Xinglong Bai

[xinglonb@usc.edu](mailto:xinglonb@usc.edu)

Bidirectional-RNN for Part of Speech Tagging

**Structure**

1 Embedding Layer

2 Bidirectional RNN Layer

3 Dense Layer

4 Batch-Normalization Layer

**Embedding Layer**

Turns integers into dense vectors of fixed size word embeddings, and use it as the first layer. The size of word embedding is 40. According to experiment with different size, it appears that smaller word embedding like 10 or 20 can result in higher training speed but lower accuracy, and larger embedding size can result in higher accuracy but lower training speed.

**Bidirectional rnn layer**

The rnn layer is built with SimpleRNNCell from Keras package. Adding l2-regularizer with value 1e-6 to kernel weights of rnn cell to prevent overfitting. Compared with a single RNN layer, using a bidirectional wrapper to the rnn layer can significantly improve accuracy. In a single rnn layer, a hidden state ht only represents the information about the sequence from starting cell to t. If using bidirectional RNN layer, one rnn layer from left to right, and the other goes from right to left, and combing the output sequences of both rnn layers, each hidden state ht will represent information of the whole sequence.

**Dense Layer**

Before passing the logits returned by bi-rnn layer to a fully connected layer, applying drop out to logits with 0.5 dropout rate. Then pass it to a dense layer from Keras package. Add a l2 regularizer with value 1e-6 to kernel weights to prevent over fitting.

**Batch-Normalization Layer**

Using batch normalization to adjust the scale of logits returned by dense layer to reduce covariate shift.

**Learning Rate**

I did experiments with different learning rate. The basic idea is to apply large learning rate in the early epochs, then decay in each further epoch. I found that in the late epochs, the smaller decay step of learning rate can achieve higher improvement to accuracy, such as reducing learning rate by 1e-6 per epoch after 5th epoch. Thus, I start learning rate from 0.016, and decay by 0.004 per epoch in the first 4 epochs, then start from 0.000353 and decay by 1e-6 per epoch.

**Summary**

Batch size=32,

learn\_rate = for lr in range(start=0.016, end= 0.000353, step=-0.004) for lr in range(start = 0.000353, step=- 0.000001)

Embedding size = 40,

RNN cell units = number of tags

RNN layer = Bidirectional RNN layer

Dense dropout = 0.5

Number of Fully connected layer = 1

Kernel regularizer = l2 regularizer

Optimizer = Adam Optimizer

Japanese Accuracy = 95.1%

Italian Accuracy = 95%

**Suggestion for Further Improvement**

The packages I used for building RNN layers are from Keras. However, Keras is mainly used for fast prototyping, and it needs more training time than the model built with tensorflow contrib package. It will be faster to use tf.contrib packages than Keras. Furthermore, the rnn cell I used is very simple version of rnn cell. It can achieve higher accuracy if using more complicated rnn cell such as LSTM or GRU. For the embedding size and batch size, it would be better to set different values according to the number of tags and terms of different training data set.