1 Data Processing

Word Vector Representation

Since the dataset contains totally 166971 words in both training set and test set, the word embedding vector representation is required for the training, as the requirement says. Therefore, we first transfer the all the text files in the dataset into one text file, then we feed that text file into the GloVe. After training in the GloVe, a "vocab.txt" and a "vectors.txt" are created. Note that in this assignment, we use a vector sized $(1, 50)$ to represent a word.

Based on the files produced by GloVe, we first create a word dictionary into a python dictionary as format,

```
{
    str(word1) : 0,
    str(word2) : 1,
    ...
}
```

Then, we transfer every word in the dataset to an dictionary index. This immediately reduces the dataset size occupied in the RAM, which helps us train the model faster.

Similarly, we transfer the vector as a numpy array with format,

```
[[float, float, float, ..., float, float],
[float, float, float, ..., float, float],
...,]
```

with the first axis standing for the index in the word dictionary. After such processing, each word in the dataset can be transferred into a digit array with size $(1, 50)$.

Maximum Sequence Length

Since the length of each review in the dataset varies dramatically from each other, ranging from 4 words to 2470 words, we must set a limit length to filter the input to avoid feeding too many data into the model causing huge resource assumption.

A plot illustrating the distribution of file lengths is shown as Figure 1.

Clearly, most data files have length less than 400. After weighting the loss of information of data itself and the efficiency of training the model, we finally set our length limit as 300 words. So the number of kernels in RNN is 300. Vanilla RNN final state model with longer sequence length is prone to occur gradient vanishing and gradient exploding. Although mean pooling can reduce gradient vanishing but it do not work for gradient exploding. LSTM can

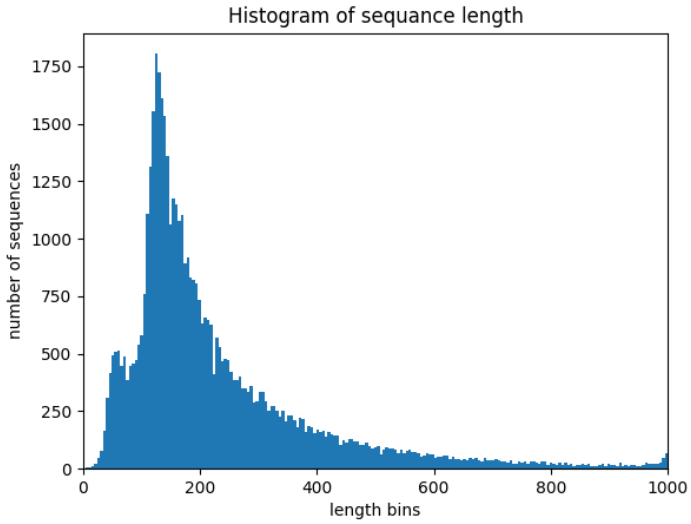


Figure 1: Distribution of dataset file lengths.

work well for dealing with the problem of gradient exploding and gradient vanishing, but LSTM may occur gradient exploding sometimes.

One thing should be mentioned is: in order to get the fixed input length, we fill the input up with zeros if the data length is less than 300, which is called sequence padding and truncate input length down to the maximum sequence length, which is called sequence truncation.

2 Modelling

First of all, the model is built with both TensorFlow API and Keras. As for the model with TensorFlow API all hyper-parameters are set as the following Table 1, For each model, we tuned several values of the hyper-parameter *learning rate decay steps* to get the stable results. Therefore, all the results we got are based on the different decay steps.

Second, in the assignment, we used one of the typical model introduced in the lecture (shown as Figure 2),

Mean pooling is a good method as the output of the model. First, since we have 300 sequence steps, the sequence length is relatively long. When doing the back propagation in final state RNN model, it is easy to occur gradient vanishing because gradient is exponentially small if the largest eigenvalue of parameter matrix W_h is less than one. If the mean pooling RNN model is applied, the prior gradient will not easily be vanishing, so the total gradient is not vanishing and mean pooling RNN model can keep more information. This is the reason why, as experiments show, in such cases, the mean pooling performs better than only extract last output as the output of model.

Table 1: hyper-parameters settings.

Parameter	Value
sequence length limit	300
batch size	1000
epochs	150
learning rate (start)	0.001
learning rate decay rate	0.96
test checking point	every 20 steps
classification function	softmax
loss function	cross entropy
optimizer	Adam

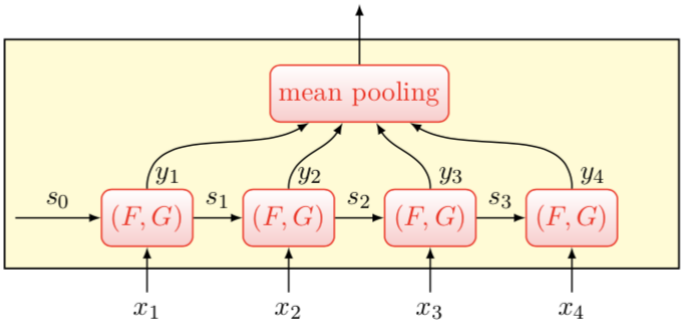


Figure 2: Mean pool of the output.

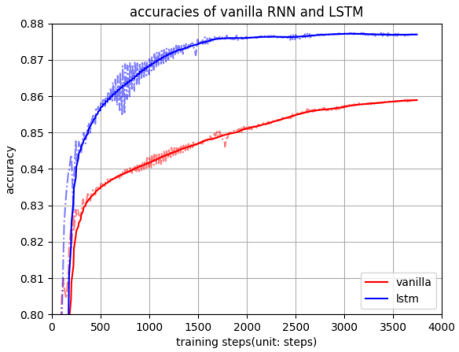
3 Results

In this session, we will go first into the comparison of the performances of different models, then we will show the results of the influence of the model complexities.

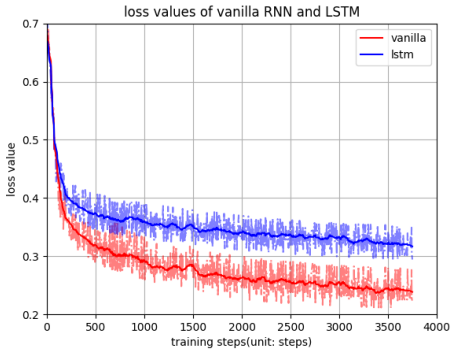
Vanilla RNN and LSTM

The comparison of Vanilla RNN and LSTM are shown as Figure 3. below.

For a brief conclusion, it is obvious that the LSTM outperforms the Vanilla RNN, although the loss value of the LSTM is generally higher than the Vanilla. The main reason is that the LSTM contains more parameters which amplify the loss value.



(a) Accuracy of Vanilla RNN and LSTM with state dimension 50.



(b) Loss value of Vanilla RNN and LSTM with state dimension 50.

Figure 3: Compare Vanilla RNN and LSTM.

Influence of the State Dimension

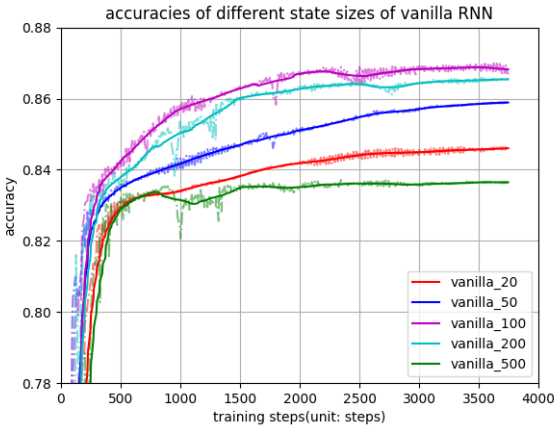
First of all, let us use some graphs(see Figure 4.) to illustrate the overall influence of the state dimension. These graphs are tuned well to avoid gradient vanishing and exploding.

Apparently, increasing the state dimension does not always improve the training speed and final test results. On the contrary, the performance goes up from dimension 20 to 50 at the beginning. Then, it starts showing that further increase of the dimension may worsen the model performance.

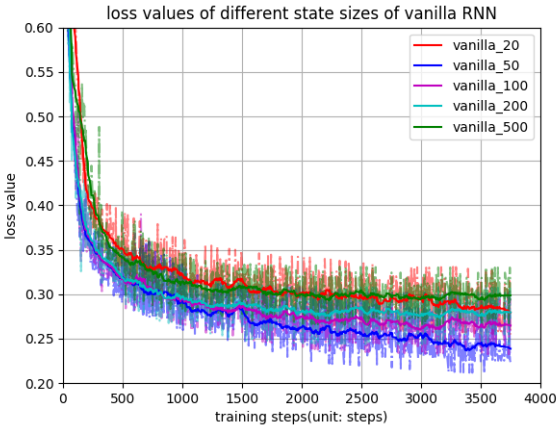
In order to find further influence of the state dimension on the model, also in order to meet the requirements of our assignment, a table is built showing the final stable results of models(see Table 2).

Table 2: Final results.

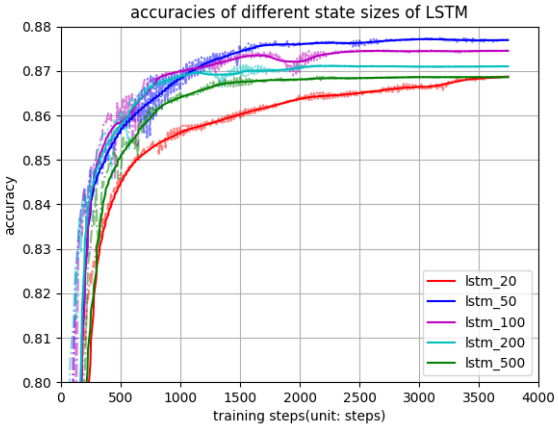
Metrics Comparisons		
Model	*Vanilla RNN*	*LSTM*
Metrics	*accuracy*	*accuracy*
State Size 20	84.56%	86.86%
State Size 50	85.91%	87.71%
State Size 100	86.67%	87.44%
State Size 200	86.54%	87.10%
State Size 500	83.65%	86.86%



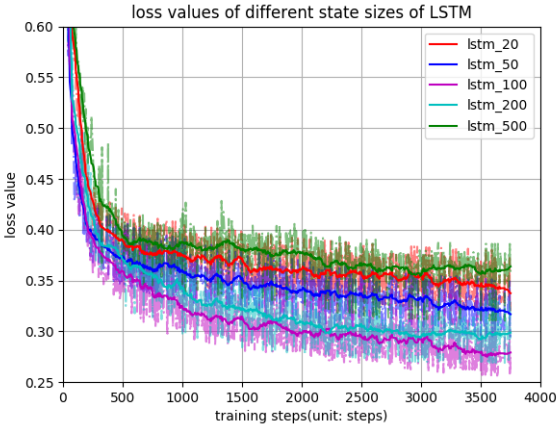
(a) Results on accuracy of Vanilla RNN.



(b) Results on loss of Vanilla RNN.



(c) Results on accuracy of LSTM.

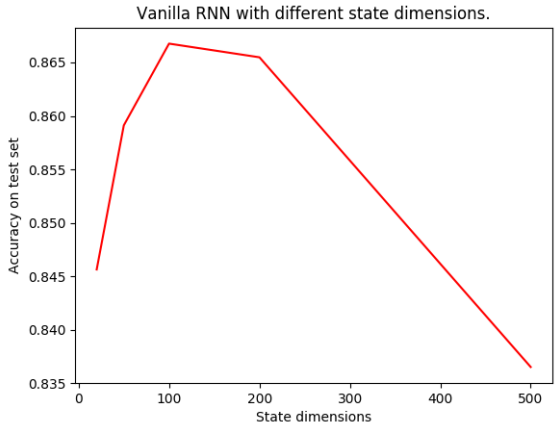


(d) Results on loss of LSTM.

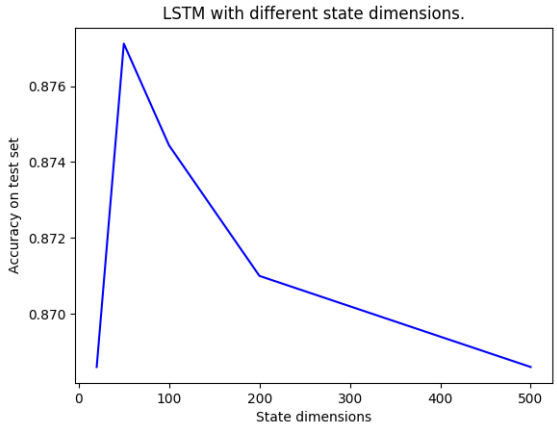
Figure 4: Influence of state dimensions on 2 models.

A figure 5 is also produced in order to make the results more easily to be visualized.

By reviewing all the results above, one can conclude that: there exists an optimal solution on the state dimensions of RNN families when given a specific dataset. Other state dimensions on both side of the optimal one may not perform as well as the optimal solution. Therefore, in order to choose the best solution for state dimensions, one may conduct several experiments on the specific RNN model.



(a) State dimensions on Vanilla RNN.

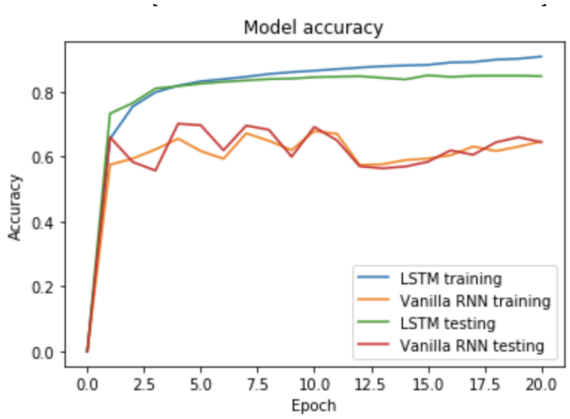


(b) State dimensions on LSTM.

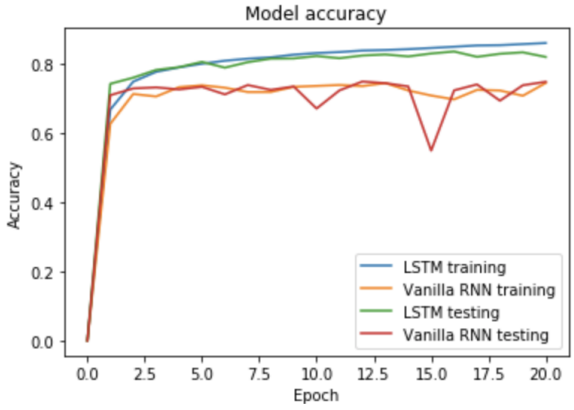
Figure 5: Influence of state dimensions on 2 models' accuracy.

4 Some experiments about different RNN models

To understand RNN models deeply, we have done several experiments without tuning to find some interesting conclusions.



(a) Comparing Vanilla RNN and LSTM based on final state.



(b) Comparing Vanilla RNN and LSTM based on mean pooling.

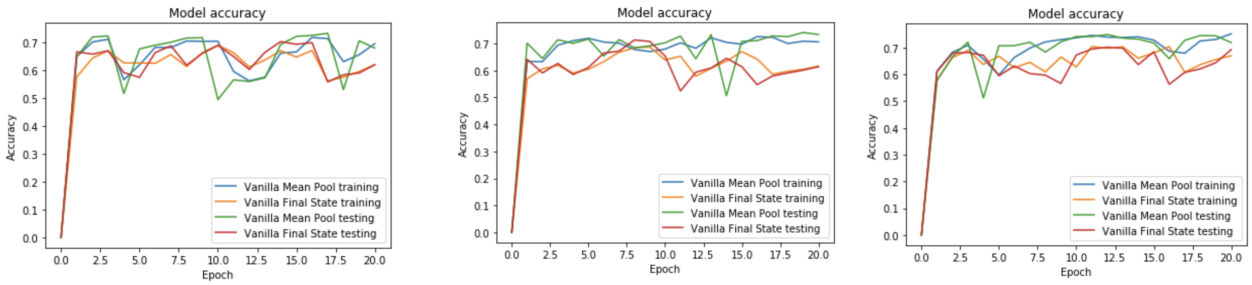
Figure 6: Comparing Vanilla RNN and LSTM based on extracting mean pooling and final state.

By comparing Vanilla RNN and LSTM based on extracting mean pooling and final state(see Figure 6), we see both extracting mean pooling and final state in Vanilla RNN are all gradient exploding sometimes, which is the disadvantage of Vanilla RNN, and LSTM can fix this problem up well. LSTM is more complex than Vanilla RNN and it seldom occur gradient

exploding and vanishing, so its accuracy is higher than that of Vanilla RNN.

In these graphs, sometimes testing accuracy is higher than training accuracy. There are several reasons to interpret this phenomenon. First reason is training accuracy is measured during each epoch while testing accuracy is measured after each epoch. Another reason is the validation set maybe easier than the training set or their data distribution are not same. This happens when testing set is smaller than training set or was not properly sampled because these experiments' testing data worked as validation data(smaller than training dataset).Training set was divided into training data and validation data to save training time in these experiments. Another reason is data leakage happens, meaning training samples getting accidentally mixed in with validation samples.

The accuracy of mean pooling Vanilla RNN is higher than the one of final state. One reasonable way to illustrate this phenomenon is mean pooling can keep more information than extracting final state and mean pooling can reduce gradient vanishing.



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3

Figure 7: Comparing mean pooling and final state in vanilla RNN.

To verify this phenomenon, three experiments are done(see Figure 7). If less gradient exploding happening, extracting mean pooling can perform well than final state.

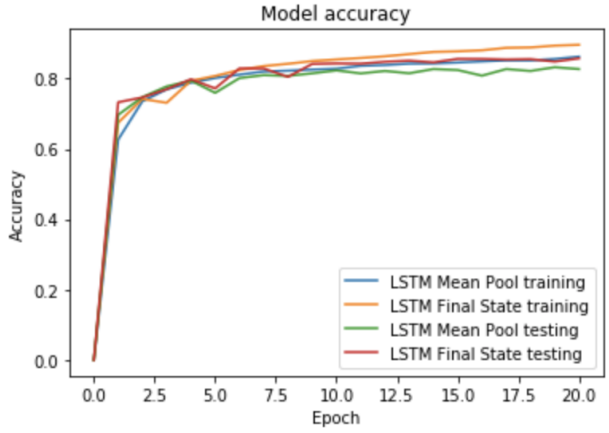
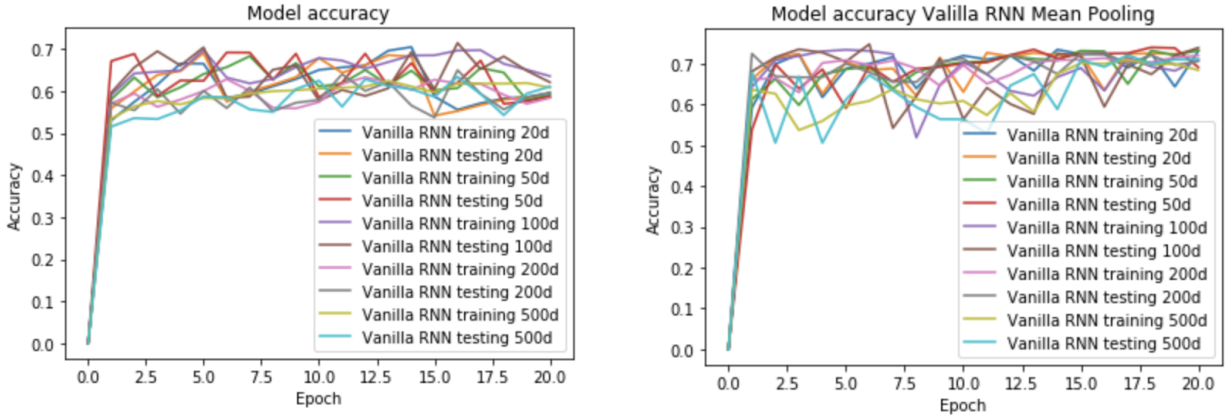


Figure 8: Comparing final state and mean pooling in LSTM

In addition, by comparing final state and mean pooling in LSTM, their accuracy are nearly same, but the accuracy of final state LSTM is slightly higher than the one of mean pooling(see Figure 8). This is possibly because mean pooling extracting too much information than needed.



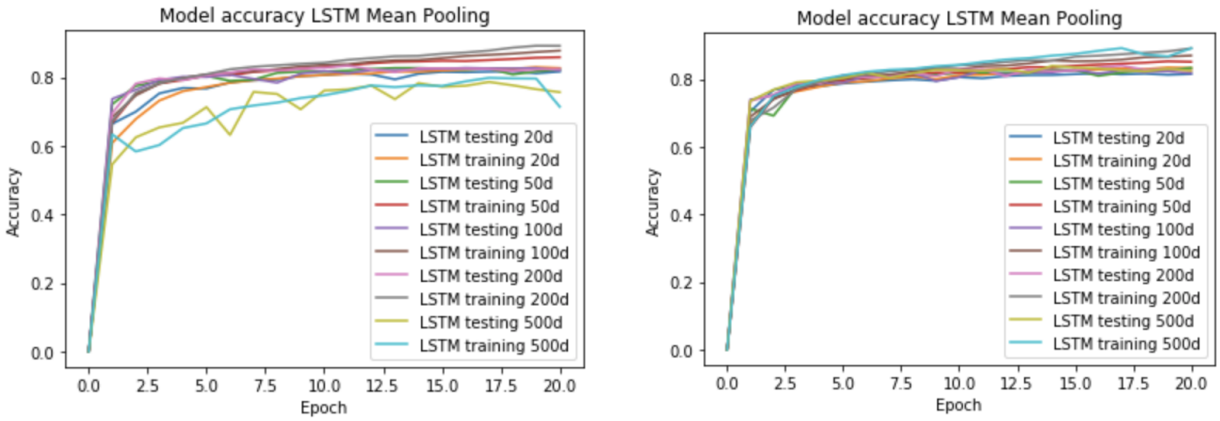
(a) Extracting final state.

(b) Extracting mean pooling.

Figure 9: Comparing different state dimension in Vanilla RNN

Since gradient exploding happens frequently in Vanilla RNN, different state dimension comparison in Vanilla RNN whether using meal pooling or final state extraction could not access easily without tuning hyper-parameters(See Figure 9).

Vanilla RNN with extracting final state is harder to tune than mean pooling, so tuned comparison model with mean pooling Vanilla RNN is shown in Figure 4 (a).



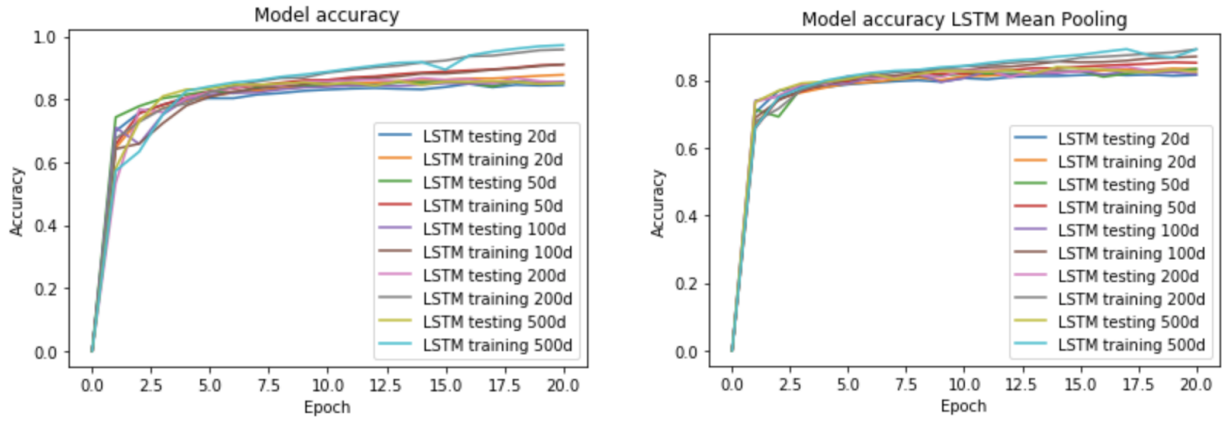
(a) LSTM with gradient exploding

(b) LSTM without gradient exploding

Figure 10: Comparing different state dimension in mean pooling LSTM

LSTM may occur gradient exploding too, although the possibility is extremely small. In

Figure 10(a), gradient exploding happens in 500 state dimension, when it trained again, gradient exploding vanished(see Figure 10(b)).



(a) final state LSTM

(b) mean pooling LSTM

Figure 11: Comparing different state dimension in two LSTM models

By comparing different state dimension both in final state LSTM and mean pooling LSTM(see Figure 11), it is shown that the model occur overfitting too much and the training accuracy increase of model with state dimension climbing. This is because when state dimension increasing, more parameters should be learned, the model become more complex, and training accuracy increases as well as overfitting occurs finally.