Midterm Project: CSE 587

Archisman Ghosh
CSE Department
Penn State University
State College, PA, USA
apg6127@psu.edu

Debarshi Kundu
CSE Department
Penn State University
State College, PA, USA
dqk5620@psu.edu

***Abstract*—Poetry generation using deep learning presents a challenge in modeling the complexities of human language, creativity, and structure. Traditional rule-based approaches lack the adaptability required for generating diverse and meaningful poetry. This project explores the application of recurrent neural networks, specifically LSTMs, to generate poetry by learning linguistic patterns from a corpus of classical and modern poems. The motivation behind this research is to enhance computational creativity, enabling AI to assist in literary composition, creative writing, and automated content generation. The model is trained on publicly available poetry datasets using pre-trained word embeddings such as GloVe to improve contextual understanding. Performance is evaluated using perplexity scores, loss values, and accuracy. The results indicate that LSTMs effectively capture poetic structures and produce meaningful text, demonstrating their potential for real-world applications in AI-assisted literature, creative writing tools, and automated storytelling.**

***Index Terms*—Recurrent Neural Networks, Long Short-Term Memory Networks, Convolution Neural Networks**

I. Introduction

Text completion is a fundamental task in natural language processing, playing a crucial role in applications such as predictive text, chatbot responses, and creative writing assistance. Effective text generation requires models that understand context, grammar, and coherence to produce meaningful and natural-sounding output [1]. Rule-based approaches, which rely on predefined templates and heuristics, struggle to achieve this level of sophistication. They lack adaptability, making it difficult to generate diverse and contextually relevant text, especially in complex forms like poetry where rhythm, structure, and semantics must align harmoniously. Recurrent neural networks, particularly LSTMs, provide a more effective solution by learning sequential dependencies from large text corpora. Unlike rule-based methods, LSTMs dynamically adjust to different writing styles and structures by capturing long-term dependencies, allowing for more coherent and context-aware text generation. Pre-trained word embeddings, such as GloVe, further enhance performance by providing rich semantic understanding. This project applies LSTMs to poetry generation, training a model on publicly available poetic datasets. The effectiveness of the approach is evaluated using perplexity scores from the loss values, ultimately leading to enhancement in moving the research in text completion forward.

A. Motivation and Problem Statement

Generating poetry with artificial intelligence is a challenging task due to the need for creativity, coherence, and contextual understanding. Unlike structured text generation, poetry requires maintaining rhythm, semantic depth, and stylistic consistency, making it difficult for traditional rule-based approaches to produce meaningful and diverse compositions. Rule-based systems rely on predefined templates and grammatical rules, which fail to capture the flexibility and artistic nuances required in poetry.

The problem is that existing deterministic methods lack the ability to adapt to different poetic styles, recognize long-range dependencies, and generate text that feels natural and expressive. Deep learning, particularly LSTMs, provides a solution by learning from a large corpus of poetry, capturing linguistic patterns, word relationships, and poetic structures dynamically. This project explores how LSTM-based models can improve poetry generation, ensuring better fluency, coherence, and diversity. The goal is to develop a framework that can generate human-like poetry with creativity and contextual awareness.

B. Contributions

The major contributions of this project are as follows:

1) We develop an LSTM model for the task of text completion with respect to generating poetry from a prompt.
2) We analyze the efficacy of the model based on the perplexity score.

C. Report Structure

Section II provides a background and related works. Section III covers the dataset. Section IV describes the procedure and architecture of the LSTM-based classifier. Section V presents the results. We discuss the results in Section VI and conclude in Section VII.

II. Background

A. Long Short-Term Memory

Long Short-Term Memory (LSTM) [2] networks are a specialized type of recurrent neural network (RNN) designed to capture long-range dependencies in sequential data. Standard RNNs suffer from the vanishing gradient problem, where long-term dependencies are lost during training due to repeated multiplication of small gradients. LSTMs address this issue using a gated mechanism that selectively retains or forgets information through time. An LSTM unit consists of three key gates: forget gate, input gate, and output gate. For an input sequence x_t, the LSTM cell updates its state using the forget, input, and output gates. The forget gate, $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$, determines how much past information should be retained. The input gate controls how much new information is added. It is represented by i_t:

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$\tilde{C}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c)$$

The Cell State updates itself by $C_t = f_t C_{t-1} + i_t \tilde{C}_t$; and the output gate ($o_t$) determines the final output.

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$h_t = o_t \tanh(C_t)$$

Here, σ is the sigmoid activation function, and $tanh()$ ensures smooth updates. LSTMs effectively model long-term dependencies, making them ideal for text generation tasks like poetry, where maintaining context and style is crucial.

B. GloVe

GloVe (Global Vectors for Word Representation) [3] is an unsupervised learning method for obtaining word embeddings based on co-occurrence statistics from large text corpora. Unlike Word2Vec, which relies on local context windows, GloVe captures global word relationships by factorizing a word co-occurrence matrix. Key GloVe parameters include the vector dimension (e.g., 50, 100, or 300), window size, and number of iterations. Larger dimensions improve accuracy but require more computation. GloVe embeddings, pre-trained on massive datasets like Wikipedia and Common Crawl, are widely used in NLP tasks such as text classification, machine translation, and poetry generation. The core idea is that the ratio of word co-occurrence probabilities encodes meaningful semantic relationships. Given a word-word co-occurrence matrix X, where X_{ij} represents how often the word j appears in the context of word i, GloVe minimizes the objective function

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where, w_i and $\tilde{w}_j$ are word vectors and b_i and $\tilde{b}_j$ are biases, and $f(X_{ij})$ is a weighing function. GloVe effectively captures semantic similarity, making it ideal for NLP tasks like poetry generation and text classification.

III. Dataset

The dataset used for training the LSTM-based poetry generation model is the Shakespeare corpus, a publicly available dataset from TensorFlow Datasets [4]. This dataset consists of 40,000 lines of Shakespeare from a variety of Shakespeare's plays, including sonnets, and poems. It provides a structured representation of poetic text, making it ideal for training language models to understand rhythmic and stylistic patterns. Each line of text is treated as a training sample, allowing the model to learn sentence structure, vocabulary, and sequential dependencies.

IV. Procedure

A. Dataset Preprocessing

Before training the LSTM-based poetry generation model, the dataset undergoes several preprocessing steps to convert raw text into a structured format suitable for deep learning. These steps include tokenization, sequence generation, and embedding initialization using GloVe.

1) Tokenization: The dataset consists of lines of poetry, which need to be converted into numerical representations for the model. This is done using the Keras Tokenizer, which assigns a unique integer to each word. A padding of one is done for the total vocabulary size.

2) Sequence Generation: Each line of poetry is transformed into sequences where each sequence is a progressive n-gram of a line. This helps the LSTM to learn the natural progression of words and predict the next word in a sequence. All sequences are padded to ensure uniformity in the size of the inputs. Each sequence is divided into the input and target sub-sequences and the target labels are converted to one-hot encoded vectors, making it suitable for categorical classification.

3) Embedding: To improve word representations, we use pre-trained GloVe embeddings (50-dimensional vectors per word). Instead of learning embeddings from scratch, we map each word in our vocabulary to its corresponding GloVe vector. We follow it up by creating an embedding matrix, where each word in our dataset is assigned its corresponding pre-trained vector. The unknown words are initialized to zeroes.

B. LSTM Architecture

The LSTM-based poetry generation model (Table I) consists of an embedding layer that maps words to pre-trained GloVe vectors, providing semantic understanding. It is followed by three stacked LSTM layers with 256, 128, and 64 neurons, enabling the model to capture both short-term and long-term dependencies in text sequences. A dense layer with 256 neurons and ReLU activation refines learned features, while a dropout layer (0.3) prevents overfitting. Finally, a softmax-activated dense layer predicts the next word based on context. *Total Words* is 12633 for our chosen dataset.

TABLE I
LSTM Model Architecture

Layer	**# Neurons**	**Activation Function**
LSTM	256	*Tanh*
LSTM	128	*Tanh*
LSTM	64	*Tanh*
Dense	256	ReLU
Dropout	*N/A*	*N/A*
Dense	*Total Words*	Softmax

C. Training

The LSTM is trained using the Adam optimizer, an adaptive learning rate optimization algorithm that combines momentum (from SGD) and RMSProp, making it well-suited for sequential data like poetry. The default learning rate for Adam is 0.001, ensuring stable and efficient updates to the weights. The loss function used is categorical cross-entropy, which is appropriate for multi-class classification, as the model predicts the next word from a vocabulary of possible words. The training runs for 100 epochs, allowing the model sufficient time to learn meaningful word patterns while monitoring for overfitting. A batch size of 64 balances computational efficiency and convergence stability.

V. Results

A. Simulation Setup

The classifier was implemented using TensorFlow libraries and run on a machine with 16GB RAM on an Intel Core i7-6700 CPU at a clock frequency of 3.40 GHz.

B. Classification

We present the trends in training in Fig. 1. From the figure we can observe that there is an increasing trend in the accuracy with the peak reaching $\sim 35\%$ and the loss is decreasing, which denotes a decent convergence over 100 epochs. On increasing the epochs of training a better convergence will be noticed. The loss function used in the classifier is categorical cross-entropy, which is applied in multi-class classification tasks where the model predicts probabilities over multiple possible classes. Since the poetry generation model predicts the next word in a sequence from a vocabulary of words, categorical cross-entropy ensures the model learns to assign high probabilities to the correct words while minimizing incorrect predictions. Categorical cross-entropy is calculated as

$$J = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

where N is the total number of training samples, C is the number of classes (total vocabulary size), y_{ij} is a one-hot encoded ground truth label (1 if word j is correct for sample i, otherwise 0), and $\hat{y}_{ij}$ is the predicted probability of word j.

C. Metrics

The classification report and perplexity score [5] of the model provide a detailed evaluation of the model's performance. Perplexity is a widely used evaluation metric for language models, measuring how well a probabilistic model predicts a given sequence. A lower perplexity indicates that the model assigns higher probabilities to correct words, meaning it is more confident and fluent in text generation. Conversely, a higher perplexity suggests uncertainty and poor generalization. Mathematically, perplexity score is $e^{\mathcal{L}}$, where $\mathcal{L}$ is the average categorical cross-entropy loss per word. Since perplexity is exponentially related to cross-entropy loss, small improvements in loss can lead to significant reductions in perplexity. It is an essential metric for evaluating text generation models, ensuring they produce fluent, contextually accurate outputs. For our model, at 100 epochs we find the perplexity score to be ~ 21, which is considered good [5] indicating high-quality text generation.

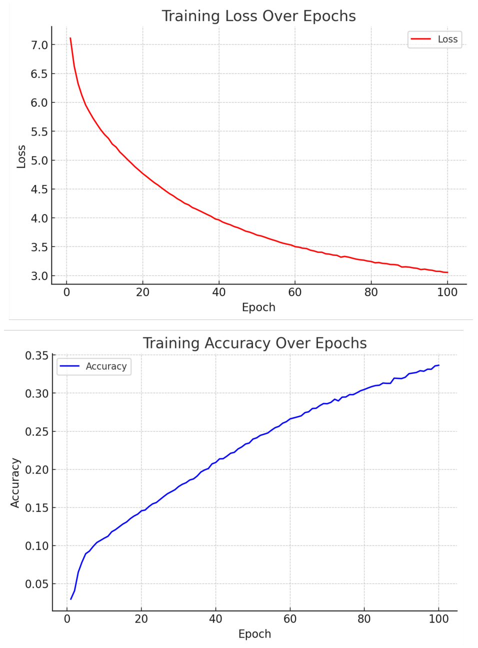

Fig. 1. Plot representing the trend in training accuracy and loss. We can observe that the trend in loss is decreasing and that the accuracy is increasing which is desirable.

VI. Discussion

In this section, we discuss the approach to the problem, potential improvements, and future research.

A. Dataset

To improve the task of poetry completion, additional diverse poetry datasets can be utilized. One such dataset is the Gutenberg Poetry Corpus, which contains over 3 million lines from classical and modern poetry, providing a vast range of styles and structures. Another useful dataset is the Poetry Foundation Corpus, which includes contemporary poems, helping the model generalize beyond classical styles. The Haiku Dataset can also be integrated to expose the model to shorter, structured poetic forms. Using multi-lingual poetry datasets could further enhance stylistic richness and improve the model's adaptability to different poetic conventions. For text completion in general, incorporating datasets such as WikiText, OpenWebText, and Project Gutenberg Books can help the model learn broader linguistic structures and grammar rules. Fine-tuning the model on domain-specific corpora (e.g., Shakespearean text for classical poetry, and modern news articles for structured text completion) improves contextual understanding. Another approach is unsupervised pretraining on large-scale corpora followed by fine-tuning on poetry, allowing the model to first learn general language structures before specializing in poetic form. Data augmentation techniques, such as paraphrasing and controlled text generation, can further improve robustness.

B. LSTM

A simple LSTM architecture (Table I was chosen for its ease of implementation and efficiency in solving the task. However, augmenting the existing model with CNN layers (Table II) improves the efficacy to some extent. The CNN + LSTM hybrid model performs better than a pure LSTM because it combines local pattern extraction with long-range dependency learning. The Conv1D layer captures n-gram features, rhyme structures, and syllabic patterns, which are crucial for poetry generation. MaxPooling1D reduces noise, ensuring the LSTM layers focus on meaningful patterns. The stacked LSTMs then model sequential dependencies, improving coherence and fluency. This combination allows the model to understand both micro-level word relationships and macro-level poetic structure, leading to more diverse, contextually relevant, and stylistically consistent poetry. This model (Table II) is trained under the same simulation setup and hyperparameters for 20 epochs (due to a lack of resources). From Fig. 2, we can observe that the CNN+LSTM model obtains a perplexity score of $\sim$ 92 after 10 epochs of training under the same conditions as the LSTM model ($\sim$ 180). Given the decreasing trend observed in the loss values during the training of the CNN+LSTM model, the final text generation will be better than the singular LSTM model.

TABLE II
CNN + LSTM MODEL ARCHITECTURE

Layer	# Neurons/Filters	Activation Function
Conv1D	256	ReLU
MaxPooling1D	*N/A*	*N/A*
LSTM	256	*Tanh*
LSTM	128	*Tanh*
LSTM	64	*Tanh*
Dense	256	ReLU
Dropout	*N/A*	*N/A*
Dense	*Total Words*	Softmax

C. Future Work

To improve on this project idea, we could extend the study in a plethora of ways:

1) Embedding Techniques: The model currently uses GloVe, which provides static embeddings, meaning each word has the same vector representation regardless of context. This limits the model's ability to differentiate between polysemous words. A generic direction would be to experiment with contextual embeddings like ELMo [6] or FastText [7], where the embedding changes based on surrounding words, given the fact that they help capture deeper semantic meaning and poetic nuances, making the generated text more natural and contextually aware. Also, an idea would be to use trainable embeddings. Using a hybrid approach, where the first embedding layer is initialized with pre-trained vectors (GloVe/FastText) but remains trainable, allowing fine-tuning, the model will benefit from pre-trained knowledge while adapting to the poetry dataset's unique language structures.

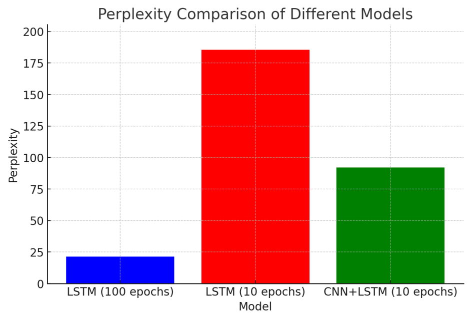

Fig. 2. Plot representing the improvement in perplexity score on using CNN+LSTM model in place of a single LSTM model. The hybrid CNN+LSTM model has been trained for 10 epochs. For an apple-to-apple comparison, we provide the perplexity score for the LSTM model (both 10 and 100 epochs).

2) LSTM: We show good results of text generation in the form of poetry using an LSTM network and a hybrid CNN+LSTM network. The work could be extended by experimenting with adaptive learning rates, and hyperparameter tuning. Since we are limited by the scope of the project to use CNNs and/or RNNs, experiments can be conducted with BiLSTMs to capture the context from both ends providing better text-generation capabilities. Implementing Variational Autoencoders to introduce randomness, allowing for the creation of more creative, diverse, and stylistically unique poetry and improving the quality of the generated text, can be explored. The evaluation metric being set as a reward function for an RL model will help it capture the feedback on rhythm, style, and creativity and generate better text. Eventually, experimenting with Transformers and fine-tuning the self-attention mechanisms to allow them to capture the global context efficiently can be done.

3) Lessons Learnt: One key lesson learned from this project is the importance of model architecture choice in text generation tasks. Traditional LSTMs, while effective at capturing sequential dependencies, struggle with long-range context retention compared to modern architectures like Transformers. Integrating CNN layers with LSTMs improved local feature extraction, enhancing the quality of generated poetry. Another important insight is the impact of word embeddings on model performance. Pre-trained embeddings like GloVe provide semantic knowledge but lack contextual adaptability. Finally, the evaluation of generative models requires metrics beyond accuracy, as traditional classification metrics do not fully capture fluency, coherence, and creativity. The use of perplexity scores proved essential in assessing model performance, guiding further refinements in training strategies.

VII. CONCLUSION

The project successfully demonstrates the effectiveness of LSTM-based models for poetry generation, highlighting their ability to capture linguistic patterns, structure, and coherence in poetic text. The integration of CNN layers with LSTMs further improved performance by extracting local features while maintaining long-range dependencies. The evaluation metrics, particularly perplexity scores, confirm the model's ability to generate fluent and contextually relevant poetry. Overall, this project establishes a strong foundation for AI-assisted poetry generation, demonstrating the potential for deep learning in creative writing applications.

References

[1] Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Pretrained language models for text generation: A survey, 2022.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[4] Andrej Karpathy. char-rnn. https://github.com/karpathy/char-rnn, 2015.

[5] Nayeon Lee, Yejin Bang, Andrea Madotto, Madian Khabsa, and Pascale Fung. Towards few-shot fact-checking via perplexity, 2021.

[6] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2017.