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

Лабораторная работа №6
По дисциплине: «Основы машинного обучения»
Тема: «Рекуррентные нейронные сети»

Выполнил:
Студент 3 курса
Группы АС-66
Гончерёнок К. А.
Проверил:
Крощенко А. А.

Брест 2025

Цель работы: изучить применение нелинейной искусственной нейронной сети с одним скрытым слоем для решения задачи регрессии и прогнозирования, реализовать обучение сети на синтетических данных и оценить точность полученной модели.

Вариант 2

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
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:

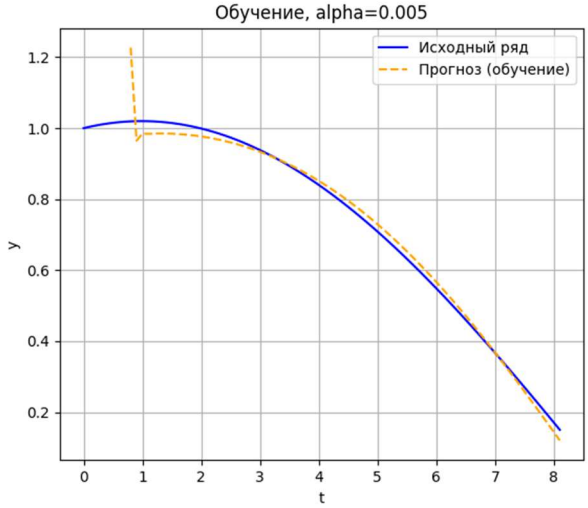
Код программы:

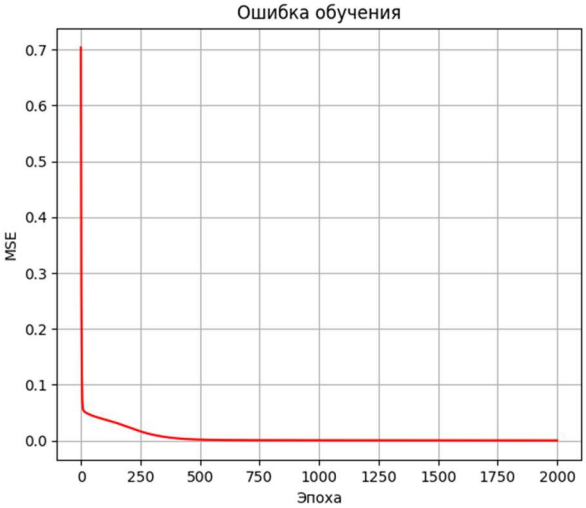
```
import numpy as np
import matplotlib.pyplot as plt

a = 0.2
b = 0.2
c = 0.06
d = 0.2
num_inputs = 8
hidden_neurons = 3
alpha_values = [0.001, 0.005, 0.01, 0.05, 0.1]


def generate_data(a, b, c, d, t_start=0, t_end=10, step=0.1):
    t = np.arange(t_start, t_end + step, step)
    y = a * np.sin(b * t) + np.cos(d*t)
    return t, y


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


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


def linear(x):
    return x


def create_sequences(data, seq_length):
    X, Y = [], []
    for i in range(len(data) - seq_length):
```

```
        X.append(data[i:i + seq_length])
        Y.append(data[i + seq_length])
    return np.array(X), np.array(Y)


def initialize_weights(input_size, hidden_size, output_size):
    W_input_hidden = np.random.randn(input_size, hidden_size) * 0.1
    W_hidden_output = np.random.randn(hidden_size, output_size) * 0.1
    W_context_hidden = np.random.randn(hidden_size, hidden_size) * 0.1
    return W_input_hidden, W_hidden_output, W_context_hidden


def forward_pass(X, W_ih, W_ho, W_ch, context):
    hidden_input = np.dot(X, W_ih) + np.dot(context, W_ch)
    hidden_output = sigmoid(hidden_input)
    output = linear(np.dot(hidden_output, W_ho))
    return hidden_output, output


def train_jordan(X_train, Y_train, hidden_neurons, alpha, epochs=2000):
    input_size = X_train.shape[1]
    output_size = 1
    W_ih, W_ho, W_ch = initialize_weights(input_size, hidden_neurons,
output_size)

    errors = []
    context = np.zeros((1, hidden_neurons))

    for epoch in range(epochs):
        epoch_error = 0
        for i in range(len(X_train)):
            x = X_train[i].reshape(1, -1)
            y_true = Y_train[i]

            hidden, y_pred = forward_pass(x, W_ih, W_ho, W_ch, context)

            error = y_true - y_pred.item()
            epoch_error += error ** 2

            delta_output = error
            delta_hidden = delta_output * W_ho.T * sigmoid_derivative(hidden)

            W_ho += alpha * hidden.T * delta_output
            W_ih += alpha * x.T * delta_hidden
            W_ch += alpha * context.T * delta_hidden

            context = hidden.copy()

        avg_error = epoch_error / len(X_train)
        errors.append(avg_error)
        if epoch % 500 == 0:
            print(f"Эпоха {epoch}, ошибка: {avg_error:.6f}")

    return W_ih, W_ho, W_ch, errors


def predict_jordan(X, W_ih, W_ho, W_ch):
    predictions = []
    context = np.zeros((1, hidden_neurons))
    for i in range(len(X)):
        x = X[i].reshape(1, -1)
```

```
        hidden, y_pred = forward_pass(x, W_ih, W_ho, W_ch, context)
        predictions.append(y_pred.item())
        context = hidden.copy()
    return np.array(predictions)


t, y = generate_data(a, b, c, d, step=0.1)
seq_length = num_inputs
X, Y = create_sequences(y, seq_length)

split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
Y_train, Y_test = Y[:split], Y[split:]

best_alpha = None
best_error = float('inf')
best_weights = None
best_errors = []

for alpha in alpha_values:
    print(f"\nОбучение с alpha = {alpha}")
    W_ih, W_ho, W_ch, errors = train_jordan(X_train, Y_train, hidden_neurons,
alpha, epochs=2000)
    predictions = predict_jordan(X_test, W_ih, W_ho, W_ch)
    test_error = np.mean((predictions - Y_test) ** 2)
    print(f"Ошибка на тесте: {test_error:.6f}")

    if test_error < best_error:
        best_error = test_error
        best_alpha = alpha
        best_weights = (W_ih, W_ho, W_ch)
        best_errors = errors

print(f"\nЛучший alpha: {best_alpha}, ошибка на тесте: {best_error:.6f}")

W_ih, W_ho, W_ch = best_weights
predictions_train = predict_jordan(X_train, W_ih, W_ho, W_ch)
predictions_test = predict_jordan(X_test, W_ih, W_ho, W_ch)

fig, axes = plt.subplots(2, 2, figsize=(12, 10))

axes[0, 0].plot(t[:split + seq_length], y[:split + seq_length],
label='Исходный ряд', color='blue')
axes[0, 0].plot(t[seq_length:split + seq_length], predictions_train,
label='Прогноз (обучение)', linestyle='--',
                color='orange')
axes[0, 0].set_title(f'Обучение, alpha={best_alpha}')
axes[0, 0].set_xlabel('t')
axes[0, 0].set_ylabel('y')
axes[0, 0].legend()
axes[0, 0].grid()
axes[0, 1].plot(best_errors, color='red')
axes[0, 1].set_title('Ошибка обучения')
axes[0, 1].set_xlabel('Эпоха')
axes[0, 1].set_ylabel('MSE')
axes[0, 1].grid()
axes[1, 0].plot(t[split + seq_length:], y[split + seq_length:],
label='Исходный ряд', color='blue')
axes[1, 0].plot(t[split + seq_length:], predictions_test, label='Прогноз
(тест)', linestyle='--', color='green')
axes[1, 0].set_title('Прогнозирование на тесте')
```

```
axes[1, 0].set_xlabel('t')
axes[1, 0].set_ylabel('y')
axes[1, 0].legend()
axes[1, 0].grid()
axes[1, 1].axis('off')
table_data = []
for i in range(min(5, len(Y_test))):
    table_data.append([f"{t[split + seq_length + i]:.1f}",
                       f"{Y_test[i]:.3f}",
                       f"{predictions_test[i]:.3f}",
                       f"{abs(Y_test[i] - predictions_test[i]):.3f}"])

table = axes[1, 1].table(cellText=table_data,
                         colLabels=['t', 'Эталон', 'Прогноз', 'Отклонение'],
                         loc='center',
                         cellLoc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.5)
axes[1, 1].set_title('Таблица отклонений (первые 5 значений)')
plt.tight_layout()
plt.show()
print("\n" + "=" * 60)
print("Результаты обучения (первые 5 значений):")
print("-" * 60)
print(f"{'t':>5} {'Эталон':>10} {'Прогноз':>10} {'Отклонение':>12}")
print("-" * 60)
for i in range(5):
    t_val = t[seq_length + i]
    print(
        f"{t_val:5.1f} {Y_train[i]:10.3f} {predictions_train[i]:10.3f}
{abs(Y_train[i] - predictions_train[i]):12.3f}")

print("\n" + "=" * 60)
print("Результаты прогнозирования (первые 5 значений):")
print("-" * 60)
print(f"{'t':>5} {'Эталон':>10} {'Прогноз':>10} {'Отклонение':>12}")
print("-" * 60)
for i in range(5):
    t_val = t[split + seq_length + i]
    print(f"{t_val:5.1f} {Y_test[i]:10.3f} {predictions_test[i]:10.3f}
{abs(Y_test[i] - predictions_test[i]):12.3f}")
print("\n" + "=" * 60)
print(f"Среднее отклонение на обучении: {np.mean(abs(Y_train -
predictions_train)):.4f}")
print(f"Среднее отклонение на тесте: {np.mean(abs(Y_test -
predictions_test)):.4f}")
print("=" * 60)
```





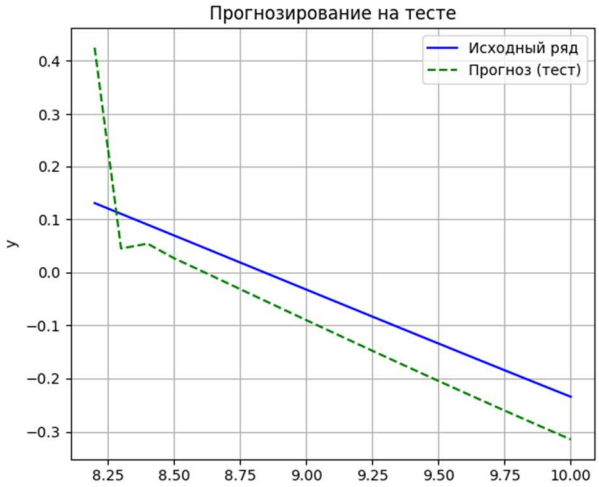


Таблица отклонений (первые 5 значений)

t	Эталон	Прогноз	Отклонение
8.2	0.130	0.425	0.295
8.3	0.110	0.045	0.065
8.4	0.090	0.054	0.036
8.5	0.069	0.027	0.043
8.6	0.049	0.004	0.046

Вывод: изучил применение нелинейной искусственной нейронной сети с одним скрытым слоем для решения задачи регрессии и прогнозирования, реализовал обучение сети на синтетических данных и оценил точность полученной модели.