COMP6714 Project Report

Keshi Chen z5142821

Chieh-an Liang z5141016

Section One:

How do you implement evaluate()?

First of all, we convert golden list and predict list into list of labels in structure [Tag, lower_boundary, upper_boundary] where boundary is represented by index, e.g. we convert ['B-TAR', 'I-TAR', 'O', 'B-HYP'] into [[TAR, 0, 1], [HYP, 3, 3]]. And, at the same time, we get label counts of both list (golden_count and predict_count). (See our method 'count_labels')

Then, from the two label lists, we get the count of matched labels (match_count) (See our method 'count_match'), so that we can derive the precision by applying the formula:

precision = match_count / predict_count

Similarly, we can derive the racall by applying the formula:

recall = match_count / golden_count

Then, if precision or recall is none-zero, i.e. TP is none-zero, we can get the F1 score by the formula:

F1 = 2*precision*recall / (precision + recall)

Specially, to avoid zero division, i.e. both predict_count and golden_count are zero (in other word, true positives, false positives and false negatives are all 0), we define the F1 score to be 1.0. Otherwise, if match_count is zero while either predict_count or golden_count is not zero, we define the F1 score as zero (C, 2017)

Section 2:

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

By testing with test data after 80 epochs of training, we get following result:

Without Modification 1:

loss: 0.1648554503917694

time: 3min

With Modification 1:

F1: 0.7950065703022339

loss: 0.1648554503917694

time: 3min

Section 3:

How do you implement new_LSTMCell()?

Let ingate = 1 – forgetgate, so that it only forget when we're going to input something in its place. We only input new values to the state when it forget something older.

Section 4:

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

Without Modification 2:

F1: 0.7950065703022339

loss: 0.1648554503917694

time: 3min

With Modification 2:

F1: 0.7907276239536382

loss: 0.20874348282814026

time: 5min

Section 5:

How do you implement get_char_sequence()?

Firstly, reshape batch_char_index_matrices and batch_word_len_lists into [batch_size * max_sent_len, max_word_len] and [batch_size * max_sent_len], respectively.

Then get corresponding char_Embeddings, we will have a Final Tensor of the shape [14, 14, 50]. Sort the the mini-batch w.r.t. word-lengths, to form a pack_padded sequence. Feed the the pack_padded sequence to the char_LSTM layer and get hidden state. Finally, recover the hidden_states corresponding to the sorted index and then for each row in the batch, concatenate the first two character embeddings from each word and append it to the result and return the result.

Section 6:

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

Without Modification 3:

F1: 0.7950065703022339

loss: 0.1648554503917694

time: 3min

With Modification 3:

F1: 0.8104918032786885

loss: 0.2612205445766449

time: 15min

Reference:

C, F. (2017). dice-group/gerbil. [online] GitHub. Available at: https://github.com/dice-group/gerbil/wiki/Precision,-Recall-and-F1-measure [Accessed 23 Oct. 2018].