In-Order Transition-based Constituent Parsing

Jiangming Liu and **Yue Zhang**
Singapore University of Technology and Design,
8 Somapah Road, Singapore, 487372
jmliunlp@gmail.com, yue_zhang@sutd.edu.sg

Abstract

Both bottom-up and top-down strategies have been used for neural transition-based constituent parsing. The parsing strategies differ in terms of the order in which they recognize productions in the derivation tree, where bottom-up strategies and top-down strategies take post-order and pre-order traversal over trees, respectively. Bottom-up parsers benefit from rich features from readily built partial parses, but lack lookahead guidance in the parsing process; top-down parsers benefit from non-local guidance for local decisions, but rely on a strong encoder over the input to predict a constituent hierarchy before its construction. To mitigate both issues, we propose a novel parsing system based on in-order traversal over syntactic trees, designing a set of transition actions to find a compromise between bottom-up constituent information and top-down lookahead information. Based on stack-LSTM, our psycholinguistically motivated constituent parsing system achieves 91.8 F_1 on the WSJ benchmark. Furthermore, the system achieves 93.6 F_1 with supervised reranking and 94.2 F_1 with semi-supervised reranking, which are the best results on the WSJ benchmark.

1 Introduction

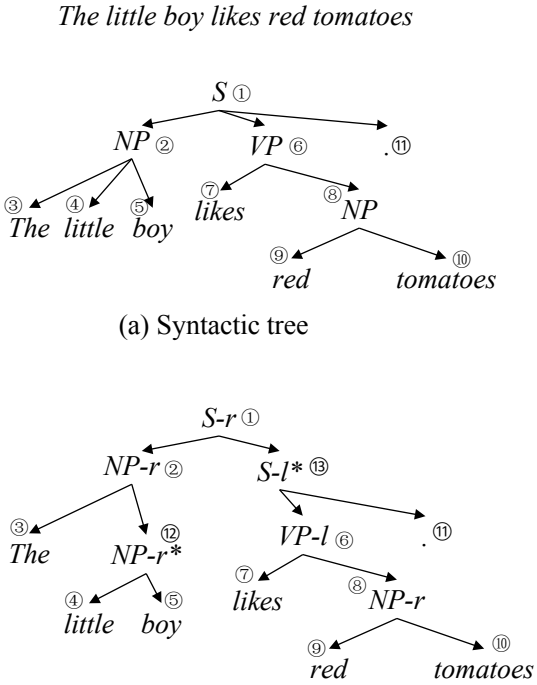
Transition-based constituent parsing employs sequences of local transition actions to construct constituent trees over sentences. There are two popular transition-based constituent parsing systems, namely bottom-up parsing (Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013; Watanabe and Sumita, 2015) and top-down parsing (Dyer et al., 2016; Kuncoro et al., 2017). The parsing strategies differ in terms of the order in which they recognize productions in the derivation tree.

The process of bottom-up parsing can be regarded as *post-order* traversal over a constituent tree. For example, given the sentence in Figure 1, a bottom-up shift-reduce parser takes the action sequence in Table 2(a)[1] to build the output, where the word sequence "The little boy" is first read, and then an *NP* recognized for the word sequence. After the system reads the verb "likes" and its subsequent *NP*, a *VP* is recognized. The full order of recognition for the tree nodes is ③→④→⑤→②→⑦→⑨→⑩→⑧→⑥→⑪→①. When making local decisions, rich information is available from readily built partial trees (Zhu et al., 2013; Watanabe and Sumita, 2015; Cross and Huang, 2016), which contributes to local disambiguation. However, there is lack of top-down guidance from lookahead information, which can be useful (Johnson, 1998; Roark and Johnson, 1999; Charniak, 2000; Liu and Zhang, 2017). In addition, *binarization* must be applied to trees, as shown in Figure 1(b), to ensure a constant number of actions (Sagae and Lavie, 2005), and to take advantage of lexical head information (Collins, 2003). However, such *binarization* requires a set of language-specific rules, which hampers adaptation of parsing to other languages.

On the other hand, the process of top-down parsing can be regarded as *pre-order* traversal over a tree. Given the sentence in Figure 1, a top-down

[1] The action sequence is taken on unbinarized trees.

Transactions of the Association for Computational Linguistics, vol. 5, pp. 413–424, 2017. Action Editor: Brian Roark.
Submission batch: 6/2017; Published 11/2017.



(a) Syntactic tree

(b) *Binarized* syntactic tree

Figure 1: Syntactic trees of the sentence "The little boy likes red tomatoes.". (a) syntactic tree; (b) binarized syntactic tree, where r and l mean the head is the right branch and the left branch, respectively, and $*$ means this constituent is not completed.

stack	buffer	action	node
[]	[The little ...]	SHIFT	③
[The]	[little boy ...]	SHIFT	④
[The little]	[boy likes ...]	SHIFT	⑤
[... little boy]	[likes red ...]	REDUCE-R-NP	②
...	...	...	...

(a) bottom-up system

stack	buffer	action	node
[]	[The little ...]	NT-S	①
[(S]	[The little ...]	NT-NP	②
[(S (NP]	[The little ...]	SHIFT	③
[... (NP The]	[little boy ...]	SHIFT	④
[... The little]	[boy likes ...]	SHIFT	⑤
[... little boy]	[likes red ...]	REDUCE	/
...	...	...	...

(b) top-down system

stack	buffer	action	node
[]	[The little ...]	SHIFT	③
[The]	[little boy ...]	PJ-NP	②
[The NP]	[little boy ...]	SHIFT	④
[... NP little]	[boy likes...]	SHIFT	⑤
[... little boy]	[likes red ...]	REDUCE	/
...	...	...	...

(c) in-order system

Figure 2: Action sequences of three types of transition constituent parsing system. Details of the action system are introduced in Section 2.1, Section 2.2 and Section 3, respectively.

shift-reduce parser takes the action sequence in Table 2(b) to build the output, where an *S* is first made and then an *NP* is generated. After that, the system makes a decision to read the word sequence "The little boy" to complete the *NP*. The full order of recognition for the tree nodes is ①→②→③→④→⑤→⑥→⑦→⑧→⑨→⑩→⑪. The top-down lookahead guidance contributes to non-local disambiguation. However, it is difficult to generate a constituent before its sub constituents have been realized, since no explicit features can be extracted from their subtree structures. Thanks to the use of recurrent neural networks, which make it possible to represent a sentence globally before syntactic tree construction, seminal work of neural top-down parsing directly generates bracketed constituent trees using sequence-to-sequence models (Vinyals et al., 2015). Dyer et al. (2016) design a set of top-down transition actions for standard transition-based parsing.

In this paper, we propose a novel transition system for constituent parsing, mitigating issues of both bottom-up and top-down systems by finding a compromise between bottom-up constituent information and top-down lookahead information. The process of the proposed constituent parsing can be regarded as *in-order* traversal over a tree. Given the sentence in Figure 1, the system takes the action sequence in Table 2(c) to build the output. The system reads the word "The" and then projects an NP, which is based on bottom-up evidence. After this, based on the projected NP, the system reads the word sequence "little boy", with top-down guidance from NP. Similarly, based on the completed constituent "(NP The little boy)", the system projects an S, with the bottom-up evidence. With the S and the word "likes", the system projects

an VP, which can serve as top-down guidance. The full order of recognition for the tree nodes is ③→②→④→⑤→①→⑦→⑥→⑨→⑧→⑩→⑪. Compared to *post-order* traversal, *in-order* traversal can potentially resolve non-local ambiguity better by top-down guidance. Compared to *pre-order* traversal, *in-order* traversal can potentially resolve local ambiguity better by bottom-up evidence.

Furthermore, *in-order* traversal is psycholinguistically motivated (Roark et al., 2009; Steedman, 2000). Empirically, a human reader comprehends sentences by giving lookahead guesses for constituents. For example, when reading a word "likes", a human reader could guess that it could be a start of a constituent VP, instead of waiting to read the object "red tomatoes", which is the procedure of a bottom-up system.

We compare our system with the two baseline systems (i.e. a top-down system and a bottom-up system) under the same neural transition-based framework of Dyer et al. (2016). Our final models outperform both of the bottom-up and top-down transition-based constituent parsing by achieving a 91.8 F_1 in English and a 86.1 F_1 in Chinese for greedy fully-supervised parsing, respectively. Furthermore, our final model obtains a 93.6 F_1 with supervised reranking (Choe and Charniak, 2016) and a 94.2 F_1 with semi-supervised reranking, achieving the state-of-the-art results on constituent parsing on the English benchmark. By converting to Stanford dependencies, our final model achieves the state-of-the-art results on dependency parsing by obtaining a 96.2% UAS and a 95.2% LAS. To our knowledge, we are the first to systematically compare top-down and bottom-up constituent parsing under the same neural framework. We release our code at `https://github.com/LeonCrashCode/InOrderParser`.

2 Transition-based constituent parsing

Transition-based constituent parsing takes a left-to-right scan of the input sentence, where a stack is used to maintain partially constructed phrase-structures, while the input words are stored in a buffer. Formally, a *state* is defined as $[\sigma, i, f]$, where σ is the stack, i is the front index of the buffer, and f is a boolean value showing that the parsing is finished. At each step, a transition action is applied to consume an input word or construct a new phrase-structure. Different parsing systems employ their own sets of actions.

2.1 Bottom-up system

We take the bottom-up system of Sagae and Lavie (2005) as our bottom-up baseline. Given a state, the set of transition actions are:

- SHIFT: pop the front word from the buffer, and push it onto the stack.
- REDUCE-L/R-X: pop the top two constituents off the stack, combine them into a new constituent with label X, and push the new constituent onto the stack.
- UNARY-X: pop the top constituent off the stack, raise it to a new constituent with label X, and push the new constituent onto the stack.
- FINISH: pop the root node off the stack and end parsing.

The bottom-up parser can be summarized as the deductive system in Figure 3(a). Given the sentence with the binarized syntactic tree in Figure 1(b), the sequence of actions SHIFT, SHIFT, SHIFT, REDUCE-R-NP, REDUCE-R-NP, SHIFT, SHIFT, SHIFT, REDUCE-R-NP, REDUCE-L-VP, SHIFT, REDUCE-L-S, REDUCE-R-S and FINISH, can be used to construct its constituent tree.

2.2 Top-down system

We take the top-down system of Dyer et al. (2016) as our top-down baseline. Given a state, the set of transition actions are:

- SHIFT: pop the front word from the buffer, and push it onto the stack.
- NT-X: open a nonterminal with label X on top of the stack.
- REDUCE: repeatedly pop completed subtrees or terminal symbols from the stack until an open nonterminal is encountered, and then this open NT is popped and used as the label of a new constituent that has the popped subtrees as

$$\textsc{Shift} \quad \frac{[\sigma, i, false]}{[\sigma|w_i, i+1, false]}$$

$$\textsc{Reduce-l/r-X} \quad \frac{[\sigma|s_1|s_0, i, false]}{[\sigma|X_{s_1 s_0}, i, false]}$$

$$\text{Unary-X} \quad \frac{[\sigma|s_0, i, false]}{[\sigma|X_{s_0}, i, false]}$$

$$\textsc{Finish} \quad \frac{[\sigma, i, false]}{[\sigma, i, true]}$$

(a) bottom-up system

$$\textsc{Shift} \quad \frac{[\sigma, i, /]}{[\sigma|w_i, i+1, /]}$$

$$\text{NT-X} \quad \frac{[\sigma, i, /]}{(\sigma|X, i, /]}$$

$$\textsc{Reduce} \quad \frac{[\sigma|X|s_j|...|s_0, i, /]}{[\sigma|X_{s_j ... s_0}, i, /]}$$

(b) top-down system

$$\textsc{Shift} \quad \frac{[\sigma, i, false]}{[\sigma|w_i, i+1, false]}$$

$$\text{PJ-X} \quad \frac{[\sigma|s_0, i, false]}{(\sigma|s_0|X, i, false]}$$

$$\textsc{Reduce} \quad \frac{[\sigma|s_j|X|s_{j-1}|...|s_0, i, false]}{[\sigma|X_{s_j s_{j-1} ... s_0}, i, false]}$$

$$\textsc{Finish} \quad \frac{[\sigma, i, false]}{[\sigma, i, true]}$$

(c) in-order system

Figure 3: Different transition systems. The start state is $[\phi, 0, false]$ and the final state is $[\sigma, n, true]$.

its children. This new completed constituent is pushed onto the stack as a single composite item.

The deduction system for the process is shown in Figure 3(b)[2]. Given the sentence in Figure 1, the sequence of actions NT-S, NT-NP, SHIFT, SHIFT, SHIFT, REDUCE, NT-VP, SHIFT, NT-NP, SHIFT, SHIFT, REDUCE, REDUCE, SHIFT and REDUCE, can be used to construct its constituent tree.

[2]Due to unary decisions, with exception of the top-down system, we use completed marks to make the finish decisions.

3 In-order system

We propose a novel in-order system for transition-based constituent parsing. Similar to the bottom-up and top-down systems, the in-order system maintains a stack and a buffer for representing a state. The set of transition actions are defined as:

- SHIFT: pop the front word from the buffer, and push it onto the stack.
- PJ-X: project a nonterminal with label X on top of the stack.
- REDUCE: repeatedly pop completed subtrees or terminal symbols from the stack until a projected nonterminal encountered, and then this projected nonterminal is popped and used as the label of a new constituent. Furthermore, one more item on the top of stack is popped and inserted as the leftmost child of the new constituent. The popped subtrees are inserted as the rest of the children. This new completed constituent is pushed onto the stack as a single composite item.
- FINISH: pop the root node off the stack and end parsing.

The deduction system for the process is shown in Figure 3(c). Given the sentence in Figure 1, the sequence of actions SHIFT, PJ-NP, SHIFT, SHIFT, REDUCE, PJ-S, SHIFT, PJ-VP, SHIFT, PJ-NP, SHIFT, REDUCE, REDUCE, SHIFT, REDUCE, FINISH can be used to construct its constituent tree.

Variants The in-order system can be generalized into variants by modifying k, the number of leftmost nodes traced before the parent node. For example, given the tree "(S a b c d)", the traversal is "a S b c d" if $k = 1$ while the traversal is "a b S c d" if $k = 2$. We name each variant with a certain k value as k-in-order systems. In this paper, we only investigate the in-order system with $k = 1$, the 1-in-order system. Note that the top-down parser can be regarded as a special case of a generalized version of the in-order parser with $k = 0$, and the bottom-up parser can be regarded as a special case with $k = \infty$.

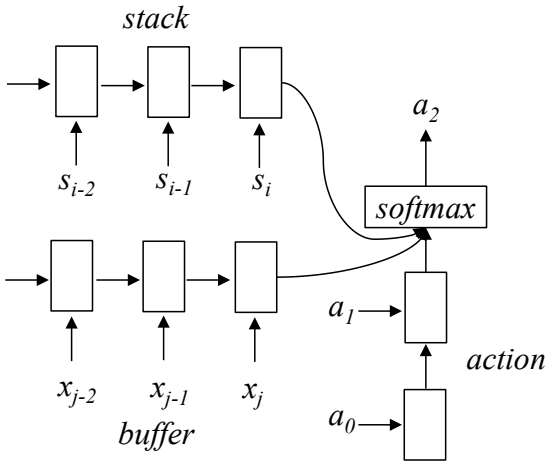
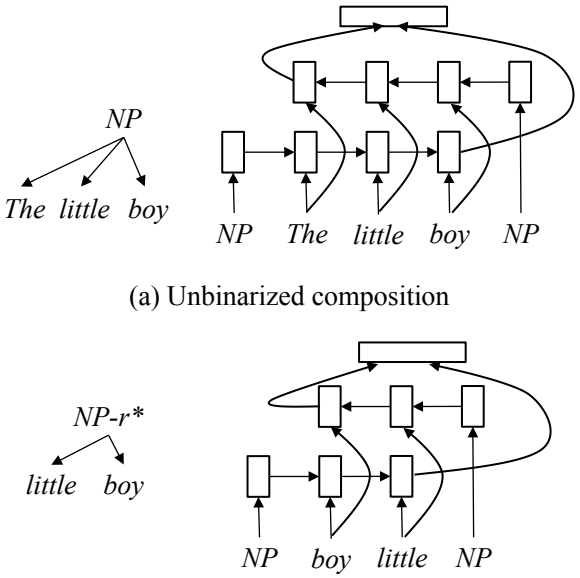


Figure 4: Framework of our transition-based parsers.



(a) Unbinarized composition

(b) Binarized composition

Figure 5: The composition function. (a) is for unbinarized trees and (b) is for binarized trees, where "NP-r*" means that "little boy" is a non-completed noun phrase with head "boy".

4 Neural parsing model

We employ the stack-LSTM parsing model of Dyer et al. (2016) for the three types of transition-based parsing systems in Section 2.1, 2.2 and 3, respectively, where a stack-LSTM is used to represent the stack, a stack-LSTM is used to represent the buffer, and a vanilla LSTM is used to represent the action history, as shown in Figure 4.

4.1 Word representation

We follow Dyer et al. (2015), representing each word using three different types of embeddings, including pretrained word embedding, $\overline{e}_{w_i}$, which is not fine-tuned during the training of the parser, randomly initialized embeddings e_{w_i}, which is fine-tuned, and the randomly initialized part-of-speech embeddings, which is fine-tuned. The three embeddings are concatenated, and then fed to nonlinear layer to derive the final word embedding:

$$x_i = f(W_{input}[e_{p_i}; \overline{e}_{w_i}; e_{w_i}] + b_{input}),$$

where W_{input} and b_{input} are model parameters, w_i and p_i denote the form and the POS tag of the ith input word, respectively, and f is an nonlinear function. In this paper, we use ReLu for f.

4.2 Stack representation

We employ a bidirectional LSTM as the composition function to represent constituents on stack[3]. For top-down parsing and in-order parsing, following Dyer et al. (2016), as shown in Figure 5(a), the composition representation s_{comp} is computed as:

$$s_{comp} = \begin{array}{l}(\text{LSTM}_{fwd}[e_{nt}, s_0, ..., s_m]; \\ \text{LSTM}_{bwd}[e_{nt}, s_m, ..., s_0]),\end{array}$$

where e_{nt} is the representation of a non-terminal, $s_j, j \in [0, m]$ is the jth child node, and m is the number of the child nodes. For bottom-up parsing, we make use of the head information in the composition function by requiring the order that the head node is always before the non-head node in the bidirectional LSTM, as shown in Figure 5(b)[4]. The bi-

[3]To be fair, we use a bidirectional LSTM as composition function for all parsing systems

[4]A bidirectional LSTM consists of two LSTMs, making it balanced for composition. However, they have different parameters so that one represents information of head-first while other represents information of head-last.

narized composition is computed as:

$$s_{bcomp} = (\text{LSTM}_{fwd}[e_{nt}, s_h, s_o]; \text{LSTM}_{bwd}[e_{nt}, s_o, s_h]),$$

where s_h and s_o is the representation of the head and the non-head node, respectively.

4.3 Greedy action classification

Given a sentence $w_0, w_1, ..., w_{n-1}$, where w_i is the ith word, and n is the length of the sentence, our parser makes local action classification decisions incrementally. For the kth parsing state like $[s_j, ..., s_1, s_0, i, \textit{false}]$, the probability distribution of the current action p is:

$$p = \text{SOFTMAX}(W[h_{stk}; h_{buf}; h_{ah}] + b), \quad (*)$$

where W and b are model parameters, the representation of stack information h_{stk} is:

$$h_{stk} = \text{stack-LSTM}[s_0, s_1, ..., s_j],$$

the representation of buffer information h_{buf} is:

$$h_{buf} = \text{stack-LSTM}[x_i, x_{i+1}, ..., x_n],$$

x is the word representation, and the representation of action history h_{ah} is:

$$h_{ah} = \text{LSTM}[e_{act_{k-1}}, e_{act_{k-2}}, ..., e_{act_0}],$$

where $e_{act_{k-1}}$ is the representation of action in the k-1th parsing state.

Training Our models are trained to minimize a cross-entropy loss objective with an l_2 regularization term, defined by

$$L(\theta) = -\sum_i \sum_j log\ p_{a_{ij}} + \frac{\lambda}{2}||\theta||^2,$$

where θ is the set of parameters, $p_{a_{ij}}$ is the probability of the jth action in the ith training example given by the model and λ is a regularization hyper-parameter ($\lambda = 10^{-6}$). We use stochastic gradient descent with a 0.1 initialized learning rate with a 0.05 learning rate decay.

Parameter	Value
LSTM layer	2
Word embedding dim	32
English pretrained word embedding dim	100
Chinese pretrained word embedding dim	80
POS tag embedding dim	12
Action embedding dim	16
Stack-LSTM input dim	128
Stack-LSTM hidden dim	128

Table 1: Hyper-parameters.

5 Experiments

5.1 Data

We empirically compare our bottom-up, top-down and in-order parsers. The experiments are carried out on both English and Chinese. For English data, we use the standard benchmark of WSJ sections in PTB (Marcus et al., 1993), where the Sections 2-21 are taken for training data, Section 22 for development data and Section 23 for testing both dependency parsing and constituency parsing. We adopt the pretrained English word embeddings generated on the AFP portion of English Gigaword.

For Chinese data, we use Version 5.1 of the Penn Chinese Treebank (CTB) (Xue et al., 2005). We use articles 001- 270 and 440-1151 for training, articles 301-325 for system development, and articles 271-300 for final performance evaluation. We adopt the pretrained Chinese word embeddings generated on the complete Chinese Gigaword corpus.

The POS tags in both the English data and the Chinese data are automatically assigned the same as the work of Dyer et al. (2016), using Stanford tagger. We follow the work of Choe and Charniak (2016) and adopt the AFP portion of the English Gigaword as the extra resources for the semi-supervised reranking.

5.2 Settings

Hyper-parameters For both English and Chinese experiments, we use the same hyper-parameters as the work of Dyer et al. (2016) without further optimization, as shown in Table 1.

Reranking experiments Following the same reranking setting of Dyer et al. (2016) and Choe and Charniak (2016), we obtain 100 samples from our bottom-up, top-down, and in-order model (Sec-

Model	LR	LP	F_1
Top-down parser	91.59	91.66	91.62
Bottom-up parser	91.89	91.83	91.86
In-order parser	91.98	91.86	91.92

Table 2: Development results (%) on WSJ 22.

Model	F_1
fully-supervise	
Top-down parser	91.2
Bottom-up parser	91.3
In-order parser	91.8
rerank	
Top-down parser	93.3
Bottom-up parser	93.3
In-order parser	93.6

Table 3: Final results (%) on WSJ Section 23.

tion 4), respectively, with an exponentiation strategy ($\alpha = 0.8$), by using the probability distribution of action (equation *). We adopt the reranker of Choe and Charniak (2016) as both our English fully-supervised reranker and semi-supervised reranker, and the generative reranker of Dyer et al. (2016) as our Chinese supervised reranker.

5.3 Development experiments

Table 2 shows the development results of the three parsing systems. The bottom-up system performs slightly better than the top-down system. The in-order system outperforms both the bottom-up and the top-down system.

5.4 Results

Table 3 shows the parsing results on the English test dataset. We find that the bottom-up parser and the top-down parser have similar results under the greedy setting, and the in-order parser outperforms both of them. Also, with supervised reranking, the in-order parser achieves the best results.

English constituent results We compare our models with previous work, as shown in Table 4. With the fully-supervise setting[5], the in-order parser outperforms the state-of-the-art discrete parser (Shindo et al., 2012; Zhu et al., 2013), the state-of-the-art neural parsers (Cross and Huang, 2016; Watanabe and Sumita, 2015) and the state-of-the-art hybrid parsers (Durrett and Klein, 2015; Liu and Zhang, 2017), achieving state-of-the-art results. With the reranking setting, the in-order parser outperforms the best discrete parser (Huang, 2008) and has the same performance as Kuncoro et al. (2017), which extends the work of Dyer et al. (2016) by adding a gated attention mechanism on composition functions. With the semi-supervised setting, the in-order parser outperforms the best semi-supervised parser (Choe and Charniak, 2016) by achieving 94.2 F_1 (the oracle is 97.9 F_1).

[5]Here, we only consider the work of a single model.

Model	F_1
fully-supervise	
Socher et al. (2013)	90.4
Zhu et al. (2013)	90.4
Vinyals et al. (2015)	90.7
Watanabe and Sumita (2015)	90.7
Shindo et al. (2012)	91.1
Durrett and Klein (2015)	91.1
Dyer et al. (2016)	91.2
Cross and Huang (2016)	91.3
Liu and Zhang (2017)	91.7
Top-down parser	91.2
Bottom-up parser	91.3
In-order parser	**91.8**
reranking	
Huang (2008)	91.7
Charniak and Johnson (2005)	91.5
Choe and Charniak (2016)	92.6
Dyer et al. (2016)	93.3
Kuncoro et al. (2017)	**93.6**
Top-down parser	93.3
Bottom-up parser	93.3
In-order parser	**93.6**
semi-supervised reranking	
Choe and Charniak (2016)	93.8
In-order parser	**94.2**

Table 4: Final results (%) on WSJ Section 23.

English dependency results As shown in Table 5, by converting to Stanford Dependencies, without additional training data, our models achieve a similar performance with the state-of-the-art system (Choe and Charniak, 2016); with the same additional training data, our models achieve new state-of-the-art results on dependency parsing by achieving 96.2% UAS and 95.2% LAS on standard benchmark.

Chinese constituent results Table 6 shows the final results on the Chinese test dataset. The in-

Model	UAS	LAS
Kiperwasser and Goldberg (2016)†	93.9	91.9
Cheng et al. (2016) †	94.1	91.5
Andor et al. (2016)	94.6	92.8
Dyer et al. (2016) -re	95.6	94.4
Dozat and Manning (2017)†	95.7	94.0
Kuncoro et al. (2017) -re	95.7	94.5
Choe and Charniak (2016) -sre	95.9	94.1
In-order parser	94.5	93.4
In-order parser -re	95.9	94.9
In-order parser -sre	**96.2**	**95.2**

Table 5: Stanford Dependency accuracy (%) on WSJ Section 23. † means graph-based parsing. "-re" means fully-supervised reranking and "-sre" means semi-supervised reranking.

Parser	F_1
fully-supervision	
Zhu et al. (2013)	83.2
Wang et al. (2015)	83.2
Dyer et al. (2016)	84.6
Liu and Zhang (2017)	85.5
Top-down parser	84.6
Bottom-up parser	85.7
In-order parser	**86.1**
rerank	
Charniak and Johnson (2005)	82.3
Dyer et al. (2016)	86.9
Top-down parser	86.9
Bottom-up parser	87.5
In-order parser	**88.0**
semi-supervision	
Zhu et al. (2013)	85.6
Wang and Xue (2014)	86.3
Wang et al. (2015)	86.6

Table 6: Final results on test set of CTB.

order parser achieves the best results under the fully-supervised setting. With the supervised reranking, the in-order parser outperforms the state-of-the-art models by achieving 88.0 F_1 (the oracle is 93.47 F_1).

Chinese dependency results As shown in Table 7, by converting the results to dependencies[6], our final model achieves the best results among transition-based parsing, and obtains comparable results to the state-of-the-art graph-based models.

[6]The Penn2Malt tool is used with Chinese head rules https://stp.lingfil.uu.se/ nivre/research/Penn2Malt.html.

Model	UAS	LAS
Dyer et al. (2016)	85.5	84.0
Ballesteros et al. (2016)	87.7	86.2
Kiperwasser and Goldberg (2016)	87.6	86.1
Cheng et al. (2016) †	88.1	85.7
Dozat and Manning (2017) †	89.3	88.2
In-order parser	87.4	86.4
In-order parser -re	**89.4**	**88.4**

Table 7: Dependency accuracy (%) on CTB test set. † means graph-based parsing. "-re" means supervised reranking.

6 Analysis

We analyze the results of Section 23 in WSJ given by our model (i.e. in-order parser) and two baseline models (i.e. the bottom-up parser and the top-down parser) against the sentence length, the span length and the constituent type, respectively.

6.1 Influence of sentence length

Figure 6 shows the F_1 scores of the three parsers on sentences of different lengths. Compared to the top-down parser, the bottom-up parser performs better on the short sentences with the length falling in the range [20-40]. This is likely because the bottom-up parser takes advantages of rich local features from partially-built trees, which are useful for parsing short sentences. However, these local structures are can be insufficient for parsing long sentences due to error propagation. On the other hand, the top-down parser obtains better results on long sentences with the length falling in the range [40-50]. This is because, as the length of sentences increase, lookahead features become rich and they could be correctly represented by the LSTM, which is beneficial for parsing non-local structures. We find that the in-order parser performs the best for both short and long sentences, showing the advantages of integrating bottom-up and top-down information.

6.2 Influence of span length

Figure 7 shows the F_1 scores of the three parsers on spans of different lengths. The trend of performances of the two baseline parsers are similar. Compared to the baseline parsers, the in-order parser obtains significant improvement on long spans. Linguistically, it is because the in-order traversal, (over

	NP	VP	S	PP	SBAR	ADVP	ADJP	WHNP	QP
Top-down parser	92.87	92.51	91.36	87.96	86.74	85.21	75.41	96.44	89.41
Bottom-up parser	93.01	92.20	91.46	87.95	86.81	84.58	74.84	94.99	89.95
In-order parser	93.23	92.83	91.87	88.97	88.05	86.30	76.62	96.75	92.16
Improvement	+0.22	+0.32	+0.41	+1.01	+1.04	+1.09	+1.21	+0.31	+2.01

Table 8: Comparison on different phrases types.

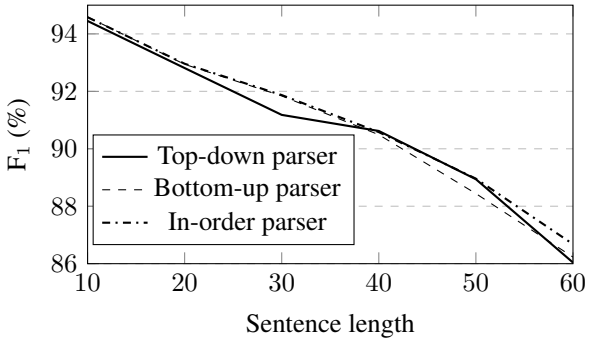


Figure 6: F_1 score against sentence length. (the number of words in a sentence, in bins of size 10, where 20 contains sentences with lengths in [10, 20).)

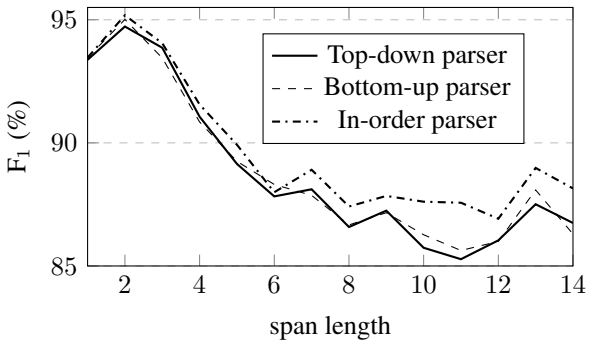


Figure 7: F_1 score against span length.

a tree) allows constituent types of spans to be correctly projected based on the information of the beginning (leftmost nodes) of the spans. Then the projected constituents constrain long span construction, which is different from the top-down parser, generating constituent types of spans without trace of the spans.

6.3 Influence of constituent type

Table 7 shows the F_1 scores of the three parsers on frequent constituent types. The bottom-up parser performs better than the top-down parser on constituent types including NP, S, SBAR, QP. We find that the prediction of these constituent types requires, explicitly, modeling of bottom-up structures. In other words, bottom-up information is necessary for us to know if the span can be a noun phrase (NP) or sentence (S), for example. On the other hand, the top-down parser has better performance on WHNP, which is likely because a WHNP starts with a certain question word, making the prediction easy without bottom-up information. The in-order parser performs the best on all constituent types, demonstrating that the in-order parser can benefit from both bottom-up and top-down information.

6.4 Examples

We give output examples from the test set to qualitatively compare the performances of the three parsers using the fully-supervised model without reranking, as shown in Table 9. For example, given the Sentence #2006, the bottom-up and the in-order parsers give both correct results. However, the top-down parser makes an incorrect decision to generate an S, leading to subsequent incorrect decisions on VP to complete S. Sentence pattern ambiguity allows top-down guidance to over-parsing the sentence by recognizing the word "Plans" as a verb, while more bottom-up information is useful for the local disambiguation.

Given the Sentence #308, the bottom-up parser prefers construction of local constituents such as "once producers and customers", ignoring the possible clause SBAR, however, which is captured by the in-order parser. The parser projects a constituent SBAR from the word "stick" and continues to complete the clause, showing that top-down lookahead information is necessary for non-local disambiguation. The in-order parser gives the correct output for the Sentence #2066 and the Sentence #308, showing that it can benefit from bottom-up and top-down information.

Sent #2066	Employee Benefit Plans Inc. –
Gold	(NP Employee Benefit Plans Inc. -)
Top-down	(S (NP Employee Benefit) (VP Plans (NP Inc.) –))
Bottom-up	(NP Employee Benefit Plans Inc. -)
In-order	(NP Employee Benefit Plans Inc. -)
Sent #308	... whether the new posted prices will stick once producers and customers start to haggle .
Gold	... (VP will (VP stick (SBAR once (S (NP producers and customers) (VP start (S ...)))))) ...
Top-down	... (VP will (VP stick (SBAR once (S (NP producers and customers) (VP start (S ...)))))) ...
Bottom-up	... (VP will (VP stick (NP once producers and customers))) ... (VP start (S ...)) ...
In-order	... (VP will (VP stick (SBAR once (S (NP producers and customers) (VP start (S ...)))))) ...
Sent #1715	This has both made investors uneasy and the corporations more vulnerable .
Gold	(S (NP This) (VP has (VP both made (S (S investors uneasy) and (S the corporations ...)))) .)
Top-down	(S (S (NP This) (VP has (S (NP both) (VP made investors uneasy)))) and (S the corporations ...) .)
Bottom-up	(S (S (NP This) (VP has (S both (VP made investors uneasy)))) and (S the corporations ...) .)
In-order	(S (NP This) (VP has both (VP made (S (S investors uneasy) and (S the corporations ...))))) .)

Table 9: Output examples of the three parsers on the English test set. Incorrect constituents are marked in red.

In the Sentence #1715, there are coordinated objects such as "investors uneasy" and "the corporations more vulnerable". All of the three parsers can recognize coordination. However, the top-down and the bottom-up parsers incorrectly recognize the "This has both made investors uneasy" as a complete sentence. The top-down parser incorrectly generates S, marked in red, at a early stage, leaving no choice but to follow this incorrect non-terminal. The bottom-up parser without lookahead information makes incorrect local decisions. By contrast, the in-order parser reads the word "and" and projects a non-terminal S for coordination after completing "(S investors uneasy)". On the other hand, the in-order parser is confused by projecting for the word "made" or the word "both" into an VP, which we think could be addressed by using a in-order system variant with k=2 described in Section 3.

7 Related work

Our work is related to left corner parsing. Rosenkrantz and Lewis (1970) formalize this in automata theory, which have appeared frequently in the compiler literature. Roark and Johnson (1999) apply the strategy into parsing. Typical works investigate the transformation of syntactic trees based on left-corner rules (Roark, 2001; Schuler et al., 2010; Van Schijndel and Schuler, 2013). In contrast, we propose a novel general transition-based in-order constituent parsing system.

Neural networks have achieved the state-of-the-art for parsing under various grammar formalisms, including dependency (Dozat and Manning, 2017), constituent (Dyer et al., 2016; Kuncoro et al., 2017), and CCG parsing (Xu, 2016; Lewis et al., 2016). Seminal work employs transition-based methods (Chen and Manning, 2014). This method has been extended by investigating more complex representations of configurations for constituent parsing (Watanabe and Sumita, 2015; Dyer et al., 2016). Dyer et al. (2016) employ stack-LSTM onto the top-down system, which is the same as our top-down parser. Watanabe and Sumita (2015) employ tree-LSTM to model the complex representation in the stack in bottom-up system. We are the first to investigate in-order traversal by designing a novel transition-based system under the same neural structure model framework.

8 Conclusion

We proposed a novel psycho-linguistically motivated constituent parsing system based on the in-order traversal over syntactic trees, aiming to find a compromise between bottom-up constituent information and top-down lookahead information. On the standard WSJ benchmark, our in-order system outperforms bottom-up parsing on a non-local ambiguity and top-down parsing on local decision. The resulting parser achieves the state-of-the-art constituent parsing results by obtaining 94.2 F_1 and dependency parsing results by obtaining 96.2% UAS and 95.2% LAS.

Acknowledgments

We thank the anonymous reviewers for their detailed and constructive comments and the action editor Brian Roark. Yue Zhang is the corresponding author.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *ACL*, pages 2442–2452.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. 2016. Training with exploration improves a greedy stack-LSTM parser. In *EMNLP*, pages 2005–2010.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *ACL*, pages 173–180.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *NAACL*, pages 132–139.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *EMNLP*, pages 2204–2214.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *EMNLP*, pages 2331–2336.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *EMNLP*, pages 1–11.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *ICLR*.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *ACL*, pages 302–312.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL*, pages 334–343.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *NAACL*, pages 199–209

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association of Computational Linguistics*, 4:313–327.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *EACL*, pages 1249–1258.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *NAACL*, pages 221–231.

Jiangming Liu and Yue Zhang. 2017. Shift-Reduce Constituent Parsing with Neural Lookahead Features. *Transactions of the Association of Computational Linguistics*, 5:45–58.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Brian Roark and Mark Johnson. 1999. Efficient probabilistic top-down and left-corner parsing. In *ACL*, pages 421–428.

Brian Roark, Asaf Bachrach, Carlos Cardenas, and Christophe Pallier. 2009. Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing. In *EMNLP*, pages 324–333.

Brian Roark. 2001. Robust probabilistic predictive syntactic processing: motivations, models, and applications. In *Ph.D. thesis*.

Daniel J. Rosenkrantz and Philip M. Lewis. 1970. Deterministic left corner parsing. In *IEEE Conference Record of 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *IWPT*, pages 125–132.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints. *Computational Linguistics*, 36(1):1–30.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *ACL*, pages 440–448.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with compositional vector grammars. In *ACL*, pages 455–465.

Mark Steedman. 2000. *The syntactic process*, volume 24. MIT Press.

Marten Van Schijndel and William Schuler. 2013. An analysis of frequency-and memory-based processing costs. In *HLT-NAACL*, pages 95–105.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *ICLR*.

Zhiguo Wang and Nianwen Xue. 2014. Joint POS tagging and transition-based constituent parsing in Chinese with non-local features. In *ACL*, pages 733–742.

Zhiguo Wang, Haitao Mi, and Nianwen Xue. 2015. Feature optimization for constituent parsing via neural networks. In *ACL-IJCNLP*, pages 1138–1147.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *ACL*, pages 1169–1179.

Wenduan Xu. 2016. LSTM shift-reduce CCG parsing. In *EMNLP*, pages 1754–1764.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Martha Palmer. 2005. The Penn Chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *IWPT*, pages 162–171.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL*, pages 434–443.