Final Project

Group 6 : Raktim Roychoudhury (rr4081)

Goal:

For our project, we are tasked with building different classifiers to perform sentiment analysis on Twitter data.

Data:

To source our data, we initially attempt to source the tweet data directly using the Twitter API. However, due to recent changes to their usage terms, we are unable to directly access tweets and therefore use a Kaggle dataset.

The dataset contains 300,000 tweets classified with 'positive' and 'negative' labels. The dataset is balanced in an approximately 50:50 ratio. We use a relatively large dataset because tweets have a lot of different permutations for the same word. For example, 'you' and 'u' are frequently used interchangeably in tweets. This results in a larger tokenization vocabulary than ordinary English text data, resulting in a larger word embedding size for which we need more training data.

Pre-processing:

We first perform some basic pre-processing to remove elements like hashtags and usernames and replace them with standardized lowercase text throughout the corpus.

We next process our text corpus of tweets using two approaches :

- Developing our own vocabulary

We identify each unique word in the text corpus with a minimum frequency of N = 10. We decide to use a minimum frequency to eliminate unnecessary words like typos, twitter usernames etc. and ensure that the vocabulary size isn't too large. We replace the words with tokens corresponding to each word in the vocabulary. For new/missing words, we replace them with the '<unk>' token. The resulting vocabulary has 6,568 unique tokens.

- Using a pre-trained BERT vocabulary

To compare approaches, we also tokenize our text corpus using a vocabulary from a pre-trained Bi-directional Encoding Representation using Transformers (BERT) model. The BERT vocabulary contains 30,522 unique tokens. Hence, we compare a Word2Vec embedding vs a Transformer embedding.

We then randomly split the dataset of tweets into training and validation sets in a 90:10 ratio.

Models:

For the sentiment analysis task, we decide to take two approaches. Firstly, we build a Word2Vec model with Gated Recurrent Units (GRUs) networks and Long-Short Term Memory (LSTMs) networks. We use both uni-directional and bi-directional GRU/LSTM layers which take the input from the embedding layer.

For the Word2Vec model, we experiment with two different embedding sizes, 256 and 128. The embedding size corresponds to the length of the embedding vector for each token in the vocabulary. During training, we first generate the embedding for each word in a tweet and pass the resultant embedding vectors to our GRU/LSTM layers, using a sigmoid activation at the output layer to predict a sentiment probability between 0 and 1. We then calculate the cross-entropy loss and update the weights of the recurrent and embedding layers using backpropagation. We use an Adaptive Moment Estimation (ADAM) optimizer with learning rate = 0.01.

We compare this model against a much larger pre-trained BERT models with GRU/LSTM layers. BERT has an embedding-size of 768, and has been trained on a very large corpus of English text. The training approach is similar to the Word2Vec model, however, we do not update the embedding i.e. BERT layer weights.

Results and Observations

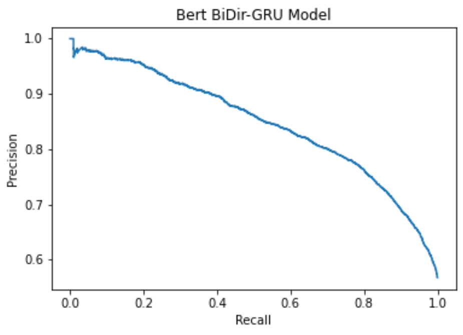
No.	Embedding	Layer Type	No. of Layers	Bi-Directional	Accuracy	Precision	Recall	F1-Score	AUROC
1	Word2Vec (256)	GRU	2	Yes	0.57	0.57	1.00	0.72	0.50
2	Word2Vec (256)	GRU	4	Yes	0.57	0.57	1.00	0.72	0.50
3	Word2Vec (128)	GRU	2	Yes	0.43	1.00	0.00	0.00	0.48
4	Word2Vec (128)	GRU	4	Yes	0.43	1.00	0.00	0.00	0.48
5	Word2Vec (128)	LSTM	4	Yes	0.57	0.57	1.00	0.72	0.50
6	Pre-trained BERT	GRU	4	Yes	0.74	0.75	0.82	0.78	0.81
7	Pre-trained BERT	GRU	4	No	0.43	1.00	0.00	0.00	0.57
8	Pre-trained BERT	LSTM	4	Yes	0.43	1.00	0.00	0.00	0.56
9	Pre-trained BERT	LSTM	4	No	0.57	0.57	1.00	0.72	0.50

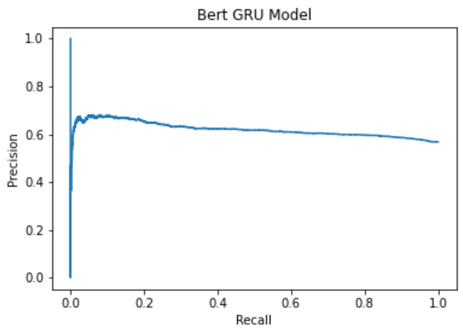
Table 1 : Results

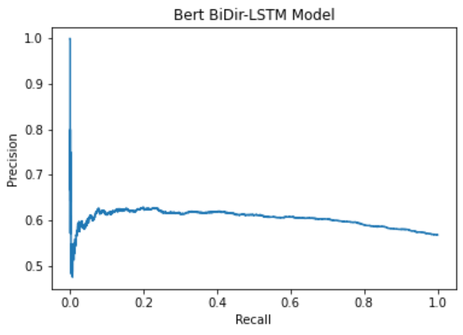
We present the results of our model performance on the test set of 10,000 tweets in Table 1. For our custom Word2Vec embedding, we observe that the model does not perform very well. For both the embedding sizes, 128 and 256 the model has a very poor precision/recall score. This indicates that the model predictions are either only 1 or only 0, hence although the model has a 50% accuracy, the entropy of the model is very poor. One possible reason for this could be due to the relatively small size of our custom vocabulary. As highlighted earlier, internet text tend to have a lots of different permutations and spellings for similar words which could result in extremely sparse matrices, which bring about issues of optimisation. Moreover, the vanishing gradient problem frequently observed while training recurrent networks could also result in the poor results observed. A possible solution to the latter problem would be to use intermediate batch-normalization, which has shown to resolve the exploding/vanishing gradient issue.

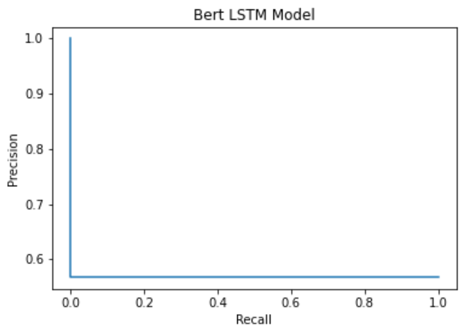
For the BERT embeddings, we observe that we achieve the best results of a 74% test set accuracy and 0.78 F1-score, for the case of 4 bi-directional GRUs. This highlights that the higher embedding size and well trained embedding play a significant role in the overall performance of the sentiment analysis task. However, we also highlight that the same performance is not reflected for the uni-directional or the LSTM case. This reinforces our hypothesis that optimising deep recurrent networks is not very efficient, therefore we would need to add batch-norm layers to improve our performance.

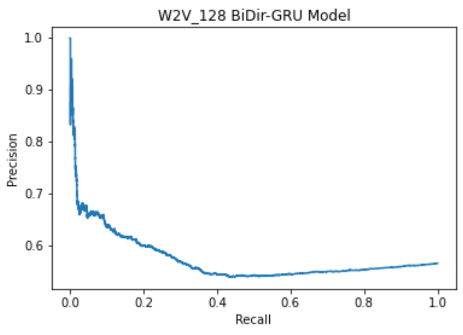
Precision-Recall Graphs

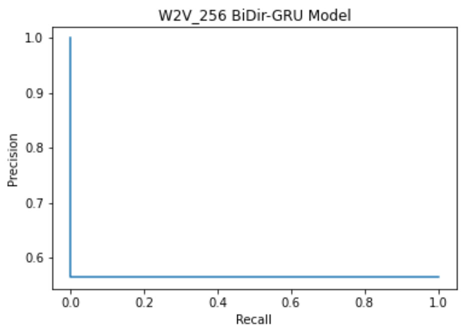












From the precision-recall graphs, we can see that the BERT Bidirectional GRU model with 4 layers performs best on the sentiment analysis task. Since we don't observe a similar performance for the LSTM architecture, we hypothesize that the simplicity of the GRU network is better for optimisation using backpropagation. The observations from the other precision-recall curves highlight that the model is not well specified, as can be seen by the sharp drop offs as we increase the classification threshold. Therefore, this indicates that the distribution learned by the model during training has very low entropy. Therefore, we need to improve our architecture to get better results.