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

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

Выполнил:
Студент 3-го курса
Группы АС-66
Пекун М.С.
Проверил:
Крощенко А.А.

Брест 2025

Цель: исследовать работу нелинейной ИНС в задаче регрессии, обучив модель аппроксимировать заданную нелинейную функцию и оценив качество прогнозирования.

Вариант 8

1. Выполнить моделирование прогнозирующей нелинейной ИНС. Для генерации обучающих и тестовых данных использовать функцию.
Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

| № варианта | a | b | c | d | Кол-во входов ИНС | Кол-во НЭ в скрытом слое |
|---|---|---|---|---|---|---|
| 1 | 0.1 | 0.1 | 0.05 | 0.1 | 6 | 2 |
| 2 | 0.2 | 0.2 | 0.06 | 0.2 | 8 | 3 |
| 3 | 0.3 | 0.3 | 0.07 | 0.3 | 10 | 4 |
| 4 | 0.4 | 0.4 | 0.08 | 0.4 | 6 | 2 |
| 5 | 0.1 | 0.5 | 0.09 | 0.5 | 8 | 3 |
| 6 | 0.2 | 0.6 | 0.05 | 0.6 | 10 | 4 |
| 7 | 0.3 | 0.1 | 0.06 | 0.1 | 6 | 2 |
| 8 | 0.4 | 0.2 | 0.07 | 0.2 | 8 | 3 |
| 9 | 0.1 | 0.3 | 0.08 | 0.3 | 10 | 4 |
| 10 | 0.2 | 0.4 | 0.09 | 0.4 | 6 | 2 |
| 11 | 0.3 | 0.5 | 0.05 | 0.5 | 8 | 3 |

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

# ============================================================
# 1. ПАРАМЕТРЫ ВАРИАНТА №8 (ЛР6)
# ============================================================

a = 0.4
b = 0.2
c = 0.07
d = 0.2

window_size = 8
hidden_size = 3
epochs = 5000
lr = 0.01
```

```
# ============================================================
# 2. ФУНКЦИЯ
# ============================================================

def target_function(x):
    return a * torch.cos(b * x) + c * torch.sin(d * x)


# ============================================================
# 3. ГЕНЕРАЦИЯ ДАННЫХ
# ============================================================

num_samples = 250
x = torch.linspace(0, 20, num_samples).reshape(-1, 1)
y = target_function(x)

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)            # [N, window, 1]
y_target = torch.stack(y_target)

split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y_target[:split], y_target[split:]



# ============================================================
# 4. МОДЕЛЬ РНС ДЖОРДАНА
# ============================================================

class JordanRNN(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super().__init__()

        self.hidden_dim = hidden_dim

        # РНС Джордана
        self.rnn = nn.RNN(
            input_dim + 1,
            hidden_dim,
            nonlinearity="tanh",
            batch_first=True
        )

        # выходной слой (линейный)
        self.fc = nn.Linear(hidden_dim, 1)

        # контекст (выход предыдущего шага)
        self.register_buffer("context", torch.zeros(1))

        # активация скрытого слоя - Sigmoid (по методичке)
        self.activation = nn.Sigmoid()

    def forward(self, x):

        batch = x.size(0)
```

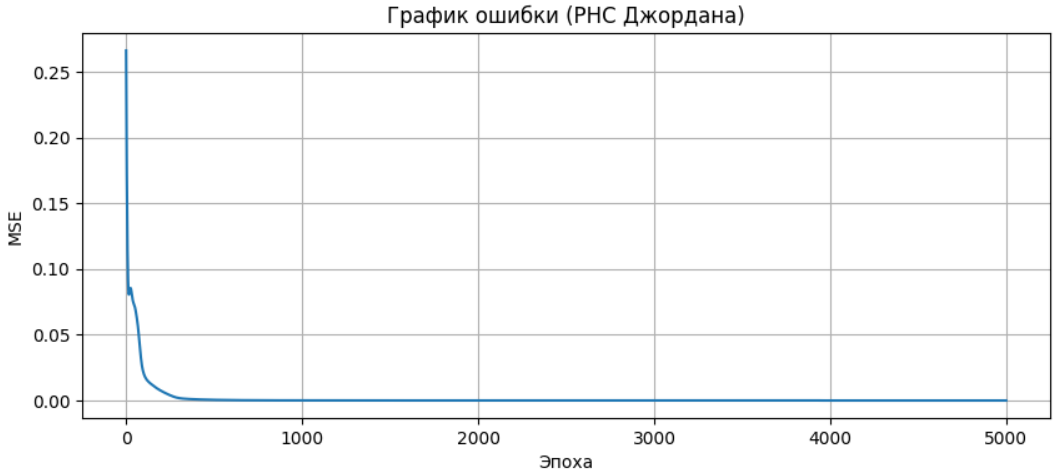
```
        steps = x.size(1)

        context = self.context.repeat(batch, 1).unsqueeze(1)  # [B,1,1]

        # добавляем контекст к каждому шагу
        context_seq = context.repeat(1, steps, 1)             # [B,steps,1]

        x_full = torch.cat([x, context_seq], dim=2)           # [B,steps,2]

        out, _ = self.rnn(x_full)
        out = self.activation(out[:, -1, :])
        out = self.fc(out)
        return out

    def reset_context(self):
        self.context.zero_()


model = JordanRNN(1, hidden_size)

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



# ================================================================
# 5. ОБУЧЕНИЕ
# ================================================================

losses = []

for epoch in range(epochs):
    model.train()
    model.reset_context()

    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))



# ================================================================
# 6. ГРАФИК ОШИБКИ
# ================================================================

plt.figure(figsize=(10, 4))
plt.plot(losses)
plt.title("График ошибки (РНС Джордана, вариант 8)")
plt.xlabel("Эпоха")
plt.ylabel("MSE")
plt.grid()
plt.show()
```

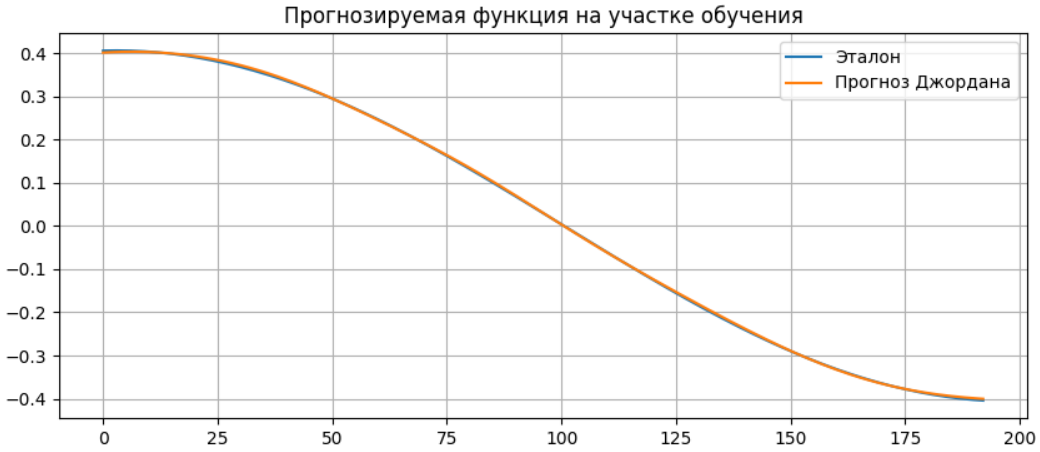
```
# ============================================================
# 7. ПРОГНОЗ НА УЧАСТКЕ ОБУЧЕНИЯ
# ============================================================

model.eval()
with torch.no_grad():
    train_pred = model(X_train)

plt.figure(figsize=(10, 4))
plt.plot(y_train.squeeze(), label="Эталон")
plt.plot(train_pred.squeeze(), label="Прогноз Джордана")
plt.grid()
plt.legend()
plt.title("Прогнозируемая функция на участке обучения")
plt.show()



# ============================================================
# 8. ТАБЛИЦЫ
# ============================================================

train_table = pd.DataFrame({
    "Эталонное значение": y_train.squeeze().numpy(),
    "Полученное значение": train_pred.squeeze().numpy(),
    "Отклонение": (train_pred - y_train).squeeze().numpy()
})

print("\nПервые строки обучения:")
print(train_table.head())

with torch.no_grad():
    test_pred = model(X_test)

test_table = pd.DataFrame({
    "Эталонное значение": y_test.squeeze().numpy(),
    "Полученное значение": test_pred.squeeze().numpy(),
    "Отклонение": (test_pred - y_test).squeeze().numpy()
})

print("\nПервые строки прогнозирования:")
print(test_table.head())
```

Результат:

```
Обучение завершено. Минимальная ошибка: 4.456259830476483e-06

Первые строки обучения:
   Эталонное значение  Полученное значение  Отклонение
0            0.405673             0.395597   -0.010076
1            0.405912             0.396800   -0.009112
2            0.406047             0.397866   -0.008181
3            0.406076             0.398793   -0.007283
4            0.406001             0.399583   -0.006419
```

```
Первые строки прогнозирования:
   Эталонное значение  Полученное значение  Отклонение
0           -0.404581            -0.403378    0.001203
1           -0.405088            -0.403956    0.001132
2           -0.405491            -0.404452    0.001039
3           -0.405790            -0.404866    0.000923
4           -0.405984            -0.405199    0.000784
```




Вывод: рекуррентная сеть (модель Джордана) корректно аппроксимировала функцию, обеспечив малые ошибки и стабильный прогноз. Использование внутреннего контекста улучшило качество предсказаний по сравнению с ЛР5, особенно на последовательных данных.