A Deep Convolutional Neural Model for Character-Based Chinese Word Segmentation

Zhipeng Xie and Junfeng Hu

Shanghai Key Laboratory of Data Science
School of Computer Science, Fudan University, China
{xiezp, 15210240075}@fudan.edu.cn

Abstract. This paper proposes a deep convolutional neural model for character-based Chinese word segmentation. It first constructs position embeddings to encode unigram and bigram features that are directly related to single positions in input sentence, and then adaptively builds up hierarchical position representations with a deep convolutional net. In addition, a multi-task learning strategy is used to further enhance this deep neural model by treating multiple supervised CWS datasets as different tasks. Experimental results have shown that our neural model outperforms the existing neural ones, and the model equipped with multi-task learning has successfully achieved state-of-the-art F-score performance for standard benchmarks: 0.964 on PKU dataset and 0.978 on MSR dataset.

1 Introduction

Chinese, as well as most east Asian languages, is written without explicit word delimiters. Therefore, *word segmentation* becomes a preliminary but fundamental procedure that has to be executed before miscellaneous downstream syntactic and semantic analysis. In the past two decades, many statistical methods have been proposed to solve the problem of Chinese word segmentation (or CWS in short), which can be categorized roughly into character-based and word-based approaches.

The character-based models [17] treat word segmentation as a sequence labeling problem, where tags are used to indicate the relative position of characters inside the words that they belong to. Tag prediction is usually made on the basis of extracted features from local windows, inclusive of character identity features, reduplication features [16], and history predictions of previous characters. One disadvantage of character-based models exists in that they can only use limited contextual information, and other valuable contextual information such as surrounding words is hard to get included. To allow word information to be used as features, word-based approaches [1, 21] were proposed to rank candidate segmented outputs directly, with the aid of semi-CRF or transition-based methods where word-level features can be easily integrated.

These conventional statistical CWS methods, no matter character-based or word-based, rely heavily to a large extent on the hand-crafted features. 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 [2,12] trained on large quantities of unlabelled text, and thus greatly reduce efforts of hand-crafted feature engineering [5]. Following the trend, several neural models for word segmentation have been developed, among which some are character-based [4,10,11,13,22], and others are word-based [3,9,20]. These neural models have achieved competitive performance.

This paper follows the character-based approach, which has a simpler algorithmic framework and prevails in the task of CWS. The contribution of this paper is three-fold: (1) *Deep* convolutional network is applied to CWS, where position embeddings are automatically constructed from the unigram and bigram features directly related to positions (Section 2.1), and hierarchical position representations are built up adaptively from position embeddings such that the representation of each position has a large receptive field (Section 2.2). (2) To utilize the large-scale unsupervised text corpus, character bigram embeddings that are pretrained on unsupervised corpus play an important role in the construction of position embeddings (Section 2.1). The pretrained bigram embeddings can be thought of features at the midpoint from characters to words; (3) We designed a homogeneous multitask learning framework for Chinese word segmentation (Section 3), which treats each supervised dataset a different separate task. It is shown that the segmentation model built for one domain can benefit from the dataset from other domain.

2 The Deep Convolutional Neural Model for CWS

The deep convolutional neural model (or **DCN** model in short) can be thought of as an instantiation of a simple and straightforward deep neural architecture as shown in Figure 1. The overall architecture takes a modular structure consisting of three main modules, one stacked over another. **Position Embedding Module** constructs a vector representation for each position in input sentence, called *position embedding*, which is expected to encode the information that are directly related to the position. **Deep Representation Module** adaptively constructs hierarchical *position representations* to combine the lower-level representations of each position and its surrounding positions into a higher-level representation of the position. **Tag Scoring Module** assigns tag scores to each position based on the top-level position representation from the deep representation module.

The DCN model proposed in this paper is different from the existing neural ones in several aspects:

– *Deep* convolutional network is used for CWS, which is able to automatically construct best hierarchical representations for positions in sentence. By convolution, the information at a position can flow to its adjacent positions bi-directionally. Deep convolutional network can provide sufficiently large receptive fields. In contrast, most previous methods rely on shallow networks that have only limited receptive field and work on the basis of traditional window-based segmentation [10,13,22]

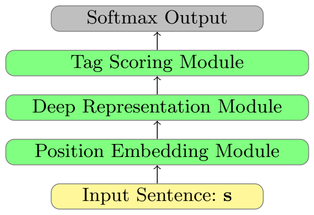

Fig. 1. Deep neural network architecture for CWS

- Recent approaches make use of recurrent neural networks (typically, LSTM), which have potentially infinite receptive field, to build the feature representation for positions in an input senntence. However, the sequential processing mechanism of recurrent neural networks makes it too costly to build hierarchical representations. On the contrary, the deep convolutional network is suitable for the exploitation of modern multi-core computation ability such as GPU.
- To model the tag-tag or tag-character interactions, some models explicitly designed tag-related features, while others explicitly used tag-tag transition probabilities in a viterbi-like decoder. In our model, it is assumed that the induced position representations have naturally contains information about the tags for positions, and therefore, the tag-related information can easily flow implicitly between adjacent positions during their hierarchical representation learning process.

2.1 Position Embeddings

For each position j $(1 \leq j \leq n)$ in a given input sentence $\mathbf{s} = c_1 \cdots c_n$ of length n, we would like to first represent it as a vector $\mathbf{x}_j^{(0)}$ of fixed size, called *position embedding*. To distinguish it from *position representation* in Section 2.2, it is required that a position embedding is constructed from features that are *directly* related to the position, while a position representation is obtained by aggregating features from the position and its surroundings.

As shown in Figure 2, the position embedding $\mathbf{x}_j^{(0)}$ at position j is the concatenation of its unigram embedding $\mathbf{e}_j^{(u)}$ and its bigram embedding $\mathbf{e}_j^{(b)}$.

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

Given a sentence of n characters, $\mathbf{s} = c_1c_2 \ldots c_n$, where c_j is the character unigram at position j $(1 \leq j \leq n)$. The unigram embedding $\mathbf{e}_j^{(u)} \in \mathbb{R}^{d^{(u)} \times 1}$ at position j can be obtained by applying a lookup-table operation on the unigram

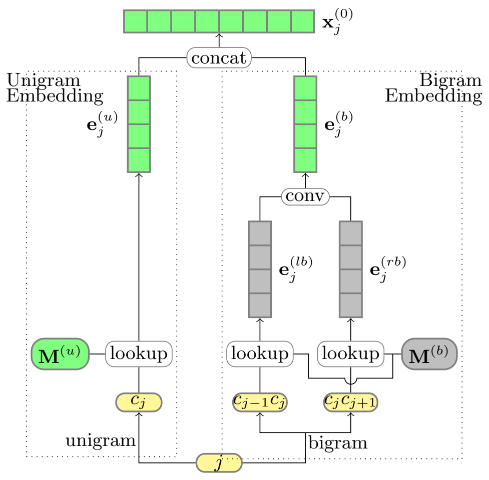

Fig. 2. Module for position embedding.

embedding matrix:

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

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 $ind(c_j)$.

In our model, the character embedding matrix $\mathbf{M}^{(u)}$ is randomly initialized, and it will get trained via back propagation.

Bigram Embeddings Besides the unigram dictionary, we also have a character bigram dictionary (denoted by $\Sigma^{(b)}$) of size $|\Sigma^{(b)}|$, and a character bigram embedding matrix (denoted by $\mathbf{M}^{(b)}$) of size $d^{(b)} \times |\Sigma^{(b)}|$. In our model, the character bigram embeddings are pretrained on unsupervised corpus, and the dimensionality of these embeddings is set to 300 as default. Each bigram $cc' \in \Sigma^{(b)}$ has an associated index $ind(cc')$ into the column of the bigram embedding matrix.

Each position j in the input sentence $\mathbf{s}$ is directly related to two character bigrams: the adjacent left character bigram $c_{j-1}c_j$ and the adjacent right character bigram c_jc_{j+1}, which can be transformed into the left character bigram embedding $\mathbf{e}_j^{(lb)}$ and the right character bigram embedding $\mathbf{e}_j^{(rb)}$ by looking-up the bigram embedding matrix:

$$\mathbf{e}_j^{(lb)} = \mathbf{M}^{(b)} \cdot e_{ind(c_{j-1}c_j)}$$

$$\mathbf{e}_j^{(rb)} = \mathbf{M}^{(b)} \cdot e_{ind(c_jc_{j+1})}$$

where $e_{ind(cc')}$ is a $|\Sigma^{(b)}|$-dimensional binary column vector that is zero for all elements except for the element at the index $ind(cc')$.

A convolutional layer with filter size 2, stride 1 and zero-padding is then used to get the *position bigram embedding* $\mathbf{e}_j^{(b)}$ for each position j from its left and right character bigram embeddings. Formally,

$$\mathbf{e}_j^{(b)} = \mathbf{W}^{(0)} \left[\mathbf{e}_j^{(lb)} \ \mathbf{e}_j^{(rb)} \right] + \mathbf{b}^{(0)}$$

where $\mathbf{W}^{(0)} \in \mathbb{R}^{d^{(0)} \times d^{(b)} \times 2}$ is a 3-way tensor, $\mathbf{b}^{(0)} \in \mathbb{R}^{d^{(0)}}$ is the bias vector, and $d^{(0)}$ is the dimension of the position bigram embedding space (with 300 as default value).

The use of pretrained bigram embeddings in CWS is based on the following consideration:

– If position j is in the middle of a multi-character word, it is likely that $c_{j-1}c_j$ and c_jc_{j+1} share similar contexts in the unsupervised corpus, and thus have similar embeddings to some degree.
– If position j is at the begin (or end) of a multi-character word, it is unlikely that $c_{j-1}c_j$ and c_jc_{j+1} share similar contexts and thus their embeddings are dissimilar to each other.

2.2 Deep Representation Module

To build up the hierarchical feature representation for character-based CWS, we adopt a deep convolutional representation module which consists of multiple stacked convolutional blocks.

Each convolutional block is a weighted layer followed by a ReLU nonlinearity activation, as shown in Figure 3.

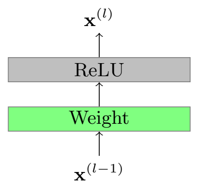

Fig. 3. Structure of a convolutional block

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 block 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 block (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 blocks in the 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 batch normalization layer goes immediately after the convolutional layer. It normalizes the output of the convolutional layer to zero mean and unit variance, and then transforms it linearly.
– Finally, a element-wise ReLU activation function is applied, so negative activations are discarded.

Stacking L layers of such convolution blocks 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 than the receptive field of the units in the shallow layers. 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.

2.3 Tag Scoring Module

After processed by the deep convolutional network, each position j is represented by a vector $\mathbf{x}_j^{(L)}$ of size F. The tag scoring module transforms each position representation from a F-dimensional vector $\mathbf{x}_j^{(L)}$ into a K-dimension vector of tag scores $\mathbf{y}_j$. The tag set used is {'B', 'M', 'E', 'S'}, and hence $K = 4$, where 'S' denotes a single character word, while 'B', 'M' and 'E' denotes the begin, middle and end of a multi-character word respectively.

A two-layer feed-forward neural network implements the module:

$$\mathbf{y}_j = f_2\left(g\left(f_1\left(\mathbf{x}_j^{(L)}\right)\right)\right)$$

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

Specifically, we have:

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

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 [14], convolutional networks, or even recurrent neural networks [18]. In the DCN model, dropout is applied to both the output of point embedding module and the input of the final layer in the deep representation module, with the same dropout rate.

Furthermore, in order to make the model robust to unknown character unigrams or bigrams, it also drops a position randomly 20% of the time during training.

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:

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

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 both dropout and batch normalization have been used to regularize our model.

We used Adam [7] to train our models with a learning rate of 0.001, 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 Multi-task Learning

Multi-task learning usually can obtain better generalization ability by making part of a model be shared across tasks. Previous related work mainly focused on jointly modeling heterogeneous multi-tasks such as word segmentation, POS tagging or dependency parsing [8].

Here, we are concerned about homogeneous multi-tasks, all for word segmentation. Suppose that there are multiple supervised datasets for CWS, it is not a good idea to merge them into a single large one, because of the following reasons:

- Different datasets may be used for different NLP tasks, while different NLP tasks may require different segmentation criteria.
- Different datasets may be annotated according to different segmentation standards. For example, the PKU dataset used a standard derived from the Chinese government standard for text segmentation in computer application, while the MSR dataset was segmented according to Microsoft's internal standard.

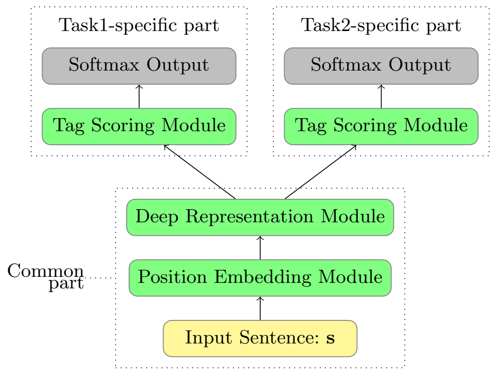

Fig. 4. MT-DCN: A multi-task model for CWS.

Instead, we treat each dataset as a separate task and propose a multi-task model for CWS (called **MT-DCN**) in Figure 4. For illustration, only two CWS datasets are used here. The common part at the bottom of the model is shared by the two CWS tasks, and as a result, the unigram and bigram embedding matrices and the deep convolutional network for build up hierachical position representations (together with the corresponding parameter) are all shared by the two tasks. Besides this shared common part, the two tasks have their own specific parts at the top of the model, which are different from each other. The task-specific part is expected to encode the special factors of the corresponding task.

At training time, we divide the two training datasets into a same number of minibatches (1500 minibatches by default), and alternately update one task-specific part with a minibatch from the corresponding dataset. Note that the common part is always updated for each minibatch, no matter which dataset it comes from. At prediction time, which task-specific part to use depends on where the data comes from.

4 Experiments

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

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.

We implemented the models in Python with Theano (`http://deeplearning.net/software/theano/`) and Lasagne (`http://lasagne.readthedocs.io`). We used *word2vec* to derive the embeddings of character bigrams pretained on Chinese Wikipedia corpus[1].

4.1 Empirical Comparison with Other Models

Table 1. Comparison of F-scores with other state-of-the-art CWS systems.

Neural Models	PKU	MSR	**Non-neural Model**	PKU	MSR
Zheng et al. [22]	92.8	93.9	Zhang and Clark [21]	95.1	97.2
Pei et al. [13]	95.2	97.2	Sun et al. [15]	95.4	97.4
Ma and Hinrichs [10]	95.1	96.6	Zhang et al. [19]	96.1	97.4
Cai and Zhao [3]	95.5	96.5			
Zhang et al. [20]	95.7	97.7			
Liu et al. [9]	95.7	97.6			
Our models					
DCN	95.9	97.7	**MT-DCN**	**96.4**	**97.8**

Table 1 summarizes the F-scores of DCN and MT-DCN on the testing datasets of PKU and MSR. Compared with state-of-the-art neural or non-neural models, we have the following observations:

– The DCN model substantially outperforms the other existing character-based neural models inclusive of [22], [13], and [10]. It also obtains observable higher F-score than the word-based neural models inclusive of [3], [20], and [9]. In addition, it outperforms two cutting-edge non-neural models of [21] and [15], and is competitive to the model of [19];
– The MT-DCN model achieves the best performance on both datasets among all the compared models, neural or non-neural. With multi-task strategy, the F-score improvement on PKU is more than that on MSR. Possible reason is that the MSR dataset is much larger.

[1] https://dumps.wikimedia.org/zhwiki/20161120/zhwiki-20161120-pages-articles.xml.bz2

4.2 Learning Curves of DCN Model

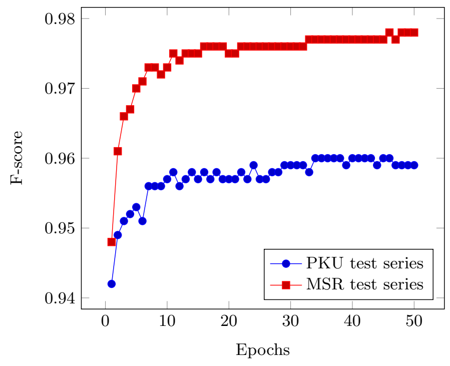

Fig. 5. The learning curve of our DCN model

Figure 5 shows that the training precedure converges quickly. After the first epoch, the F-score is 94.2 on PKU testing set, and 94.8 on MSR testing set. Just after 10 epoches, the F-scores have been very near to their best.

4.3 Contributions of Techniques

Table 2. Technique contributions in MT-DCN

Model	Precision	Recall	F-measure
MT-DCN model	96.7	96.1	96.4
- Mulitask(msr)	96.3	95.6	95.9
- Bigrams	96.5	95.6	96.0
- Dropout	96.5	95.9	96.2

In MT-DCN model, there are three main working techniques: multi-task, pretrained bigram embeddings, and dropout. To investigate their contributions, we removed each of them from the model, the results are shown in Table 2. The dropout is relatively weak, and multitask and pretrained bigram are comparable in effect. Conclusion can be drawn that data and features are most crucial to CWS.

4.4 Shallow versus Deep Representations

To investigate the effect of the deep convolutional network used in the deep representation module, we replaced it with a shallow one with only a single convolutional layer. With the filter size varying (or equivalently, the size of receptive

Table 3. Comparison of shallow representations and deep representations on PKU dataset.

	Size of receptive field				
	3	5	9	13	17
Shallow	95.3	**95.5**	95.3	95.1	95.2
Deep	95.3	95.7	95.8	95.8	**95.9**

field), the F-scores on PKU testing data are shown in the "**Shallow**" row. It can be observed that the best performance occurs when the filter size is set to 5, which coincides with the fact that most previous neural models claimed the best context window size to be 5. The "**Deep**" row lists the results of our DCN model with varying number of convolutional layers in $\{1, 2, 4, 6, 8\}$, with corresponding receptive field size in $\{3, 5, 9, 13, 17\}$. Observable improvement is still made when the model is going deeper from 6 to 8. Therefore, the use of deep network has gained about 0.4 (from 95.5 to 95.9) of F-score over the shallow network.

5 Conclusion

In this paper, we propose a deep convolutional neural model for Chinese word segmentation. It uses position embeddings to encode the information directly related to single positions, and then builds up hierarchical position representations adaptively with a deep convolutional network. The model outperforms the other state-of-the-art neural models. In addition, a multi-task strategy is used to enhance the neural model, which improves the performance furtherly.

Acknowledgments This work is supported by National High-Tech R&D Program of China (863 Program) (No. 2015AA015404). 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 (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 Fourth 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. Kruengkrai, C., Uchimoto, K., Kazama, J., Wang, Y., Torisawa, K., Isahara, H.: An error-driven word-character hybrid model for joint Chinese word segmentation and POS tagging. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing. pp. 513–521 (2009)
9. Liu, Y., Che, W., Guo, J., Qin, B., Liu, T.: Exploring segment representations for neural segmentation models. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. pp. 2880–2886 (2016)
10. 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)
11. Mansur, M., Pei, W., Chang, B.: Feature-based neural language model and Chinese word segmentation. In: Proceedings of IJCNLP. pp. 1271–1277 (2013)
12. 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)
13. Pei, W., Ge, T., Chang, B.: Max-margin tensor neural network for Chinese word segmentation. In: ACL (1). pp. 293–303 (2014)
14. Srivastava, N.: Improving neural networks with dropout. Ph.D. thesis, University of Toronto (2013)
15. Sun, X., Wang, H., Li, W.: Fast online training with frequency-adaptive learning rates for Chinese word segmentation and new word detection. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 253–262 (2012)
16. 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)
17. 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)
18. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv preprint arXiv:1409.2329 (2014)
19. Zhang, L., Wang, H., Sun, X., Mansur, M.: Exploring representations from unlabeled data with co-training for Chinese word segmentation. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 311–321 (2013)
20. 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 (2016)
21. 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)
22. 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)