Introduction:

Sentiment analysis or text classification on IMDB Movie Reviews Dataset [1] using text classification model Vanilla RNN [2] and LSTM [3] models. The task of the model is, given a review it needs to predict whether the review is positive or negative. For this, we converted the reviews into a sequence of tokens and added pre-trained word embedding vectors trained using Glove [4] to the embedding layer of the sequential model which are further fed into the RNN layer and LSTM layer. For the LSTM layer, we used LSTM with a Bidirectional layer [5]. Various hyperparameters were considered during training the model which includes state dimension, Learning rate, Dropouts. The categorization can help to predict the sentiment or the polarity of the review.

IMDB Movie Reviews Dataset is a binary sentiment classification dataset used to determine the polarity of the review. It has 25,000 training movie reviews and 25,000 testing movie reviews. I have used the IMDB data from the TensorFlow datasets [6]. This data is preprocessed and fed into the model using RNN and LSTM layers to determine the polarity of the review. Since it is a binary classification model, the loss metric that would be more suitable is 'binary_crossentropy' and the performance metric used to evaluate the model is Accuracy. EarlyStopping [7] was used in this process to avoid overfitting of the model and the hyperparameters used and the results of each model are represented in the prettytable [8].

Objective:

The main motto of the assignment is to develop text classification models using Vanilla RNN and LSTM for the IMDB dataset and find the best classification result of each model.

Exploratory Data Analysis:

The IMDB dataset is downloaded from TensorFlow datasets.

TensorFlow Dataset:

IMDB dataset is a standard dataset for sentiment analysis where we have positive and negative labels, so, it is a binary classification problem.

Training Samples: 25,000

Testing Samples: 25,000

The text samples cannot be fed into the models directly. They should be pre-processed to train them. The text vectors are tokenized using a tokenizer where each word is assigned with a token value and in my assignment, I used a vocabulary size of 10,000 so that instead of tokenizing all the words, only the top 10,000 words present across the corpus of the data are tokenized. This tokenizer is then assigned to the sequence of words present in the sentences, the purpose of doing this is to consider the meaning present in the sequence of words.

Next, padding is performed on all these tokenized sequences. The reason for padding is to make sure all the tokenized text sequences are of the same length. In my case, I have assigned a maximum length of 200 and truncating the words after that. By performing this, we can use the points (text reviews) as a batch to train the model, this is one of the main reasons to performs padding on the text sequences.

Model and Performance:

We train the data on two RNN models with various hyper-parameters.

1. Vanilla RNN
2. LSTM

For each of these models, I have used Glove Embedding for weights in the Embedding layer. Among the models, I have used "glove.6B.zip" which is the smallest package of embeddings and it is trained on one billion words with a vocabulary of 400,000 words. There are four embedding vector sizes (50, 100, 200, 300 dimensions) available. In our experiment, I have used a 200-dimensional version of the embedding.

For both models EarlyStopping was introduced to avoid overfitting, to make sure we get the highest accuracy, a patience value of 5 was used which makes the model run 5 more epoch to attain the right value.

I executed the vanilla RNN and LSTM based model using the below Hyperparameters:

- Dimensions = [20,50,100,200,500]
- Dropouts = [0,0.5]

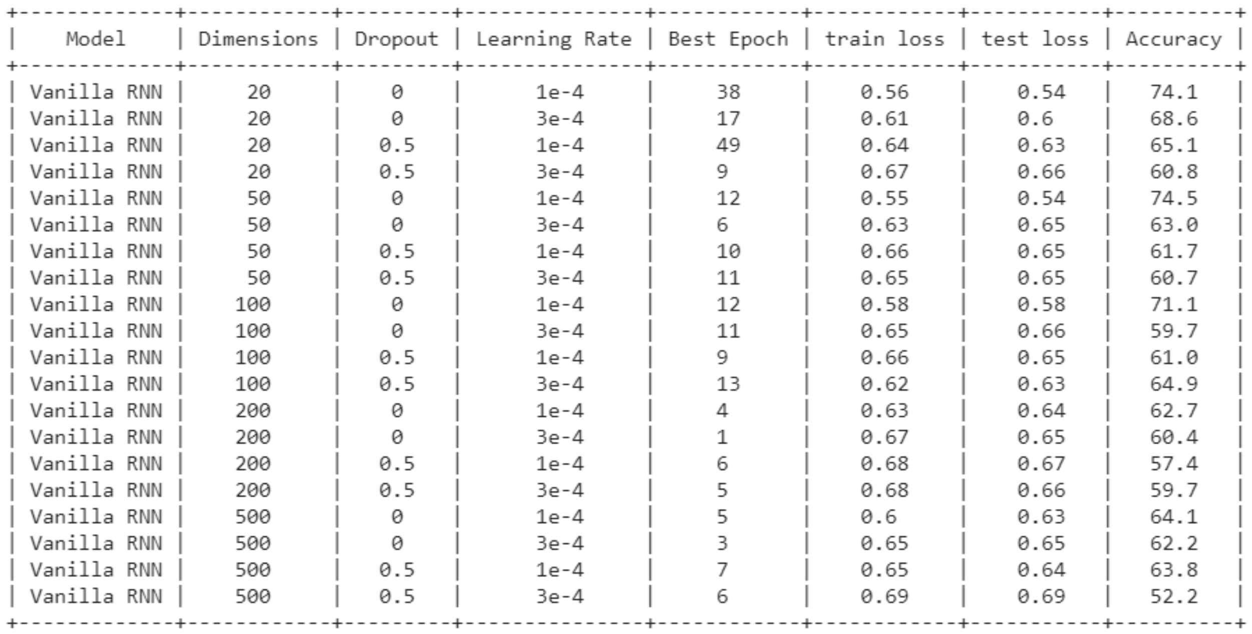
- Learning rates = [1e-4,3e-4]

The compilation parameters used in the experiment:

- loss function: ‘binary_crossentropy’
- Optimizer: Adam
- Performance metric: Accuracy

Vanilla RNN:

In this model, I have built a sequential model with an Embedding layer with pre-trained weights and fed it to an RNN layer. Multiple models were built using various combinations of hyperparameters and the results are depicted in table 1.



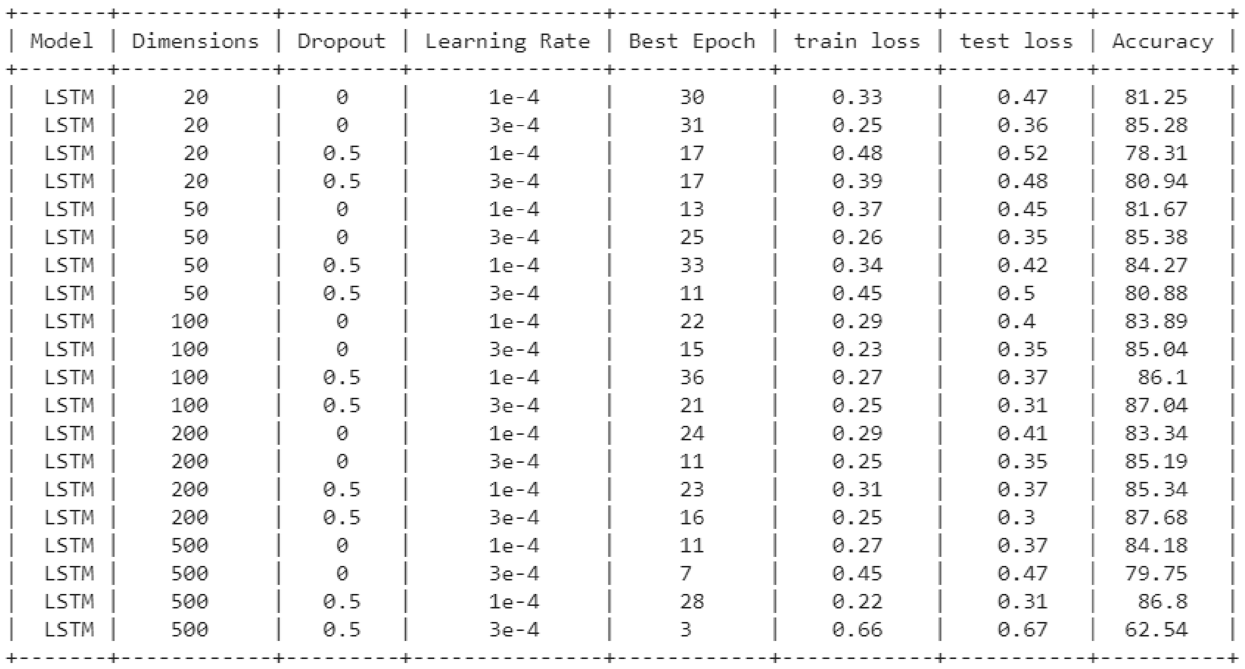
```
+-------------+------------+---------+---------------+------------+------------+-----------+----------+
|    Model    | Dimensions | Dropout | Learning Rate | Best Epoch | train loss | test loss | Accuracy |
+-------------+------------+---------+---------------+------------+------------+-----------+----------+
| Vanilla RNN |     20     |    0    |      1e-4     |     38     |    0.56    |    0.54   |   74.1   |
| Vanilla RNN |     20     |    0    |      3e-4     |     17     |    0.61    |    0.6    |   68.6   |
| Vanilla RNN |     20     |   0.5   |      1e-4     |     49     |    0.64    |    0.63   |   65.1   |
| Vanilla RNN |     20     |   0.5   |      3e-4     |     9      |    0.67    |    0.66   |   60.8   |
| Vanilla RNN |     50     |    0    |      1e-4     |     12     |    0.55    |    0.54   |   74.5   |
| Vanilla RNN |     50     |    0    |      3e-4     |     6      |    0.63    |    0.65   |   63.0   |
| Vanilla RNN |     50     |   0.5   |      1e-4     |     10     |    0.66    |    0.65   |   61.7   |
| Vanilla RNN |     50     |   0.5   |      3e-4     |     11     |    0.65    |    0.65   |   60.7   |
| Vanilla RNN |    100     |    0    |      1e-4     |     12     |    0.58    |    0.58   |   71.1   |
| Vanilla RNN |    100     |    0    |      3e-4     |     11     |    0.65    |    0.66   |   59.7   |
| Vanilla RNN |    100     |   0.5   |      1e-4     |     9      |    0.66    |    0.65   |   61.0   |
| Vanilla RNN |    100     |   0.5   |      3e-4     |     13     |    0.62    |    0.63   |   64.9   |
| Vanilla RNN |    200     |    0    |      1e-4     |     4      |    0.63    |    0.64   |   62.7   |
| Vanilla RNN |    200     |    0    |      3e-4     |     1      |    0.67    |    0.65   |   60.4   |
| Vanilla RNN |    200     |   0.5   |      1e-4     |     6      |    0.68    |    0.67   |   57.4   |
| Vanilla RNN |    200     |   0.5   |      3e-4     |     5      |    0.68    |    0.66   |   59.7   |
| Vanilla RNN |    500     |    0    |      1e-4     |     5      |    0.6     |    0.63   |   64.1   |
| Vanilla RNN |    500     |    0    |      3e-4     |     3      |    0.65    |    0.65   |   62.2   |
| Vanilla RNN |    500     |   0.5   |      1e-4     |     7      |    0.65    |    0.64   |   63.8   |
| Vanilla RNN |    500     |   0.5   |      3e-4     |     6      |    0.69    |    0.69   |   52.2   |
+-------------+------------+---------+---------------+------------+------------+-----------+----------+
```

Table 1: Vanilla RNN with various combinations of state dimensions, Dropout, Learning Rate, Best Epoch to train the model, Loss and Accuracy.

In our experiment with Vanilla RNN, the highest validation accuracy of 74.5% was obtained with the model trained with the parameters 50(State Dimensions), 0 (Dropout), 1e-4 (Learning rate) and this value is attained in 12 epochs.

LSTM Model:

In this model, multiple Bidirectional layers with LSTM were used as they can perform forward and backward detection on the sequences. The return_sequences parameter of the LSTM must be set to true when it is input to another bi-directional layer as it matches with the input of the next LSTM.



```
+-------+-----------+--------+-------------+----------+----------+---------+---------+
| Model | Dimensions| Dropout| Learning Rate| Best Epoch| train loss| test loss| Accuracy |
+-------+-----------+--------+-------------+----------+----------+---------+---------+
| LSTM |    20     |   0    |    1e-4     |    30    |   0.33   |   0.47  |  81.25  |
| LSTM |    20     |   0    |    3e-4     |    31    |   0.25   |   0.36  |  85.28  |
| LSTM |    20     |  0.5   |    1e-4     |    17    |   0.48   |   0.52  |  78.31  |
| LSTM |    20     |  0.5   |    3e-4     |    17    |   0.39   |   0.48  |  80.94  |
| LSTM |    50     |   0    |    1e-4     |    13    |   0.37   |   0.45  |  81.67  |
| LSTM |    50     |   0    |    3e-4     |    25    |   0.26   |   0.35  |  85.38  |
| LSTM |    50     |  0.5   |    1e-4     |    33    |   0.34   |   0.42  |  84.27  |
| LSTM |    50     |  0.5   |    3e-4     |    11    |   0.45   |   0.5   |  80.88  |
| LSTM |    100    |   0    |    1e-4     |    22    |   0.29   |   0.4   |  83.89  |
| LSTM |    100    |   0    |    3e-4     |    15    |   0.23   |   0.35  |  85.04  |
| LSTM |    100    |  0.5   |    1e-4     |    36    |   0.27   |   0.37  |   86.1  |
| LSTM |    100    |  0.5   |    3e-4     |    21    |   0.25   |   0.31  |  87.04  |
| LSTM |    200    |   0    |    1e-4     |    24    |   0.29   |   0.41  |  83.34  |
| LSTM |    200    |   0    |    3e-4     |    11    |   0.25   |   0.35  |  85.19  |
| LSTM |    200    |  0.5   |    1e-4     |    23    |   0.31   |   0.37  |  85.34  |
| LSTM |    200    |  0.5   |    3e-4     |    16    |   0.25   |   0.3   |  87.68  |
| LSTM |    500    |   0    |    1e-4     |    11    |   0.27   |   0.37  |  84.18  |
| LSTM |    500    |   0    |    3e-4     |    7     |   0.45   |   0.47  |  79.75  |
| LSTM |    500    |  0.5   |    1e-4     |    28    |   0.22   |   0.31  |   86.8  |
| LSTM |    500    |  0.5   |    3e-4     |    3     |   0.66   |   0.67  |  62.54  |
+-------+-----------+--------+-------------+----------+----------+---------+---------+
```

Table 2: LSTM (Bi-directional) with various combinations of state dimensions, Dropout, Learning Rate, Best Epoch to train the model, Loss and Accuracy.

In our experiment with Bi-directional LSTM, the highest validation accuracy of 87.68% was obtained with the model trained with the parameters 200 (State Dimensions), 0.5 (Dropout), 3e-4 (Learning rate) and this value is attained in 16 epochs.

Conclusions:

Presented two different models using Vanilla RNN and LSTM on IMDB Movie Reviews Dataset, we measured Accuracy as the performance metric and used binary_loss_entropy as a loss measure, Adam as the optimizer. From table 1 and table 2 we can conclude that the LSTM model showed exceptional results compared to the RNN model where for the LSTM model the accuracy is more than 80% in most of the cases and RNN model the accuracy remained in the range [60-70%] in most of the cases.

In the cases of RNN, the model with state dimensions 20 and 50 yields good results. Whereas in the case of LSTM, we can observe good results across all the dimensions.

In the Vanilla RNN based model, as the number of dimensions increase, the accuracy value keeps decreasing. The models with 1e-4 learning rate simulated better results than 3e-4.

In the LSTM based model, as we increase the number of state dimensions the accuracy tends to increase. Also, the model with a 3e-4 learning rate yielded better results than 1e-4.

In the Vanilla RNN based model when the dropout is changed from 0 to 0.5, we can see a decrease in the accuracy. However, there's no impact of dropout on the LSTM based model.

References:

[1] Large Movie Review Dataset - https://ai.stanford.edu/~amaas/data/sentiment/

[2] SimpleRNN layer - https://keras.io/api/layers/recurrent_layers/simple_rnn/

[3] LSTM layer - https://keras.io/api/layers/recurrent_layers/lstm/

[4] Glove: Global Vectors for Word Representation - https://nlp.stanford.edu/projects/glove/

[5] Bidirectional Layer - https://keras.io/api/layers/recurrent_layers/bidirectional/

[6] TensorFlow Datasets - https://www.tensorflow.org/datasets/catalog/overview

[7] tf.keras.callbacks.EarlyStopping - https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[8] Prettytable - https://pypi.org/project/prettytable/