

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)

# Параметры варианта
a, b, c, d = 0.2, 0.4, 0.09, 0.4
input_size = 6
hidden_size = 2
num_samples = 200
train_ratio = 0.8

# Входы: 6 признаков — степени x
x = torch.linspace(0, 10, num_samples).reshape(-1, 1)
X = torch.cat([x ** i for i in range(1, input_size + 1)], dim=1)
y = target_function(x, a, b, c, d)

# Train/Test split
split_idx = int(num_samples * train_ratio)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

print("Данные сгенерированы для функции  $y = a \cdot \cos(bx) + c \cdot \sin(dx)$ ")
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.hidden_dim = hidden_dim
        # RNN с tanh внутри (ограничение PyTorch)
        self.rnn = nn.RNN(input_dim, hidden_dim, nonlinearity="tanh",
batch_first=True)
```

```

        self.sigmoid = nn.Sigmoid()
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        # PHC ожидает входы в формате [batch, seq, features]
        x = x.unsqueeze(1)
        out, _ = self.rnn(x)
        out = self.sigmoid(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 = 500

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))

```

Данные сгенерированы для функции $y = a \cdot \cos(bx) + c \cdot \sin(dx)$
 Размер обучающей выборки: `torch.Size([160, 6])`, тестовой: `torch.Size([40, 6])`
 Обучение завершено. Минимальная ошибка: 0.005632504355162382

```

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

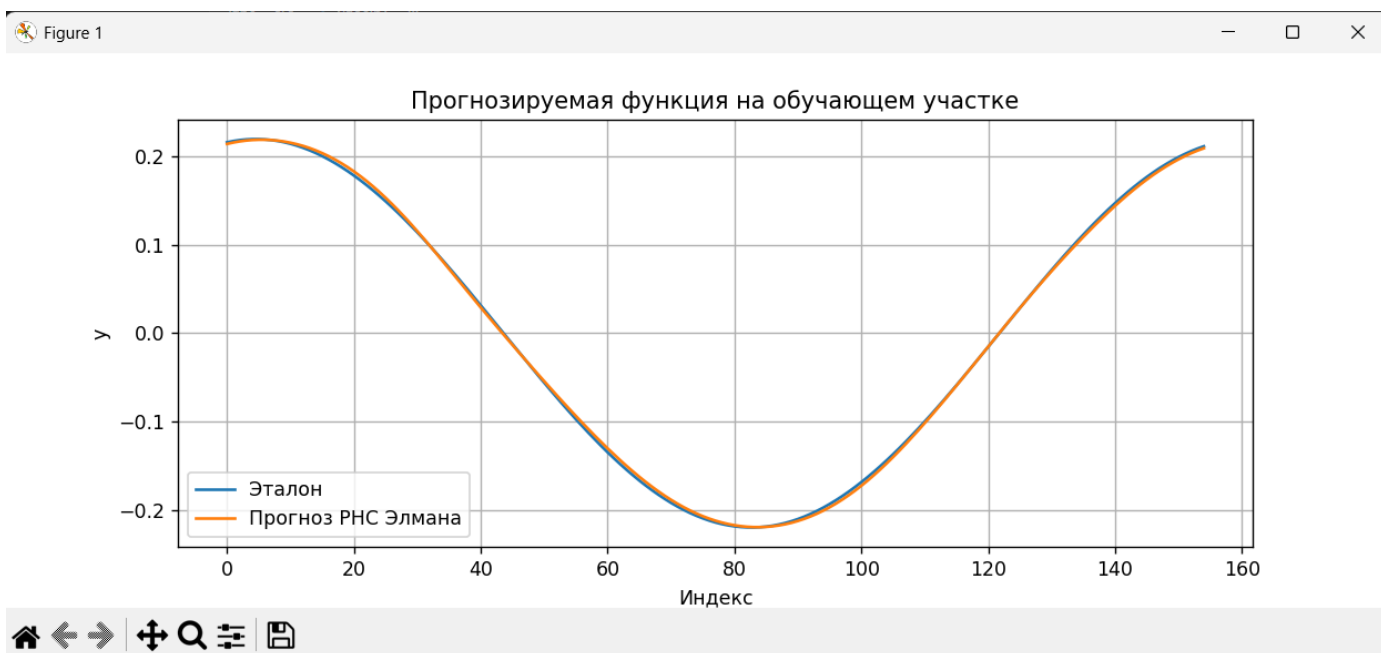
```



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: Научился применять классические алгоритмы классификации — метод k-ближайших соседей, дерево решений и метод опорных векторов — для решения задачи бинарной классификации. Получил практический опыт загрузки и предобработки данных, кодирования категориальных признаков, построения моделей с использованием библиотек Pandas, Scikit-learn и визуализации результатов с помощью Matplotlib и Seaborn. Освоил подбор гиперпараметров, расчёт метрик качества моделей и интерпретацию precision для оценки способности алгоритмов выявлять целевой класс.