Ao Zhang
0300039680

# Assignment 3

## 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.

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.

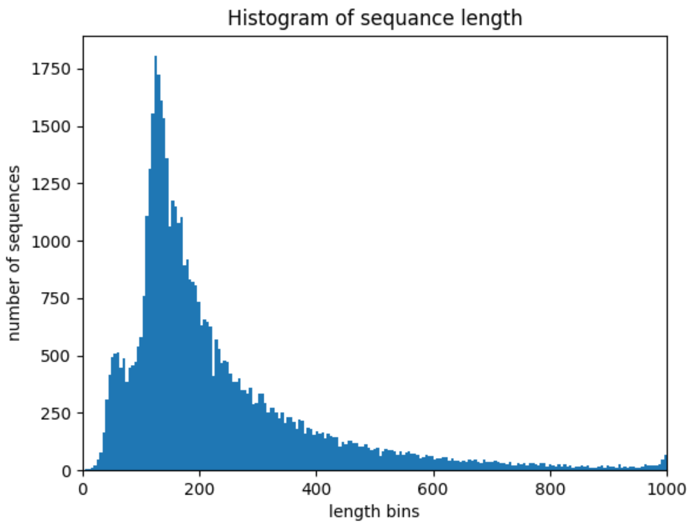


Figure 1: Distribution of dataset file lengths.

## 2 Modelling

First of all, the model is built 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*

Table 1: hyper-parameters settings.

| Parameter | Value |
|---|---|
| sequence length limit | 500 |
| batch size | 1000 |
| epoches | 150 |
| learning rate (start) | 0.001 |
| learning rate decay rate | 0.96 |
| test checking point | every 20 steps |
| gating function | softmax |
| loss function | cross entropy |
| optimizer | Adam |

*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), There are two reasons why we choose mean pooling as the output of the model. First, since we have 300 sequence steps, the order of the model parameters will

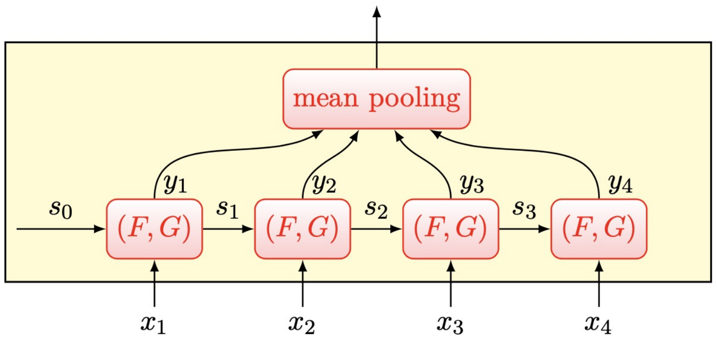


Figure 2: Mean pool of the output.

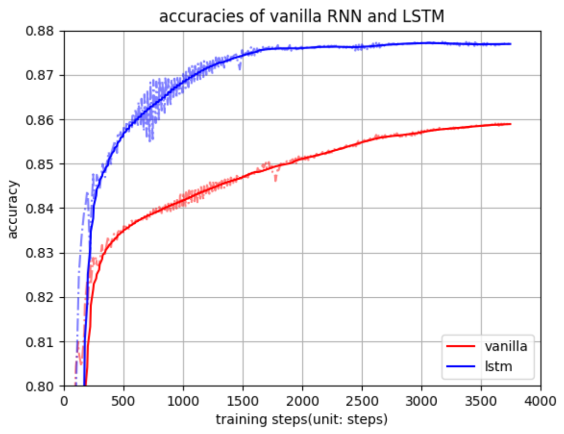
become extremely high, which is totally not the best case for back propagation. Second, as experiments show, in such cases, the mean pooling performs better than only extract last output as the output of model.

## 3 Results

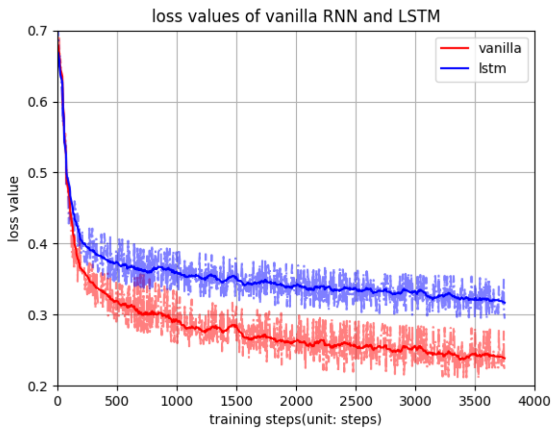
In this session, we will go first into the comparison of the performances of different models, then we will show the results of the influence of the model complexities.

### Vanilla RNN and LSTM

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



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

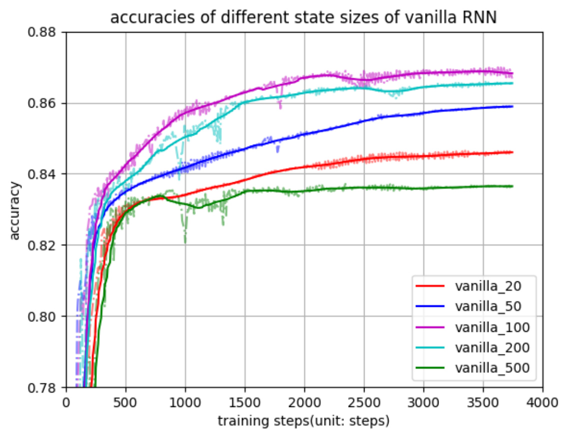


(b) Loss value of Vanilla RNN and LSTM with state dimension 50.

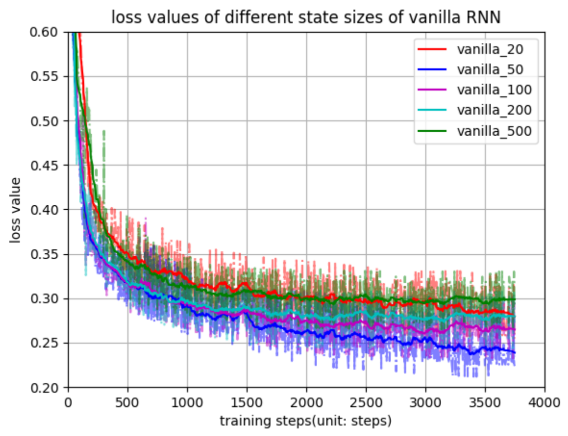
For a brief conclusion, it is obvious that the LSTM outperforms the Vanilla RNN, although the loss value of the LSTM is generally higher than the Vanilla. The main reason is that the LSTM contains more parameters which amplify the loss value.

## Influence of the State Dimension

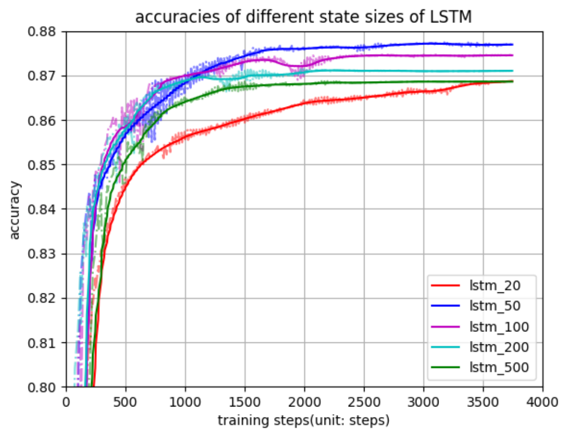
First of all, let us use some graphs to illustrate the overall influence of the state dimension.



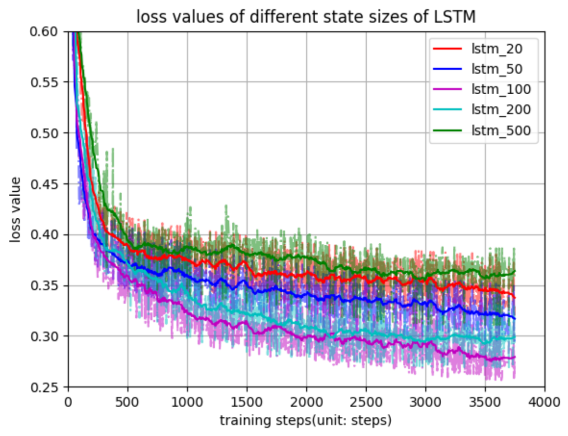
(a) Results on accuracy of Vanilla RNN.



(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.

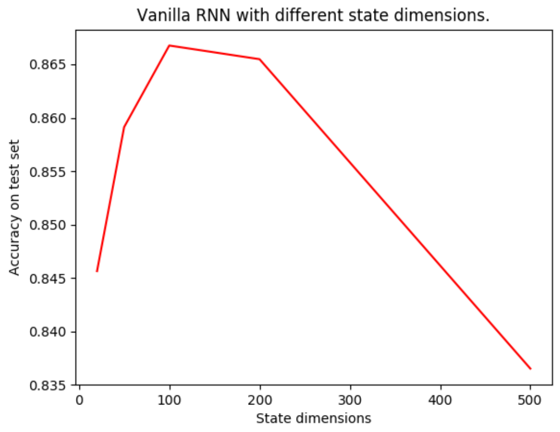
Apparently, increasing the state dimension does not always improve the training speed and final test results. On the contrary, the performance goes up from dimension 20 to 50 at the beginning. Then, it starts showing that further increase of the dimension may worsen the model performance.

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.

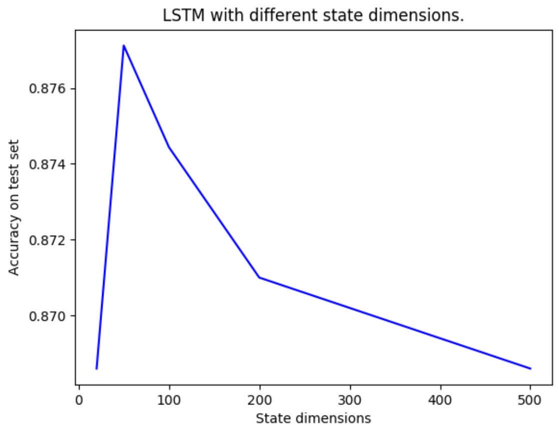
Table 2: Final results.

| **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 figure is also produced in order to make the results more easily to be visualized.



(a) State dimensions on Vanilla RNN.



(b) State dimensions on LSTM.

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.