Yansong Li,300083962
Xuanyu Su,300083967

Assignment 3

CSI5138 Fall
October 24, 2019

1 Data preprocess

1. Create TextContent list for all movie reviews and extract original text data from data files by using os.open() and encapsulate the review into list element.

2. Extract rating scores for each specific review from their corresponding file name and append rating score into TextRating list.

$$TextContent = [review1, review2, ...reviewn]$$

$$TextRating = [rating1, rating2, ...ratingn]$$

3. Convert numeric rating element $[1 : 9]$ into binary data $[0, 1]$ by setting up filter based on $thresholdscore =' 7'$ which is the lower bound of positive scores or upper bound of negative scores.

4. Initialize Tokenizer function as our text processing model which collects 99475 unique words from TextContent list and establish dictionary to save all words, each word in the dictionary is assigned by an unique index.

$$Dictionaries = [word1, word2, word3, ...word99475]$$

5. Return the corresponding index from dictionary for each word in our original sentences.

$$Exp : Sentence1 = [index\,of\,word1, index\,of\,word2, ...index\,of\,wordN]$$

6. In order to organize our data into same Tensor shape for the model training step, we normalize our sentences into same 'MAX SEQUENCE LENGTH' according to the distribution of word length upon the whole data set.

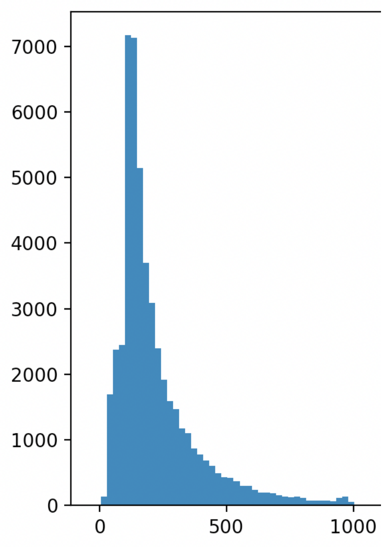

figure 1: Sentence length distribution

7. According to the data distribution, the majority length of words is 500, hence, MAX SEQUENCE LENGTH = 500.

8. Padding the words with '0' for the words who don't have length of 500.

9. Import glove file as our embedding vector and build an embedding vector dictionary. For each word in vector dictionary, the words in original text sentences would find their corresponding embedding vectors.

2 Model processs

1. Initialize the model with adding placeholder for the training data and corresponding labels.

2. Using tf.nn.embedding lookup function to find the embedding vector for each input data.

3. Initialize bias and weight with random variables.

4. Construct Vanilla RNN/LSTM basic cell with key parameter state dimension.

5. Generate output and state by using tf.nn.dynamic rnn which would automatically run through all states in the same RNN layer.

6. Load the mean value for the outputs generated from step 5 and implement Softmax function in the last layer.

7. Applying following parameters to iteratively run the model based on batch size and epoch.

Max sequence length	batch size	epoch	learning rate	dropout	optimizer
500	200	10	0.01	0.5	Adam

2.1 Vanilla RNN

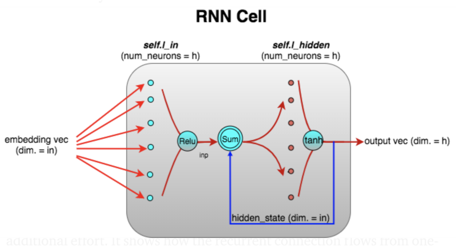

figure 2: RNN cell model.

The RNN mdoel's performance was extremely unstable and pretty low on the final result. The model didn't generalize well with $epoch = 10, learningrate = 0.01, dropoutrate = 0.5$, The result might be caused by the 'Short term memory of RNN', since all RNNs have feedback loops in the recurrent layer. This lets them maintain information in 'memory' over time. But, it can be difficult to train standard RNNs to solve problems that requires learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time (called the vanishing gradient problem). LSTM networks are a type of RNN that uses special units in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for a long period of time. What's more, insufficient epoch number is another important factor which might cause this problem. However, our laptop don't support GPU for massive calculation for long time running, especially, when we set state dimensions equal to 200 or higher.

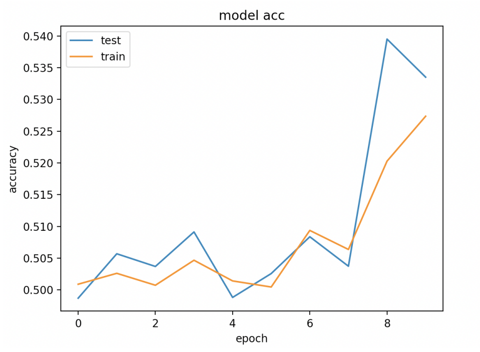

figure 3: State size = 20.

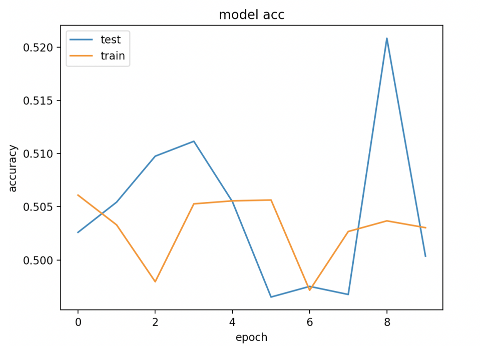

figure 4: State size = 50.

State dimension	Training accuracy	Test accuracy	Training loss	Test loss
20	0.53196	0.52812	0.69164	0.69308
50	0.50141	0.51102	0.6962	0.6981
100	0.50011	0.49102	0.7061	0.7171

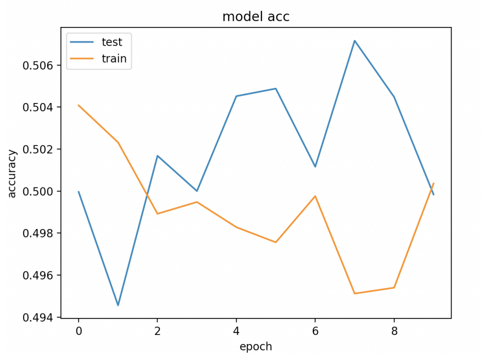

figure 5: State size = 100.

With the increment of state dimensions, the fluctuation of training and test accuracy is sharply increased, We tried implement Vanilla RNN with both Keras API and Tensorflow, however, each technique returns the same abnormal fluctuation. The data preprocessing part has no problem since we got optimal performance by using LSTM model with the same initialization method. Therefore, we concluded that RNN model need more than 20 epochs to stabilize generalization ability. And with the increment of state dimension, RNN's short term memory shortage would severely affect model robustness, which would massively increase calculation cost. Hence, we stop trying test RNN with state dimension of 200 and 500. Sorry for the experiment condition limitation.

2.2 LSTM

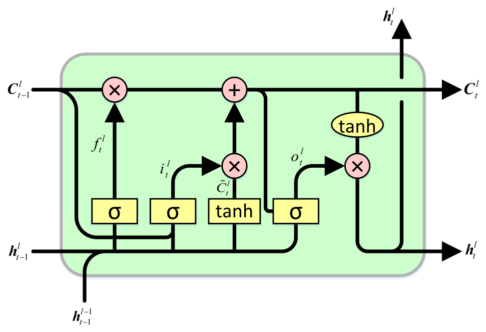

figure 6: The caption of this figure.

The overall performance of LSTM is greater than RNN, since LSTM(Long Short Term Memory) deals with memory forgotten problems by introducing new gates, such as input and forget gates, which allows for a better control over the gradient flow and enable better preservation of "long-range dependencies".

Batch size	Training accuracy	Test accuracy	Training loss	Test loss
20	0.74401	0.68102	0.6132	0.6283
50	0.86141	0.82102	0.3812	0.3935
100	0.8789	0.8433	0.3710	0.3821
200	0.7628	0.6684	0.5428	0.5427
500	0.7401	0.67001	0.6032	0.6283

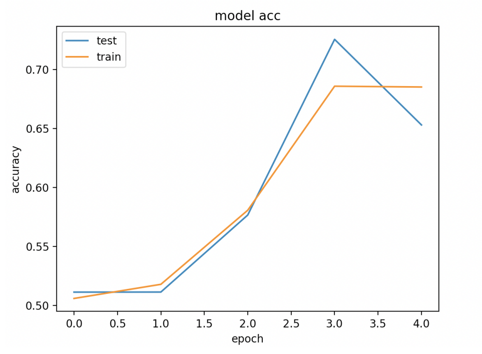

figure 7: State size = 20.

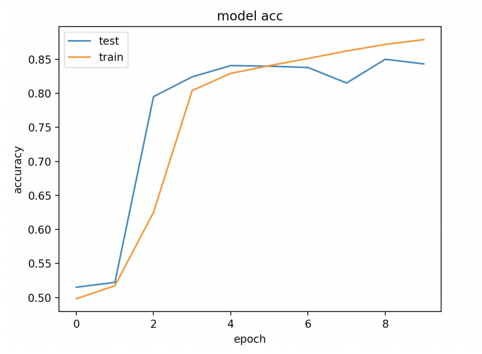

figure 8: State size = 50.

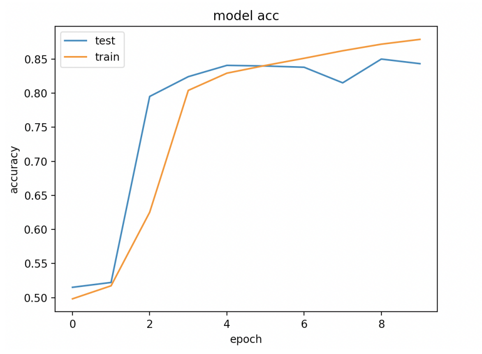

figure 9: State size = 100.

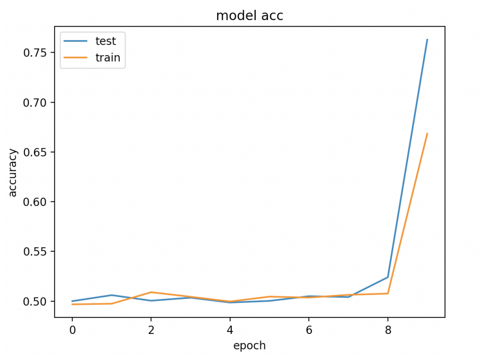

figure 10: State size = 200.

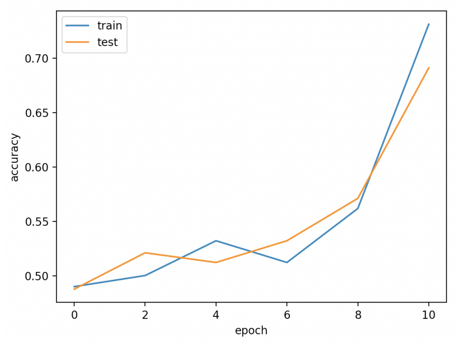

figure 11: State size = 500.

According to the result, we observed that the optimal state dimension for LSTM is 100, since the model got highest accuracy when $statesize = 100$,The following graph shows how accuracy changes with adjustment of state dimension.

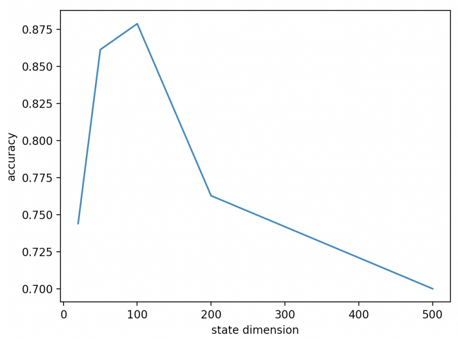

figure 12: The changing flow of LSTM

3 Comparison and discussion

Vanilla RNN: The performance of Vanilla RNN is not optimal since the plot of accuracy curve has **massive fluctuation** with the changes of epochs and the overall accuracy is extremely low.
The possible cause is that **short term memory** of Vanilla RNN model need more **epochs** to stabilize the generalization ability. However, according to the result observation, with the increasing of state dimension, the entropy of model loss would exponentially increase. Finding optimal parameters for Vanilla would have expensive time cost and calculation cost, which is hard for our laptop without using extra **GPU**.
LSTM: The overall performance of LSTM is better than Vanilla RNN, since LSTM has both **cell states** and a **hidden state**. The cell state has the ability to **remove** or **add** information to the cell, regulated by "forgotten gates". According to the result, the optimal state dimension for LSTM is 100.