Experiment 09

Aim:

To design and implement a Recurrent Neural Network (RNN) for text generation.

Apparatus / Software Requirements:

- Google Colab or Jupyter Notebook
- Python 3.8+
- PyTorch Library
- Requests Library (for dataset download)
- GPU (optional, for faster training)
- Internet connection (to access dataset)

Theory:

A Recurrent Neural Network (RNN) is a class of neural networks designed to process sequential data, such as text, speech, or time series. Unlike traditional feedforward networks, RNNs maintain a hidden state that carries information across time steps, allowing the network to capture temporal dependencies.

However, standard RNNs suffer from vanishing and exploding gradients, making them ineffective for long sequences. To overcome this, the Long Short-Term Memory (LSTM) architecture was introduced.
LSTMs include special units called gates (input, forget, and output gates) that control the flow of information and enable the model to remember long-term dependencies effectively.

In this experiment, a character-level LSTM is trained on the Tiny Shakespeare dataset to learn patterns of characters and generate new text sequences that resemble Shakespearean writing.

Procedure:

1. **Dataset Loading:**
   - Download the *Tiny Shakespeare dataset* (a text corpus) using the `requests` library.
2. **Data Preprocessing:**
   - Create a character vocabulary.
   - Convert each character to a numerical index and encode the entire text.
3. **Model Definition:**
   - Define an **LSTM-based RNN** using PyTorch.
   - Include:
     - Embedding layer
     - LSTM layers
     - Fully connected output layer
4. **Training Setup:**
   - Define hyperparameters such as sequence length, learning rate, and batch size.
   - Use **CrossEntropyLoss** and **Adam optimizer**.

    - Train the model for multiple epochs.
5. **Text Generation:**
    - Use a user-provided starting prompt.
    - Generate new text character-by-character using the trained model.
6. **Output Display:**
    - Display generated Shakespeare-style text as final output.

Code:

```
import torch
import torch.nn as nn
import requests

# 1. Download Tiny Shakespeare dataset
url = 'https://raw.githubusercontent.com/karpathy/char-
rnn/master/data/tinyshakespeare/input.txt'
text = requests.get(url).text
print(f"Dataset length: {len(text)} characters")

# 2. Create character vocabulary
chars = sorted(set(text))
char2idx = {ch: i for i, ch in enumerate(chars)}
idx2char = {i: ch for ch, i in char2idx.items()}
vocab_size = len(chars)

# 3. Encode entire text to indices
data = [char2idx[c] for c in text]

# 4. Hyperparameters
seq_length = 100
hidden_size = 256
num_layers = 2
lr = 0.002
batch_size = 64
num_epochs = 50
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 5. Prepare batches
def get_batches(data, batch_size, seq_length):
    num_batches = len(data) // (batch_size * seq_length)
    data = data[:num_batches * batch_size * seq_length]
    data = torch.tensor(data).view(batch_size, -1)
    for i in range(0, data.size(1) - seq_length, seq_length):
        x = data[:, i:i+seq_length]
        y = data[:, i+1:i+seq_length+1]
        yield x.to(device), y.to(device)

# 6. Define the RNN model (LSTM)
class CharRNN(nn.Module):
    def __init__(self, vocab_size, hidden_size, num_layers):
        super().__init__()
        self.embed = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size, num_layers,
batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)
        self.hidden_size = hidden_size
        self.num_layers = num_layers

    def forward(self, x, hidden):
```

```
        x = self.embed(x)
        out, hidden = self.lstm(x, hidden)
        out = self.fc(out.reshape(-1, out.size(2)))
        return out, hidden

    def init_hidden(self, batch_size):
        return (torch.zeros(self.num_layers, batch_size,
self.hidden_size).to(device),
                torch.zeros(self.num_layers, batch_size,
self.hidden_size).to(device))

# 7. Initialize model, loss, optimizer
model = CharRNN(vocab_size, hidden_size, num_layers).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)

# 8. Training loop
model.train()
for epoch in range(num_epochs):
    hidden = model.init_hidden(batch_size)
    total_loss = 0
    for x, y in get_batches(data, batch_size, seq_length):
        optimizer.zero_grad()
        output, hidden = model(x, hidden)
        loss = criterion(output, y.reshape(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        hidden = (hidden[0].detach(), hidden[1].detach())
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {total_loss:.4f}")

# 9. Text generation function
def generate_text_with_context(model, start_str, length=200,
temperature=0.5, window_size=100):
    model.eval()
    generated = start_str
    hidden = model.init_hidden(1)

    input_seq = torch.tensor([char2idx[ch] for ch in generated],
dtype=torch.long).unsqueeze(0).to(device)
    with torch.no_grad():
        for i in range(len(generated) - 1):
            _, hidden = model(input_seq[:, i:i+1], hidden)

        for _ in range(length):
            window = generated[-window_size:]
            input_seq = torch.tensor([char2idx[ch] for ch in window],
dtype=torch.long).unsqueeze(0).to(device)
            output, hidden = model(input_seq[:, -1:].to(device), hidden)
            logits = output / temperature
            probs = torch.softmax(logits, dim=1)
            char_idx = torch.multinomial(probs, num_samples=1).item()
            generated += idx2char[char_idx]

    return generated

# 10. User prompt
prompt = input("Enter your starting text: ")
print("\nGenerated text:\n")
print(generate_text_with_context(model, prompt, temperature=0.8,
window_size=seq_length))
```

Result:

The model successfully trained on the Tiny Shakespeare dataset.
After training, it generated text that resembled Shakespearean writing style, maintaining realistic word patterns and sentence structure.

```
Enter your starting text: Shall I

Generated text:

Shall I be a man;
Where I may be not quickly mouldst see
But it is far a servant not so part with her?
Be calling of mercy, to call it learn
To your commissant.

HERMIONE:
But, told me, go with her.

COMINIU
```

```
Enter your starting text: How are you

Generated text:

How are you a Christand and much?
Who shall call on the mind that hast thou very crown
To speak preferring his burthen arm'd
To the world's marriage and a woman,
We have found with my birthment is noble:
Errow s
```

```
Dataset length: 1115394 characters
Epoch 1/50, Loss: 383.9168
Epoch 2/50, Loss: 284.0148
Epoch 3/50, Loss: 258.4928
Epoch 4/50, Loss: 246.2579
Epoch 5/50, Loss: 238.3392
Epoch 6/50, Loss: 232.9240
Epoch 7/50, Loss: 228.4646
Epoch 8/50, Loss: 224.5125
Epoch 9/50, Loss: 221.3681
Epoch 10/50, Loss: 218.6040
Epoch 11/50, Loss: 216.0289
Epoch 12/50, Loss: 213.8648
Epoch 13/50, Loss: 211.7111
Epoch 14/50, Loss: 209.6872
Epoch 15/50, Loss: 208.0684
Epoch 16/50, Loss: 206.6341
Epoch 17/50, Loss: 205.2118
Epoch 18/50, Loss: 203.7587
Epoch 19/50, Loss: 202.4186
Epoch 20/50, Loss: 201.3712
Epoch 21/50, Loss: 200.4852
Epoch 22/50, Loss: 199.4407
Epoch 23/50, Loss: 198.2106
Epoch 24/50, Loss: 196.7881
Epoch 25/50, Loss: 195.7191
Epoch 26/50, Loss: 194.7964
Epoch 27/50, Loss: 193.7404
Epoch 28/50, Loss: 192.4391
Epoch 29/50, Loss: 191.3039
Epoch 30/50, Loss: 190.4985
Epoch 31/50, Loss: 189.9588
Epoch 32/50, Loss: 189.2725
Epoch 33/50, Loss: 188.2615
Epoch 34/50, Loss: 187.3273
Epoch 35/50, Loss: 186.5760
Epoch 36/50, Loss: 186.0110
Epoch 37/50, Loss: 185.5798
Epoch 38/50, Loss: 185.3846
Epoch 39/50, Loss: 184.9918
Epoch 40/50, Loss: 184.3855
Epoch 41/50, Loss: 183.7020
Epoch 42/50, Loss: 182.7044
Epoch 43/50, Loss: 181.9200
Epoch 44/50, Loss: 181.0929
Epoch 45/50, Loss: 180.4792
Epoch 46/50, Loss: 180.0364
Epoch 47/50, Loss: 179.3260
Epoch 48/50, Loss: 178.7530
Epoch 49/50, Loss: 178.1865
Epoch 50/50, Loss: 177.8419
```

Conclusion:

The experiment successfully demonstrated the design and implementation of a Recurrent Neural Network (RNN) using LSTM for text generation.
The model was able to learn character-level patterns from the input corpus and generate coherent Shakespeare-like text.
Thus, RNNs with LSTM units effectively capture sequential dependencies and can be applied to various natural language generation tasks.