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.

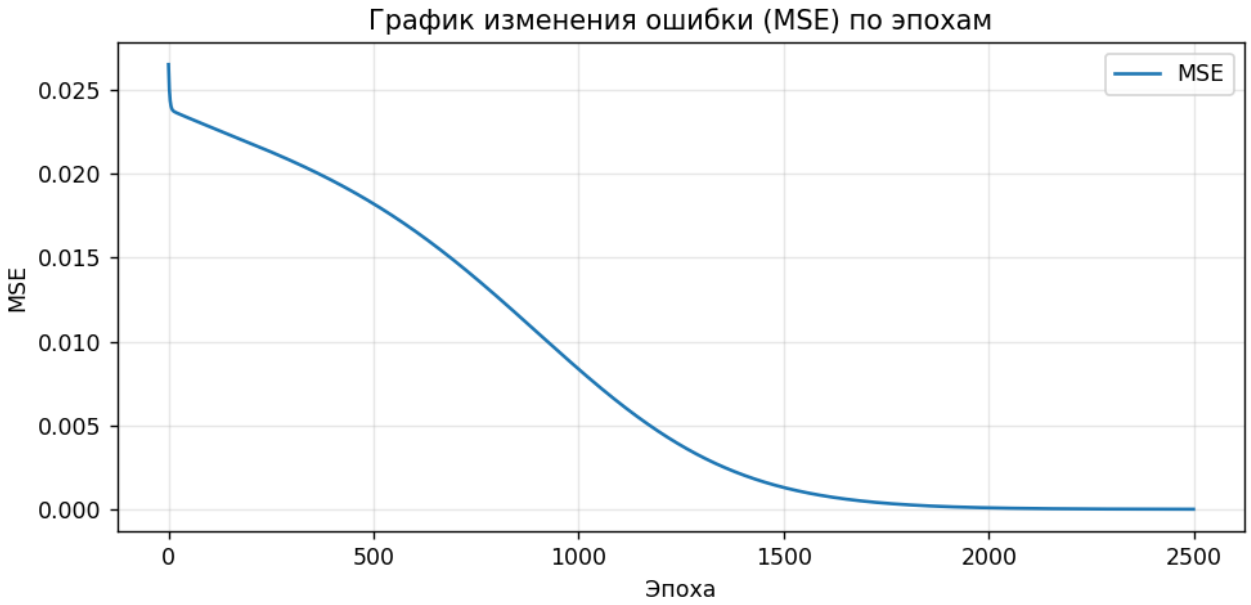
```
import os
import numpy as np
import matplotlib.pyplot as plt

a, b, c, d = 0.2, 0.2, 0.06, 0.2
window size = 8
hidden_size = 3
epochs = 2500
lr = 0.05
train_ratio = 0.7

ar_start_offset_from_train_end = 300

OUT_DIR = "lab5_variant2_results"
os.makedirs(OUT_DIR, exist_ok=True)

def sigmoid(x):
    x_clip = np.clip(x, -50, 50)
    return 1.0 / (1.0 + np.exp(-x_clip))

def sigmoid_deriv_from_activation(a_sigmoid):
    return a_sigmoid * (1.0 - a_sigmoid)

def generate_series(a, b, c, d, x_min=-50, x_max=50, step=0.01):
    x_vals = np.arange(x_min, x_max + step, step)
    y_vals = a * np.cos(b * x_vals) + c * np.sin(d * x_vals)
    return x_vals, y_vals
```

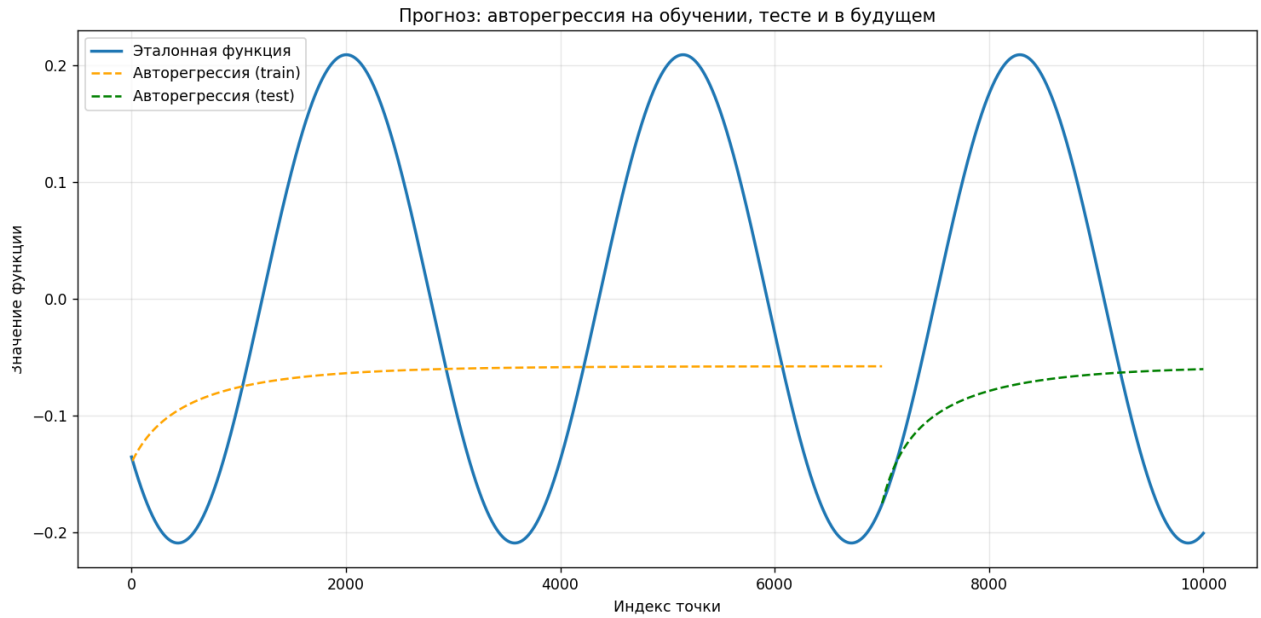
```
def make_supervised_from_series(y_vals, window):
    X, Y = [], []
    for i in range(len(y_vals) - window):
        X.append(y_vals[i:i + window])
        Y.append(y_vals[i + window])
    X = np.array(X, dtype=np.float64)
    Y = np.array(Y, dtype=np.float64).reshape(-1, 1)
    return X, Y

def init_weights(input_size, hidden_size, output_size=1, seed=42):
    rng = np.random.default_rng(seed)
    W1 = rng.normal(0.0, np.sqrt(1.0 / input_size), size=(input_size,
hidden_size))
    b1 = np.zeros((1, hidden_size), dtype=np.float64)
    W2 = rng.normal(0.0, np.sqrt(1.0 / hidden_size), size=(hidden_size,
output_size))
    b2 = np.zeros((1, output_size), dtype=np.float64)
    return W1, b1, W2, b2

def forward(X, W1, b1, W2, b2):
    z1 = X @ W1 + b1
    a1 = sigmoid(z1)
    z2 = a1 @ W2 + b2
    y_pred = z2
    cache = (X, z1, a1, z2)
    return y_pred, cache

def predict(X, W1, b1, W2, b2):
    z1 = X @ W1 + b1
    a1 = sigmoid(z1)
    return a1 @ W2 + b2

def compute_mse(y_pred, y_true):
    err = y_pred - y_true
    return np.mean(err ** 2), err

def backward_and_update(W1, b1, W2, b2, cache, err, lr):
    X, z1, a1, z2 = cache
    N = X.shape[0]

    grad_out = 2.0 * err / N

    dW2 = a1.T @ grad_out
    db2 = np.sum(grad_out, axis=0, keepdims=True)

    da1 = grad_out @ W2.T
    dz1 = da1 * sigmoid_deriv_from_activation(a1)
    dW1 = X.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    W1 -= lr * dW1
    b1 -= lr * db1
    W2 -= lr * dW2
    b2 -= lr * db2

    return W1, b1, W2, b2

def train(X_train, y_train, W1, b1, W2, b2, epochs, lr):
    loss_history = []
    for epoch in range(1, epochs + 1):
        y_pred, cache = forward(X_train, W1, b1, W2, b2)
        loss, err = compute_mse(y_pred, y_train)
        loss_history.append(loss)
```

```
            W1, b1, W2, b2 = backward_and_update(W1, b1, W2, b2, cache, err, lr)

            if epoch % 250 == 0 or epoch == 1:
                print(f"Epoch {epoch:4d} | MSE={loss:.8f}")

    return W1, b1, W2, b2, loss_history

def autoregressive_predict(y_start, W1, b1, W2, b2, steps, window_size):
    preds = []
    window = y_start.copy().reshape(1, -1)

    for _ in range(steps):
        y_next = predict(window, W1, b1, W2, b2).flatten()[0]
        preds.append(y_next)
        window = np.roll(window, -1, axis=1)
        window[0, -1] = y_next

    return np.array(preds)

def main():
    x_vals, y_vals = generate_series(a, b, c, d, x_min=-50, x_max=50)

    X, Y = make_supervised_from_series(y_vals, window_size)
    split = int(train_ratio * len(X))
    X_train, X_test = X[:split], X[split:]
    y_train, y_test = Y[:split], Y[split:]

    W1, b1, W2, b2 = init_weights(window_size, hidden_size, output_size=1,
seed=42)
    W1, b1, W2, b2, loss_history = train(X_train, y_train, W1, b1, W2, b2,
epochs, lr)

    y_train_start = y_train[:window_size].flatten()
    y_train_ar_pred = autoregressive_predict(y_train_start, W1, b1, W2, b2,
steps=len(y_train), window_size=window_size)

    y_test_start = y_train[-window_size:].flatten()
    y_test_ar_pred = autoregressive_predict(y_test_start, W1, b1, W2, b2,
steps=len(y_test), window_size=window_size)

    ar_start_index_points = max(window_size, int(train_ratio * len(y_vals)) -
ar_start_offset_from_train_end)
    window_start = max(0, ar_start_index_points - window_size)
    window_end = min(window_start + window_size, len(y_vals))
    y_start = y_vals[window_start:window_end]
    steps = 300
    y_future_pred = autoregressive_predict(y_start, W1, b1, W2, b2,
steps=steps, window_size=window_size)
    ar_plot_start = window_end
    ar_plot_indices = range(ar_plot_start, ar_plot_start +
len(y_future_pred))

    plt.figure(figsize=(8, 4))
    plt.plot(loss_history, color="tab:blue", label="MSE")
    plt.title("График изменения ошибки (MSE) по эпохам")
    plt.xlabel("Эпоха")
    plt.ylabel("MSE")
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(OUT_DIR, "loss_curve.png"), dpi=150)
    plt.show()
```

```
    plt.figure(figsize=(12, 6))
    plt.plot(range(len(y_vals)), y_vals, lw=2, label="Эталонная функция")

    plt.plot(range(window_size, window_size + len(y_train_ar_pred)),
             y_train_ar_pred, "--", color="orange", label="Авторегрессия
(train)")

    plt.plot(range(window_size + len(y_train_ar_pred),
                   window_size + len(y_train_ar_pred) + len(y_test_ar_pred)),
             y_test_ar_pred, "--", color="green", label="Авторегрессия
(test)")

    plt.title("Прогноз: авторегрессия на обучении, тесте и в будущем")
    plt.xlabel("Индекс точки")
    plt.ylabel("Значение функции")
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(os.path.join(OUT_DIR, "prediction_autoregressive_all.png"),
dpi=150)
    plt.show()

    print(f"\nФайлы сохранены в: {OUT_DIR}")


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

Результаты:





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