COMP6714 Project1

1.

a) How do you implement `evaluate()`?

Firstly, the length of both lists is scaled to make sure the input is valid;

Then, the parameters are initialized by 0, the below is the mean of the parameters:

right: true positive (tp);

p_denominator: true positive + false positive (tp + fp);

r_denominator: true positive + false negative (tp + fn);

pointer: used to traverse the list;

The frequency of words in each sentence is counted by ***evaluate_single()***;

The tags of each sentence are counted by ***find_labels ()***;

The 'O' tags are removed by this function, meanwhile, other tags are sorted into a list via the regulation of IOB2;

Example:	Bill	works	for	Bank	of	America	and	takes	the	Boston	Philadelphia	train.
IO:	I-PER	O	O	I-ORG	I-ORG	I-ORG	O	O	O	I-LOC	I-LOC	O
IOB1:	I-PER	O	O	I-ORG	I-ORG	I-ORG	O	O	O	I-LOC	B-LOC	O
IOB2:	B-PER	O	O	B-ORG	I-ORG	I-ORG	O	O	O	B-LOC	B-LOC	O

Finally, some special cases are considered, because when the precision and recall is calculated, the divisor cannot become 0;

Therefore, the special cases which several denominators are zero are taken separately for discussion and calculation.

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

processing speed	-	training percent	↘
f1	-	loss	-

Almost unaffected after modification 1, only training percent lower than then before;

After the f1 measure function was added, the processing speed, loss and f1 cannot be compared with the previous process, because it reflect the model accuracy on the dev data set, not train set.

2.

a) How do you implement `new_LSTMCell()`?

```
def new_LSTMCell(input, hidden, w_ih, w_hh, b_ih=None, b_hh=None):
    hx, cx = hidden
    gates = F.linear(input, w_ih, b_ih) + F.linear(hx, w_hh, b_hh) # caculation combination
    ingate, forgetgate, cellgate, outgate = gates.chunk(4, 1) #seperate gates
    forgetgate = torch.sigmoid(forgetgate)
    ingate = 1 - forgetgate
    cellgate = torch.tanh(cellgate)
    outgate = torch.sigmoid(outgate)
    cy = (forgetgate * cx) + (ingate * cellgate)
    hy = outgate * torch.tanh(cy)
    return hy, cy
```

As the structure of LSTM and the direction of the requirements, all we need to do is to change the ingate of the previous LSTMCell, from 'torch.sigmod(ingate)' to 1-forgetgate.

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

processing speed	↘↘	training percent	↘
f1	↗	loss	↘

After we change the ingate function from sigmoid(ingate) to 1-forgetgate, the efficiency of the training has been decreasing.

3.

a) How do you implement `get_char_sequence()`?

Input (example from the 1st batch of sentences):
batch_char_index_matrices: 10*7*11, which means there are 10 sentences, every sentence with 7 words, and the max length of every words in a sentence is 11 chars; batch_word_len_lists: 10*7, which means that there are 10 sentences with 7 words each sentence ;
Implementation:
Firstly, the embedding has been created in the model by the variable called char_embeds, we use the class nn.embedding to add 50 dimensions to each character of each word in batch_char_index_matrices; >>>batch_char_index_matrices.shape: 10*7*11*50; Then, the size of batch_char_index_matrices was changed with function view(); >>>batch_char_index_matrices.shape: 70*11*50;

>>> batch_word_len_lists.shape: 70*10;

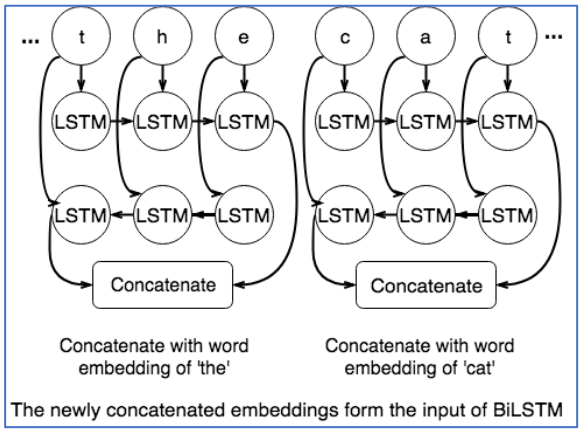
The batch_char_index_matrices and batch_word_len_lists was sorted in descending;

At this time, it is necessary to use the padding method for text length alignment on different texts in the same batch, so that the training data can be input to the LSTM model for training.

After trained from LSTM, we can get out as:

>>> output = out_sequence,(hn,cn);

All we need to do is to sort the hidden as the previous sequence. Then as the following graph shows, we need to concatenate the first h_n of the reverse direction and the last cell h_n of the normal direction.



Output:

Finally, the forward and backward is connected via torch.cat () and which will be concatenated with the data come from input_word_embeds in model.py.

>>> output :10*7*100;

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

Processing speed drops slightly, processing time increases slightly, f1 increase, training percent decrease, loss increase.

processing speed	↘	training percent	↘
f1	↗	loss	↗

After we concatenate the new char_lstm vectors with the word_embeds, that means we add more dimensions into every word, so we can increase the accuracy of the training to get better

model. The embedding matrix assigns a fixed-length vector representation to each word, especially for very large data, with resource-saving occupancy (to avoid excessive sparseness, excessive resource occupation) and ease of calculation, this is why the performance is better than before.