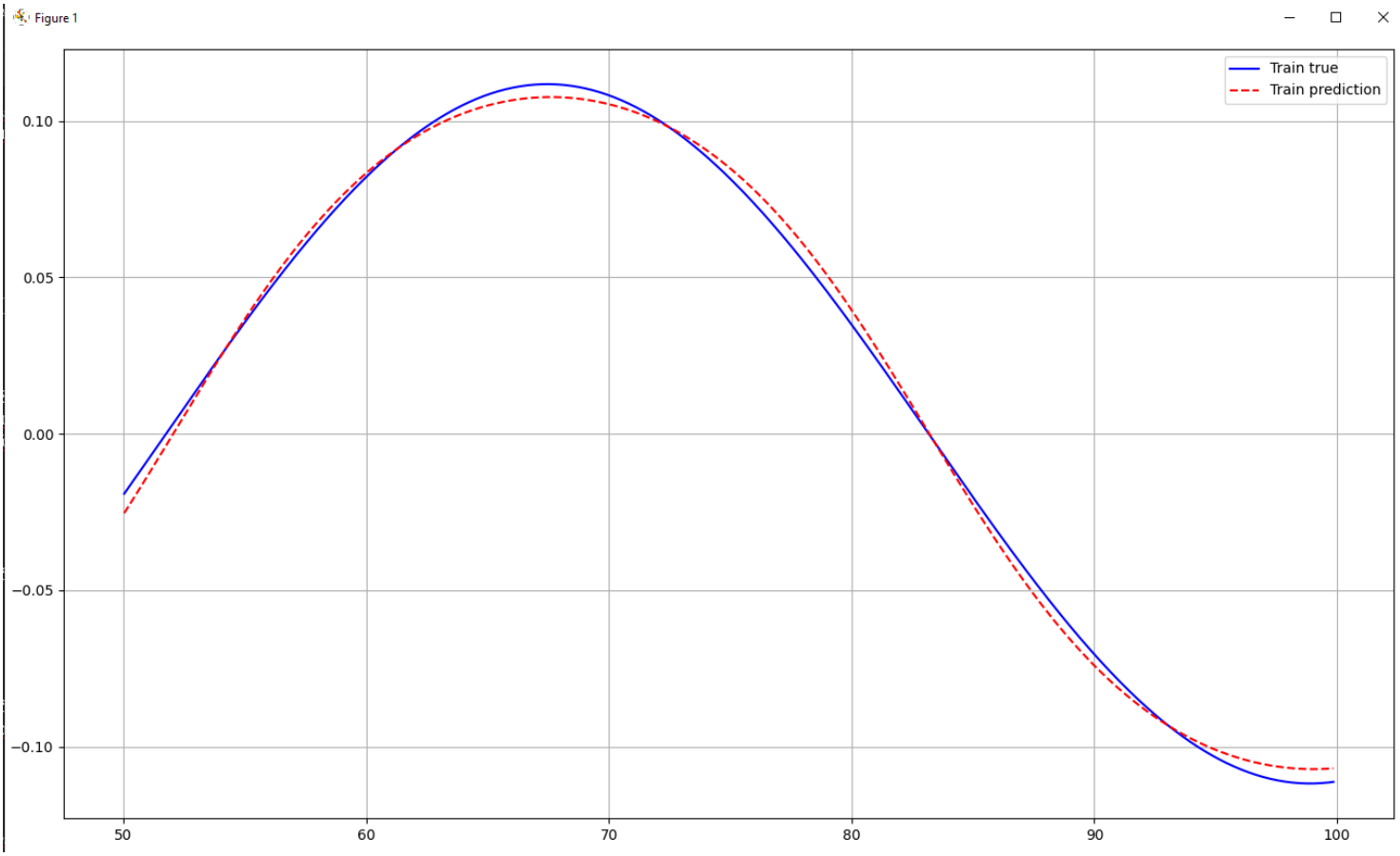
Figure 1
Train true
Train prediction
0.10
0.05
0.00
−0.05
−0.10
50
60
70
80
90
100

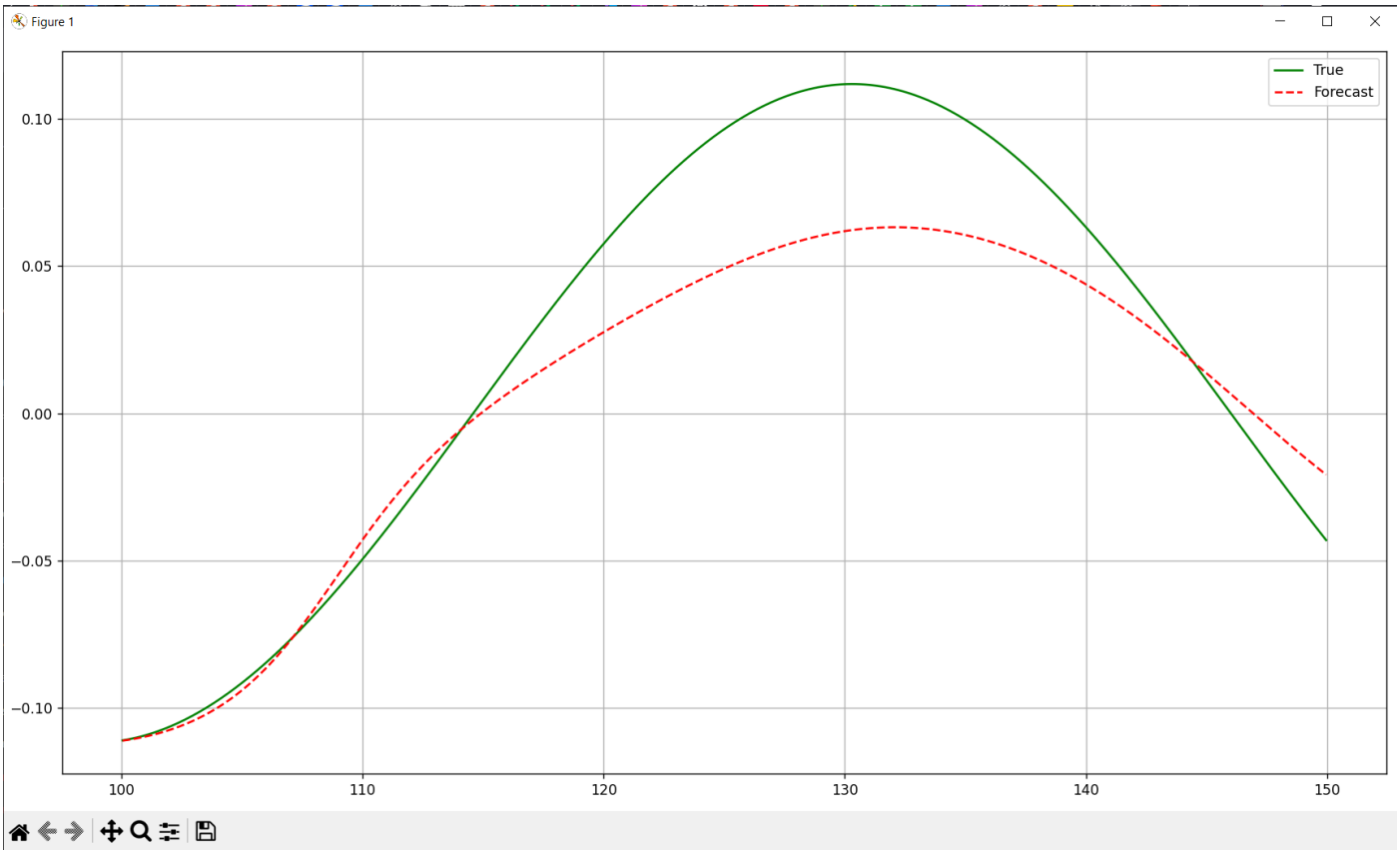
Figure 1
True
Forecast
0.10
0.05
0.00
−0.05
−0.10
100
110
120
130
140
150

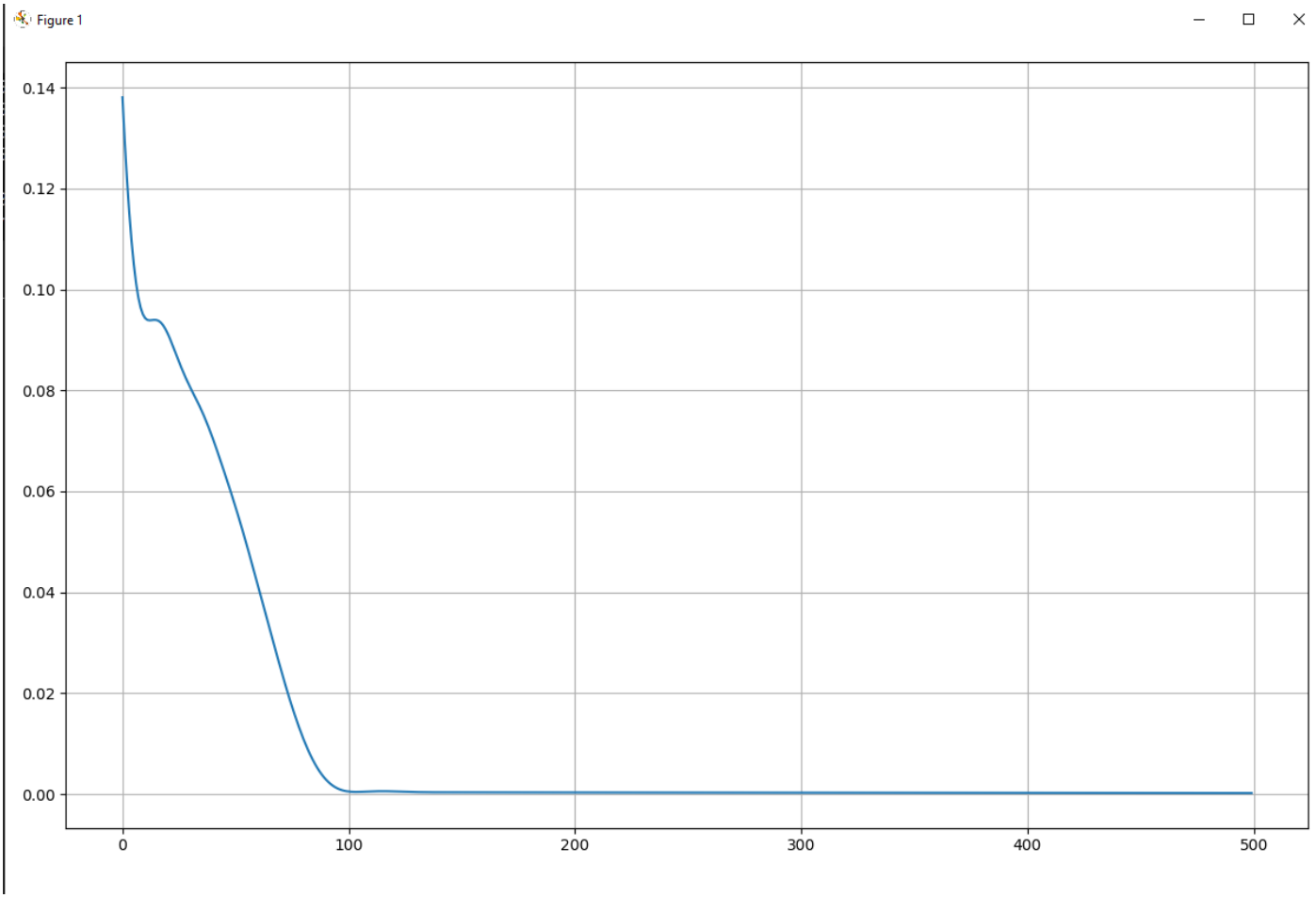
Figure 1
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
100
200
300
400
500

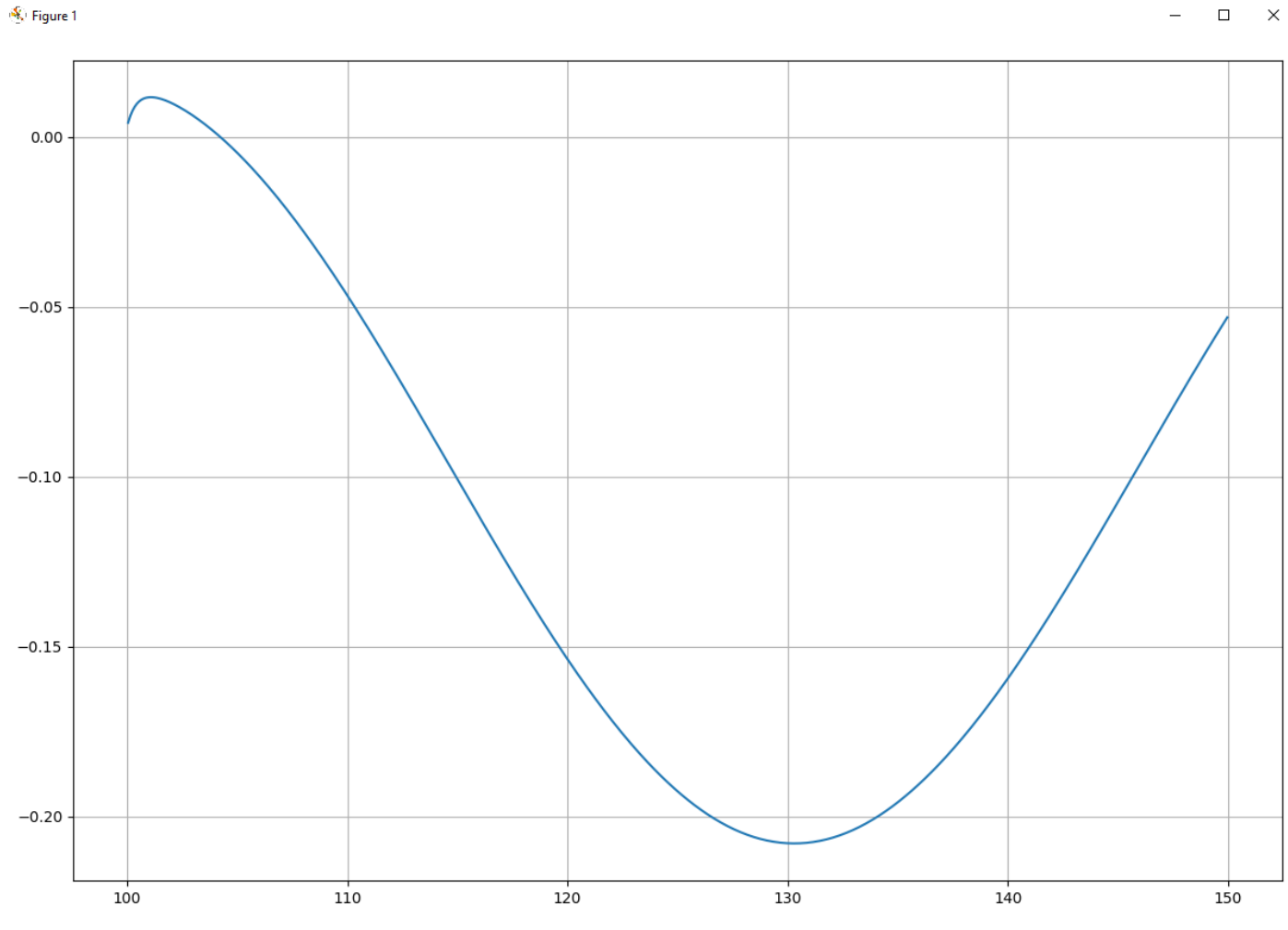
Figure 1
0.00
−0.05
−0.10
−0.15
−0.20
100
110
120
130
140
150

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