Simplify the Usage of Lexicon in Chinese NER

Minlong Peng, Ruotian Ma, Qi Zhang, Xuanjing Huang
School of Computer Science, Fudan University, Shanghai, China
{mlpeng16,rtma19,qz,xjhuang}@fudan.edu.cn

Abstract

Recently, many works have tried to utilizing word lexicon to augment the performance of Chinese named entity recognition (NER). As a representative work in this line, Lattice-LSTM (Zhang and Yang, 2018) has achieved new state-of-the-art performance on several benchmark Chinese NER datasets. However, Lattice-LSTM suffers from a complicated model architecture, resulting in low computational efficiency. This will heavily limit its application in many industrial areas that require real-time NER response. In this work, we ask the question: if we can simplify the usage of lexicon and, at the same time, achieve comparative performance with Lattice-LSTM for Chinese NER?

Started with this question and motivated by the idea of Lattice-LSTM, we propose a concise but effective method to incorporate the lexicon information into the vector representations of characters. This way, our method can avoid introducing a complicated sequence modeling architecture to model the lexicon information. Instead, it only needs to subtly adjust the character representation layer of the neural sequence model. Experimental study on four benchmark Chinese NER datasets shows that our method can achieve much faster inference speed, comparative or better performance over Lattice-LSTM and its follwees. It also shows that our method can be easily transferred across different neural architectures.

1   Introduction

Named Entity Recognition (NER) is concerned with identifying named entities, such as person, location, product, and organization names, in unstructured text. In languages where words are naturally separated (e.g., English), NER was conventionally formulated as a sequence labeling problem, and the state-of-the-art results have been achieved by those neural-network-based models (Huang et al., 2015; Chiu and Nichols, 2016; Lample et al., 2016; Liu et al., 2018).

Compared with NER in English, Chinese NER is more difficult since sentences in Chinese are not previously segmented. Thus, one common practice in Chinese NER is first performing word segmentation using an existing CWS system and then applying a word-level sequence labeling model to the segmented sentence (Yang et al., 2016; He and Sun, 2017b). However, it is inevitable that the CWS system will wrongly segment the query sequence. This will, in turn, result in entity boundary detection errors and even entity category prediction errors in the following NER. Take the character sequence "南京市 (Nanjing) / 长江大桥 (Yangtze River Bridge)" as an example, where "/" indicates the gold segmentation result. If the sequence is segmented into "南京 (Nanjing) / 市长 (mayor) / 江大桥 (Daqiao Jiang)", the word-based NER system is definitely not able to correctly recognize "南京市 (Nanjing)" and "长江大桥 (Yangtze River Bridge)" as two entities of the location type. Instead, it is possible to incorrectly treat "南京 (Nanjing)" as a location entity and predict "江大桥 (Daqiao Jiang)" to be a person's name. Therefore, some works resort to performing Chinese NER directly on the character level, and it has been shown that this practice can achieve better performance (He and Wang, 2008; Liu et al., 2010; Li et al., 2014; Zhang and Yang, 2018).

A drawback of the purely character-based NER method is that word information, which has been proved to be useful, is not fully exploited. With this consideration, Zhang and Yang (2018) proposed to incorporating word lexicon into the character-based NER model. In addition, instead of heuristically choosing a word for the character if it matches multiple words of the lexicon, they

arXiv:1908.05969v1 [cs.CL] 16 Aug 2019

proposed to preserving all matched words of the character, leaving the following NER model to determine which matched word to apply. To achieve this, they introduced an elaborate modification to the LSTM-based sequence modeling layer of the LSTM-CRF model (Huang et al., 2015) to jointly model the character sequence and all of its matched words. Experimental studies on four public Chinese NER datasets show that Lattice-LSTM can achieve comparative or better performance on Chinese NER over existing methods.

Although successful, there exists a big problem in Lattice-LSTM that limits its application in many industrial areas, where real-time NER responses are needed. That is, its model architecture is quite complicated. This slows down its inference speed and makes it difficult to perform training and inference in parallel. In addition, it is far from easy to transfer the structure of Lattice-LSTM to other neural-network architectures (e.g., convolutional neural networks and transformers), which may be more suitable for some specific datasets.

In this work, we aim to find a easier way to achieve the idea of Lattice-LSTM, i.e., incorporating all matched words of the sentence to the character-based NER model. The first principle of our method design is to achieve a fast inference speed. To this end, we propose to encoding the matched words, obtained from the lexicon, into the representations of characters. Compared with Lattice-LSTM, this method is more concise and easier to implement. It can avoid complicated model architecture design thus has much faster inference speed. It can also be quickly adapted to any appropriate neural architectures without redesign. Given an existing neural character-based NER model, we only have to modify its character representation layer to successfully introduce the word lexicon. In addition, experimental studies on four public Chinese NER datasets show that our method can even achieve better performance than Lattice-LSTM when applying the LSTM-CRF model. *Our source code is published at* https://github.com/v-mipeng/LexiconAugmentedNER.

2 Generic Character-based Neural Architecture for Chinese NER

In this section, we provide a concise description of the generic character-based neural NER model, which conceptually contains three stacked layers. The first layer is the character representation layer, which maps each character of a sentence into a dense vector. The second layer is the sequence modeling layer. It plays the role of modeling the dependence between characters, obtaining a hidden representation for each character. The final layer is the label inference layer. It takes the hidden representation sequence as input and outputs the predicted label (with probability) for each character. We detail these three layers below.

2.1 Character Representation Layer

For a character-based Chinese NER model, the smallest unit of a sentence is a character and the sentence is seen as a character sequence $s = \{c_1, \cdots, c_n\} \in \mathcal{V}_c$, where $\mathcal{V}_c$ is the character vocabulary. Each character c_i is represented using a dense vector (embedding):

$$\mathbf{x}_i^c = \boldsymbol{e}^c(c_i), \tag{1}$$

where $\boldsymbol{e}^c$ denotes the character embedding lookup table.

Char + bichar. In addition, Zhang and Yang (2018) has proved that character bigrams are useful for representing characters, especially for those methods not use word information. Therefore, it is common to augment the character representation with bigram information by concatenating bigram embeddings with character embeddings:

$$\mathbf{x}_i^c = [\boldsymbol{e}^c(c_i) \oplus \boldsymbol{e}^b(c_i, c_{i+1})], \tag{2}$$

where $\boldsymbol{e}^b$ denotes the bigram embedding lookup table, and $\oplus$ denotes the concatenation operation. The sequence of character representations $\mathbf{x}_i^c$ form the matrix representation $\mathbf{x}^s = \{\mathbf{x}_1^c, \cdots, \mathbf{x}_n^c\}$ of s.

2.2 Sequence Modeling Layer

The sequence modeling layer models the dependency between characters built on vector representations of the characters. In this work, we explore the applicability of our method to three popular architectures of this layer: the LSTM-based, the CNN-based, and the transformer-based.

LSTM-based

The bidirectional long-short term memory network (BiLSTM) is one of the most commonly used architectures for sequence modeling (Ma and Hovy, 2016; Lample et al., 2016; Greenberg et al., 2018). It contains two LSTM (Hochreiter and Schmidhuber, 1997) cells that model the sequence in the left-to-right (forward) and right-to-left (backward) directions with two distinct sets of parameters. Here, we precisely show the definition of the forward LSTM:

$$
\begin{bmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \widetilde{\mathbf{c}}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W} \begin{bmatrix} \mathbf{x}_t^c \\ \mathbf{h}_{t-1} \end{bmatrix} + \mathbf{b} \right),
$$
$$
\mathbf{c}_t = \widetilde{\mathbf{c}}_t \odot \mathbf{i}_t + \mathbf{c}_{t-1} \odot \mathbf{f}_t,
$$
$$
\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\widetilde{\mathbf{c}}_t \odot \mathbf{i}_t + \mathbf{c}_{t-1} \odot \mathbf{f}_t). \quad (3)
$$

where σ is the element-wise sigmoid function and $\odot$ represents element-wise product. $\mathbf{W} \in \mathbb{R}^{4k_h \times (k_h + k_w)}$ and $\mathbf{b} \in \mathbb{R}^{4k_h}$ are trainable parameters. The backward LSTM shares the same definition as the forward one but in an inverse sequence order. The concatenated hidden states at the ith step of the forward and backward LSTMs $\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_i \oplus \overleftarrow{\mathbf{h}}_i]$ forms the context-dependent representation of c_i.

CNN-based

Another popular architecture for sequence modeling is the convolution network (Kim, 2014), which has been proved (Strubell et al., 2017) to be effective for Chinese NER. In this work, we apply a convolutional layer to model trigrams of the character sequence and gradually model its multigrams by stacking multiple convolutional layers. Specifically, let $\mathbf{h}_i^l$ denote the hidden representation of c_i in the lth layer with $\mathbf{h}_i^0 = \mathbf{x}_i^c$, and $\mathbf{F}^l \in \mathbb{R}^{k_l \times k_c \times 3}$ denote the corresponding filter used in this layer. To obtain the hidden representation $\mathbf{h}_i^{l+1}$ of c_i in the $(l+1)^{th}$ layer, it takes the convolution of $\mathbf{F}^l$ over the 3-gram representation:

$$
\mathbf{h}_i^{l+1} = \tanh(\langle \mathbf{h}_{<i-1,i+1>}^l, \mathbf{F}^l \rangle + b_1), \quad (4)
$$

where $\mathbf{h}_{<i-1,i+1>}^l = [\mathbf{h}_{i-1}^l; \mathbf{h}_i^l; \mathbf{h}_{i+1}^l]$ and $\langle A, B \rangle_i = \mathrm{Tr}(AB[i,:,:]^T)$. This operation applies L times, obtaining the final context-dependent representation, $\mathbf{h}_i = \mathbf{h}_i^L$, of c_i.

Transformer-based

Transformer (Vaswani et al., 2017) is originally proposed for sequence transduction, on which it has shown several advantages over the recurrent or convolutional neural networks. Intrinsically, it can also be applied to the sequence labeling task using only its encoder part.

In similar, let $\mathbf{h}_i^l$ denote the hidden representation of c_i in the lth layer with $\mathbf{h}_i^0 = \mathbf{x}_i^c$, and f^l denote a feedforward module used in this layer. To obtain the hidden representation matrix $\mathbf{h}^{l+1}$ of s in the $(l+1)^{th}$ layer, it takes the self-attention of $\mathbf{h}^l$:

$$
\mathbf{h}^{l+1} = f^l \left(\mathrm{softmax}(\frac{\mathbf{h}^{l^T} \mathbf{h}^l}{\sqrt{d^l}}) \mathbf{h}^l \right) + \mathbf{h}^l, \quad (5)
$$

where d^l is the dimension of $\mathbf{h}_i^l$. This process applies L times, obtaining $\mathbf{h}^L$. After that, the position information of each character c_i is introduced into $\mathbf{h}_i^L$ to obtain its final context-dependent representation $\mathbf{h}_i$:

$$
\mathbf{h}_i = [\mathbf{h}_i^L; PE_i], \quad (6)
$$

where $PE_i = sin(i/1000^{2j/d^L} + j\%2 \cdot \pi/2)$. We recommend you to refer to the excellent guides "The Annotated Transformer."[1] for more implementation detail of this architecture.

2.3 Label Inference Layer

On top of the sequence modeling layer, a sequential conditional random field (CRF) (Lafferty et al., 2001) layer is applied to perform label inference for the character sequence as a whole:

$$
p(\mathbf{y}|s; \boldsymbol{\theta}) = \frac{\prod_{t=1}^{n} \phi_t(\mathbf{y}_{t-1}, \mathbf{y}_t|s)}{\sum_{\mathbf{y}' \in \mathcal{Y}_s} \prod_{t=1}^{n} \phi_t(\mathbf{y}_{t-1}', \mathbf{y}_t'|s)} \quad (7)
$$

where $\mathcal{Y}_s$ denotes all possible label sequences of s, $\phi_t(y', y|\mathbf{s}) = \exp(\boldsymbol{w}_{y',y}^T \mathbf{h}_t + b_{y',y})$, where $\boldsymbol{w}_{y',y}$ and $b_{y',y}$ are trainable parameters corresponding to the label pair (y', y), and $\boldsymbol{\theta}$ denotes model parameters. For label inference, it searches for the label sequence $\mathbf{y}^*$ with the highest conditional probability given the input sequence s:

$$
\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|s; \boldsymbol{\theta}), \quad (8)
$$

which can be efficiently solved using the Viterbi algorithm (Forney, 1973).

---

[1] http://nlp.seas.harvard.edu/2018/04/03/attention.html

3 Lattice-LSTM for Chinese NER

Lattice-LSTM designs to incorporate word lexicon into the character-based neural sequence labeling model. To achieve this purpose, it first performs lexicon matching on the input sentence. It will add an directed edge from c_i to c_j, if the sub-sequence $\{c_i, \cdots, c_j\}$ of the sentence matches a word of the lexicon for $i < j$. And it preserves all lexicon matching results on a character by allowing the character to connect with multiple characters. Concretely, for a sentence $\{c_1, c_2, c_3, c_4, c_5\}$, if both its sub-sequences $\{c_1, c_2, c_3, c_4\}$ and $\{c_2, c_3, c_4\}$ match a word of the lexicon, it will add a directed edge from c_1 to c_4 and a directed edge from c_2 to c_4. This practice will turn the input form of the sentence from a chained sequence into a graph.

To model the graph-based input, Lattice-LSTM accordingly modifies the LSTM-based sequence modeling layer. Specifically, let $s_{<*,j>}$ denote the list of sub-sequences of a sentence s that match the lexicon and end with c_j, $\mathbf{h}_{<*,j>}$ denote the corresponding hidden state list $\{\mathbf{h}_i, \forall s_{<i,j>} \in s_{<*,j>}\}$, and $\mathbf{c}_{<*,j>}$ denote the corresponding memory cell list $\{\mathbf{c}_i, \forall s_{<i,j>} \in s_{<*,j>}\}$. In Lattice-LSTM, the hidden state $\mathbf{h}_j$ and memory cell $\mathbf{c}_j$ of c_j are now updated by:

$$\mathbf{h}_j, \mathbf{c}_j = f(\mathbf{h}_{j-1}, \mathbf{c}_{j-1}, \mathbf{x}_j^c, s_{<*,j>}, \mathbf{h}_{<*,j>}, \mathbf{c}_{<*,j>}), \tag{9}$$

where f is a simplified representation of the function used by Lattice-LSTM to perform memory update. Note that, in the updating process, the inputs now contains current step character representation $\mathbf{x}_j^c$, last step hidden state $\mathbf{h}_{j-1}$ and memory cell $\mathbf{c}_{j-1}$, and lexicon matched sub-sequences $s_{<*,j>}$ and their corresponding hidden state and memory cell lists, $\mathbf{h}_{<*,j>}$ and $\mathbf{c}_{<*,j>}$. We refer you to the paper of Lattice-LSTM (Zhang and Yang, 2018) for more detail of the implementation of f.

A problem of Lattice-LSTM is that its speed of sequence modeling is much slower than the normal LSTM architecture since it has to additionally model $s_{<*,j>}$, $\mathbf{h}_{<*,j>}$, and $\mathbf{c}_{<*,j>}$ for memory update. In addition, considering the implementation of f, it is hard for Lattice-LSTM to process multiple sentences in parallel (in the published implementation of Lattice-LSTM, the batch size was set to 1). This raises the necessity to design a simpler way to achieve the function of Lattice-LSTM for incorporating the word lexicon into the character-based NER model.

4 Proposed Method

In this section, we introduce our method, which aims to keep the merit of Lattice-LSTM and at the same time, make the computation efficient. We will start the description of our method from our thinking on Lattice-LSTM.

From our view, the advance of Lattice-LSTM comes from two points. The first point is that it preserve all possible matching words for each character. This can avoid the error propagation introduced by heuristically choosing a matching result of the character to the NER system. The second point is that it can introduce pre-trained word embeddings to the system, which bring great help to the final performance. While the disadvantage of Lattice-LSTM is that it turns the input form of a sentence from a chained sequence into a graph. This will greatly increase the computational cost for sentence modeling. Therefore, the design of our method should try to keep the chained input form of the sentence and at the same time, achieve the above two advanced points of Lattice-LSTM.

With this in mind, our method design was firstly motivated by the Softword technique, which was originally used for incorporating word segmentation information into downstream tasks (Zhao and Kit, 2008; Peng and Dredze, 2016). Precisely, the Softword technique augments the representation of a character with the embedding of its corresponding segmentation label:

$$\mathbf{x}_j^c \leftarrow [\mathbf{x}_j^c; \boldsymbol{e}^{seg}(seg(c_j))]. \tag{10}$$

Here, $seg(c_j) \in \mathcal{Y}_{seg}$ denotes the segmentation label of the character c_j predicted by the word segmentor, $\boldsymbol{e}^{seg}$ denotes the segmentation label embedding lookup table, and commonly $\mathcal{Y}_{seg} = \{\mathrm{B, M, E, S}\}$ with B, M, E indicating that the character is the beginning, middle, and end of a word, respectively, and S indicating that the character itself forms a single-character word.

The first idea we come out based on the Softword technique is to construct a word segmenter using the lexicon and allow a character to have multiple segmentation labels. Take the sentence $s = \{c_1, c_2, c_3, c_4, c_5\}$ as an example. If both its sub-sequences $\{c_1, c_2, c_3, c_4\}$ and $\{c_3, c_4\}$

match a word of the lexicon, then the segmentation label sequence of s using the lexicon is $segs(s) = \{\{B\}, \{M\}, \{B,M\}, \{E\}, \{O\}\}$. Here, $segs(s)_1 = \{B\}$ indicates that there is at least one sub-sequence of s matching a word of the lexicon and beginning with c_1, $segs(s)_3 = \{B,M\}$ means that there is at least one sub-sequence of s matching the lexicon and beginning with c_3 and there is also at least one lexicon matched sub-sequence in the middle of which c_3 occurs, and $segs(s)_5 = \{O\}$ means that there is no sub-sequence of s that matches the lexicon and contains c_5. The character representation is then obtained by:

$$\mathbf{x}_j^c \leftarrow [\mathbf{x}_j^c; \boldsymbol{e}^{seg}(segs(s)_j)], \qquad (11)$$

where $\boldsymbol{e}^{seg}(segs(s)_j)$ is a 5-dimensional binary vector with each dimension corresponding to an item of $\{B, M, E, S, O\}$. We call this method as *ExSoftword* in the following.

However, through the analysis of ExSoftword, we can find out that the ExSoftword method cannot fully inherit the two merits of Lattice-LSTM. Firstly, it cannot not introduce pre-trained word embeddings. Secondly, though it tries to keep all the lexicon matching results by allowing a character to have multiple segmentation labels, it still loses lots of information. In many cases, we cannot restore the matching results from the segmentation label sequence. Consider the case that in the sentence $s = \{c_1, c_2, c_3, c_4\}$, $\{c_1, c_2, c_3\}$ and $\{c_2, c_3, c_4\}$ match the lexicon. In this case, $segs(s) = \{\{B\}, \{B,M\}, \{M,E\}, \{E\}\}$. However, based on $segs(s)$ and s, we cannot say that it is $\{c_1, c_2, c_3\}$ and $\{c_2, c_3, c_4\}$ matching the lexicon since we will obtain the same segmentation label sequence when $\{c_1, c_2, c_3, c_4\}$ and $\{c_2, c_3\}$ match the lexicon.

To this end, we propose to preserving not only the possible segmentation labels of a character but also their corresponding matched words. Specifically, in this improved method, each character c of a sentence s corresponds to four word sets marked by the four segmentation labels "BMES". The word set $B(c)$ consists of all lexicon matched words on s that begin with c. Similarly, $M(c)$ consists of all lexicon matched words in the middle of which c occurs, $E(c)$ consists of all lexicon matched words that end with c, and $S(c)$ is the single-character word comprised of c. And if a word set is empty, we will add a special

word "NONE" to it to indicate this situation. Consider the sentence $s = \{c_1, \cdots, c_5\}$ and suppose that $\{c_1, c_2\}$, $\{c_1, c_2, c_3\}$, $\{c_2, c_3, c_4\}$, and $\{c_2, c_3, c_4, c_5\}$ match the lexicon. Then, for c_2, $B(c_2) = \{\{c_2, c_3, c_4\}, \{c_2, c_3, c_4, c_5\}\}$, $M(c_2) = \{\{c_1, c_2, c_3\}\}$, $E(c_2) = \{\{c_1, c_2\}\}$, and $S(c_2) = \{NONE\}$. In this way, we can now introduce the pre-trained word embeddings and moreover, we can exactly restore the matching results from the word sets of each character.

The next step of the improved method is to condense the four word sets of each character into a fixed-dimensional vector. In order to retain information as much as possible, we choose to concatenate the representations of the four word sets to represent them as a whole and add it to the character representation:

$$\begin{aligned} \boldsymbol{e}^s(B, M, E, S) &= [\boldsymbol{v}^s(B) \oplus \boldsymbol{v}^s(M) \oplus \boldsymbol{v}^s(E) \oplus \boldsymbol{v}^s(S)], \\ \mathbf{x}^c &\leftarrow [\mathbf{x}^c; \boldsymbol{e}^s(B, M, E, S)]. \end{aligned} \qquad (12)$$

Here, $\boldsymbol{v}^s$ denotes the function that maps a single word set to a dense vector.

This also means that we should map each word set into a fixed-dimensional vector. To achieve this purpose, we first tried the mean-pooling algorithm to get the vector representation of a word set $\mathcal{S}$:

$$\boldsymbol{v}^s(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{w \in \mathcal{S}} \boldsymbol{e}^w(w). \qquad (13)$$

Here, $\boldsymbol{e}^w$ denotes the word embedding lookup table. However, the empirical studies, as depicted in Table 2, show that this algorithm performs not so well . Through the comparison with Lattice-LSTM, we find out that in Lattice-LSTM, it applies a dynamic attention algorithm to weigh each matched word related to a single character. Motivated by this practice, we propose to weighing the representation of each word in the word set to get the pooling representation of the word set. However, considering the computational efficiency, we do not want to apply a dynamical weighing algorithm, like attention, to get the weight of each word. With this in mind, we propose to using the frequency of the word as an indication of its weight. The basic idea beneath this algorithm is that the more times a character sequence occurs in the data, the more likely it is a word. Note that, the frequency of a word is a static value and can be obtained offline. This can greatly accelerate the calculation of the weight of each word (e.g., using a lookup table).

Specifically, let w_c denote the character sequence constituting w and $z(w)$ denote the frequency of w_c occurring in the statistic data set (in this work, we combine training and testing data of a task to construct the statistic data set. Of course, if we have unlabelled data for the task, we can take the unlabeled data as the statistic data set). Note that, we do not add the frequency of w if w_c is covered by that of another word of the lexicon in the sentence. For example, suppose that the lexicon contains both "南京 （Nanjing）" and "南京市 （Nanjing City）". Then, when counting word frequency on the sequence "南京市长江大桥", we will not add the frequency of "南京" since it is covered by "南京市" in the sequence. This can avoid the situation that the frequency of "南京" is definitely higher than "南京市". Finally, we get the weighted representation of the word set S by:

$$\boldsymbol{v}^s(S) = \frac{1}{Z} \sum_{w \in S} z(w) \boldsymbol{e}^w(w), \qquad (14)$$

where

$$Z = \sum_{w \in B \cup M \cup E \cup S} z(w).$$

Here, we perform weight normalization on all words of the four word sets to allow them compete with each other across sets.

Further, we have tried to introducing a smoothing to the weight of each word to increase the weights of infrequent words. Specifically, we add a constant c into the frequency of each word and re-define $\boldsymbol{v}^s$ by:

$$\boldsymbol{v}^s(S) = \frac{1}{Z} \sum_{w \in S} (z(w) + c) \boldsymbol{e}^w(w), \qquad (15)$$

where

$$Z = \sum_{w \in B \cup M \cup E \cup S} z(w) + c.$$

We set c to the value that there are 10% of training words occurring less than c times within the statistic data set.

In summary, our method mainly contains the following four steps. **Firstly**, we scan each input sentence with the word lexicon, obtaining the four 'BMES' word sets for each character of the sentence. **Secondly**, we look up the frequency of each word counted on the statistic data set. **Thirdly**, we obtain the vector representation of the four word sets of each character according

Datasets	Type	Train	Dev	Test
OntoNotes	Sentence	15.7k	4.3k	4.3k
	Char	491.9k	200.5k	208.1k
MSRA	Sentence	46.4k	-	4.4k
	Char	2169.9k	-	172.6k
Weibo	Sentence	1.4k	0.27k	0.27k
	Char	73.8k	14.5	14.8k
Resume	Sentence	3.8k	0.46	0.48k
	Char	124.1k	13.9k	15.1k

Table 1: Statistics of datasets.

to Eq. (14), and add it to the character representation according to Eq. (12). Finally, based on the augmented character representations, we perform sequence labeling using any appropriate neural sequence labeling model, like LSTM-based sequence modeling layer + CRF label inference layer.

5 Experiments

5.1 Experiment Design

Firstly, we performed a development study on our method with the LSTM-based sequence modeling layer, in order to compare the implementations of $\boldsymbol{v}^s$ and to determine whether or not to use character bigrams in our method. Decision made in this step will be applied to the following experiments. **Secondly**, we verified the computational efficiency of our method compared with Lattice-LSTM and LR-CNN (Gui et al.), which is a followee of Lattice-LSTM for faster inference speed. **Thirdly**, we verified the effectiveness of our method by comparing its performance with that of Lattice-LSTM and other comparable models on four benchmark Chinese NER data sets. **Finally**, we verified the applicability of our method to different sequence labeling models.

5.2 Experiment Setup

Most experimental settings in this work follow the protocols of Lattice-LSTM (Zhang and Yang, 2018), including tested datasets, compared baselines, evaluation metrics (P, R, F1), and so on. To make this work self-completed, we concisely illustrate some primary settings of this work.

Datasets

The methods were evaluated on four Chinese NER datasets, including OntoNotes (Weischedel et al., 2011), MSRA (Levow, 2006), Weibo NER (Peng

and Dredze, 2015; He and Sun, 2017a), and Re-sume NER (Zhang and Yang, 2018). OntoNotes and MSRA are from the newswire domain, where gold-standard segmentation is available for training data. For OntoNotes, gold segmentation is also available for development and testing data. Weibo NER and Resume NER are from social media and resume, respectively. There is no gold standard segmentation in these two datasets. Table 1 shows statistic information of these datasets. As for the lexicon, we used the same one as Lattice-LSTM, which contains 5.7k single-character words, 291.5k two-character words, 278.1k three-character words, and 129.1k other words.

Implementation Detail

When applying the LSTM-based sequence modeling layer, we followed most implementation protocols of Lattice-LSTM, including character and word embedding sizes, dropout, embedding initialization, and LSTM layer number. The hidden size was set to 100 for Weibo and 256 for the rest three datasets. The learning rate was set to 0.005 for Weibo and Resume and 0.0015 for OntoNotes and MSRA with Adamax (Kingma and Ba, 2014).

When applying the CNN- and transformer-based sequence modeling layers, most hyper-parameters were the same as those used in the LSTM-based model. In addition, the layer number L for the CNN-based model was set to 4, and that for transformer-based model was set to 2 with h=4 parallel attention layers. Kernel number k_f of the CNN-based model was set to 512 for MSRA and 128 for the other datasets in all layers[2].

5.3 Development Experiments

In this experiment, we compared the implementations of v^s with the LSTM-based sequence modeling layer. In addition, we study whether or not character bigrams can bring improvement to our method.

Table 2 shows performance of three implementations of v^s without using character bigrams. From the table, we can see that the weighted pooling algorithm performs generally better than the other two implementations. Of course, we may

---

[2]Please refer to the attached source code for more implementation detail of this work and access `https://github.com/jiesutd/LatticeLSTM` for pretrained word and character embeddings.

Dataset	MP	WP	SWP
NoteNotes	0.7257	0.7554	0.7544
MSRA	0.9276	0.9350	0.9349
Weibo	0.5772	**0.6124**	0.5702
Resume	0.9533	**0.9559**	0.9427
Average	0.7560	0.8131	0.8006

Table 2: F1-score of our method with different implementations of v^s. MP denotes the mean-pooling algorithm depicted in Eq. (13), WP denotes the frequency weighted pooling algorithm depicted in Eq. (14), and SWP denotes the smoothed weighted pooling algorithm depicted in Eq. (15).
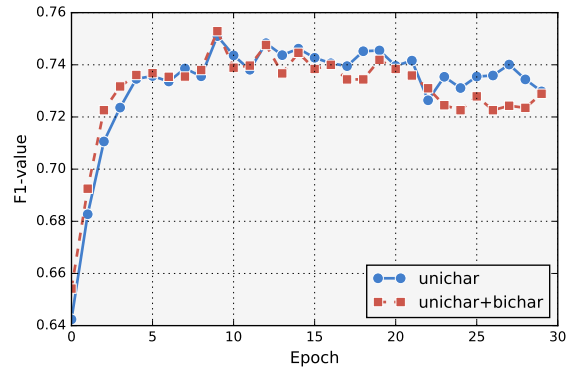

Figure 1: F1 of our proposed method against the number of training iterations on OntoNotes when using bichar or not.

obtain better results with the smoothed weighted pooling algorithm by reducing the value of c (when $c = 0$, it is equivalent to the weighted pooling algorithm). We did not do so for two reasons. The first one is to guarantee the generality of our system for unexplored tasks. The second one is that the performance of the weighted pooling algorithm is good enough compared with other state-of-the-art baselines. *Therefore, in the following experiments, we in default applied the weighted pooling algorithm to implement v^s.*

Figure 1 shows the F1-score of our method against the number of training iterations when using character bigram or not. From the figure, we can see that additionally introducing character bigrams cannot bring considerable improvement to our method. A possible explanation of this phenomenon is that the introduced word information by our proposed method has covered the bichar information. *Therefore, in the following experiments, we did not use bichar in our method.*

5.4 Computational Efficiency Study

Models	OntoNotes	MSRA	Weibo	Resume
Lattice-LSTM	11.99	14.78	11.09	15.61
LR-CNN	26.73	23.20	26.73	22.48
proposed (LSTM)	73.72	85.43	67.67	95.76
proposed (CNN)	**80.75**	**92.76**	**74.24**	**106.55**
proposed (Transformer)	63.86	58.18	58.75	61.11

Table 3: Inference speed (average sentences per second, the larger the better) of our method with different implementations of the sequence modeling layer compared with Lattice-LSTM and LR-CNN.

Table 3 shows the inference speed of our method when implementing the sequnece modeling layer with the LSTM-based, CNN-based, and Transformer-based architecture, respectively. The speed was evaluated by average sentences per second using a GPU (NVIDIA TITAN X). For a fair comparison with Lattice-LSTM and LR-CNN, *we set the batch size of our method to 1 at inference time*. From the table, we can see that our method has a much faster inference speed than Lattice-LSTM when using the LSTM-based sequence modeling layer, and it was also much faster than LR-CNN, which used an CNN architecture to implement the sequence modeling layer. And as expected, our method with the CNN-based sequence modeling layer showed some advantage in inference speed than those with the LSTM-based and Transformer-based sequence model layer.

5.5 Effectiveness Study

Table 4−7[3] show the performance of method with the LSTM-based sequence modeling layer compared with Lattice-LSTM and other comparative baselines.

OntoNotes. Table 4 shows results on OntoNotes[4], which has gold segmentation for both training and testing data. The methods of the "Gold seg" and "Auto seg" group are word-based that build on the gold word segmentation results and the automatic segmentation results, respectively. The automatic segmentation results were generated by the segmenter trained on training data of OntoNotes. Methods of the

---

[3]In Table 4−6, ∗ indicates that the model uses external labeled data for semi-supervised learning. † means that the model also uses discrete features.

[4]A result in boldface indicates that it is statistically significantly better ($p < 0.01$ in pairwise $t−$test) than the others in the same box.

input	Models	P	R	F1
Gold seg	Yang et al. 2016	65.59	71.84	68.57
	Yang et al. 2016*†	72.98	**80.15**	**76.40**
	Che et al. 2013*	77.71	72.51	75.02
	Wang et al. 2013*	76.43	72.32	74.32
	Word-based (LSTM)	76.66	63.60	69.52
	+ char + bichar	**78.62**	73.13	75.77
Auto seg	Word-based (LSTM)	72.84	59.72	65.63
	+ char + bichar	73.36	70.12	71.70
No seg	Char-based (LSTM)	68.79	60.35	64.30
	+ bichar + softword	74.36	69.43	71.89
	+ ExSoftword	69.90	66.46	68.13
	+ bichar + ExSoftword	73.80	71.05	72.40
	Lattice-LSTM	76.35	71.56	73.88
	LR-CNN (Gui et al.)	76.40	72.60	74.45
	Proposed (LSTM)	**77.31**	**73.85**	**75.54**

Table 4: Performance on OntoNotes. A method followed by (LSTM) (e.g., Proposed (LSTM)) indicates that its sequence modeling layer is LSTM-based.

"No seg" group are character-based. From the table, we can obtain several informative observations. First, by replacing the gold segmentation with the automatically generated segmentation, the F1-score of the Word-based (LSTM) + char + bichar model decreased from 75.77% to 71.70%. This shows the problem of the practice that treats the predicted word segmentation result as the true one for the word-based Chinese NER. Second, the Char-based (LSTM)+bichar+ExSoftword model achieved a 71.89% to 72.40% improvement over the Char-based (LSTM)+bichar+softword baseline on the F1-score. This indicates the feasibility of the naive extension of ExSoftword to softword. However, it still greatly underperformed Lattice-LSTM, showing its deficiency in utilizing word information. Finally, our proposed method, which is a further extension of Exsoftword, obtained a statistically significant improvement over Lattice-LSTM and even performed similarly to those word-based methods with gold segmentation, verifying its effectiveness on this data set.

MSRA. Table 5 shows results on MSRA. The word-based methods were built on the automatic segmentation results generated by the segmenter trained on training data of MSRA. Compared methods included the best statistical models on this data set, which leveraged rich handcrafted features (Chen et al., 2006; Zhang et al., 2006; Zhou et al., 2013), character embedding features (Lu et al., 2016), and radical features Dong et al. (2016). From the table, we observe that

Models	P	R	F1
Chen et al. 2006	91.22	81.71	86.20
Zhang et al. 2006*	92.20	90.18	91.18
Zhou et al. 2013	91.86	88.75	90.28
Lu et al. 2016	-	-	87.94
Dong et al. 2016	91.28	90.62	90.95
Word-based (LSTM)	90.57	83.06	86.65
+char+bichar	91.05	89.53	90.28
Char-based (LSTM)	90.74	86.96	88.81
+ bichar+softword	92.97	90.80	91.87
+ ExSoftword	90.77	87.23	88.97
+ bichar+ExSoftword	93.21	91.57	92.38
Lattice-LSTM	93.57	92.79	93.18
LR-CNN (Gui et al.)	**94.50**	92.93	**93.71**
proposed (LSTM)	93.56	**93.44**	93.50

Table 5: Performance on MSRA.

Models	NE	NM	Overall
Peng and Dredze 2015	51.96	61.05	56.05
Peng and Dredze 2016*	**55.28**	**62.97**	**58.99**
He and Sun 2017a	50.60	59.32	54.82
He and Sun 2017b*	54.50	62.17	58.23
Word-based (LSTM)	36.02	59.38	47.33
+char+bichar	43.40	60.30	52.33
Char-based (LSTM)	46.11	55.29	52.77
+ bichar+softword	50.55	60.11	56.75
+ ExSoftword	44.65	55.19	52.42
+ bichar+ExSoftword	58.93	53.38	56.02
Lattice-LSTM	53.04	62.25	58.79
LR-CNN (Gui et al.)	57.14	**66.67**	59.92
proposed (LSTM)	56.99	61.41	**61.24**

Table 6: Performance on Weibo. NE, NM and Overall denote F1-scores for named entities, nominal entities (excluding named entities) and both, respectively.

our method obtained a statistically significant improvement over Lattice-LSTM and other comparative baselines on the recall and F1-score, verifying the effectiveness of our method on this data set.

Weibo/Resume. Table 6 shows results on Weibo NER, where NE, NM, and Overall denote F1-scores for named entities, nominal entities (excluding named entities) and both, respectively. The existing state-of-the-art system (Peng and Dredze, 2016) explored rich embedding features, cross-domain data, and semi-supervised data. From the table, we can see that our proposed method achieved considerable improvement over the compared baselines on this data set. Table 7 shows results on Resume. Consistent with observations on the other three tested data sets,

Models	P	R	F1
Word-based (LSTM)	93.72	93.44	93.58
+char+bichar	94.07	94.42	94.24
Char-based (LSTM)	93.66	93.31	93.48
+ bichar+softword	94.53	94.29	94.41
+ ExSoftword	95.29	94.42	94.85
+ bichar+ExSoftword	**96.14**	94.72	95.43
Lattice-LSTM	94.81	94.11	94.46
LR-CNN (Gui et al.)	95.37	94.84	95.11
proposed (LSTM)	95.53	**95.64**	**95.59**

Table 7: Performance on Resume.

Model	OntoNotes	MSRA	Weibo	Resume
proposed (LSTM)	75.54	93.50	61.24	95.59
ExSoftword (CNN)	68.11	90.02	53.93	94.49
proposed (CNN)	**74.08**	**92.19**	**59.65**	**95.02**
ExSoftword (Transformer)	64.29	86.29	52.86	93.78
proposed (Transformer)	**71.21**	**90.48**	**61.04**	**94.59**

Table 8: F1-score with different implementations of the sequence modeling layer. ExSoftword is the shorthand of Char-based+bichar+ExSoftword.

our proposed method significantly outperformed Lattice-LSTM and the other comparable methods on this data set.

5.6 Transferability Study

Table 8 shows performance of our method with different sequence modeling architectures. From the table, we can first see that the LSTM-based architecture performed better than the CNN- and transformer- based architectures. In addition, our methods with different sequence modeling layers consistently outperformed their corresponding ExSoftword baselines. This shows that our method is applicable to different neural sequence modeling architectures for exploiting lexicon information.

6   Conclusion

In this work, we address the computational efficiency for utilizing word lexicon in Chinese NER. To achieve a high-performing NER system with fast inference speed, we proposed to adding lexicon information into the character representation and keeping the input form of a sentence as a chained sequence. Experimental study on four benchmark Chinese NER datasets shows that our method can obtain faster inference speed than the comparative methods and at the same time, achieve high performance. It also shows that our methods can apply to different neural sequence labeling models for Chinese NER.

References

Wanxiang Che, Mengqiu Wang, Christopher D Manning, and Ting Liu. 2013. Named entity recognition with bilingual constraints. In *NAACL*, pages 52–62.

Aitao Chen, Fuchun Peng, Roy Shan, and Gordon Sun. 2006. Chinese named entity recognition with conditional probabilistic models. In *SIGHAN Workshop on Chinese Language Processing*.

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association of Computational Linguistics*, 4(1):357–370.

Chuanhai Dong, Jiajun Zhang, Chengqing Zong, Masanori Hattori, and Hui Di. 2016. Character-based lstm-crf with radical-level features for chinese named entity recognition. In *Natural Language Understanding and Intelligent Applications*, pages 239–250. Springer.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Nathan Greenberg, Trapit Bansal, Patrick Verga, and Andrew McCallum. 2018. Marginal likelihood training of bilstm-crf for biomedical named entity recognition from disjoint label sets. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2824–2829.

Tao Gui, Ruotian Ma, Qi Zhang, Lujun Zhao, Yu-Gang Jiang, and Xuanjing Huang. Cnn-based chinese ner with lexicon rethinking.

Hangfeng He and Xu Sun. 2017a. F-score driven max margin neural network for named entity recognition in chinese social media. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 713–718.

Hangfeng He and Xu Sun. 2017b. A unified model for cross-domain and semi-supervised named entity recognition in chinese social media. In *AAAI*.

Jingzhou He and Houfeng Wang. 2008. Chinese named entity recognition and word segmentation based on character. In *Proceedings of the Sixth SIGHAN Workshop on Chinese Language Processing*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *Empirical Methods in Natural Language Processing*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*, pages 260–270.

Gina-Anne Levow. 2006. The third international chinese language processing bakeoff: Word segmentation and named entity recognition. In *SIGHAN Workshop on Chinese Language Processing*, pages 108–117.

Haibo Li, Masato Hagiwara, Qi Li, and Heng Ji. 2014. Comparison of the impact of word segmentation on name tagging for chinese and japanese. In *LREC*, pages 2532–2536.

Liyuan Liu, Jingbo Shang, Xiang Ren, Frank Xu, Huan Gui, Jian Peng, and Jiawei Han. 2018. Empower sequence labeling with task-aware neural language model. *AAAI Conference on Artificial Intelligence*.

Zhangxun Liu, Conghui Zhu, and Tiejun Zhao. 2010. Chinese named entity recognition with a sequence labeling approach: based on characters, or based on words? In *Advanced intelligent computing theories and applications. With aspects of artificial intelligence*, pages 634–640. Springer.

Yanan Lu, Yue Zhang, and Dong-Hong Ji. 2016. Multi-prototype chinese character embedding. In *LREC*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1064–1074.

Nanyun Peng and Mark Dredze. 2015. Named entity recognition for chinese social media with jointly trained embeddings. In *EMNLP*.

Nanyun Peng and Mark Dredze. 2016. Improving named entity recognition for chinese social media with word segmentation representation learning. In *ACL*, page 149.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *EMNLP*, pages 2670–2680.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Mengqiu Wang, Wanxiang Che, and Christopher D Manning. 2013. Effective bilingual constraints for semi-supervised learning of named entity recognizers. In *AAAI*.

Ralph Weischedel, Sameer Pradhan, Lance Ramshaw, Martha Palmer, Nianwen Xue, Mitchell Marcus, Ann Taylor, Craig Greenberg, Eduard Hovy, Robert Belvin, et al. 2011. Ontonotes release 4.0. *LDC2011T03, Philadelphia, Penn.: Linguistic Data Consortium*.

Jie Yang, Zhiyang Teng, Meishan Zhang, and Yue Zhang. 2016. Combining discrete and neural features for sequence labeling. In *CICLing*. Springer.

Suxiang Zhang, Ying Qin, Juan Wen, and Xiaojie Wang. 2006. Word segmentation and named entity recognition for sighan bakeoff3. In *SIGHAN Workshop on Chinese Language Processing*, pages 158–161.

Yue Zhang and Jie Yang. 2018. Chinese ner using lattice lstm. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), 1554-1564*.

Hai Zhao and Chunyu Kit. 2008. Unsupervised segmentation helps supervised learning of character tagging for word segmentation and named entity recognition. In *Proceedings of the Sixth SIGHAN Workshop on Chinese Language Processing*.

Junsheng Zhou, Weiguang Qu, and Fen Zhang. 2013. Chinese named entity recognition via joint identification and categorization. *Chinese journal of electronics*, 22(2):225–230.