EE782-AML
Assignment 2 Report

Shyam Thombre 170010023
Nitish Tongia 170010042
Rishabh Dahale 17D070008

November 18, 2020

1 Introduction

The given problem statement is analyzing a sequence for sentiment analysis and performing binary classification. This hints towards using Recurrent Neural Networks (RNN) as the deep learning model. Also, since we know LSTM is much better at avoiding vanishing and exploding gradients, we will be using the LSTM network and experimenting with dropouts, glove embeddings, number of LSTM layers, hidden dimension and lemmatization.

2 Word Embedding

We will be using the GloVe embeddings for the baseline model. Let us look at the semantic relationship that is established in GloVe embeddings which is trained on a giant corpus (6B tokens), for various embedding lengths - 50, 100, 200, 300. The code for checking this and the results are shown below.

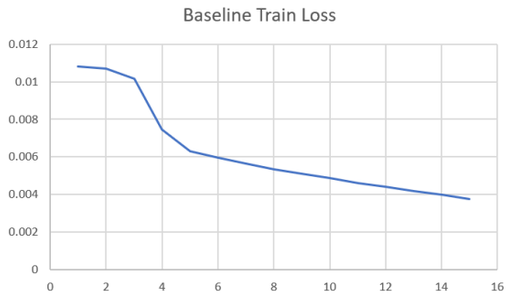
```
t1 = 'prince'
t2 = 'boy'
t3 = 'girl'
t4 = 'princess'

def semantic_relationship(embedding_dims, t1, t2, t3, t4):
  for i in embedding_dims:
    embed = {}
    with open(f'./GloVe/glove.6B.{i}d.txt', 'r') as f:
      for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embed[word] = coefs
    u = embed[t1] - embed[t2] + embed[t3]
    v = embed[t4]
    w = v-u
    dist = np.linalg.norm(w, 2)
    print(f"For embedding length {i}, the L2 norm of difference is {dist}")

semantic_relationship([50, 100, 200, 300], t1, t2, t3, t4)
```

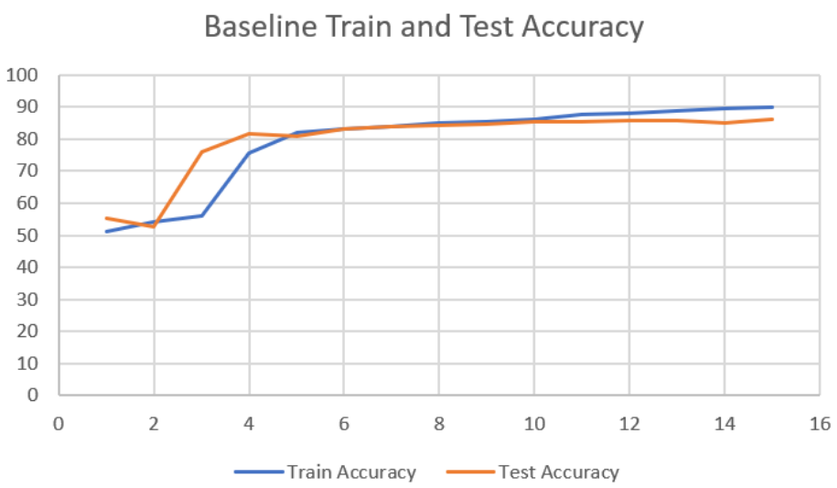
```
For embedding length 50, the L2 norm of difference is 3.250
For embedding length 100, the L2 norm of difference is 3.823
For embedding length 200, the L2 norm of difference is 5.421
For embedding length 300, the L2 norm of difference is 5.801
```

3 Baseline Model



(a) Train Loss

(b) Validation Loss



(c) Train and Validation Accuracy

Figure 1: Loss and accuracy of baseline model

		Confusion Matrix	
		True	False
Baseline	Positive	4310	688
	Negative	4289	713

F1 Score = 0.860
Precision = 0.862
Recall = 0.858

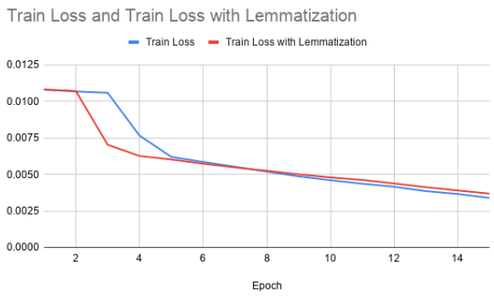
In the baseline model, we have used 1 LSTM layer along with glove embedding of 50 dimensions. For the baseline model we have kept the glove embedding untrainable. The hiden dimension of LSTM was kept as 64 and the model was trained with learning rate as 0.002. For baseline we unidirectional LSTM were used and dropout probability were kept at 0. For the corpus preprocessing, lemmatization was not used. This model was trained with Adam optimizer for 15 epochs. For numerical stability pytorch provides BCE with logits loss which combines BCE loss and sigmoid activation which was used.

4 Preprocessing

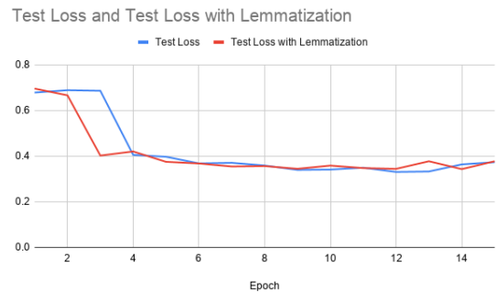
4.1 With and Without Lemmatization

Here, we observed the variation in the model's performance for LSTM with and without Lemmatization. From the Train and Validation Loss plots, we can observe that with Lemmatization, LSTM learns much faster initially as compared to without Lemmatization. But in the long run, we can see that the Train and validation loss converge to more or less same values.

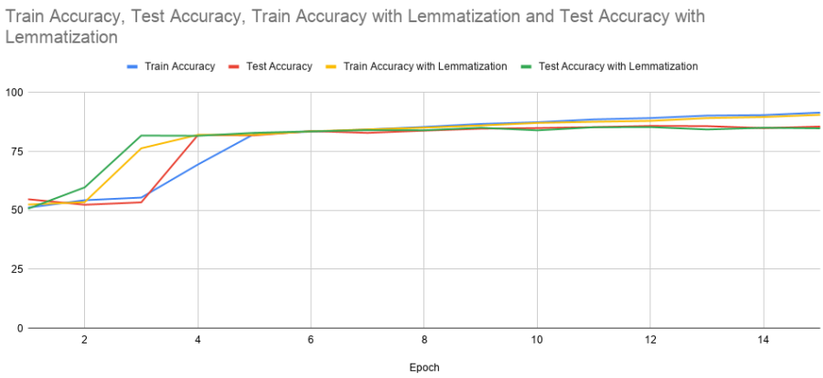
Also, from the Train and Validation Accuracy plot, we can see that the accuracy is improved much faster with Lemmatization as compared to without Lemmatization. Also the Accuracy's converge to a larger value with Lemmatization than without Lemmatization.



(a) Train Loss



(b) Validation Loss



(c) Train and Validation Accuracy

Figure 2: Comparison of loss and accuracy's for training and validation set with and without Lemmatization

		Confusion Matrix		
Without Lemmatization		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
With Lemmatization		True	False	F1 Score = 0.854
	Positive	4458	953	Precision = 0.823
	Negative	4024	565	Recall = 0.887

5 Hyperparameter tuning and Architectural Changes

5.1 Different Embedding Lengths

We can observe from the plot below that the Training loss decreases as we increase the embedding length. The difference is much more profound for lower values of embedding length. For very large values (above 200) after sufficient number of epochs, the training loss more or less converge to the same value.

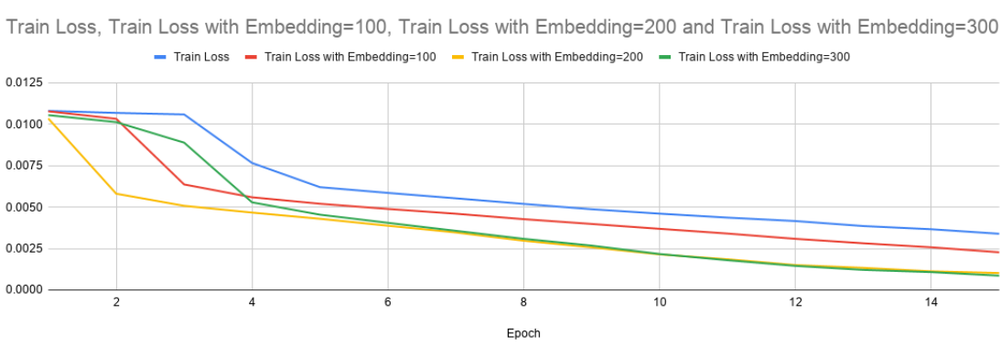


Figure 3: Train Loss

We can observe from the plot below that the Validation loss does not follow a similar trend as Training loss did. Initially it does decreases as we increase the embedding length, but after sufficient number of epochs, the loss values increases again. This can be explained with the understanding that the Training data is being over fitted due to which we see an increase in validation loss in-spite of the fact that the training loss reduces.

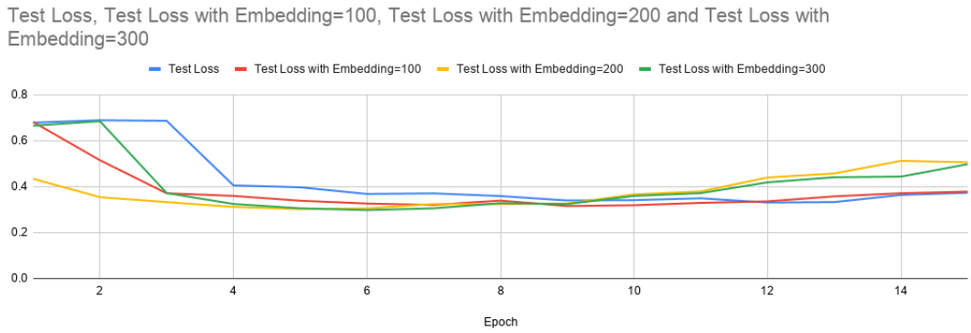


Figure 4: Validation Loss

The Figure below shows that the training accuracy plot behaves quite similar to the training loss as we can see that the accuracy keeps on increasing with increasing embedding lengths. However the Validation accuracy, just like validation loss, shows a decrease in value after some time. This again can be explained on the basis that due to over fitting, the test accuracy is reduced

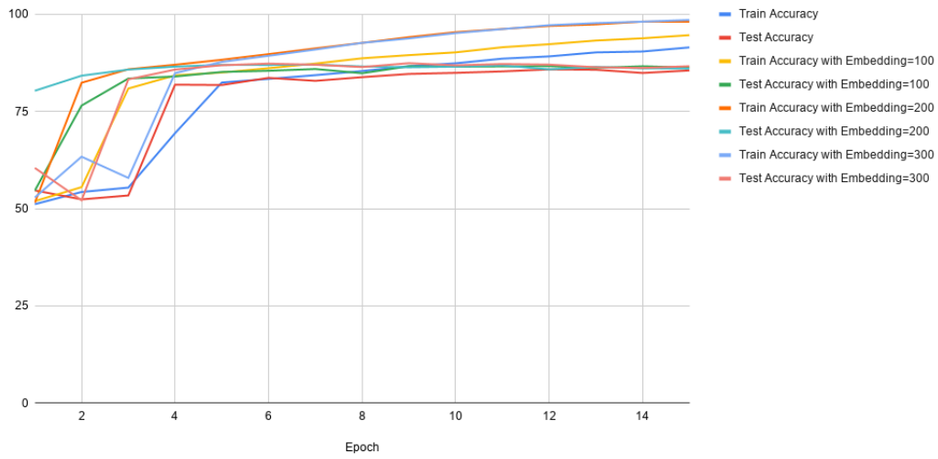
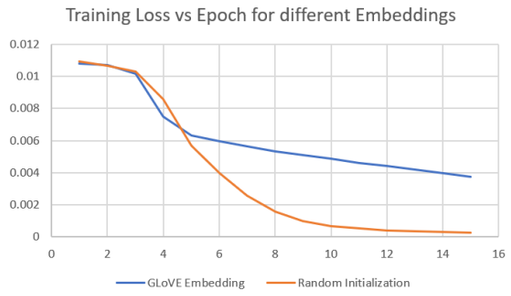


Figure 5: Train and Validation Accuracy

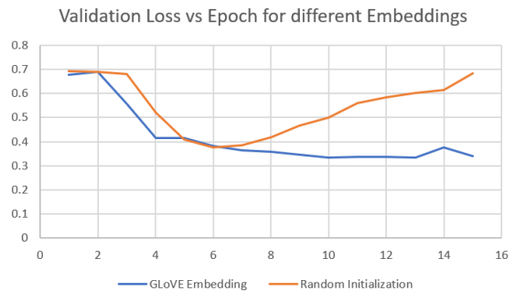
		Confusion Matrix		
Embedding Length = 50		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
Embedding Length = 100		True	False	F1 Score = 0.867
	Positive	4468	814	Precision = 0.845
	Negative	4163	555	Recall = 0.889
		Confusion Matrix		
Embedding Length = 200		True	False	F1 Score = 0.863
	Positive	4409	781	Precision = 0.849
	Negative	4196	614	Recall = 0.877
		Confusion Matrix		
Embedding Length = 300		True	False	F1 Score = 0.866
	Positive	4339	651	Precision = 0.869
	Negative	4326	684	Recall = 0.863

5.2 Embedding

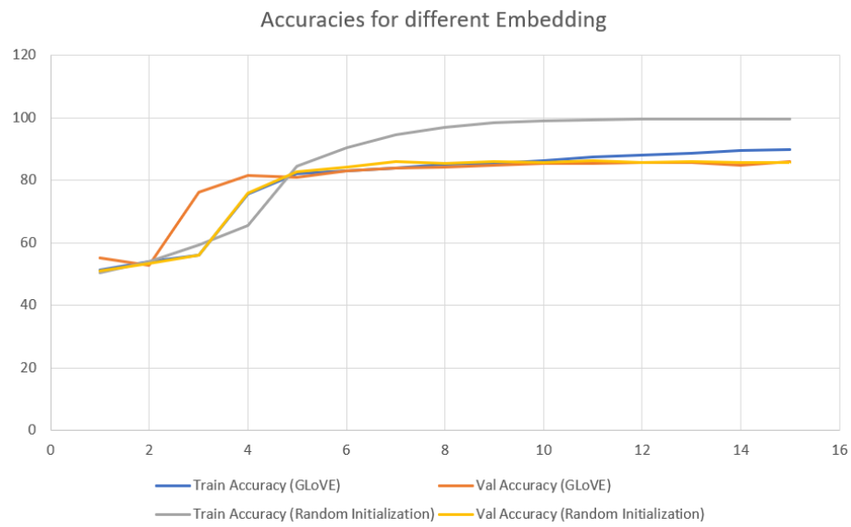
We have tested our model with 2 embedding: Glove and random initialization. We have used Glove embedding in baseline with non trainable parameters. We have compared this with a randomly initialized embedding of 50 dimension. The results for this can be seen in figure 6



(a) Train Loss



(b) Validation Loss



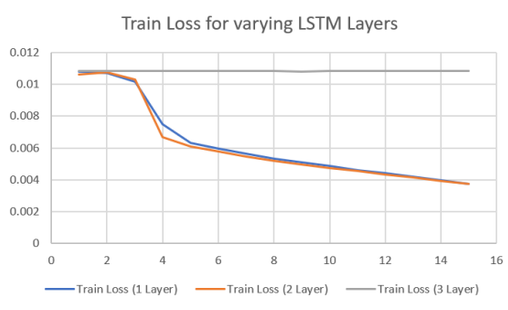
(c) Train and Validation Accuracy

Figure 6: Comparison loss and accuracy's for training and validation set for Glove embedding and randomly initialized embedding

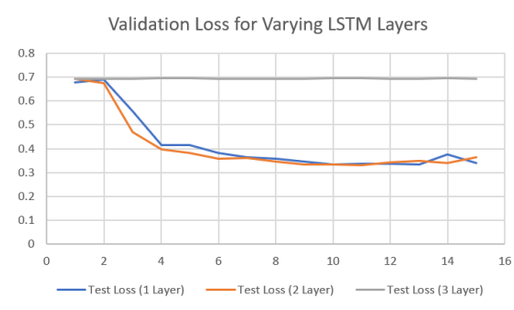
From the plots in figure 6 we can see that when we used randomly initialized embedding of same dimension as that of Glove in baseline, the randomly initialized embedding were able to perform better on the training set, but for validation set Glove embedding have lower loss indicating that the trainable embedding have overfit on the training set. This can be also seen from the accuracy plot in the figure. If we look at the training and validation accuracy, it can be seen that for random initialization embedding, the training accuracy's are as high as 99.65% where as validation accuracy is around 85%. This implies that non trainable glove embedding were able to achieve better generalization than the trainable randomly initialized embedding.

		Confusion Matrix		
Glove Embedding		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
Random Initialization		True	False	F1 Score = 0.860
	Positive	4417	827	Precision = 0.842
	Negative	4150	606	Recall = 0.879

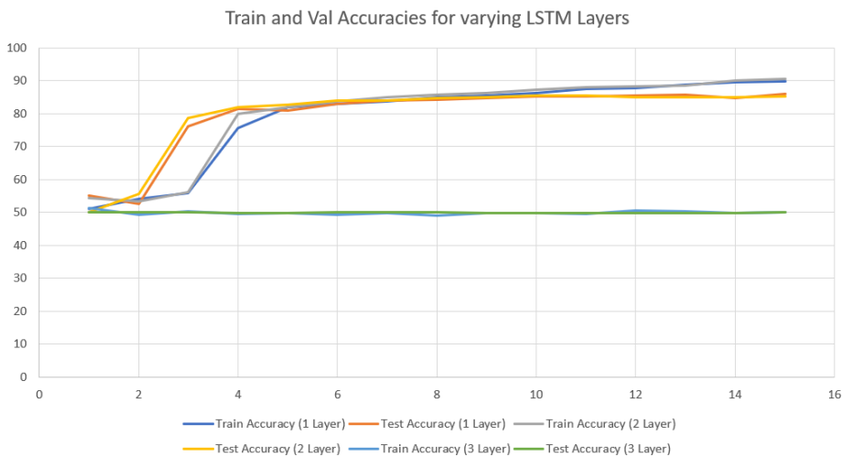
5.3 Number of LSTM Layers



(a) Train Loss



(b) Validation Loss



(c) Train and Validation Accuracy

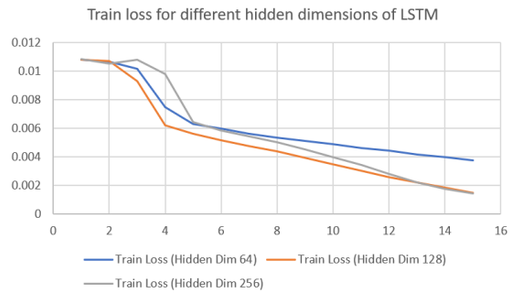
Figure 7: Comparison loss and accuracy's for training and validation set for different number of LSTM Layers

To understand the effect of changing the LSTM layers, we trained our model with 1, 2 and 3 LSTM layers. The results are present in the figure 7. It can be seen that the model was able to learn only when the number of LSTM layers were 1 and 2. For 3 LSTM layers, there was no training with non trainable glove embeddings. When we changed the embedding to random initialization, there was some training, but again there we could see model overfitting on the training set.

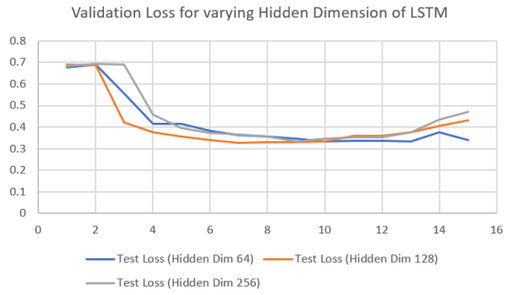
		Confusion Matrix		
1 LSTM Layer		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
2 LSTM Layers		True	False	F1 Score = 0.859
	Positive	4511	964	Precision = 0.824
	Negative	4013	512	Recall = 0.898
		Confusion Matrix		
3 LSTM Layers		True	False	F1 Score = 0.668
	Positive	5023	4977	Precision = 0.502
	Negative	0	0	Recall = 1

5.4 LSTM Hidden Dimension

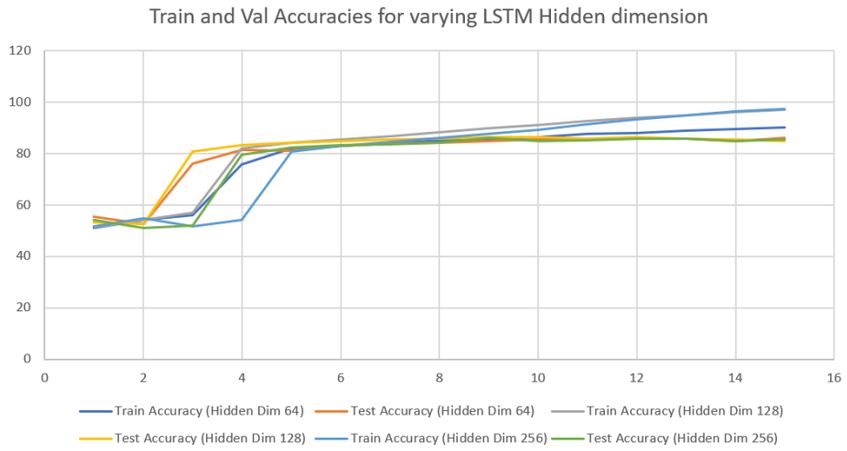
To understand the effect of changing the dimension of hidden layer of LSTM, we tried out 3 different values: 64, 128 and 256. The results are present in the figure 8. It can be seen that as the size of hidden dimension increased, the model was able to better generalize the training set. This can be said because training loss for higher dimension is lower than that for lower dimension, while validation loss is almost the same.



(a) Train Loss



(b) Validation Loss



(c) Train and Validation Accuracy

Figure 8: Comparison loss and accuracy's for training and validation set for varying LSTM hidden dimension

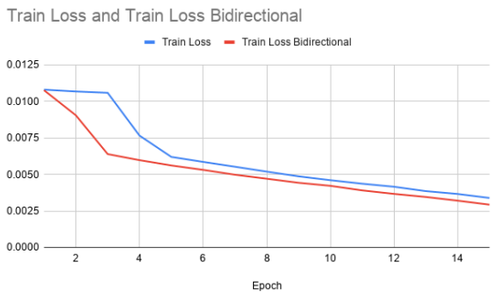
		Confusion Matrix		
64 LSTM Hidden Dimension		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
128 LSTM Hidden Dimension		True	False	F1 Score = 0.849
	Positive	4338	852	Precision = 0.836
	Negative	4125	685	Recall = 0.863
		Confusion Matrix		
256 LSTM Hidden Dimension		True	False	F1 Score = 0.854
	Positive	4262	698	Precision = 0.859
	Negative	4279	761	Recall = 848

6 BONUS

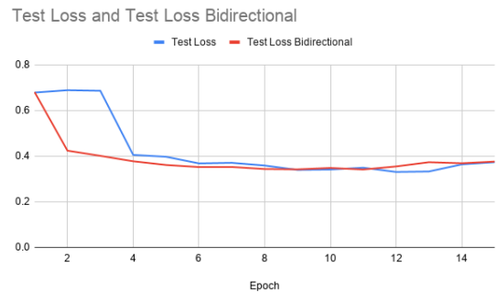
6.1 Unidirectional vs Bidirectional LSTM

We wish to study the behaviour of our model when the LSTM is unidirectional and bidirectional. In the plots below, we compare the variation in the model's performance for unidirectional and bidirectional LSTM's. From the Train and Validation Loss plots, we can observe that bidirectional LSTM learns much faster initially as compared to the unidirectional LSTM. But in the long run, we can see that the Train and validation loss converge to more or less same values. A little increase in the validation loss values at larger number of epochs is because of overfitting.

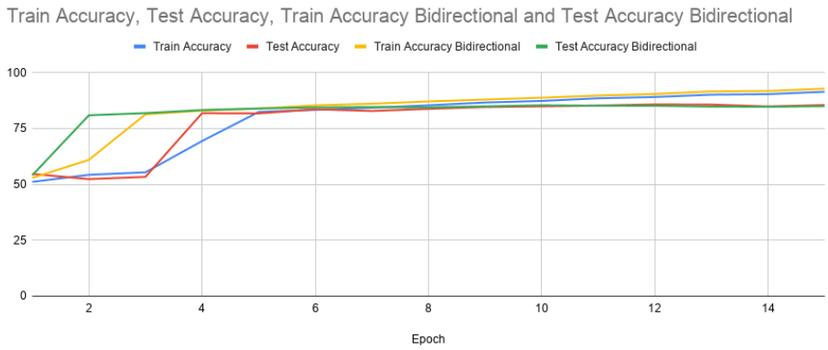
Also, from the Train and Validation Accuracy plots shown below, we can see that the accuracy is improved much faster when the LSTM is bidirectional as compared to the unidirectional LSTM. Also the Accuracy's converge to a significant larger value for bidirectional LSTM.



(a) Train Loss



(b) Validation Loss



(c) Train and Validation Accuracy

Figure 9: Comparison of loss and accuracy's for training and validation set for Bidirectional and Unidirectional LSTM

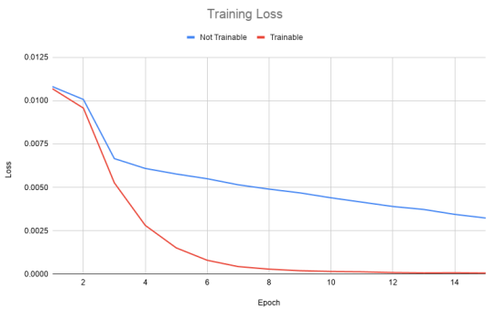
		Confusion Matrix		
Unidirectional LSTM		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
Bidirectional LSTM		True	False	F1 Score = 0.862
	Positive	4322	795	Precision = 0.826
	Negative	4182	701	Recall = 0.901

6.2 Freezing GloVe Embeddings vs Keeping Trainable

Using the GloVe embeddings gave very good results with just a few epochs during training. However, we can either keep the weights of of the embedding layer as fixed, or allow room for fine tuning of the weights. In the plots given below, we can observe the Training Loss, Test Loss and the corresponding Accuracy for both the cases.

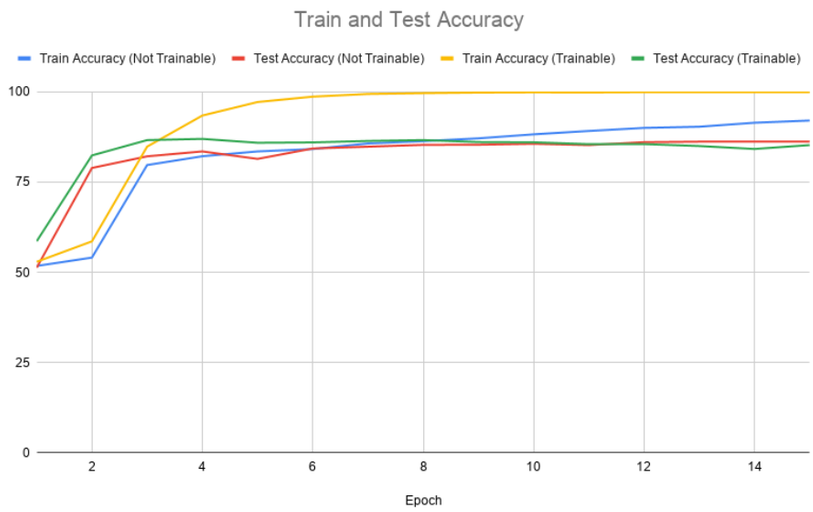
The loss curves indicate that both the models learn well, but it is clearly observed that when the weights are trainable, the overfitting occurs much earlier, whereas, the overfitting is not observed in the other case. However, the test accuracy doesn't seem to change much in either case. It is possible that the learning rate during fine tuning was a bit higher which led to quicker overfitting.

		Confusion Matrix		
Frozen Embeddings		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
Trainable Embeddings		True	False	F1 Score = 0.857
	Positive	4435	888	Precision = 0.833
	Negative	4089	588	Recall = 0.883



(a) Train Loss

(b) Validation Loss



(c) Train and Validation Accuracy

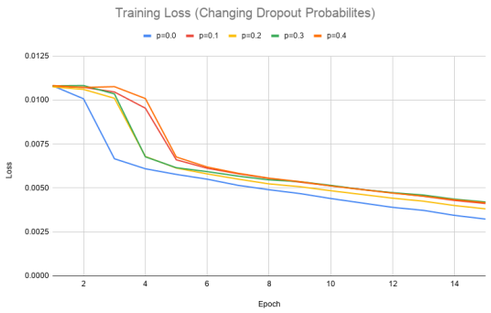
Figure 10: Checking if fine tuning of embeddings improves performance

6.3 Using Dropouts with Non-trainable GloVe Embeddings

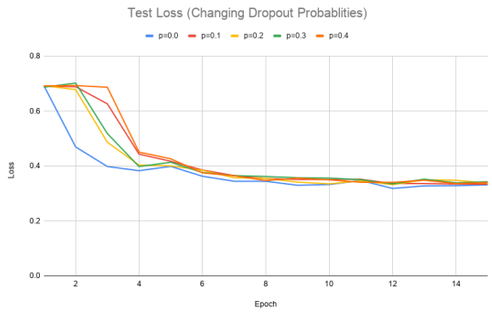
		Confusion Matrix		
Without Dropouts		True	False	F1 Score = 0.860
	Positive	4310	688	Precision = 0.862
	Negative	4289	713	Recall = 0.858
		Confusion Matrix		
Dropout - p=0.1		True	False	F1 Score = 0.861
	Positive	4340	716	Precision = 0.858
	Negative	4261	683	Recall = 0.864
		Confusion Matrix		
Dropout - p=0.2		True	False	F1 Score = 0.858
	Positive	4320	729	Precision = 0.856
	Negative	4248	703	Recall = 0.860
		Confusion Matrix		
Dropout - p=0.3		True	False	F1 Score = 0.851
	Positive	4228	680	Precision = 0.861
	Negative	4297	795	Recall = 0.842
		Confusion Matrix		
Dropout - p=0.4		True	False	F1 Score = 0.852
	Positive	4155	578	Precision = 0.878
	Negative	4399	868	Recall = 0.827

Overfitting was observed in many cases mentioned above. So we tried using dropout for regularization and see how the performance changes. The dropout probabilities (1 - keep-probability) were varied as 0.0, 0.1, 0.2, 0.3, 0.4 and keeping the GloVe Embeddings as not trainable. The regularization effect didn't affect the final loss or accuracy. This is evident from the F1-score for each experiment. The plots for this can be seen below.

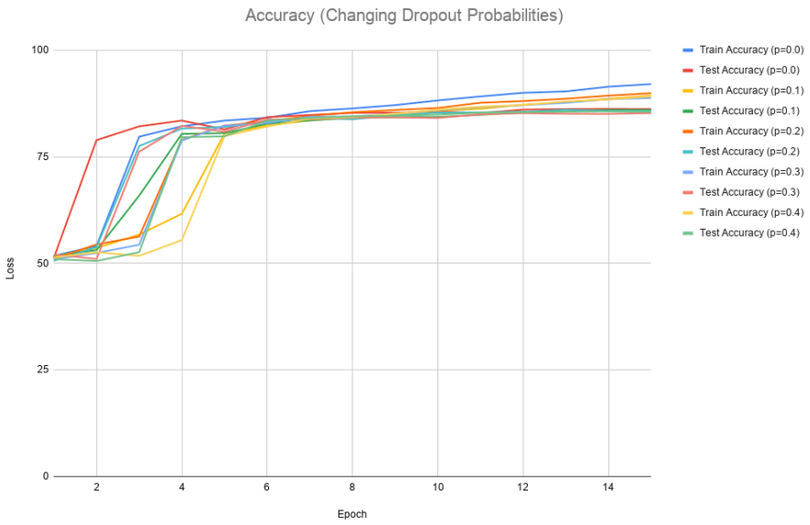
For all dropout probabilities the trend of the loss and accuracy curves is observed to be the same. However, with different probabilities, the time (in epochs) for the convergence changed. The baseline model itself converged fastest. This might be indicative of low expressive power of the model, and hence dropout regularization is not useful for the baseline model that has been specified.



(a) Train Loss



(b) Validation Loss



(c) Train and Validation Accuracy

Figure 11: Using Dropout for regularization with non-trainable GloVe Embeddings

7 References

- GloVe Embedding
- NLTK Lemmatization
- Tokenizing
- Creating Custom Dataset
- Debugging by checking gradient flow