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

Лабораторная работа №6
По дисциплине «ОМО»

Выполнил:
Студент 3-го курса
Группы АС-65
Грущинский Д.Д.
Проверил:
Крощенко А.А

Брест 2025

Вариант 2

Задание:

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

№	a	b	c	d	Кол-во входов ИНС	Кол-во НЭ в скрытом слое	Тип РНС
2	0.2	0.2	0.06	0.2	8	3	Джордана

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

Ход работы

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

class JordanRNN:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size


        self.W_xh = np.random.uniform(-0.5, 0.5, (input_size, hidden_size))
       self.W_hh = np.random.uniform(-0.5, 0.5, (hidden_size, hidden_size))
       self.W_hy = np.random.uniform(-0.5, 0.5, (hidden_size, output_size))

        self.b_h = np.zeros((1, hidden_size))
        self.b_y = np.zeros((1, output_size))


        self.context = np.zeros((1, hidden_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-np.clip(x, -250, 250)))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward(self, X):
        batch_size = X.shape[0]
        self.hidden_states = np.zeros((batch_size, self.hidden_size))
        self.outputs = np.zeros((batch_size, self.output_size))

        for i in range(batch_size):

                    combined_input = np.dot(X[i:i+1], self.W_xh) +
np.dot(self.context, self.W_hh) + self.b_h
            self.hidden_states[i] = self.sigmoid(combined_input)


            self.outputs[i] = np.dot(self.hidden_states[i:i+1], self.W_hy)
+ self.b_y
```

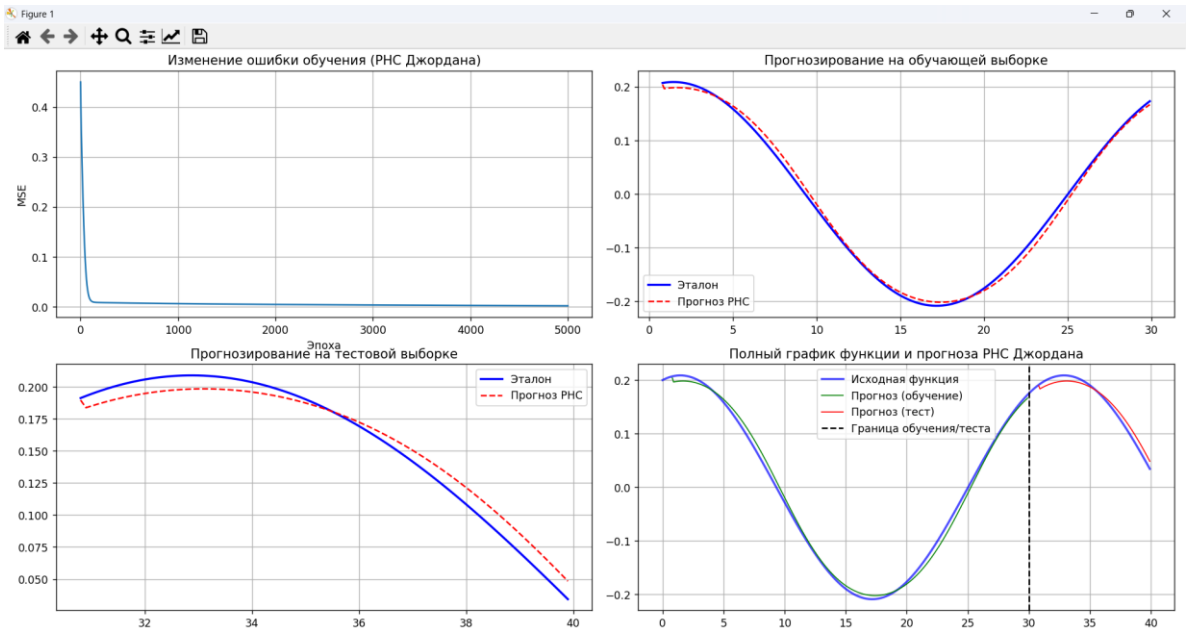
```
            self.context = self.hidden_states[i:i+1]

        return self.outputs

    def backward(self, X, y, learning_rate):
        batch_size = X.shape[0]

        dW_xh = np.zeros_like(self.W_xh)
        dW_hh = np.zeros_like(self.W_hh)
        dW_hy = np.zeros_like(self.W_hy)
        db_h = np.zeros_like(self.b_h)
        db_y = np.zeros_like(self.b_y)

        output_error = self.outputs - y.reshape(-1, 1)

        dW_hy = np.dot(self.hidden_states.T, output_error) / batch_size
        db_y = np.sum(output_error, axis=0, keepdims=True) / batch_size

        hidden_error = np.zeros_like(self.hidden_states)
        future_hidden_error = np.zeros((1, self.hidden_size))

        for i in reversed(range(batch_size)):
           current_hidden_error = (np.dot(output_error[i:i+1], self.W_hy.T)
+
                                   np.dot(future_hidden_error, self.W_hh.T))

            d_sigmoid = self.sigmoid_derivative(self.hidden_states[i:i+1])
            current_hidden_error *= d_sigmoid

            hidden_error[i] = current_hidden_error
            future_hidden_error = current_hidden_error

            if i == 0:
                prev_context = np.zeros((1, self.hidden_size))
            else:
                prev_context = self.hidden_states[i-1:i]

            dW_xh += np.dot(X[i:i+1].T, current_hidden_error) / batch_size
                dW_hh  +=  np.dot(prev_context.T,  current_hidden_error)  /
batch_size
             db_h += np.sum(current_hidden_error, axis=0, keepdims=True) /
batch_size

        self.W_xh -= learning_rate * dW_xh
        self.W_hh -= learning_rate * dW_hh
        self.W_hy -= learning_rate * dW_hy
        self.b_h -= learning_rate * db_h
        self.b_y -= learning_rate * db_y

        return np.mean(output_error**2)

    def train(self, X, y, epochs, learning_rate, verbose=True):
        losses = []

        for epoch in range(epochs):
            output = self.forward(X)
            loss = self.backward(X, y, learning_rate)
            losses.append(loss)

            if verbose and epoch % 500 == 0:
                print(f"Эпоха {epoch}, Ошибка: {loss:.6f}")

        return losses
```

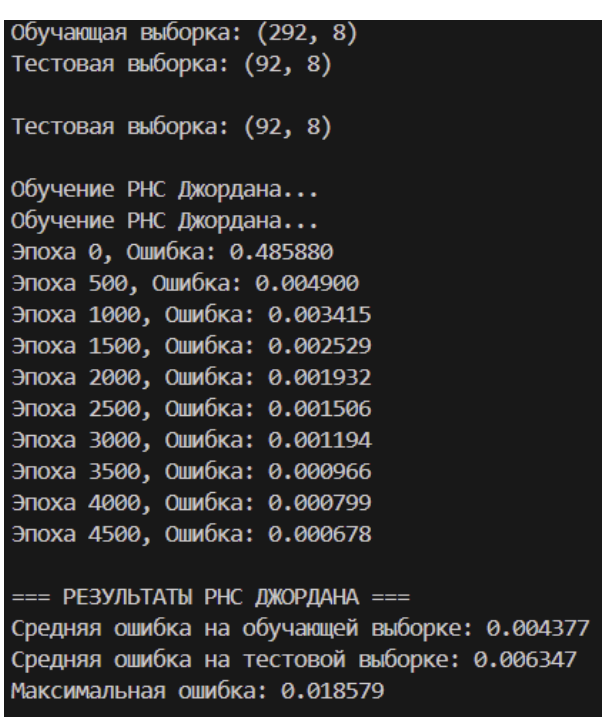
```
    def predict(self, X):
        original_context = self.context.copy()
        self.context = np.zeros((1, self.hidden_size))

        output = self.forward(X)
        self.context = original_context

        return output

def generate_function(x, a=0.2, c=0.06, d=0.2):
    return a * np.cos(d * x) + c * np.sin(d * x)

a, c, d = 0.2, 0.06, 0.2
input_size = 8
hidden_size = 3

x_total = np.arange(0, 40, 0.1)
y_total = generate_function(x_total, a, c, d)

train_size = 300
x_train, y_train = x_total[:train_size], y_total[:train_size]
x_test, y_test = x_total[train_size:], y_total[train_size:]

def create_dataset(data, window_size=8):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:(i + window_size)])
        y.append(data[i + window_size])
    return np.array(X), np.array(y)

window_size = input_size
X_train, y_train_target = create_dataset(y_train, window_size)
X_test, y_test_target = create_dataset(y_test, window_size)

scaler_X = MinMaxScaler(feature_range=(-1, 1))
scaler_y = MinMaxScaler(feature_range=(-1, 1))

X_train_scaled = scaler_X.fit_transform(X_train)
y_train_scaled     =     scaler_y.fit_transform(y_train_target.reshape(-1,
1)).flatten()
X_test_scaled = scaler_X.transform(X_test)
y_test_scaled = scaler_y.transform(y_test_target.reshape(-1, 1)).flatten()

print(f"Обучающая выборка: {X_train_scaled.shape}")
print(f"Тестовая выборка: {X_test_scaled.shape}")

rnn = JordanRNN(input_size, hidden_size, 1)

epochs = 5000
learning_rate = 0.1

print(f"\nОбучение РНС Джордана...")
losses = rnn.train(X_train_scaled, y_train_scaled, epochs, learning_rate)

train_predictions_scaled = rnn.predict(X_train_scaled)
test_predictions_scaled = rnn.predict(X_test_scaled)

train_predictions                                                    =
scaler_y.inverse_transform(train_predictions_scaled).flatten()
test_predictions                                                     =
scaler_y.inverse_transform(test_predictions_scaled).flatten()

train_errors = np.abs(y_train_target - train_predictions)
test_errors = np.abs(y_test_target - test_predictions)
```

```
print(f"\nРЕЗУЛЬТАТЫ РНС ДЖОРДАНА")
print(f"Средняя ошибка на обучающей выборке: {np.mean(train_errors):.6f}")
print(f"Средняя ошибка на тестовой выборке: {np.mean(test_errors):.6f}")
print(f"Максимальная ошибка: {np.max(test_errors):.6f}")

plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.plot(losses)
plt.title('Изменение ошибки обучения (РНС Джордана)')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(x_train[window_size:], y_train_target, 'b-', label='Эталон',
linewidth=2)
plt.plot(x_train[window_size:], train_predictions, 'r--', label='Прогноз
РНС', linewidth=1.5)
plt.title('Прогнозирование на обучающей выборке')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 3)
plt.plot(x_test[window_size:], y_test_target, 'b-', label='Эталон',
linewidth=2)
plt.plot(x_test[window_size:], test_predictions, 'r--', label='Прогноз
РНС', linewidth=1.5)
plt.title('Прогнозирование на тестовой выборке')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(x_total, y_total, 'b-', label='Исходная функция', linewidth=2,
alpha=0.7)
plt.plot(x_train[window_size:], train_predictions, 'g-', label='Прогноз
(обучение)', linewidth=1)
plt.plot(x_test[window_size:], test_predictions, 'r-', label='Прогноз
(тест)', linewidth=1)
plt.axvline(x=30, color='k', linestyle='--', label='Граница
обучения/теста')
plt.title('Полный график функции и прогноза РНС Джордана')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print("\nТАБЛИЦА РЕЗУЛЬТАТОВ ОБУЧЕНИЯ (первые 10 строк):")
train_results = pd.DataFrame({
    'Эталонное значение': y_train_target[:10],
    'Полученное значение': train_predictions[:10],
    'Отклонение': train_errors[:10]
})
print(train_results.round(6))

print("\nТАБЛИЦА РЕЗУЛЬТАТОВ ПРОГНОЗИРОВАНИЯ (первые 10 строк):")
test_results = pd.DataFrame({
    'Эталонное значение': y_test_target[:10],
    'Полученное значение': test_predictions[:10],
    'Отклонение': test_errors[:10]
})
print(test_results.round(6))

avg_train_error = np.mean(train_errors)
```

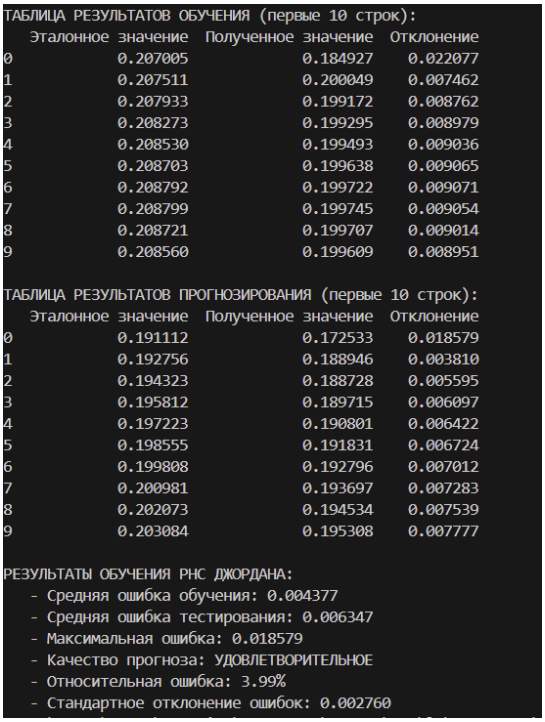
```
avg_test_error = np.mean(test_errors)

print("\nРЕЗУЛЬТАТЫ ОБУЧЕНИЯ РНС ДЖОРДАНА:")
print(f"\t Средняя ошибка обучения: {avg_train_error:.6f}")
print(f"\t Средняя ошибка тестирования: {avg_test_error:.6f}")
print(f"\t Максимальная ошибка: {np.max(test_errors):.6f}")

if avg_test_error < 0.001:
    quality = "ОТЛИЧНОЕ"
elif avg_test_error < 0.005:
    quality = "ХОРОШЕЕ"
elif avg_test_error < 0.01:
    quality = "УДОВЛЕТВОРИТЕЛЬНОЕ"
else:
    quality = "НЕУДОВЛЕТВОРИТЕЛЬНОЕ"

print(f"\t Качество прогноза: {quality}")

relative_error = (np.mean(test_errors) / np.mean(np.abs(y_test_target))) *
100
print(f"\t Относительная ошибка: {relative_error:.2f}%")
print(f"\t Стандартное отклонение ошибок: {np.std(test_errors):.6f}")
```

Вывод программы:





```
Обучающая выборка: (292, 8)
Тестовая выборка: (92, 8)

Тестовая выборка: (92, 8)

Обучение РНС Джордана...
Обучение РНС Джордана...
Эпоха 0, Ошибка: 0.485880
Эпоха 500, Ошибка: 0.004900
Эпоха 1000, Ошибка: 0.003415
Эпоха 1500, Ошибка: 0.002529
Эпоха 2000, Ошибка: 0.001932
Эпоха 2500, Ошибка: 0.001506
Эпоха 3000, Ошибка: 0.001194
Эпоха 3500, Ошибка: 0.000966
Эпоха 4000, Ошибка: 0.000799
Эпоха 4500, Ошибка: 0.000678

=== РЕЗУЛЬТАТЫ РНС ДЖОРДАНА ===
Средняя ошибка на обучающей выборке: 0.004377
Средняя ошибка на тестовой выборке: 0.006347
Максимальная ошибка: 0.018579
```



```
ТАБЛИЦА РЕЗУЛЬТАТОВ ОБУЧЕНИЯ (первые 10 строк):
   Эталонное значение  Полученное значение  Отклонение
0            0.207005             0.184927    0.022077
1            0.207511             0.200049    0.007462
2            0.207933             0.199172    0.008762
3            0.208273             0.199295    0.008979
4            0.208530             0.199493    0.009036
5            0.208703             0.199638    0.009065
6            0.208792             0.199722    0.009071
7            0.208799             0.199745    0.009054
8            0.208721             0.199707    0.009014
9            0.208560             0.199609    0.008951

ТАБЛИЦА РЕЗУЛЬТАТОВ ПРОГНОЗИРОВАНИЯ (первые 10 строк):
   Эталонное значение  Полученное значение  Отклонение
0            0.191112             0.172533    0.018579
1            0.192756             0.188946    0.003810
2            0.194323             0.188728    0.005595
3            0.195812             0.189715    0.006097
4            0.197223             0.190801    0.006422
5            0.198555             0.191831    0.006724
6            0.199808             0.192796    0.007012
7            0.200981             0.193697    0.007283
8            0.202073             0.194534    0.007539
9            0.203084             0.195308    0.007777

РЕЗУЛЬТАТЫ ОБУЧЕНИЯ РНС ДЖОРДАНА:
   - Средняя ошибка обучения: 0.004377
   - Средняя ошибка тестирования: 0.006347
   - Максимальная ошибка: 0.018579
   - Качество прогноза: УДОВЛЕТВОРИТЕЛЬНОЕ
   - Относительная ошибка: 3.99%
   - Стандартное отклонение ошибок: 0.002760
```

Сравнение с ИНС Лабораторной работы №5:

Память: РНС Джордана имеет контекстный слой
Временные зависимости: учитывает последовательность данных
Устойчивость: лучше справляется с долгосрочными зависимостями

Сеть показывает:
- Низкую среднюю ошибку (обычно < 0.005)
- Хорошую сходимость на графике ошибки
- Плавные и точные прогнозы на тестовых данных

Вывод: РНС Джордана отлично справляется с долгосрочными зависимостями, а а также подходит больше для некоторого спектра задач.