# Generative Incremental Dependency Parsing with Neural Networks

**Jan Buys**[1] and **Phil Blunsom**[1,2]
[1]Department of Computer Science, University of Oxford [2]Google DeepMind
{jan.buys,phil.blunsom}@cs.ox.ac.uk

## Abstract

We propose a neural network model for scalable generative transition-based dependency parsing. A probability distribution over both sentences and transition sequences is parameterised by a feed-forward neural network. The model surpasses the accuracy and speed of previous generative dependency parsers, reaching 91.1% UAS. Perplexity results show a strong improvement over $n$-gram language models, opening the way to the efficient integration of syntax into neural models for language generation.

## 1 Introduction

Transition-based dependency parsers that perform incremental local inference with a discriminative classifier offer an appealing trade-off between speed and accuracy (Nivre, 2008; Zhang and Nivre, 2011; Choi and Mccallum, 2013). Recently neural network transition-based dependency parsers have been shown to give state-of-the-art performance (Chen and Manning, 2014; Dyer et al., 2015; Weiss et al., 2015). However, the downstream integration of syntactic structure in language understanding and generation tasks is often done heuristically.

Neural networks have also been shown to be powerful generative models for language modelling (Bengio et al., 2003; Mikolov et al., 2010) and machine translation (Kalchbrenner and Blunsom, 2013; Devlin et al., 2014; Sutskever et al., 2014). However, currently these models lack awareness of syntax, which limits their ability to include longer-distance dependencies even when potentially unbounded contexts are used.

In this paper we propose a generative model for incremental parsing that offers an efficient way to incorporate syntactic information into a generative model. It relies on the strength of neural networks to overcome sparsity in the long conditioning contexts required for an accurate model, while also offering a principled approach to learn dependency-based word representations (Levy and Goldberg, 2014; Bansal et al., 2014).

Generative models for graph-based dependency parsing (Eisner, 1996; Wallach et al., 2008) are much less accurate than their discriminative counterparts. Syntactic language models based on PCFGs (Roark, 2001; Charniak, 2001) and incremental parsing (Chelba and Jelinek, 2000; Emami and Jelinek, 2005) have been proposed for speech recognition and machine translation. However, these models are also limited in either scalability, expressiveness, or both. A generative transition-based dependency parser based on recurrent neural networks (Titov and Henderson, 2007) obtains high accuracy, but training and decoding is prohibitively expensive.

We perform efficient linear-time decoding with a particle filtering-based beam-search method where derivations after pruned after every word generation and the beam size depends on the uncertainty in the model (Buys and Blunsom, 2015).

The model obtains 91.1% UAS on the WSJ, which is 0.2% UAS better than the previous highest accuracy generative dependency parser (Titov and Henderson, 2007), while also being much more efficient. As a language model its perplexity reaches 111.8, a 23% reduction over an $n$-gram baseline, when combining supervised training with unsupervised fine-tuning. Finally, we find that the model is able to generate sentences that display both local and syntactic coherence.

## 2 Generative Transition-based Parsing

Our parsing model is based on transition-based arc-standard projective dependency parsing (Nivre and Scholz, 2004). The generative formulation is similar to previous generative transition-based parsers (Titov and Henderson, 2007; Cohen et al., 2011; Buys and Blunsom, 2015), and also related to the joint tagging and parsing model of Bohnet and Nivre (2012).

The model predicts a sequence of parsing transitions: A shift transition generates a word (and its POS tag), while a reduce transition adds an arc $(i, l, j)$, where $i$ is the head node, $j$ the dependent and $l$ is the dependency label.

The joint probability distribution over a sentence with words $\boldsymbol{w}_{1:n}$, tags $\boldsymbol{t}_{1:n}$ and a transition sequence $\boldsymbol{a}_{1:2n}$ is defined as

$$\prod_{i=1}^{n} \Big( p(t_i|\boldsymbol{h}_{m_i}) p(w_i|t_i, \boldsymbol{h}_{m_i}) \prod_{j=m_i+1}^{m_{i+1}} p(a_j|\boldsymbol{h}_j) \Big),$$

where $m_i$ is the number of transitions that have been performed when $(t_i, w_i)$ is shifted and $\boldsymbol{h}_j$ is the conditioning context at the $j$th transition.

A parser configuration $(\sigma, \beta, A)$ for sentence $\boldsymbol{s}$ consists of a stack $\sigma$ of indices in $\boldsymbol{s}$, an index $\beta$ to the next word to be generated, and a set of arcs $A$. The stack elements are referred to as $\sigma_1, \ldots, \sigma_{|\sigma|}$, where $\sigma_1$ is the top element. For any node $a$, $lc_1(a)$ refers to the leftmost child of $a$ in $A$, and $rc_1(a)$ to its rightmost child. A root node is added to the beginning of the sentence, and the head word of the sentence (we assume there is only one) is the dependent of the root.

The initial configuration is $([], 0, \emptyset)$, while A terminal configuration is reached when $\beta > |\boldsymbol{s}|$ and $|\sigma| = 1$.

The transition types are shift, left-arc and right-arc. *Shift* generates the next word of the sentence and pushes it on the stack. *Left-arc* adds an arc $(\sigma_1, l, \sigma_2)$ and removes $\sigma_2$ from the stack. *Right-arc* adds $(\sigma_2, l, \sigma_1)$ and pops $\sigma_1$.

The parsing strategy adds arcs bottom-up. In a valid transition sequence the last transition is a right-arc from the root to the head word, and the root node is not involved in any other dependencies. We use an oracle to extract transition sequences from the training data: The oracle prefers reduce over shift transitions when both may lead to a valid derivation.

| Order | Elements |
|---|---|
| 1 | $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ |
| 2 | $lc_1(\sigma_1), rc_1(\sigma_1), lc_1(\sigma_2), rc_1(\sigma_2)$<br>$lc_2(\sigma_1), rc_2(\sigma_1), lc_2(\sigma_2), rc_2(\sigma_2)$ |
| 3 | $lc_1(lc_1(\sigma_1)), rc_1(rc_1(\sigma_1))$<br>$lc_1(lc_1(\sigma_2)), rc_1(rc_1(\sigma_2))$ |

Table 1: Conditioning context elements for neural network input: First, second and third order dependencies are used.

## 3 Neural Network Model

Our probability model is based on neural network language models with distributed representations (Bengio et al., 2003; Mnih and Hinton, 2007), as well as feed-forward neural network models for transition-based dependency parsing (Chen and Manning, 2014; Weiss et al., 2015). We estimate the distributions $p(t_i|\boldsymbol{h}_i)$, $p(w_i|t_i, \boldsymbol{h}_i)$ and $p(a_j|\boldsymbol{h}_j)$ with neural networks with shared input and hidden layers but separate output layers.

The templates for the conditioning context used are defined in Table 1. In the templates we obtain sentence indexes, which are then mapped to the corresponding words, tags and labels (for the dependencies of 2nd and 3rd order elements). The neural network allows us to include a large number of elements without suffering from sparsity.

In the input layer we make use of additive representations (Botha and Blunsom, 2014) so that for each word input position $i$ we can include the word type, tag and other features, and learn input representations for each of these. Each context feature $f$ has an input representation $\mathbf{q}_f \in \mathbb{R}^D$. The composite representation is computed as $\mathbf{q}_i = \sum_{f \in \mu(w_i)} \mathbf{q}_f$, where $\mu(w_i)$ are the word features.

The hidden layer is then defined as

$$\phi(\boldsymbol{h}) = g(\sum_{j=1}^{L} \mathbf{C}_j \mathbf{q}_{h_j}),$$

where $\mathbf{C}_j \in \mathbb{R}^{D \times D}$ are transformation matrices defined for each position in sequence $\boldsymbol{h}$, $L = |\boldsymbol{h}|$ and $g$ is a (usually non-linear) activation function applied element-wise. The matrices $\mathbf{C}_j$ can be approximated to be diagonal to reduce the number of model parameters and speed up the model by avoiding expensive matrix multiplications.

For the output layer predicting the next transition $a$, the hidden layer is mapped with a scoring function

$$\chi(a, \boldsymbol{h}) = \mathbf{k}_a^T \phi(\boldsymbol{h}) + e_a,$$

where $\mathbf{k}_a$ is the transition output representation and $e_a$ is the bias weight. The score is normalised with the soft-max function:

$$p(a|\boldsymbol{h}) = \frac{\exp(\chi(a, \boldsymbol{h}))}{\sum_{a' \in A} \exp(\chi(a', \boldsymbol{h}))}.$$

The output layer for predicting the next tag has a similar form, using the scoring function

$$\tau(t, \boldsymbol{h}) = \mathbf{t}_t^T \phi(\boldsymbol{h}) + o_t$$

for tag representation $\mathbf{t}_t$ and bias $o_t$.

The probability $p(w|t, \boldsymbol{h})$ can be estimated similarly. However, to reduce the computational cost of normalising over the entire vocabulary, we factorize the probability as $P(w|\boldsymbol{h}) = P(c|t, \boldsymbol{h})P(w|c, t, \boldsymbol{h})$, where $c = c(w)$ is the unique class of word $w$. For each $c$, let $\Gamma(c)$ be the set of words in that class. The vocabulary is clustered into approximately $\sqrt{|V|}$ classes using Brown clustering (Brown et al., 1992), reducing the number of items to sum over in the normalisation factor from $O(|V|)$ to $O(\sqrt{|V|})$. Class-based factorization has been shown to be an effective strategy in normalizing neural language models (Baltescu and Blunsom, 2015),

The class prediction score is defined as $\psi(c, \boldsymbol{h}) = \mathbf{s}_c^T \phi(\boldsymbol{h}) + d_c$, where $\mathbf{s}_c \in \mathbb{R}^D$ is the output weight vector for class $c$ and $d_c$ is the class bias weight. The output layer then consists of a softmax function for $p(c|\boldsymbol{h})$ and another softmax for the word prediction

$$p(w|c, \boldsymbol{h}) = \frac{\exp(\Phi(w, \boldsymbol{h}))}{\sum_{w' \in \Gamma(c)} \exp(\Phi(w', \boldsymbol{h}))},$$

where $\Phi(w, \boldsymbol{h}) = \mathbf{r}_w^T \phi(\boldsymbol{h}) + b_w$ is the word scoring function with output word representation $\mathbf{r}_w$ and bias weight $b_w$.

The model is trained with minibatch stochastic gradient descent (SGD) with Adagrad (Duchi et al., 2011) and L2 regularisation, to minimise the negative log likelihood of the joint distribution over parsed training sentences. For our experiments we train the model while the training objective improves, and choose the parameters of the iteration with the best development set accuracy (early stopping). The model obtains high accuracy with only a few training iterations.

## 4 Decoding

Beam-search decoders for transition-based parsing (Zhang and Clark, 2008) keep a beam of partial derivations, advancing each derivation by one transition at a time. When the size of the beam exceeds a set threshold, the lowest-scoring derivations are removed. However, in an incremental generative model we need to compare derivations with the same number of words shifted, rather than transitions performed. To let the decoding time remain linear, we also need to bound the total number of reduce transitions that can be performed over all derivations between two shift transitions.

To achieve this, we use a decoding method recently proposed for generative incremental parsing (Buys and Blunsom, 2015) based on particle filtering (Doucet et al., 2001), a sequential Monte Carlo sampling method.

In the algorithm, a fixed number of particles are divided among the partial derivations in the beam. Suppose $i$ words have been shifted in all the derivations on the beam. To predict the next transition from derivation $d_j$, its particles are divided according to $p(a|\boldsymbol{h})$. In practice, adding only shift and the most likely reduce transition leads to almost no accuracy loss. After all the derivations have been advanced to shift word $i+1$, a selection step is performed: The number of particles of each derivation is redistributed according to its probability, weighted by its current number of particles. Some derivations may be assigned 0 particles, in which case they are removed.

The particle filtering method lets the beam size depend of the uncertainty of the model, somewhat similar to Choi and Mccallum (2013), while fixing the total number of particles constrains the decoding time to be linear. The particle filter also allow us to sample outputs, and to marginalise over the syntax when generating.

## 5 Experiments

We evaluate our model for parsing and language modelling on the English Penn Treebank (Marcus et al., 1993) WSJ parsing setup[1]. Constituency trees are converted to projective CoNLL syntactic dependencies (Johansson and Nugues, 2007) with the LTH converter[2]. For some experiments

[1] Training on sections 02-21, development on section 22, and testing on section 23.

[2] http://nlp.cs.lth.se/software/treebank_converter/

| Activation | UAS | LAS |
|---|---|---|
| linear | 88.40 | 86.48 |
| rectifier | 89.99 | 88.31 |
| tanh | 90.91 | 89.22 |
| sigmoid | 91.48 | 89.94 |

Table 2: Parsing accuracies using different neural network activation functions.

| Model | UAS | LAS |
|---|---|---|
| Wallach et al. (2008) | 85.7 | - |
| Titov and Henderson (2007) | 90.93 | 89.42 |
| **NN-GenDP** | **91.11** | **89.41** |
| Chen and Manning (2014) | 92.0 | 90.7 |

Table 3: Parsing accuracies for dependency parsers on the WSJ test set, CoNLL dependencies.

we also use the Stanford dependency representation (De Marneffe and Manning, 2008) (SD)[3].

Our neural network implementation is partly based on the OxLM neural language modelling framework (Baltescu et al., 2014). The model parameters are initialised randomly by drawing from a Gaussian distribution with mean 0 and variance $0.1$, except for the bias weights, which are initialised by the unigram distributions of their output. We use minibatches of size 128, the L2 regularization parameter is 10, and the word representation and hidden layer of size is 256. The Adagrad learning rate is initialised to 0.05.

POS tags for the development and test sets are obtained with the Stanford POS tagger (Toutanova et al., 2003), with $97.5\%$ test set accuracy. Words that occur only once in the training data are treated as unknown words. Unknown words are replaced by tokens representing morphological surface features (based on capitalization, numbers, punctuation and common suffixes) similar to those used in the implementation of generative constituency parsers (Klein and Manning, 2003).

### 5.1 Parsing results

We report unlabelled attachment score (UAS) and labelled attachment score (LAS) in our results, excluding punctuation. On the development set, we consider the effect of the choice of activation function (Table 2), finding that a sigmoid activation (logistic function) performs best, following by `tanh`. Under our training setup the model can obtain up to 91.0 UAS after only 1 training iteration, thereby performing pure online learning.

We found that including third order dependencies in the conditioning context performs just $0.1\%$ UAS better than including only first and second order dependencies. Including additional elements does not improve performance further. The model can obtain $91.18$ UAS, $89.02$ LAS when trained only on words, not POS tags. Dependency parsers that do not use distributed representations tend to rely much more on the tags.

Test set results comparing generative dependency parsers are given in Table 3 (our model is refered to as NN-GenDP). The graph-based generative baseline (Wallach et al., 2008), parameterised by Pitman-Yor Processes, is quite weak. Our model outperforms the generative model of Titov and Henderson (2007), which we retrained on our dataset, by $0.2\%$, despite that model being able to condition on arbitrary-sized contexts. The decoding speed of our model is around 20 sentences per second, against less than 1 sentence per second for Titov and Henderson's model. Using diagonal transformation matrices further increases our model's speed, but reduces parsing accuracy.

On the Stanford dependency representation our model obtains $90.63\%$ UAS, $88.27\%$ LAS. Although this performance is promising, it is still below the discriminative neural network models of Dyer et al. (2015) and Weiss et al. (2015), who obtained $93.1\%$ UAS and $94.0\%$ UAS respectively.

### 5.2 Language modelling

We also evaluate our parser as a language model, on the same WSJ data used for the parsing evaluation[4]. We perform unlabelled parsing, as experiments show that including labels in the conditioning context has a very small impact on performance. Neither do we use POS tags, as they are too expensive to predict in language generation applications.

Perplexity results on the WSJ are given in Table 4. As baselines we report results on modified Knesser-Ney (Kneser and Ney, 1995) and neural network 5-gram models. For our dependency-based language models we report perplexities based on the most likely parse found by the decoder, which gives an upper bound on the true

[3] Converted with version 3.4.1 of the Stanford parser, available at http::/nlp.stanford.edu/software/lex-parser.shtml.

[4] However instead of using multiple unknown word classes, we replace all numbers by 0 and have a single unknown word token.

| |
|---|
| the u.s. union board said revenue rose 11 % to $ NUM million , or $ NUM a share . |
| mr. bush has UNK-ed a plan to buy the company for $ NUM to NUM million , or $ NUM a share . |
| the plan was UNK-ed by the board 's decision to sell its $ NUM million UNK loan loan funds . |
| in stocks coming months , china 's NUM shares rose 10 cents to $ NUM million , or $ NUM a share . |
| in the case , mr. bush said it will sell the company business UNK concern to buy the company . |
| it was NUM common shares in addition , with $ NUM million , or $ NUM a share , according to mr. bush . |
| in the first quarter , 1989 shares closed yesterday at $ NUM , mr. bush has increased the plan . |
| last year 's retrenchment price index index rose 11 cents to $ NUM million , or $ NUM million is asked . |
| last year earlier , net income rose 11 million % to $ NUM million , or 91 cents a share . |
| the u.s. union has UNK-ed $ NUM million , or 22 cents a share , in 1990 , payable nov. 9 . |

Table 5: Sentences of length 20 or greater generated by the neural generative dependency model.

| Model | Perplexity |
|---|---|
| KN 5-gram | 145.7 |
| NN 5-gram | 142.5 |
| **NN-GenDP** | **132.2** |
| **NN-GenDP + unsup** | **111.8** |

Table 4: WSJ Language modelling test results. We compare our model, with and without unsupervised tuning, to $n$-gram baselines.

value of the model perplexity.

First we only perform standard supervised training with the model - this already leads to an improvement of 10 perplexity points over the neural $n$-gram model. Second we consider a training setup where we first perform 5 supervised iterations, and then perform unsupervised training, treating the transition sequence as latent. For each minibatch parse trees are sampled with a particle filter. This approach further improves the perplexity to 111.8, a $23\%$ reduction relative to the Knesser-Ney model.

The unsupervised training stage lets the parsing accuracy fall from 91.48 to 89.49 UAS. We postulate that the model is learning to make small adjustments to favour of parsing structures that explain the data better than the annotated parse trees, leading to the improvement in perplexity.

To test the scalability of our model, we also trained it on a larger unannotated corpus – a subset (of around 7 million words) of the billion word language modeling benchmark dataset (Chelba et al., 2013). After training the model on the WSJ, we parsed the unannotated data with the model, and continued to train on the obtained parses. We observed a small increase in perplexity, from 203.5 for a neural $n$-gram model to 200.7 for the generative dependency model. We expect larger improvements when training on more data and with more sophisticated inference.

To evaluate our generative model qualitatively, we perform unconstrained generation of sentences (and parse trees) from the model, and found that sentences display a higher degree of syntactic coherence than sentences generated by an $n$-gram model. See Table 5 for examples generated by the model. The highest-scoring sentences of length 20 or more are given, from 1000 samples generated. Note that the generation includes unknown word tokens (here NUM, UNK and UNK-ed are used).

## 6 Conclusion

We presented an incremental generative dependency parser that can obtain accuracies competitive with discriminative models. The same model can be applied as an efficient syntactic language model, and for future work it should be integrated into language generation tasks such as machine translation.

## Acknowledgements

We acknowledge the financial support of the Oxford Clarendon Fund and the Skye Foundation.

## References

Paul Baltescu and Phil Blunsom. 2015. Pragmatic neural language modelling in machine translation. In *Proceedings of NAACL-HTL*, pages 820–829.

Paul Baltescu, Phil Blunsom, and Hieu Hoang. 2014. Oxlm: A neural language modelling framework for machine translation. *The Prague Bulletin of Mathematical Linguistics*, 102(1):81–92, October.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the ACL*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of EMNLP-CONLL*, pages 1455–1465.

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Jan Buys and Phil Blunsom. 2015. A Bayesian model for generative transition-based dependency parsing. *arXiv preprint arXiv:1506.04334*.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of ACL*, pages 124–131. Association for Computational Linguistics.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech & Language*, 14(4):283–332.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Jinho D. Choi and Andrew Mccallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of ACL*.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of EMNLP*, pages 1234–1245.

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*, pages 1–8.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of ACL*, pages 1370–1380.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. 2001. *Sequential Monte Carlo methods in practice*. Springer.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Joural of Machine Learning Research*, 12:2121–2159.

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 ACL 2015*.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING*, pages 340–345.

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning*, 60(1-3):195–227.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *16th Nordic Conference of Computational Linguistics*, pages 105–112, Tartu, Estonia.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of EMNLP*, pages 1700–1709.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, pages 423–430.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *ICASSP*, volume 1, pages 181–184. IEEE.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of ACL: Short Papers*.

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.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine learning*, pages 641–648.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING*.

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

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL*, pages 173–180.

Hanna M Wallach, Charles Sutton, and Andrew McCallum. 2008. Bayesian modeling of dependency trees using hierarchical Pitman-Yor priors. In *ICML Workshop on Prior Knowledge for Text and Language Processing*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL 2015*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL-HLT: Short papers*, pages 188–193.