ASSIGNMENT-04

REPORT

Vineeth Rudraksha

INTRODUCTION: - The purpose of this assignment is to apply RNNs, or Transformers, to text and sequence data.

This assignment aligns with the following module outcomes:

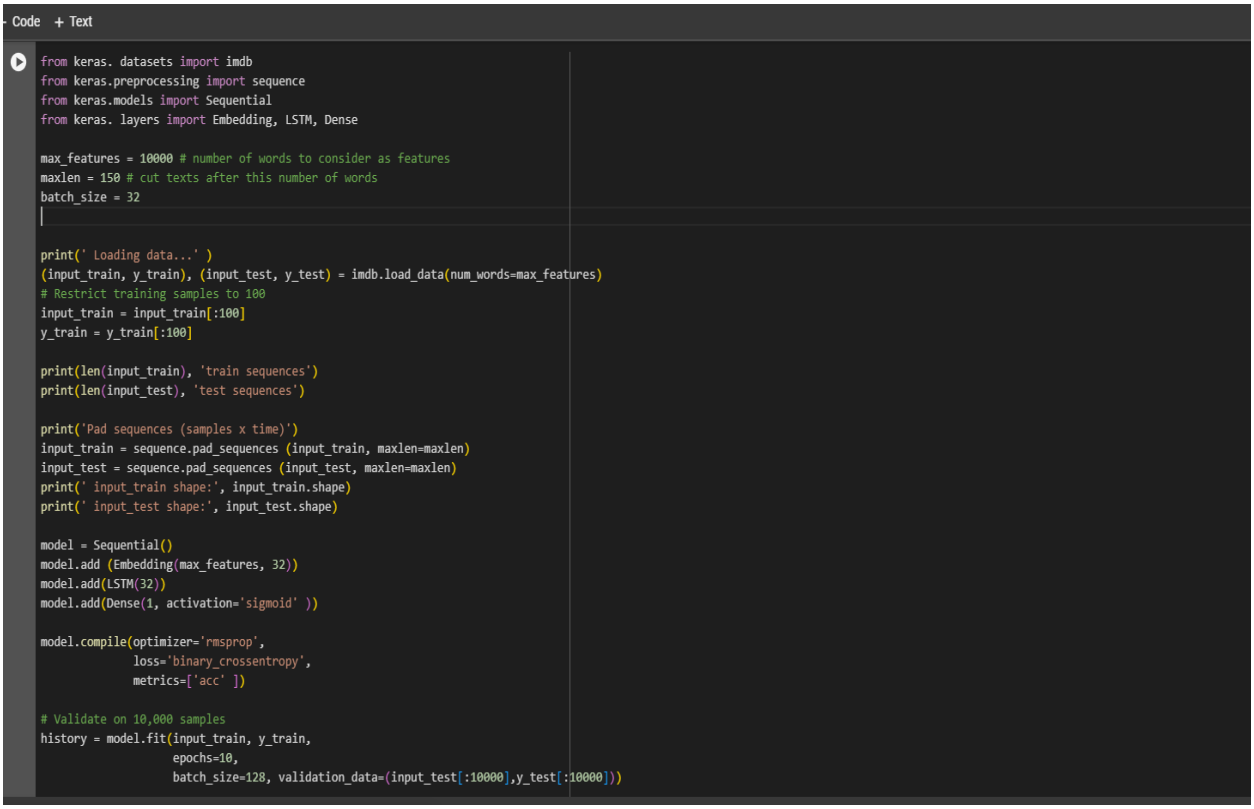
- Applying RNNs or Transformers to text and sequence data.
- Explaining the Transformer Architecture, especially the use of the Attention Mechanism.
- Explaining differences between RNNs and the Transformer Architecture.
- Demonstrating to improve the performance of the network, especially when dealing with limited data.

DATASET: - IMDB dataset link -https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

As per the given condition in the question, I have implemented the following: -

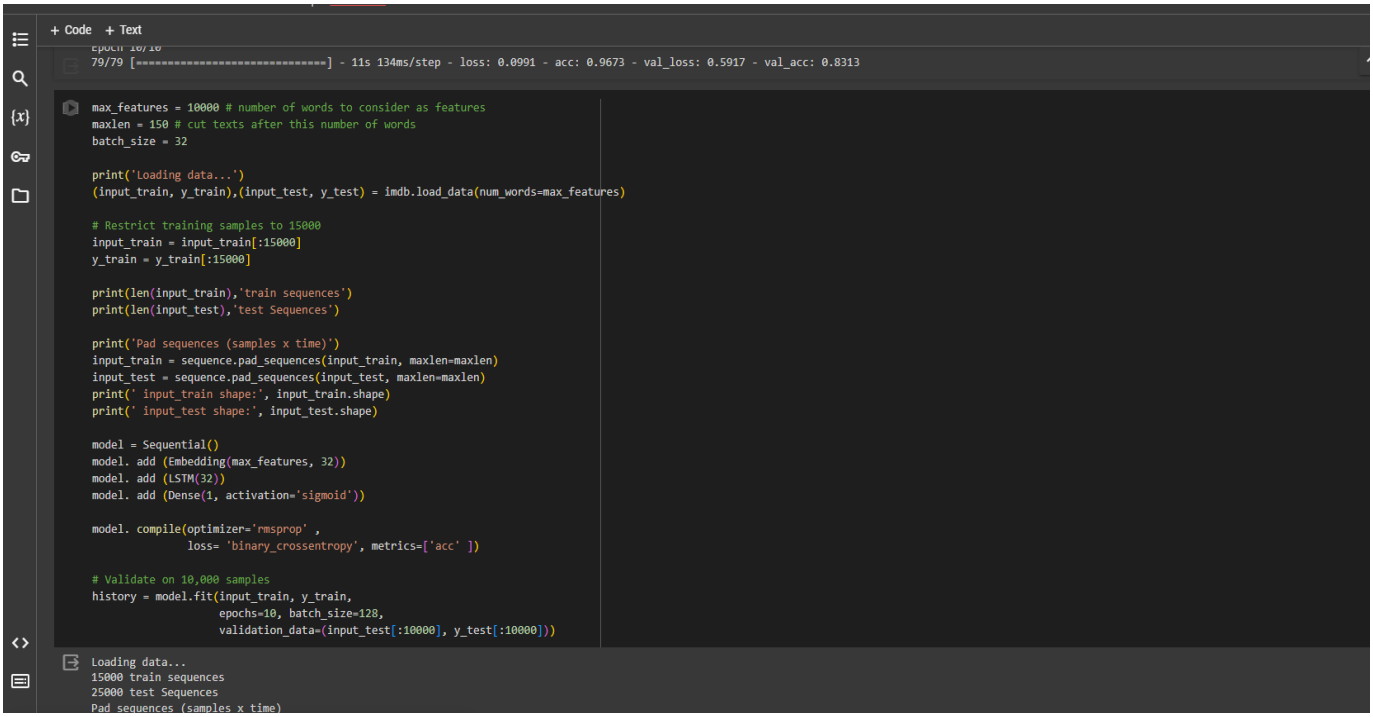
Consider the IMDB example from Chapter 6. Re-run the example modifying the following:

1. Cutoff reviews after 150 words.
2. Restrict training samples to 100.
3. Validate 10,000 samples.
4. Consider only the top 10,000 words.
5. Consider both an embedding layer and a pre-trained word embedding. Which approach did better? Now try changing the number of training samples to determine at what point the embedding layer gives better performance.



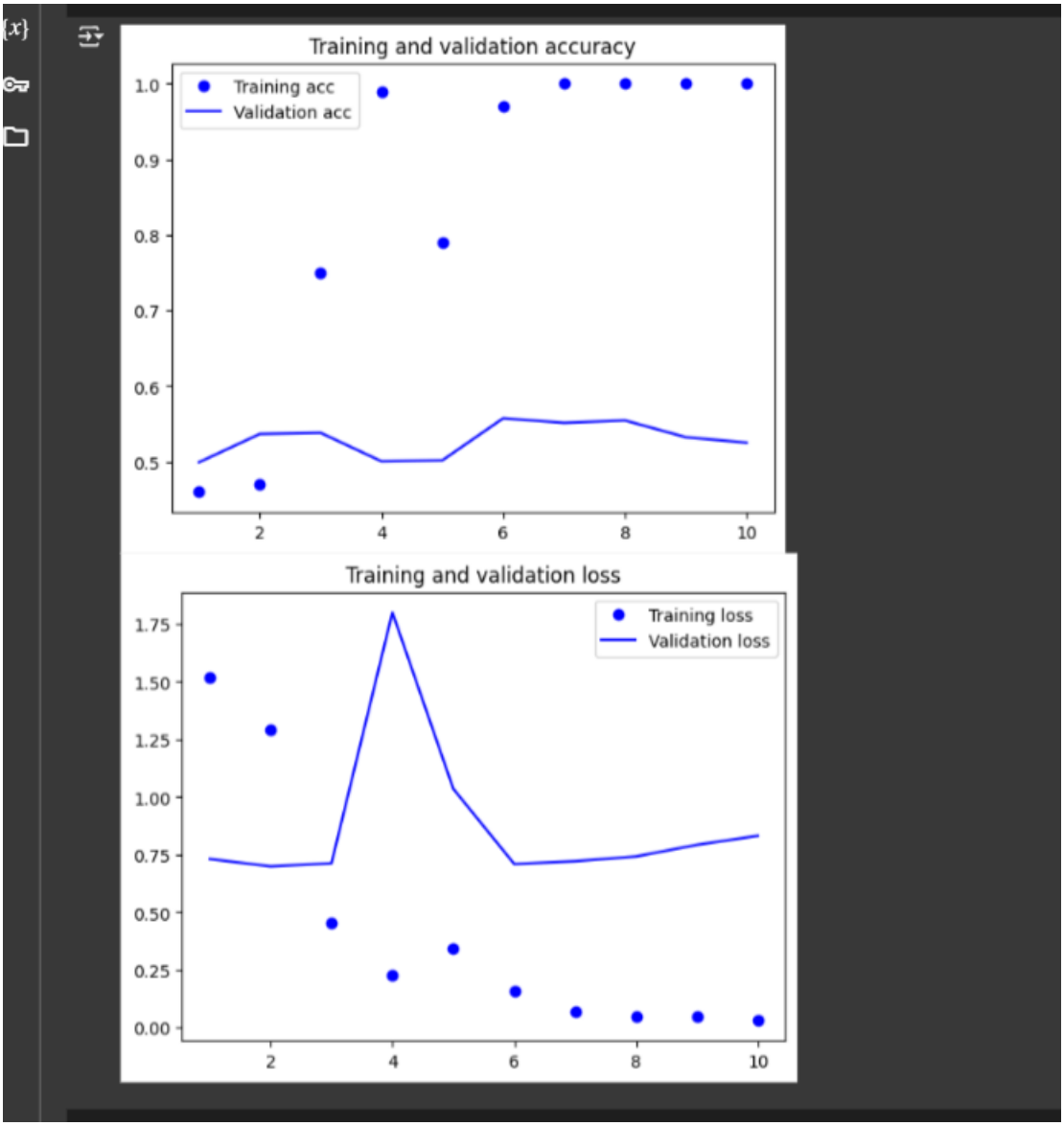
```
+ Code + Text

from keras. datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras. layers import Embedding, LSTM, Dense

max_features = 10000 # number of words to consider as features
maxlen = 150 # cut texts after this number of words
batch_size = 32

print(' Loading data...' )
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
# Restrict training samples to 100
input_train = input_train[:100]
y_train = y_train[:100]

print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences (input_train, maxlen=maxlen)
input_test = sequence.pad_sequences (input_test, maxlen=maxlen)
print(' input_train shape:', input_train.shape)
print(' input_test shape:', input_test.shape)

model = Sequential()
model.add (Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid' ))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc' ])

# Validate on 10,000 samples
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128, validation_data=(input_test[:10000],y_test[:10000]))
```

Now, A recurrent neural network (RNN) architecture, specifically a Long Short-Term Memory (LSTM) network, is used to categorize IMDb movie ratings as positive or negative. The dataset is loaded and preprocessed, with a vocabulary size of 10,000 words and each review is trimmed or padded to a maximum length of 150 words. A sequential model is built, starting with an embedding layer that converts integer-encoded words into dense vectors, and ending with a 32-unit LSTM layer for sequence processing. The model is then built using the RMSprop optimizer and binary cross-entropy loss function. Training is done on a subset of 10,000 training samples across 10 epochs with a batch size of 128, while validation is done on 10,000 test data. The network's performance indicators, throughout the training process, accuracy is monitored.

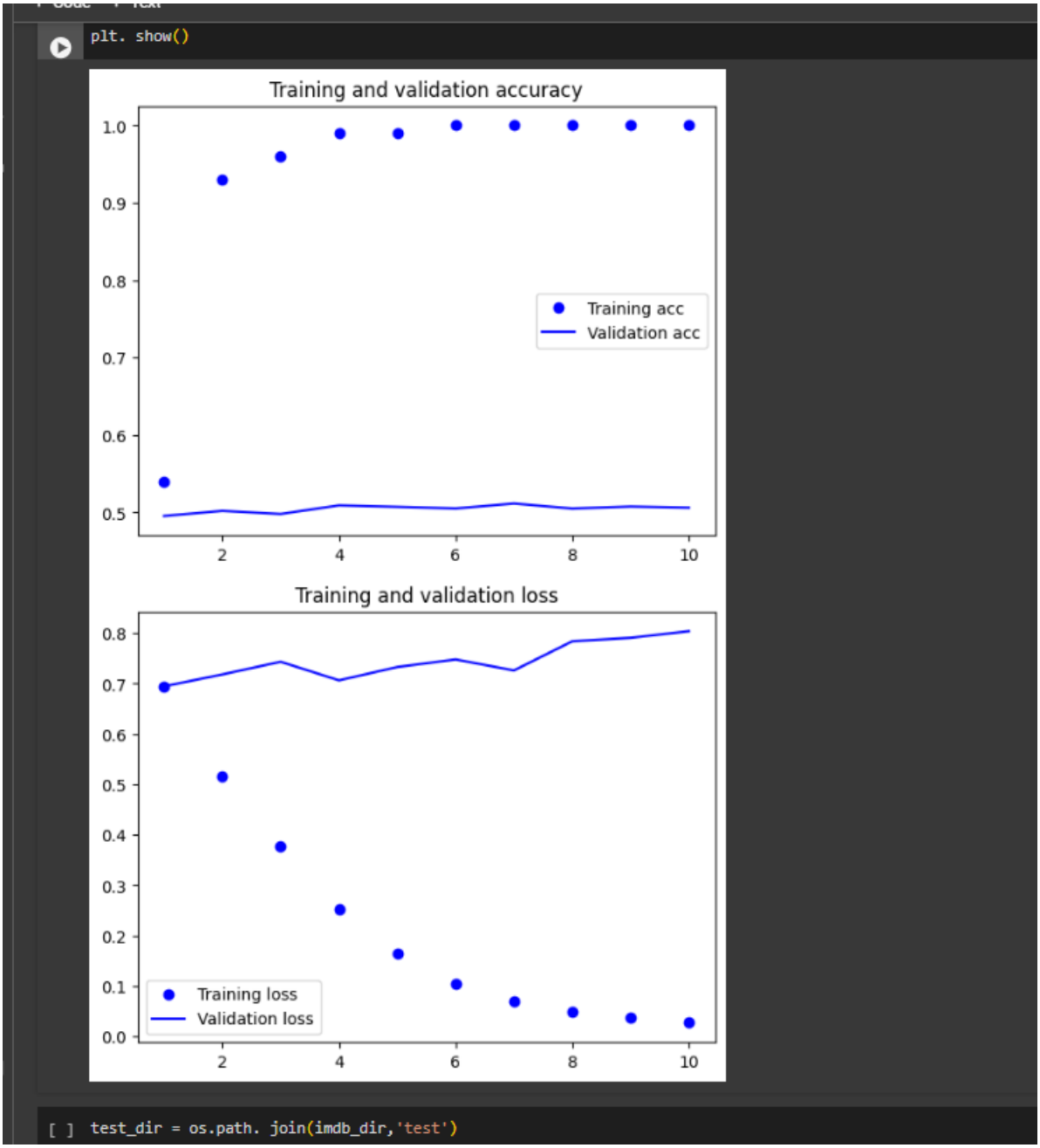


Two activities are now carried out with shell commands in a Jupyter Notebook or a comparable environment. To download a dataset, execute the wget command with the URL: **https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz**. This dataset includes movie reviews gathered from IMDb for sentiment analysis. Second, the tar command is used to extract the contents of the downloaded aclImdb_v1.tar.gz file. This dataset extraction stage provides access to the movie review data, which will most likely be used to train and test a sentiment analysis model.

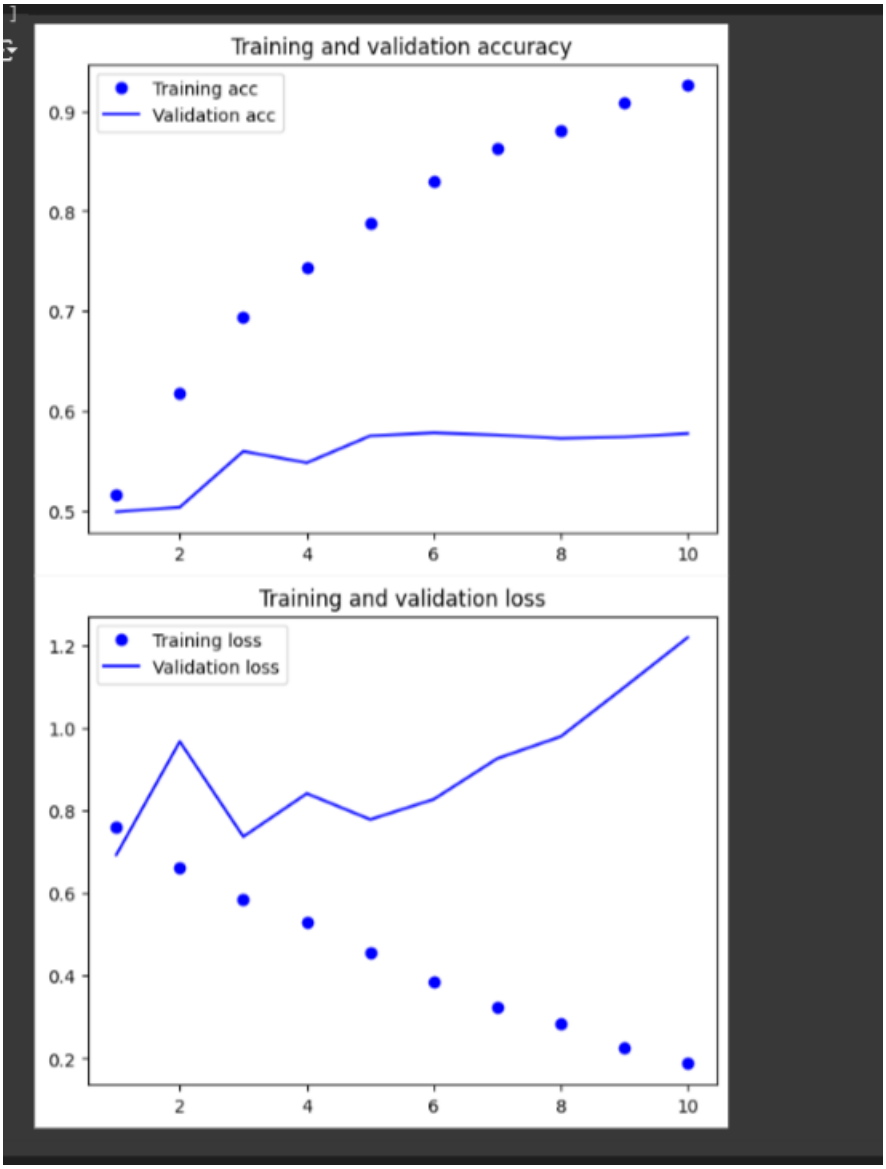
Now, Matplotlib is used to create a visualization of a sentiment analysis model's training and validation performance metrics based on IMDb movie reviews. The accuracy and loss values from the training and validation sets, obtained from the model's training history, are shown versus the number of epochs. Two distinct graphs are generated: one exhibiting training and validation accuracy over epochs, and the other depicting training and validation loss. By comparing performance on the training and validation datasets across numerous epochs, the plots reveal information about the model's learning progress as well as potential overfitting or underfitting issues.



In this code, The visualization method is the same as in the preceding excerpt. However, there is a little discrepancy between the plotting of training and validation accuracy and loss curves. The update involves removing superfluous spaces before the title and legend functions, which ensures proper syntax. This change improves code readability and formatting uniformity while leaving the visualization's functionality and output unchanged.



In this code, The structure and functionality of the previous excerpt remain the same, focusing on showing the training and validation accuracy, as well as the training and validation loss of a sentiment analysis model. The main distinction is in the structuring and layout of the code. Specifically, removing superfluous spaces between method calls and their arguments improves code readability while also maintaining style consistency. The overall functionality and output of the visualization remain intact; however, this update enhances the code's clarity and presentation.



```
[ ] model.load_weights('pre_trained_glove_model.h5')
    model.evaluate(x_test, y_test)

    782/782 [==============================] - 3s 4ms/step - loss: 1.0288 - acc: 0.5018
    [1.0288457870483398, 0.5018399953842163]
```

Summary Table: -

	method	Training Size	Training Accuracy	Validation Accuracy
1.	Embedding Layer	100	0.5800	0.5027
2.	Embedding Layer	10000	0.9670	0.8390
3.	Embedding Layer	15000	0.9640	0.8209
4.	Pre-trained	25000	0.9521	0.8430

Conclusion: -

In conclusion, the improvements made to the IMDB sentiment analysis example revealed that when dealing with limited data and review durations, both the embedding layer and pre-trained word embeddings contributed to improved performance. However, the number of training samples provided impacts when one strategy outperforms another. Initially, with a modest number of training samples (e.g., 100), the pre-trained word embedding strategy may outperform because of its capacity to exploit external knowledge. However, as the number of training samples grows, the embedding layer becomes increasingly successful at extracting domain-specific information from the data. The size of the available dataset and the intended trade-off between using external information should be considered while deciding between these options and learning from the dataset to perform optimally in sentiment analysis jobs.