Report

Student id: z5192086 Student name: Pan Luo

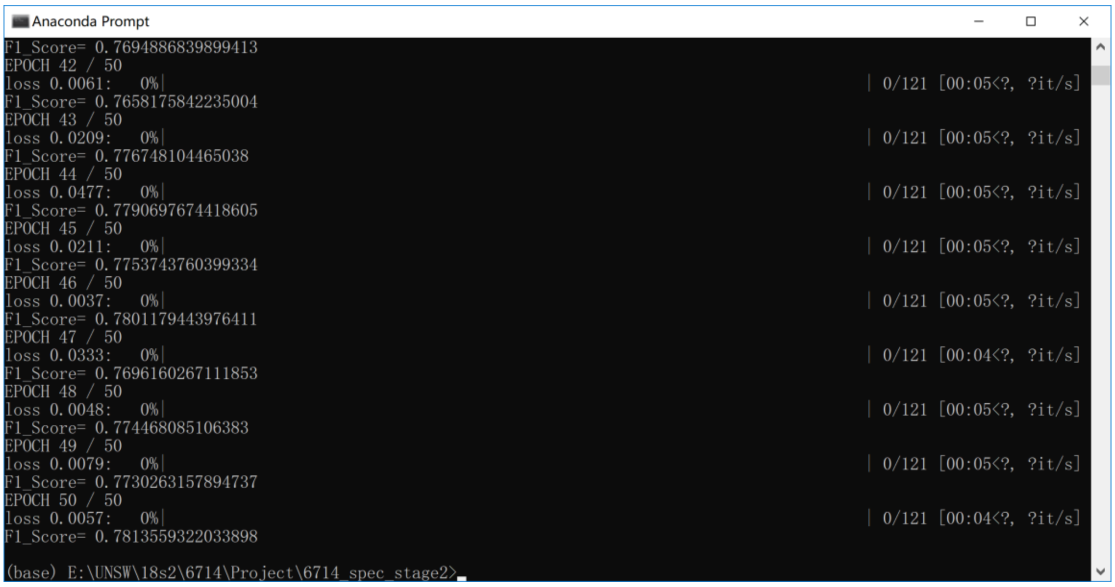
1.How do you implement evaluate()?

The evaluate() function aims at calculating F1 score. The F1 score is the harmonic mean of precision and recall. So in order to calculate F1, I should calculate precision and recall first. Precision=tp/(tp+fp), Recall=tp/(tp+fn), and the task is to match the label in two lists. According to the definition of precision and recall, first I should find tp, fp and fn in these two list.

According to the IOB2 tagging rules, "I" can only be followed by "B". In order to calculate the matching label, both the label's boundaries and the tags are the same. So the algorithm I use is building two cursor. First, traverse the predict_list, and if there are tags which equal to 'B-TAR' and 'B-HYP' in predict_list but they are not equal to golden_list at the same location, then these tags are fp. Then in golden list, move the cursor to next "B" element. If the elements in predict list at the same location is not equal to golden_list, then these tags are fn. In order to find tp, there are different conditions. First, I judge whether the last element of two tags are 'B' elements, if they are and the same, so they are tp. Then during the elements in two lists, if the element after the golden_list cursor is not 'I' and at the same location in predict_list the element after the curson is not 'I' , then these tags are tp. Unless they are fn and fp in two lists.

2.How does Modification 1 (i.e., storing model with best performance on the development set) affect the performance?

The modification of evaluate() function will calculate the F1 score in each epoch. And according to train.py, the model will judge whether the new F1 score is greater than old one, if yes, then the model will store the new F1 score and always keep on this F1 score on development set. As is shown below, the F1 score goes up at each epoch.



```
F1_Score= 0.7694886839899413
EPOCH 42 / 50
loss 0.0061:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.7658175842235004
EPOCH 43 / 50
loss 0.0209:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.776748104465038
EPOCH 44 / 50
loss 0.0477:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.7790697674418605
EPOCH 45 / 50
loss 0.0211:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.7753743760399334
EPOCH 46 / 50
loss 0.0037:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.7801179443976411
EPOCH 47 / 50
loss 0.0333:   0%|                                                        | 0/121 [00:04<?, ?it/s]
F1_Score= 0.7696160267111853
EPOCH 48 / 50
loss 0.0048:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.774468085106383
EPOCH 49 / 50
loss 0.0079:   0%|                                                        | 0/121 [00:05<?, ?it/s]
F1_Score= 0.7730263157894737
EPOCH 50 / 50
loss 0.0057:   0%|                                                        | 0/121 [00:04<?, ?it/s]
F1_Score= 0.7813559322033898

(base) E:\UNSW\18s2\6714\Project\6714_spec_stage2>
```

3.How do you implement new_LSTMCell()?

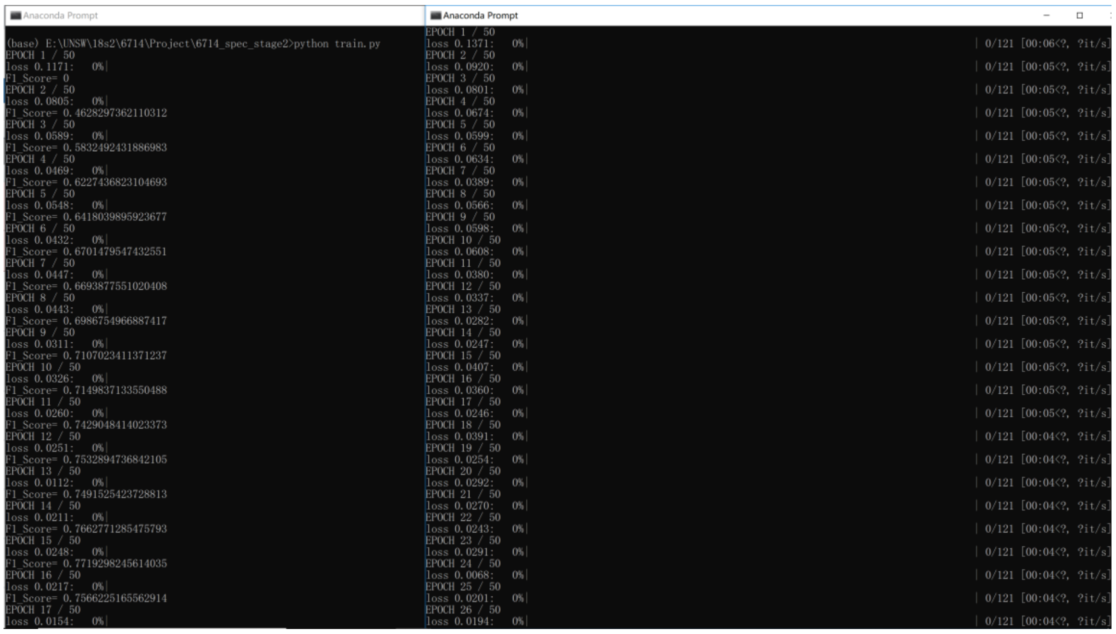
First, I read the basic LSTM model and understand the implementation of function called torch.nn._functions.rnn.LSTMCell(). The original function define input gate, forget gate and output gate. In these three gate, they use different activation function to deal with the input (X_t,h_{t-1}). Then the LSTM cell use tanh to update cells by forgetting and inputting some message. According to the requirements about new_LSTMCell() function, what I should change is the method of cell updating. The original LSTM use $C_t = f_t * C_{t-1} + i_t * c_t^{new}$, and the new LSTM use $C_t=f_t*C_{t-1}+(1-f_t)*C_t^{new}$. So I change the code from

cy = (forgetgate * cx) + (ingate * cellgate) to

cy = (forgetgate * cx) + ((1 - forgetgate) * cellgate)

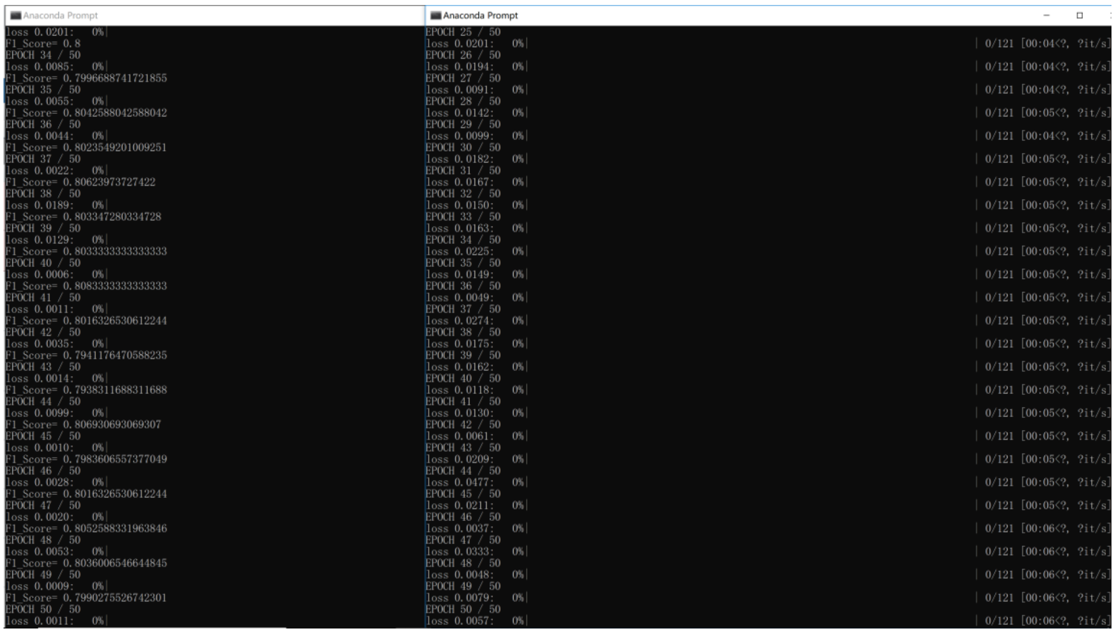
4.How does Modification 2 (i.e., re-implemented LSTM cell) affect the performance?

The modification of re-implemented LSTM cell change the learning behavior of LSTM. The original LSTM forget and input separately, but after modification, new LSTM forget and input together. So only when there is new message input then the LSTM begin to forget, and vise versa. In this way, as for performance, I do the comparison of loss at the same epoch and the speed of each epoch. As is shown below:



```
(base) E:\UNSW\18s2\6714\Project\6714_spec_stage2>python train.py
EPOCH 1 / 50
loss 0.1171:   0%|
F1_Score= 0
EPOCH 2 / 50
loss 0.0805:   0%|
F1_Score= 0.4628297362110312
EPOCH 3 / 50
loss 0.0589:   0%|
F1_Score= 0.5832492431886983
EPOCH 4 / 50
loss 0.0469:   0%|
F1_Score= 0.6227436823104693
EPOCH 5 / 50
loss 0.0548:   0%|
F1_Score= 0.6418039895923677
EPOCH 6 / 50
loss 0.0432:   0%|
F1_Score= 0.6701479547432551
EPOCH 7 / 50
loss 0.0447:   0%|
F1_Score= 0.6693877551020408
EPOCH 8 / 50
loss 0.0443:   0%|
F1_Score= 0.6986754966887417
EPOCH 9 / 50
loss 0.0311:   0%|
F1_Score= 0.7107023411371237
EPOCH 10 / 50
loss 0.0326:   0%|
F1_Score= 0.7149837133550488
EPOCH 11 / 50
loss 0.0260:   0%|
F1_Score= 0.7429048414023373
EPOCH 12 / 50
loss 0.0251:   0%|
F1_Score= 0.7532894736842105
EPOCH 13 / 50
loss 0.0112:   0%|
F1_Score= 0.7491525423728813
EPOCH 14 / 50
loss 0.0211:   0%|
F1_Score= 0.7662771285475793
EPOCH 15 / 50
loss 0.0248:   0%|
F1_Score= 0.7719298245614035
EPOCH 16 / 50
loss 0.0217:   0%|
F1_Score= 0.7566225165562914
EPOCH 17 / 50
loss 0.0154:   0%|
```

```
EPOCH 1 / 50
loss 0.1371:   0%|                                        | 0/121 [00:06<?, ?it/s]
EPOCH 2 / 50
loss 0.0920:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 3 / 50
loss 0.0801:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 4 / 50
loss 0.0674:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 5 / 50
loss 0.0599:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 6 / 50
loss 0.0634:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 7 / 50
loss 0.0389:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 8 / 50
loss 0.0566:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 9 / 50
loss 0.0598:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 10 / 50
loss 0.0608:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 11 / 50
loss 0.0380:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 12 / 50
loss 0.0337:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 13 / 50
loss 0.0282:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 14 / 50
loss 0.0247:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 15 / 50
loss 0.0407:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 16 / 50
loss 0.0360:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 17 / 50
loss 0.0246:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 18 / 50
loss 0.0391:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 19 / 50
loss 0.0254:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 20 / 50
loss 0.0292:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 21 / 50
loss 0.0270:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 22 / 50
loss 0.0243:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 23 / 50
loss 0.0291:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 24 / 50
loss 0.0068:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 25 / 50
loss 0.0201:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 26 / 50
loss 0.0194:   0%|                                        | 0/121 [00:04<?, ?it/s]
```



```
loss 0.0201:   0%|
F1_Score= 0.8
EPOCH 34 / 50
loss 0.0085:   0%|
F1_Score= 0.7996688741721855
EPOCH 35 / 50
loss 0.0055:   0%|
F1_Score= 0.8042588042588042
EPOCH 36 / 50
loss 0.0044:   0%|
F1_Score= 0.8023549201009251
EPOCH 37 / 50
loss 0.0022:   0%|
F1_Score= 0.80623973727422
EPOCH 38 / 50
loss 0.0189:   0%|
F1_Score= 0.803347280334728
EPOCH 39 / 50
loss 0.0129:   0%|
F1_Score= 0.8033333333333333
EPOCH 40 / 50
loss 0.0006:   0%|
F1_Score= 0.8083333333333333
EPOCH 41 / 50
loss 0.0011:   0%|
F1_Score= 0.8016326530612244
EPOCH 42 / 50
loss 0.0035:   0%|
F1_Score= 0.7941176470588235
EPOCH 43 / 50
loss 0.0014:   0%|
F1_Score= 0.7938311688311688
EPOCH 44 / 50
loss 0.0099:   0%|
F1_Score= 0.806930693069307
EPOCH 45 / 50
loss 0.0010:   0%|
F1_Score= 0.7983606557377049
EPOCH 46 / 50
loss 0.0028:   0%|
F1_Score= 0.8016326530612244
EPOCH 47 / 50
loss 0.0020:   0%|
F1_Score= 0.8052588331963846
EPOCH 48 / 50
loss 0.0053:   0%|
F1_Score= 0.8036006546644845
EPOCH 49 / 50
loss 0.0009:   0%|
F1_Score= 0.7990275526742301
EPOCH 50 / 50
loss 0.0011:   0%|
```

```
EPOCH 25 / 50
loss 0.0201:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 26 / 50
loss 0.0194:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 27 / 50
loss 0.0091:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 28 / 50
loss 0.0142:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 29 / 50
loss 0.0099:   0%|                                        | 0/121 [00:04<?, ?it/s]
EPOCH 30 / 50
loss 0.0182:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 31 / 50
loss 0.0167:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 32 / 50
loss 0.0150:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 33 / 50
loss 0.0163:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 34 / 50
loss 0.0225:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 35 / 50
loss 0.0149:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 36 / 50
loss 0.0049:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 37 / 50
loss 0.0274:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 38 / 50
loss 0.0175:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 39 / 50
loss 0.0162:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 40 / 50
loss 0.0118:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 41 / 50
loss 0.0130:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 42 / 50
loss 0.0061:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 43 / 50
loss 0.0209:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 44 / 50
loss 0.0477:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 45 / 50
loss 0.0211:   0%|                                        | 0/121 [00:05<?, ?it/s]
EPOCH 46 / 50
loss 0.0037:   0%|                                        | 0/121 [00:06<?, ?it/s]
EPOCH 47 / 50
loss 0.0333:   0%|                                        | 0/121 [00:06<?, ?it/s]
EPOCH 48 / 50
loss 0.0048:   0%|                                        | 0/121 [00:06<?, ?it/s]
EPOCH 49 / 50
loss 0.0079:   0%|                                        | 0/121 [00:06<?, ?it/s]
EPOCH 50 / 50
loss 0.0057:   0%|                                        | 0/121 [00:06<?, ?it/s]
```

At the beginning of the training, the loss using new LSTM model almost lower than the original LSTM model at each epoch. At in the end, the final loss of new LSTM is also lower than original one. So considering loss reflects the "extent of error", the new LSTM model makes less error. And I notice that the training time in new LSTM model increase from 5 seconds to 11 seconds per epoch. I think the reason is that I only use CPU to train the model. The new LSTM model combine forget and input together which the weight matrix can be twice larger than original one. I think CPU is not good at doing it. If I use GPU to train the model, I think the training time will decrease. So the new model has better performance about loss than original one.

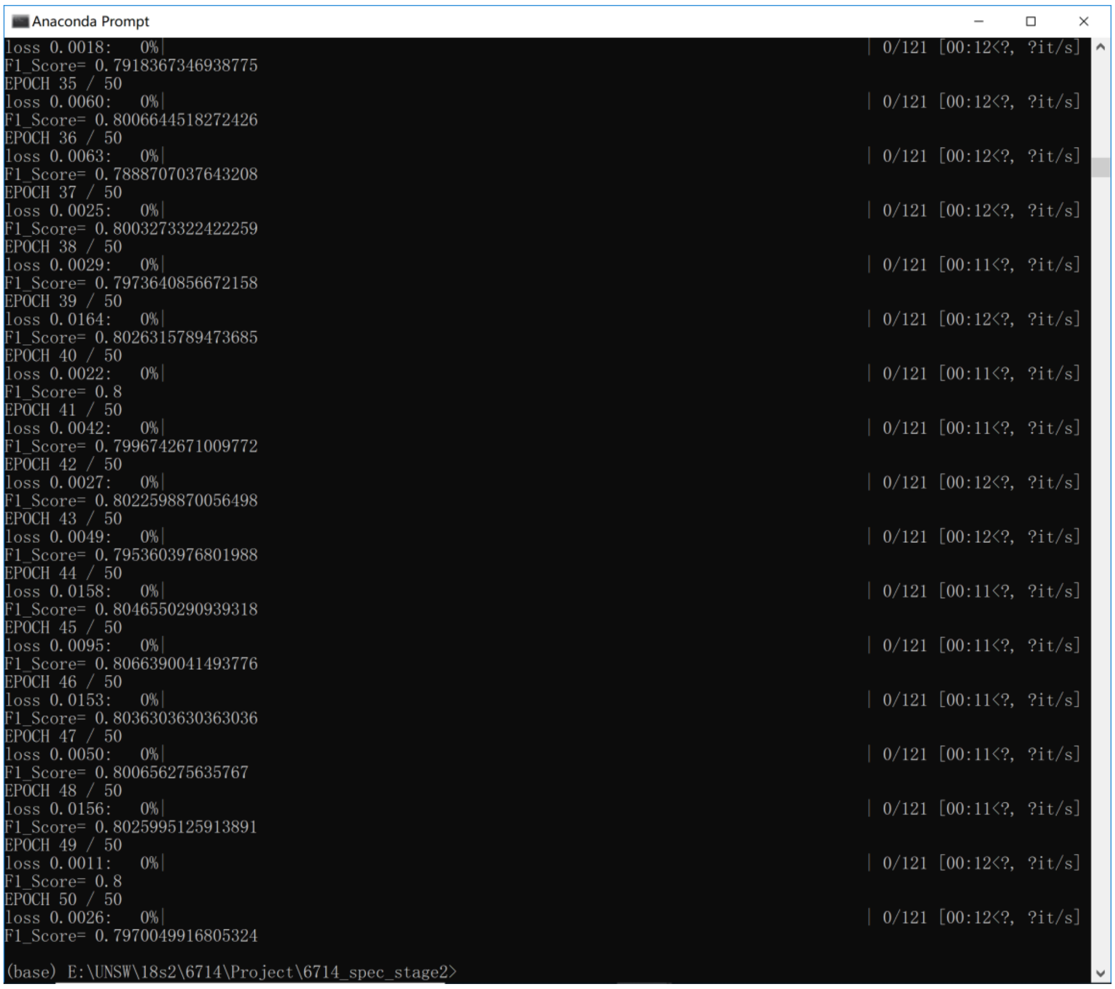
What's more, after the modification of new LSTM model, the F1 score goes up slightly at each epoch, and finally get 0.8026 which is greater than original one at 0.7813.

5.How do you implement get_char_sequence()?

The get_char_sequence() function receive two parameters, one is the batch_char matrix and another is the length of batch_world. The first step is to extract the batch size and length of word from the input in different dimension. Then use .view() function to reshape the two inputs. Because of the length of each word is not the same, so I sort the batch words in order to implement padding later. Then I record each word's index in the original matrix in order to recover the original location of each word. Then I use pad_padded_sequence which is from pytorch in order to make every word at the same length. After that, I feed the pack_padded sequence to char_LSTM layer and recover the hidden states' corresponding to the sorted index. Finally, I reshape the hidden states and return it.

6.How does Modification 3 (i.e., adding Char BiLSTM layer) affect the performance?

The modification of adding Char BiLSTM layer will increase the performance of neural network theoretically. Because in NLP, it is much better to process at char level than word level. In English, some words have affix, and vocabulary has it's original word. So processing language at char level will be have accuracy than word level. The model used in this project concatenate word embedding from the data preprocessing to the hidden state of char LSTM with char embedding from the data processing and form the new input tensor. And the advantage of BiLSTM is to make the neural network "see" all the input when it need a output at one time step. After change some hyper parameter and add char BiLSTM, I get the performance below:



```
loss 0.0018:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.7918367346938775
EPOCH 35 / 50
loss 0.0060:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.8006644518272426
EPOCH 36 / 50
loss 0.0063:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.7888707037643208
EPOCH 37 / 50
loss 0.0025:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.8003273322422259
EPOCH 38 / 50
loss 0.0029:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.7973640856672158
EPOCH 39 / 50
loss 0.0164:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.8026315789473685
EPOCH 40 / 50
loss 0.0022:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.8
EPOCH 41 / 50
loss 0.0042:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.7996742671009772
EPOCH 42 / 50
loss 0.0027:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.8022598870056498
EPOCH 43 / 50
loss 0.0049:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.7953603976801988
EPOCH 44 / 50
loss 0.0158:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.8046550290939318
EPOCH 45 / 50
loss 0.0095:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.8066390041493776
EPOCH 46 / 50
loss 0.0153:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.8036303630363036
EPOCH 47 / 50
loss 0.0050:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.800656275635767
EPOCH 48 / 50
loss 0.0156:   0%|                                                              | 0/121 [00:11<?, ?it/s]
F1_Score= 0.8025995125913891
EPOCH 49 / 50
loss 0.0011:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.8
EPOCH 50 / 50
loss 0.0026:   0%|                                                              | 0/121 [00:12<?, ?it/s]
F1_Score= 0.7970049916805324

(base) E:\UNSW\18s2\6714\Project\6714_spec_stage2>
```

I notice that after around 30th epoch, the loss does not go down obviously and the F1 score does not go up too. So I think the model is over-fitting at that step. The training after 30th step is not helpful for development. And in this project, I think the pre-processing of training set is not enough. In conclusion, the char LSTM model will theoretically increase the performance, but not do that in this model because of some pre-working and the choose of hyper parameters.