Closed-Set Chinese Word Segmentation Based on Convolutional Neural Network Model

Zhipeng Xie

School of Computer Science
Fudan University, Shanghai, China
xiezp@fudan.edu.cn

Abstract. This paper proposes a neural model for closed-set Chinese word segmentation. The model follows the character-based approach which assigns a class label to each character, indicating its relative position within the word it belongs to. To do so, it first constructs shallow representations of characters by fusing unigram and bigram information in limited context window via an element-wise maximum operator, and then build up deep representations from wider contextual information with a deep convolutional network. Experimental results have shown that our method achieves better closed-set performance compared with several state-of-the-art systems.

Keywords: Chinese word segmentation · Deep learning · Convolutional neural networks

1 Introduction

Chinese word segmentation (or CWS in short) is an fundamental task in Chinese information processing. It has to be performed before downstream syntactic and semantic analysis of Chinese text that has no explicit word delimiters. A lot of statistical methods have been proposed to solve the CWS problem [1, 12, 14, 15, 18].

Recently, with the upsurge of deep learning, there is a trend of applying neural network models to NLP tasks, which adaptively learn important features from word/character embeddings [2, 10] trained on large quantities of unlabelled text, and thus greatly reduce efforts of hand-crafted feature engineering [5].

Zheng et al. [19] made the first try to apply embedding-based neural model to Chinese word segmentation. They used unigram embeddings in local windows as input for a two-layer feedforward network to calculate tag scores for each character position. The final decision is made by a Viterbi-like decoder algorithm based on the predicted tag scores and the transition probabilities between tags, where the transition probabilities have explicitly modeled the strong dependencies between tags.

Mansur et al. [9] worked in a way similar to [19]. They proposed the feature-based neural network where features are represented as feature embeddings, and

used character bigram embeddings as additional features to improve segmentation.

Pei et al. [11] proposed a max-margin tensor neural network to explicitly model the interactions between history tags and context characters by exploiting tag embeddings and tensor-based transformation. It also adopts the window approach, which concatenates the embeddings of characters within the window. The concatenated vector is then fed into the next layer which performs linear transformation followed by an element-wise activation function such as tanh.

Ma and Hinrichs [8] proposed an embedding matching approach which models the matching between configurations and character-specific decisions. It makes decision by considering character-action combinations instead of atomic, character independent actions in [11].

The models described above build up shallow-presentations of characters from their fixed-size windows, which can be thought of as a single convolutional layer with limited receptive field. To solve this problem, Chen et al. [4] introduced the LSTM neural network to build feature representations for CWS, where the LSTM exploits input, output and forget gates to decide how to utilize and update the memory of previous information.

These character-based neural models described have achieved competitive word segmentation results. However, there still exist two main problems:

Firstly, the methods in [8, 11, 19] works with only a single convolutional layer (or equivalently, a feedforward layer on a fixed-size window), where the problem is that the information can not flow from one position to another. To facilitate the information flow, one choice is to represent history predictions (of previous character positions) as embeddings in tag scoring pha olyse, as done in [11] and [8], but this design decision is made at the cost of losing the ability of fully exploiting the current multi-core architecture. Another possibility is to model the complicated tag-tag interactions in the inference phase [4, 19], which, however, complicates the algorithmic framework and also deprives the model of the ability of modeling the tag-character interactions.

The second problem is how to compose the feature embeddings from its context. Methods based on windows of fixed size is rigid and have only limited context size. The LSTM layer has a potentially unbounded dependency range within their receptive field, but this comes with a computational cost as each state needs to be computed sequentially in both training and testing phases.

The solution we propose here is to make use of a deep convolutional network, which has a large, but not unbounded, receptive field. The information (inclusive of tag-related information) at the lower level can flow to adjacent positions and get composed adaptively into higher-level representations. Such an implicit modeling of tag-tag and tag-character interactions is more flexible. Another advantage is that the convolutional model is easy to process all the characters of a sentence in parallel during both training and testing, leading to highly-efficient segmentation. We have tested BiLSTM model on the commonly-used PKU and MSR datasets, and the computational cost is about 10 times higher than the model with four convolutional layers.

Furthermore, different from existing CWS systems which often make use of external linguistic resources such as unsupervised text corpus (or the character embeddings pretrained on large text corpus) and dictionary, our model work in a closed-set scenario, which can make us focus on the method itself.

2 The Proposed Model

The architecture of our model is briefly illustrated in Figure 1, which consists of three main component:

- *Shallow representation.* It first constructs a shallow representation for each character in the given sentence, by fusing the unigram and bigram information from its local context window.
- *Deep representation* It then adaptively constructs hierarchically deeper representations to combine the lower-level representations of each position, where information can flow between adjacent character representations. **Tag Scoring Module** It assigns tag scores to each character based on its deep representation.

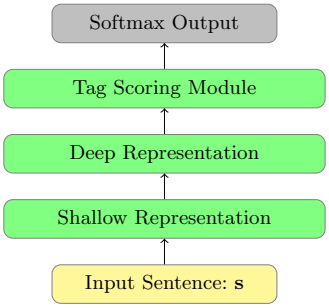

Fig. 1. The architecture for our model

Let $\mathbf{s} = c_1 \cdots c_{|\mathbf{s}|}$ denote the input Chinese sentence, where c_j is the j-th Chinese character in $\mathbf{s}$ and $|\mathbf{s}|$ is the length of the sentence. Each unigram (or character) in $\mathbf{s}$ can be simply represented as an embedding vector that is independent of its context. However, in Chinese word segmentation, the unigram embedding itself is usually not enough to determine the relative position of the character within the word it belongs to, because a specific character may appear at the beginning, in the middle or at the end of a word, depending on the particular context (or the surrounding characters).

2.1 Shallow Reprensentations

For each specific character c_j at position j in the input sentence $\mathbf{s}$, its shallow representation is a fixed-size vector which contains the important features fused

from the unigrams and the bigrams within a small context window. In this paper, the small context window is of size 3 by default.

As shown in Figure 2, the shallow representation $\mathbf{x}_j^{(0)}$ of a character c_j can be obtained by applying an elementwise max operator on its unigram shallow representation $\mathbf{x}_j^{(u)}$ and its bigram shallow representation $\mathbf{x}_j^{(b)}$, where $\mathbf{x}_j^{(u)}$ is the combination of three unigram embeddings of the characters c_{j-1}, c_j, and c_{j+1}, while $\mathbf{x}_j^{(b)}$ is the combination of two bigram embeddings of the bigram b_{j-1} and b_j. The details are described as follows.

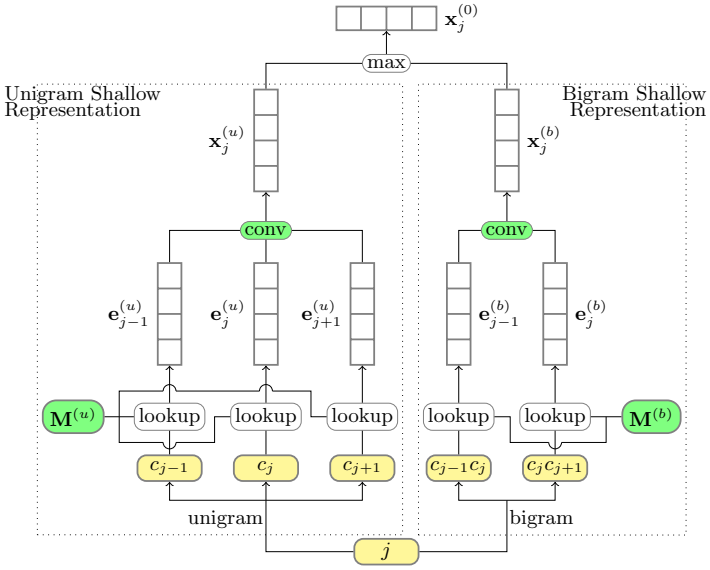

Fig. 2. Shallow representation for the character at position j

An input sentence $\mathbf{s}$ is normally represented as a sequence of unigrams $c_1 \cdots c_{|s|}$ where $|s|$ is the length of the sentence and c_i $(1 \leq i \leq |s|)$ is the i-th character in $\mathbf{s}$. A special character (\$ in this paper) can be used to denote the begin or the end of sentence, i.e. $c_0 = \$$ and $c_{|s|+1} = \$$. An alternative is to represent $\mathbf{s}$ as a sequence of bigrams $b_0 \cdots b_{|s|}$, where $b_i = c_i c_{i+1}$ $(0 \leq i \leq |s|)$ denotes the i-th bigram in $\mathbf{s}$. Here, the 0-th bigram $b_0 = \$c_1$ and the $|s|$-th bigram $b_{|s|} = c_{|s|}\$$. Please note that the sequence of unigrams and the sequence of bigrams are of different lengths.

Unigram Shallow Representations Let $\Sigma^{(u)}$ denote the unigram dictionary of size $|\Sigma^{(u)}|$, and $\mathbf{M}^{(u)}$ denote the (character) unigram embedding matrix of size $d_u \times |\Sigma^{(u)}|$, where d_u is the dimensionality of unigram embeddings (300 by default). Each character $c \in \Sigma^{(u)}$ has an associated index $ind(c)$ into the column of the embedding matrix.

At position j of the sentence $\mathbf{s}$, the unigram embedding $\mathbf{e}_j^{(u)} \in \mathbb{R}^{d_u \times 1}$ of c_j can be obtained by applying a lookup-table operation on $\mathbf{M}^{(u)}$:

$$\mathbf{e}_j^{(u)} = \text{lookup}(\mathbf{M}^{(u)}, c_j) = \mathbf{M}^{(u)} \cdot e_{ind(c_j)}$$

where $e_{ind(c_j)}$ is the one-hot representation of the character c_j, i.e. a $|\Sigma^{(u)}|$-dimensional binary column vector that is zero for all elements except for the element at the index $ind(c_j)$.

The shallow unigram representation of the character c_j can be composed by fusing the embeddings of its left unigram, its right unigram and itself. Specifically, it is calculated as:

$$\mathbf{x}_j^{(u)} = \mathbf{W}^{(us)} \left[\mathbf{e}_{j-1}^{(u)} \; \mathbf{e}_j^{(u)} \; \mathbf{e}_{j+1}^{(u)} \right] + \mathbf{b}^{(us)}$$

where $\mathbf{W}^{(us)} \in \mathbb{R}^{d_s \times d_u \times 3}$ is a 3-way tensor, $\mathbf{b}^{(us)} \in \mathbb{R}^{d_s}$ is the bias vector, and d_s is the dimensionality of the shallow representations (with 300 as default value). A convolutional layer with filter size 3, stride 1 and same-padding is used to efficiently compute all the shallow unigram representations.

Bigram Shallow Representations Similar to unigram embeddings, we also have a bigram dictionary (denoted by $\Sigma^{(b)}$) of size $|\Sigma^{(b)}|$, and a bigram embedding matrix (denoted by $\mathbf{M}^{(b)}$) of size $d_b \times |\Sigma^{(b)}|$. Each bigram $b \in \Sigma^{(b)}$ has an associated index $ind(b)$ into the column of $\mathbf{M}^{(b)}$. Similary, each bigram $b_j = c_j c_{j+1}$ $(0 \leq j \leq |s|)$ can be transformed into a bigram embedding:

$$\mathbf{e}_j^{(b)} = \mathbf{M}^{(b)} \cdot e_{ind(b_j)}$$

At position j, the shallow bigram represention of the character c_j can be composed by fusing the embeddings of its left bigram $b_{j-1} = c_{j-1}c_j$ and its right bigram $b_j = c_j c_{j+1}$, which c_j belongs to. In particular, the shallow bigram representation is defined as follows:

$$\mathbf{x}_j^{(b)} = \mathbf{W}^{(bs)} \left[\mathbf{e}_{j-1}^{(b)} \; \mathbf{e}_j^{(b)} \right] + \mathbf{b}^{(bs)}$$

where $\mathbf{W}^{(bs)} \in \mathbb{R}^{d_s \times d_b \times 2}$ is a 3-way tensor, $\mathbf{b}^{(bs)} \in \mathbb{R}^{d_s}$ is the bias vector. To efficiently compute all the shallow bigram representations, a convolutional layer with filter size 2, stride 1 and valid-padding is used.

Feature Fusion via Elementwise Maximum Merging Since our model have got unigram shallow representation and bigram shallow representation in parallel from the same input sentence to get, the next problem is how to fuse them into a single representation. One straightforward method is to concatenate them into a representation with double-sized dimensionality. However, it is

observed that such a concatenation leads to the phenomenon of overfitting. Instead, we fuse the shallow unigram and bigram representations via elementwise maximum merging:

$$\mathbf{x}_j^{(0)} = max\left(\mathbf{x}_j^{(u)}, \mathbf{x}_j^{(b)}\right)$$

It is expected that such an elementwise maximum operator could yield to a high-quality shallow feature representation.

In our model, both the unigram embedding and the bigram embedding matrices are both initialized randomly and get updated during the training phase, which makes our model a closed-set segmentor that does not rely on any external data or knowledge resource. In addition, the dimensionalities of the unigram embeddings, bigram embeddings and shallow representations are all set to 300 by default.

2.2 Deep Representations

The shallow representations have modeled the contextual information of characters, but the contextual information is limited on a small context window of size 3. To incorporate wider contextual information into the representations, we make use of a deep module that consists of multiple stacked convolutional layers, as illustrated in Figure 3. Let L to be the number of convolutional layers in the deep module ($L = 4$ by default)

Because the task at present is to perform character-based CWS, we have to preserve the temporal resolution throughout the module. Therefore, we make the following design choices:

- The filter size in each convolutional layer is set to a fixed integer S (by default, $S = 3$), with padding such that the temporal resolution is preserved;
- The stride is set to 1 in each convolutional layer (otherwise, the temporal resolution would be reduced);
- We do not use any down sampling (pooling layer) between adjacent convolutional blocks, because the functionality of pooling is to reduce the temporal dimensions.

Let L denote the number of convolutional layers in the deep module. The working mechanism of the l-th convolutional block ($1 \leq l \leq L$) is described below:

- A convolutional layer with F filters is performed by taking the dot-product between each filter (or kernel) matrix and each window of size S in the input sequence $\mathbf{x}^{(l-1)}$, resulting in F scalar values for each position j in input sentence. By default, the value of F is set to 600 for all convolutional layers in this module.
- Next, a element-wise ReLU activation function is applied to make nonlinear transformation, so negative activations are discarded.

Stacking such L convolutional layers together results in a receptive field of $((S-1)\times L+1)$ positions of the original input sentence. The receptive field of the units in the deeper layers of a convolutional network is larger. Deep neural networks can adaptively learn how to best combine the lower-level representations

of S positions into a higher-level representation in a hierarchically layer-by-layer manner.

One simplest way is to use the output $\mathbf{x}^{(L)}$ from the L-th layer as the final deep representation. But we adopt another way in this paper: the final deep representation is calculated as the elementwise summation of the outputs from all the L layers in the deep module:

$$\mathbf{r} = \sum_{l=1}^{L} \mathbf{x}^{(l)}$$

where the elementwise summation has some sense of short-cut connections. Please note that it is only a problem of design choice, and both of them have similar segmentation performance.

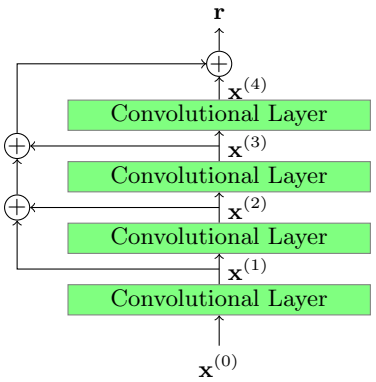

Fig. 3. A deep module of 4 convolutional layers

2.3 Tag Scores

After the final deep representations have been calculated, each character c_j ($1 \leq j \leq |\mathbf{s}|$) is now represented as an F-dimensional vector $\mathbf{r}_j$. The next step is to transform the deep representation $\mathbf{r}_j$ into a K-dimensional vector of tag scores $\mathbf{y}_j$, where the tag set adopted here is {'B', 'M', 'E', 'S'}, and hence $K = 4$. In the tag set, 'S' denotes a single character word, while 'B', 'M' and 'E' denotes the begin, middle and end of a multi-character word respectively.

To implement this transformation, our model uses a two-layer feed-forward neural network:

$$\mathbf{y}_j = f_2\left(g\left(f_1\left(\mathbf{r}_j\right)\right)\right)$$

where f_2 and f_1 are two affine transformations, and g is a element-wise ReLU activation.

Specifically, we have:

$$\mathbf{h}_j = \text{ReLU}\left(\mathbf{W}^{(s,1)} \cdot \mathbf{r}_j + \mathbf{b}^{(s,1)}\right)$$

and

$$\mathbf{y}_j = \mathbf{W}^{(s,2)} \cdot \mathbf{h}_j + \mathbf{b}^{(s,2)}$$

where $\mathbf{W}^{(s,1)}$ is a matrix of size $F \times F$, $\mathbf{b}^{(s,1)}$) is a vector of size F, $\mathbf{W}^{(s,2)}$ is a $F \times K$ matrix, and $\mathbf{b}^{(s,2)}$ is a vector of size K.

2.4 Dropout

Dropout is an effective technique to regularize neural networks by randomly drop units during training. It has achieved a great success when working with feed-forward networks [13], convolutional networks, or even recurrent neural networks [16].

In our model, dropout is applied to both the output of shallow representations and the input of the final layer in the deep representation module, which sets the values of units to zero with the same dropout rate (set to 0.6 as default value). We call it the the technique $dropout_{normal}$, to distinguish from the following technique of $dropout_{block}$.

Besides the normal dropout technique, we also use another $dropout_{block}$ technique to make the model robust to unknown character unigrams or bigrams (OOVs). It drops unigrams and bigrams according to their frequencies in the training data. More specifically, the probability that a unigram c gets dropped is:

$$p_{blockdrop}(c) = \frac{M_u}{M_u + freq(c)}$$

and the probability to drop out a bigram b is

$$p_{blockdrop}(b) = \frac{M_b}{M_b + freq(b)}$$

where $freq(\cdot)$ denotes the frequency number of a unigram or bigram in the dataset, M_u and M_b are two positive numbers (set as 30 and 60 by default). Clearly, a unigram or bigram is more likely to be dropped out, if it appears less frequently in the training corpus.

2.5 Tag Prediction and Word Segmentation

Given the tag scores for a position j, the prediction $\hat{t}_j$ is the tag with the highest predicted tag score:

$$\hat{t}_j = \arg\max_k y_{j,k}$$

where $y_{j,k}$ is the predicted score of tag k at position j.

After all positions have their tags predicted, the sentence is segmented in a simple heuristic way: A character with tag 'B' or 'S' will start a new word,

while a character with tag 'M' or 'E' will append itself to the previous word. As a result, the potential inconsistencies in predicted tags are resolved in a near-random manner. For example, the inconsistent adjacent predictions "BMB" will be implicitly changed to "BEB", "BBS" to "BES", etc.

2.6 Model Training

Given the training sentences and ground truth $\{\mathbf{s}_i, \mathbf{t}_i\}_{i=1}^{N}$, our goal is to learn the parameters that minimize the cross-entropy loss function:

$$\mathbf{L}(\Theta) = \frac{1}{\sum_{i=1}^{N} |\mathbf{s}_i|} \sum_{i=1}^{N} \sum_{j=1}^{|\mathbf{s}_i|} \log \frac{\exp y_{i,j,t_{i,j}}}{\sum_k \exp y_{i,j,k}}$$

where Θ is the set of all parameters, $t_{i,j}$ denotes the gold tag for the position j in sentence $\mathbf{s}_i$, $y_{i,j,k}$ denotes the score of tag k for the position j in $\mathbf{s}_i$.

Here, as a rule of thumb, we do not include a L2-regularization term in the loss function because dropout has been used to regularize our model.

We used Adam [7] to train our models with a learning rate of 0.0005, a first momentum coefficient $\beta_1 = 0.9$, and a second momentum coefficient $\beta_2 = 0.999$. Each model was trained for 50 epochs with minibatch size of 16 sentences.

3 Experiments

Datasets. To evaluate our model, we used two widely used benchmark datasets, PKU and MSR, provided by the Second SIGHAN International Chinese Word Segmentation Bakeoff[1] [6].The segmentation results are evaluated by the *F-score*.

To make the comparison fair, we converted the Arabic numbers and English characters in the testing set of PKU corpus from half-width form to full-width form, because they are in full-width form in the training set. This conversion is commonly performed before segmentation in related research work. Except this conversion, we did not make any preprocessing on the datasets.

3.1 Results

Table 1 lists the closed-set results of our neural model, together with the best closed-set results in 2nd SIGHAN bakeoff (Best05) and several state-of-the-art neural models, on PKU and MSR datasets. It can be easily seen that our model has achieved the best performance among all the neural models in closed-set settings.

Table 2 summarizes the closed-set results of our model and the open-set results of several state-of-the-art neural CWS systems. It is astonishing that our model without using any pretrained embeddings has also achieved the best performance even when compared with the state-of-the-art neural systems that make use of various pretrained unigram, bigram or word embeddings.

[1] http://sighan.cs.uchicago.edu/bakeoff2005/

Table 1. Comparison of closed-set F-score with other closed-set neural CWS systems

Models	PKU	MSR
Best05 (closed-set)	95.0	96.4
Zheng et al. [19] (closed-set)	92.4	93.3
Pei et al. [11] (closed-set)	93.5	94.4
Ma and Hinrichs [8] (closed-set)	95.1	96.6
Cai and Zhao [3] (closed set)	95.2	96.4
Our model (closed set)	**95.6**	**97.4**

Table 2. Comparison of closed-set F-score with other open-set neural CWS systems

Models	PKU	MSR
Zheng et al. [19]+*pretraining*	92.8	93.9
Pei et al. [11]+*pretraining*	94.0	94.9
Pei et al. [11]+*pretraining & bigram*	95.2	97.2
Cai and Zhao [3]+*pretraining*	95.5	96.5
Zhang et al. [17] (*with pretrained unigram, bigram and word embeddings*)	95.1	97.0
Our model (closed-set	**95.6**	**97.4**

In addition, when we replace the deep convolutional module with a BiLSTM module, it achieves similar F-score on both the datasets, but runs about 10 times slower on the same desktop with a Nvidia GTX1080Ti graphics card. Both of them are implemented with Tensorflow in Python.

3.2 Ablation Analysis

Table 3. Ablation analysis of our model on PKU dataset

Models	PKU
Our model	**95.6**
$-bigram$	95.4
$-deepmodule$	93.5
$-dropout_{normal}$	95.3
$-dropout_{block}$	94.5
$-$(both $dropout_{normal}$ and $dropout_{block}$)	94.3

In our model, there are three main working techniques: bigram, deep module, and dropout. To investigate their contributions, we removed each of them from the model, the results are shown in Table 3. In addition, we also consider the contributions of the two distinct dropout techniques, $dropout_{normal}$ and $dropout_{block}$, respectively.

It can be easily seen that all the main techniques have their own contribution to the performance of our model. Among all these techniques, the $dropout_{normal}$ and the *bigram* are relatively weak, while *deepmodule* and $dropout_{block}$ are more important. It can also be observed that $dropout_{block}$ has more contribution than $dropout_{normal}$, but they are complementary to each other.

4 Conclusion

In this paper, we propose a novel neural model for CWS, which is based on convolutional architecture. It uses an elementwise maximum operator to fuse the unigram and bigram features from a local context window into shallow representations, and then builds up deeper and deeper representations adaptively via a deep convolutional network. Experiments on two commonly-used datasets PKU and MSR have shown that our closed-set model has better performance, not only than several closed-set neural methods, but also than several open-set state-of-the-art neural models.

Our model is different from the existing neural ones in several aspects:

- The model makes use of deep convolutional network for CWS, where the receptive fields are sufficiently large for CWS, and the information at each position can flow to its adjacent positions bi-directionally. In comparision, existing methods that work on the basis of traditional window-based segmentation [8, 11, 19] have only limited receptive field.
- Recent approaches that make use of recurrent neural networks (typically, LSTM) have potentially infinite receptive field in building representations for characters in an input sentence. However, the sequential processing mechanism of recurrent neural networks makes it too costly to build hierarchical representations. Instead, the deep convolutional network is suitable for the exploitation of modern multi-core computation ability such as GPU.

Finally, as future work, it is possible to integrate external resources, such as pretrained unigram and bigram embeddings or domain lexicons, into our model, which is expected to achieve a better open-set segmentation performance.

Acknowledgments

This work is supported by National High-Tech R&D Program of China (863 Program) (No. 2015AA015404), and Science and Technology Commission of Shanghai Municipality (No. 14511106802). We are grateful to the anonymous reviewers for their valuable comments.

References

1. Andrew, G.: A hybrid markov/semi-Markov conditional random field for sequence segmentation. In: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing. pp. 465–472 (2006)

2. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. Journal of machine learning research 3, 1137–1155 (2003)
3. Cai, D., Zhao, H.: Neural word segmentation learning for Chinese. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 409–420. Association for Computational Linguistics, Berlin, Germany (August 2016)
4. Chen, X., Qiu, X., Zhu, C., Liu, P., Huang, X.: Long short-term memory neural networks for Chinese word segmentation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1197–1206 (2015)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. The Journal of Machine Learning Research 12, 2493–2537 (2011)
6. Emerson, T.: The second international Chinese word segmentation bakeoff. In: Proceedings of the Second SIGHAN Workshop on Chinese Language Processing. pp. 123–133 (2005)
7. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
8. Ma, J., Hinrichs, E.: Accurate linear-time Chinese word segmentation via embedding matching. In: Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics. pp. 1733–1743 (2015)
9. Mansur, M., Pei, W., Chang, B.: Feature-based neural language model and Chinese word segmentation. In: Proceedings of IJCNLP. pp. 1271–1277 (2013)
10. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems (NIPS). pp. 3111–3119 (2013)
11. Pei, W., Ge, T., Chang, B.: Max-margin tensor neural network for Chinese word segmentation. In: ACL (1). pp. 293–303 (2014)
12. Peng, F., Feng, F., McCallum, A.: Chinese segmentation and new word detection using conditional random fields. In: Proceedings of Coling. pp. 562–568 (2004)
13. Srivastava, N.: Improving neural networks with dropout. Ph.D. thesis, University of Toronto (2013)
14. Tseng, H., Chang, P., Andrew, G., Jurafsky, D., Manning, C.: A conditional random field word segmenter for SIGHAN bakeoff 2005. In: Proceedings of the fourth SIGHAN workshop on Chinese language Processing. pp. 168–171 (2005)
15. Xue, N., Shen, L.: Chinese word segmentation as LMR tagging. In: Proceedings of the second SIGHAN workshop on Chinese language processing - Volume 17. pp. 176–179 (2003)
16. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv preprint arXiv:1409.2329 (2014)
17. Zhang, M., Zhang, Y., Fu, G.: Transition-based neural word segmentation. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (volume 1: Long Papers). pp. 421–431. Association for Computational Linguistics, Berlin, Germany (August 2016)
18. Zhang, Y., Clark, S.: Chinese segmentation with a word-based perceptron algorithm. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics. pp. 840–847 (2007)
19. Zheng, X., Chen, H., Xu, T.: Deep learning for Chinese word segmentation and POS tagging. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 647–657 (2013)