BiLSTM-CRF for Chinese NER

Jiyuan Lu (jl11046@nyu.edu) Licheng Shen(ls4330@nyu.edu)

NER

Named Entity Recognition (NER) is an important area in the study of NLP. It plays an important role in many different NLP applications like information retrieval, machine translation, etc. In recent years, it is trending to use DL-based NER models and achieved state-of-the art results. Compared to classical NER models, DL-based NER has the advantage that it detects features automatically from the corpus and saves a significant effort on designing features.

Compared to English NER, some characteristics of Chinese language make it more difficult for NER. The first reason is that there is no capitalization in Chinese. In English, the initials of a named entity are usually capitalized, which is a convenient feature for detecting named entities, but in Chinese there is no such feature that we can rely on. Another reason is that in English, words in a sentence are separated by whitespaces, but Chinese sentences are not. Thus, a single sentence in Chinese may have multiple ways to be segmented into words, which makes it more difficult to determine the boundaries of named entities.

Model

Recurrent Neural Networks

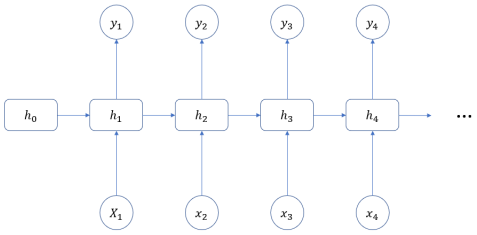

Figure 1. Basic structure of RNN

Recurrent Neural Networks (RNN) are a kind of neural networks that takes a sequence of input and in each neuron, there are hidden states served as “memory” of prior inputs in the sequence. (FIGURE 1) shows the basic structure of an RNN. Unlike Feedforward Neural Networks or Convolutional Neural Networks, the input of an RNN can be a sequence with arbitrary length, which makes it suitable for NLP tasks since we are handling sentences of different sizes. Also, the feature that prior inputs in the sequence can influence the output of current input coincides well with NLP models since the property or meaning of a word in a sentence is usually determined by not only the word itself, but also the context before or after it.

LSTM

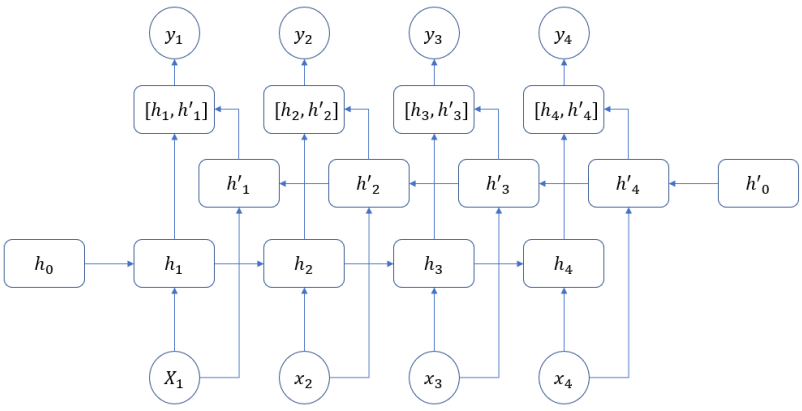

Figure 1.2 Basic structure of BiLSTM network

Although in theory, RNN can capture long term dependencies, it turned out to be exceedingly difficult for it to do so. Multiplicative gradients can be exponentially decreasing or increasing with growth of the number of layers, which makes long term dependencies suffer severely from vanishing gradient problems.

Long Short-Term Memory Networks (LSTM) are introduced to tackle such problems. In LSTM, a cell is used to store long term memory in a hidden layer and several gates are implemented to control how inputs influence cell memory.

Basic equation implementation for LSTM is as formulas below.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$o_t = \sigma(W_{xo}x_t + w_{ho}h_{t-1} + b_o)$$
$$f_t = \sigma\left(w_{xf}x_t + W_{hfh_{t-1}} + b_f\right)$$
$$\hat{c}_t = tanh\left(W_{xc}x_t + W_{hf}h_{t-1} + b_c\right)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \hat{c}_t$$
$$h_t = o_t \circ tanh(c_t)$$

Where x, o, f denote the input gate, output gate and forget gate for input x with corresponding index, h and c are hidden states and cell memories, $\sigma()$is the sigmoid function, $\circ$ represents Hadamard product, W and b are model parameters.

Note that hidden states and cell memory in LSTM implementations only reflect information from prior inputs, but in natural language, it is very common that words that come later also provide vital information for a current word. So, to capture such a feature, bidirectional LSTM is introduced. FIGURE 1.2 shows the basic structure of BiLSTM. It is a very simple trick that we use two LSTMs, the forward one gets the input sequence in normal order while the backward one gets the input in reverse order and then we concatenate them together as the hidden states output for x_t for our BiLSTM model.

CRF

We can simply get the emission score of an input for each tag through a linear mapping from the hidden states given by BiLSTM above to the space of tags, but this score only consider

Experiment Design Flow

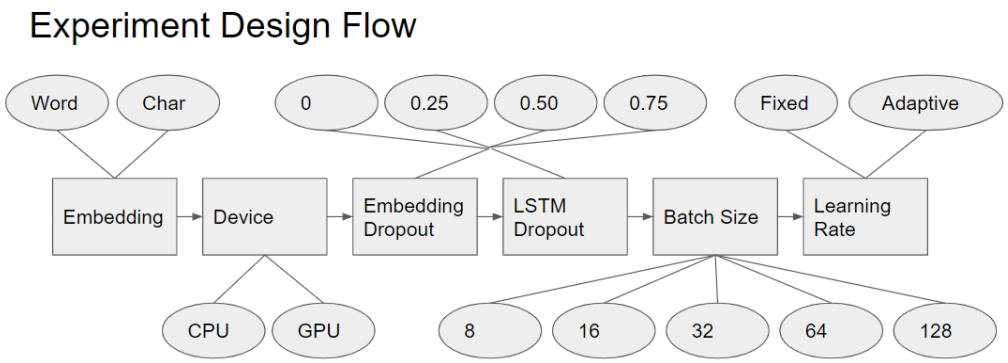

Figure 2. Experiment Design Flow

We have 6 different hyperparameters to experiment with. First we decide whether our model uses a word embedding or a character embedding. Second we decide whether to train our model on CPU or GPU. Third we vary the dropout rate applied to the embedding layer. And similarly fourth we vary the dropout rate applied to the LSTM output. Finally we vary the batch size and decide whether to use a fixed learning rate or an adaptive learning rate corresponding to different batch sizes.

Experimental Evaluation

Word Model

Dropout on the LSTM output improves word models (Figure 3.2), however, dropout on embedding has a negative effect for word models (Figure 3.1). Applying dropout to both layers does not further help improve performance (Figure 3.3) and results in a performance just slightly better than applying dropout on the embedding layer.

Moreover, with a 0.75 dropout rate on the embedding layer, the model almost cannot learn anything.

This suggests that for word embedding models, one should apply dropout on the LSTM output.

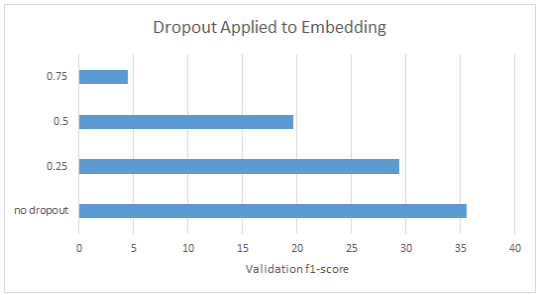

Figure 3.1 Highest validation f1-score with different dropout rate on embedding

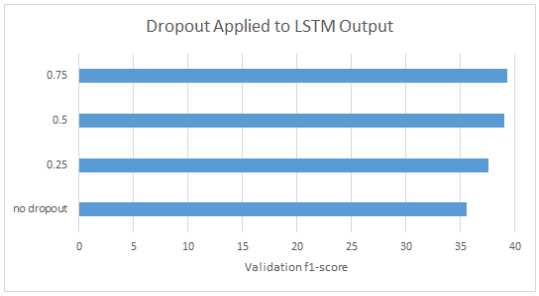

Figure 3.2 Highest validation f1-score with different dropout rate on LSTM output

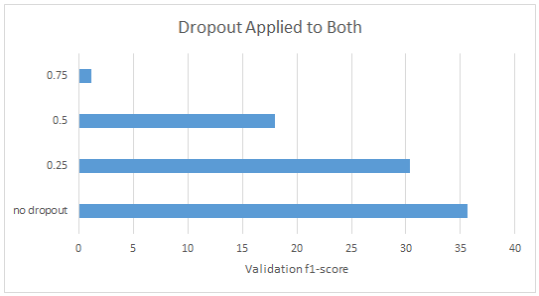

Figure 3.3 Highest validation f1-score with different dropout rate on both embedding and LSTM

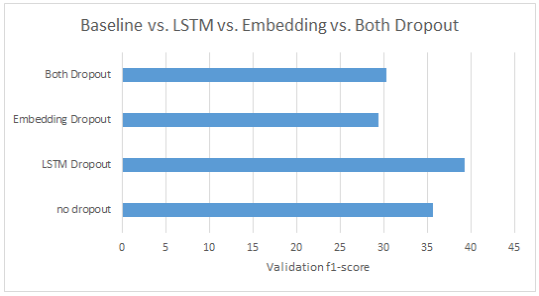

Figure 3.4 Highest validation f1-score for each scenario

Char Model

Dropout on embedding improves char models (Figure 4.2), however, dropout on the LSTM output has a negative effect for word model (Figure 4.1). Applying dropout to both layers does not further help improve performance (Figure 3.3) and results in a performance slightly worse than applying dropout on the embedding layer.

Moreover, with a 0.5 dropout rate on the embedding layer, the model the best validation f1-score across all experiments at 46.23%.

This suggests that for word embedding models, one should apply dropout after the embedding layer.

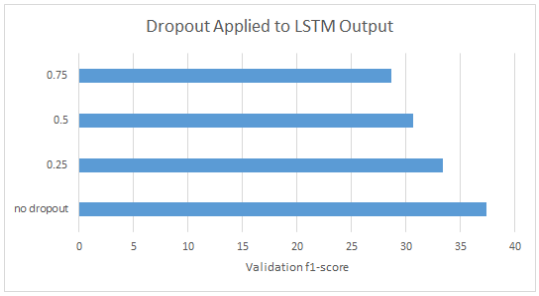

Figure 4.1 Highest validation f1-score with different dropout rate on embedding

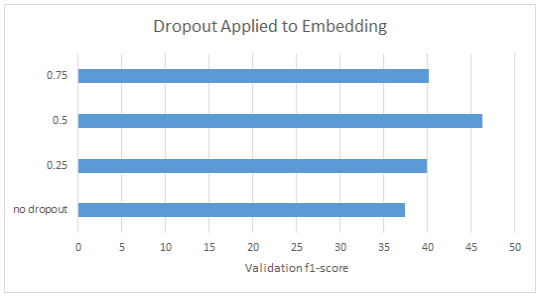

Figure 4.2 Highest validation f1-score with different dropout rate on LSTM output

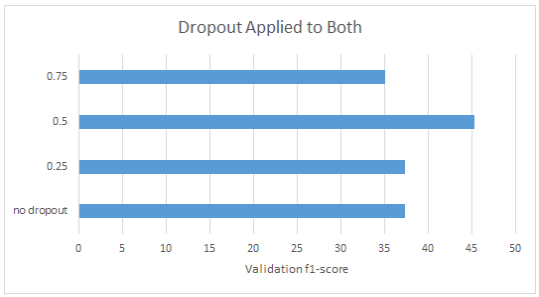

Figure 4.3 Highest validation f1-score with different dropout rate on both embedding and LSTM

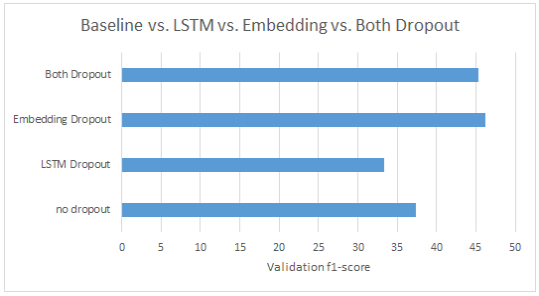

Figure 4.4 Highest validation f1-score for each scenario

Epoch Speed Up

Time-To-F1-Score Speed Up

Conclusion

Training on GPU leads to considerable speed up and faster Time-To-F1-Score with adaptive learning rate for BiLSTM CRF model on Chinese NER task.