Next Word Prediction using LSTM

– By Blessy Chinthapalli

Table of Contents

Executive Summary

We have created a next word prediction model using the LSTM network. The model was trained on the first 1000 reviews of the IMDB dataset and tested on the next 100 reviews. The choice of dataset was IMDB reviews because it represents natural language used in modern-day writing. Our RNN architecture consists of an embedding layer, two LSTM layers, and two dense layers. It uses X,Y pairs of words where X is the previous word and Y is the next word. The output layer employs the softmax function to give probabilities for all the words in the vocabulary and the word with the highest probability is selected.

However, the current model has its limitations. It is not capable of predicting words that are outside the training vocabulary and does not understand the context of the sentence, only predicting based on the last word. The model's accuracy is 19.43% on the training set and 12.83% on the test set, indicating the need for more training data.

Through the practical application of building the next word prediction model, we were able to get a deeper understanding of the Neural Network Architecture and the various hyperparameter tuning required.

Introduction

We are trying to build a next word prediction model using the Long Short-Term Memory (LSTM) network. LSTM is a type of Recurrent Neural Network (RNN) that is designed to handle long term dependencies in the input dataset. It is well-suited for natural language processing tasks because it can handle sequences of varying lengths and remember important information over long periods of time. The model is trained on the first 1000 reviews of the IMDB dataset and predicts the next word for any given sentence by saving the last word of the sentence and predicting the most likely next word based on the probability. If the word is not present in the vocabulary on which the model is trained, then the model gives the output ‘No prediction’.

Dataset

We deliberately selected the IMDB reviews dataset for training our language model because we wanted it to learn from how people naturally write. While there are many open free ebooks available on Project Gutenberg, they mostly consist of older books, and people tend to write differently with vocabulary usage changing over time. Therefore, the IMDB reviews dataset, which contains text written by people, was more suitable for our purposes.

However, due to the computational resource constraints in Google Collaboratory, we had to limit the dataset to the first 1000 reviews. One limitation of the dataset is that predictions can be often limited to the context of movies.

Data Preprocessing

To preprocess the data we followed the following steps:

1. The first step is to remove HTML tags, punctuations, periods, digits and convert to lowercase. We did not remove the stop words like ‘a’, ‘an’, ‘the’ as then the model will not give any predictions for these inputs.
2. Tokenization: The pre-processed text is then tokenized using the Keras tokenizer. This involves converting each word in the text into a unique integer. This allows the text data to be processed and analyzed by the model.
3. Sequencing: The tokenized text is then converted into sequences of words. Each sequence consists of two words, where the first word is the input and the second word is the output. For example, for the input "The movie", ‘the’ would be X and ‘movie’ would be Y.

Neural Network Architecture

The RNN used consists of an embedding layer, two LSTM layers, and two dense layers. The embedding layer converts the input sequence of integers into a dense vector representation. The two LSTM layers are responsible for understanding the context of the input sequence and predicting the next word. The dense layers are used to produce the final output, which is a probability distribution of the next word.

- **Embedding Layer:** The embedding layer is the first layer in the network, and it is responsible for converting the input data into a dense vector representation. The embedding layer has three parameters: the size of the vocabulary, the dimensionality of the embedding, and the length of the input sequences. In this project, the vocabulary size is the total number of unique words in the input data, the dimensionality of the embedding is set to 10, and the length of the input sequences is set to 1.
    1. Dimensionality (10) - The dimensionality of the embedding is a hyperparameter that needs to be tuned during the model design phase. It represents the size of

the dense vector representation that the embedding layer produces for each word in the vocabulary. The main objective of the embedding layer is to map each word in the vocabulary to a dense vector representation that captures the semantic and syntactic similarity between words. In this project, the dimensionality of the embedding layer is set to 10. This means that each word in the vocabulary is represented by a dense vector of 10 values. A lower value for the embedding dimensionality, such as 5 or 8, may not capture enough information about the word's semantics, resulting in poor model performance. On the other hand, a higher value, such as 50 or 100, may capture too much information, leading to overfitting and computational complexity. Therefore, the choice of 10 for the embedding dimensionality strikes a balance between model performance and computational efficiency. Moreover, a lower dimensionality also means fewer parameters in the embedding layer, which reduces the computational complexity of the model.

2. Input_Length (1) - The length of the input sequence determines the number of words that the model will consider as context to predict the next word. In the current project, the length of the input sequences is set to 1, which means that the model will consider only one word as context to predict the next word. For example, suppose the input sentence is "The cat sat on the", and the length of the input sequence is 1. In that case, the model will consider "the" as context to predict the next word, which could be "mat," "hat," "rat," or any other word that frequently occurs after "the" in the input data. If the length of the input sequence is increased to, say, 2 or 3, the model will consider more words as context to predict the next word, which could lead to more accurate predictions. However, increasing the length of the input sequence also increases the number of input parameters, making the model more computationally expensive to train.

- **LSTM Layers:** The LSTM layers are the core components of the network, and they are responsible for capturing long-term dependencies in the input data. The LSTM layers have two parameters: the number of LSTM units and the return_sequences parameter. In this project, the number of LSTM units is set to 1000, and the return_sequences parameter is set to True for the first LSTM layer and False for the second LSTM layer.
    1. Number of LSTM units (1000): The number of LSTM units determines the complexity of the LSTM layer. A larger number of LSTM units allows the model to learn more complex patterns in the input data but can also lead to overfitting. In this project, the number of LSTM units is set to 1000, which is a relatively high value. This high number of units enables the LSTM layer to capture more complex and long-term dependencies in the input data, which is beneficial in generating coherent and meaningful text.
    2. Return sequences parameter: The return_sequences parameter is a boolean value that determines whether the LSTM layer should return the entire sequence of outputs or just the last output. When set to True, the LSTM layer returns the entire sequence of outputs, which is useful when stacking multiple LSTM layers. In this project, the return_sequences parameter is set to True for the first LSTM layer and False for the second LSTM layer. This configuration allows the first LSTM layer to return the entire sequence of outputs, which is then fed as input to the second LSTM layer. The second LSTM layer then only returns the last output, which is used as input for the dense layer.
- **Dense Layers:** The dense layers in the neural network are responsible for taking the output from the LSTM layers and performing the final classification. The first dense layer reduces the dimensionality of the output from the LSTM layers, while the second dense layer maps the output to the size of the vocabulary and produces a probability distribution over the vocabulary. The dense layers have two parameters: the number of

units and the activation function. In this project, the number of units in the first dense layer is set to 1000, and the activation function is set to relu. The number of units in the second dense layer is set to the size of the vocabulary, and the activation function is set to softmax.

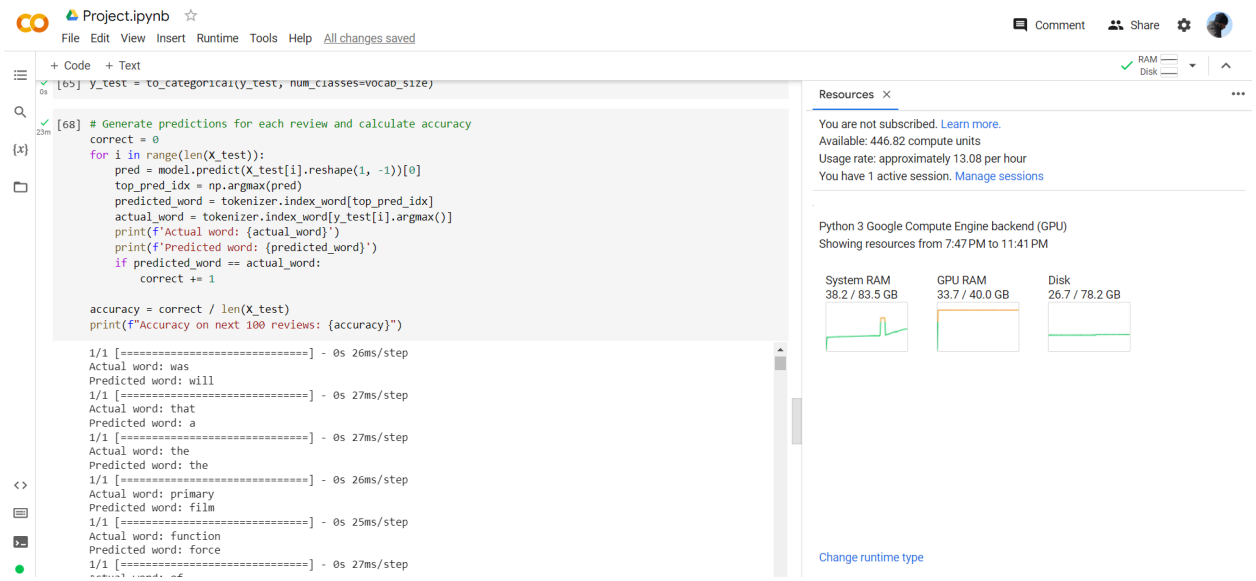
1. Rectified Linear Unit (ReLU) Activation Function in Layer 1: ReLU has several advantages, such as being computationally efficient and reducing the likelihood of the vanishing gradient problem during backpropagation. The ReLU activation function is also known for its ability to learn and represent complex non-linear relationships between the input and output data. The ReLU activation function is used in the first dense layer to introduce non-linearity to the output of the LSTM layers, which can capture complex relationships between the input data and the target variable.
2. Softmax Activation Function in Layer 2: The number of units in the second dense layer is set to the size of the vocabulary, which is necessary for mapping the output to a probability distribution over the vocabulary. The activation function used in the second dense layer is softmax, which produces a probability distribution over the vocabulary.

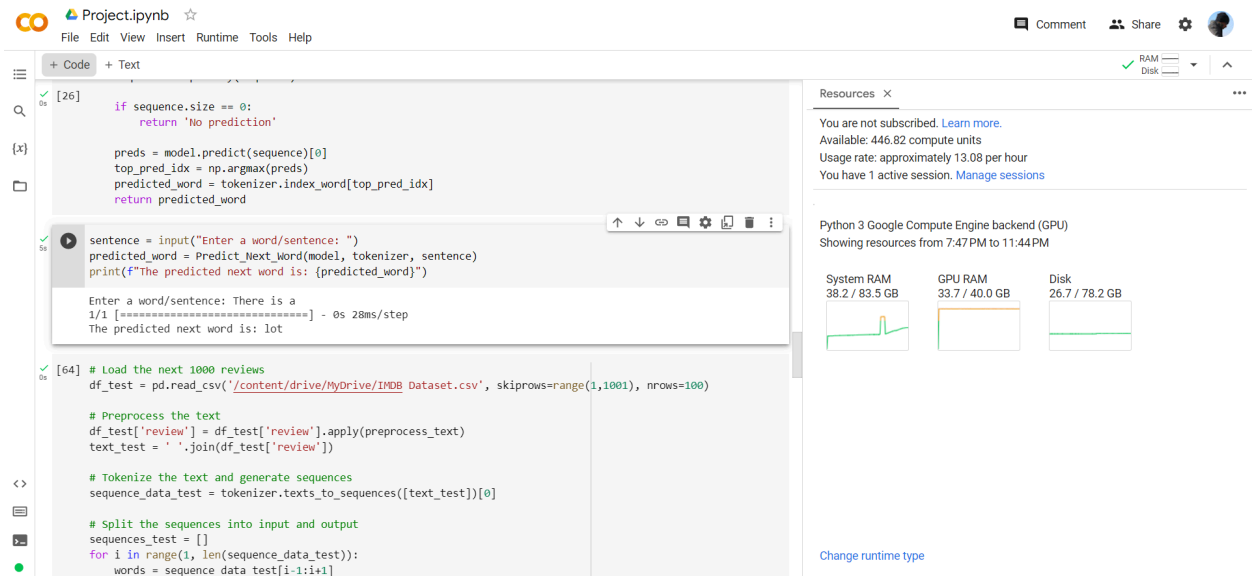
- **Loss Function and Optimizer:** The loss function used in this project is categorical cross-entropy, which is commonly used for multi-class classification problems. The optimizer used is Adam, which is a popular stochastic gradient descent (SGD) optimizer that is well-suited for deep learning applications.
    1. Adam Optimizer: Adam stands for Adaptive Moment Estimation, and it is an extension of the stochastic gradient descent (SGD) optimizer. Adam calculates adaptive learning rates for each parameter by estimating the first and second moments of the gradients. This enables Adam to achieve faster convergence and better performance compared to other optimization algorithms.

- **Model Checkpoint:** The ModelCheckpoint callback is used to save the weights of the network after each epoch if the loss has improved. This ensures that we always have access to the best performing weights of the network.
- **Reduce Learning Rate on Plateau:** The ReduceLROnPlateau callback is used to reduce the learning rate of the optimizer if the loss has stopped improving after a certain number of epochs. This helps the network converge faster and prevents it from getting stuck in local minima.
- **Tensorboard Visualization:** The TensorBoard callback is used to visualize the training process of the network. It allows us to monitor the loss and accuracy of the network over time and to visualize the architecture of the network.

Sample Predictions

Following are sum of the sample predictions of the model:





Accuracy on Test Set

The accuracy of the model on the train set is 19.43%. The accuracy is calculated for every input word, we are checking the index of predicted next word and the index of the actual word.

For checking the accuracy on the test set, we followed the pre-processing steps above on the next 100 reviews and checked the predictions with the actual word. The accuracy obtained was 12.28%.

While for common words, the model is able to predict the words pretty accurately, training the model on a bigger dataset would help in increasing the accuracy.

Limitations

The model has following limitations:

1. The model will not be able to predict for words which are not in the vocabulary on which the Neural Network was trained.

2. The model currently works by checking the last word of the input sentence to predict the next word and does not understand the context of the sentence. For example for the input ‘There is a’ the prediction would be ‘lot’ and just for the input ‘A’ the prediction would still be ‘lot’.
3. For the sentences ending with periods like full stops and commas, the model will not be able to give any predictions as while training the model we had removed them.

Conclusion

Through the practical application of predicting the next word, the project helped us get a better understanding of RNN, the various hyperparameters required for training a model to predict the next word. While the accuracy of the model is low, it can be improved significantly with an increase in computation power to train the model on a bigger corpus of data.

When changing the input from 1 word to a set of 3 words, we observed that most of the predictions were ‘no predictions’. Future work can include building a model that understands the context of the sentence to make a prediction.

Exhibit

We have shared the code for training and testing the model in Project.py, tokenizer_1000.pkl file, model_1000.h5 file and To_apply.py file. Please ensure that tensorflow, keras and pickle libraries are installed on the system before running the following.

To use the model, please use the To_apply.py file. Please follow the following steps:

1. Change the working directory to the desired directory.

2. On the directory, save the tokenizer_1000.pkl file and model_1000.h5 file.

3. Enter a sentence where the code requests to input the sentence and the model should be able to make the predictions.