Lab 3 Report | Submission | Deep Learning & Neural Nets

Submitted by: Anusha Basavaraja
(anba9017@colorado.edu)

Part 1: Report

Impact of RNN Architecture:

Methods:

Dataset:

I am using a spam classification dataset from Kaggle available at – (https://www.kaggle.com/datasets/chandramoulinaidu/spam-classification-for-basic-nlp) which has about 5796 instances in it with two categories (spam/not spam) in it. The label '1' indicates that the message is spam and '0' indicates that the message is not spam. The dataset consists of 3900 instances of 'not spam' category and 1896 instances of 'spam' category data in it. Even though the dataset has multiple columns in it, I am only considering the 'MESSAGE' and 'CATEGORY' columns for this Lab assignment which serves as 'text' input and 'label' output respectively for the models. As mentioned in the assignment, the split between train-test dataset is 70%-30%. So, the test set consists of 1165 instances of 'not spam' category and 575 instances of 'spam' category data in it.

Experiment:

For the experiment part, I have chosen three RNN architectures – SimpleRNN, LSTM and GRU models which has the following hyperparameters that are set constant throughout this experiment.

- Layers: Sequential model with following layers in the order of input – Embedding layer – RNN layer – output Dense layer
- The raw text is converted into vector form of size 1000 which serves as input to the model at the input layer.
- The Embedding layer has the dimension of vocabulary size * 64 (vocabulary size set to 10,000) which takes the input vectors of size 1000 from the previous input layer.
- The RNN layer (simpleRNN/LSTM/GRU) used has 64 units specified with default activation function 'tanh'.

- The output dense layer has 1 node with ‘sigmoid’ activation function which follows the technique of predicting ‘1’ if the output layer results in the value > 0.5 and ‘0’ if the output layer results in the value <= 0.5
- The loss considered is ‘binary_crossentropy’
- The optimizer used is ‘adam’
- The metrics used for compiling the model is ‘accuracy’
- No. of training iterations used is 3 epochs (to avoid overfitting)
- No regularization incorporated.
- The models are executed on Google Colab without GPU support.

These hyperparameters are set with the values shown throughout the experiment because after experimenting for while with the different values of hyperparameters, these values resulted in good results and I thought it helps in understanding the model behaviors with respect to different model executions.

For experimenting with different architecture of RNN, I have used three models – simpleRNN, LSTM and GRU which replaces the RNN layer respectively in the model architecture.

For testing the model based on the size of the input (short/medium/long), same 30% of the test set is divided into three categories depending on the size of the input message/text length. The short message category consists of 482-97 spam-not spam instances, medium message category consists of 426-153 spam-not spam instances, long messages category consists of 257-322 spam-not spam instances in it. For each of these 3 categories – short, medium and long messages, all three models were executed resulting in 9 different outputs.

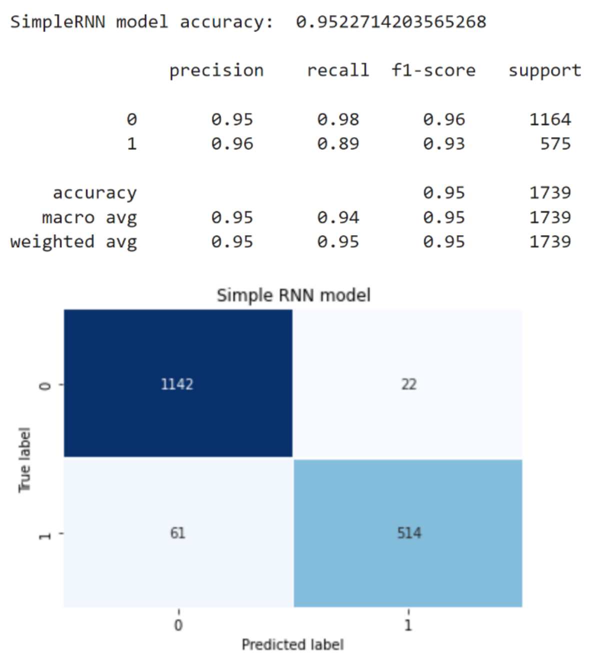
Results:

a) For testing the model on 30% of test dataset using three different RNN architectures.
    - SimpleRNN model
    - LSTM
    - GRU

__Note__: No. of ‘spam’ instances = 575 and ‘not spam’ instances = 1164 in the test dataset.

Sl No.	*Model*	*Accuracy*	*Precision*	*Recall*	*# TP*	*# TN*	*# FP*	*# FN*
1.	SimpleRNN	95.23%	95%	95%	1142	512	22	61
2.	LSTM	98.62%	99%	99%	1158	557	6	18
3.	GRU	98.96%	99%	99%	1162	559	2	16

Table: Evaluation metrics for predicting the model performance on the test dataset while experimenting with different RNN architectures.

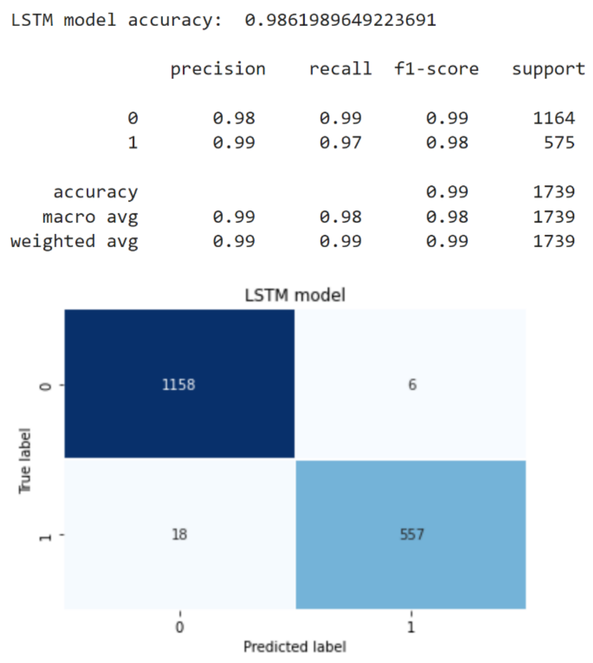


```
SimpleRNN model accuracy:  0.9522714203565268

              precision    recall  f1-score   support

           0       0.95      0.98      0.96      1164
           1       0.96      0.89      0.93       575

    accuracy                           0.95      1739
   macro avg       0.95      0.94      0.95      1739
weighted avg       0.95      0.95      0.95      1739
```

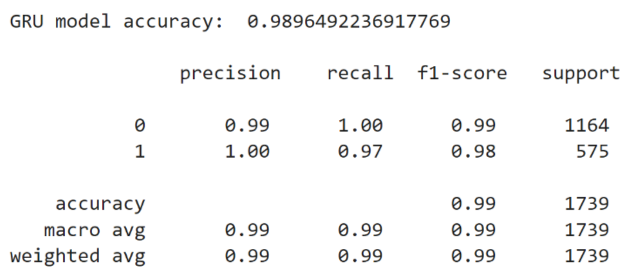


```
LSTM model accuracy:  0.9861989649223691

              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1164
           1       0.99      0.97      0.98       575

    accuracy                           0.99      1739
   macro avg       0.99      0.98      0.98      1739
weighted avg       0.99      0.99      0.99      1739
```



```
GRU model accuracy:  0.9896492236917769

              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1164
           1       1.00      0.97      0.98       575

    accuracy                           0.99      1739
   macro avg       0.99      0.99      0.99      1739
weighted avg       0.99      0.99      0.99      1739
```

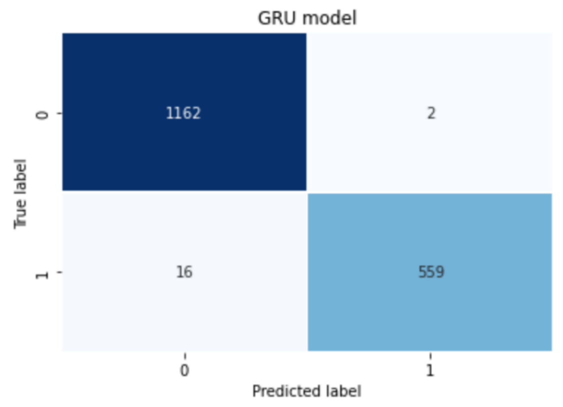


Fig: Model evaluation metrics for the prediction on test dataset using
1) SimpleRNN model   2) LSTM model   3) GRU model architectures

b) For testing the model on the input size of test dataset (short, medium, long text in the dataset). All three RNN architectures are again tested on short, medium and long length messages in the test dataset.

Note: Short text in test dataset has 482 ‘spam’ and 97 ‘not spam’ instances.
Medium text in test dataset has 426 ‘spam’ and 153 ‘not spam’ instances.
Long text in test dataset has 257 ‘spam’ and 322 ‘not spam’ instances.

Sl No.	***Model***	***Accuracy***	***Precision***	***Recall***	***# TP***	***# TN***	***# FP***	***# FN***
1.	SimpleRNN_short	97.93%	98%	98%	480	87	2	10
2.	SimpleRNN_medium	98.10%	98%	98%	421	147	5	6
3.	SimpleRNN_long	96.03%	96%	96%	250	306	7	16
4.	LSTM_short	99.48%	99%	99%	481	95	1	2
5.	LSTM_medium	98.96%	99%	99%	423	150	3	3
6.	LSTM_long	98.96%	99%	99%	256	317	1	5
7.	GRU_short	99.65%	100%	100%	482	95	0	2
8.	GRU_medium	99.65%	100%	100%	426	151	0	2
9.	GRU_long	99.13%	99%	99%	255	319	2	3

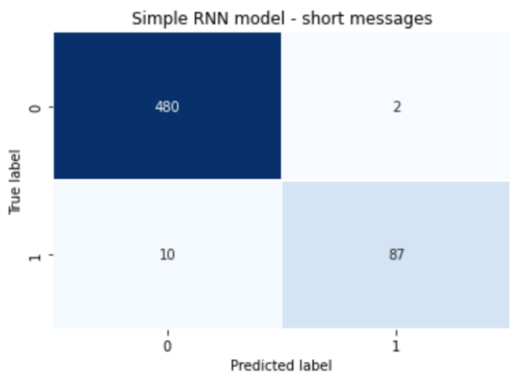
Table: Evaluation metrics for predicting the model performance on the short, medium and long input size text sequences in the test dataset

```
SimpleRNN model accuracy on short_messages:  0.9792746113989638

              precision    recall  f1-score   support

           0       0.98      1.00      0.99       482
           1       0.98      0.90      0.94        97

    accuracy                           0.98       579
   macro avg       0.98      0.95      0.96       579
weighted avg       0.98      0.98      0.98       579
```

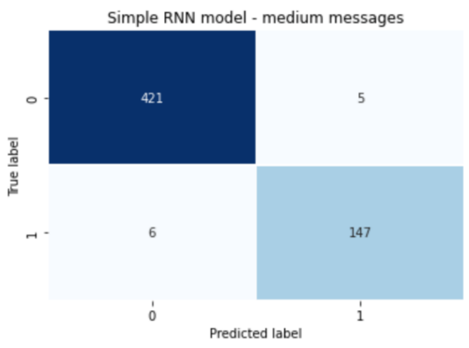


```
SimpleRNN model accuracy on medium_messages:  0.9810017271157168

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       426
           1       0.97      0.96      0.96       153

    accuracy                           0.98       579
   macro avg       0.98      0.97      0.98       579
weighted avg       0.98      0.98      0.98       579
```

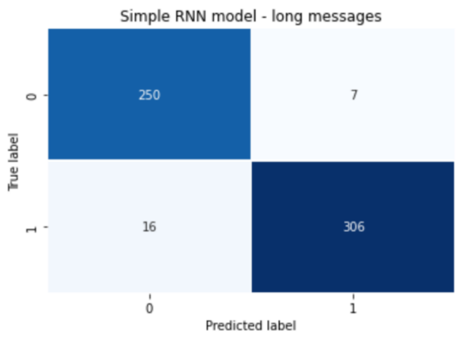


```
SimpleRNN model accuracy on long_messages:  0.9602763385146805

              precision    recall  f1-score   support

           0       0.94      0.97      0.96       257
           1       0.98      0.95      0.96       322

    accuracy                           0.96       579
   macro avg       0.96      0.96      0.96       579
weighted avg       0.96      0.96      0.96       579
```

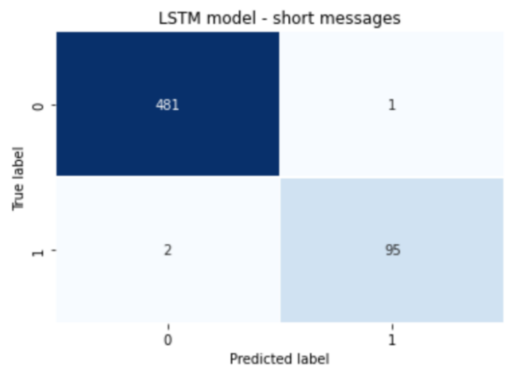


```
LSTM model accuracy on short_messages:  0.9948186528497409

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       482
           1       0.99      0.98      0.98        97

    accuracy                           0.99       579
   macro avg       0.99      0.99      0.99       579
weighted avg       0.99      0.99      0.99       579
```

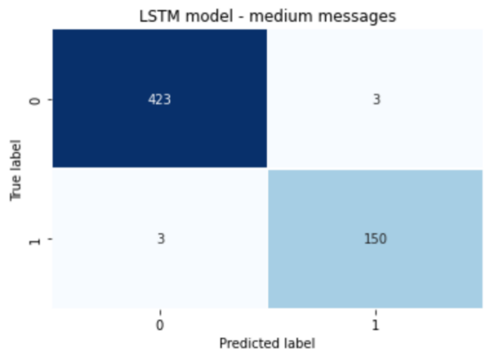


```
LSTM model accuracy on medium_messages:  0.9896373056994818

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       426
           1       0.98      0.98      0.98       153

    accuracy                           0.99       579
   macro avg       0.99      0.99      0.99       579
weighted avg       0.99      0.99      0.99       579
```

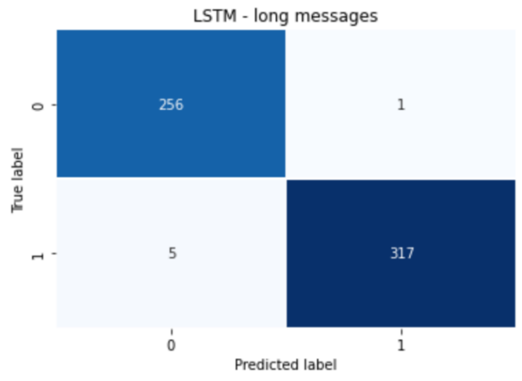


```
LSTM model accuracy on long_messages:  0.9896373056994818

              precision    recall  f1-score   support

           0       0.98      1.00      0.99       257
           1       1.00      0.98      0.99       322

    accuracy                           0.99       579
   macro avg       0.99      0.99      0.99       579
weighted avg       0.99      0.99      0.99       579
```

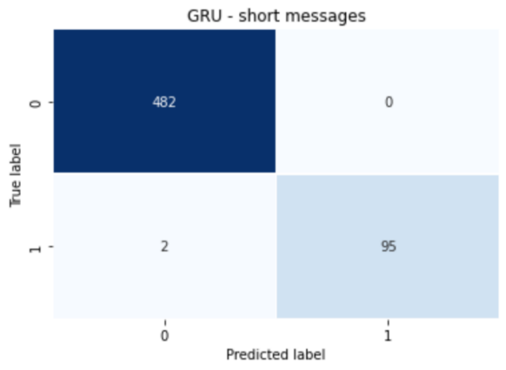


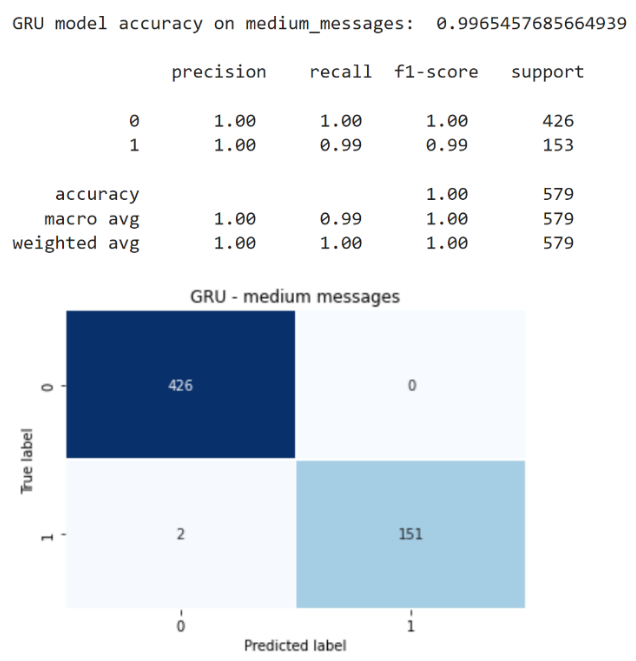
```
GRU model accuracy on short_messages:  0.9965457685664939

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       482
           1       1.00      0.98      0.99        97

    accuracy                           1.00       579
   macro avg       1.00      0.99      0.99       579
weighted avg       1.00      1.00      1.00       579
```

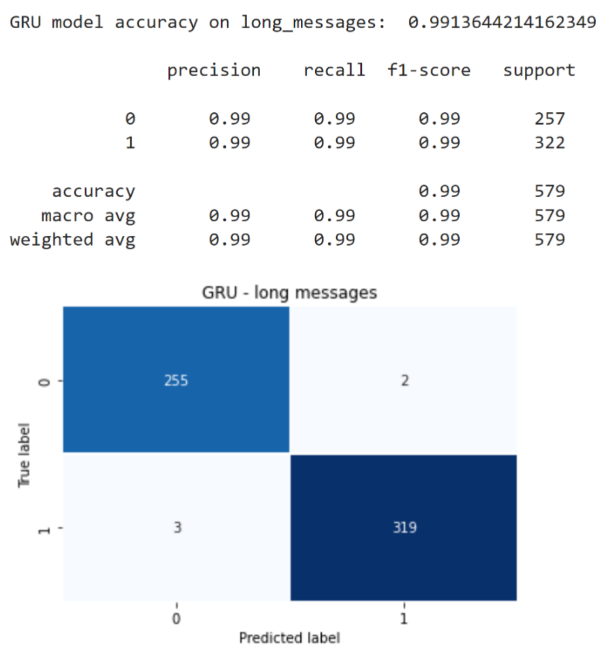




```
GRU model accuracy on medium_messages:  0.9965457685664939
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       426
           1       1.00      0.99      0.99       153

    accuracy                           1.00       579
   macro avg       1.00      0.99      1.00       579
weighted avg       1.00      1.00      1.00       579
```



```
GRU model accuracy on long_messages:  0.9913644214162349
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       257
           1       0.99      0.99      0.99       322

    accuracy                           0.99       579
   macro avg       0.99      0.99      0.99       579
weighted avg       0.99      0.99      0.99       579
```

<u>Fig</u>: Model evaluation metrics for the prediction on partial test dataset (short/medium/long) using 1) SimpleRNN model on short input 2) SimpleRNN model on medium input 3) SimpleRNN model on long input 4) LSTM model on short input 5) LSTM model on medium input 6) LSTM model on long input 7) GRU model on short input 8) GRU model on medium input 9) GRU model on long input

Analysis:

1) *Did a certain type of RNN architecture lead to better results?*

   When we observe the 1st table with results by experimenting with different RNN architectures, we can notice that both LSTM and GRU models have comparable results by performing better than simple RNN or vanilla RNN model. This might be because vanilla RNN have vanishing gradients problem and whereas both LSTM and GRU has gate mechanism that allows the model to retain and forget the information as needed thereby allowing the model to have the required sufficient information in predicting the accurate results.

2) *Are there performance trends across three models with respect to how they handle short vs medium vs long inputs?*

If we observe the 2nd table with results by testing all three models on short, medium and long inputs, we can again notice that both LSTM and GRU has very good performance on short, medium and long inputs with very little to no difference when we compare their evaluation metrics. So, in order to answer this question, I would like to consider only the results from SimpeRNN/VanillaRNN model on short vs medium vs long inputs. If we closely observe the model performance, we can notice that the model performs exceptionally well on medium sized inputs and comparatively the performs drops a little for short and long sized inputs respectively. This might be because, in case of short sized inputs, the model might not get enough information from the input sequence (as most of the values in the input vector will be zero from padding) resulting in underfitting scenario and in case of long sized inputs, the more available information might lead to overfitting scenario. But, in case of medium sized inputs, the model might receive optimal information sufficient enough for good predictions thereby avoiding both underfitting and overfitting scenarios.

Also, comparing the results on short, medium and long inputs using LSTM and GRU models – LSTM and GRU models are basically upgradation of SimpleRNN model. SimpleRNN has the problem of vanishing gradients when dealing with memory from longer input sequences. This problem is kind of addressed in LSTM and GRU by having the gating mechanism which can retain and forget needed information. So, when closely observing the LSTM and GRU model performance on short, medium and long inputs, the common thing is the both LSTM and GRU has good results for short and medium inputs (looks like gating mechanism do perform well on short inputs in comparison with simpleRNN) but for long inputs, the accuracy very slightly drops. This might be because of minor scenario of overfitting.

Note: Majorly analyzing the model performance on accuracy rather than precision and recall because the models really have very good overall precision and recall values which is hard for comparison. Accuracy has more specific values which can be compared in this scenario for fine-grain comparison between the models.

3) *What, if any, insights are gained by looking at the different evaluation approaches (i.e., precision, recall)?*

Since we are dealing with binary classification task (spam/not spam) with slightly imbalanced dataset, I believe relying only on the accuracy is not a good way of evaluating the model. So, indeed we need to observe the precision and recall not only with respect to one class (label '0' for identifying 'not spam' category), but also considering the precision and recall for other class (label '1' for identifying 'spam' category) as well. This is because, when dealing with spam classification with two categories, identifying the input text as 'spam' rather than ignoring it or not identifying it is much more important. It helps the customers to lessen their burden thereby the model fulfilling its purpose.

One more observation with respect to different evaluation approaches is – we have noticed that with respect to precision and recall metrics, both LSTM and GRU model performance is almost the same. But if we closely observe the count of True Positives, True Negatives, False Positives and False Negatives, we can notice that GRU model performs slightly better than LSTM model. This is because both precision and recall range is limited (i.e., 0 to 1 or 0 to 100%) whereas the range of True Positives, True Negatives, False Positives and False Negatives lies in the actual count of instances and these counts will be usually large (usually in thousands to millions) when dealing with the Neural networks which gives slightly in-depth information with respect to model performance. The GRU model has simplified architecture compared to LSTM which allows it to perform slightly better and faster.

Impact of Pre-trained Word Embeddings:

Methods:

Dataset:

I am using a spam classification dataset from Kaggle available at – (https://www.kaggle.com/datasets/chandramoulinaidu/spam-classification-for-basic-nlp) which has about 5796 instances in it with two categories (spam/not spam) in it. The label '1' indicates that the message is spam and '0' indicates that the message is not spam. The dataset consists of 3900 instances of 'not spam' category and 1896 instances of ‘spam’ category data in it. Even though the dataset has multiple columns in it, I am only considering the ‘MESSAGE’ and ‘CATEGORY’ columns for this Lab assignment which serves as ‘text’ input and ‘label’ output respectively for the models. As mentioned in the assignment, the split between train-test dataset is 70%-30%. So, the test set consists of 1163 instances of ‘not spam’ category and 576 instances of ‘spam’ category data in it.

Experiment:

For this experiment, we are using pre-trained word embeddings such as GLOVE and WORD2VEC which is sandwiched between the input and RNN layer respectively for two different models in the model architecture that has the following hyperparameters.

- Layers: Sequential model with following layers in the order of
  input – Pre-trained Embedding layer – RNN layer – output Dense layer
- The raw text is converted into vector form with size of 1000 before inputting it into the model.
- The Embedding layer has the dimension of vocabulary size * 100 which takes the input vectors of size 1000. This embedding layer is fixed and is not trained during the course of model execution.
- The RNN layer used is LSTM that has 64 units specified with default activation function ‘tanh’.
- The output dense layer has 1 node with ‘sigmoid’ activation function which follows the technique of predicting ‘1’ if the output layer results in the value > 0.5 and ‘0’ if the output layer results in the value <= 0.5
- The loss considered is ‘binary_crossentropy’
- The optimizer used is ‘adam’
- The metrics used for compiling the model is ‘accuracy’
- No. of training iterations used is 3 epochs (to avoid overfitting)
- No regularization incorporated.
- The models are executed on Google Colab without GPU support.

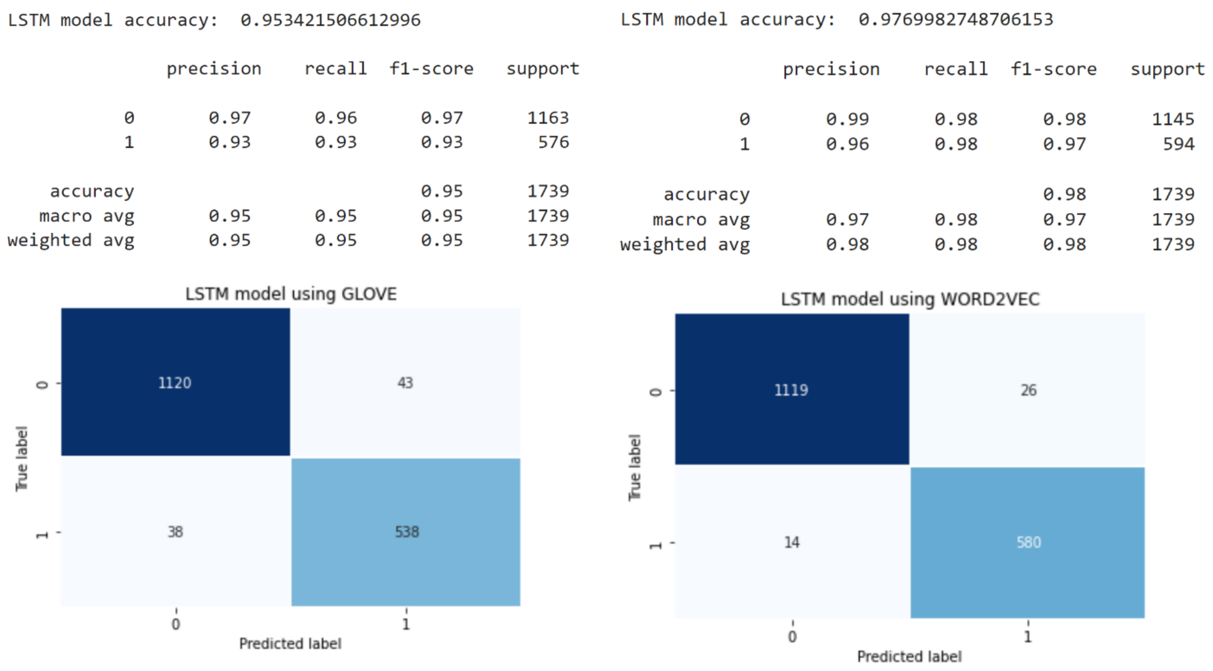
These hyperparameters are set with the values shown throughout the experiment because after experimenting for while with the different values of hyperparameters, these values resulted in good results and I thought it helps in understanding the model behaviors with respect to different model executions.

Results:

The results obtained after executing two different models with different pre-trained words embeddings are as follows:

Sl No.	*Model*	*Accuracy*	*Precision*	*Recall*	*# TP*	*# TN*	*# FP*	*# FN*
1.	GLOVE	95.34%	95%	95%	1120	538	43	38
2.	WORD2VEC	97.70%	98%	98%	1119	580	26	14

Table: Evaluations metrics observed for LSTM models with GLOVE and WORD2VEC pre-trained word embeddings on the test dataset.



```
LSTM model accuracy:  0.953421506612996

              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1163
           1       0.93      0.93      0.93       576

    accuracy                           0.95      1739
   macro avg       0.95      0.95      0.95      1739
weighted avg       0.95      0.95      0.95      1739
```

```
LSTM model accuracy:  0.9769982748706153

              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1145
           1       0.96      0.98      0.97       594

    accuracy                           0.98      1739
   macro avg       0.97      0.98      0.97      1739
weighted avg       0.98      0.98      0.98      1739
```

Fig: Model evaluation metrics for the prediction on test dataset using pre-trained word embeddings – 1) GLOVE and 2) WORD2VEC.

Analysis:

1) *Dis a certain type of word embedding lead to better results?*

   Yes, using the word2vec pre-trained word embeddings resulted in better model performance compared to the use of glove pre-trained word embeddings. This might be because the glove model is based on the co-occurrence of the words in the global context whereas the word2vec model is based on the co-occurrence of the words in the local context (neighboring words). The conclusion on this reasoning is based on the medium article found in the link - https://machinelearninginterview.com/topics/natural-language-processing/what-is-the-difference-between-word2vec-and-glove/. Since the spam messages are used in and around the similar context with respect to the subject which is more like the local context rather than global context, I believe that word2vec embeddings has slightly more edge over the glove embeddings.

2) *What, if any, insights are gained by looking at the different evaluation approaches (i.e., confusion matrix, precision, recall)?*

   Even though, the accuracy, precision and recall are slightly comparable, the confusion matrix gives more precise information about the model prediction with respect to the classes in the test dataset. By comparing the confusion matrix, we can notice that GLOVE model does predict 'not spam' instances very well but fail to predict 'spam' instances to the greater extent whereas on the other hand, if we observe the confusion matrix for WORD2VEC, we can notice that it does a great job in identifying both the 'not spam' and 'spam' instances with very low counts for False positives and False negatives. This is because, the precision, recall and accuracy give the information on the overall performance of the model whereas the confusion matrix gives the in-depth information with respect to the model performance with more fine-grain details on even the prediction of the different classes in the dataset.

3) *How would you expect these word embeddings to compare to a baseline model that uses the tokenized representation as input instead?*

   If I compare the pre-trained word embeddings with the tokenized representation of the input, I believe that the pre-trained word embeddings results in adding more value with respect to the context and the relation between the neighboring words whereas the tokenized representation of input serves as just a bag of words that are mapped to some integers. This is because the pre-trained word embeddings have multi-dimensional representation in the space in order to bring in the relation between the other words that are used in the different contexts whereas the tokenization of inputs is simply one-dimensional representation in the space.

When we compare the model performance between the use of the pre-trained word embedding and tokenized representation of words, I believe that the tokenized representation of the inputs did perform well compared to the pre-trained word embeddings. The possible reason I can think of is, when I tried to prepare the embedding matrix using the pre-trained word embeddings, there are many words that are present in the dataset vocabulary but are absent in the vocabulary of the pre-trained word embeddings. The resulting embeddings are just the zeros for those absent words. So, when we use these generalized pre-trained word embeddings, we need to make sure that the dataset used is also generic one. The more and more specific dataset used with respect to certain context, then the majority of words in the text of the dataset will be missing their respective word embeddings in the pre-trained word embedding matrix which results in the loss of the information when passing the input onto the next layers in the network resulting in the drop of the model performance.