COMP6714 Project Report

Fu ZHENG | z3369444
Kuan-Chun Hwang | z5175539

Task 1

The evaluate function is used to compute the F1 score which is based on precision and recall to see the accuracy of the model. The implementation iterates through the golden list, which is the ground truth) and predict list and counts the number of true positives, false positives and false negatives.

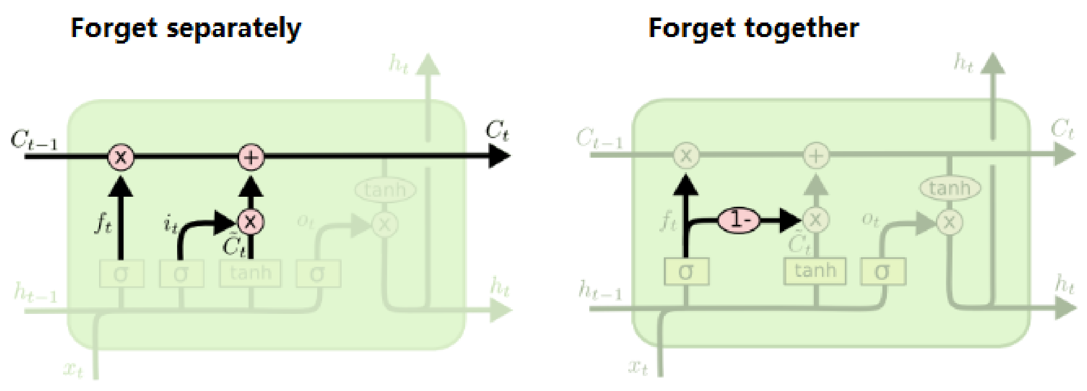
Since the length of the two lists are the instances inside them are the same, we can safely iterate through them without handling alignment issues. The implementation iterates through each i-th instance in both list simultaneously and perform classify them as a true positive or false positive or false negative when we see a ‘B’ tag. If we detect a tag that indicates beginning of an entity, which is a tag that contains ‘B’, in the predict list, but the element in golden list is different, then we have found a false positive since we have predicted true but the ground truth does not match. Furthermore, if the current element in golden list is one of the entity beginning tag, which is a tag that contains ‘B’, there are two cases we will need to handle. Firstly, if the ground truth doesn’t equal to the predicted, we have found a false negative because we know that the ground truth is one of the tag with ‘B’ but the prediction has either classified it with a different ‘B’ tag, or one of the other tag without ‘B’. This indicate that we have predicted false when it was actually true, resulting in a false negative. Otherwise, we will need to check the remaining list to check the rest. An edge case for this scenario is that we are already at the end of a list and in this case, we have found a true positive since we know the current element in predict list and golden list are the same and there is nothing else to check. Otherwise, there are two cases where true positive occurs. If the current element in golden list and predict list are both not an inside tag, which are tags that contains ‘I’, then we have found a true positive as it indicates we have detected another new entity without having the current entity being classified as a false negative or false positive. The other case is simply an edge case where if we reach the end of the list and the last element in both predict list and golden list are equal, then it is a true positive. However, an entity can be classified as a false positive or false negative before it reaches the next entity. False negative and false positive counts are update simultaneously in these cases. These cases occurs when we have detect that the current element in predict list is different from the current element in golden list.

After finding out the total number of true positives, false negatives, and false positive, we simply input those variables into the F1 formula, which is (2*TP)/(2*TP + FN + FP) to calculate the F1 score. Overall, the time complexity of the evaluation algorithm is O(n^2)

The overall effect of modification 1 has little effect on the running performance in the training process based on the training data given even though the evaluation function has time complexity of O(n^2). However, the effect on performance can be more significant if an even larger data set is given such that the training time increase is more evident.

Task 2

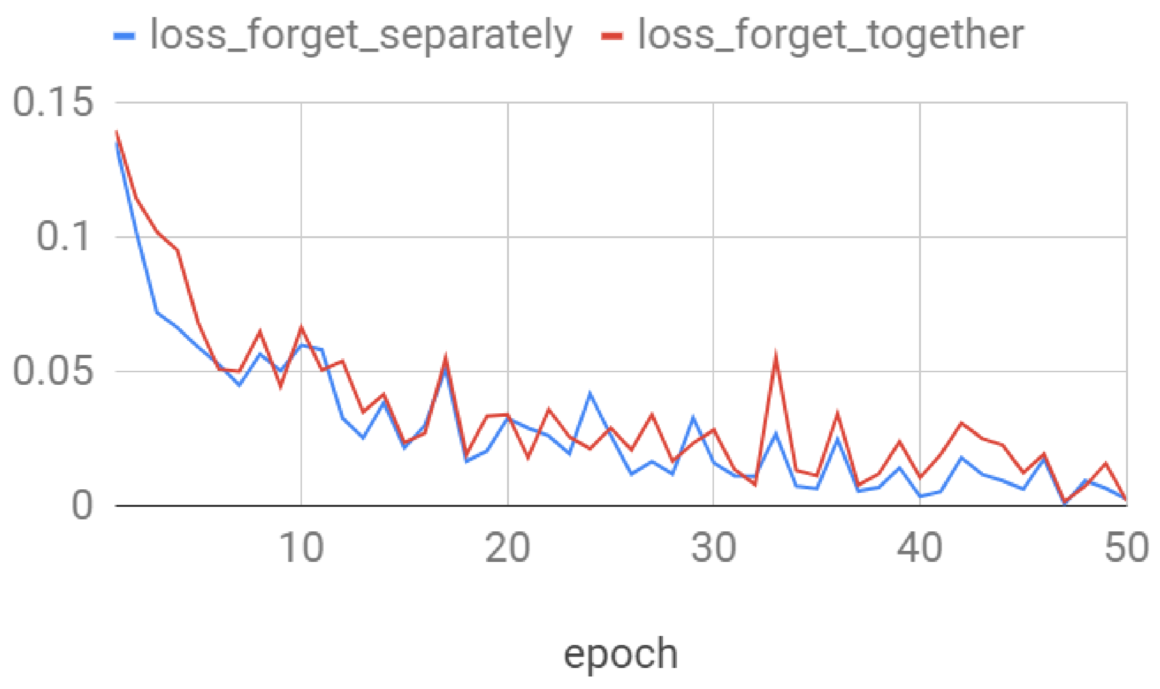
In this task, instead of separately deciding what to forget and what the model should add new information to, the method will make all the decisions together. On the other words, it will only forget when it is going to input something in its place. Also, only input new values to the state when it forget something older. The following chart illustrate the changes on the model (left to right).



$$C_t = f_t * C_{t-1} + \boxed{i_t} * \tilde{C}_t \qquad C_t = f_t * C_{t-1} + \boxed{(1 - f_t)} * \tilde{C}_t$$

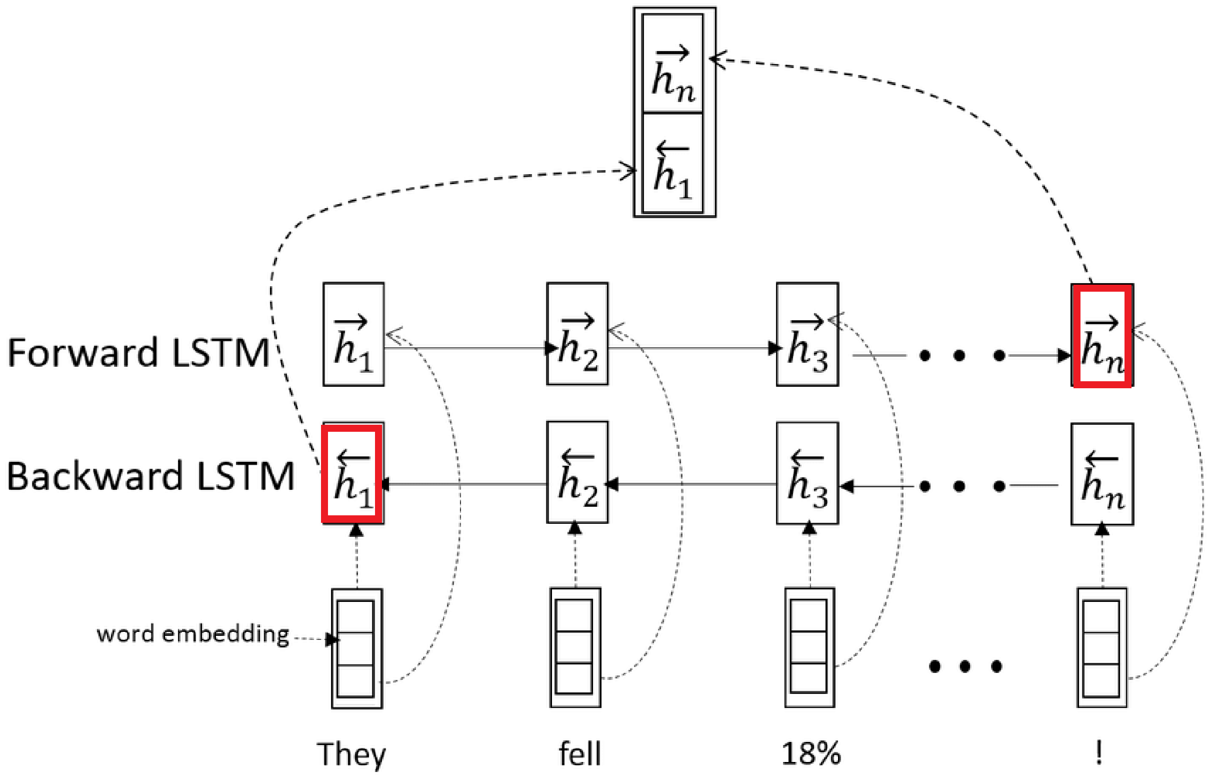
Output results comparison:

The following chart shows the training loss with different LSTM cell setting. As the chart indicated, the forget together method (red) is performing poorer than forget separate model (blue)



Task 3

In this task, the key idea is to train the LSTM model in both forward and backward directions with input characters. In particular, the model is implemented by concatenating the last output of the forward layer and the first output of the backward layer (image shown below).



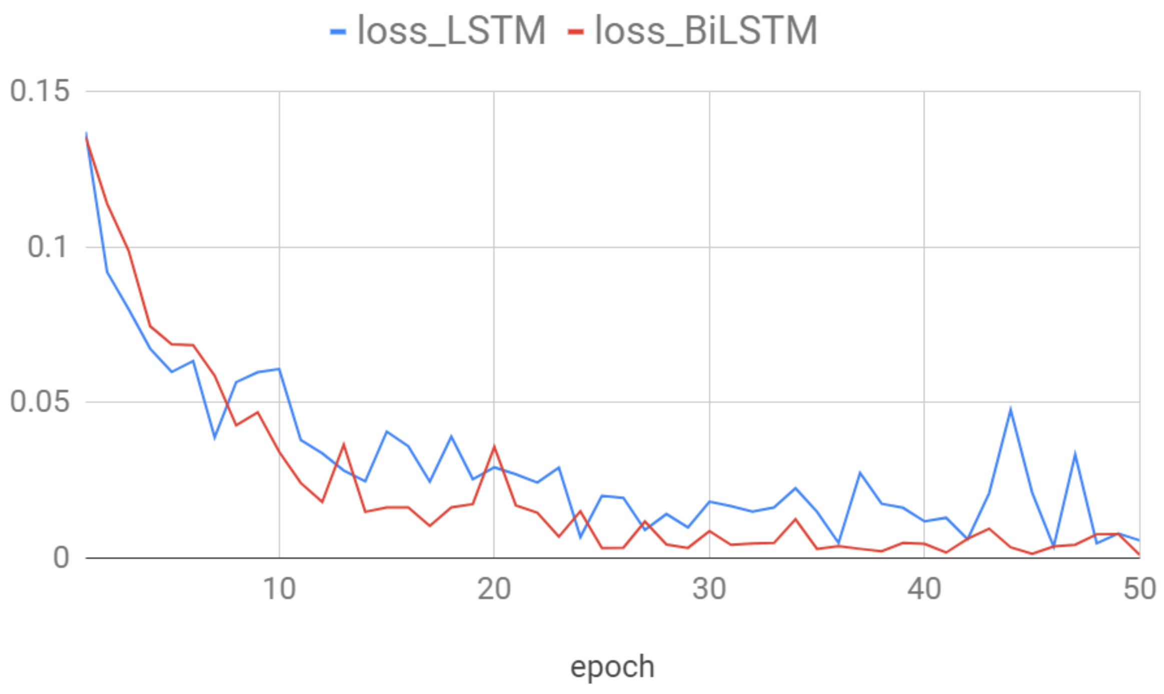
Following are the detail of each step:

1. The batch_char_index_matrices (input) has 3 dimensions which include the number of sentence of the batch, the number of word in each sentence and the number of characters in each word. E.g. [2, 7, 14]. By joining all the sentences together, we will have a 2D mini batch (e.g. [14, 14]). Which represent all 70 words with maximum 14 characters in each of them.
2. Apply the predefined char_embeds() method from the BiLSTM model to the mini batch will give us a final tensor of shape [14, 14, 50], where the 50 represents the word embedding value of each character.
3. The batch_word_len_lists (input) is a 2D matrix which contain the word length in each sentence, since we have joining all the sentences together, we will need to reshape this input into a 1D array by simply apply view(-1) to it.
4. Since out model is built with Tensorflow and RNN, it require a descending sorting on the length of each word. Hence the sort_input() method is called here. As a output, we receive both sorted matrix and the sorted index, perm_idx from the original index.
5. We then generate the desorted_indices from perm_idx for reverse the sorted matrix to original matrix in the later step.
6. Before the training take places, we use pack_padded_sequence() method to pack all the input embedded words to the same max length.
7. The char_lstm training will generate both the output and hidden state from the input data. Here, the hidden state is a tuple which contains the h_n 3D tensor and c_n 3D tensor.

8. Since we only interesting in the h_n tensor (indicated red in above image), we will concatenate the last layer and the last layer of h_n[0]
9. Applying the desorted_indices from step 5 to the concatenated result, we could get back the original matrix.
10. Reshape the above result to a sentence structure matrix.

Output results comparison:

By running the training with and without BiLSTM, the chart below shows that BiLSTM has significant lower training loss to single direction LSTM. Especially after 10 epoch.



Reference:

1. Quick Recap, Understanding Bidirectional RNN in PyTorch, Towards Data Science, by Ceshine Lee, accessd in 5/10/2018
https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66
2. Step-by-Step LSTM Walk Through, Understanding LSTM Networks, Colah's blog, accessed in 15/10/2018
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
3. LSTM's in Pytorch, Sequence Models and Long-Short Term Memory Networks, PyTorch 1.00 documents, accessed in 7/10/2018
https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html