COMP6714 (18S2) Project Report
Submission Deadline: Friday, 26th October 2018 at 23:59:59
z5136414 Andrew Wirjaputra

How do you implement evaluate()?

To compute the F1 score of the given predicted and golden tags, we will loop through the tags in both golden_list and predict_list at the same time, while recording the starting position, phrase type and length of each tag in separate dictionary (one for golden_list and one for predict_list). We can then determine the number of correct labels (tp), golden labels (tp + fn) and predicted labels (tp + fp) by comparing the two dictionaries. The time complexity of the algorithm will be linear based on the number of tags.

The algorithm work as follow:

1. Iterate through the golden_list (and predict_list)
2. For each list in the golden_list (and predict_list), iterate through the tags
3. For each tag:
   a. If it's a "B" tag, record its position (list index, tag index), phrase type and current length (= 1).
   b. Else if the previous tag is a "B" tag:
      i. If it's an "I" tag of the same phrase, increment the previously recorded tag length.
      ii. Else if it's a different "B" tag, record the new tag position, phrase type and current length (= 1).
4. After the last tag of each list, loop through the keys of the predicted dictionary. For each key that is also found in the golden dictionary, and with the same value (meaning same position, phrase type and tag length), add to the number of correct labels. Then add the length of the golden dictionary to the number of golden labels, and the length of the predicted dictionary to the number of predicted labels.
5. Calculate F1 score (taking care of when the number of correct labels (tp) = 0).

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

Since the model is only saved (regardless of whether it is the model with best f1 or just the newest model), the loss after each EPOCH stays the same.

How do you implement new_LSTMCell()?

According to the instruction, we will replace the activation vector of the input gate with 1 – the activation vector of the forget gate.

So we will replace the following line:

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

with:

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

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

The loss after each EPOCH decreases. By using coupled forget and input gates, we can only forget when we're going to input something in its place. This would be beneficial for hyponymy classification as we need to consider tags with a length of more than 1 word (we need to consider the phrase level matching for TAR and HYP).

How do you implement get_char_sequence()?

First, we will observe the shape of batch_char_index_matrices to obtain the number of sentences, the maximum number of words in each sentence, and the maximum number of characters in each word. We will then reshape batch_char_index_matrices into [n_sentences x n_words, n_characters], and batch_word_len_lists into [n_sentences x n_words].

After obtaining the corresponding char_embeddings, we will have a final Tensor of the shape [n_sentences x n_words, n_characters, 50]. We will then sort the mini-batch with respect to word length to form a pack_padded sequence, before feeding it into the char_LSTM layer.

Finally, we can obtain the hidden state of the char_LSTM layer that has a shape of [2, n_sentences x n_words, 50]. Notice that the output of the two layers of BiLSTM is still separated. Therefore, we need to concatenate the output of the forward-layer and the backward-layer, before recovering the hidden states corresponding to the unsorted index. We can then reshape it into its original [n_sentences, n_words, 100] form (the characters are now represented by the output of BiLSTM).

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

When used alongside the re-implemented LSTM cell**,** the loss after each EPOCH increases. However, when used alongside the original LSTM cell, the loss after each EPOCH is the same as when training is done using the re-implemented LSTM cell but without adding the char BiLSTM layer, and occasionally better in later EPOCHs.

The BiLSTM layer can see the past and future context of a word and therefore are better suited to classify the word. However, when used alongside the re-implemented LSTM, the BiLSTM layer cannot function optimally, as the LSTM cell can only forget when we're going to input something in its place. Whereas with the original LSTM, both the forget and input gates can be controlled independently, and can therefore be better adjusted.