# Word-Level LSTM for Sentence Completion: A Study on Shakespeare's Plays

Syed Muhammad Murtaza Kazmi
*Department of Computer Science*
*National University of Computer & Emerging Sciences*
Islamabad, Pakistan
i210685@nu.edu.pk.com

***Abstract*—This report presents the development of a word-level Long Short-Term Memory (LSTM) model for sentence completion, trained on Shakespeare's plays. The model predicts the next word in a sequence, providing real-time suggestions in a user-friendly interface. We explore the impact of different hyperparameters on the model's performance and evaluate the coherence and fluency of the generated sentences.**

***Index Terms*—LSTM, Word Prediction, Natural Language Processing, Shakespeare, Deep Learning**

## I. Introduction

Natural Language Processing (NLP) has seen significant advancements with the advent of deep learning techniques, particularly Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) networks. Word prediction is a fundamental task in NLP, with applications ranging from text autocompletion to assistive technologies.

This project aims to build a word-level LSTM model capable of predicting the next word in a sequence, trained on a text dataset consisting of Shakespeare's plays [1]. The model is integrated into a web-based user interface, allowing users to input partial sentences and receive real-time word suggestions. This report details the methodology, results, and discussions surrounding the model's development and performance.

## II. Methodology

### A. Dataset

The dataset used comprises the collected works of William Shakespeare, available from Kaggle [1]. It contains various plays, each broken down into individual lines spoken by characters.

### B. Data Preprocessing

Data preprocessing is a crucial step in preparing the text data for model training. The following steps were undertaken:

- **Text Cleaning**: Lowercasing, removal of punctuation and numbers, and stripping of extra whitespaces.
- **Tokenization**: The text was tokenized into words using NLTK's word tokenizer [2].
- **Vocabulary Creation**: A vocabulary of unique words was created, mapping each word to an integer index.
- **Sequence Creation**: The text was converted into sequences of integers, with overlapping sequences of fixed length (10 words per sequence). Each sequence's target is the next word following the sequence.
- **Train-Test Split**: The data was split into training and validation sets with a 90-10 split.

### C. Model Architecture

An LSTM-based neural network was constructed without using pre-trained embeddings like GloVe. The model architecture is summarized as follows:

- **Embedding Layer**: An embedding layer of size (27,365, 50), transforming integer word indices into dense vector representations. Number of parameters: 1,368,250.
- **LSTM Layer**: A two-layer LSTM with an input size of 50 and a hidden size of 256, using a dropout of 0.2. Number of parameters: 841,728.
- **Dropout Layer**: A dropout layer with a rate of 0.2 to prevent overfitting.
- **Fully Connected Layer**: A linear layer mapping from 256 to the vocabulary size of 27,365. Number of parameters: 7,032,805.

The total number of trainable parameters is 9,242,783.

### D. Training Procedure

The model was trained using the Adam optimizer with a learning rate of 0.001 and the Cross-Entropy Loss function. Training was conducted over 20 epochs with a batch size of 256.

## III. Results

### A. Training and Validation Loss

The training and validation losses over 20 epochs are presented in Table I.

TABLE I: Training and Validation Loss over 20 Epochs

| Epoch | Train Loss | Validation Loss |
|---|---|---|
| 1 | 4.4834 | 6.8080 |
| 2 | 4.4496 | 6.8541 |
| 3 | 4.4170 | 6.8943 |
| 4 | 4.3875 | 6.9027 |
| 5 | 4.3620 | 6.9457 |
| 6 | 4.3359 | 6.9706 |
| 7 | 4.3120 | 7.0048 |
| 8 | 4.2877 | 7.0185 |
| 9 | 4.2690 | 7.0515 |
| 10 | 4.2465 | 7.0781 |
| 11 | 4.2271 | 7.0948 |
| 12 | 4.2087 | 7.1265 |
| 13 | 4.1903 | 7.1445 |
| 14 | 4.1745 | 7.1623 |
| 15 | 4.1546 | 7.1861 |
| 16 | 4.1405 | 7.2104 |
| 17 | 4.1271 | 7.2259 |
| 18 | 4.1110 | 7.2561 |
| 19 | 4.0979 | 7.2668 |
| 20 | 4.0849 | 7.2969 |

As shown, the training loss decreased steadily over the epochs, indicating that the model was learning from the data. However, the validation loss began to plateau and slightly increase after initial epochs, suggesting potential overfitting.

### B. Examples of Sentence Completion

The model was integrated into a Streamlit app that provides next-word predictions in real-time. Examples of user inputs and the model's predictions are shown in Table II.

TABLE II: Examples of Next-Word Predictions

| User Input | Predicted Next Words |
|---|---|
| how are you doing | there, to, this |
| will you please | you, her, him |
| i want | the, him, it |
| my name | is, and, the |
| how old are | you, the, her |
| what is your | name, own, will |
| can you help | me, him, her |
| the weather is | not, as, so |
| i love | thee, you, him |
| she said that | he, i, she |

## IV. Discussion

### A. Analysis of Sentence Coherence

The model's predictions often include common words like "the", "you", and "him", reflecting frequent words in the training corpus. While the predictions may not always be contextually accurate, they are syntactically plausible, especially considering the archaic language of Shakespeare's plays.

### B. Effects of Hyperparameters

Adjusting hyperparameters such as sequence length, embedding dimensions, and hidden units can significantly impact the model's performance. A longer sequence length provides more context but increases computational complexity. The embedding dimension and hidden units were chosen to balance performance and resource constraints.

### C. Model Complexity

The model comprises approximately 9.2 million trainable parameters. The large vocabulary size (27,365 unique words) contributes to the high number of parameters in the embedding and fully connected layers. This complexity allows the model to capture a wide range of vocabulary but also poses challenges in terms of computational resources and overfitting.

### D. Challenges Encountered

One challenge was the model's tendency to predict high-frequency words, possibly due to the imbalanced word distribution in Shakespeare's texts. The validation loss plateauing suggests overfitting, indicating the need for regularization techniques or more diverse data.

## V. Conclusion

This project successfully developed a word-level LSTM model capable of predicting the next word in a sequence, trained on Shakespeare's plays. The integration into a web interface allows for interactive user engagement. The code for this project is available on GitHub at:

https://github.com/smmk47/word-prediction-lstm

Future work could involve using larger and more contemporary datasets, implementing attention mechanisms, or incorporating pre-trained embeddings to enhance performance.

## VI. Research Questions

To further explore the capabilities and limitations of our model, we pose the following research questions:

1) **How do different sequence lengths affect the model's predictions?**
   Analyze the trade-offs between context length and computational efficiency in sequence modeling.
2) **What are the effects of reducing vocabulary size?**
   Determine how limiting the vocabulary impacts model performance and training time.
3) **Is a character-level LSTM more effective for this task?**
   Compare the performance of character-level models with word-level models in capturing Shakespeare's writing style.
4) **What strategies can mitigate overfitting on limited datasets?**
   Investigate regularization techniques and data augmentation methods suitable for specialized corpora.
5) **How do different regularization techniques influence generalization?**
   Evaluate the impact of varying dropout rates and other regularization methods on model performance.
6) **Can syntactic or semantic information enhance predictions?**

Consider incorporating part-of-speech tags or semantic embeddings to improve contextual accuracy.

These research questions aim to guide future work and contribute to the development of more advanced models in natural language processing.

## References

## References

[1] Kaggle, "Shakespeare Plays Dataset," https://www.kaggle.com/datasets/kingburrito666/shakespeare-plays, Accessed: October 2023.

[2] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.