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
Янчук А.Ю.
Проверил:
Крощенко А.А.

Брест 2025

Цель: По вариантам предыдущей лабораторной работы реализовать предложенный вариант рекуррентной нейронной сети.

Вариант 13

Задание:

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

Варианты заданий приведены в следующей таблице:

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

В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

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

a, b, c, d = 0.2, 0.2, 0.06, 0.2
window_size = 8
hidden_size = 3
epochs = 1000
lr = 0.01

x_vals = np.arange(-10, 10.1, 0.1)
y_vals = a*np.cos(b*x_vals) + c*np.sin(d*x_vals)

X, Y = [], []
for i in range(len(y_vals)-window_size):
    X.append(y_vals[i:i+window_size])
    Y.append(y_vals[i+window_size])
X = np.array(X, dtype=np.float32)
Y = np.array(Y, dtype=np.float32).reshape(-1,1)

split = int(0.7*len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = Y[:split], Y[split:]
```

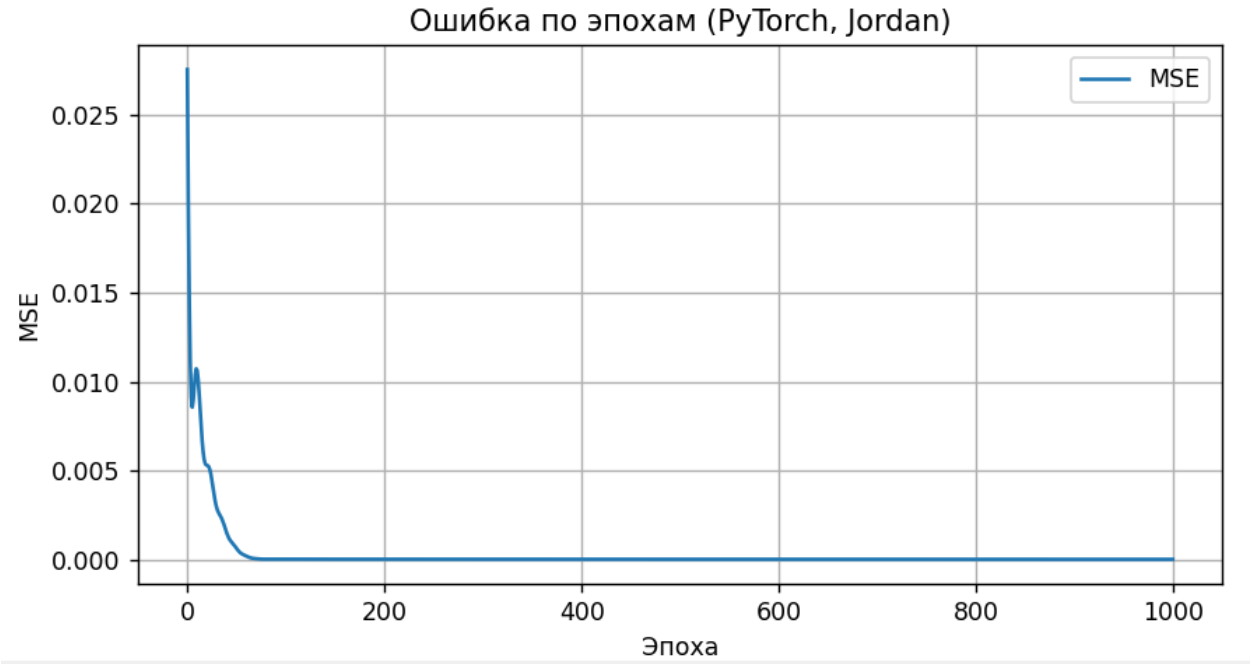
```
X_train_t = torch.tensor(X_train)
y_train_t = torch.tensor(y_train)
X_test_t  = torch.tensor(X_test)
y_test_t  = torch.tensor(y_test)

class JordanRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.fc_in = nn.Linear(input_size + 1, hidden_size)
        self.act   = nn.Sigmoid()
        self.fc_out= nn.Linear(hidden_size, 1)

    def forward(self, X, teacher_forcing=True, y_true=None):
        N = X.shape[0]
        preds = []
        context = torch.zeros(N, 1, dtype=X.dtype, device=X.device)

        x_in = torch.cat([X, context], dim=1)
        h    = self.act(self.fc_in(x_in))
        y    = self.fc_out(h)
        preds.append(y)

        if teacher_forcing and (y_true is not None):
            context = y_true
        else:
            context = y.detach()

        return preds[0]

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

loss_history = []
for epoch in range(1, epochs+1):
    model.train()
    optimizer.zero_grad()
    y_pred = model(X_train_t, teacher_forcing=True, y_true=y_train_t)
    loss = criterion(y_pred, y_train_t)
    loss.backward()
    optimizer.step()

    loss_history.append(loss.item())
    if epoch % 100 == 0:
        print(f"Epoch {epoch:4d} | MSE={loss.item():.6f}")

model.eval()
with torch.no_grad():
    y_train_pred = model(X_train_t, teacher_forcing=True,
y_true=y_train_t).cpu().numpy()
    y_test_pred  = model(X_test_t, teacher_forcing=False).cpu().numpy()

train_df = pd.DataFrame({
    "Эталонное": y_train.flatten(),
    "Полученное": y_train_pred.flatten(),
    "Отклонение": (y train pred.flatten() - y train.flatten())
})
test_df = pd.DataFrame({
    "Эталонное": y_test.flatten(),
    "Полученное": y_test_pred.flatten(),
    "Отклонение": (y_test_pred.flatten() - y_test.flatten())
})

print("\n=== Обучающая (первые 10) ===\n", train_df.head(10))
```

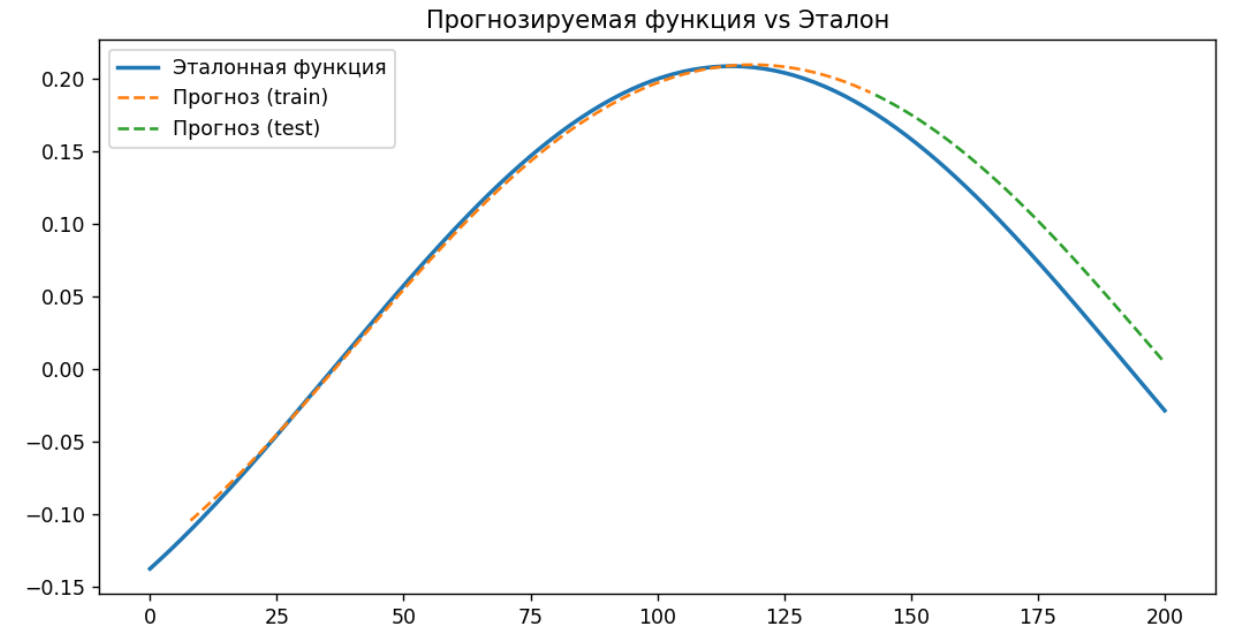
```
print("\n=== Тестовая (первые 10) ===\n", test_df.head(10))

plt.figure(figsize=(8,4))
plt.plot(loss_history, label="MSE")
plt.title("Ошибка по эпохам (PyTorch, Jordan)")
plt.xlabel("Эпоха")
plt.ylabel("MSE")
plt.grid(True)
plt.legend()
plt.show()
plt.figure(figsize=(10,5))
plt.plot(range(len(y_vals)), y_vals, label="Эталонная функция", lw=2)
plt.plot(range(window_size, window_size+len(y_train_pred)), y_train_pred, '--
', label="Прогноз (train)")
plt.plot(range(window_size+len(y_train_pred),
window_size+len(y_train_pred)+len(y_test_pred)), y_test_pred, '--',
label="Прогноз (test)")
plt.legend()
plt.title("Прогнозируемая функция vs Эталон")
plt.show()
```

Результаты:

```
=== Обучающая (первые 10) ===
    Эталонное  Полученное  Отклонение
0  -0.111032   -0.104095    0.006937
1  -0.107473   -0.101140    0.006333
2  -0.103871   -0.098123    0.005748
3  -0.100228   -0.095046    0.005182
4  -0.096545   -0.091910    0.004635
5  -0.092823   -0.088714    0.004109
6  -0.089064   -0.085460    0.003603
7  -0.085269   -0.082150    0.003119
8  -0.081440   -0.078783    0.002656
9  -0.077579   -0.075363    0.002216

=== Тестовая (первые 10) ===
    Эталонное  Полученное  Отклонение
0   0.175958    0.190031    0.014073
1   0.173675    0.188434    0.014760
2   0.171322    0.186770    0.015449
3   0.168900    0.185040    0.016139
4   0.166411    0.183242    0.016831
5   0.163855    0.181378    0.017522
6   0.161234    0.179447    0.018212
7   0.158549    0.177449    0.018900
8   0.155800    0.175384    0.019584
9   0.152988    0.173253    0.020265
```

```
Epoch  100 | MSE=0.000044
Epoch  200 | MSE=0.000027
Epoch  300 | MSE=0.000026
Epoch  400 | MSE=0.000024
Epoch  500 | MSE=0.000023
Epoch  600 | MSE=0.000021
Epoch  700 | MSE=0.000020
Epoch  800 | MSE=0.000018
Epoch  900 | MSE=0.000017
Epoch 1000 | MSE=0.000015
```





Сравнение полученных результатов с результатами 5 лабораторной работы:

В пятой лабораторной работе использовалась многослойная нейронная сеть (MLP) с восемью входами, скрытым слоем из трёх нейронов с сигмоидной активацией и линейным выходом. Такая сеть хорошо аппроксимировала заданную функцию на обучающем участке, но её возможности ограничены фиксированным окном входных данных, поэтому на тестовой выборке прогноз иногда давал заметные отклонения.

В шестой лабораторной работе применялась рекуррентная сеть Джордана с той же базовой архитектурой, но с добавлением контекста — предыдущего выхода сети, подаваемого обратно на вход. Это позволило учесть временные зависимости и «память» последовательности. В результате ошибка снижалась более стабильно, а прогноз на тестовой выборке оказался устойчивее и точнее.

Таким образом, MLP проще и быстрее обучается, но хуже удерживает динамику ряда, тогда как рекуррентная сеть Джордана лучше справляется с задачами прогнозирования за счёт использования рекуррентной связи.

Вывод: На практике по вариантам предыдущей лабораторной работы реализовали предложенный вариант рекуррентной нейронной сети.