





Improved Character-Based Chinese Dependency Parsing by Using Stack-Tree LSTM

Hang Liu , Mingtong Liu, Yujie Zhang , Jinan Xu,
and Yufeng Chen

School of Computer and Information Technology,
Beijing Jiaotong University, Beijing, China
{16120389, 16112075, yjzhang, jaxu, chenylf}@bjtu.edu.cn

Abstract. Almost all the state-of-the-art methods for Character-based Chinese dependency parsing ignore the complete dependency subtree information built during the parsing process, which is crucial for parsing the rest part of the sentence. In this paper, we introduce a novel neural network architecture to capture dependency subtree feature. We extend and improve recent works in neural joint model for Chinese word segmentation, POS tagging and dependency parsing, and adopt bidirectional LSTM to learn n-gram feature representation and context information. The neural network and bidirectional LSTMs are trained jointly with the parser objective, resulting in very effective feature extractors for parsing. Finally, we conduct experiments on Penn Chinese Treebank 5, and demonstrate the effectiveness of the approach by applying it to a greedy transition-based parser. The results show that our model outperforms the state-of-the-art neural joint models in Chinese word segmentation, POS tagging and dependency parsing.

Keywords: Chinese word segmentation
POS tagging and dependency parsing · Dependency subtree
Neural network architecture

1 Introduction

Transition-based parsers [1–4] have been shown to be both fast and efficient for dependency parsing. These dependency parsers can be very accurate for languages that have natural separators such as blanks between words, but for Chinese that do not contain natural separators, these parsers maybe get worse.

One reason for the lower accuracy of Chinese dependency parser is error propagation: Chinese dependency parsing requires word segmentation and POS tagging as pre-processing steps; once the pipeline model makes an error in word segmentation, more errors are likely to follow. In order to address the issue, transition-based Chinese word segmentation, POS tagging and dependency parsing joint model are proposed, jointly learning the three tasks [5–8]. Modern approaches to joint model can be broadly categorized into feature engineering joint model and neural joint model. The feature engineering joint model [5–7] needs to manually define a large number of feature

templates, and extracts the features from feature templates. The neural joint model [8] automatically extracts features by neural network such as RNN or LSTM, and then uses a small number of feature templates for model parsing and decision. These models perform better than pipeline models, but they ignore the complete dependency subtree feature, which has been proven to be an effective information for improving model performance in previous works [9–11].

In this paper, to improve Character-based dependency parsing, we extend the work of Kurita [8] using bidirectional LSTMs to learn n-gram feature and context information, and introduce a novel neural network architecture to encode the built dependency subtrees information, which use richer information and avoiding feature engineering. The neural network architecture is a stack structure combined with LSTM cell and Tree LSTM cell, called Stack-Tree LSTM, which can capture all the built dependency subtrees information. Then the subtree feature and n-gram feature are fed into a neural network classifier to make parsing decisions within a transition-based dependency parsing.

In the experiments, we evaluate our parser on the CTB-5 dataset and experimental results show that F1 scores of the Chinese word segmentation, POS tagging and dependency parsing reach 97.78%, 93.51% and 79.66% respectively, which are better than the baseline model in each task.

2 Related Work

In Chinese, the character-based dependency parsing solution was first proposed in Hatori [5]. He assumed that there was a dependency between the characters in the internal words, and unified the three tasks in one framework. The benefit of the solutions is that it can start with the character-level, and Chinese word segmentation, POS tagging and dependency parsing can be done in a joint framework, which improves the accuracies of three tasks and does not suffer from the error propagation. Zhang [6] studied the character-based Chinese dependency parsing by using pseudo and annotated word structures, and obtained better accuracies on three tasks. Moreover, they further analyzed some important factors for intra-word dependencies and demonstrated that intra-word dependencies can improve the performance of the three tasks. Guo [7] proposed a solution to transform the conventional word-based dependency tree into character-based dependency tree by using the internal structure of words and proposed a semi-supervised joint model for exploiting 2-g string feature and 2-g dependency subtree feature.

These methods achieve high accuracy on three tasks but rely heavily on feature engineering, which requires a lot of expertise and is usually incomplete. In addition, these methods are not able to learn the context information of the sentence being parsing. Recently, Kurita [8] proposed the first neural joint parsing model and explored the neural network with few features using n-gram bidirectional LSTMs avoiding the detailed feature engineering. Our model is similar to theirs. However, these methods are lacking in that it cannot capture all word dependencies in a subtree and all dependency subtrees. To provide richer information, we consider all word dependencies by using subtree distributional representation.

Specific to the subtree representation, Dyer [9] developed the Stack Long Short-Term Memory (Stack LSTM) architecture, which does incorporate recursive neural network and look-ahead, and yields high accuracy on the word-level dependency parsing. However, the architecture has a risk of suffering from gradient vanishing when dealing with deep subtrees. In this paper, we solve the issue by using Tree LSTM [10] in the Stack LSTM, which has similar gates to LSTM cell and has the capability to memorize important information and forget unimportant one.

3 Character-Level Neural Network Parser

In this section, we describe the architecture of our model and its main components, which is summarized in Fig. 1. Our model is clearly inspired by and based on the work of Kurita [8], which uses four bidirectional LSTMs to capture N-gram feature. There are a few structural differences: (1) we use Stack-Tree LSTM to capture dependency subtrees feature, (2) we use the POS tags to participate in actions decision.

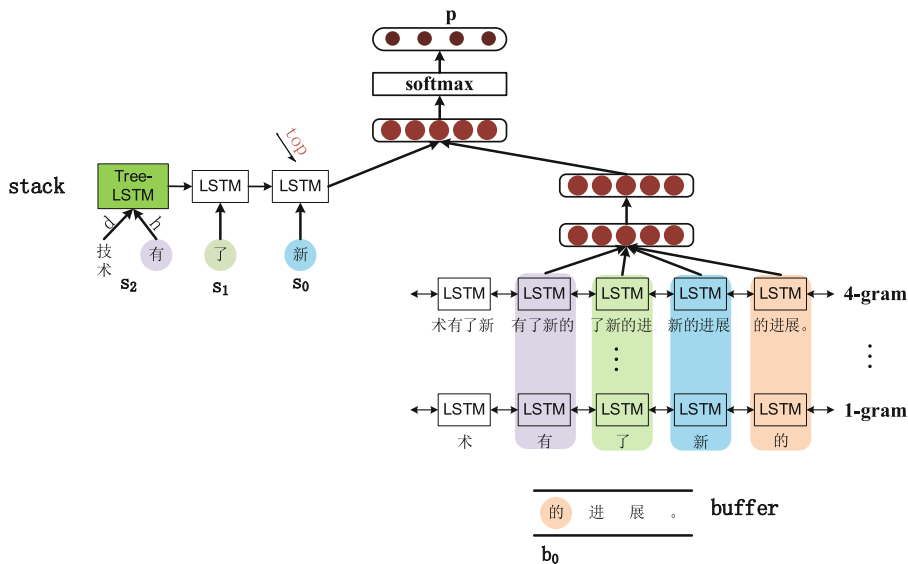


Fig. 1. The neural joint model for Chinese word segmentation, POS tagging and dependency parsing. The model consists of four bidirectional LSTM for extracting n-gram feature, Stack Tree LSTM for extracting subtree feature and MLP for predicting the possible action. Parser state computation encountered while parsing the sentence “技术有了新的进展”. s_i represents the $i + 1$ th element of the top of the stack; b_0 represents the first element of the buffer.

3.1 Transition System

Transition-based dependency parsing scan an input sentence from left to right, and perform a sequence of transition actions to predict its parse tree. The input sentence is

put into a buffer and partially built tree fragments are organized by a stack. Paring starts with an empty stack and a buffer consisting of the whole input sentence. As the basis of our parser, we employ the arc-standard system [2] for dependency parsing, one of the popular transition systems, and follow Hatori [5] to modify the system for character-level transition, for which the actions are:

- SH (t) (shift): Move the front character b_0 from the buffer onto the top of the stack as a new word, and the POS tag t is attached to the word.
- AP (append): Append the front character b_0 of the buffer to the end of the top word of the stack.
- RR (reduce-right): Add a right arc between the top element s_0 and the second top element s_1 on the stack ($s_1 \rightarrow s_0$), and remove s_0 from the stack.
- RL (reduce-left): Add a left arc between the top element s_0 and the second top element s_1 on the stack ($s_1 \leftarrow s_0$), and remove s_1 from the stack.

In the character-level transition system, while the goal of SH(t) and AP operations is to construct a new word, where each word is initialized by the action SH(t) whereas AP makes the word longer by adding one character, the goal of RR and RL operations is to construct a dependency subtree. In this paper, we examine only greedy parsing, and this class of parsers is of great interest because of their efficiency. At each time step, a transition action is taken to consume the characters from the buffer and build the dependency tree.

3.2 Subtree Feature Layer

Inspired by Dyer [9], we propose a novel neural network (Stack-Tree LSTM) architecture that integrates Stack LSTM and Tree LSTM, improving the representational capacity of Stack LSTM and solving the problem of Tree LSTM that all built subtrees information cannot be captured at the same time. The neural network architecture is presented in Fig. 2. When the input of the stack node is a subtree, the LSTM cell is replaced by Tree LSTM cell in the stack. And the inputs of the Tree LSTM cell are the right child representation and left child representation encoded by the Tree LSTM.

Specifically, when SH operation is performed, the top item on the stack provides previous time state and a new LSTM cell is pushed into the stack. The new state is computed by the LSTM cell and the cell's input is the character vector to be shifted. Different with SH operation, AP operation updates the stack state by using appended character string vector.

When RL or RR operation is performed, the representations of top two items are popped off of the stack and fed into a Tree LSTM cell. Intuitively, the Tree LSTM cell combines two vectors representations from the stack into another vector, which represents a new dependency subtree and historical information. For example, in Fig. 1(d), a new dependency tree is built, where w_4 is the head node and w_5 is the dependency node, and the result of the Tree LSTM cell, namely new state, are computed as follows:

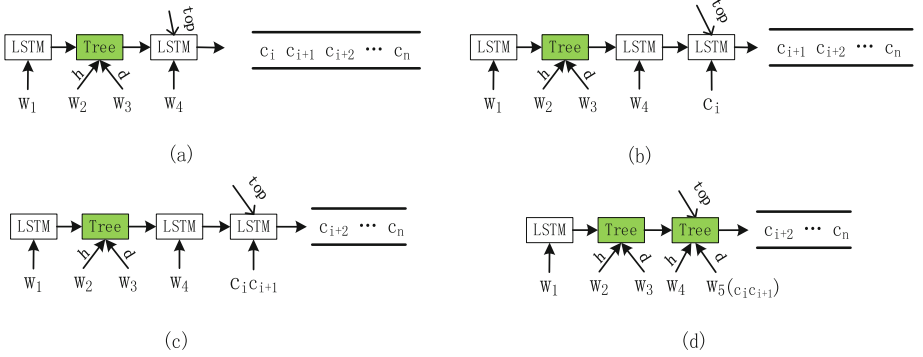


Fig. 2. The Stack Tree LSTM consists of LSTM cell and Tree LSTM cell. The figures show four configurations: (a) a stack with the input of two words w_1, w_4 and a subtree, (b) the result of a SH operation to this, (c) the result of a AP operation to (b), and (d) the result of applying a RR operation. The *top* pointer is used to access the output of the network.

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

$$c_t = f_{head} * c_{head} + f_{dep} * c_{dep} + i_t * \tilde{c}_t \quad (2)$$

$$h_t = o_t * \tanh(c_t) \quad (3)$$

Where σ is the sigmoid activation function, $*$ is the elementwise product. The resulting vector embeds the subtree in the same space as the words and other subtrees.

At each time step, all built subtrees are encoded by Stack-Tree LSTM, which integrates all items information to the top item of the stack. By querying the d -dimensional vector s_{top} of the top item, a continuous-space embedding of the contents of the current stack state is available. Then the s_{top} is fed into multi-layer perceptron (MLP) to predict the next possible transition action. When a predicted transition action is performed, the state of the stack will be updated and the output at the top of the stack will represent the new stack state.

3.3 N-Gram Feature Layer

A k -dimensional n -gram feature representation of each character is learned in this layer. Given n -characters input sentence s with characters c_1, \dots, c_n , we extract the uni-gram features c_i , bi-gram character string $c_i c_{i+1}$, tri-gram character string $c_i c_{i+1} c_{i+2}$ and four-gram character string $c_i c_{i+1} c_{i+2} c_{i+3}$, and create four sequences of input vectors $uni_{1:n}$, $bi_{1:n}$, $tri_{1:n}$, and $four_{1:n}$, in which all n -gram embeddings are given by the embedding of words and characters or the dynamically generated embedding of character strings.

And then these four sequences are fed as input to four bidirectional LSTMs respectively. These bidirectional LSTMs capture the input element with their contexts, which learn a good feature representation for parsing.

In this paper, a simple feature function $\phi(c)$ is used to extract four atomic features from a parsing configuration, which consists of the top 3 items on the stack and the first item on the buffer. Given a parse configuration $q = (\dots|s_2|s_1|s_0, b_0|\dots)$, the feature function is defined as:

$$\phi(c) = v_{s_2} \circ v_{s_1} \circ v_{s_0} \circ v_{b_0} \quad (4)$$

$$v_i = biLSTM_{uni}(i) \circ biLSTM_{bi}(i) \circ biLSTM_{mi}(i) \circ biLSTM_{four}(i) \quad (5)$$

Where \circ is the concatenate operation.

3.4 Actions Decision Layer

This layer learns a classifier to predict the correct transition actions, based on n-gram features and subtree features extracted from the configuration itself. We implement three hidden layers composed h rectified linear units (Relu).

First, two feed-forward neural layers with Relu activation function project the n-gram layer's output from 4 k -dimensional vector space into a h -dimensional vector space. The purpose of the two layers is to fine-tune the n-gram feature embedding, which helps the model to capture deeper n-gram feature and more effective global information. Next, the resulting embedding h_2 is concatenated with the output s_{top} of Stack-Tree LSTM, and fed into the last hidden layer with Relu activation function.

$$h_3 = \max\{0, W_{com}(h_2 \circ s_{top}) + b_{com}\} \quad (6)$$

Where $W_{com} \in \mathbb{R}^{h \times (h+d)}$ is the learned parameter matrix, $b_{com} \in \mathbb{R}^h$ is bias term.

Finally, h_3 is mapped into a softmax layer that outputs class probabilities for each possible transition operation:

$$p = \text{softmax}(Wh_3) \quad (7)$$

Where $W \in \mathbb{R}^{m \times h}$ and m is the number of transition actions.

3.5 Training

Given a set of training examples, the training objective of the greedy neural joint parser is to minimize the cross-entropy loss, plus a l_2 -regularization term:

$$\mathcal{L}(\theta) = - \sum_{i \in A} \log p_i + \frac{\lambda}{2} \|\theta\|^2 \quad (8)$$

A is the set of all gold actions in the training data and θ is the set of all parameters. The parameters are learned by minimizing the loss on the training data via the Adam

optimizer [12]. The initial learning rate is 0.001. To avoid overfitting, we use dropout [13] with the rate of 0.5 for regularization, which is applied to all feedforward connections.

4 Word and String Representation

In n-gram layer, we prepare the same training corpus with the segmented word files and the segmented character files. Both files are concatenated and learned by word2vec [14]. The characters and words are embedded in the same vector space during pre-training. The embedding of the unknown character string, consisting of character c_1, c_2, \dots, c_n , is obtained by the mean of each character embedding $v(c_i)$ it contains. Embeddings of words, characters and character strings have the same dimension.

In subtree feature layer, when a character string becomes a word, the embedding of the word is queried in the pre-trained embeddings, if not, the same way encoding for unknown character string is adopted to get the word embedding. Different with Kurita [8], we use the predicted POS tags in our model, provided as auxiliary input to the parser. Specifically, the predicted POS tag embedding t of the word is concatenated with the word embedding w . A linear map is applied to the resulting vector and passed through a component-wise Relu.

$$v = \max\{0, W_{word}[w; t] + b_{word}\} \quad (9)$$

The POS embedding and word embedding are learned together with the model.

5 Experiments

5.1 Experimental Settings

In this section, we evaluate our parsing model on the Penn Chinese Treebank 5.1 (CTB-5), splitting the corpora into training, development and test sets, following the splitting of [15]. The development set is used for parameter tuning. Pre-trained word and characters embeddings are learned from the Gigaword corpus and word2vec [14], as segmented by the Stanford Chinese Segmenter [16].

We use standard measures of word-level precision, recall, and F1 score to evaluate model performance on three tasks, following previous works [5–8]. Dependency parsing task is evaluated with the unlabeled attachment scores excluding punctuations. The POS tags and dependencies cannot be correct unless the corresponding words are segmented correctly.

Dimensionality. Our model sets dimensionalities as follows. Bidirectional LSTM and Stack Tree LSTM hidden states are of size 200. Embeddings of POS tags used in Stack Tree LSTM have 32 dimensions. Pre-trained word and character embeddings have 200 dimensions and three hidden layers in classifier have 400 dimensions.

5.2 Experimental Results and Analysis

Effects of Different Composition Methods in Joint Model. We conducted experiments to verify the capability of the Stack-Tree LSTM (ST-LSTM) on the dependency tree representations. We compare our ST-LSTM with three popular composition methods: Stack LSTM [9], recursive convolutional neural network [11], and a composition function based on bidirectional LSTM [17]. Table 1 show the comparison of F1 scores on three tasks. Clearly, our model is superior in terms of POS tagging and dependency parsing. However, we notice that the composition function of recursive convolutional neural network outperforms our model on Chinese word segmentation tasks. A likely reason for the close performance with our model may be the feature of relative distance between words. In future work, we also try to use distance feature to improve model performance.

Table 1. Experimental results for different composition functions. S-LSTM, B-LSTM, and RCNN denote Stack LSTM, bidirectional LSTM and recursive convolutional neural network respectively. ST-LSTM denotes Stack-Tree LSTM.

	Seg	POS	Dep
S-LSTM	97.71	93.36	79.21
B-LSTM	96.69	92.10	79.02
RCNN	98.03	93.35	79.58
ST-LSTM	97.78	93.51	79.66

Effects of POS Tags in Joint Model. Furthermore, we also conducted experiments to test the effectiveness of the predicted POS tagging on each task. We implemented two model: ST-LSTM and ST-LSTM model without POS tags (-POS). Concretely, we use predicted POS tagging and pre-trained embedding as word representations in ST-LSTM, but we only use pre-trained embedding as word representations in -POS. As shown in Table 3, performance of model without POS tags is weaker than the basic model in word segmentation and dependency parsing. In contrast, the basic model with POS tags gives a 0.21% accuracy improvement in dependency parsing.

Final Results. Table 2 shows the final test results of our parser for Chinese word segmentation, POS tagging and dependency parsing. Considering that the model proposed can extract the information of children’s nodes, we only implement the feature function of four features. We also include in the table results from the first joint parser of Hatori [5], the using inter-word dependencies and intra-word dependencies parser of Guo [7], the arc-eager model of Zhang [6], the feature based parser of Kurita [8], and the n-gram bidirectional LSTM greedy model with four and eight features of Kurita [8].

Overall, our parser substantially outperforms the four features n-gram bidirectional LSTM model of Kurita [8], both in the full configuration and in the -POS conditions we report. Moreover, we find that our model can learn better dependency tree representations and achieve higher accuracies in each task than other composition function. And we note that this is a significant improvement in dependency parsing only after

Table 2. Final results. * denotes the feature engineering.

Model	Method	Seg	POS	Dep
Hatori12*	Beam	97.75	94.33	81.56
Guo14*	Beam	97.52	93.93	79.55
Zhang14*	Beam	97.67	94.28	81.63
Kurita17*	Greedy	98.24	94.49	80.15
Kurita17(4feat)	Greedy	97.72	93.12	79.03
Kurita17(8feat)	Greedy	97.70	93.37	79.38
ST-LSTM	Greedy	97.78	93.51	79.66
-POS	Greedy	97.74	93.53	79.45

using predicted POS information in word representation. Our model performs slightly worse than these joint models using large feature set, but we do not rely on feature engineering.

6 Conclusion

In this paper, we introduced Stack-Tree LSTM, a novel composition function to encode dependency subtrees from characters and words for Chinese word segmentation, POS tagging and dependency parsing. Our model only relies on effectively feature function and architecture design, and is able to automatically learn these useful features for making decision. Through a series of experiments, we demonstrated that our approach provides substantial improvement over the baseline methods, by capturing the subtree nodes information and more dependency structures.

In the future, we will expand the scale of the experiment and further verify the effectiveness of the proposed method. In addition, we further explore better way to learning dependency trees representations.

Acknowledgments. The authors are supported by the National Nature Science Foundation of China (Contract 61370130 and 61473294), and the Beijing Municipal Natural Science Foundation (4172047).

References

1. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: International Workshop on Parsing Technologies 2003, Nancy, France, pp. 195—206 (2003)
2. Nivre, J.: Incrementality in deterministic dependency parsing. In: Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together, pp. 50–57. Association for Computational Linguistics (2004)
3. Zhang, Y., Clark, S.: A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam search. In: Proceedings of EMNLP, Hawaii, USA (2008)

4. Huang, L., Sagae, K.: Dynamic programming for linear-time incremental parsing. In: Proceedings of ACL, Uppsala, Sweden, pp. 1077–1086, July 2010
5. Hatori, J., Matsuzaki, T., Miyao, Y., Tsujii, J.I.: Incremental joint approach to word segmentation, pos tagging, and dependency parsing in Chinese. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics. Long Papers, vol. 1, pp. 1045–1053. Association for Computational Linguistics (2012)
6. Zhang, M., Zhang, Y., Che, W., Liu, T.: Character-level Chinese dependency parsing. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. Long Papers, vol. 1, pp. 1326–1336. Association for Computational Linguistics (2014)
7. Guo, Z., Zhang, Y., Su, C., Xu, J.: Character-level dependency model for joint word segmentation, POS tagging, and dependency parsing in Chinese. *J. Chin. Inf. Process.* **E99**,D(1), 257–264 (2014)
8. Kurita, S., Kawahara, D., Kurohashi, S.: Neural joint model for transition-based chinese syntactic analysis. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. Long Papers, vol. 1, pp. 1204–1214. Association for Computational Linguistics (2017)
9. Dyer, C., Ballesteros, M., Ling, W., Matthews, A., Smith, N.A.: Transition-based dependency parsing with stack long short-term memory. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. Long Papers, vol. 1, pp. 334–343. Association for Computational Linguistics (2015)
10. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. Long Papers, vol. 1, pp. 1556–1566, Beijing, China. Association for Computational Linguistics (2015)
11. Zhu, C., Qiu, X., Chen, X., Huang, X.: A re-ranking model for dependency parser with recursive convolutional neural network. *Comput. Sci.* (2015)
12. Kingma, D.P., Adam, J.B.: A method for stochastic optimization. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. Long Papers, vol. 1 (2015)
13. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)
14. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. Volume abs/1301.3781 (2013)
15. Jiang, W., Huang, L., Liu, Q., Lu, Y.: A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In: Proceedings of ACL-2008: HLT, pp. 897–904. Association for Computational Linguistics (2008)
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 (2005)
17. Dyer, C., Kuncoro, A., Ballesteros, M., Smith, N.A.: Recurrent Neural Network Grammars, pp. 199–209. The North American Chapter of the Association for Computational Linguistics (2016)