Experiment 5

Aim: To implement sequence prediction using RNN.

Tools Used: Python

Theory: Sequence generation using RNNs involves predicting the next element in a sequence based on previous ones. RNNs maintain "memory" through feedback loops, making them suitable for tasks like text or time series generation. Each output is fed back as input for the next step, allowing the generation of sequences one element at a time. However, traditional RNNs struggle with long-term dependencies, so variants like LSTMs and GRUs are often used for better performance.

Code:

```
import numpy as np

class RNN:
    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.Wh = np.random.randn(hidden_size, input_size) * 0.01
        self.Wx = np.random.randn(hidden_size, hidden_size) * 0.01
        self.Wy = np.random.randn(output_size, hidden_size) * 0.01
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))
        self.h = np.zeros((hidden_size, 1))

    def forward(self, x_seq):
        self.h = np.zeros((self.hidden_size, 1))
        outputs = []
        for x in x_seq:
            self.h = np.tanh(np.dot(self.Wh, x) + np.dot(self.Wx, self.h) + self.bh)
            y = np.dot(self.Wy, self.h) + self.by
            outputs.append(y)
        return outputs

    def backward(self, x_seq, dy_seq, learning_rate=0.01):
        dWh, dWx, dWy = np.zeros_like(self.Wh), np.zeros_like(self.Wx), np.zeros_like(self.Wy)
        dbh, dby = np.zeros_like(self.bh), np.zeros_like(self.by)
        dh_next = np.zeros_like(self.h)
        for t in reversed(range(len(x_seq))):
            dy = dy_seq[t]
            dWy += np.dot(dy, self.h.T)
            dby += dy
            dh = np.dot(self.Wy.T, dy) + dh_next
            dh_raw = (1 - self.h * self.h) * dh
            dWh += np.dot(dh_raw, x_seq[t].T)
            dWx += np.dot(dh_raw, self.h.T)
            dbh += dh_raw
            dh_next = np.dot(self.Wx.T, dh_raw)
        self.Wh -= learning_rate * dWh
        self.Wx -= learning_rate * dWx
        self.Wy -= learning_rate * dWy
        self.bh -= learning_rate * dbh
        self.by -= learning_rate * dby

x_seq = [np.random.randn(3, 1) for _ in range(5)]
dy_seq = [np.random.randn(2, 1) for _ in range(5)]
rnn = RNN(3, 4, 2)
outputs = rnn.forward(x_seq)
rnn.backward(x_seq, dy_seq)

for i, output in enumerate(outputs):
    print(f"Output at timestep {i}: {output}")
```

Output:

```
Output at timestep 0: [[-6.11010491e-05]
 [ 3.33636745e-05]]
Output at timestep 1: [[ 9.72167061e-05]
 [-1.25788085e-04]]
Output at timestep 2: [[ 9.25871642e-05]
 [-1.77057989e-04]]
Output at timestep 3: [[ 0.00014987]
 [-0.0012731 ]]
Output at timestep 4: [[ 7.76708915e-05]
 [-7.43939635e-04]]
```

Result: Sequence prediction using RNN has been successfully implemented.

Criteria	Total Marks	Marks Obtained	Comments
Concept (A)			
Implementation (B)			
Performance (C)			
Total			