МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №6

Специальность АС-66

Выполнила
Е. С. Неруш,
студент группы АС-66

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«___» _____2025 г.

Брест 2025

Цель работы: На практике изучить Рекуррентные нейронные сети.

Задачи:

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

Варианты заданий приведены в следующей таблице:

Вариант 10

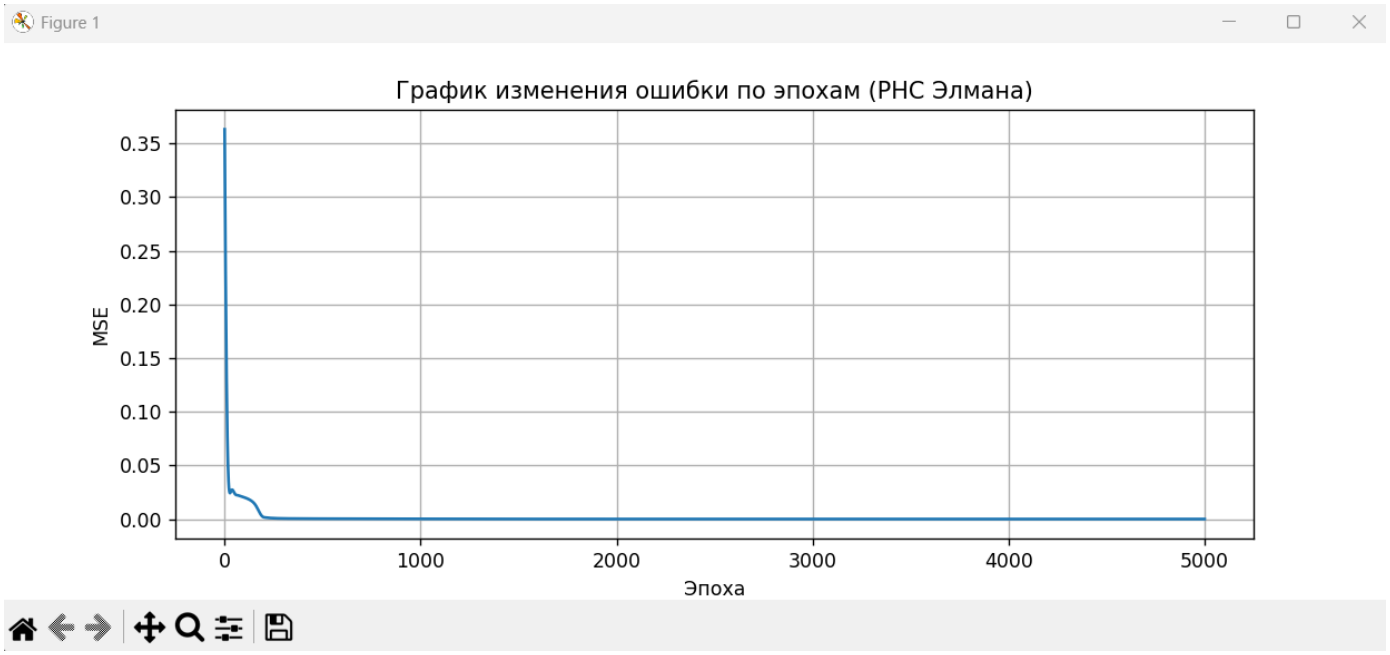
a=0.2, b=0.4, c=0.09, d=0.4
Кол-во входов ИНС - 6
Кол-во НЭ в скрытом слое - 2
Тип РНС - Элмана

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

# 1. Генерация обучающих и тестовых данных
def target_function(x, a=0.2, b=0.4, c=0.09, d=0.4):
    return a * torch.cos(b * x) + c * torch.sin(d * x)

a, b, c, d = 0.2, 0.4, 0.09, 0.4
input_size = 1
hidden_size = 2
window_size = 6
num_samples = 200
train_ratio = 0.8

x = torch.linspace(0, 20, num_samples).reshape(-1, 1)
y = target_function(x, a, b, c, d)

X = []
y_target = []
for i in range(num_samples - window_size):
    X.append(y[i:i+window_size])
    y_target.append(y[i+window_size])
X = torch.stack(X)
y_target = torch.stack(y_target)

split_idx = int(X.shape[0] * train_ratio)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y_target[:split_idx], y_target[split_idx:]

print("Данные сгенерированы для функции y = a*cos(bx) + c*sin(dx)")
print(f"Размер обучающей выборки: {X_train.shape}, тестовой: {X_test.shape}")

# 2. Архитектура РНС Элмана
class ElmanRNN(nn.Module):
    def __init__(self, input_dim, hidden_dim):
```

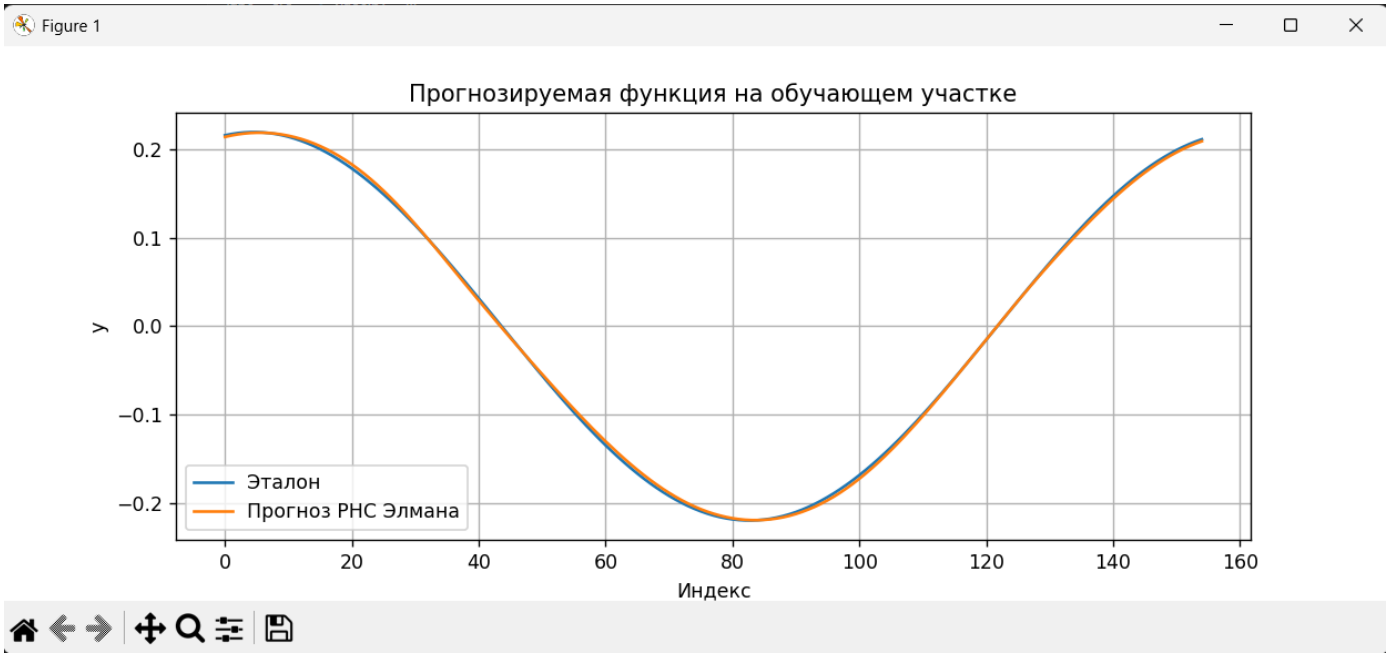
```
        super().__init__()
        self.rnn = nn.RNN(input_dim, hidden_dim, nonlinearity="tanh",
batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        out, _ = self.rnn(x)
        out = out[:, -1, :]
        out = self.fc(out)
        return out

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

# 3. Обучение модели
losses = []
epochs = 5000

for epoch in range(epochs):
    model.train()
    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    losses.append(loss.item())

print("Обучение завершено. Минимальная ошибка:", min(losses))
```

```
Задание 3: Данные сгенерированы для функции y = a*cos(bx) + c*sin(dx)
Размер обучающей выборки: torch.Size([155, 6, 1]), тестовой: torch.Size([39, 6, 1])
Обучение завершено. Минимальная ошибка: 3.419484983169241e-06
```

```
# 4. График ошибки по эпохам
plt.figure(figsize=(10, 4))
plt.plot(losses)
plt.title("График изменения ошибки по эпохам (РНС Элмана)")
plt.xlabel("Эпоха")
plt.ylabel("MSE")
plt.grid(True)
plt.show()
```




```
# 5. График прогнозируемой функции на обучающем участке
model.eval()
with torch.no_grad():
    y_train_pred = model(X_train)

plt.figure(figsize=(10, 4))
plt.plot(x[:split_idx], y_train, label="Эталон")
plt.plot(x[:split_idx], y_train_pred, label="Прогноз РНС Элмана")
plt.title("Прогнозируемая функция на обучающем участке")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```




```
# 6. Таблица результатов обучения
train_table = pd.DataFrame({
    "Эталонное значение": y_train.squeeze().numpy(),
```

```
    "Полученное значение": y_train_pred.squeeze().numpy(),
    "Отклонение": (y_train_pred - y_train).squeeze().numpy()
})
print("\nРезультаты обучения (первые строки):")
print(train_table.head())
```

```
Результаты обучения (первые строки):
   Эталонное значение  Полученное значение  Отклонение
0            0.215709             0.163006   -0.052702
1            0.217127             0.163350   -0.053777
2            0.218194             0.163625   -0.054569
3            0.218909             0.163833   -0.055075
4            0.219269             0.163976   -0.055294
```

```
# 7. Таблица результатов прогнозирования
with torch.no_grad():
    y_test_pred = model(X_test)
    test_table = pd.DataFrame({
        "Эталонное значение": y_test.squeeze().numpy(),
        "Полученное значение": y_test_pred.squeeze().numpy(),
        "Отклонение": (y_test_pred - y_test).squeeze().numpy()
    })
print("\nРезультаты прогнозирования (первые строки):")
print(test_table.head())
```

```
Результаты прогнозирования (первые строки):
   Эталонное значение  Полученное значение  Отклонение
0            0.213356             0.162454   -0.050902
1            0.215225             0.162892   -0.052333
2            0.216746             0.163256   -0.053490
3            0.217917             0.163551   -0.054365
4            0.218735             0.163779   -0.054956
```

Вывод: В ходе лабораторной работы №6 реализована рекуррентная сеть Элмана с функциями активации tanh и сигмоидой. Сеть обучалась на окнах из прошлых значений функции и показала способность учитывать временные зависимости. В отличие от многослойного перцептрона из лабораторной работы №5, который аппроксимировал функцию как статическую зависимость, РНС Элмана использует скрытое состояние и обладает «памятью» о предыдущих шагах. Это позволяет ей лучше моделировать динамику временного ряда. Однако при малом числе нейронов и длинных интервалах возникают трудности с точностью прогноза из-за затухающих градиентов. Таким образом, MLP проще и эффективен на ограниченных интервалах, а RNN Элмана даёт преимущество при учёте последовательности, но требует более сложной настройки для длинных временных рядов.