Left-to-Right Dependency Parsing with Pointer Networks

Daniel Fernández-González
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Elviña, 15071 A Coruña, Spain
d.fgonzalez@udc.es

Carlos Gómez-Rodríguez
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Group
Departamento de Computación
Elviña, 15071 A Coruña, Spain
carlos.gomez@udc.es

Abstract

We propose a novel transition-based algorithm that straightforwardly parses sentences from left to right by building n attachments, with n being the length of the input sentence. Similarly to the recent stack-pointer parser by Ma et al. (2018), we use the pointer network framework that, given a word, can directly point to a position from the sentence. However, our left-to-right approach is simpler than the original top-down stack-pointer parser (not requiring a stack) and reduces transition sequence length in half, from $2n - 1$ actions to n. This results in a quadratic non-projective parser that runs twice as fast as the original while achieving the best accuracy to date on the English PTB dataset (96.04% UAS, 94.43% LAS) among fully-supervised single-model dependency parsers, and improves over the former top-down transition system in the majority of languages tested.

1 Introduction

Dependency parsing, the task of automatically obtaining the grammatical structure of a sentence expressed as a dependency tree, has been widely studied by natural language processing (NLP) researchers in the last decades. Most of the models providing competitive accuracies fall into two broad families of approaches: *graph-based* (McDonald et al., 2005a,b) and *transition-based* (Yamada and Matsumoto, 2003; Nivre, 2003) dependency parsers.

Given an input sentence, a graph-based parser scores trees by decomposing them into factors, and performs a search for the highest-scoring tree.

In the past two years, this kind of dependency parsers have been ahead in terms of accuracy thanks to the graph-based neural architecture developed by Dozat and Manning (2016), which not only achieved state-of-the-art accuracies on the Stanford Dependencies conversion of the English Penn Treebank (hereinafter, PTB-SD), but also obtained the best results in the majority of languages in the CoNLL 2017 Shared Task (Dozat et al., 2017). This tendency recently changed, since a transition-based parser developed by Ma et al. (2018) managed to outperform the best graph-based model in the majority of datasets tested.

Transition-based parsers incrementally build a dependency graph for an input sentence by applying a sequence of transitions. This results in more efficient parsers with linear time complexity for parsing projective sentences, or quadratic for handling non-projective structures, when implemented with greedy or beam search. However, their main weakness is the lack of access to global context information when transitions are greedily chosen. This favours error propagation, mainly affecting long dependencies that require a larger number of transitions to be built (McDonald and Nivre, 2011).

Many attempts have been made to alleviate the impact of error propagation in transition-based dependency parsing, but the latest and most successful approach was developed by Ma et al. (2018). In particular, they make use of *pointer networks* (Vinyals et al., 2015) to implement a new neural network architecture called *stack-pointer network*. The proposed framework provides a global view of the input sentence by capturing information from the whole sentence and all the arcs previously built, crucial for reducing the effect of error propagation; and, thanks to an attention mechanism (Bahdanau et al., 2014; Luong et al., 2015), is able to return a position in that sentence that corresponds to a word related to the word currently on top of the stack. They take advantage of this and propose a novel transition system that follows a top-down depth-first strategy to perform the syntactic analysis. Concretely, it considers the word

Proceedings of NAACL-HLT 2019, pages 710–716
Minneapolis, Minnesota, June 2 - June 7, 2019.

pointed by the neural network as the child of the word on top of the stack, and builds the corresponding dependency relation between them. This results in a transition-based algorithm that can process unrestricted non-projective sentences in $O(n^2)$ time complexity and requires $2n$-1 actions to successfully parse a sentence with n words.

We also take advantage of pointer network capabilities and use the neural network architecture introduced by Ma et al. (2018) to design a non-projective left-to-right transition-based algorithm, where the position value pointed by the network has the opposite meaning: it denotes the index that corresponds to the head node of the current focus word. This results in a straightforward transition system that can parse a sentence in just n actions, without the need of any additional data structure and by just attaching each word from the sentence to another word (including the root node). Apart from increasing the parsing speed twofold (while keeping the same quadratic time complexity), it achieves the best accuracy to date among fully-supervised single-model dependency parsers on the PTB-SD, and obtains competitive accuracies on twelve different languages in comparison to the original top-down version.

2 Preliminaries

Ma et al. (2018) propose a novel neural network architecture whose main backbone is a pointer network (Vinyals et al., 2015). This kind of neural networks are able to learn the conditional probability of a sequence of discrete numbers that correspond to positions in an input sequence (in this case, indexes of words in a sentence) and, by means of attention (Bahdanau et al., 2014; Luong et al., 2015), implement a pointer that selects a position from the input at decoding time.

Their approach initially reads the whole sentence, composed of the n words $w_1, \ldots, w_n$, and encodes each w_i one by one into an *encoder hidden state* e_i. As encoder, they employ a combination of CNNs and bi-directional LSTMs (Chiu and Nichols, 2016; Ma and Hovy, 2016). For each word, CNNs are used to obtain its character-level representation that is concatenated to the word and PoS embeddings to finally be fed into BiLSTMs that encode word context information.

As decoder they present a top-down transition system, where parsing configurations use the classic data structures (Nivre, 2008): a *buffer* (that contains unattached words) and a *stack* (that holds partially processed words).

The available parser actions are two transitions that we call Shift-Attach-p and Reduce. Given a configuration with word w_i on top of the stack, as the pointer network just returns a position p from a given sentence, they proceed as follows to determine which transition should be applied:

- If $p \neq i$, then the pointed word w_p is considered as a child of w_i; so the parser chooses a Shift-Attach-p transition to move w_p from the buffer to the stack and build an arc $w_i \rightarrow w_p$.
- On the other hand, if $p = i$, then w_i is considered to have found all its children, and a Reduce transition is applied to pop the stack.

The parsing process starts with a dummy root \$ on the stack and, by applying $2n$-1 transitions, a dependency tree is built for the input in a top-down depth-first fashion, where multiple children of a same word are forced during training to be created in an inside-out manner. More in detail, for each parsing configuration c_t, the decoder (implemented as a uni-directional LSTM) receives the encoder hidden state e_i of the word w_i on top of the stack to generate a *decoder hidden state* d_t. After that, d_t, together with the sequence s_i of encoder hidden states from words still in the buffer plus e_i, are used to compute the attention vector a^t as follows:

$$v_i^t = score(d_t, s_i) \quad (1)$$

$$a^t = softmax(v^t) \quad (2)$$

As attention scoring function ($score()$), they adopt the biaffine attention mechanism described in (Luong et al., 2015; Dozat and Manning, 2016). Finally, the attention vector a^t will be used to return the highest-scoring position p and choose the next transition. The parsing process ends when only the root remains on the stack.

As extra high-order features, Ma et al. (2018) add grandparent and sibling information, whose encoder hidden states are added to that of the word on top of the stack to generate the corresponding decoder hidden state d_t. They prove that these additions improve final accuracy, especially when children are attached in an inside-out fashion.

According to the authors, the original stack-pointer network is trained to maximize the likelihood of choosing the correct word for each possible top-down path from the root to a leaf. More

in detail, a dependency tree can be represented as a sequence of top-down paths $p_1, \ldots, p_k$, where each path p_i corresponds to a sequence of words $\$, w_{i,1}, w_{i,2}, \ldots, w_{i,l_i}$ from the root to a leaf. Thus, the conditional probability $P_\theta(y|x)$ of the dependency tree y for an input sentence x can be factorized according to this top-down structure as:

$$P_\theta(y|x) = \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, x)$$
$$= \prod_{i=1}^{k} \prod_{j=1}^{l_i} P_\theta(w_{i,j}|w_{i,<j}, p_{<i}, x)$$

where θ represents model parameters, $p_{<i}$ stands for previous paths already explored, $w_{i,j}$ denotes the jth word in path p_i and $w_{i,<j}$ represents all the previous words on p_i.

For more thorough details of the stack-pointer network architecture and the top-down transition system, please read the original work by Ma et al. (2018).

3 Our approach

We take advantage of the neural network architecture designed by Ma et al. (2018) and introduce a simpler left-to-right transition system that requires neither a stack nor a buffer to process the input sentence and where, instead of selecting a child of the word on top of the stack, the network points to the parent of the current focus word.

In particular, in our proposed approach, the parsing configuration just corresponds to a *focus word pointer* i, that is used to point to the word currently being processed. The decoding process starts with i pointing at the first word of the sentence and, at each parsing configuration, only one action is available: the parameterized Attach-p transition, that links the focus word w_i to the head word w_p in position p of the sentence (producing the dependency arc $w_p \rightarrow w_i$) and moves i one position to the right. Note that, in our algorithm, p can equal 0, attaching, in that case, w_i to the dummy root node. The parsing process ends when the last word from the sentence is attached. This can be easily represented as a loop that traverses the input sentence from left to right, linking each word to another from the same sentence or to the dummy root. Therefore, we just need n steps to process the n words of a given sentence and build a dependency tree.

While our novel transition system intrinsically holds the single-head constraint (since, after attaching the word w_i, i points to the next word w_{i+1} in the sentence), it can produce an output with cycles.[1] Therefore, in order to build a well-formed dependency tree during decoding, attachments that generate cycles in the already-built dependency graph must be forbidden. Please note that the need of a cycle-checking extension does not increase the overall quadratic runtime complexity of the original implementation by Ma et al. (2018) since, as in other transition-based parsers such as (Covington, 2001; Gómez-Rodríguez and Nivre, 2010), cycles can be incrementally identified in amortized constant time by keeping track of connected components using path compression and union by rank. Therefore, the left-to-right algorithm requires n steps to produce a parse. In addition, at each step, the attention vector a_t needs to be computed and cycles must be checked, both in $O(n) + O(n) = O(n)$ runtime. This results in a $O(n^2)$ time complexity for decoding.[2]

On the other hand, while in the top-down decoding only available words in the buffer (plus the word on top of the stack) can be pointed to by the network and they are reduced as arcs are created (basically to keep the single-head constraint); our proposed approach is less rigid: all words from the sentence (including the root node and excluding w_i) can be pointed to, as long as they satisfy the acyclicity constraint. This is necessary because two different words might be attached to the same head node and the latter can be located in the sentence either before or after w_i. Therefore, the sequence s_i, required by the attention score function (Eq.(1)), is composed of the encoder hidden states of all words from the input, excluding e_i, and prepending a special vector representation denoting the root node.

We also add extra features to represent the current focus word. Instead of using grandparent and sibling information (more beneficial for a top-down approach), we just add the encoder hidden

[1] In practice, even with the cycle detection mechanism disabled, the presence of cycles in output parses is very uncommon (for instance, just in 1% of sentences in the PTB-SD dev set) since our system seems to adequately model well-formed tree structures.

[2] A practically faster version of the left-to-right parser might be implemented by just ignoring the presence of cycles during decoding, and destroying the cycles generated as a post-processing step that simply removes one of the arcs involved.

states of the previous and next words in the sentence to generate d_t, which seems to be more suitable for a left-to-right decoding.

In dependency parsing, a tree for an input sentence of length n can be represented as a set of n directed and binary links $l_1, \ldots, l_n$. Each link l_i is characterized by the word w_i in position i in the sentence and its head word w_h, resulting in a pair (w_i, w_h). Therefore, to train this novel variant, we factorize the conditional probability $P_\theta(y|x)$ to a set of head-dependent pairs as follows:

$$P_\theta(y|x) = \prod_{i=1}^{n} P_\theta(l_i|l_{<i}, x)$$
$$= \prod_{i=1}^{n} P_\theta(w_h|w_i, l_{<i}, x)$$

Therefore, the left-to-right parser is trained by maximizing the likelihood of choosing the correct head word w_h for the word w_i in position i, given the previous predicted links $l_{<i}$.

Finally, following a widely-used approach (also implemented in (Ma et al., 2018)), dependency labels are predicted by a multiclass classifier, which is trained in parallel with the parser by optimizing the sum of their objectives.

4 Experiments

4.1 Data and Settings

We use the same implementation as Ma et al. (2018) and conduct experiments on the Stanford Dependencies (de Marneffe and Manning, 2008) conversion (using the Stanford parser v3.3.0)[3] of the English Penn Treebank (Marcus et al., 1993), with standard splits and predicted PoS tags. In addition, we compare our approach to the original top-down parser on the same twelve languages from the Universal Dependency Treebanks[4] (UD) that were used by Ma et al. (2018).[5]

Following standard practice, we just exclude punctuation for evaluating on PTB-SD and, for each experiment, we report the average Labelled and Unlabelled Attachment Scores (LAS and UAS) over 3 and 5 repetitions for UD and PTB-SD, respectively.

[3] https://nlp.stanford.edu/software/lex-parser.shtml

[4] http://universaldependencies.org

[5] Please note that, since they used a former version of UD datasets, we reran also the top-down algorithm on the latest treebank version (2.2) in order to perform a fair comparison.

Parser	UAS	LAS
Chen and Manning (2014)	91.8	89.6
Dyer et al. (2015)	93.1	90.9
Weiss et al. (2015)	93.99	92.05
Ballesteros et al. (2016)	93.56	91.42
Kiperwasser and Goldberg (2016)	93.9	91.9
Alberti et al. (2015)	94.23	92.36
Qi and Manning (2017)	94.3	92.2
Fernández-G and Gómez-R (2018)	94.5	92.4
Andor et al. (2016)	94.61	92.79
Ma et al. (2018)*	95.87	94.19
This work*	**96.04**	**94.43**
Kiperwasser and Goldberg (2016)	93.1	91.0
Wang and Chang (2016)	94.08	91.82
Cheng et al. (2016)	94.10	91.49
Kuncoro et al. (2016)	94.26	92.06
Zhang et al. (2017)	94.30	91.95
Ma and Hovy (2017)	94.88	92.96
Dozat and Manning (2016)	95.74	94.08
Ma et al. (2018)*	95.84	94.21

Table 1: Accuracy comparison of state-of-the-art fully-supervised single-model dependency parsers on PT-SD. The first block contains transition-based algorithms and the second one, graph-based models. Systems marked with *, including the improved variant described in (Ma et al., 2018) of the graph-based parser by (Dozat and Manning, 2016), are implemented under the same framework as our approach and use the same training settings. Like (Ma et al., 2018), we report the average accuracy over 5 repetitions.

Finally, we use the same hyper-parameter values, pre-trained word embeddings and beam size (10 for PTB-SD and 5 for UD) as Ma et al. (2018).

4.2 Results

By outperforming the two current state-of-the-art graph-based (Dozat and Manning, 2016) and transition-based (Ma et al., 2018) models on the PTB-SD, our approach becomes the most accurate fully-supervised dependency parser developed so far, as shown in Table 1.[6]

In addition, in Table 2 we can see how, under the exactly same conditions, the left-to-right algorithm improves over the original top-down variant in nine out of twelve languages in terms of LAS, obtaining competitive results in the remaining three datasets.

Finally, in spite of requiring a cycle-checking procedure, our approach proves to be twice as fast as the top-down alternative in decoding time,

[6] It is worth mentioning that all parsers reported in this section make use of pre-trained word embeddings previously learnt from corpora beyond the training dataset. However, it is common practice in the literature that systems that only use standard pre-trained word embeddings are classed as fully-supervised models, even though, strictly, they are not trained exclusively on the official training data.

	Top-down		Left-to-right	
	UAS	LAS	UAS	LAS
bu	**94.42±0.02**	**90.70±0.04**	94.28±0.06	90.66±0.11
ca	93.83±0.02	91.96±0.01	**94.07±0.06**	**92.26±0.05**
cs	93.97±0.02	91.23±0.03	**94.19±0.04**	**91.45±0.05**
de	**87.28±0.07**	**82.99±0.07**	87.06±0.05	82.63±0.01
en	90.86±0.15	88.92±0.19	**90.93±0.11**	**88.99±0.11**
es	93.09±0.05	91.11±0.03	**93.23±0.03**	**91.28±0.02**
fr	**90.97±0.09**	**88.22±0.12**	90.90±0.04	88.14±0.10
it	94.08±0.04	92.24±0.06	**94.28±0.06**	**92.48±0.02**
nl	**93.23±0.09**	90.67±0.07	93.13±0.07	**90.74±0.08**
no	95.02±0.05	93.75±0.05	**95.23±0.06**	**93.99±0.07**
ro	91.44±0.11	85.80±0.14	**91.58±0.08**	**86.00±0.07**
ru	94.43±0.01	93.08±0.03	**94.71±0.07**	**93.38±0.09**

Table 2: Parsing accuracy of the top-down and left-to-right pointer-network-based parsers on test datasets of twelve languages from UD. Best results for each language are shown in bold and, apart from the average UAS and LAS, we also report the corresponding standard deviation over 3 runs.

achieving, under the exact same conditions, a 23.08-sentences-per-second speed on the PTB-SD compared to 10.24 of the original system.[7]

5 Related work

There is previous work that proposes to implement dependency parsing by independently selecting the head of each word in a sentence, using neural networks. In particular, Zhang et al. (2017) make use of a BiLSTM-based neural architecture to compute the probability of attaching each word to one of the other input words, in a similar way as pointer networks do. During decoding, a post-processing step is needed to produce well-formed trees by means of a maximum spanning tree algorithm. Our approach does not need this post-processing, as cycles are forbidden during parsing instead, and achieves a higher accuracy thanks to the pointer network architecture and the use of information about previous dependencies.

Before Ma et al. (2018) presented their top-down parser, Chorowski et al. (2017) had already employed pointer networks (Vinyals et al., 2015) for dependency parsing. Concretely, they developed a pointer-network-based neural architecture with multitask learning able to perform pre-processing, tagging and dependency parsing exclusively by reading tokens from an input sentence, without needing POS tags or pre-trained word embeddings. Like our approach, they also use the capabilities provided by pointer networks to undertake the parsing task as a simple process of attaching each word as dependent of another. They also try to improve the network performance with POS tag prediction as auxiliary task and with different approaches to perform label prediction. They do not exclude cycles, neither by forbidding them at parsing time or by removing them by post-processing, as they report that their system produces parses with a negligible amount of cycles, even with greedy decoding (matching our observation for our own system, in our case with beam-search decoding). Finally, the system developed by Chorowski et al. (2017) is constrained to projective dependencies, while our approach can handle unrestricted non-projective structures.

[7] Please note that the implementation by Ma et al. (2018), also used by our novel approach, was not optimized for speed and, therefore, the reported speeds are just intended for comparing algorithms implemented under the same framework, but not to be considered as the best speed that a pointer-network-based system can potentially achieve.

6 Conclusion

We present a novel left-to-right dependency parser based on pointer networks. We follow the same neural network architecture as the stack-pointer-based approach developed by Ma et al. (2018), but just using a focus word index instead of a buffer and a stack. Apart from doubling their system's speed, our approach proves to be a competitive alternative on a variety of languages and achieves the best accuracy to date on the PTB-SD.

The good performance of our algorithm can be explained by the shortening of the transition sequence length. In fact, it has been proved by several studies (Fernández-González and Gómez-Rodríguez, 2012; Qi and Manning, 2017; Fernández-González and Gómez-Rodríguez, 2018) that by reducing the number of applied transitions, the impact of error propagation is alleviated, yielding more accurate parsers.

Our system's source code is freely available at `https://github.com/danifg/Left2Right-Pointer-Parser`.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from MINECO (FFI2014-51978-C2-2-R, TIN2017-85160-C2-1-R) and from Xunta de Galicia (ED431B 2017/01).

References

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1354–1359.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

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, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2005–2010.

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)*, pages 740–750, Doha, Qatar.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214. Association for Computational Linguistics.

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Jan Chorowski, Michal Zapotoczny, and Pawel Rychlikowski. 2017. Read, tag, and parse all at once, or fully-neural dependency parsing. *CoRR*, abs/1609.03441.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, New York, NY, USA. ACM.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, August 3-4, 2017*, pages 20–30.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 334–343.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2012. Improving transition-based dependency parsing with buffer transitions. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 308–319. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. Non-projective dependency parsing with non-local transitions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 693–700. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1492–1501.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

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*, pages 1744–1753. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 59–69. Asian Federation of Natural Language Processing.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard H. Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1403–1414.

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

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser '08, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Comput. Linguist.*, 37:197–230.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 149–160. ACL/SIGPARSE.

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

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 110–117.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315. Association for Computational Linguistics.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 323–333.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 665–676.