Report

- ## Dataset creation & Pre-processing Steps:

1] The data has been provided in the form of multiple text files, where the content of the file is the input, while the label is represented by the parent folder's name (**'neg'** – meaning negative, **'pos'** – meaning positive)
2] For each of 'test' and 'train' folders, the following pre-processing steps were performed:

a] Positive and negative directory files were traversed & the corresponding inputs and labels were created:

Inputs: String containing file content
Labels: One hot-encoded, with [1, 0] representing positive and [0,1] representing negative reviews.

The labels do not require any further processing.

b] Next, we needed the input in a form that is better understandable to the machine. Hence, we tokenized the input; i.e.: mapped reviews to vector integers, with unique words represented by unique numbers.

c] Converted the input matrix to a uniform format:
We need the 2D array to have the same number of features (columns) per each input (row), to be used inside the network/model. Hence, we zero pad to have a 'balanced length'.
We choose the length as either 'half of the maximum length among inputs', or as the 'average length of inputs' (whichever is higher)

- ## How Final Design was chosen ?

1] We referred multiple online resources & learned how different approaches can be applied using RNN, CNN. Dense Layers etc (references are provided at the end). With this knowledge, we tried our own configurations.

2] We found a configuration with multiple Convolution layers to be performing best for our setup

3] Eventually we pinned down on the following Design for our network:

- **FINAL DESIGN:**

Input: Number of samples (90% of 25000) with 'n' features ('n' is the length of each input after padding)
Embedding: input_dim = 'No. of unique features' + 1 (consistent with format in keras), output_dim = 64
Convolution1D-1: with 64 filters, and 4 as the kernel_size
Convolution1D-2: with 32 filters, and 3 as the kernel_size
GlobalMaxPooling: With default setting
Dense-Layer: With 64 neurons and 'relu' activation
Dropout: with rate = 0.1
Dense-Layer: With 32 neurons and 'relu' activation
Dropout: with rate = 0.1
Dense-Layer: With 16 neurons and 'relu' activation
Dropout: with rate = 0.1
Dense-Layer: With only 2 neurons and 'softmax' activation

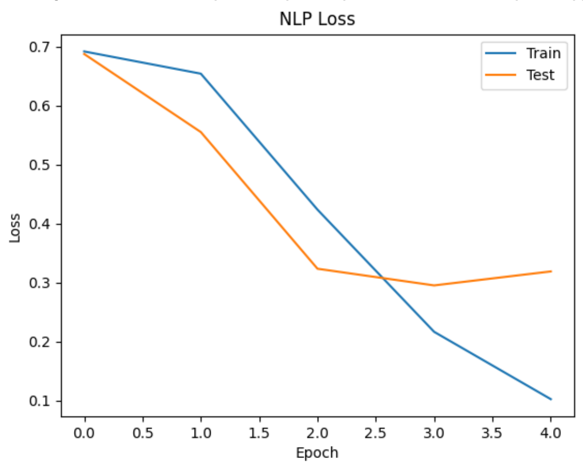
- 'Word Embeddings' help machine get a closer understanding of the words; hence used in the beginning. It is an approach, which provides similar representations for words with similar context.
- Word vectors are some points in space. By stacking a bunch of words (which in our case are represented by numeric vectors), we get something similar to an image. We can use the filters to slide over the word embeddings. Hence, we tried using the 2 convolution layers.
- Max-pooling, as we are well aware, is traditionally used in CNNs and is believed to improve the network performance. The same happens to be true in our case.
- The 3 dense layers with 'ReLu' as activation function, help the network capture complex relationships. Without above configuration of these dense layers, the testing accuracy was limited to 86%.

- We use Dropout layers in-between, so as to avoid overfitting (as we had observed and learned in Question 1 of this assignment).
- Since, we have a one-hot-encoded output, hence it makes sense to use Softmax here (which would normalize the output to a network of probability distribution).

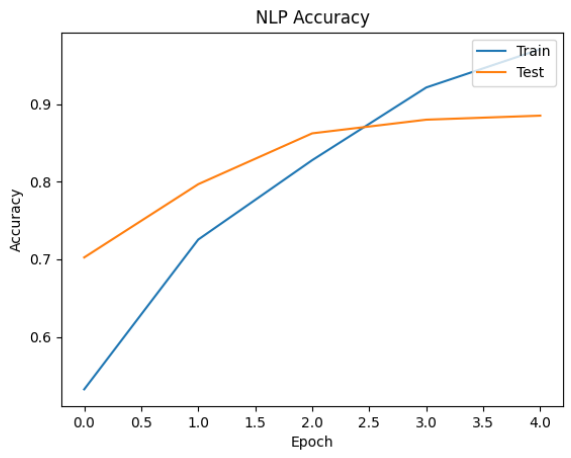
- **Number of Epochs = 5:** Post epoch=5, we observed that the training accuracy was getting very close to 97-99%; and hence limited the epochs, taking it as a sign of convergence.
- **Loss Function = Binary Cross-entropy:** Cross-entropy is an ideal loss function to be used, when we have 'softmax' at the output layer - returning a one-hot encoded vector. And since, we have only 2 classes, hence we use Binary Cross-entropy as our loss function.
- **Optimizer** = 'Adam', with default initial learning rate
- **Batch Size** = 1024

- **Training Output:** (With training data split into 90%-10% train-validation sets.)

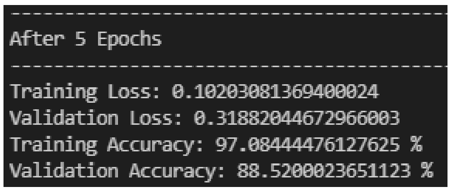
Epoch vs Loss (Train(90%) + Validation (10%))



Epoch vs Accuracy (Train(90%) + Validation(10%))



After 5 epochs, we were able to achieve the following results:

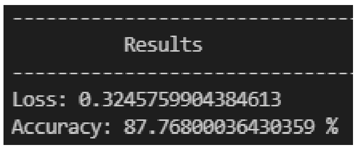


```
----------------------------------------
After 5 Epochs
----------------------------------------
Training Loss: 0.10203081369400024
Validation Loss: 0.31882044672966003
Training Accuracy: 97.08444476127625 %
Validation Accuracy: 88.5200023651123 %
```

As we can see in the above figures, even though training accuracy keeps increasing with each epoch; after a certain point – the validation accuracy starts fluctuating (mostly decreasing). The highest validation accuracy that we were able to achieve was ~88.5%.

The same trend can be seen in case of loss, albeit in the opposite direction (as loss is expected to decrease).

- **Testing Output:**



```
--------------------------------
          Results
--------------------------------
Loss: 0.3245759904384613
Accuracy: 87.76800036430359 %
```

As we can see, despite a high training accuracy, we achieve a relatively lower testing accuracy. We tried many different configurations, and found the above network to provide the best testing accuracy.

We also tried, by filtering out some special characters, but the difference was not significant.

We believe that having a larger training corpus (with better distribution) would help improve the testing accuracy.

References:

1 **Sentiment Analysis using LSTM & RNN:** (https://analyticsindiamag.com/how-to-implement-lstm-rnn-network-for-sentiment-analysis/)

2 **Text Classification using CNN:** (https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9)

3 **Text Sentiment Classification with LSTM & CNN:** (https://medium.com/@mrunal68/text-sentiments-classification-with-cnn-and-lstm-f92652bc29fd)

4 **Sentiment Classification using Feed Forward Neural Network:** (https://medium.com/swlh/sentiment-classification-using-feed-forward-neural-network-in-pytorch-655811a0913f)