Sequence Labeling via Bi-directional LSTM-CNNs-CRF

Abstract—This project is aimed to implement NER task on CoNLL 2003 dataset. Referring to work of Ma X etal, we constructed an end-to-end model relying on no task-specific resources or features. CNN is used to get char-level word representation and pretrained glove word vectors work as the word-level representation. Then, they are combined and processed by a BiLSTM model to capture the contextual information. Finally, we use CRF to jointly decode best label sequences. For experiments, I used Dev-test set and hyper parameters in the article. For model evaluation, we not only pay attention to accuracy, but also take precision, recall and F1-score into account. Finally, the best F1-score was achieved at 0.7624 for train set, 0.7617 for validation set and 0.7206 for test set.

I. Introduction

Sequence Labeling tasks such as part-of-speech tagging (POS) and named entity recognition (NER) lie the foundation for deep language understanding. Traditionally, linear statistical model like Hidden Markov Model (HMM) and Conditional Random Fields (CRF), which rely heavily on hand-crafted features and task-specific resources, are used to complete sequence labeling tasks. With the development of deep learning research in recent years, non-linear neural networks have been applied to many NLP problems. However, even systems that have utilized distributed representations as inputs have used these to augment, rather than replace, hand-crafted features. So the performance can decrease when the systems solely depend neural embeddings.

In this project, we refer to work of Xuezhe Ma etal[1], to construct an end-to-end sequence labeling model via Bi-directional LSTM-CNNs-CRF. In this way, we don't need task-specific resources or feature engineering, combining advantages of both linear statistical model and non-linear neural networks.

II. Models

For a word in the corpus, we have its word-level representation and char-level representation. Pretrained word embeddings can serve for the word-level representation. In this project, glove-100d[2] word vectors are used. When it comes to char-level representation, CNN model is utilized to extract morphological information of words.

A.CNN for char embedding

CNN is an effective approach to extract morphological information like prefix and suffix. We use character embeddings as input of CNN, without character type features. Figure 1 shows the CNN structure we used in our project.

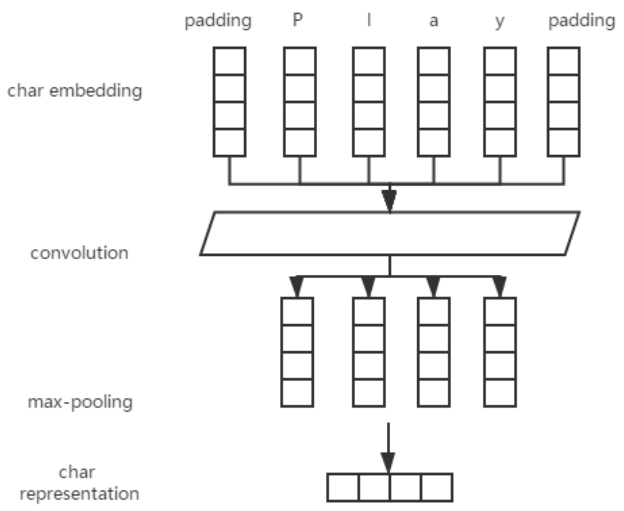

Fig 1. CNN structure

For char embeddings, which are the input of the CNN model, we initialize them randomly, as 30-dimensional embeddings uniformly sampled from range [-$\sqrt{\frac{3}{dim}}$, $\sqrt{\frac{3}{dim}}$], where dim is the dimension of embeddings.

As word length ranges a lot, it's necessary for us to pad the short words so that we can get convolution results of the same length. Here, we make words have the same length within each batch, by defining the **collate_fn** function of DataLoader.

To get better performance, hyper-parameters mentioned in article[1] are used in our model. For NER task in our project, we use 30 filters with window length 3.

B. Bi-LSTM

When we get char-level representation of words, it was concatenated with glove pretrained embedding to form the final embedding of a word. Then, we input the embeddings into Bi-LSTM model to get the past and future context information of each word.

1. LSTM Unit

LSTM can capture and store information for a long period of time, using three types of gates that control the flow of information into and out of these cells: input gates, forget gates and output gates. Given an input vector $\mathrm{x_t}$ at time step t, the previous output $\mathrm{h_{t-1}}$ and cell state $\mathrm{c_{t-1}}$, an LSTM with hidden size k computes the next output $\mathrm{h_t}$ and cell state $\mathrm{c_t}$ as:

$$\begin{bmatrix} \tilde{c}_t \\ o_t \\ i_t \\ f_t \end{bmatrix} = \begin{bmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b \right)$$

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

Figure 2 shows structure of LSTM unit.

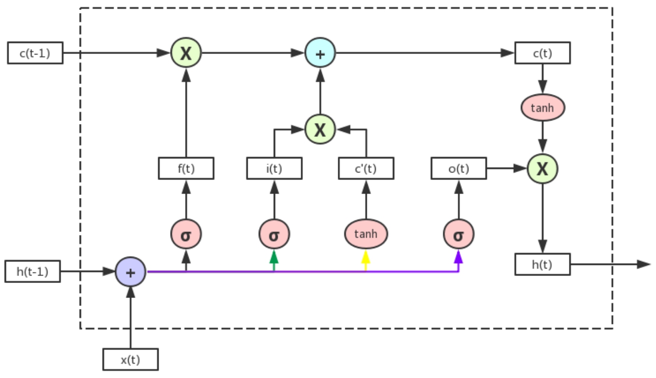

Fig 2. LSTM unit structure

2. Bi-LSTM

LSTM hidden state can only get information from past, knowing nothing about the future. However, in sequence labeling tasks, both past (left) and future (right) context can be beneficial. Therefore, bi-directional LSTM can be constructed to model each sequence forwards and backwards, to get past and future information respectively. Then the two hidden states are concatenated to form the final output.

Also, we used hyper-parameters mentioned in article[1] to get better performance. We set the state size of LSTM to 200.

C. CRF

After Bi-LSTM, we get a probability distribution of tags for each word, the dimension of the output being the number of tag types. In this way, we can use the argmax function to decide the tag of a word. However, for sequence labeling tasks, it's beneficial to consider relations between labels in neighborhoods and jointly decode the best path for a sequence. Therefore, conditional random field (CRF) is modeled for jointly labeling sequence.

Use $\mathbf{z} = \{z_1, z_2, \dots z_n\}$ to represent the input sequence w, where z_i is the word vector of ith word. And $\mathbf{y} = \{y_1, y_2, \dots y_n\}$ is a generic sequence of labels for z. Y(z) denotes the set of possible label sequences for z. Then CRF defines a conditional probability:

$$\mathrm{p}(\mathbf{y}|\mathbf{z}; \mathrm{W}, \mathrm{b}) = \frac{\prod_{i=1}^{n} \varphi_i(y_{i-1}, y_i, z)}{\sum_{y' \in Y(z)} \prod_{i=1}^{n} \varphi_i(y'_{i-1}, y'_i, z)}$$

where $\varphi_i(y', y, z) = \exp(\boldsymbol{W}_{y',y}^T z_i + \boldsymbol{b}_{y',y})$ is the potential function. **W** and **b** respectively represent weight and bias of label pair (y', y).

For CRF training, our goal is to maximize the conditional likelihood. The log likelihood for a training set $\{(\boldsymbol{z}_i, \boldsymbol{y}_i)\}$ is given by:

$$\mathrm{L(W, b)} = \sum_i \log(\, \mathrm{p}(\mathbf{y}|\mathbf{z}; \mathrm{W, b}))$$

In other words, using gradient descent method, we need to minimize the negative log likelihood: - $\mathrm{L(W, b)}$

Then, Viterbi algorithm is used to decode the best label sequence $\boldsymbol{y}^*$

$$\boldsymbol{y}^* = \underset{y \in \in Y(z)}{\operatorname{argmax}}\, \mathrm{p}(\mathbf{y}|\mathbf{z}; \mathrm{W, b})$$

To implement CRF layer, we have following important functions:

1. negative log likelihood

As we need to use gradient descent method to train the models, so we take the negative number of the log likelihood $\mathrm{L(W, b)}$. As a result, the loss can be expressed as:

$$\text{loss} = \log(\sum_{\tilde{y} \in Y(z)} e^{S(Z,\tilde{y})}) - S(Z, y)$$

where S(Z,y) means the score of the true label sequence and the other item means the sum of scores of every possible label sequence. In our code, we use this negative log likelihood as our loss function.

2. Score sentence

This function is used to compute the second item of loss, that is, the so-called gold score $S(Z, y)$. The true label sequence is used to get this score. As the formula mentioned, feat matrix and transition matrix are utilized.

$$\mathrm{S(Z, y)} = \sum_{i=0}^{n} Trans_{y_i, y_{i+1}} + \sum_{i=1}^{n} Feats_{i, y_i}$$

3. Forward computation

This function is used to compute the first item of loss, that is, $\log(\sum_{\tilde{y} \in Y(z)} e^{S(Z,\tilde{y})})$. As it is time consuming to calculate scores for all the possible sequences, a log_exp_sum function is necessary here to improve efficiency. Use idea of dynamic programming and perform iterative operation like this:

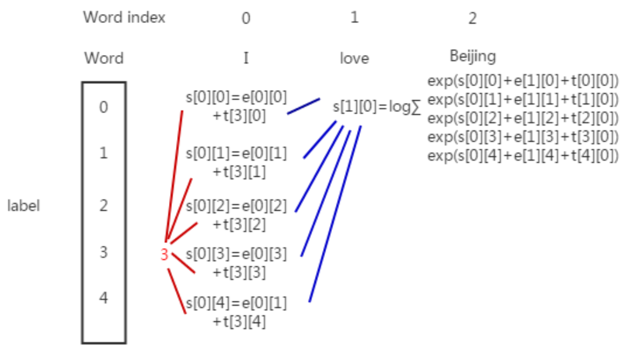

Fig 3. Computation process

Take s[1][0] as an example. s[1][0] represents the total score of all paths to this node. Here, each node represents the sum of all scores from all previous paths to the current node path. In this way, the final s [2] [4] is the sum of the final scores, which is our goal. What's more, to avoid overflow, before we sum the exponent scores, we need to minus the max score first. So in our code torch.log(torch.sum(torch.exp(vec))) is replaced by: max_score + torch.log(torch.sum(torch.exp (vec-max_score_bc)))

4. Viterbi decode

Viterbi decode uses dynamic programming to find the shortest path. Through this function, we can get the score and best path for prediction. The big problem is divided into small problems. Based on the results of each step, we can find the best strategy for next step. Finally, the global optimum can be reached through the local optimum after each step. Viterbi algorithm can be described as follows:

Tab 1. Viterbi Algorithm

```
Input: observed sequence $\mathrm{x} = (x_1, x_2 \ldots, x_n)$ model $\mathrm{F(y, x)}$
Output: best path $\mathrm{y} = (y_1, y_2 \ldots, y_n)$
Initialization: $\delta_1(j) = F(y_0 = start, y_1 = j, x)$, j=1,2,…m
Recurrence: for i=2,3,…n
    $\delta_i(l) = \max_{1 \le j \le m}\{\delta_{i-1}(j) + F_i(y_{i-1} = j, y_1 = l, x)\}$ $l = 1,2, \ldots m$
    $\varphi_i(l) = \operatorname{argmax}_{1 \le j \le m}\{\delta_{i-1}(j) + F_i(y_{i-1} = j, y_1 = l, x)\}$ $l = 1,2, \ldots m$
Stop    $\max_y F(y, x) = \max_{1 \le j \le m} \delta_n(j)$
        $y_n^* = \operatorname{argmax}_{1 \le j \le m} \delta_n(j)$
Return $y_i^* = \varphi_{i+1}(y_{i+1}^*)$
```

D. BiLSTM-CNNs-CRF

As a result, overall flow chart of BiLSTM-CNNs-CRF model is as follows:

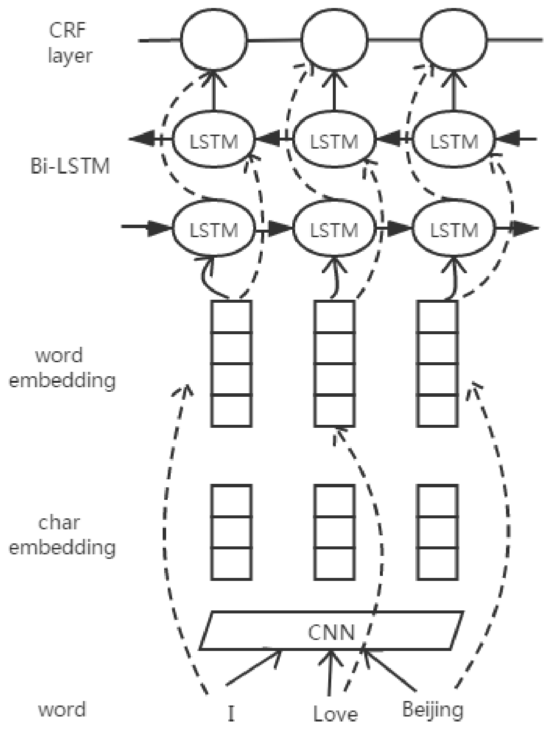

Fig 4. Overall flow

III. Experiments

A. Training

The corpus is divided into Development set and Test set, go a step further, the Development set is divided into Training set and Dev-Test set. Hence, we do training on training set, adjusting parameters and choosing the best classifier base on the results from Dev-Test set.

For word embedding, All these models are run using Stanford's GloVe[2] 100 dimensional word embeddings. For char embedding, we initialize them randomly, as 30-dimensional embeddings. For hyper-parameters, we use those in article[1].

B. Results

Here, we not only pay attention to accuracy of our models, criterions including precision, recall and f1-score are taken into account. In order to see the role of CRF, we experiment our data both on BiLSTM -CNN model and BiLSTM-CNN-CRF model. Finally, we get following results:

Tab 2. Experiment results

Model	Train				Dev				Test			
	Ac	Pre	Re	F1	Ac	Pre	Re	F1	Ac	Pre	Re	F1
BiLSTM-CNNs	0.9695	0.7362	0.7263	0.7305	0.9481	0.7561	0.7522	0.7536	0.9320	0.6859	0.7218	0.7028
BiLSTM-CNNs-CRF	0.9517	0.7527	0.7723	0.7624	0.9414	0.7621	0.7614	0.7617	0.9327	0.7032	0.7389	0.7206

According to the results shown in Table 2, BiLSTM-CNN-CRF obtains better performance than BiLSTM-CNN on all evaluation metrics of both the two tasks. This demonstrates that jointly decoding label sequences can benefit the final performance of neural network models.

IV. Conclusion

In this project, we constructed a neural network architecture for sequence labeling. Referring to work of Ma X etal[1], we learn the end-to-end model relying on no task-specific resources, feature engineering or data preprocessing. CNN is used to get char embeddings of words. BiLSTM help capture contextual information of the word embeddings. Then, CRF is used to jointly decode best label sequences. Finally, best F1-score of our model on CoNLL2003 dataset was achieved at 0.7206.

In the future, we can improve our models by exploring multi-task learning approaches to combine more information. For example, we can jointly train a neural network model with both the POS and NER tags.

Appendix

[1] Ma X , Hovy E . End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF[J]. 2016.
[2] https://nlp.stanford.edu/projects/glove/