Conversion and Exploitation of Dependency Treebanks with Full-Tree LSTM

Bo Zhang, Zhenghua Li(✉), and Min Zhang

School of Computer Science and Technology, Soochow University, Suzhou, China
bzhang17@stu.suda.edu.cn, {zhli13,minzhang}@suda.edu.cn

Abstract. As a method for exploiting multiple heterogeneous data, supervised treebank conversion can straightforwardly and effectively utilize linguistic knowledge contained in heterogeneous treebank. In order to efficiently and deeply encode the source-side tree, we for the first time investigate and propose to use Full-tree LSTM as a tree encoder for treebank conversion. Furthermore, the corpus weighting strategy and the concatenation with fine-tuning approach are introduced to weaken the noise contained in the converted treebank. Experimental results on two benchmark datasets with bi-tree aligned trees show that (1) the proposed Full-Tree LSTM approach is more effective than previous treebank conversion methods, (2) the corpus weighting strategy and the concatenation with fine-tuning approach are both useful for the exploitation of the noisy converted treebank, and (3) supervised treebank conversion methods can achieve higher final parsing accuracy than multi-task learning approach.

Keywords: Supervised treebank conversion · Full-tree LSTM · Treebank exploitation · Corpus weighting · Concatenation with fine-tuning · Multi-task learning

1 Introduction

In recent years, neural network based dependency parsing has made great progress and outperforms the traditional discrete-feature based approaches [1,3–5,17]. In particular, Dozat and Manning (2017) [4] propose a simple yet effective deep biaffine parser, which achieves the state-of-the-art performance in many languages and datasets.

Meanwhile, exploiting various heterogeneous treebanks for boosting parsing performance is always the focus in the research community [6–10,12–14,18]. However, due to the lack of manually labeled data where each sentence has two

Supported by National Natural Science Foundation of China (Grant No. 61525205, 61876116). Zhenghua Li is the corresponding author. We thank the anonymous reviewers for the helpful comments and Qingrong Xia and Houquan Zhou for their help on preparing this English version.

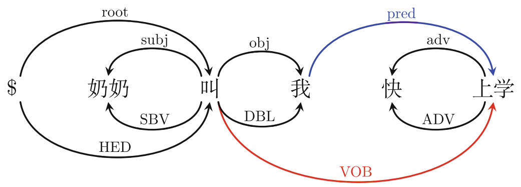

Fig. 1. Example of treebank conversion from the source-side HIT (under) to the target-side SU (upper).

syntactic trees complying with two different annotation guidelines, called bi-tree aligned data, as shown in Fig. 1, previous works mainly focus on the unsupervised treebank conversion methods [8,10,12,13,18] and indirect treebank exploitation methods [6,9,14].

Jiang et al. (2018) [7] first propose the task of supervised treebank conversion by manually constructing a bi-tree aligned dataset. As shown in Fig. 1, given an input sentence x, treebank conversion aims to convert the source-side tree $\mathbf{d}^{src}$ to the target-side tree $\mathbf{d}^{tgt}$.

There are two main challenges for treebank exploitation via treebank conversion. One is how to convert $\mathbf{d}^{src}$ to $\mathbf{d}^{tgt}$ with high quality (*treebank conversion*), and the other is how to effectively exploit the converted treebank for higher parsing accuracy of target side (*treebank exploitation*).

Jiang et al. (2018) [7] then propose the pattern embedding (PE) and the shortest path TreeLSTM (SP-Tree) approach for treebank conversion. The PE approach uses a custom pattern to encode $\mathbf{d}^{src}$, which captures the syntactic structure correspondence between the target-side and source-side. Given a dependency $i \leftarrow j$ in $\mathbf{d}^{tgt}$, they define nine patterns in accordance with the structure and the distance of w_i and w_j in $\mathbf{d}^{src}$. And then these patterns are mapped into embedded vectors as the representation of $\mathbf{d}^{src}$. The SP-Tree approach uses the bidirectional shortest path TreeLSTM to deeply encode $\mathbf{d}^{src}$. For scoring a dependency $i \leftarrow j$ in $\mathbf{d}^{tgt}$, they use the concatenation of the TreeLSTM hidden vectors of w_i, w_j and w_a as the representation of $\mathbf{d}^{src}$, where a is the lowest common ancestor node of w_i and w_j in $\mathbf{d}^{src}$.

Following Jiang et al. (2018) [7], our preliminary experiments show that (1) the performance of the PE approach is unstable since it only encodes $\mathbf{d}^{src}$ quite locally, especially when the two treebanks are very divergent, and (2) the SP-Tree approach is very inefficient. for a sentence with n words, the SP-Tree approach requires running TreeLSTM n^2 times. In order to solve these problems, we propose to use Full-tree LSTM as a tree encoder for treebank conversion, called Full-Tree approach. We find that with proper dropping-out of the outputs of the Full-Tree LSTM hidden cells, the Full-Tree approach can achieve slightly higher conversion performance than the SP-Tree approach and is much more efficient.

In terms of treebank exploitation, since the target-side treebank and converted treebank comply with same guideline, Jiang et al. (2018) [7] simply concatenate them to train a parser, called the concatenation approach, which cannot handle the noise contained in the converted treebank. We propose two ways to better exploit the converted treebank, i.e., the corpus weighting strategy and the concatenation with fine-tuning approach.

We conduct experiments on two benchmarks of bi-tree aligned data, and the results show that (1) compared with the treebank conversion approaches of Jiang et al. (2018) [7], the Full-Tree LSTM approach can stably and efficiently achieve higher accuracy, (2) the corpus weighting strategy and the concatenation with fine-tuning approach can both effectively exploit the converted treebank and further improve the performance of the target-side parsing, and (3) our approaches are significantly better than Jiang et al. (2018) [7] both on treebank conversion and exploitation and can significantly outperform the multi-task learning baseline.

2 Related Works

At present, major languages in the world often possess multiple large-scale heterogeneous treebanks, e.g., Tiger and Tüba-D/Z for German, Talbanken and Syntag for Swedish, ISST and TUT for Italian, etc. We focus on exploitation of Chinese heterogeneous treebanks, and select HIT-CDT [2], PCTB7 [16], and SU-CDT [7] for case study.

Unsupervised Treebank Conversion. Niu et al. (2009) [12], Zhu et al. (2011) [18], and Li et al. (2013) [8] propose to convert source-guideline treebank into the target guideline with a statistical model, and use the converted treebank as extra training data for boosting parsing performance. Without bi-tree aligned data, these methods rely on heuristic rules or use automatically generated target-side trees as pseudo gold-standard reference during training.

Indirect Treebank Exploitation. Li et al. (2012) [9], Guo et al. (2016) [6], and Stymne et al. (2018) [14] propose to indirectly exploit heterogeneous treebanks without explicit conversion based on guide features, multi-task learning (MTL), or treebank embedding.

Supervised Treebank Conversion. Jiang et al. (2018) [7] first propose supervised treebank conversion task based on a newly constructed bi-tree ($SU^{\mathtt{HIT}}$) aligned dataset. They also propose two treebank conversion approaches (PE and SP-Tree) and use the concatenation approach to exploit the converted treebank.

In this work, we follow Jiang et al. (2018) [7] and focus on the supervised treebank conversion task. We propose to use the Full-Tree LSTM to encode the source-side tree to deal with the disadvantages of their PE and SP-Tree approaches in efficiency and efficacy. For treebank exploitation, we propose to use the corpus weighting strategy and the concatenation with fine-tuning approach, in order to increase the impact of manually labeled data and weaken the noise contained in converted treebank.

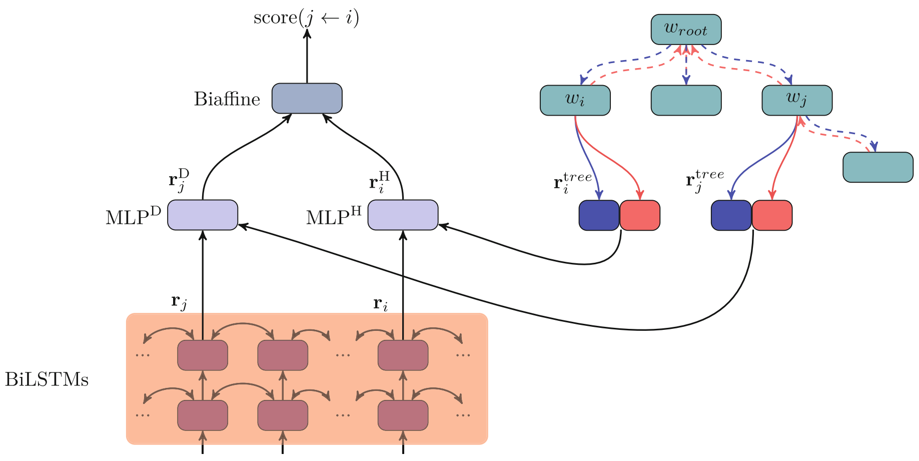

Fig. 2. Computation of score($j \leftarrow i$) by conversion model. The left part is the biaffine parser and the right part is the Full-Tree LSTM structure.

3 Treebank Conversion Based on Full-Tree LSTM

Basic Parser. We build all our models based on the state-of-the-art biaffine parser. As a graph-based dependency parser, it applies multi-layer bidirectional LSTMs (BiLSTM) to encode the input sentence, and employs a deep biaffine transformation to compute the scores of all possible dependencies and uses viterbi decoding to find the highest-scoring tree. The left part of Fig. 2 shows how to score a dependency $j \leftarrow i$. Due to space limitation, please refer to Dozat and Manning (2017) [4] and Jiang et al. (2018) [7] for more details.

Full-Tree LSTM is proposed by Tai et al. (2015) [15] to use the information of the whole syntactic tree, whereas SP-Tree LSTM is proposed by Miwa and Bansal (2016) [11] as an extension of Full-Tree LSTM in task of relation extraction, which only encodes the information of nodes on the shortest path of two focused words. In this work, we use Full-Tree LSTM as the encoder of the source-side tree to treebank conversion. Compared with the shallow PE approach, the Full-Tree approach uses bidirectional TreeLSTM to deeply encode $\mathbf{d}^{src}$ with strong stability. Compared with the SP-Tree approach, the Full-Tree approach requires less calculation to obtain word-level representation incorporating $\mathbf{d}^{src}$ for all words in sentence.

As shown in the right part of Fig. 2, we use the bidirectional Full-Tree (BiFull-Tree) LSTM to encode the $\mathbf{d}^{src}$ (bottom-up and top-down). Given a $\mathbf{d}^{src}$, the bottom-up Full-Tree LSTM starts from the leaves node and accumulates information until the root node (the inner red dashed line), whereas the top-down Full-Tree LSTM propagates information in opposite direction (the outer blue dashed line).

Following Jiang et al. (2018), we stack Full-Tree LSTM on the top of the BiLSTM layer of the basic biaffine parser. Moreover, we also consider the dependency label of the $\mathbf{d}^{src}$, and embed a label into a dense vector as the extra input of Full-tree LSTM. For example, the input vector for w_k in the Full-tree LSTM is $\mathbf{x}_k = \mathbf{h}_k \oplus \mathbf{e}^{l_k}$, where $\mathbf{h}_k$ is the top-level BiLSTM output vector at w_k, and l_k is the label between w_k and its head word in $\mathbf{d}^{src}$, and $\mathbf{e}^{l_k}$ is the label embedding.

In the bottom-up Full-tree LSTM, an LSTM node computes a hidden vector based on the combination of the input vector and the hidden vectors of its children in $\mathbf{d}^{src}$. The right part of Fig. 2 and Eq. 1 illustrate the computation of the Full-Tree LSTM output vector at w_k.

$$\begin{aligned}
\tilde{\mathbf{h}}_k &= \sum_{m \in \mathcal{C}(k)} \mathbf{h}_m \\
\mathbf{i}_k &= \sigma\left(\mathbf{W}^{(i)}\mathbf{x}_k + \mathbf{U}^{(i)}\tilde{\mathbf{h}}_k + \mathbf{b}^{(i)}\right) \\
\mathbf{f}_{k,m} &= \sigma\left(\mathbf{W}^{(f)}\mathbf{x}_k + \mathbf{U}^{(f)}\mathbf{h}_m + \mathbf{b}^{(f)}\right) \\
\mathbf{o}_k &= \sigma\left(\mathbf{W}^{(o)}\mathbf{x}_k + \mathbf{U}^{(o)}\tilde{\mathbf{h}}_k + \mathbf{b}^{(o)}\right) \\
\mathbf{u}_k &= \tanh\left(\mathbf{W}^{(u)}\mathbf{x}_k + \mathbf{U}^{(u)}\tilde{\mathbf{h}}_k + \mathbf{b}^{(u)}\right) \\
\mathbf{c}_k &= \mathbf{i}_k \odot \mathbf{u}_k + \sum_{m \in \mathcal{C}(k)} \mathbf{f}_{k,m} \odot \mathbf{c}_m \\
\mathbf{h}_k &= \mathbf{o}_k \odot \tanh\left(\mathbf{c}_k\right)
\end{aligned} \tag{1}$$

where $\mathcal{C}(k)$ means the children of w_k in the $\mathbf{d}^{src}$, and $\mathbf{f}_{k,m}$ is the forget vector for w_k's child w_m.

In the top-down Full-tree LSTM, an LSTM node computes a hidden vector based on the combination of the input vector and the hidden vector of its single father node in the $\mathbf{d}^{src}$. The calculation process is consistent with Eq. (1).

At each time step, the hidden vector of the BiFull-Tree LSTM is the concatenation of the bottom-up and top-down Full-Tree LSTM hidden vectors, denoted as $\mathbf{r}^{tree}$. For example, the output vector of the BiFull-Tree LSTM at w_k is $\mathbf{r}_k^{tree} = \mathbf{h}_k^{\uparrow} \oplus \mathbf{h}_k^{\downarrow}$, where $\mathbf{h}_k^{\uparrow}$ is the hidden vector of the bottom-up Full-Tree LSTM at w_k, and $\mathbf{h}_k^{\downarrow}$ is the hidden vector of the top-down Full-Tree LSTM.

Taking scoring a dependency $j \leftarrow i$ as an example, we integrate information of $\mathbf{d}^{src}$ encoded by the BiFull-Tree LSTM into target-side parser by the Eq. (2).

$$\begin{aligned}
\mathbf{r}_j^{\mathrm{D}} &= \mathrm{MLP}^{\mathrm{D}}\left(\mathbf{h}_j \oplus \mathbf{r}_j^{tree}\right) \\
\mathbf{r}_i^{\mathrm{H}} &= \mathrm{MLP}^{\mathrm{H}}\left(\mathbf{h}_i \oplus \mathbf{r}_i^{tree}\right)
\end{aligned} \tag{2}$$

Finally, the $\mathbf{r}_j^{\mathrm{D}}$ and $\mathbf{r}_i^{\mathrm{H}}$ are fed into the biaffine layer to compute a more reliable score of the dependency $i \leftarrow j$, with the help of the guidance of $\mathbf{d}^{src}$.

4 Treebank Exploitation

Treebank exploitation is how to make full use of the converted source-side treebank to improve the performance of the target-side parsing in our scenario. Considering the noise contained in the converted data and the big gap between the

Table 1. Data statistics.

Data	Sent num	Dependency num	Consistency	
			Dependency	Label
$SU^{\mathtt{HIT}}$	10,761	50,866	81.68%	73.73%
HIT train	52,450	980,791		
$SU^{\mathtt{PCTB}}$	11,579	49,979	66.37%	55.14%
PCTB train	43,114	961,654		

size of the converted and manually labeled data ($SU^{\mathtt{HIT}}$ has 980,791 converted dependencies, and only 50,866 manually labeled dependencies), the concatenation approach is too simple to effectively exploit converted data. In this work, we use the corpus weighting strategy and the concatenation with fine-tuning approach to strengthen the impact of manually labeled data on the parser, which brings significant improvement over the concatenation approach.

The corpus weighting strategy is an effective method of using multiple corpus training models. We apply it as a method to rationally use data containing noise. Specifically, before each iteration, we randomly sample training sentences separately from the target-side and converted data in the proportion of $1 : M$. Then we merge and randomly shuffle the sampled data for one-iteration training. We treat $M \geq 1$ as a hyper-parameter tuned on the dev data. The concatenation with fine-tuning approach first train on the concatenation of the target-side and converted data and follow by fine-tuning for the target-side data.

Multi-task learning, as a strong baseline for treebank exploitation, aims to incorporate labeled data of multiple related tasks for improving performance. Following Jiang et al. (2018) [7], we treat the source-side and target-side parsing as two individual tasks, which share parameters of word/tag embeddings and multi-layer BiLSTM and separate parameters of the MLP and biaffine layers. Due to space limitation, please refer to Jiang et al. (2018) [7] for the details.

5 Experimental Results and Analysis

5.1 Data

We employ the $SU^{\mathtt{HIT|PCTB}}$ conversion datasets to conduct our experiments. Table 1 shows the number of sentences and annotated dependencies, respectively. For $SU^{\mathtt{HIT}}$ dataset, we directly employ the data settings of Jiang et al. (2018) [7]. For $SU^{\mathtt{PCTB}}$, we randomly choose 1k and 2k sentences as the dev and test datasets, and use the remaining 8k sentences for training.

In order to measure the similarity (or homogeneity) between the source-side and target-side guidelines, we list the ratios of consistent dependencies and relations following the practice in Jiang et al. (2018) [7]. We can see that the PCTB is much more divergent from SU than HIT, and thus it is more difficult to convert PCTB into the SU guideline.

Table 2. Dropping-out the outputs of the Full-Tree LSTM hidden cells on conversion accuracy on the dev data.

Dropout ratio	SU^{HIT}		SU^{PCTB}	
	UAS	LAS	UAS	LAS
0	86.00	81.13	80.49	75.92
0.3	86.20	81.46	81.50	77.05
0.5	86.32	81.66	81.69	77.54
0.7	**86.42**	**81.69**	81.92	77.68
0.8	85.93	81.21	**82.31**	**78.02**
0.9	85.78	81.18	81.64	77.31

5.2 Settings and Evaluation Metrics

We implement all the models with Pytorch 0.4.1, and release the codes at https://github.com/sdzhangbo/Supervised-Treebank-Conversion.

In order to fairly compare with our Full-Tree method with the PE and SP-Tree methods, we strictly follow the experimental settings of Jiang et al. (2018) [7]: two-layer BiLSTMs with 300 output dimension; The MLP output dimension is 200 for the biaffine parser and MTL; embedding dimension of the source-side dependency labels is 50, the output dimension of TreeLSTM is 100, and the output dimension of MLP is 300 for the conversion model.

For evaluation, we employ the unlabeled attachment score (UAS) and labeled attachment score (LAS) for both conversion and exploitation.

5.3 Results of Treebank Conversion on the Dev Data

We count the time of encoding 1k sentences of the three approaches, and find that the Full-Tree approach is little slower than the PE approach (2 vs. 1) and is much more efficient than the SP-Tree approach (2 vs. 229).

We carefully tune the dropout ratio of the Full-Tree LSTM output, as shown in Table 2. Specifically, 0 means that dropout is unused. We omit the results of 0.1/0.2/0.4/0.6 due to space limitation. On SU^{HIT}, the model achieves best LAS performance of 81.69% when we use the dropout ratio of 0.7, which outperforms the dropout ratio of 0 by 0.56% (81.69–81.13). The model achieves the highest LAS of 78.02% when the dropout ratio is 0.8 on SU^{PCTB}, which outperforms the model without Full-Tree LSTM output dropout by 2.1% (78.02–75.92). We can conclude that dropout of Full-Tree LSTM output has a positive impact on the conversion performance, i.e., the lower the data consistency, the more obvious of the performance improvement.

5.4 Results of Treebank Exploitation on the Dev Data

Table 3 shows the results comparison of the three treebank exploitation methods on the dev data. The results of concatenation approach of Jiang et al. (2018)

Table 3. Effect of corpus weighting strategy and concatenation with fine-tuning approach on dev data.

Methods		Exploit converted HIT		Exploit converted PCTB	
		UAS	LAS	UAS	LAS
Concatenation		81.50	76.30	79.73	75.09
Corpus weighting	$M = 1$	81.37	76.14	79.80	75.25
	$M = 2$	82.10	**76.97**	79.89	**75.46**
	$M = 3$	81.85	76.61	**80.01**	75.30
	$M = 4$	81.66	76.55	79.50	75.31
	$M = 5$	**82.14**	76.84	79.34	74.77
	$M = 6$	81.37	76.26	79.84	75.23
Concatenation with fine-tuning		**82.24**	**77.17**	**80.56**	**76.11**

[7] is shown in the third row, The results of corpus weighting strategy is shown from row 4 to row 9. The tenth row shows the results of concatenation with fine-tuning approach. We can see the best model of corpus weighting outperforms the concatenation model by 0.67% (76.97–76.30) and 0.37% (75.46–75.09) on the two datasets, respectively. And the concatenation with fine-tuning approach outperforms the concatenation model by 0.87% (77.17–76.30) and 1.02% (76.11–75.09) on the two datasets, respectively.

Table 4. Conversion accuracy of PE, SP-Tree and Full-Tree approaches on test data.

Methods	$SU^{\mathtt{HIT}}$		$SU^{\mathtt{PCTB}}$	
	UAS	LAS	UAS	LAS
PE	86.66	82.03	74.71	67.11
SP-Tree	**86.69**	**82.09**	81.94	77.95
Full-Tree	86.28	82.04	**82.45**	**78.45**

We can see that the corpus weighting strategy and the concatenation with fine-tuning approach can effectively exploit converted treebank that contains noise, which can further improve the target-side parser performance.

5.5 Results of Conversion and Exploitation on the Test Data

Table 4 shows the results comparison of the three treebank conversion methods. The Full-Tree method achieves almost the same performance compared with PE and SP-Tree on $SU^{\mathtt{HIT}}$, which has high data consistency. However, on the $SU^{\mathtt{PCTB}}$ data with lower data consistency, the Full-Tree method outperforms the

Table 5. Parsing accuracy of different parsers on test data.

Methods	SU^{HIT}		SU^{PCTB}	
	UAS	LAS	UAS	LAS
Single parser	75.57	70.81	76.78	72.48
MTL and HIT	80.08	75.46	78.80	74.61
Jiang et al. (2018)	81.33	76.73	80.09	76.03
Ours	**81.86**	**77.27**	**80.90**	**76.77**

SP-Tree by 0.5% (78.45–77.95), and advances the PE by large margin (11.34%, 78.45–67.11) in LAS, which indicates that the PE method heavily relies on the conversion data and cannot exploit the source-side data stably. The Full-Tree and SP-Tree method can both stably encode the source-side tree information, which can achieve better convert performance over data with low consistency. Considering the speed, the Full-Tree is a superior encoding method.

Finally, Table 5 lists the performance of the target-side parser with the help of the source-side treebank by different exploitation methods. The third row shows the results of our basic model that only use the target-side data. The experimental results of using MTL to exploit the source-side data is shown in row 4. The fifth and sixth row show the results of applying the concatenation approach and the concatenation with fine-tuning approach to exploit the target-side and converted source-side data, respectively.

On the two datasets, we find that the MTL method respectively outperforms the baseline by 4.65% (75.46–70.81) and 2.13% (74.61–72.48) and our method respectively outperforms the baseline by 6.46% (77.27–70.81) and 4.29% (76.77–72.48), which indicate that heterogeneous data with common information can effectively improve the performance of the target-side parsing. Additionally, our method also outperforms the concatenation approach of Jiang et al. (2018) [7] by 0.54% (77.27–76.73) and 0.74% (76.77–76.03).

Moreover, the treebank conversion method can further improve the performance of target side parser, which outperforms the MTL method by 1.81% (77.27–75.46) and 2.16% (76.77–74.61) on the two datasets, which indicates the treebank conversion method is a more straightforward and effective way to utilize heterogeneous data.

6 Conclusions

To exploit multiple heterogeneous Chinese dependency treebanks, we propose a supervised treebank conversion method based on Full-Tree LSTM. Experimental results on two conversion datasets show that (1) our Full-Tree LSTM approach can efficiently and deeply encode the source-side tree, and is superior to the PE and SP-Tree approaches of Jiang et al. (2018) [7]; (2) the corpus weighting strategy and the concatenation with fine-tuning approach are both helpful in exploiting noisy converted data for further improving the target-side parsing performance; (3) compared with the MTL method, supervised treebank conversion is a more effective way to exploit heterogeneous treebanks and can achieve higher final parsing accuracy. Moreover, we also find that proper dropping-out of the outputs of TreeLSTM hidden cells significantly affects the performance of the Full-Tree LSTM approach.

References

1. Andor, D., et al.: Globally normalized transition-based neural networks. In: Proceedings of ACL, pp. 2442–2452 (2016)
2. Che, W., Li, Z., Liu, T.: Chinese dependency treebank 1.0 (LDC2012T05). In: Philadelphia: Linguistic Data Consortium (2012)
3. Chen, D., Manning, C.: A fast and accurate dependency parser using neural networks. In: Proceedings of EMNLP, pp. 740–750 (2014)
4. Dozat, T., Manning, C.D.: Deep biaffine attention for neural dependency parsing. In: Proceedings of ICLR (2017)
5. Dyer, C., Ballesteros, M., Ling, W., Matthews, A., Smith, N.A.: Transition-based dependency parsing with stack long short-term memory. In: Proceedings of ACL, pp. 334–343 (2015)
6. Guo, J., Che, W., Wang, H., Liu, T.: A universal framework for inductive transfer parsing across multi-typed treebanks. In: Proceedings of COLING, pp. 12–22 (2016)
7. Jiang, X., Li, Z., Zhang, B., Zhang, M., Li, S., Si, L.: Supervised treebank conversion: data and approaches. In: Proceedings of ACL, pp. 2706–2716 (2018)
8. Li, X., Jiang, W., Lü, Y., Liu, Q.: Iterative transformation of annotation guidelines for constituency parsing. In: Proceedings of ACL, pp. 591–596 (2013)
9. Li, Z., Che, W., Liu, T.: Exploiting multiple treebanks for parsing with quasi-synchronous grammar. In: Proceedings of ACL, pp. 675–684 (2012)
10. Magerman, D.M.: Natural language parsing as statistical pattern recognition. arXiv preprint cmp-lg/9405009 (1994)
11. Miwa, M., Bansal, M.: End-to-end relation extraction using LSTMs on sequences and tree structures. In: Proceedings of ACL, pp. 1105–1116 (2016)
12. Niu, Z.Y., Wang, H., Wu, H.: Exploiting heterogeneous treebanks for parsing. In: Proceedings of ACL, pp. 46–54 (2009)
13. Nivre, J., Scholz, M.: Deterministic dependency parsing of English text. In: Proceedings of COLING 2004, pp. 64–70 (2004)
14. Stymne, S., de Lhoneux, M., Smith, A., Nivre, J.: Parser training with heterogeneous treebanks. arXiv preprint arXiv:1805.05089 (2018)
15. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. In: Proceedings of ACL, pp. 1556–1566 (2015)
16. Xue, N., Xia, F., Chiou, F.D., Palmer, M.: The penn Chinese treebank: phrase structure annotation of a large corpus. Nat. Lang. Eng. **11**(2), 207–238 (2005)
17. Zhou, H., Zhang, Y., Huang, S., Chen, J.: A neural probabilistic structured-prediction model for transition-based dependency parsing. In: Proceedings of ACL, pp. 1213–1222 (2015)
18. Zhu, M., Zhu, J., Hu, M.: Exploitating multiple treebanks using a feature-based approach. In: Proceedings of ACL, pp. 715–719 (2011)