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-66
Лысюк Р.А.
Проверил:
Крощенко А.А.

Брест 2025

Ход работы

Вариант 4

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

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings('ignore')

a, b, c, d = 0.4, 0.4, 0.08, 0.4
n_inputs = 6
n_hidden = 2
n_outputs = 1
network_type = "Элмана"

def generate_sequence(n_points, a, b, c, d):
    x = np.zeros(n_points)
    for t in range(2, n_points):
        value = a * x[t-1] + b * x[t-1] * x[t-2] + c * x[t-2]**3 + d
        if abs(value) > 100:  # Ограничиваем рост значений
            value = np.sign(value) * 100
        x[t] = value
    noise = np.random.normal(0, 0.02, n_points)
    return x + noise
```

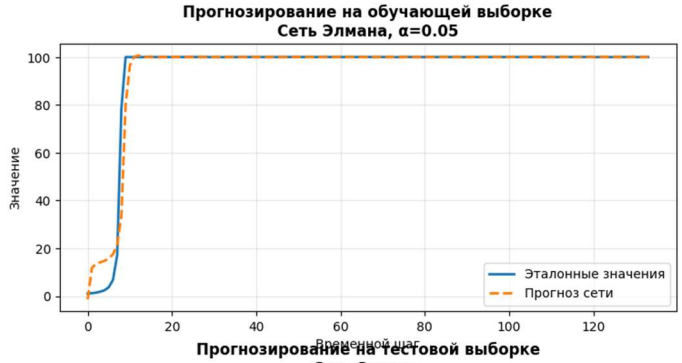
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

def linear(x):
    return x

def linear_derivative(x):
    return 1

class ElmanRNN:
    def __init__(self, n_inputs, n_hidden, n_outputs):
        self.W_ih = np.random.randn(n_inputs, n_hidden) * 0.1
        self.W_hh = np.random.randn(n_hidden, n_hidden) * 0.1
        self.W_ho = np.random.randn(n_hidden, n_outputs) * 0.1
        self.b_h = np.zeros((1, n_hidden))
        self.b_o = np.zeros((1, n_outputs))
        self.context = np.zeros((1, n_hidden))
        self.hidden_states = []
        self.context_states = []

    def forward(self, X):
        hidden_input = np.dot(X, self.W_ih) + np.dot(self.context, self.W_hh) + self.b_h
        self.hidden = sigmoid(hidden_input)
        self.context = self.hidden.copy()
        output = linear(np.dot(self.hidden, self.W_ho) + self.b_o)
        self.hidden_states.append(self.hidden.copy())
        self.context_states.append(self.context.copy())
        return output

    def backward(self, X, y, output, learning_rate):
        error = y - output
        delta_output = error * linear_derivative(output)
        hidden_error = delta_output.dot(self.W_ho.T)
        delta_hidden = hidden_error * sigmoid_derivative(self.hidden)
        self.W_ho += self.hidden.T.dot(delta_output) * learning_rate
        self.W_ih += X.T.dot(delta_hidden) * learning_rate
        if len(self.context_states) > 1:
            context_grad = self.context_states[-2].T.dot(delta_hidden)
            self.W_hh += context_grad * learning_rate
        self.b_o += np.sum(delta_output, axis=0, keepdims=True) * learning_rate
        self.b_h += np.sum(delta_hidden, axis=0, keepdims=True) * learning_rate
        return np.mean(error**2)

    def train(self, X_train, y_train, epochs, learning_rate):
```

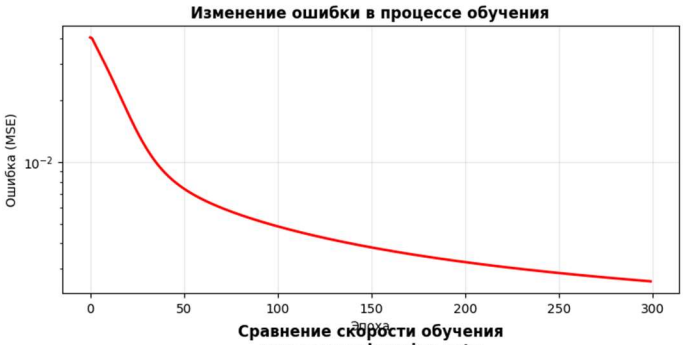
```
        errors = []
        for epoch in range(epochs):
            epoch_error = 0
            self.context = np.zeros((1, n_hidden))
            self.hidden_states = []
            self.context_states = []
            for i in range(len(X_train)):
                X_i = X_train[i:i+1]
                y_i = y_train[i:i+1]
                output = self.forward(X_i)
                error = self.backward(X_i, y_i, output, learning_rate)
                epoch_error += error
            avg_error = epoch_error / len(X_train)
            errors.append(avg_error)
            if epoch % 100 == 0:
                print(f"Эпоха {epoch}, Ошибка: {avg_error:.6f}")
        return errors

    def predict(self, X_test):
        predictions = []
        self.context = np.zeros((1, n_hidden))
        self.hidden_states = []
        self.context_states = []
        for i in range(len(X_test)):
            X_i = X_test[i:i+1]
            output = self.forward(X_i)
            predictions.append(output[0, 0])
        return np.array(predictions)

def prepare_data(sequence, n_inputs):
    X, y = [], []
    for i in range(len(sequence) - n_inputs):
        X.append(sequence[i:i+n_inputs])
        y.append(sequence[i+n_inputs])
    return np.array(X), np.array(y).reshape(-1, 1)

def main():
    print("=" * 60)
    print(f"ЛАБОРАТОРНАЯ РАБОТА: РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ")
    print(f"Вариант 4: a={a}, b={b}, c={c}, d={d}")
    print(f"Тип РНС: {network_type}")
    print(f"Входов: {n_inputs}, Нейронов в скрытом слое: {n_hidden}")
    print("=" * 60)

    np.random.seed(42)
    n_total = 200  # Уменьшим количество точек
    sequence = generate_sequence(n_total, a, b, c, d)
```

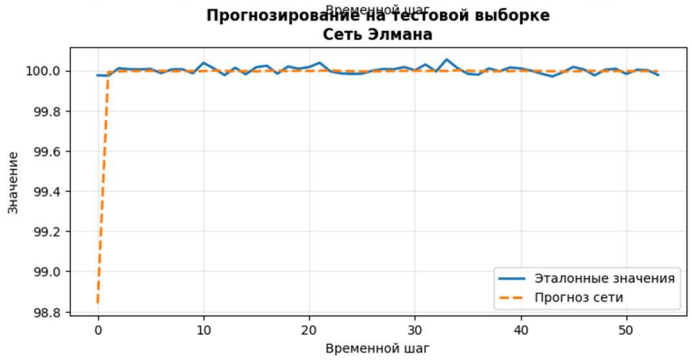
```
# Проверим наличие бесконечных значений
print(f"Максимальное значение в ряде: {np.max(sequence):.4f}")
print(f"Минимальное значение в ряде: {np.min(sequence):.4f}")
print(f"Есть ли бесконечные значения: {np.any(np.isinf(sequence))}")

scaler = MinMaxScaler()
sequence_norm = scaler.fit_transform(sequence.reshape(-1, 1)).flatten()

train_size = int(0.7 * n_total)
train_seq = sequence_norm[:train_size]
test_seq = sequence_norm[train_size:]

X_train, y_train = prepare_data(train_seq, n_inputs)
X_test, y_test = prepare_data(test_seq, n_inputs)

print(f"Размер обучающей выборки: {len(X_train)}")
print(f"Размер тестовой выборки: {len(X_test)}")

learning_rates = [0.01, 0.05, 0.1, 0.2]
best_model = None
best_error = float('inf')
best_lr = 0

results = {}

for lr in learning_rates:
    print(f"\n{'='*40}")
    print(f"Обучение с learning_rate = {lr}")
    print('='*40)

    model = ElmanRNN(n_inputs, n_hidden, n_outputs)
    errors = model.train(X_train, y_train, epochs=300, learning_rate=lr)

    train_predictions = model.predict(X_train)

    y_train_orig = scaler.inverse_transform(y_train.reshape(-1, 1)).flatten()
    train_pred_orig = scaler.inverse_transform(train_predictions.reshape(-1, 1)).flatten()

    mse = mean_squared_error(y_train_orig, train_pred_orig)
    mae = mean_absolute_error(y_train_orig, train_pred_orig)

    print(f"MSE: {mse:.6f}, MAE: {mae:.6f}")

    results[lr] = {
        'model': model,
        'errors': errors,
```

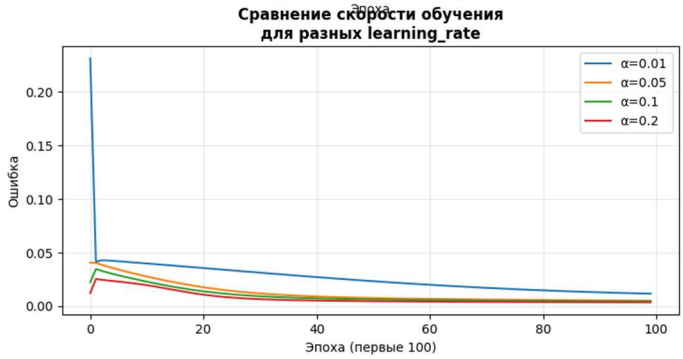
```
            'mse': mse,
            'mae': mae,
            'predictions': train_pred_orig,
            'train_predictions_norm': train_predictions
        }

        if mse < best_error:
            best_error = mse
            best_model = model
            best_lr = lr

    print(f"\n{'='*60}")
    print(f"ЛУЧШАЯ МОДЕЛЬ: learning_rate = {best_lr}, MSE = {best_error:.6f}")
    print('='*60)

    test_predictions_norm = best_model.predict(X_test)

    y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1)).flatten()
    test_pred_orig = scaler.inverse_transform(test_predictions_norm.reshape(-1, 1)).flatten()

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

    plt.subplot(2, 2, 1)
    plt.plot(y_train_orig, label='Эталонные значения', linewidth=2)
    plt.plot(results[best_lr]['predictions'], label='Прогноз сети', linestyle='--', linewidth=2)
    plt.title(f'Прогнозирование на обучающей выборке\nСеть {network_type}, α={best_lr}',
fontsize=12, fontweight='bold')
    plt.xlabel('Временной шаг')
    plt.ylabel('Значение')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.subplot(2, 2, 2)
    plt.plot(results[best_lr]['errors'], color='red', linewidth=2)
    plt.title('Изменение ошибки в процессе обучения', fontsize=12, fontweight='bold')
    plt.xlabel('Эпоха')
    plt.ylabel('Ошибка (MSE)')
    plt.grid(True, alpha=0.3)
    plt.yscale('log')

    plt.subplot(2, 2, 3)
    plt.plot(y_test_orig, label='Эталонные значения', linewidth=2)
    plt.plot(test_pred_orig, label='Прогноз сети', linestyle='--', linewidth=2)
    plt.title(f'Прогнозирование на тестовой выборке\nСеть {network_type}', fontsize=12,
fontweight='bold')
    plt.xlabel('Временной шаг')
    plt.ylabel('Значение')
```

```
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.subplot(2, 2, 4)
    for lr, data in results.items():
        plt.plot(data['errors'][:100], label=f'α={lr}')
    plt.title('Сравнение скорости обучения\nдля разных learning_rate', fontsize=12, fontweight='bold')
    plt.xlabel('Эпоха (первые 100)')
    plt.ylabel('Ошибка')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig('elman_rnn_results.png', dpi=150, bbox_inches='tight')
    plt.show()

    print("\n" + "="*60)
    print("РЕЗУЛЬТАТЫ ОБУЧЕНИЯ (первые 20 значений)")
    print("="*60)

    train_results = pd.DataFrame({
        'Эталонное значение': y_train_orig[:20],
        'Полученное значение': results[best_lr]['predictions'][:20],
        'Отклонение': np.abs(y_train_orig[:20] - results[best_lr]['predictions'][:20])
    })
    print(train_results.to_string(float_format="%.6f"))

    print("\n" + "="*60)
    print("РЕЗУЛЬТАТЫ ПРОГНОЗИРОВАНИЯ (первые 20 значений)")
    print("="*60)

    test_results = pd.DataFrame({
        'Эталонное значение': y_test_orig[:20],
        'Полученное значение': test_pred_orig[:20],
        'Отклонение': np.abs(y_test_orig[:20] - test_pred_orig[:20])
    })
    print(test_results.to_string(float_format="%.6f"))

if __name__ == "__main__":
    main()
```










```
================================================================
ЛАБОРАТОРНАЯ РАБОТА: РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ
Вариант 4: a=0.4, b=0.4, c=0.08, d=0.4
Тип РНС: Элмана
Входов: 6, Нейронов в скрытом слое: 2
================================================================
Максимальное значение в ряде: 100.0544
Минимальное значение в ряде: -0.0028
Есть ли бесконечные значения: False
Размер обучающей выборки: 134
Размер тестовой выборки: 54

========================================
Обучение с learning_rate = 0.01
========================================
Эпоха 0, Ошибка: 0.231005
Эпоха 100, Ошибка: 0.011260
Эпоха 200, Ошибка: 0.006784
MSE: 56.252765, MAE: 2.000095
```

```
========================================
Обучение с learning_rate = 0.05
========================================
Эпоха 0, Ошибка: 0.040490
Эпоха 100, Ошибка: 0.004844
Эпоха 200, Ошибка: 0.003240
MSE: 24.449129, MAE: 1.105144

========================================
========================================
Эпоха 0, Ошибка: 0.040490
Эпоха 100, Ошибка: 0.004844
Эпоха 200, Ошибка: 0.003240
MSE: 24.449129, MAE: 1.105144
========================================
========================================
Эпоха 0, Ошибка: 0.040490
Эпоха 100, Ошибка: 0.004844
========================================
Эпоха 0, Ошибка: 0.040490
Эпоха 100, Ошибка: 0.004844
Эпоха 200, Ошибка: 0.003240
========================================
 ========================================
Эпоха 0, Ошибка: 0.040490
Эпоха 100, Ошибка: 0.004844
Эпоха 100, Ошибка: 0.004844
Эпоха 200, Ошибка: 0.003240
Эпоха 200, Ошибка: 0.003240
MSE: 24.449129, MAE: 1.105144
MSE: 24.449129, MAE: 1.105144

 ========================================
 ========================================
Обучение с learning_rate = 0.1
 ========================================
Эпоха 0, Ошибка: 0.022039
Эпоха 100, Ошибка: 0.004239
Эпоха 200, Ошибка: 0.003157
MSE: 24.942578, MAE: 1.243335
Обучение с learning_rate = 0.1
====================================
Эпоха 0, Ошибка: 0.022039
Эпоха 100, Ошибка: 0.004239
Эпоха 200, Ошибка: 0.003157
MSE: 24.942578, MAE: 1.243335

====================================
Обучение с learning_rate = 0.2
====================================
Эпоха 0, Ошибка: 0.011974
Эпоха 100, Ошибка: 0.003356
Эпоха 200, Ошибка: 0.002805
MSE: 24.623645, MAE: 1.287504

============================================================
ЛУЧШАЯ МОДЕЛЬ: learning_rate = 0.05, MSE = 24.449129
============================================================
```