DL-POS Assignment Report **Shanlin Sun**
Prof. Ron Artstein CSCI-544
TA: Sami Abu-El-Haija Date: 05/04/2019

Baseline Model

The very high-level view of my model is shown in Figure 1. We could see it as a simplified Seq2Seq model, where the decoder is degraded as a linear classifier. The logic behind it is that the dependency between tags is totally inherited from that between input sentence, in other words, there is no need to explore the interrelation among model outputs using another recurrent neural network in our case.

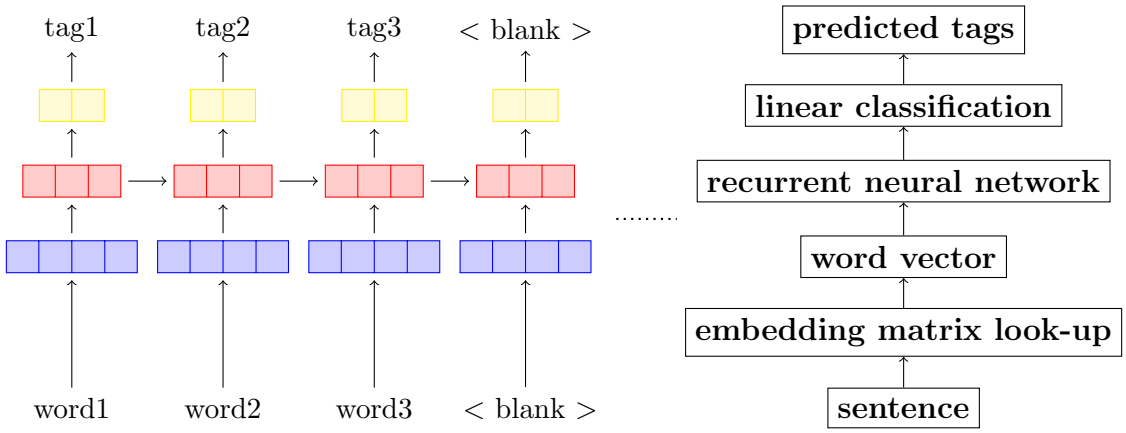


Fig. 1. Baseline Model

Even though the above is just a sketch of my baseline model, it is still compatible with my final model.

Algorithm Checklist and Analysis

In this section, I would share some algorithms, from my perspective of view, is well worthwhile to try to boost our model performance, given the hardware restriction. And I would try to explain why some algorithms seemingly in common-use doesn't work here. To test the effectiveness of each algorithm, basically, I did very simple ablation experiment.

☑ **Bidirectional LSTM:**
Undoubtedly, it will bring the major leap to our model. LSTM is very effective in dealing with sequence signal whose time order contains much information. Bidirectional mechanism lets our model be able to look back and forth.

☒ **Multi-layer RNN:**
It would bring more nonlinearity into our model, so rationally, it would enrich the representation ability of the feature extracted by RNN. But I didn't observe any consistent improvement in tagging performance. I think the problem is, too few training time and computation power makes the model far from over-fitting even using a single-layer RNN, thus it is impossible to release the potential of multi-layer RNN.

☒ **Character Embedding:**
Actually, I never tested this method because I didn't realized its value last week. If I could start again, I would definitely put it on the top of my list, although I don't know whether it would really elevate the accuracy. My reason is, If we divide the whole model into two parts, taking word embedding as the language model and following procedures as a tagging model, character embedding can somewhat supply supplement prior information to the language model. But I am sure it is not necessary.

☒ **Attention Mechanism:**
Attention Mechanism is widely used in deep learning tasks, through assigning different weights to different outputs across time or space, it would guide the model to pay more attention to some specific parts of input. It is especially intuitive to apply attention mechanism to Seq2Seq task because the sequence length is a constant no matter which layer it is in the model. But it should not be very powerful in our case, also shown by my own experiment, there might be two reasons. First, most of sequence length in the dataset is not that long, LSTM is strong enough to decide which part the model should 'remember' or 'forget'. Second, more importantly, POS problem requires us to treat every word equally. Oppositely, a translation model would tend to care which words play more important roles in deciding the meaning of the sentence.

☑ **CRF (Conditional Random Field):**
It works well in my solution. but actually I have a reason against applying it. First, after decoding our input using word embedding and recurrent neural network, the benefits of Markov network is overlapped with that of RNN, even though the theory of probabilistic model is much more mature. Plus, it would do harm to the training efficiency. In my opinion, the real value that CRF brings into my model is a new loss function, that is the inverse log-likelihood of each predicated tags, given the sequence of tag indices.

Tricks and Insights

Many works based on DL are empirical but effective. This section will introduce all tricks I tried and share some of my personal views about how to get higher percentile on secret language. Actually, all of the tricks

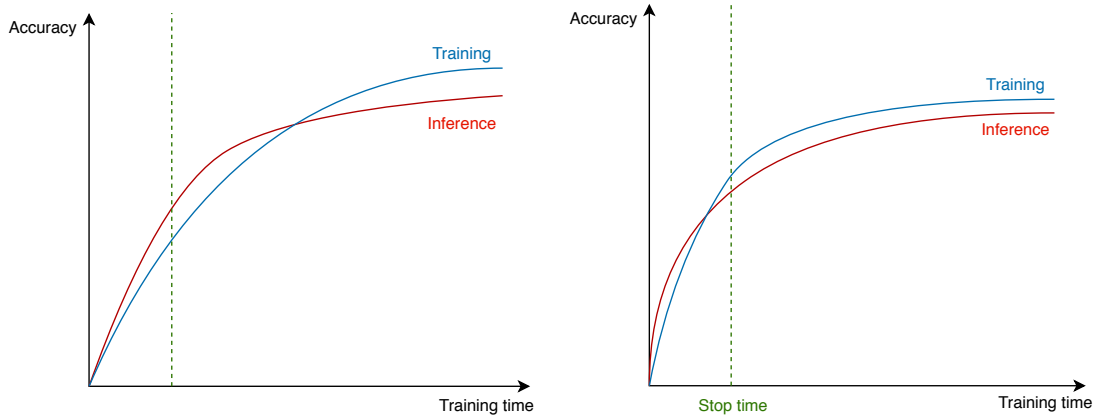
- **Use FusedRNNCell instead of RNN cell:**
  The only reason why we use FusedRNNCell is concerned about training speed. Training more epoches means we are more likely to push the model over-fitting. Before making my model reach over-fitting, I will never introduce any method to avoid it such as dropout. 'A FusedRNNCell operates on the entire time sequence at once, by putting the loop over time inside the cell. This usually leads to much more efficient, but more complex and less flexible implementations.' [1] According to my experiment, LSTMBlockFusedCell truly runs faster than LSTMCell. However, FusedRNNCell isn't compatible with normal wrapper designed for RNNCell, such as attention wrapper and bidirectional wrapper, you should implement it by yourself.

- **Adaptive Batch Size:**
  Training my model, batch size of 16 is best for Japanses dataset and batch size of 32 is best for Italian dataset. Because we make all computation on CPU, we should not expect that increasing batch size would bring much higher training efficiency. The benefits that bigger batch size could give us is a more reliable gradient decent direction each step. I

guess the reason why small batch size would cause higher accuracy on Japanese dataset is that randomness would give surprise when we deal with harder scenario.

- **Design Learning Rate Decay Schedule:**
  Normally, when we apply Adam optimization method, it is unnecessary to tune the learning rate with the growth of iterations. To get a better training result, we tend to use Adam first and then use regular gradient descent method combined with specific learning rate schedule to optimize loss function. Considering the hardware limitation, we should apply more aggressive learning rate decay schedule. Eventually, I choose step decay, more specifically, I started from the learning rate of 0.01 and halved it every 300 iterations.

- **Hyper-parameter tuning** :
  When it comes to how to select the proper parameters, there is no universal rules. In our assignment, we are only allowed to train on CPU for 11min-process time, which means the best result we could expect is over-fitting, the final measure metric is accuracy, which means I don't need to design a more complicated loss function or sampling method to beat the unbalanced labels. Personally, I prefer simpler models instead of a more delicate one if they could get almost the same accuracy, because the simpler model tend to be more explainable. At first I usually failed to get a reasonable set of hyper-parameter, but I did use some tricks to find out which direction to tune the hyper-parameter and whether I have reached the limitation of my model.



(i) Too big Model (ii) Too small Model

Fig. 2. Training and Inference Curve on Models with Different Size

Empirically, as shown in Figure 2, inference accuracy will normally be higher than training accuracy at the very beginning of training, and the bigger the model is, the more time it takes for these two accuracy curves to converge. Having Noticed this, I boldly assumed that, if the training stops when the training and inference accuracy curves intersect, the model is trained in a most efficient way. Therefore, my parameter tuning strategy is to minimize the difference between 'final accuracy' and 'last iteration accuracy'. To put it more specific, if the final accuracy is much bigger than the last iteration accuracy, I would decrease the amount of variables within the model, conversely, I would enlarge my model.

- **Data Leak:**
  Technically speaking, utilizing data leak is not illegal, but since it is unnecessary to build a impeccable grading script for a course assignment, we should not take it as an bonus to

find some leaking data. I think the most excessive data leak we are allowed to use is the size of data, otherwise, it would make seriously unfair. I found that the secret dataset has similar size of Japanese dataset, thus I make a very bold assumption that the accuracy on Japanese might be a more referrible indicator.

Final Solution

Figure 3 shows my final model, but with it, we are far away from getting a good result, further careful adjustment of hyper-parameters is equally important.

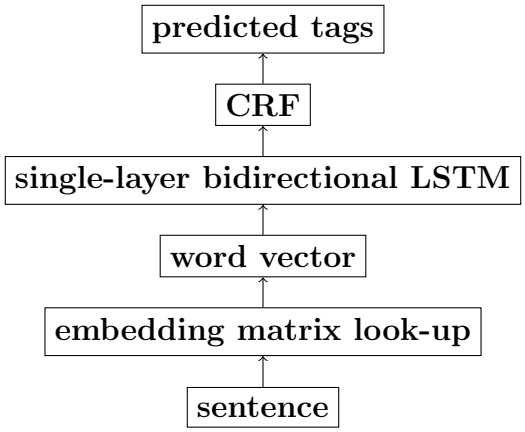


Fig. 3. Final Model

Solution details:

- word embedding feature size = 80
- bidirectional LSTM state size = 128
- number of RNN layer = 1
- no dropout
- Adam gradient descent
- adaptive batch size calculated by $\max\left(16, 16 * \left\lfloor \frac{\text{number of sequence}}{6000} \right\rfloor\right)$
- lr decay schedule formulated as $\max\left(\frac{10^{-2}}{2^{\left\lfloor \frac{\text{iteration index}}{300} \right\rfloor}}, 10^{-5}\right)$

Bibliography

[1] The TensorFlow Authors. “tf.contrib.rnn.FusedRNNCell Document”. Accessed 04 May 2019. https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/FusedRNNCell. 2016.