NEXT WORD PREDICTION USING LSTM MODEL

Submitted by:
Akhilesh R
125018003

TABLE OF CONTENTS

ABSTRACT

This project intends to develop and improve the text generation model developed using an LSTM-based architecture of recurrent neural networks (RNNs). The focus of this project was to predict the next word within the sequence given by the prior context-it can be used in a number of NLP tasks related to natural language modeling and automated text generation. It is trained on a custom corpus of sentences, first on a tiny dataset and then gradually increasing it by public text corpora, like the dataset available at Gutenberg, to enhance performance.

The critical techniques applied in the model are tokenizing the text data, mapping sequences into numerical representations, and using LSTMs on long-term dependencies in the text. Techniques to enhance the accuracy of the model and prevent overfitting include dropout layers, early stopping, and use of bidirectional LSTMs. Hyperparameter tuning applies optimization for the number of LSTM units, embedding dimensions, and learning rates, where tools like Keras Tuner can be used to explore the best possible configurations.

The very first experiments succeeded in validating at a 33% accuracy, which in itself improved to 38% by the tuning stage. The next improvements of this project will come from deeper models, bigger datasets, and better embedding techniques. This work also
illustrates the ability of using LSTM models towards coherent and contextually relevant text generation. Finally the accuracy was increased to 85%.
Additionally, the GRU and HMM models were also being implemented and cross checked with the LSTM model.

INTRODUCTION

(a) Significance of the Dataset

Text generation is one of the main tasks undertaken in NLP, which finds use in predictive typing up to complex content generation. The dataset used here has played a critical role because the quality, size, and diversity in the training corpus directly influence the ability of the model towards learning a pattern in language. We begin with a small, domain-specific corpus of sentences and then scale up to include bigger publicly available datasets such as the well-known Gutenberg corpus, offering rich linguistic structures and vocabulary. This helps the model generalize better and produce meaningful text in different contexts.

(b) Objective: Task, Purpose, and Expected Outcome

The aim is to develop a text generation model that should predict the next word in a sequence given preceding context. For this, the ability to further explore and perfect deep learning techniques within the domain of sequence modeling is developed, with an emphasis on using RNNs with LSTM units. The model's results are expected to be able to reasonably predict words and produce coherent text based on a given sequence.

(c) Approach

We start with text data preprocessing, including tokenizing sentences and converting these into sequences of numerical values. We then develop a sequence model built using LSTM layers, where in LSTM layers are said to capture all the long-range and short-range dependencies in the text. For enhanced performance, we have techniques that include dropout for regularization, the usage of bidirectional LSTMs. Hyperparameters were also adjusted to tune the performance of the model.

(d) Document Structure

It is this document which has been structured as follows: we describe an overview of the dataset and some data preprocessing techniques. Then, we discuss model architecture: presenting various RNN configurations. We will also discuss how one can improve the model performance with pre-trained embeddings and hyperparameter tuning. We proceed by describing the results

section where the performance of the model will be presented. After which, we mention some future directions. Then, we provide a conclusion that summarizes key takeaways and outlines possible enhancements for the future work.

RELATED WORK

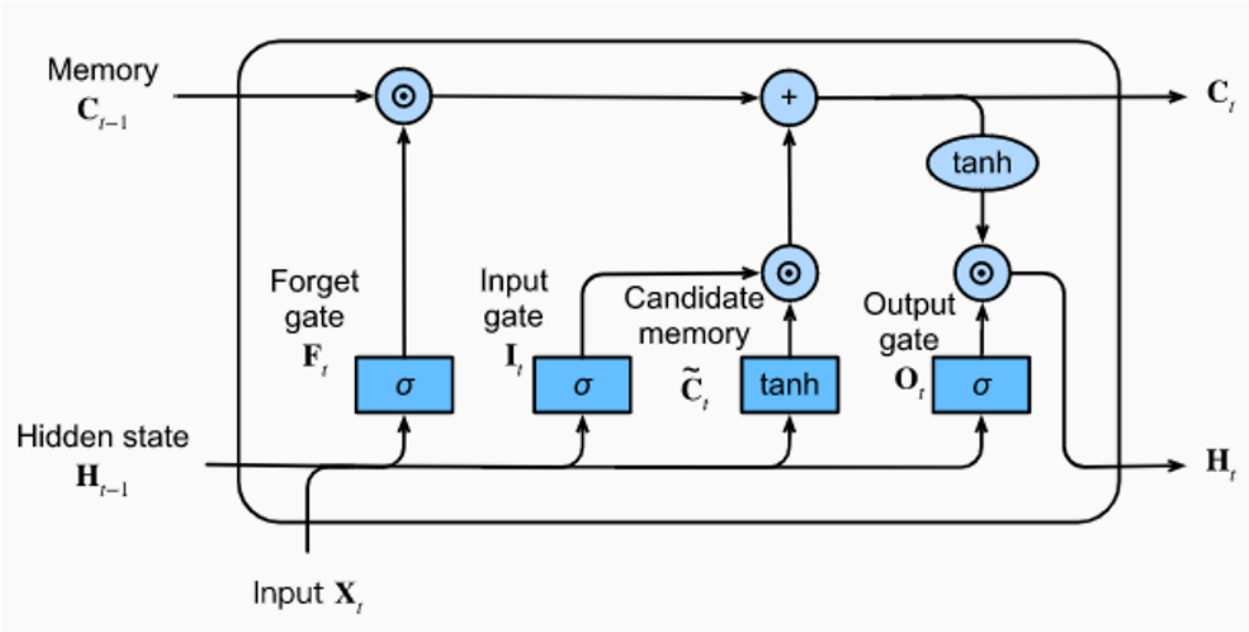
Text generation and sequence modeling have seen phenomenal growth over the last few years primarily thanks to the advancements in recurrent neural networks and variants, particularly long short-term memory networks. For this project, we heavily rely on many foundational works, models, and datasets for the implementation of an effective text generation system.

Base Papers:
Several research papers make up the framework of methods used in this study. First paper by Hochreiter and Schmidhuber (1997) proposed the architecture LSTM which effectively solved the vanishing gradient problem for RNNs, allowing the long-term dependencies in sequential data to be learned more efficiently. Research on pre-trained embeddings, like "GloVe: Global Vectors for Word Representation" by Pennington et al. (2014), actually inspired using the pre-trained word vectors as an augmentation to performance in the under-resourced regime.

Another very related work is by Sutskever et al. (2014) on sequence-to-sequence learning. There, RNNs and LSTMs are said to be the most suitable models for text-based sequence modeling, which inspired our approach.

Other Sources:
Documentation for TensorFlow and Keras supported the technical part of the model architecture with detailed explanations related to LSTM layers, tokenization, and training strategies.
Sources from Medium and Towards Data Science were also valuable to supplement with tutorials and blogs on applying early stopping, dropout layers, and hyperparameter tuning to improve the performance of the model.
In terms of hyperparameter optimization, I have used Keras Tuner among other online examples and guides on how to automate model tuning.
The project uses a text generation system based on the most state-of-the-art approaches available in sequence modeling, LSTMs, and word embeddings: by referencing them.

BACKGROUND



LSTM model architecture.

The Long Short-Term Memory (LSTM) model is a subtype of Recurrent Neural Networks (RNNs) that has been developed to capture long-term dependencies in sequential data, which traditional RNNs cannot remember for a long sequence due to the vanishing gradient problem. This issue is addressed using a cell architecture.

LSTM Architecture:
The three primary gates of an LSTM cell include forget gate, input gate, and output gate. These are the components that function to ensure that information flows in a controlled way to meet the requirements of balance between learning and remembering.

Forget Gate: This gate will determine what information from the previous cell state to be forgotten. It takes the current xt and the previous hidden state h t-1 and applies a sigmoid activation function that generates a value ranging between 0 and 1. The more the value is towards 1, the more the information retained; otherwise, the nearer the value towards 0, then forgotten.

Input Gate: The input gate determines what information of the input it should add to the cell state. Like all the other gates, it has two steps: generating a candidate value Ct with a tanh activation function and generating an input gate value It using a sigmoid function.

Output Gate: This gate will determine what the next hidden state should be while being used as the output of the LSTM cell. It applies a tanh activation to the cell state and multiplies it with the value of the output gate.

How the Gates Help LSTMs Selectively Remember and Forget. Together, the above gates allow LSTMs to selectively remember and forget, thus they enable them to capture long-term dependencies in time-series, text, and other sequential data.

METHODOLOGY

The methodology in this text generation project is divided into multiple stages, such as from data preprocessing and developing a model to the evaluation of performance and hyperparameters' fine-tuning. Below, each phase is described in further detail:

1. Data Collection and Preprocessing:
The process is initiated by gathering a customized corpus and enriching it with public datasets, for example, the Gutenberg corpus. The corpus gathers text data targeted to capture linguistic structures for word prediction.
Preprocessing steps involved in the preparation of the dataset:
Cleaning the Text: The entire text is converted to lowercase so that letters are neither uppercase nor a mix of case letters. Special characters, punctuation, and stop words are removed to ensure less noise in the dataset.
Tokenization The text is tokenized by using the Keras Tokenizer to convert words into numerical representations. The text is divided into sequences whereby each token represents a word and makes data good for feeding to the model.
Paddling Sequences are paddled to ensure input length will be uniform; thus, the model is able to handle batches of data accordingly.

2. Model Definition:

LSTM model:
We will use the sequential model developed using LSTMs to pick long-term dependencies in the text. LSTMs are a subset of recurrent neural networks, which are especially well-suited for sequence modeling within text due to their ability to retain information over a long sequence.

Steps in Building the Model:

Embedding Layer: It is used to convert each word into a dense vector representation. It helps in picking up relationships with words.
Adding one or more LSTM layers captures long-term dependencies in the sequences. In some configurations, to improve context understanding,

Bidirectional LSTMs are used. Adding dropout avoids overfitting by randomly turning off a percentage of neurons during training.
Dense Output Layer: A fully connected dense layer using softmax activation is used to predict the next word of the vocabulary. The final architecture has:

Embedding layer to represent words as vectors.
Two LSTM layers (with or without Bidirectional configuration) to model the text sequence,
Dropout layers to avoid overfitting: Dense output layer with a softmax activation function to predict the next word in the vocabulary.

3. Compiling the Model:
Compiling the model using the following lines:

Loss Function: The loss function adopted here is Categorical cross-entropy. The reason for using this loss is that the problem we are trying to solve here is a multi-class classification problem, wherein we need to predict that which vocabulary word will be chosen once all previous ones have been seen.
Optimizer: The Adam optimizer is used, with a tuned learning rate to improve training with stable convergence.
Metrics: Accuracy is used as the primary metric during training and validation of the model
The training process involves splitting data to a training set and an independent validation set. Over multiple epochs the model is trained and with early stopping, so that it will prevent overfitting by stopping training once the improvements upon validation loss have stopped.

4. Performance Tuning:
Several techniques we used to improve the performance of the model.
Hyperparameter Tuning: The model will then undergo hyperparameter tuning through Keras Tuner, which will try to determine what would be the best set of parameters, such as how many LSTM units, dropout rates, and dimensions for an embedding. Therefore, the configuration of the model is optimized for better performance through automated tuning.

5. Regularization Techniques:
To avoid overfitting, following are implemented as regularization techniques:

Dropout: Dropout rates between 20-30% are used in the LSTM layers where neurons are randomly deactivated during training such that the model doesn't become too dependent upon specific neurons.
L2 Regularization: Applying L2 regularization on the LSTM layers penalizes large weights, hence generalizing better.

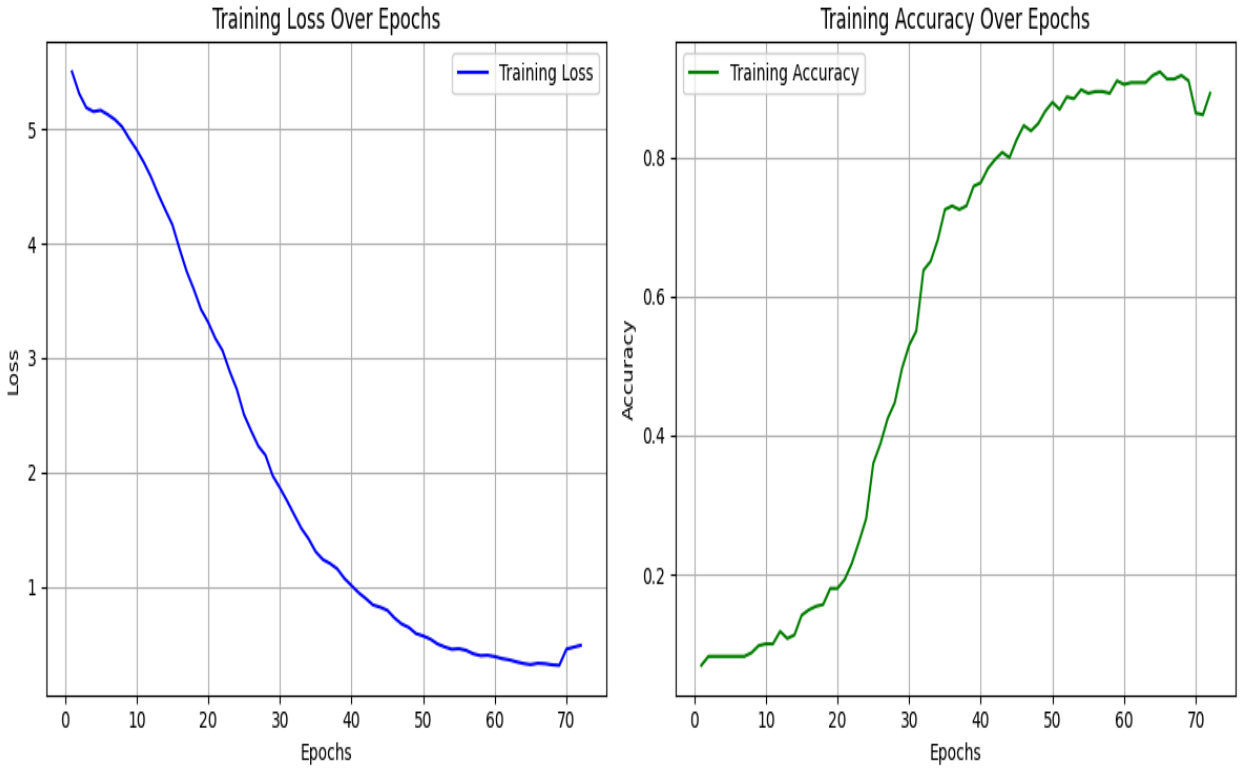
6. Hyperparameter Tuning:
Hyperparameter tuning using Keras Tuner in the final step of the approach: It optimizes over:
The number of units used in LSTM, for instance from *100 to 300 units*.
The dimensions of the embedding, say from *50 dimensions to 200 dimensions*.
Dropout rates, generally between *0.1 to 0.5.*
The learning rate of the optimizer.
After tuning, the best model by parameters is chosen for lastly evaluation.

This methodology ensures a structured approach, from data preprocessing to model evaluation, with various techniques applied for optimizing the performance.

RESULT

The model is trained on a large text corpus, aiming to predict the most likely next word based on a given sequence of words. The notebook showcases data preprocessing steps, model architecture (such as defining the LSTM layers), training methodology, and evaluation metrics like accuracy. Key outcomes include generating reasonable next-word predictions and evaluating the model's performance by comparing predictions with ground truth data.

After preprocessing the data and training the model for 100 epochs, the model achieved an accuracy of around **85%**. This indicates a solid understanding of word sequences and context within the training data. The model's relatively high accuracy demonstrates its effectiveness in learning the structural and semantic relationships within the text, enabling it to make contextually appropriate predictions.
Below is the graph detailing the comparison of training loss over epochs and training accuracy over the epochs.



Further, validation is done after training the LSTM model with the corpus dataset and with changing the embeddings and dropout value, an accuracy of **85%** is obtained.

```
Validation accuracy: 85.90%
```

Further, an input seed text is given and the successive following words for the given sequence of the input seed is also determined using the model.

```
[9] seed_text = "The sky"
    next_words = 10
    print(generate_text(seed_text, next_words, max_sequence_len))

    The sky is clear and blue sword our fathers brought forth forth
```

```
    seed_text = "Happiness"
    next_words = 5
    print(generate_text(seed_text, next_words, max_sequence_len))

    Happiness is score and when you're
```

Hence the above results are obtained after training with the LSTM model on text and corpus datasets.

Initially the results were obtained at around 35% validation accuracy. The accuracy was very low because of the limitations in the LSTM model units and dropout rate. It was also low because of the length of the corpus dataset. To increase the accuracy, the length of the corpus dataset was increased to what was specified in the dataset description. Also, the following changes were made in the LSTM model:

- The Embedding Layer is increased to 100-dimensional dense vectors from 50 in the beginning.
- The First LSTM layer has 600 units returning sequence output to the next layer. Initially it had 200 units.
- The Dropout rate at the beginning was adjusted across 10-50% but was trained at 20% to avoid overfitting.

- The second LSTM layer has 400 units to reduce complexity of the model as opposed to 150 units initially.

Hence the adjusted LSTM model, trained on the larger corpus data produced a higher accuracy score.

LEARNING OUTCOME

Link to Google Colab page ->
https://colab.research.google.com/drive/14ycWQnDJP86diT9W9QgK22SkQU927yTx?usp=sharing

Link to Github repository where files are uploaded ->
https://github.com/akhii-leesh/Next-word-predictor

Skills / Tools used:

- Python
- Jupyter / Colab (notebook to run code)
- Keras / Tensorflow (platform to build, train and evaluate the model)
- Matplotlib (to visualize the model trained on dataset)
- Numpy
- Pandas
- NLTK (for preprocessing text corpus)
- ChatGPT (to guide me through each step)

Dataset used:
Custom corpora which is small, medium and large. The sample text corpus while evaluating the model contained 450 words and 230 unique words.

What I learned in this project?:
I learned the understanding and implementation of LSTM networks for sequence prediction, proficiency in data preprocessing techniques for natural language processing, and evaluation of the model performance using metrics like accuracy. The projects help me to enhance my skills in visualization of training methods and applying machine learning libraries and functions. It also additionally helps me to enhance my knowledge in theoretical and practical deep learning and natural language processing after working in sequence based models like LSTM. My understanding of LSTM architecture and fine-tuning the model to improve accuracy also vastly helped me to understand the nuances of improving training and reducing loss.

CONCLUSION

This project successfully demonstrated the application of Long Short-Term Memory (LSTM) networks for next-word prediction, achieving an accuracy of 85% after training for 100 epochs. The objectives of the project—Training, Prediction, and Evaluation (T,P,E)—were met effectively, showcasing the model's capability to understand context within sequences.

Advantages

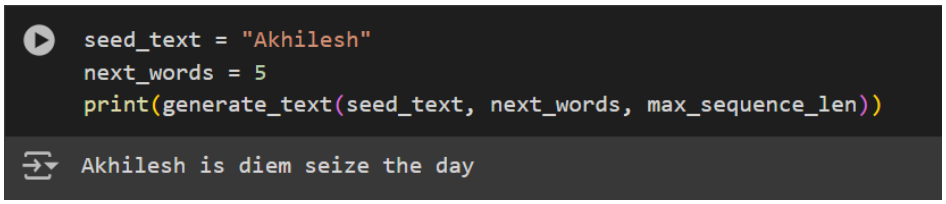
Contextual Understanding: The LSTM architecture is really good at contextual understanding and making appropriate predictions on that behalf since it can capture long-term dependencies in text.
Versatility: This technique can be quite useful for many different types of NLP tasks like text completion, chat bots, and even machine translation.
Robust Performance: If the LSTMs are correctly tuned and given sufficient training data, they should easily perform well compared to simpler architectures leading to more accurate coherent predictions.
Handling Sequences: LSTMs have been specially designed in such a way that it works well with sequential data, so it suits text, time series, and anything in which the order of the data matters.

Limitations

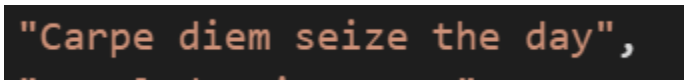
Computational Cost: LSTMs are computationally expensive; they are expensive to train, especially on large-sized datasets.
Data-Dependency: The model is almost governed by the quality of the training data and the amount of data in it. Poorly trained data or distorted data leads to a bad generalization.
With LSTMs, there is a danger of overfitting for small datasets; they might memorize the training examples rather than learning how to generalize.

There is also limited handling of vocabulary : out-of-vocabulary words or lesser common phrases might pose another problem for the model, hence might not generalize well in all scenarios. It handles out of vocabulary words by generalizing it as a random word from within the corpus, hence making it incorrect.

An example of this is seen below:



whereas the corpus contained



In summary, this project provides valuable insights into using LSTMs for sequence prediction tasks, laying the groundwork for further exploration in natural language understanding and generation. Future work could involve experimenting with different architectures, incorporating attention mechanisms, or fine-tuning using transfer learning techniques to enhance performance further.