Convolutional Neural Network with Word Embeddings for Chinese Word Segmentation

Chunqi Wang\textsuperscript{1,2} and Bo Xu\textsuperscript{1}
\textsuperscript{1}Institute of Automation, Chinese Academy of Sciences
\textsuperscript{2}University of Chinese Academy of Sciences
chqiwang@126.com, xubo@ia.ac.cn

Abstract
Character-based sequence labeling framework is flexible and efficient for Chinese word segmentation (CWS). Recently, many character-based neural models have been applied to CWS. While they obtain good performance, they have two obvious weaknesses. The first is that they heavily rely on manually designed bigram feature, i.e. they are not good at capturing $n$-gram features automatically. The second is that they make no use of full word information. For the first weakness, we propose a convolutional neural model, which is able to capture rich $n$-gram features without any feature engineering. For the second one, we propose an effective approach to integrate the proposed model with word embeddings. We evaluate the model on two benchmark datasets: PKU and MSR. Without any feature engineering, the model obtains competitive performance — 95.7\% on PKU and 97.3\% on MSR. Armed with word embeddings, the model achieves state-of-the-art performance on both datasets — 96.5\% on PKU and 98.0\% on MSR, without using any external labeled resource.

1 Introduction
Unlike English and other western languages, most east Asian languages, including Chinese, are written without explicit word delimiters. However, most natural language processing (NLP) applications are word-based. Therefore, word segmentation is an essential step for processing those languages. CWS is often treated as a character-based sequence labeling task (Xue et al., 2003; Peng et al., 2004). Figure 1 gives an intuitive explanation. Linear models, such as Maximum Entropy (ME) (Berger et al., 1996) and Conditional Random Fields (CRF) (Lafferty et al., 2001), have been widely used for sequence labeling tasks. However, they often depend heavily on well-designed hand-crafted features.

Recently, neural networks have been widely used for NLP tasks. Collobert et al. (2011) proposed a unified neural architecture for various sequence labeling tasks. Instead of exploiting hand-crafted input features carefully optimized for each task, their system learns internal representations automatically. As for CWS, there are a series of works, which share the main idea with Collobert et al. (2011) but vary in the network architecture. In particular, feed-forward neural network (Zheng et al., 2013), tensor neural network (Pei et al., 2014), recursive neural network (Chen et al., 2015a), long-short term memory (LSTM) (Chen et al., 2015b), as well as the combination of LSTM and recursive neural network (Xu and Sun, 2016) have been used to derive contextual representations from input character sequences, which are then fed to a prediction layer.

Despite of the great success of above models, they have two weaknesses. The first is that they are not good at capturing $n$-gram features automatically. Experimental results show that their models perform badly when no bigram feature is explicitly used. One of the strengths of neural networks is the ability to learn features automatically. However, this strength has not been well exploited in

<table>
<thead>
<tr>
<th>S</th>
<th>S</th>
<th>B</th>
<th>E</th>
<th>B</th>
<th>M</th>
<th>E</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l)</td>
<td>(have)</td>
<td>(a)</td>
<td>(computer)</td>
<td>( )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Chinese word segmentation as a sequence labeling task. This figure presents the common BMES (Beginning, Middle, End, Singleton) tagging scheme.
their works. The second is that they make no use of full word information. Full word information has shown its effectiveness in word-based CWS systems (Andrew, 2006; Zhang and Clark, 2007; Sun et al., 2009). Recently, Liu et al. (2016); Zhang et al. (2016) utilized word embeddings to boost performance of word-based CWS models. However, for character-based CWS models, word information is not easy to be integrated.

For the first weakness, we propose a convolutional neural model, which is also character-based. Previous works have shown that convolutional layers have the ability to capture rich n-gram features (Kim et al., 2016). We use stacked convolutional layers to derive contextual representations from input sequence, which are then fed into a CRF layer for sequence-level prediction. For the second weakness, we propose an effective approach to incorporate word embeddings into the proposed model. The word embeddings are learned from large auto-segmented data. Hence, this approach belongs to the category of semi-supervised learning.

We evaluate our model on two benchmark datasets: PKU and MSR. Experimental results show that even without the help of explicit n-gram feature, our model is capable of capturing rich n-gram information automatically, and obtains competitive performance — 95.7% on PKU and 97.3% on MSR (F score). Furthermore, armed with word embeddings, our model achieves state-of-the-art performance on both datasets — 96.5% on PKU and 98.0% on MSR, without using any external labeled resource. 1

2 Architecture

In this section, we introduce the architecture from bottom to top.

2.1 Lookup Table

The first step to process a sentence by deep neural networks is often to transform words or characters into embeddings (Bengio et al., 2003; Collobert et al., 2011). This transformation is done by lookup table operation. A character lookup table $M_{char} \in \mathbb{R}^{|V_{char}| \times d}$ (where $|V_{char}|$ denotes the size of the character vocabulary and $d$ denotes the dimension of embeddings) is associated with all characters. Given a sentence $S = (c_1, c_2, ..., c_L)$, after the lookup table operation, we obtain a matrix $X \in \mathbb{R}^{L \times d}$ where the $i$'th row is the character embedding of $c_i$.

Besides the character, other features can be easily incorporated into the model (we shall see word feature in section 3). We associate to each feature a lookup table (some features may share the same lookup table) and the final representation is calculated as the concatenation of all corresponding feature embeddings.

2.2 Convolutional Layer

Many neural network models have been explored for CWS. However, experimental results show that they are not able to capture n-gram information automatically (Pei et al., 2014; Chen et al., 2015a,b). To achieve good performance, n-gram feature must be used explicitly. To overcome this weakness, we use convolutional layers (Waibel et al., 1989) to encode contextual information. Convolutional neural networks (CNNs) have shown its great effectiveness in computer vision tasks (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; He et al., 2016). Recently, Zhang et al. (2015) applied character-level CNNs to text classification task. They showed that CNNs tend to outperform traditional n-gram models as the dataset goes larger. Kim et al. (2016) also observed that character-level CNN learns to differentiate between different types of n-grams — prefixes, suffixes and others, automatically.

Our network is quite simple — only convolutional layers is used (no pooling layer). Gated lin-
Correlations between adjacent labels can be modeled as a transition matrix $T \in \mathbb{R}^{C \times C}$. Given a sentence $S = (c_1, c_2, ..., c_L)$, we have corresponding scores $E \in \mathbb{R}^{L \times C}$ given by the convolutional layers. For a label sequence $y = (y_1, y_2, ..., y_L)$, we define its unnormalized score to be

$$s(S, y) = \sum_{i=1}^{L} E_{i, y_i} + \sum_{i=1}^{L-1} T_{y_i, y_{i+1}}.$$

then the probability of the label sequence is defined as

$$P(y | S) = \frac{e^{s(S, y)}}{\sum_{y' \in Y} e^{s(S, y')}}$$

where $Y$ is the set of all valid label sequences. This actually takes the form of linear chain CRF (Lafferty et al., 2001). Then the final loss of the proposed model is defined as the negative log-likelihood of the ground-truth label sequence $y^*$

$$\mathcal{L}(S, y^*) = -\log P(y^* | S).$$

During training, the loss function is minimized by back propagation. During test, Viterbi algorithm is applied to quickly find the label sequence with maximum probability.

### 3 Integration with Word Embeddings

Character-based CWS models have the superiority of being flexible and efficient. However, full word information is not easy to be incorporated. There is another type of CWS models: the word-based models. Models belong to this category utilize not only character-level information, but also word-level (Zhang and Clark, 2007; Andrew, 2006; Sun et al., 2009). Recent works have shown that word embeddings learned from large auto-segmented data lead to great improvements in word-based CWS systems (Liu et al., 2016; Zhang et al., 2016). We propose an effective approach to integrate word embeddings with our character-based model. The integration brings two benefits. On the one hand, full word information can be used. On the other hand, large unlabeled data can be better exploited.

To use word embeddings, we design a set of word features, which are listed in Table 1. We associate to the word features a lookup table $M_{\text{word}}$. Then the final representation of $c_i$ is defined as

$$R(c_i) = M_{\text{char}}[c_i] \oplus M_{\text{word}}[c_i] \oplus M_{\text{word}}[c_{i-1}c_i] \oplus \cdots \oplus M_{\text{word}}[c_{i+1}c_{i+2}c_{i+3}]$$
Features

<table>
<thead>
<tr>
<th>Length</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(c_i)</td>
</tr>
<tr>
<td>2</td>
<td>(c_i-c_i-1, c_i-c_i+1)</td>
</tr>
<tr>
<td>3</td>
<td>(c_i-c_i-1, c_i-c_i+1, c_i-c_i+1, c_i-c_i+2)</td>
</tr>
<tr>
<td>4</td>
<td>(c_i-c_i-1, c_i-c_i+1, c_i-c_i+1, c_i-c_i+2, c_i-c_i+1, c_i-c_i+2, c_i-c_i+2, c_i-c_i+3)</td>
</tr>
</tbody>
</table>

Table 1: Word features at position \(i\) given a sentence \(S = (c_1, c_2, ... , c_t)\). Only the words that include the current character \(c_i\) (marked with underline) are considered as word feature. Hence, the number of features can be controlled in a reasonable range. We also restrict the max length of words to 4 since few words contain more than 4 characters in Chinese. Note that the feature space is still tremendous \(O(N^4)\), where \(N\) is the number of characters.

where \(\oplus\) denotes the concatenation operation. Note that the max length of word features is set to 4, therefore the feature space is extremely large \(O(N^4)\). A key step is to shrink the feature space so that the memory cost can be confined within a feasible scope. In the meanwhile, the problem of data sparsity can be eased. The solution is as following. Given the unlabeled data \(D_{un}\) and a teacher CWS model, we segment \(D_{un}\) with the teacher model and get the auto-segmented data \(D'_{un}\). A vocabulary \(V_{word}\) is generated from \(D'_{un}\), where low frequency words are discarded \(^2\). We replace \(M_{word}[\ast]\) with \(M_{word}[UNK]\) if \(\ast \notin V_{word}\) (UNK denotes the unknown words).

To better exploit the auto-segmented data \(D'_{un}\), we adopt an off-the-self tool \textit{word2vec}\(^3\) (Mikolov et al., 2013) to pretrain the word embeddings. The whole procedure is summarized as following sets:

1. Train a teacher model that does not rely on word feature.
2. Segment unlabeled data \(D\) with the teacher model and get the auto-segmented data \(D'\).
3. Build a vocabulary \(V_{word}\) from \(D'\). Replace all words not appear in \(V_{word}\) with \(UNK\).
4. Pretrain word embeddings on \(D'\) using \textit{word2vec}.
5. Train the student model with word feature using the pretrained word embeddings.

Note that no external labeled data is used in this procedure.

---

\(^2\)The threshold of frequency is set to 5, which is the default setting of \textit{word2vec}.

\(^3\)https://code.google.com/p/word2vec

4 Experiments

4.1 Settings

Datasets We evaluate our model on two benchmark datasets, PKU and MSR, from the second International Chinese Word Segmentation Bake-off (Emerson, 2005). Both datasets are commonly used by previous state-of-the-art models. For both datasets, last 10% of the training data are used as development set. The unlabeled data used in this work is news data collected by Sogou \(^4\).

We do not perform any preprocessing for these datasets, such as replacing continuous digits and English characters with a single token.

Dropout Dropout (Srivastava et al., 2014) is a very efficient method for preventing overfit, especially when the dataset is small. We apply dropout to our model on all convolutional layers and embedding layers. The dropout rate is fixed to 0.2.

Hyper-parameters For both datasets, we use the same set of hyper-parameters, which are presented in Table 2. For all convolutional layers, we just use the same number of channels. Following the practice of designing very deep CNN in computer vision (Simonyan and Zisserman, 2014), we use a small kernel width, i.e. 3, for all convolutional layers. To avoid computational inefficiency, we use a relatively small dimension, i.e. 50, for word embeddings.

Pretraining Character embeddings and word embeddings are pretrained on unlabeled or auto-segmented data by \textit{word2vec}. Since the pretrained embeddings are not task-oriented, they are fine-tuned during supervised training by normal backpropagation.\(^5\)

Optimization Adam algorithm (Kingma and Ba, 2014) is applied to optimize our model. We use default parameters given in the original paper

\(^4\)http://www.sogou.com/labs/resource/ca.php

\(^5\)We also try to use fixed word embeddings as Zhang et al. (2016) do but no significant difference is observed.
and we set batch size to 100. For both datasets, we train no more than 100 epochs. The final models are chosen by their performance on the development set.

Weight normalization (Salimans and Kingma, 2016) is applied for all convolutional layers to accelerate the training procedure and obvious acceleration is observed.

Table 3 gives the performances of our models, as well as previous state-of-the-art models. Two proposed models are shown in the table:

- **CONV-SEG** It is our preliminary model without word embeddings. Character embeddings are pretrained on large unlabeled data.

- **WE-CONV-SEG** On the basis of CONV-SEG, word embeddings are used. We use CONV-SEG as the teacher model (see section 3).

Our preliminary model CONV-SEG achieves competitive performance without any feature engineering. Armed with word embeddings, WE-CONV-SEG obtains state-of-the-art performance on both PKU and MSR datasets without using any external labeled data. WE-CONV-SEG outperforms state-of-the-art neural model (Zhang et al., 2016) in a large margin (+0.8 on PKU and +0.3 in MSR). Chen et al. (2015b) preprocessed all datasets by replacing the continuous English character and digits with a unique token and thus their model obtains excellent score on PKU dataset. However, WE-CONV-SEG achieves the same performance on PKU and outperforms their model on MSR, without any data preprocessing.

We also observe that WE-CONV-SEG converges much faster compared to CONV-SEG. Figure 4 presents the learning curves of the two models. It takes 10 to 20 epochs for WE-CONV-SEG to converge while it takes more than 60 epochs for CONV-SEG to converge.

### 4.3 Network Depth

Network depth shows great influence on the performance of deep neural networks. A too shallow network may not fit the training data very well while a too deep network may overfit or is hard to train. We evaluate the performance of the proposed model with varying depth. Figure 5 shows
the results. It is obvious that five convolutional layers is a good choice for both datasets. When we increase the depth from 1 to 5, the performance is improved significantly. However, when we increase depth from 5 to 7, even to 11 and 15, the performance is almost unchanged. This phenomenon implies that CWS rarely relies on context larger than 11.\textsuperscript{6} With more training data, deeper networks may perform better.

### 4.4 Pretraining Character Embeddings

Previous works have shown that pretraining character embeddings boost the performance of neural CWS models significantly (Pei et al., 2014; Chen et al., 2015a,b; Cai and Zhao, 2016). We verify this and get a consistent conclusion. Table 4 shows the performances with or without pretraining. Our model obtains significant improvements (+1.0 on PKU and +0.6 on MSR) with pretrained character embeddings.

#### Table 4

<table>
<thead>
<tr>
<th>Model Options</th>
<th>PKU</th>
<th>MSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>without pretraining</td>
<td>94.7</td>
<td>96.7</td>
</tr>
<tr>
<td>with pretraining</td>
<td>95.7</td>
<td>97.3</td>
</tr>
</tbody>
</table>

#### Table 5

<table>
<thead>
<tr>
<th>Models</th>
<th>PKU</th>
<th>MSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Cai and Zhao, 2016)\textsuperscript{⋄}</td>
<td>95.5</td>
<td>96.5</td>
</tr>
<tr>
<td>(Zheng et al., 2013)</td>
<td>92.8\textsuperscript{†}</td>
<td>93.9\textsuperscript{†}</td>
</tr>
<tr>
<td>(Pei et al., 2014)</td>
<td>94.0</td>
<td>94.9</td>
</tr>
<tr>
<td>(Chen et al., 2015a)</td>
<td>94.5\textsuperscript{‡}</td>
<td>95.4\textsuperscript{‡}</td>
</tr>
<tr>
<td>(Chen et al., 2015b)</td>
<td>94.8\textsuperscript{‡}</td>
<td>95.6\textsuperscript{‡}</td>
</tr>
<tr>
<td>(Xu and Sun, 2016)</td>
<td>-</td>
<td>96.1*</td>
</tr>
<tr>
<td>CONV-SEG</td>
<td>95.7</td>
<td>97.3</td>
</tr>
<tr>
<td>(Pei et al., 2014)</td>
<td>95.2</td>
<td>97.2</td>
</tr>
<tr>
<td>(Chen et al., 2015a)</td>
<td>96.4*</td>
<td>97.6*</td>
</tr>
<tr>
<td>(Chen et al., 2015b)</td>
<td>96.5*</td>
<td>97.3*</td>
</tr>
<tr>
<td>AVEBE-CONV-SEG</td>
<td>95.4</td>
<td>97.1</td>
</tr>
<tr>
<td>W2VBE-CONV-SEG</td>
<td>95.9</td>
<td>97.5</td>
</tr>
</tbody>
</table>

#### Table 5

The first/second group summarize results of models without/with bigram feature. The number in the parentheses is the absolute improvement given by explicit bigram feature. Results with \textsuperscript{★} used external dictionary. Results with \textsuperscript{†} come from Cai and Zhao (2016). Results with \textsuperscript{‡} come from Pei et al. (2014). \textsuperscript{⋄} marks word-based models.

#### 4.5 N-gram Features

In this section, we test the ability of our model in capturing n-gram features. Since unigram is indispensable and trigram is beyond memory limit, we only consider bigram.

Bigram feature has shown to play a vital role in character-based neural CWS models (Pei et al., 2014; Chen et al., 2015a,b). Without bigram feature, previous models perform badly. Table 5 gives a summarization. Without bigram feature, our model outperforms previous character-based models in a large margin (+0.9 on PKU and +1.7 on MSR). Compared with word-based model (Cai and Zhao, 2016), the improvements are also significant (+0.2 on PKU and +0.8 on MSR).

Then we arm our model with bigram feature. The bigram feature we use is the same with Pei et al. (2014); Chen et al. (2015a,b). The dimension of bigram embedding is set to 50. Following Pei et al. (2014); Chen et al. (2015a,b), the bigram embeddings are initialized by the average of corresponding pretrained character embeddings. The result model is named AVEBE-CONV-SEG and the performance is shown in Table 5. Unexpectedly, the performance of AVEBE-CONV-SEG is worse than the preliminary model CONV-SEG that uses no bigram feature (-0.3 on PKU and -0.2 on MSR).

\textsuperscript{6}Context size is calculated by $(k - 1) \times d + 1$, where $k$ and $d$ denotes the kernel size and the number of convolutional layers, respectively.
This result is dramatically inconsistent with previous works, in which the performance is significantly improved by the method. We also observe that the training cost of AVEBE-CONV-SEG is much lower than CONV-SEG. Hence we can conclude that the inconsistency is caused by overfitting. A reasonable conjecture is that the model CONV-SEG already capture abundant bigram feature automatically, therefore the model is tend to overfit when bigram feature is explicitly added.

A practicable way to overcome overfitting is to introduce priori knowledge. We introduce priori knowledge by using bigram embeddings directly pretrained on large unlabeled data, which is simmillar with (Mansur et al., 2013). We convert the unlabeled text to bigram sequence and then apply word2vec to pretrain the bigram embeddings directly. The result model is named W2VBE-CONV-SEG, and the performance is also shown in Table 5. This method leads to substantial improvements (+0.5 on PKU and +0.4 MSR) over AVEBE-CONV-SEG. However, compared to CONV-SEG, there are only slight gains (+0.2 on PKU and MSR).

All above observations verify that our proposed network has considerable superiority in capturing n-gram, at least bigram features automatically.

### 4.6 Word Embeddings

Word embeddings lead to significant improvements over the strong baseline model CONV-SEG. The improvements come from the teacher model and the large unlabeled data. A natural question is how much unlabeled data can lead to significant improvements. We study this by halving the unlabeled data. Figure 6 presents the results. As the unlabeled data becomes smaller, the performance remains unchanged at the beginning and then becomes worse. This demonstrates that the mass of unlabeled data is a key factor to achieve high performance. However, even with only 68MB unlabeled data, we can still observe remarkable improvements (+0.4 on PKU and MSR). We also observe that MSR dataset is more robust to the size of unlabeled data than PKU dataset. We conjecture that this is because MSR training set is larger than PKU training set.

We also study how the teacher’s performance influence the student. We train other two models that use different teacher models. One of them uses a worse teacher and the other uses a better teacher. The results are shown in Table 6. As expected, the worse teacher indeed creates a worse student, but the effect is marginal (-0.1 on PKU and MSR). And the better teacher brings no improvements. These facts demonstrate that the student’s performance is relatively insensitive to the teacher’s ability as long as the teacher is not too weak.

Not only the pretrained word embeddings, we also build a vocabulary $V_{word}$ from the large auto-segmented data. Both of them should have positive impacts on the improvements. To figure out their contributions quantitatively, we train a contrast model, where the pretrained word embeddings are not used but the word features and the vocabulary are persisted, i.e. the word embeddings are randomly initialized. The results are shown in Table 7. According to the results, we conclude that the pretrained word embeddings and the vocabulary have roughly equal contributions to the final
<table>
<thead>
<tr>
<th>Models</th>
<th>PKU</th>
<th>MSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>WE-CONV-SEG</td>
<td>96.5</td>
<td>98.0</td>
</tr>
<tr>
<td>-word emb</td>
<td>96.1</td>
<td>97.6</td>
</tr>
<tr>
<td>-word feature</td>
<td>95.7</td>
<td>97.3</td>
</tr>
</tbody>
</table>

Table 7: Performances of our models with different word feature options. “-word emb” denotes the model in which word features and the vocabulary are used but the pretrained word embeddings are not. “-word feature” denotes the model that uses no word feature, i.e. CONV-SEG.

5 Related Work

CWS has been studied with considerable efforts in NLP community. Xue et al. (2003) firstly modeled CWS as a character-based sequence labeling problem. They used a sliding-window maximum entropy classifier to tag Chinese characters into one of four position tags, and then coverted these tags into a segmentation using rules. Following their work, Peng et al. (2004) applied CRF to the problem for sequence-level prediction. Recently, under the sequence labeling framework, various neural models have been explored for CWS. Zheng et al. (2013) firstly applied a feed-forward neural network for CWS. Pei et al. (2014) improved upon Zheng et al. (2013) by explicitly modeling the interactions between local context and previous tag. Chen et al. (2015a) proposed a gated recursive neural network (GRNN) to model the combinations of context characters. Chen et al. (2015b) utilized Long short-term memory (LSTM) to capture long distant dependencies. Xu and Sun (2016) combined LSTM and GRNN to efficiently integrate local and long-distance features.

Our proposed model is also a neural sequence labeling model. The difference from above models lies in that CNN is used to encode contextual information. CNNs have been successfully applied in many NLP tasks, such as text classification (Kalchbrenner et al., 2014; Kim, 2014; Zhang et al., 2015; Conneau et al., 2016), language modeling (Kim et al., 2016; Pham et al., 2016; Dauphin et al., 2016), machine translation (Meng et al., 2015; Kalchbrenner et al., 2016; Gehring et al., 2016). Experimental results show that the convolutional layers are capable to capture more n-gram features than previous introduced networks. Collobert et al. (2011) also proposed a CNN based sequence labeling model. However, our model is significantly different from theirs since theirs adopt max-pooling to encode the whole sentence into a fixed size vector and use position embeddings to demonstrate which word to be tagged while ours does not. Our model is more efficient due to the sharing structure in lower layers. Contemporary to this work, Strubell et al. (2017) applied dilated CNN to named entity recognition.

The integration with word embeddings is inspired by word-based CWS models (Andrew, 2006; Zhang and Clark, 2007; Sun et al., 2009). Most recently, Zhang et al. (2016); Liu et al. (2016); Cai and Zhao (2016) proposed word-based neural models for CWS. Particularly, Zhang et al. (2016); Liu et al. (2016) utilized word embeddings learned from large auto-segmented data, which leads to significant improvements. Different from their word-based models, we integrate word embeddings with the proposed character-based model.

Similiar to this work, Wang et al. (2011) and Zhang et al. (2013) also enhanced character-based CWS systems by utilizing auto-segmented data. However, they didn't use word embeddings, but only used statistics features. Sun (2010) and Wang et al. (2014) combined character-based and word-based CWS model via bagging and dual decomposition respectively and achieved better performance than single model.

6 Conclusion

In this paper, we address the weaknesses of character-based CWS models. We propose a novel neural model for CWS. The model utilizes stacked convolutional layers to derive contextual representations from input sequence, which are then fed to a CRF layer for prediction. The model is capable to capture rich n-gram features automatically. Furthermore, we propose an effective approach to integrate the proposed model with word embeddings, which are pretrained on large auto-segmented data. Evaluation on two benchmark datasets shows that without any feature engineering, much better performance than previous models (also without feature engineering) is obtained. Armed with word embeddings, our model achieves state-of-the-art performance on both datasets, without using any external labeled data.
Acknowledgements

This work is supported by the National Key Research & Development Plan of China (No.2013CB329302). Thanks anonymous reviewers for their valuable suggestions. Thanks Wang Geng, Zhen Yang and Yuanyuan Zhao for their useful discussions.

References


Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate sequence labeling with iterated dilated convolutions.


Xu Sun, Houfeng Wang, and Wenjie Li. 2012. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection.


Longkai Zhang, Houfeng Wang, Xu Sun, and Mairgup Mansur. 2013. Exploring representations from unlabeled data with co-training for chinese word segmentation.


