Linear-Time Constituency Parsing with RNNs and Dynamic Programming

Juneki Hong [1]
[1]School of EECS
Oregon State University, Corvallis, OR

Liang Huang [1,2]
[2]Silicon Valley AI Lab
Baidu Research, Sunnyvale, CA

{juneki.hong, liang.huang.sh}@gmail.com

Abstract

Recently, span-based constituency parsing has achieved competitive accuracies with extremely simple models by using bidirectional RNNs to model "spans". However, the minimal span parser of Stern et al. (2017a) which holds the current state of the art accuracy is a chart parser running in cubic time, $O(n^3)$, which is too slow for longer sentences and for applications beyond sentence boundaries such as end-to-end discourse parsing and joint sentence boundary detection and parsing. We propose a linear-time constituency parser with RNNs and dynamic programming using graph-structured stack and beam search, which runs in time $O(nb^2)$ where b is the beam size. We further speed this up to $O(nb \log b)$ by integrating cube pruning. Compared with chart parsing baselines, this linear-time parser is substantially faster for long sentences on the Penn Treebank and orders of magnitude faster for discourse parsing, and achieves the highest F1 accuracy on the Penn Treebank among single model end-to-end systems.

1 Introduction

Span-based neural constituency parsing (Cross and Huang, 2016; Stern et al., 2017a) has attracted attention due to its high accuracy and extreme simplicity. Compared with other recent neural constituency parsers (Dyer et al., 2016; Liu and Zhang, 2016; Durrett and Klein, 2015) which use neural networks to model tree structures, the span-based framework is considerably simpler, only using bidirectional RNNs to model the *input sequence* and not the *output tree*. Because of this factorization, the output space is decomposable which enables efficient dynamic programming algorithm such as CKY. But existing span-based parsers suffer from a crucial limitation in terms of search: on the one hand, a greedy span parser (Cross and Huang, 2016) is fast (linear-time) but only explores one single path in the exponentially large search space, and on the other hand, a chart-based span parser (Stern et al., 2017a) performs exact search and achieves state-of-the-art accuracy, but in cubic time, which is too slow for longer sentences and for applications that go beyond sentence boundaries such as end-to-end discourse parsing (Hernault et al., 2010; Zhao and Huang, 2017) and integrated sentence boundary detection and parsing (Björkelund et al., 2016).

We propose to combine the merits of both greedy and chart-based approaches and design a linear-time span-based neural parser that searches over exponentially large space. Following Huang and Sagae (2010), we perform left-to-right dynamic programming in an action-synchronous style, with $(2n-1)$ actions (i.e., steps) for a sentence of n words. While previous non-neural work in this area requires sophisticated features (Huang and Sagae, 2010; Mi and Huang, 2015) and thus high time complexity such as $O(n^{11})$, our states are as simple as $\ell : (i, j)$ where ℓ is the step index and (i, j) is the span, modeled using bidirectional RNNs without any syntactic features. This gives a running time of $O(n^4)$, with the extra $O(n)$ for step index. We further employ beam search to have a practical runtime of $O(nb^2)$ at the cost of exact search where b is the beam size. However, on the Penn Treebank, most sentences are less than 40 words ($n < 40$), and even with a small beam size of $b = 10$, the *observed* complexity of an $O(nb^2)$ parser is *not* exactly linear in n (see Experiments). To solve this problem, we apply cube pruning (Chiang, 2007; Huang and Chiang, 2007) to improve the runtime to $O(nb \log b)$ which

renders an observed complexity that is linear in n (with minor extra inexactness).

We make the following contributions:

- We design the first neural parser that is both linear time and capable of searching over exponentially large space.[1]
- We are the first to apply cube pruning to incremental parsing, and achieves, for the first time, the complexity of $O(nb \log b)$, i.e., linear in sentence length and (almost) linear in beam size. This leads to an observed complexity strictly linear in sentence length n.
- We devise a novel loss function which penalizes wrong spans that cross gold-tree spans, and employ max-violation update (Huang et al., 2012) to train this parser with structured SVM and beam search.
- Compared with chart parsing baselines, our parser is substantially faster for long sentences on the Penn Treebank, and orders of magnitude faster for end-to-end discourse parsing. It also achieves the highest F1 score on the Penn Treebank among single model end-to-end systems.
- We devise a new formulation of graph-structured stack (Tomita, 1991) which requires no extra bookkeeping, proving a new theorem that gives deep insight into GSS.

2 Preliminaries

2.1 Span-Based Shift-Reduce Parsing

A span-based shift-reduce constituency parser (Cross and Huang, 2016) maintains a stack of spans (i, j), and progressively adds a new span each time it takes a shift or reduce action. With (i, j) on top of the stack, the parser can either *shift* to push the next singleton span $(j, j+1)$ on the stack, or it can *reduce* to combine the top two spans, (k, i) and (i, j), forming the larger span (k, j). After each shift/reduce action, the top-most span is labeled as either a constituent or with a *null* label $\varnothing$, which means that the subsequence is not a subtree in the final decoded parse. Parsing initializes with an empty stack and continues until $(0, n)$ is formed, representing the entire sentence.

[1] https://github.com/junekihong/beam-span-parser

input	$w_0 \dots w_{n-1}$
state	$\ell : \langle i, j \rangle : (c, v)$
init	$0 : \langle 0, 0 \rangle : (0, 0)$
goal	$2n-1 : \langle 0, n \rangle : (c, c)$

$$\textbf{shift} \quad \frac{\ell : \langle \_, j \rangle : (c, \_)}{\ell+1 : \langle j, j+1 \rangle : (c+\xi, \xi)} \; j < n$$

$$\textbf{reduce} \quad \frac{\ell' : \langle k, i \rangle : (c', v') \quad \ell : \langle i, j \rangle : (\_, v)}{\ell+1 : \langle k, j \rangle : (c'+v+\sigma, v'+v+\sigma)}$$

Figure 1: Our shift-reduce deductive system. Here ℓ is the step index, c and v are prefix and inside scores. Unlike Huang and Sagae (2010) and Cross and Huang (2016), ξ and σ are not shift/reduce scores; instead, they are the (best) label scores of the resulting span: $\xi = \max_X s(j, j+1, X)$ and $\sigma = \max_X s(k, j, X)$ where X is a nonterminal symbol (could be $\varnothing$). Here $\ell' = \ell - 2(j-i) + 1$.

2.2 Bi-LSTM features

To get the feature representation of a span (i, j), we use the output sequence of a bi-directional LSTM (Cross and Huang, 2016; Stern et al., 2017a). The LSTM produces $\mathbf{f}_0, ..., \mathbf{f}_n$ forwards and $\mathbf{b}_n, ..., \mathbf{b}_0$ backwards outputs, which we concatenate the differences of $(\mathbf{f}_j - \mathbf{f}_i)$ and $(\mathbf{b}_i - \mathbf{b}_j)$ as the representation for span (i, j). This eliminates the need for complex feature engineering, and can be stored for efficient querying during decoding.

3 Dynamic Programming

3.1 Score Decomposition

Like Stern et al. (2017a), we also decompose the score of a tree t to be the sum of the span scores:

$$s(t) = \sum_{(i,j,X) \in t} s(i, j, X) \quad (1)$$

$$= \sum_{(i,j) \in t} \max_X s((\mathbf{f}_j - \mathbf{f}_i; \mathbf{b}_i - \mathbf{b}_j), X) \quad (2)$$

Note that X is a nonterminal label, a unary chain (e.g., S-VP), or null label $\varnothing$.[2] In a shift-reduce setting, there are $2n-1$ steps (n shifts and $n-1$ reduces) and after each step we take the best label for the resulting span; therefore there are exactly

[2] The actual code base of Stern et al. (2017b) forces $s(i, j, \varnothing)$ to be 0, which simplifies their CKY parser and slightly improves their parsing accuracy. However, in our incremental parser, this change favors shift over reduce and degrades accuracy, so our parser keeps a learned score for $\varnothing$.

$2n-1$ such (labeled) spans (i, j, X) in tree t. Also note that the choice of the label for any span (i, j) is only dependent on (i, j) itself (and not depending on any subtree information), thus the max over label X is independent of other spans, which is a nice property of span-based parsing (Cross and Huang, 2016; Stern et al., 2017a).

3.2 Graph-Struct. Stack w/o Bookkeeping

We now reformulate this DP parser in the above section as a shift-reduce parser. We maintain a step index ℓ in order to perform action-synchronous beam search (see below). Figure 1 shows how to represent a parsing stack using only the top span (i, j). If the top span (i, j) shifts, it produces $(j, j+1)$, but if it reduces, it needs to know the second last span on the stack, (k, i), which is *not* represented in the current state. This problem can be solved by graph-structure stack (Tomita, 1991; Huang and Sagae, 2010), which maintains, for each state p, a set of predecessor states $\pi(p)$ that p can combine with on the left.

This is the way our actual code works ($\pi(p)$ is implemented as a list of pointers, or "left pointers"), but here for simplicity of presentation we devise a novel but easier-to-understand formulation in Fig. 1, where we explicitly represent the set of predecessor states that state $\ell : (i, j)$ can combine with as $\ell' : (k, i)$ where $\ell' = \ell - 2(j-i) + 1$, i.e., (i, j) at step ℓ can combine with any (k, i) for any k at step ℓ'. The rationale behind this new formulation is the following theorem:

Theorem 1 *The predecessor states $\pi(\ell : (i, j))$ are all in the same step $\ell' = \ell - 2(j-i) + 1$.*

Proof. By induction.

This Theorem bring new and deep insights and suggests an alternative implementation that does not require any extra bookkeeping. The time complexity of this algorithm is $O(n^4)$ with the extra $O(n)$ due to step index.[3]

3.3 Action-Synchronous Beam Search

The incremental nature of our parser allows us to further lower the runtime complexity at the cost of inexact search. At each time step, we maintain the top b parsing states, pruning off the rest. Thus, a candidate parse that made it to the end of decoding had to survive within the top b at every step. With $O(n)$ parsing actions our time complexity becomes linear in the length of the sentence.

3.4 Cube Pruning

However, Theorem 1 suggests that a parsing state p can have up to b predecessor states ("left pointers"), i.e., $|\pi(p)| \leq b$ because $\pi(p)$ are all in the same step, a reduce action can produce up to b subsequent new reduced states. With b items on a beam and $O(n)$ actions to take, this gives us an overall complexity of $O(nb^2)$. Even though b^2 is a constant, even modest values of b can make b^2 dominate the length of the sentence. [4]

To improve this at the cost of additional inexactness, we introduce cube pruning to our beam search, where we put candidate actions into a heap and retrieve the top b states to be considered in the next time-step. We heapify the top b shift-merged states and the top b reduced states. To avoid inserting all b^2 reduced states from the previous beam, we only consider each state's highest scoring left pointer,[5] and whenever we pop a reduced state from the heap, we iterate down its left pointers to insert the next non-duplicate reduced state back into the heap. This process finishes when we pop b items from the heap. The initialization of the heap takes $O(b)$ and popping b items takes $O(b \log b)$, giving us an overall improved runtime of $O(nb \log b)$.

4 Training

We use a Structured SVM approach for training (Stern et al., 2017a; Shi et al., 2017). We want the model to score the gold tree t^* higher than any other tree t by at least a margin $\Delta(t, t^*)$:

$$\forall t, s(t^*) - s(t) \geq \Delta(t, t^*).$$

Note that $\Delta(t, t) = 0$ for any t and $\Delta(t, t^*) > 0$ for any $t \neq t^*$. At training time we perform loss-augmented decoding:

$$\hat{t} = \arg\max_t s_\Delta(t) = \arg\max_t s(t) + \Delta(t, t^*).$$

[3] The word-synchronous alternative does not need the step index ℓ and enjoys a cubic time complexity, being almost identical to CKY. However, beam search becomes very tricky.

[4] The average length of a sentence in the Penn Treebank training set is about 24. Even with a beam size of 10, we already have $b^2 = 100$, which would be a significant factor in our runtime. In practice, each parsing state will rarely have the maximum b left pointers so this ends up being a loose upper-bound. Nevertheless, the beam search should be performed with the input length in mind, or else as b increases we risk losing a linear runtime.

[5] If each previous beam is sorted, and if the beam search is conducted by going top-to-bottom, then each state's left pointers will implicitly be kept in sorted order.

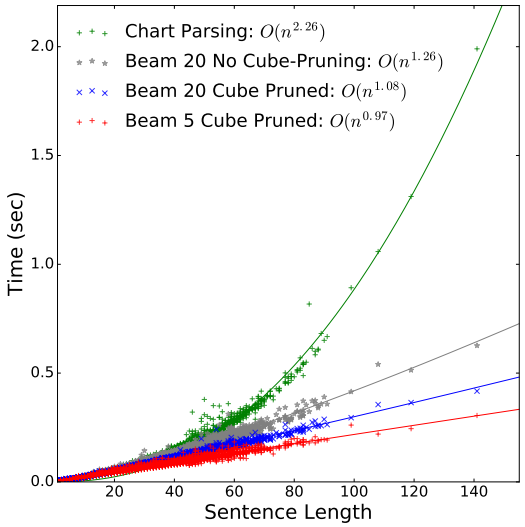

Figure 2: Runtime plots of decoding on the training set of the Penn Treebank. The differences between the different algorithms become evident after sentences of length 40. The regression curves have been empirically fitted.

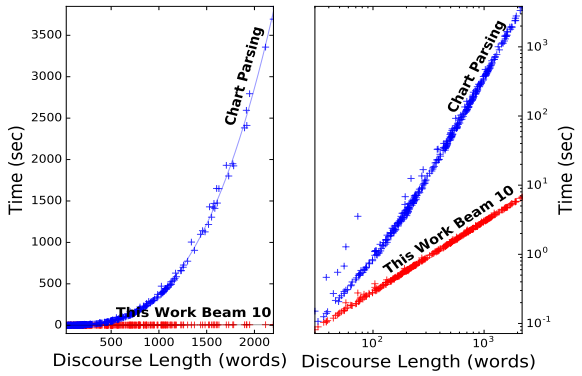

Figure 3: Runtime plot of decoding the discourse treebank training set. The log-log plot on the right shows the cubic complexity of baseline chart parsing. Whereas beam search decoding maintains linear time even for sequences of thousands of words.

where $s_\Delta(\cdot)$ is the loss-augmented score. If $\hat{t} = t^*$, then all constraints are satisfied (which implies $\arg\max_t s(t) = t^*$), otherwise we perform an update by backpropagating from $s_\Delta(\hat{t}) - s(t^*)$.

4.1 Cross-Span Loss

The baseline loss function from Stern et al. (2017a) counts the incorrect labels (i, j, X) in the predicted tree:

$$\Delta_{base}(t, t^*) = \sum_{(i,j,X)\in t} \mathbb{1}\Big(X \neq t^*_{(i,j)}\Big).$$

Note that X can be null $\varnothing$, and $t^*_{(i,j)}$ denotes the gold label for span (i, j), which could also be $\varnothing$.[6] However, there are two cases where $t^*_{(i,j)} = \varnothing$: a subspan (i, j) due to binarization (e.g., a span combining the first two subtrees in a ternary branching node), or an invalid span in t that crosses a gold span in t^*. In the baseline function above, these two cases are treated equivalently; for example, a span $(3, 5, \varnothing) \in t$ is not penalized even if there is a gold span $(4, 6, \text{VP}) \in t^*$. So we revise our loss function as:

$$\Delta_{new}(t, t^*) = \sum_{(i,j,X)\in t} \mathbb{1}\Big(X \neq t^*_{(i,j)} \vee \text{cross}(i, j, t^*)\Big)$$

[6]Note that the predicted tree t has exactly $2n - 1$ spans but t^* has much fewer spans (only labeled spans without $\varnothing$).

where $\text{cross}(i, j, t^*) = \exists\,(k, l) \in t^*$, and $i < k < j < l$ or $k < i < l < j$.

4.2 Max Violation Updates

Given that we maintain loss-augmented scores even for partial trees, we can perform a training update on a given example sentence by choosing to take the loss where it is the greatest along the parse trajectory. At each parsing time-step ℓ, the *violation* is the difference between the highest augmented-scoring parse trajectory up to that point and the gold trajectory (Huang et al., 2012; Yu et al., 2013). Note that computing the violation gives us the max-margin loss described above. Taking the largest violation from all time-steps gives us the max-violation loss.

5 Experiments

We present experiments on the Penn Treebank (Marcus et al., 1993) and the PTB-RST discourse treebank (Zhao and Huang, 2017). In both cases, the training set is shuffled before each epoch, and dropout (Hinton et al., 2012) is employed with probability 0.4 to the recurrent outputs for regularization. Updates with minibatches of size 10 and 1 are used for PTB and the PTB-RST respectively. We use Adam (Kingma and Ba, 2014) with default settings to schedule learning rates for all the weights. To address unknown words during training, we adopt the strategy described by Kiperwasser and Goldberg (Kiperwasser and Goldberg, 2016); words in the training set are replaced with the unknown word symbol `UNK` with probability $p_{unk} = \frac{1}{1+f(w)}$, with $f(w)$ being the number of

	Development Set 22 (F1)		
	Baseline	+cross-span	Time
This Work Beam 1	89.41	89.93	0.042
This Work Beam 5	91.27	91.91	0.050
This Work Beam 10	91.56	92.09	0.058
This Work Beam 15	91.74	92.16	0.062
This Work Beam 20	91.65	92.20	0.066
Chart	92.02	92.21	0.076

Table 1: Comparison of PTB development set results, with the time measured in seconds-per-sentence. The baseline chart parser is from Stern et al. (2017b), with null-label scores unconstrained to be nonzero, replicating their paper.

	Test Set 23		
End-to-End & Single Model	LR	LP	F1
Socher et al. (2013)			90.4
Durrett and Klein (2015)			91.1
Cross and Huang (2016)	90.5	92.1	91.3
Liu and Zhang (2016)	91.3	92.1	91.7
Dyer et al. (2016) (discrim.)			91.7
Stern et al. (2017a)	90.63	92.98	91.79
Stern et al. (2017a) +cross-span	91.67	91.94	91.81
Stern et al. (2017b)	91.35	92.38	91.86
This Work Beam 10	91.44	91.91	91.67
This Work Beam 15	91.64	92.04	91.84
This Work Beam 20	91.49	92.45	**91.97**
Reranking/Ensemble/Separate Decoding			
Vinyals et al. (2015) (ensem)			90.5
Dyer et al. (2016) (gen., rerank)			93.3
Choe and Charniak (2016) (rerank)			93.8
Stern et al. (2017c) (sep. decoding)	92.57	92.56	92.56
Fried et al. (2017) (ensem/rerank)			94.25

Table 2: Final PTB Test Results. We compare our models with other (neural) single-model end-to-end trained systems.

occurrences of word w in the training corpus. Our system is implemented in Python using the DyNet neural network library (Neubig et al., 2017).

5.1 Penn Treebank

We use the Wall Street Journal portion of the Penn Treebank, with the standard split of sections 2-21 for training, 22 for development, and 23 for testing. Tags are provided using the Stanford tagger with 10-way jackknifing.

Table 1 shows our development results and overall speeds, while Table 2 compares our test results. We show that a beam size of 20 can be fast while still achieving state-of-the-art performances.

5.2 Discourse Parsing

To measure the tractability of parsing on longer sequences, we also consider experiments on the

	LR	LP	F1
Zhao and Huang (2017)	81.6	83.5	82.5
This Work Beam 10	80.47	80.61	80.54
This Work Beam 20	80.86	80.73	80.79
This Work Beam 200	81.51	80.84	81.18
This Work Beam 500⋆	81.50	80.81	81.16
This Work Beam 1000⋆	81.55	80.85	81.20

Table 3: Overall test accuracies for PTB-RST discourse treebank. Starred⋆ rows indicate a run that was decoded from the beam 200 model.

	segment	structure	+nuclearity	+relation
Bach et al. (2012)	95.1	-	-	-
Hernault et al. (2010)	94.0	72.3	59.1	47.3
Zhao and Huang (2017)	95.4	78.8	65.0	52.2
This Work Beam 200	91.20	73.36	58.87	46.38
This Work Beam 500⋆	93.52	74.93	60.16	47.03
This Work Beam 1000⋆	94.06	75.60	60.61	47.37

Table 4: F1 scores comparing discourse systems. Results correspond to the accuracies in Table 3, broken down to focus on the discourse labels.

PTB-RST discourse Treebank, a joint discourse and constituency dataset with a combined representation, allowing for parsing at either level (Zhao and Huang, 2017). We compare our runtimes out-of-the-box in Figure 3. Without any pre-processing, and by treating discourse examples as constituency trees with thousands of words, our trained models represent end-to-end discourse parsing systems.

For our overall constituency results in Table 3, and for discourse results in Table 4, we adapt the split-point feature described in (Zhao and Huang, 2017) in addition to the base parser. We find that larger beamsizes are required to achieve good discourse scores.

6 Conclusions

We have developed a new neural parser that maintains linear time, while still searching over an exponentially large space. We also use cube pruning to further improve the runtime to $O(nb \log b)$. For training, we introduce a new loss function, and achieve state-of-the-art results among single-model end-to-end systems.

Acknowledgments

We thank Dezhong Deng who contributed greatly to Secs. 3.2 and 4 (he deserves co-authorship), and Mitchell Stern for releasing his code and and suggestions. This work was supported in part by NSF IIS-1656051 and DARPA N66001-17-2-4030.

References

Ngo Xuan Bach, Nguyen Le Minh, and Akira Shimazu. 2012. A reranking model for discourse segmentation using subtree features. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, pages 160–168.

Anders Björkelund, Agnieszka Faleńska, Wolfgang Seeker, and Jonas Kuhn. 2016. How to train dependency parsers with inexact search for joint sentence boundary detection and parsing of entire documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 1924–1934.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics* 33(2):201–208.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. pages 2331–2336.

James Cross and Liang Huang. 2016. 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, Austin, Texas, pages 1–11. https://aclweb.org/anthology/D16-1001.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. *arXiv preprint arXiv:1507.03641* .

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776* .

Daniel Fried, Mitchell Stern, and Dan Klein. 2017. Improving neural parsing by disentangling model combination and reranking effects. In *Proceedings of the Association for Computational Linguistics*.

Hugo Hernault, Helmut Prendinger, David A DuVerle, Mitsuru Ishizuka, and Tim Paek. 2010. Hilda: a discourse parser using support vector machine classification. *Dialogue and Discourse* 1(3):1–33.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* .

Liang Huang and David Chiang. 2007. Forest rescoring: Fast decoding with integrated language models. In *Proceedings of ACL 2007*. Prague, Czech Rep.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of NAACL*. http://www.isi.edu/ lhuang/perc-inexact.pdf.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL 2010*. Uppsala, Sweden.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR* abs/1603.04351. http://arxiv.org/abs/1603.04351.

Jiangming Liu and Yue Zhang. 2016. Shift-reduce constituent parsing with neural lookahead features. *arXiv preprint arXiv:1612.00567* .

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Haitao Mi and Liang Huang. 2015. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of NAACL 2015*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of EMNLP 2017 (to appear)*.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of the Association for Computational Linguistics*. Association for Computational Linguistics, volume 1, pages 455–465.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017a. A minimal span-based neural constituency parser. In *Proceedings of the Association for Computational Linguistics*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017b. A minimal span-based neural constituency parser (code base). `https://github.com/mitchellstern/minimal-span-parser`.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017c. Effective inference for generative neural parsing. In *Proceedings of Empirical Methods in Natural Language Processing*. pages 1695–1700.

Masaru Tomita, editor. 1991. *Generalized LR Parsing*. Kluwer Academic Publishers.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*. pages 2773–2781.

Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-violation perceptron and forced decoding for scalable mt training. In *Proceedings of EMNLP 2013*.

Kai Zhao and Liang Huang. 2017. Joint syntacto-discourse parsing and the syntacto-discourse treebank. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. pages 2117–2123.