Report On Text Generation Using Recurrent Neural Networks (RNNs)

Objective

The goal of this project was to develop a text generation model using **Recurrent Neural Networks (RNNs)** to generate novel text sequences. The model was trained on a corpus of Shakespearean texts (a collection of his plays and sonnets) to produce new sequences that mimic the writing style and linguistic structure of the original works.

Approach:

1. Dataset:

- **Shakespearean Texts**: A dataset containing Shakespeare's plays and sonnets was used for training the model. The dataset consisted of over 5 million characters, which provided a rich source for language modeling. The data was preprocessed by:
    - Lowercasing all characters.
    - Removing unnecessary punctuation and special characters.
    - Tokenizing the text into sequences of characters or words.

2. Model Architecture:

We opted to use an **RNN-based model** for text generation. The architecture consists of the following components:

- **Embedding Layer**: The model first transforms each character or word into a vector representation using an embedding layer. This allows the model to learn distributed representations of words or characters.
- **Recurrent Layers**:
    - **LSTM (Long Short-Term Memory) Layers**: LSTM units were chosen to handle the vanishing gradient problem and learn long-range dependencies within the text.
    - The network consists of 2 LSTM layers with a defined number of units (e.g., 128 units).
- **Fully Connected Layer**: After processing through the LSTM layers, the output is passed through a dense layer with a softmax activation function to predict the next character or word based on the current context.

- **Dropout**: Dropout was applied to reduce overfitting during training by randomly setting some units to zero during each update.

3. Preprocessing:

- **Text Tokenization**: The text was tokenized into characters or words (depending on the model's design).
- **Sequence Creation**: Sequences of text were created with a fixed length (e.g., 100 characters or words per sequence), with each sequence feeding into the model.
- **One-hot Encoding**: Each character or word was one-hot encoded into a vector, which was used as input for the neural network.

4. Training:

- **Loss Function**: The categorical cross-entropy loss function was used, as the task is a multi-class classification problem where each class corresponds to a character or word.
- **Optimizer**: The Adam optimizer was used to efficiently train the model.
- **Epochs**: The model was trained for 20 epochs to learn the patterns in the text and generate coherent sequences.

5. Text Generation:

After training, the model was used to generate text sequences. A given seed text (e.g., the first few characters of a Shakespearean sentence) was fed into the model, and the model iteratively predicted the next character or word in the sequence.

The generation process:

1. **Seed Input**: The model begins with a given input sequence (e.g., "To be, or not to be").
2. **Prediction**: The model predicts the next character or word.
3. **Sampling**: The predicted character or word is added to the input sequence.
4. **Iteration**: The process repeats to generate a sequence of a desired length.

6. Evaluation:

- **Text Coherence**: Generated text was qualitatively evaluated for coherence and fluency. Although RNNs with LSTM units can capture long-term dependencies, they

sometimes produce text that is repetitive or lacks overall coherence over longer sequences.

- **Creativity and Style**: The model was expected to generate text that imitates the writing style of Shakespeare, including archaic phrasing, rhythm, and syntactic structure. Generated sequences were compared with the original Shakespearean corpus to assess how well the model learned the style.

Results

- Training Loss:
  During training, the model showed decreasing loss with each epoch, indicating successful learning of text patterns from the dataset. The model was able to generate more fluent text with continued training.
- Evaluation of Generated Text
- Fluency: The generated text is fluent, with correct sentence structure and appropriate use of words.
- Creativity: The model was able to generate creative new lines that follow the Shakespearean style, although some outputs were repetitive or nonsensical.
- Coherence: The generated text is mostly coherent in short sequences, but over long passages, the model sometimes produced nonsensical or contextually mismatched sentences. This issue arises due to the limited memory of standard RNNs compared to more advanced models like Transformers.

Analysis

Strengths:

- **Mimicking Style**: The model successfully learned to replicate the writing style of Shakespeare, generating text with similar linguistic features and rhythm.
- **Simplicity and Interpretability**: RNNs with LSTM layers offer a relatively simple yet effective architecture for sequence generation tasks, and the process of learning and generating text is easily interpretable.

Weaknesses:

- **Long-Term Dependencies**: While LSTMs are better than vanilla RNNs at capturing long-range dependencies, they still struggle with very long sequences, sometimes resulting in repetition or incoherent text over time.

- **Training Time**: The training process can be slow and computationally intensive, especially with large datasets. To address this, using more advanced architectures like Transformers (e.g., GPT) could yield better results.
- **Repetition in Text**: In longer generated sequences, the model sometimes repeated phrases or words, which is a common issue with RNNs. This could be improved with advanced sampling techniques or architectures.

Conclusion

The RNN model, particularly with LSTM units, was effective in generating text that mimicked the style of Shakespeare. The model demonstrated the capability to produce creative and coherent text, albeit with limitations in generating longer and more contextually accurate sequences. For more advanced text generation, transformer-based models, such as GPT, would be better suited due to their ability to handle long-range dependencies and produce more fluent outputs. Nonetheless, this project showcases the potential of RNN-based models for creative text generation tasks, especially with structured and relatively simpler datasets.