Text Generation using RNN

1 Objective

The aim of this experiment is to explore text generation using **Recurrent Neural Networks (RNNs)** and to understand the impact of different word representations:

1. One-Hot Encoding
2. Trainable Word Embeddings

As part of the experiment, a basic RNN is first implemented from scratch using **NumPy** to understand its internal workings. The RNN model is then trained using **PyTorch** on a dataset of 100 poems, and the performance of both encoding techniques is compared.

2 Dataset

Use the provided dataset of 100 poems for training your text generation model. The dataset consists of multiple lines of poetry, which will be used to generate text sequences.

3 Implementation

Part 1: Implement RNN From Scratch

Before proceeding with model training , implement a basic RNN from scratch using NumPy to understand the internal workings of recurrent neural networks.

Once the NumPy-based RNN implementation is completed proceed with the following sections.

Part 2: One-Hot Encoding Approach

3.2.1 Preprocessing

- Tokenize the text into words.
- Convert each word into a one-hot vector.

3.2.2 Model Architecture

- Use an RNN model.
- The input should be one-hot encoded word sequences.
- Train the model to predict the next word in a sequence.

3.2.3 Implementation Steps

1. Tokenize the dataset and create a vocabulary.
2. Convert words into one-hot encoded vectors.
3. Define an RNN model using PyTorch.
4. Train the model using the dataset.
5. Generate text using the trained model.

Part 3: Trainable Word Embeddings Approach

3.3.1Preprocessing

- Tokenize the text into words.
- Convert each word into an index.

3.3.2 Model Architecture

- Use an embedding layer in the RNN model.
- Train the embedding layer along with the model.
- Predict the next word in a sequence.

3.3.3 Implementation Steps

1. Tokenize the dataset and create a vocabulary.
2. Convert words into indexed sequences.
3. Define an RNN model with an embedding layer using PyTorch.
4. Train the model and compare performance with the one-hot encoding method.
5. Generate text using the trained model.

4 Comparison and Analysis

- Compare the training time and loss for both methods.
- Evaluate the quality of generated text.
- Discuss the advantages and disadvantages of each approach.

5 Submission Guidelines

- Share the github repo link including the readme file with proper instruction. *(Note everyone must implement the model for each encoding method (One-Hot Encoding and Trainable Embeddings). Compare their performance and include your observations in the report.)*