Semi-supervised Dependency Parsing using Bilexical Contextual Features from Auto-Parsed Data

Eliyahu Kiperwasser
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
elikip@gmail.com

Yoav Goldberg
Computer Science Department
Bar-Ilan University
Ramat-Gan, Israel
yoav.goldberg@gmail.com

Abstract

We present a semi-supervised approach to improve dependency parsing accuracy by using bilexical statistics derived from auto-parsed data. The method is based on estimating the attachment potential of head-modifier words, by taking into account not only the head and modifier words themselves, but also the words surrounding the head and the modifier. When integrating the learned statistics as features in a graph-based parsing model, we observe nice improvements in accuracy when parsing various English datasets.

1 Introduction

We are concerned with semi-supervised dependency parsing, namely how to leverage large amounts of unannotated data, in addition to annotated Treebank data, to improve dependency parsing accuracy. Our method (Section 2) is based on parsing large amounts of unannotated text using a baseline parser, extracting word-interaction statistics from the automatically parsed corpus, and using these statistics as the basis of additional parser features. The automatically-parsed data is used to acquire statistics about lexical interactions, which are too sparse to estimate well from any realistically-sized Treebank. Specifically, we attempt to infer a function $assoc(head, modifer)$ measuring the "goodness" of head-modifier relations ("how good is an arc in which *black* is a modifier of *jump*"). A similar approach was taken by Chen et al. (2009) and Van Noord et al. (2007). We depart from their work by extending the scoring to include a wider lexical context. That is, given the sentence fragment in Figure 1, we score the (incorrect) dependency arc (*black*, *jump*) based on the triplets (*the black fox*, *will jump over*). Learning a function between word triplets raises an extreme data sparsity issue, which we deal with by decomposing the interaction between triplets to a sum of interactions between word pairs. The decomposition we use is inspired by recent work in word-embeddings and dense vector representations (Mikolov et al., 2013a; Mnih and Kavukcuoglu, 2013). Indeed, we initially hoped to leverage the generalization abilities associated with vector-based representations. However, we find that in our setup, reverting to direct count-based statistics achieve roughly the same results (Section 3).

Our derived features improve the accuracy of a first-order dependency parser by 0.75 UAS points (absolute) when evaluated on the in-domain WSJ test-set, obtaining a final accuracy of 92.32 UAS for a first-order parser. When comparing to the strong baseline of using Brown-clusters based features (Koo et al., 2008), we find that our triplets-based method outperform them by over 0.27 UAS points. This is in contrast to previous works (e.g. (Bansal et al., 2014)) in which improvements over using Brown-clusters features were achieved only by adding to the cluster-based features, not by replacing them. As expected, combining both our features and the brown-cluster features result in some additional gains.

2 Our Approach

Our departure point is a graph-based parsing model (McDonald et al., 2005):

$$parse(x) = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} score(x, y)$$

$$score(x, y) = w \cdot \Phi(x, y) = \sum_{part \in y} w \cdot \phi(x, part)$$

Given a sentence x we look for the highest-scoring parse tree y in the space $\mathcal{Y}(x)$ of valid dependency trees over x. The score of a tree is determined by a linear model parameterized by a weights vector w, and a feature function $\Phi(x, y)$. To make the search

Features in $\phi^{ij}_{lex}(x, y)$
bin($S^{ij}(x, y)$)
bin($S^{ij}(x, y)$) $\circ$ dist(x,y)
bin($S^{ij}(x, y)$) $\circ$ pos(x) $\circ$ pos(y)
bin($S^{ij}(x, y)$) $\circ$ pos(x) $\circ$ pos(y) $\circ$ dist(x,y)

Table 1: All features are binary indicators. x and y are token indices. S^{ij} is estimated from auto-parsed corpora as described in the text. The values of $S(\cdot, \cdot)$ are in the range $(0, 1)$, which is split by bin into 10 equally-spaced intervals. dist is the signed and binned sentence-distance between x and y. pos(x) is the part of speech of token x. $\circ$ indicates a concatenation of features.

tractable, the feature function is decomposed into local feature functions over tree-parts $\phi(x, part)$. The features in ϕ are standard graph-based dependency parsing features, capturing mostly structural information from the parse tree.

We extend the scoring function by adding an additional term capturing the strength of *lexical* association between head word h and modifier word m in each dependency arc:

$$score(x, y) = \sum_{part \in y} w \cdot \phi(x, part) + \sum_{(h,m) \in y} assoc(h, m)$$

The association function $assoc$ is also modeled as a linear model $assoc(h, m) = w_{lex} \cdot \phi_{lex}(h, m)$. While the weights w_{lex} are trained jointly with w based on supervised training data, the features in ϕ_{lex} do not look at h and m directly, but instead rely on a quantity $S(h, m)$ that reflects the "goodness" of the arc (h, m). The quantity $S(h, m)$ ranges between 0 and 1, and is estimated based on large quantities of auto-parsed data. Given a value for $S(h, m)$, ϕ_{lex} is composed of indicator functions indicating the binned ranges of $S(h, m)$, possibly conjoined with information such as the binned surface distance between the tokens h and m and their parts of speech. The complete specification of ϕ_{lex} we use is shown in Table 1 (the meaning of the ij indices will be discussed in Section 2.2).

2.1 Estimating $S(h, m)$

One way of estimating $S(h, m)$, which was also used in (Chen et al., 2009), is using rank statistics. Let D be the list of observed (h, m) pairs sorted by their frequencies, and let $rank(h, m)$ be the index of the pair (h, m) in D. We now set:

$$S_{\text{RANK}}(h, m) = \frac{rank(h, m)}{|D|}$$

While effective, this approach has two related shortcomings: first, it requires storing counts for all the pairs (h, m) appearing in the auto-parsed data, resulting in memory requirement that scales quadratically with the vocabulary size. Second, even with very large auto-parsed corpora many plausible head-modifier pairs are likely to be unobserved.

An alternative way of estimating $S(h, m)$ that does not require storing all the observed pairs and that has a potential of generalizing beyond the seen examples is using a log-bilinear embedding model similar to the skip-gram model presented by Mikolov et al. (2013b) to embed word pairs such that compatible pairs receive high scores. The model assigns two disjoint sets of d-dimensional continuous latent vectors, u and v, where $u_h \in R^d$ is an embedding of a head word h, and $v_m \in R^d$ is an embedding of a modifier word m. The embedding is done by trying to optimize the following corpus-wide objective that is maximizing the dot product of the vectors of observed (h, m) pairs and minimizing the dot product of vectors of random h and m pairs. Formally:

$$\sum_{h,m \in C} \left[ln\left(\sigma\left(u_h \cdot v_m\right)\right) - \sum_{i=1}^{k} \mathop{\mathbb{E}}_{m_i \sim P_m} ln\left(\sigma\left(u_h \cdot v_{m_i}\right)\right) \right]$$

where $\sigma(x) = 1/(1 + e^{-x})$, and k is the number of negative samples, drawn from the corpus-based Unigram distribution P_m. For further details, see (Mikolov et al., 2013b; Goldberg and Levy, 2014). We then take:[1]

$$S_{\text{EMBED}}(h, m) = \sigma(u_h \cdot v_m)$$

In contrast to the counts based method, this model is able to estimate the strength of a pair of words even if the pair did not appear in the corpus due to sparsity.

Finally, Levy and Goldberg (2014b) show that the skip-grams with negative-sampling model described above achieves its optimal solution when $u_h \cdot v_m = PMI(h, m) - \log\ k$. This gives rise to another natural way of estimating S:

[1]The embedding we derive are very similar to the ones described in (Levy and Goldberg, 2014a; Bansal et al., 2014), and which were used by Bansal et al.(2014) for deriving semi-supervised parsing features. An important difference from these previous work is that after training, they keep only one set of vectors (u or v) and ignore the other, basing the features on the derived vector representations. In contrast, we keep both sets of vectors and are interested in the association measure induced by the dot product $u_h \cdot v_m$.

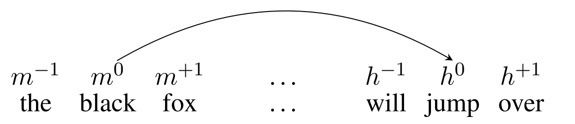

Figure 1: Illustration of the bilexical information including context. When scoring the (incorrect) arc between h^0 and m^0, we take into account also the surrounding words h^{-1}, h^{+1}, m^{-1} and m^{+1}.

$$S_{\text{PMI}}(h,m) = \sigma(PMI(h,m)) = \frac{p(h,m)}{p(h,m)+p(h)p(m)}$$

where $p(h,m)$, $p(h)$ and $p(m)$ are unsmoothed maximum-likelihood estimates based on the auto-parsed corpus.

Like S_{RANK} and unlike S_{EMBED}, S_{PMI} requires storing statistics for all observed word pairs, and is not able of generalizing beyond (h,m) pairs seen in the auto-parsed data. However, as we see in Section 3, this method performing similarly in practice, suggesting that the generalization capabilities of the embedding-based method do not benefit the parsing task.

2.2 Adding additional context

Estimating the association between a pair of words is effective. However, we would like to go a step further and take into account also the context in which these words occur. Specifically, our complete model attempts to estimate the association between word trigrams centered around the head and the modifier words.

A naive solution that defines each trigram as its own vocabulary item will increase the vocabulary size by two orders of magnitude and result in severe data sparsity. An alternative solution would be to associate each word in the triplet (h^{-1}, h^0, h^{+1}) with its own unique vocabulary item, and likewise for modifier words. In the embeddings-based model, this results in 6 vector sets $u^{-1}, u^0, u^{+1}, v^{-1}, v^0, v^{+1}$, where v^{-1}_{dog}, for example, represents the word "dog" when it appears to the left of the modifier word, and u^{+1}_{walk} the word "walk" when it appears to the right of the head word.[2] This amounts to only a 3-fold increase in the required vocabulary size. We then model the strength of association between $h^{-1}h^0h^{+1}$ and $m^{-1}m^0m^{+1}$ as a weighted sum of pairwise interactions:[3]

$$assoc(h^{-1}h^0h^{+1}, m^{-1}m^0m^{+1}) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \alpha_{ij}\, assoc_{ij}(h^i, m^j)$$

As before, the pairwise association measure $assoc_{ij}(h^i, m^j)$ is modeled as a linear model:

$$assoc_{ij}(h^i, m^j) = w^{ij}_{lex} \cdot \phi^{ij}_{lex}(h^i, m^j)$$

Where ϕ^{ij}_{lex} is again defined in terms of a goodness function $S^{ij}(x,y)$. For example, $S^{-1,+1}(the, over)$ corresponds to the goodness of a head-modifier arc where the word to the left of the head word is "the" and the word to the right of the modifier word is "over". $S^{ij}(h^i, m^j)$, the goodness of the arc induced by the pair (h^i, m^j), can be estimated by either S_{RANK}, S_{EMBED} or S_{PMI} as before. For example, in the embeddings model we set $S^{ij}(x,y) = \sigma(u^i_x \cdot v^j_y)$.

We update the bilexical features to include context as explained above. Instead of learning α and w_{lex} coefficients separately, we absorb the α_{ij} terms into w_{lex}, learning both at the same time:

$$\begin{aligned} &assoc(h^{-1}h^0h^{+1}, m^{-1}m^0m^{+1}) = \\ &\sum_{i\in\{-1,0,1\}}\sum_{j\in\{-1,0,1\}} \alpha_{ij} \cdot assoc_{ij}(h^i, m^j) = \\ &\sum_{i\in\{-1,0,1\}}\sum_{j\in\{-1,0,1\}} \alpha_{ij} \cdot w^{ij}_{lex} \cdot \phi^{ij}_{lex}(h^i, m^j) = \\ &\sum_{i\in\{-1,0,1\}}\sum_{j\in\{-1,0,1\}} w'^{ij}_{lex} \cdot \phi^{ij}_{lex}(h^i, m^j) \end{aligned}$$

Finally, the parser selects a dependency tree which maximizes the following:

$$\begin{aligned} w' \cdot \Phi(x,y) = &\sum_{part\in y} w \cdot \phi(x, part) \\ &+ \sum_{(h,m)\in y}\sum_{i=-1}^{1}\sum_{j=-1}^{1} w^{ij}_{lex} \cdot \phi^{ij}_{lex}(h^i, m^j) \end{aligned}$$

[2]Sentences are padded by special sentence-boundary symbols.

[3]In the word embeddings literature, it is common to represent a word triplet as the sum of the individual component vectors, resulting in $u_{x,y,z} \cdot v_{a,b,c} = (u^{-1}_x + u^0_y + u^{+1}_z) \cdot (v^{-1}_a + v^0_b + v^{+1}_c)$. Expanding the terms will result in a very similar formulation to our proposal, but we allow the extra flexibility of associating a different strength α_{ij} with each pairwise term.

	Dev	Test	Brown	Answers	Blogs	Email	News	Reviews
Baseline	91.97	91.57	86.86	81.58	84.88	79.75	82.59	83.22
Base + Brown	92.16	92.05	87.16	81.96	85.46	80.27	83.02	83.56
Base + HM(S_{RANK})	92.16	91.74	87.01	82.06	85.36	80.29	82.72	83.62
Base + HM(S_{EMBED})	92.29	91.98	87.04	81.97	85.34	79.93	82.76	83.33
Base + HM(S_{PMI})	92.35	92.00	87.14	82.20	85.65	80.34	82.83	83.81
Base + TRIP(S_{RANK})	92.23	91.91	87.02	82.31	85.59	**80.50**	83.30	83.79
Base + TRIP(S_{EMBED})	92.38	92.27	87.15	82.34	85.71	80.41	83.21	83.68
Base + TRIP(S_{PMI})	**92.61**	**92.32**	**87.29**	**82.58**	**85.88**	80.43	**83.57**	**84.18**
Base + Brown + TRIP(S_{RANK})	92.43	92.33	87.23	82.60	85.75	80.57	83.48	84.00
Base + Brown + TRIP(S_{EMBED})	92.51	**92.45**	87.33	82.42	86.06	80.44	83.40	83.87
Base + Brown + TRIP(S_{PMI})	**92.70**	92.40	**87.42**	**82.74**	**86.08**	**80.72**	**83.78**	**84.26**

Table 2: Parsing accuracies (UAS, excluding puctuation) of the different models on various corpora. All models are trained on the PTB training set. Dev and Test are sections 22 and 23. Brown is the Brown portion of the PTB. The other columns correspond to the test portions of the Google Web Treebanks. Automatic POS-tags are assigned in all cases. HM indicates using $assoc(h, m)$ and TRIP using $assoc(h^{-1}h^{0}h^{1}, m^{-1}m^{0}m^{1})$. + BROWN indicate using features based on Brown clustering.

3 Experiments and Results

Data Our experiments are based on the Penn Treebank (PTB) (Marcus et al., 1993) as well as the Google Web Treebanks (LDC2012T13), covering both in-domain and out-of-domain scenarios. We use the Stanford-dependencies representation (de Marneffe and Manning, 2008). All the constituent-trees are converted to Stanford-dependencies based on the settings of Version 1.0 of the Universal Treebank (McDonald et al., 2013).[4] These are based on the Stanford Dependencies converter but use some non-default flags, and change some of the dependency labels. All of the models are trained on section 2-21 of the WSJ portion of the PTB. For in-domain data, we evaluate on sections 22 (Dev) and 23 (Test). All of the parameter tuning were performed on the Dev set, and we report test-set numbers only for the "most interesting" configurations. For out-of-domain data, we use the Brown portion of the PTB (Brown), as well as the test-sets of different domains available in the Google Web Treebank: Answers, Blogs, Emails, Reviews and Newsgroups.

All trees have automatically assigned part-of-speech tags, assigned by the TurboTagger POS-tagger.[5] The train-set POS-tags were derived in a 10-fold jacknifing, and the different test datasets receive tags from a tagger trained on sections 2-21.

For auto-parsed data, we parse the text of the BLLIP corpus (Charniak, 2000) using our baseline parser. This is the same corpus used for deriving Brown clusters for use as features in (Koo et al., 2008). We use the clusters provided by Terry Koo[6]. Parsing accuracy is measured by unlabeled attachment score (UAS) excluding punctuations.

Implementation Details We focus on first-order parsers, as they are the most practical graph-based parsers in terms of running time in realistic parsing scenarios. Our base model is a re-implementation of a first-order projective Graph-based parser (McDonald et al., 2005), which we extend to support the semi-supervised ϕ_{lex} features. The parser is trained for 10 iterations of online-training with passive-aggressive updates (Crammer et al., 2006). For the Brown-cluster features, we use the feature templates described by (Koo et al., 2008; Bansal et al., 2014).

The embedding vectors are trained using the freely available `word2vecf` software[7], by conjoining each word with its relative position (-1, 0 or 1) and treating the head words as "words" and the modifier words to be "contexts". The words are embedded into 300-dimensional vectors. All code and vectors will be available at the first author's website.

Results The results are shown in Table 2. The second block (HM) compares the baseline parser to a parser including the $assoc(h, m)$ lexical component, using various ways of computing $s(h, m)$. We observe a clear improvement above the baseline from using the lexical component across all domains. The different estimation methods perform very similar to each other.

In the third block (TRIP) we switch to the triplet-based lexical association. With S_{RANK},

[4]`https://github.com/ryanmcd/uni-dep-tb/raw/master/universal_treebanks_v1.0.tar.gz`

[5]`http://www.ark.cs.cmu.edu/TurboParser/`

[6]`http://people.csail.mit.edu/maestro/papers/bllip-clusters.gz`

[7]`http://www.bitbucket.org/yoavgo/word2vecf`

there is very little advantage over looking at just pairs. However, with S_{EMBED} or S_{PMI} the improvement of using the triplet-based method over using just the head-modifier pairs is clear. The counting-based PMI method performs on par with the Embedding based approximation of it.

The second line of the first block (Base+Brown) represents the current state-of-the-art in semi-supervised training of graph-based parsing: using Brown-cluster derived features (Koo et al., 2008; Bansal et al., 2014). The Brown-derived features provide similar (sometimes larger) gains to using our HM method, and substantially smaller gains than our TRIP method. To the best of our knowledge, we are the first to show a semi-supervised method that significantly outperforms the use of Brown-clusters without using Brown-clusters as a component.

As expected, combining our features and the Brown-based features provide an additional improvement, as can be seen in the last block of Table 2 (Base+Brown+TRIP).

4 Related Work

Semi-supervised approaches to dependency parsing can be roughly categorized into two groups: those that use unannotated data and those that use automatically-parsed data. Our proposed method falls in the second group.

Among the words that use unannotated data, the dominant approach is to derive either word clusters (Koo et al., 2008) or word vectors (Chen and Manning, 2014) based on unparsed data, and use these as additional features for a supervised parsing model. While the word representations used in such methods are not specifically designed for the parsing task, they do provide useful features for parsing, and in particular the method of (Koo et al., 2008), relying on features derived using the Brown-clustering algorithm, provides very competitive state-of-the-art results. To the best of our knowledge, we are the first to show a substantial improvement over using Brown-clustering derived features without using Brown-cluster features as a component.

Among the words that use auto-parsed data, a dominant approach is self-training (McClosky et al., 2006), in which a parser A (possibly an ensemble) is used to parse large amounts of data, and a parser B is then trained over the union of the gold data and the auto-parsed data produced by parser A. In the context of dependency-parsing, successful uses of self-training require parser A to be stronger than parser B (Petrov et al., 2010) or use a selection criteria for training only on high-quality parses produced by parser A (Sagae and Tsujii, 2007; Weiss et al., 2015). In contrast, our work uses the same parser (modulo the feature-set) for producing the auto-parsed data and for training the final model, and does not employ a high-quality parse selection criteria when creating the auto-parsed corpus. It is possible that high-quality parse selection can improve our proposed method even further.

Works that derive features from auto-parsed data include (Sagae and Gordon, 2009; Bansal et al., 2014). Such works assign a representation (either cluster or vector) for individual word in the vocabulary based on their syntactic behavior. In contrast, our learned features are designed to capture *interactions* between words. As discussed in sections (1) and (2), most similar to ours is the work of (Chen et al., 2009; Van Noord, 2007). We extend their approach to take into account not only direct word-word interactions but also the lexical surroundings in which these interactions occur.

Another recent approach that takes into account various syntactic interactions was recently introduced by Chen et al. (2014), who propose to learn to embed complex features that are being used in a graph-based parser based on other features they co-occur with in auto-parsed data. Similar to our approach, the embedded features are then used as additional features in a conventional graph-based model. The approaches are to a large extent complementary, and could be combined.

Finally, our work adds additional features to a graph-based parser which is based on a linear-model. Recently, progress in dependency parsing has been made by introducing non-linear, neural-network based models (Pei et al., 2015; Chen and Manning, 2014; Weiss et al., 2015; Dyer et al., 2015; Zhou et al., 2015). Adapting our approach to work with such models is an interesting research direction.

5 Conclusions

We presented a semi-supervised method for dependency parsing and demonstrated its effectiveness on a first-order graph-based parser. Taking into account not only the (head,modifier) word-pair but also their immediate surrounding words add a clear benefit to parsing accuracy.

References

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

Eugene Charniak. 2000. Bllip 1987-89 wsj corpus release 1. In *Linguistic Data Consortium*, Philadelphia.

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

Wenliang Chen, Jun'ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. 2009. Improving dependency parsing with subtrees from auto-parsed data. In *EMNLP*, pages 570–579.

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proc. of COLING*.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

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.

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

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proc. of ACL*, pages 595–603.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *Proc. of ACL*, pages 302–308, Baltimore, Maryland.

Omer Levy and Yoav Goldberg. 2014b. Neural word embeddings as implicit matrix factorization. In *Proc. of NIPS*, pages 2177–2185.

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.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proc. of NAACL*, pages 152–159.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of ACL*.

Ryan T McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proc. of ACL*.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proc. of EMNLP-2010*, October.

Kenji Sagae and Andrew S. Gordon. 2009. Clustering words by syntactic similarity improves dependency parsing of predicate-argument structures. In *IWPT*, pages 192–201.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proc. of CoNLL-2007 Shared Task*.

Gertjan Van Noord. 2007. Using self-trained bilexical preferences to improve disambiguation accuracy. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 1–10. Association for Computational Linguistics.

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

Hao Zhou, Yue Zhang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proc. of ACL*.