Arc-Standard Spinal Parsing with Stack-LSTMs

Miguel Ballesteros
IBM T.J Watson Research Center
Yorktown Heights, NY 10598. U.S.
miguel.ballesteros@ibm.com

Xavier Carreras
Naver Labs Europe
Meylan, France
xavier.carreras@naverlabs.com

Abstract

We present a neural transition-based parser for spinal trees, a dependency representation of constituent trees. The parser uses Stack-LSTMs that compose constituent nodes with dependency-based derivations. In experiments, we show that this model adapts to different styles of dependency relations, but this choice has little effect for predicting constituent structure, suggesting that LSTMs induce useful states by themselves.

1 Introduction

There is a clear trend in neural transition systems for parsing sentences into dependency trees (Titov and Henderson, 2007; Chen and Manning, 2014; Dyer et al., 2015; Andor et al., 2016) and constituent trees (Henderson, 2004; Vinyals et al., 2014; Watanabe and Sumita, 2015; Dyer et al., 2016; Cross and Huang, 2016b). These transition systems use a relatively simple set of operations to parse in linear time, and rely on the ability of neural networks to infer and propagate hidden structure through the derivation. This contrasts with state-of-the-art factored linear models, which explicitly use of higher-order information to capture non-local phenomena in a derivation.

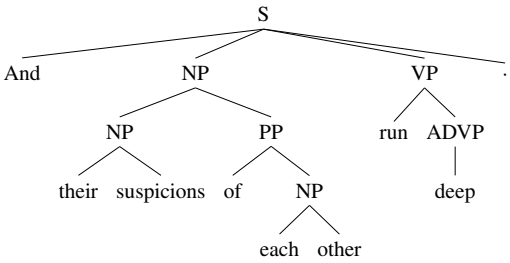
In this paper, we present a transition system for parsing sentences into spinal trees, a type of syntactic tree that explicitly represents together dependency and constituency structure. This representation is inherent in head-driven models (Collins, 1997) and was used by Carreras et al. (2008) with a higher-order factored model. We extend the Stack-LSTMs by Dyer et al. (2015) from dependency to spinal parsing, by augmenting the composition operations to include constituent information in the form of spines. To parse sentences, we use the extension by Cross and Huang (2016a) of the arc-standard system for dependency parsing (Nivre, 2004). This parsing system generalizes shift-reduce methods (Henderson, 2003; Sagae and Lavie, 2005; Zhu et al., 2013; Watanabe and Sumita, 2015) to be sensitive to constituent heads, as opposed to, for example, parse a constituent from left to right.

In experiments on the Penn Treebank, we look at how sensitive our method is to different styles of dependency relations, and show that spinal models based on leftmost or rightmost heads are as good or better than models using linguistic dependency relations such as Stanford Dependencies (De Marneffe et al., 2006) or those by Yamada and Matsumoto (2003). This suggests that Stack-LSTMs figure out effective ways of modeling non-local phenomena within constituents. We also show that turning a dependency Stack-LSTM into spinal results in some improvements.

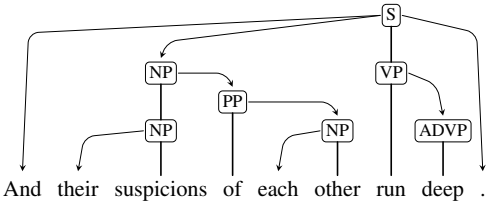
2 Spinal Trees

In a spinal tree each token is associated with a spine. The spine of a token is a (possibly empty) vertical sequence of non-terminal nodes for which the token is the head word. A spinal dependency is a binary directed relation from a node of the head spine to a dependent spine. In this paper we consider projective spinal trees. Figure 1 shows a constituency tree from the Penn Treebank together with two spinal trees that use alternative head identities: the spinal tree in 1b uses Stanford Dependencies (De Marneffe et al., 2006), while the spinal tree in 1c takes the leftmost word of any constituent as the head. It is direct to map a constituency tree with head annotations to a spinal tree, and to map a spinal tree to a constituency or a dependency tree.

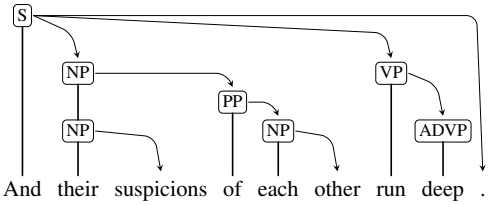
Proceedings of the 15th International Conference on Parsing Technologies, pages 115–121,
Pisa, Italy; September 20–22, 2017.



(a) A constituency tree from the Penn Treebank.



(b) The spinal tree of (1a) using Stanford Dependency heads.



(c) The spinal tree of (1a) using leftmost heads.

Figure 1: A constituency tree and two spinal trees.

3 Arc-Standard Spinal Parsing

We use the transition system by Cross and Huang (2016a), which extends the arc-standard system by Nivre (2004) for constituency parsing in a head-driven way, i.e. spinal parsing. We describe it here for completeness. The parsing state is a tuple $\langle \beta, \sigma, \delta \rangle$, where β is a buffer of input tokens to be processed; σ is a stack of tokens with partial spines; and δ is a set of *spinal* dependencies. The operations are the following:

- `shift` : $\langle i{:}\beta,\ \sigma,\ \delta \rangle \rightarrow \langle \beta,\ \sigma{:}i,\ \delta \rangle$

  Shifts the first token of the buffer i onto the stack, i becomes a base spine consisting of a single token.

- `node(`n`)` : $\langle \beta,\ \sigma{:}s,\ \delta \rangle \rightarrow \langle \beta,\ \sigma{:}s{+}n,\ \delta \rangle$

  Adds a non-terminal node n onto the top element of the stack s, which becomes $s{+}n$. At this point, the node n can receive left and right children (by the operations below) until the node is closed (by adding a node above, or by reducing the spine with an arc operation with this spine as dependent). By this single operation, the arc-standard system is extended to spinal parsing.

- `left-arc` : $\langle \beta,\ \sigma{:}t{:}s{+}n,\ \delta \rangle \rightarrow \langle \beta,\ \sigma{:}s{+}n,\ \delta \cup (n,t) \rangle$

  The stack must have two elements, the top one is a spine $s{+}n$, where n is the top node of that spine, and the second element t can be a token or a spine. The operation adds a spinal from the node n to t, and t is reduced from the stack. The dependent t becomes the leftmost child of the constituent n.

- `right-arc` : $\langle \beta,\ \sigma{:}s{+}n{:}t,\ \delta \rangle \rightarrow \langle \beta,\ \sigma{:}s{+}n,\ \delta \cup (n,t) \rangle$

  This operation is symmetric to `left-arc`, it adds a spinal dependency from the top node n of the second spine in the stack to the top element t, which is reduced from the stack and becomes the rightmost child of n.

At a high level, the system builds a spinal tree headed at token i by:

1. Shifting the i-th token to the top of the stack. By induction, the left children of i are in the stack and are complete.
2. Adding a constituency node n to i's spine.
3. Adding left children to n in head-outwards order with `left-arc`, which are removed from the stack.
4. Adding right children to n in head-outwards order with `right-arc`, which are built recursively.
5. Repeating steps 2-4 for as many nodes in the spine of i.

Transition	Buffer β	Stack σ	New Arc in δ
	[And, their, ...]	[]	
`shift`	[their, suspicions, ...]	[And]	
`shift`	[suspicions, of, ...]	[And, their]	
`shift`	[of, each, ...]	[..., their, susp.]	
`node(NP)`	[of, each, ...]	[..., their, susp.+NP^1_3]	
`left-arc`	[of, each, ...]	[And, susp.+NP^1_3]	(NP^1_3,their)
`node(NP)`	[of, each, ...]	[And, susp.+NP^1_3+NP^2_3]	
`shift`	[each, other, ...]	[..., susp.+NP^1_3+NP^2_3, of]	
`node(PP)`	[each, other, ...]	[..., susp.+NP^1_3+NP^2_3, of+PP^1_4]	
`shift`	[other, run, ...]	[..., of+PP^1_4, each]	
`shift`	[run, deep, ...]	[..., each, other]	
`node(NP)`	[run, deep, ...]	[..., each, other+NP^1_6]	
`left-arc`	[run, deep, ...]	[..., of+PP^1_4, other+NP^1_6]	(NP^1_6, each)
`right-arc`	[run, deep, ...]	[..., susp.+NP^1_3+NP^2_3, of+PP^1_4]	(PP^1_4, NP^1_6)
`right-arc`	[run, deep, ...]	[And, susp.+NP^1_3+NP^2_3]	(NP^2_3, PP^1_4)
...			

Figure 2: Initial steps of the arc-standard derivation for the spinal tree in Figure 1b, until the tree headed at "suspicions" is fully built. Spinal nodes are noted n^j_i, where n is the non-terminal, i is the position of the head token, and j is the node level in the spine.

Figure 2 shows an example of a derivation. The process is initialized with all sentence tokens in the buffer, an empty stack, and an empty set of dependencies. Termination is always attainable and occurs when the buffer is empty and there is a single element in the stack, namely the spine of the full sentence head. This transition system is correct and sound with respect to the class of projective spinal trees, in the same way as the arc-standard system is for projective dependency trees (Nivre, 2008). A derivation has $2n + m$ steps, where n is the sentence length and m is the number of constituents in the derivation.

We note that the system naturally handles constituents of arbitrary arity. In particular, unary productions add one node in the spine without any children. In practice we put a hard bound on the number of consecutive unary productions in a spine[1], to ensure that in the early training steps the model does not generate unreasonably long spines. We also note there is a certain degree of non-determinism: left and right arcs (steps 3 and 4) can be mixed as long as the children of a node are added in head-outwards order. At training time, our *oracle derivations* impose the order above (first left arcs, then right arcs), but the parsing system runs freely. Finally, the system can be easily extended with dependency labels, but we do not use them.

4 Spinal Stack-LSTMs

Dyer et al. (2015) presented an arc-standard parser that uses Stack-LSTMs, an extension of LSTMs (Hochreiter and Schmidhuber, 1997) for transition-based systems that maintains an embedding for each element in the stack.[2]. Our model is based on the same architecture, with the addition of the `node(`n`)` action. The state of our algorithm presented in Section 3 is represented by the contents of the STACK, the BUFFER and a list with the history of actions with Stack-LSTMs. This state representation is then used to predict the next action to take.

Composition: when the parser predicts a `left-arc()` or `right-arc()`, we compose the vector representation of the head and dependent elements; this is equivalent to what it is presented by Dyer et al. (2015). The representation is obtained recursively as follows:

$$\mathbf{c} = \tanh\left(\mathbf{U}[\mathbf{h}; \mathbf{d}] + \mathbf{e}\right).$$

where $\mathbf{U}$ is a learned parameter matrix, $\mathbf{h}$ represents the head spine and $\mathbf{d}$ represents the dependent spine (or token, if the dependent is just a single token) ; $\mathbf{e}$ is a bias term.

Similarly, when the parser predicts a `node(`n`)` action, we compose the embedding of the non-terminal symbol that is added ($\mathbf{n}$) with the representation of the element at the top of the stack ($\mathbf{s}$), that might represent a spine or a single terminal symbol. The representation is obtained recursively as follows:

$$\mathbf{c} = \tanh\left(\mathbf{W}[\mathbf{s}; \mathbf{n}] + \mathbf{b}\right). \quad (1)$$

where $\mathbf{W}$ is a learned parameter matrix, $\mathbf{s}$ represents the token in the stack (and its partial spine, if non-terminals have been added to it) and $\mathbf{n}$ represents the non-terminal symbol that we are adding to $\mathbf{s}$; $\mathbf{b}$ is a bias term.

As shown by Kuncoro et al. (2017) composition is an essential component in this kind of parsing models.

5 Related Work

Collins (1997) first proposed head-driven derivations for constituent parsing, which is the key idea for spinal parsing, and later Carreras et al. (2008) came up with a higher-order graph-based parser for this representation. Transition systems for spinal parsing are not new. Ballesteros and Carreras (2015) presented an arc-eager system that labels dependencies with constituent nodes, and builds the spinal tree in post-processing. Hayashi et al. (2016) and Hayashi and Nagata (2016) presented a bottom-up arc-standard system that assigns a full spine with the `shift` operation, while ours builds spines incrementally and does not depend on a fixed set of full spines. Our method is different from shift-reduce constituent parsers (Henderson, 2003; Sagae and Lavie, 2005; Zhu et al., 2013; Watanabe and Sumita, 2015) in that it is head-driven. Cross and Huang (2016a) extended the arc-standard system to constituency parsing, which in fact corresponds to spinal parsing. The main difference from that work relies on the neural model: they use sequential BiLSTMs, while we use Stack-LSTMs and composition functions. Finally, dependency parsers have been extended to

[1] Set to 10 in our experiments

[2] We refer interested readers to (Dyer et al., 2015; Ballesteros et al., 2017).

	LR	LP	F1	UAS (SD)
Leftmost heads	91.18	90.93	91.05	-
Leftmost h., no n-comp	90.20	90.76	90.48	-
Rightmost heads	91.03	91.20	91.11	-
Rightmost h., no n-comp	90.64	91.24	90.04	-
SD heads	90.75	91.11	90.93	93.49
SD heads, no n-comp	90.38	90.58	90.48	93.16
SD heads, dummy spines	-	-	-	93.30
YM heads	90.82	90.84	90.83	-

Table 1: Development results for spinal models, in terms of labeled precision (LP), recall (LR) and F1 for constituents, and unlabeled attachment score (UAS) against Stanford dependencies. Spinal models are trained using different head annotations (see text). Models labeled with *"no n-comp"* do not use node compositions. The model labeled with *"dummy spines"* corresponds to a standard dependency model.

constituency parsing by encoding the additional structure in the dependency labels, in different ways (Hall et al., 2007; Hall and Nivre, 2008; Fernández-González and Martins, 2015).

6 Experiments

We experiment with stack-LSTM spinal models trained with different types of head rules. Our goal is to check how the head identities, which define the derivation sequence, interact with the ability of Stack-LSTMs to propagate latent information beyond the local scope of each action. We use the Penn Treebank (Marcus et al., 1993) with standard splits.[3]

We start training four spinal models, varying the head rules that define the spinal derivations:[4]

- Leftmost heads as in Figure 1c.
- Rightmost heads.
- Stanford Dependencies (SD) (De Marneffe et al., 2006), as in Figure 1b.
- Yamada and Matsumoto heads (YM) (Yamada and Matsumoto, 2003).

Table 1 presents constituency and dependency metrics on the development set. The model using rightmost heads works the best at 91.11 F1, followed by the one using leftmost heads. It is worth to note that the two models using structural head identities (right or left) work better than those using linguistic ones. This suggests that the Stack-LSTM model already finds useful head-child relations in a constituent by parsing from the left (or right) even if there are non-local interactions. In this case, head rules are not useful.

The same Table 1 shows two ablation studies. First, we turn off the composition of constituent nodes into the latent derivations (Eq 1). The ablated models, tagged with *"no n-comp"*, perform from 0.5 to 1 points F1 worse, showing the benefit of adding constituent structure. Then, we check if constituent structure is any useful for dependency parsing metrics. To this end, we emulate a dependency parser using a spinal model by taking standard Stanford dependency trees and adding a dummy constituent for every head with all its children. This model, tagged "SD heads, dummy spines", is slightly outperformed by the "SD heads" model using true spines, even though the margin is small.

Tables 2 and 3 present results on the test, for constituent and dependency parsing respectively. As shown in Table 2 our model is competitive compared to the best parsers; the generative parsers by Choe and Charniak (2016b), Dyer et al. (2016) and Kuncoro et al. (2017) are better than the rest, but compared to the rest our parser is at the same level or better. The most similar system is by Ballesteros and Carreras (2015) and our parser significantly improves the performance. Considering dependency parsing, our model is worse than the ones that train with exploration as Kiperwasser and Goldberg (2016) and Ballesteros et al. (2016), but it slightly improves the parser by Dyer et al. (2015) with static training. The systems that calculate dependencies by transforming phrase-structures with conversion rules and that use generative training are ahead compared to the rest.

7 Conclusions

We have presented a neural model based on Stack-LSTMs for spinal parsing, using a simple extension of arc-standard transition parsing that adds constituent nodes to the dependency derivation. Our experiments suggest that Stack-LSTMs can figure out useful internal structure within constituents, and that the parser might work better *without* providing linguistically-derived head words. Overall, our spinal neural method is sim-

[3] We use the the same POS tags as Dyer et al. (2015).

[4] It is simple to obtain a spinal tree given a constituency tree and a corresponding dependency tree. We assume that the dependency tree is projective and nested within the constituency tree, which holds for the head rules we use.

	LR	LP	F1
Spinal (leftmost)	90.30	90.54	90.42
Spinal (rightmost)	90.23	90.77	90.50
Ballesteros and Carreras (2015)	88.7	89.2	89.0
Vinyals et al. (2014) (PTB-Only)			88.3
Cross and Huang (2016a)			89.9
Choe and Charniak (2016a) (PTB-Only)			91.2
Choe and Charniak (2016a) (Semi-sup)			93.8
Dyer et al. (2016) (Discr.)			91.2
Dyer et al. (2016) (Gen.)			93.3
Kuncoro et al. (2017) (Gen.)			93.5
Liu and Zhang (2017)	91.3	92.1	91.7

Table 2: Constituency results on the PTB test set.

	UAS test
Spinal, PTB spines + SD (TB-greedy)	93.15
Spinal, dummy spines + SD (TB-greedy)	93.10
Dyer et al. (2015) (TB-greedy)	93.1
Cross and Huang (2016a)	93.4
Ballesteros et al. (2016) (TB-dynamic)	93.6
Kiperwasser and Goldberg (2016) (TB-dynamic)	93.9
Andor et al. (2016) (TB-Beam)	94.6
Kuncoro et al. (2016) (Graph-Ensemble)	94.5
Choe and Charniak (2016a)* (Semi-sup)	95.9
Kuncoro et al. (2017)* (Generative)	95.8

Table 3: Stanford Dependency results (UAS) on PTB test set. Parsers marked with * calculate dependencies by transforming phrase-structures with conversion rules.

ple, efficient, and very accurate, and might prove useful to model constituent trees with dependency relations.

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 *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 2442–2452. http://www.aclweb.org/anthology/P16-1231.

Miguel Ballesteros and Xavier Carreras. 2015. Transition-based spinal parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 289–299. https://doi.org/10.18653/v1/K15-1029.

Miguel Ballesteros, Chris Dyer, Yoav Goldberg, and Noah Smith. 2017. Greedy transition-based dependency parsing with stack lstms. *Computational Linguistics* 43(2).

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2005–2010. https://aclweb.org/anthology/D16-1211.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, Coling 2008 Organizing Committee, chapter TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-Rich Parsing, pages 9–16. http://aclweb.org/anthology/W08-2102.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

Do Kook Choe and Eugene Charniak. 2016a. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2331–2336. https://aclweb.org/anthology/D16-1257.

Kook Do Choe and Eugene Charniak. 2016b. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2331–2336. http://aclweb.org/anthology/D16-1257.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Madrid, Spain, pages 16–23. https://doi.org/10.3115/976909.979620.

James Cross and Liang Huang. 2016a. Incremental parsing with minimal features using bi-directional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 32–37. https://doi.org/10.18653/v1/P16-2006.

James Cross and Liang Huang. 2016b. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1–11. http://aclweb.org/anthology/D16-1001.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*. Genoa, volume 6, pages 449–454.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. 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 (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 334–343. http://www.aclweb.org/anthology/P15-1033.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, pages 199–209. http://www.aclweb.org/anthology/N16-1024.

Daniel Fernández-González and T. André F. Martins. 2015. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1523–1533. https://doi.org/10.3115/v1/P15-1147.

Johan Hall and Joakim Nivre. 2008. *Proceedings of the Workshop on Parsing German*, Association for Computational Linguistics, chapter A Dependency-Driven Parser for German Dependency and Constituency Representations, pages 47–54. http://aclweb.org/anthology/W08-1007.

Johan Hall, Joakim Nivre, and Jens Nilsson. 2007. *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA 2007)*, University of Tartu, Estonia, chapter A Hybrid Constituency-Dependency Parser for Swedish, pages 284–287. http://aclweb.org/anthology/W07-2444.

Katsuhiko Hayashi and Masaaki Nagata. 2016. Empty element recovery by spinal parser operations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 95–100. https://doi.org/10.18653/v1/P16-2016.

Katsuhiko Hayashi, Jun Suzuki, and Masaaki Nagata. 2016. Shift-reduce spinal tag parsing with dynamic programming. *Transactions of the Japanese Society for Artificial Intelligence* 31(2).

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, pages 24–31.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*. Barcelona, Spain, pages 95–102. https://doi.org/10.3115/1218955.1218968.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327. https://transacl.org/ojs/index.php/tacl/article/view/885.

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 *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, Valencia, Spain, pages 1249–1258. http://www.aclweb.org/anthology/E17-1117.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one mst parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 1744–1753. https://aclweb.org/anthology/D16-1180.

Jiangming Liu and Yue Zhang. 2017. Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association of Computational Linguistics* 5:45–58. http://aclanthology.coli.uni-saarland.de/pdf/Q/Q17/Q17-1004.pdf.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):313–330.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*. Association for Computational Linguistics, Barcelona, Spain, pages 50–57.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*. Association for Computational Linguistics, Vancouver, British Columbia, pages 125–132.

http://www.aclweb.org/anthology/W/W05/W05-1513.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*. Association for Computational Linguistics, Prague, Czech Republic, pages 144–155. http://www.aclweb.org/anthology/W/W07/W07-2218.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2014. Grammar as a foreign language. *CoRR* abs/1412.7449. http://arxiv.org/abs/1412.7449.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 1169–1179. http://www.aclweb.org/anthology/P15-1113.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*. volume 3, pages 195–206.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Sofia, Bulgaria, pages 434–443. http://www.aclweb.org/anthology/P13-1043.