Simpler but More Accurate Semantic Dependency Parsing

Timothy Dozat
Stanford University
`tdozat@stanford.edu`

Christopher D. Manning
Stanford University
`manning@stanford.edu`

Abstract

While syntactic dependency annotations concentrate on the surface or functional structure of a sentence, semantic dependency annotations aim to capture between-word relationships that are more closely related to the meaning of a sentence, using graph-structured representations. We extend the LSTM-based syntactic parser of Dozat and Manning (2017) to train on and generate these graph structures. The resulting system on its own achieves state-of-the-art performance, beating the previous, substantially more complex state-of-the-art system by 0.6% labeled F1. Adding linguistically richer input representations pushes the margin even higher, allowing us to beat it by 1.9% labeled F1.

1 Introduction

Syntactic dependency parsing is arguably the most popular method for automatically extracting the low-level relationships between words in a sentence for use in natural language understanding tasks. However, typical syntactic dependency frameworks are limited in the number and types of relationships that can be captured. For example, in the sentence *Mary wants to buy a book*, the word *Mary* is the subject of both *want* and *buy*—either or both relationships could be useful in a downstream task, but a tree-structured representation of this sentence (as in Figure 1a) can only represent one of them.[1]

The 2014 SemEval shared task on Broad-Coverage Semantic Dependency Parsing (Oepen et al., 2014) introduced three new dependency representations that do away with the assumption of strict tree structure in favor of a richer graph-structured representation, allowing them to capture more linguistic information about a sentence. This opens up the possibility of providing more useful information to downstream tasks (Reddy et al., 2017; Schuster et al., 2017), but increases the difficulty of automatically extracting that information, since most previous work on parsing has focused on generating trees.

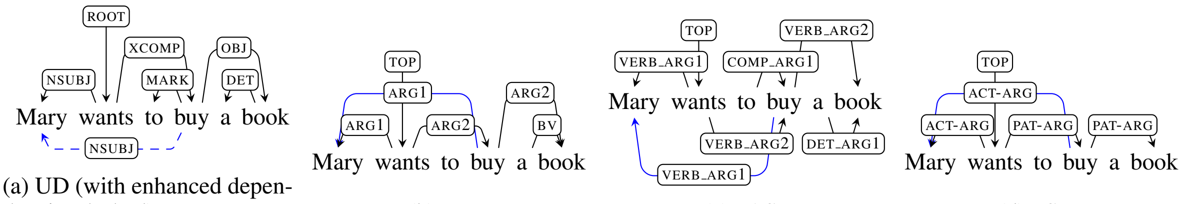
Dozat and Manning (2017) developed a successful syntactic dependency parsing system with few task-specific sources of complexity. In this paper, we extend that system so that it can train on and produce the graph-structured data of semantic dependency schemes. We also consider straightforward extensions of the system that are likely to increase performance over the straightforward baseline, including giving the system access to lemma embeddings and building in a character-level word embedding model. Finally, we briefly examine some of the design choices of that architecture, in order to assess which components are necessary for achieving the highest accuracy and which have little impact on final performance.

2 Background

2.1 Semantic dependencies

The 2014 SemEval (Oepen et al., 2014, 2015) shared task introduced three new semantic dependency formalisms, applied to the Penn Treebank (shown in Figure 1, compared to Universal Dependencies (Nivre et al., 2016)): DELPH-IN MRS, or DM (Flickinger et al., 2012; Oepen and Lønning, 2006); Predicate-Argument Structures, or PAS (Miyao and Tsujii, 2004); and Prague Semantic Dependencies, or PSD (Hajic et al., 2012). Whereas syntactic dependencies generally annotate functional relationships between words—such as *subject* and *object*—semantic dependencies aim

[1] Though efforts have been made to address this limitation; seeDe Marneffe et al. (2006); Nivre et al. (2016); Schuster and Manning (2016); Candito et al. (2017) for examples.



(a) UD (with enhanced dependencies dashed) (b) DM (c) PAS (d) PSD

Figure 1: Comparison between syntactic and semantic dependency schemes

to reflect semantic relationships—such as *agent* and *patient* (cf. semantic role labeling (Gildea and Jurafsky, 2002)). The SemEval semantic dependency schemes are also directed acyclic graphs (DAGs) rather than trees, allowing them to annotate function words as being heads without lengthening paths between content words (as in 1b).

2.2 Related work

Our approach to semantic dependency parsing is primarily inspired by the success of Dozat and Manning (2017) and Dozat et al. (2017) at syntactic dependency parsing and Peng et al. (2017) at semantic dependency parsing. In Dozat and Manning (2017) and Peng et al. (2017), parsing involves first using a multilayer bidirectional LSTM over word and part-of-speech tag embeddings. Parsing is then done using directly-optimized self-attention over recurrent states to attend to each word's head (or heads), and labeling is done with an analgous multi-class classifier.

Peng et al.'s (2017) system uses a max-margin classifer on top of a BiLSTM, with the score for each graph coming from several sources. First, it scores each word as either taking dependents or not. For each ordered pair of words, it scores the arc from the first word to the second. Lastly, it scores each possible labeled arc between the two words. The graph that maximizes these scores may not be consistent, with an edge coming from a non-predicate, for example, so they enforce hard constraints in order to prune away invalid semantic graphs. Decisions are not independent, so in order to find the highest-scoring graph that follows these constraints, they use the AD^3 decoding algorithm (Martins et al., 2011).

Dozat and Manning's (2017) approach to syntactic dependency parsing is similar, but avoids the possibility of generating invalid trees by fully factorizing the system. Rather than summing the scores from multiple modules and then finding the valid structure that maximizes that sum, the system makes parsing and labeling decisions sequentially, choosing the labels for each edge only after the edges in the tree have been finalized by an MST algorithm.

Wang et al. (2018) take a different approach in their recent work, using a transition-based parser built on stack-LSTMs (Dyer et al., 2015). They extend Choi and McCallum's (2013) transition system for producing non-projective trees so that it can produce arbitrary DAGs and they modify the stack-LSTM architecture slightly to make the network more powerful.

3 Approach

3.1 Basic approach

We can formulate the semantic dependency parsing task as labeling each edge in a directed graph, with *null* being the label given to pairs with no edge between them. Using only one module that labels each edge in this way would be an *unfactorized* approach. We can, however, factorize it into two modules: one that predicts whether or not a directed edge (w_j, w_i) exists between two words, and another that predicts the best label for each potential edge.

Our approach closely follows that of Dozat and Manning (2017). As with many successful recent parsers, we concatenate word and POS tag[2] embeddings, and feed them into a multilayer bidirectional LSTM to get contextualized word representations.[3]

$$\mathbf{x}_i = \mathbf{e}_i^{(\text{word})} \oplus \mathbf{e}_i^{(\text{tag})} \tag{1}$$
$$R = \text{BiLSTM}(X) \tag{2}$$

[2]We use the POS tags (and later, lemmas) provided with each dataset.

[3]We follow the convention of representing scalars in lowercase italics a, vectors in lowercase bold $\mathbf{a}$, matrices in uppercase italics A, and tensors in uppercase bold $\mathbf{A}$. We maintain this convention when indexing and stacking, so $\mathbf{a}_i$ is row i of matrix A and A contains the sequence of vectors $(\mathbf{a}_1, \ldots, \mathbf{a}_n)$.

For each of the two modules, we use single-layer feedforward networks (FNN) to split the top recurrent states into two parts—a *head* representation, as in Eq. (5, 6) and a *dependent* representation, as in Eq. (7, 8). This allows us to reduce the recurrent size to avoid overfitting in the classifer without weakening the LSTM's capacity. We can then use bilinear or biaffine classifiers in Eq. (3, 4)—which are generalizations of linear classifiers to include multiplicative interactions between two vectors—to predict edges and labels.[4]

$$\mathrm{Bilin}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{U} \mathbf{x}_2 \quad (3)$$

$$\mathrm{Biaff}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{U} \mathbf{x}_2 + W(\mathbf{x}_1 \oplus \mathbf{x}_2) + \mathbf{b} \quad (4)$$

$$\mathbf{h}_i^{(\text{edge-head})} = \mathrm{FNN}^{(\text{edge-head})}(\mathbf{r}_i) \quad (5)$$

$$\mathbf{h}_i^{(\text{label-head})} = \mathrm{FNN}^{(\text{label-head})}(\mathbf{r}_i) \quad (6)$$

$$\mathbf{h}_i^{(\text{edge-dep})} = \mathrm{FNN}^{(\text{edge-dep})}(\mathbf{r}_i) \quad (7)$$

$$\mathbf{h}_i^{(\text{label-dep})} = \mathrm{FNN}^{(\text{label-dep})}(\mathbf{r}_i) \quad (8)$$

$$s_{i,j}^{(\text{edge})} = \mathrm{Biaff}^{(\text{edge})}\left(\mathbf{h}_i^{(\text{edge-dep})}, \mathbf{h}_j^{(\text{edge-head})}\right) \quad (9)$$

$$\mathbf{s}_{i,j}^{(\text{label})} = \mathrm{Biaff}^{(\text{label})}\left(\mathbf{h}_i^{(\text{label-dep})}, \mathbf{h}_j^{(\text{label-head})}\right) \quad (10)$$

$$y_{i,j}^{\prime(\text{edge})} = \{s_{i,j} \geq 0\} \quad (11)$$

$$y_{i,j}^{\prime(\text{label})} = \arg\max \mathbf{s}_{i,j} \quad (12)$$

The tensor $\mathbf{U}$ can optionally be diagonal (such that $u_{i,k,j} = 0$ wherever $i \neq j$) to conserve parameters. The unlabeled parser scores every edge between pairs of words in the sentence—these scores can be decoded into a graph by keeping only edges that received a positive score. The labeler scores every label for each pair of words, so we simply assign each predicted edge its highest-scoring label and discard the rest. We can train the system by summing the losses from the two modules, backpropagating error to the labeler only through edges with a non-*null* gold label. This system is shown graphically in Figure 2. We find that sometimes the loss for one module overwhelms the loss for the other, causing the system to underfit. Thus we add a tunable interpolation constant $\lambda \in (0, 1)$ to even out the two losses.

$$\ell = \lambda \ell^{(\text{label})} + (1 - \lambda)\ell^{(\text{edge})} \quad (13)$$

Worth noting is that the removal of the maximum spanning tree algorithm and change from softmax cross-entropy to sigmoid cross-entropy in the unlabeled parser represent the only changes needed to allow the original syntactic parser to generate fully graph-structured semantic dependency output. Note also that this system is general enough that it could be used for any graph-structured dependency scheme, including the enhanced dependencies of the Universal Dependencies formalism (which allows cyclic graphs).

[4]For the labeled parser, $\mathbf{U}$ will be $(d \times c \times d)$-dimensional, where c is the number of labels. For the unlabeled parser, $\mathbf{U}$ will be $(d \times 1 \times d)$-dimensional, so that $s_{i,j}$ will be a single score.

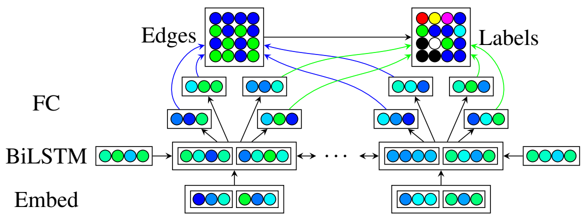


Figure 2: The basic architecture of our factorized system. Labels are only assigned to word pairs with an edge between them.

3.2 Augmentations

Ballesteros et al. (2016), Dozat et al. (2017), and Ma et al. (2018) find that character-level word embedding models improve performance for syntactic dependency parsing, so we also want to explore the impact it has on semantic dependency parsing. Dozat et al. (2017) confirm that their syntactic parser performs better with POS tags, which leads us to examine whether word lemmas—another form of low-level lexical information—might also improve dependency parsing performance.

4 Results

4.1 Hyperparameters

We tuned the hyperparameters for our basic system (with no character embeddings or lemmas) fairly extensively on the DM development data. The hyperparameter configuration for our final system is given in Table 2. All input embeddings (word, pretrained, POS, etc.) were concatenated. We used 100-dimensional pretrained GloVe embeddings (Pennington et al., 2014), but linearly transformed them to be 125-dimensional. Only words or lemmas that occurred 7 times or more were included in the word and lemma embedding matrix—including less frequent words appeared to facilitate overfitting. Character-level word embeddings were generated using a one-layer unidirectional LSTM that convolved over three character embeddings at a time, whose end state was linearly transformed to be 100-dimensional. The core BiL-

	DM		PAS		PSD		Avg	
	ID	OOD	ID	OOD	ID	OOD	ID	OOD
(Du et al., 2015)	89.1	81.8	91.3	87.2	75.7	73.3	85.3	80.8
(Almeida and Martins, 2015)	88.2	81.8	90.9	86.9	76.4	74.8	85.2	81.2
WCGL18	90.3	84.9	91.7	87.6	78.6	75.9	86.9	82.8
PTS17: Basic	89.4	84.5	92.2	88.3	77.6	75.3	87.4	83.6
PTS17: Freda3	90.4	85.3	92.7	89.0	78.5	76.4	88.0	84.4
Ours: Basic	91.4	86.9	93.9	**90.8**	79.1	77.5	88.1	85.0
Ours: +Char	92.7	87.8	**94.0**	90.6	80.5	78.6	89.1	85.7
Ours: +Lemma	93.3	88.8	93.9	90.5	80.3	78.7	89.1	86.0
Ours: +Char +Lemma	**93.7**	**88.9**	93.9	90.6	**81.0**	**79.4**	**89.5**	**86.3**

Table 1: Comparison between our system and the previous state of the art on in-domain (WSJ) and out-of-domain (Brown corpus) data, according to labeled F1 (LF1).

Hidden Sizes	
Word/Glove/POS/ Lemma/Char	100
GloVe linear	125
Char LSTM	1 @ 400
Char linear	100
BiLSTM	3 @ 600
Arc/Label	600
Dropout Rates (drop prob)	
Word/GloVe/ POS/Lemma	20%
Char LSTM (FF/recur)	33%
Char linear	33%
BiLSTM (FF/recur)	45%/25%
Arc/Label	25%/33%
Loss & Optimizer	
Interpolation (λ)	.025
L_2 regularization	$3e^{-9}$
Learning rate	$1e^{-3}$
Adam β_1	0
Adam β_2	.95

Table 2: Final hyperparameter configuration.

STM was three layers deep. The different types of word embeddings—word, GloVe, and character-level—were dropped simultaneously, but independently from POS and lemma embeddings (which were dropped independently from each other). Dropped embeddings were replaced with learned `<DROP>` tokens. LSTMs used same-mask recurrent dropout (Gal and Ghahramani, 2016). The systems were trained with batch sizes of 3000 tokens for up to 75,000 training steps, terminating early after 10,000 steps pass with no improvement in validation accuracy. The L_2 regularization penalty was so small that it likely had little impact.

4.2 Performance

We use biaffine classifiers, with no nonlinearities, and a diagonal tensor in the label classifier but not the edge classifier. The system trains at a speed of about 300 sequences/second on an nVidia Titan X and parses about 1,000 sequences/second. Du et al. (2015) and Almeida and Martins (2015) are the systems that won the 2015 shared task (closed track). *PTS17: Basic* represents the single-task versions of Peng et al. (2017), which they make multitask across the three datasets in Freda3 by adding *frustratingly easy domain adaptation* (Daumé III, 2007; Kim et al., 2016) and a third-order decoding mechanism. WCGL18 is Wang et al.'s (2018) transition-based system. Table 1 compares our performance with these systems. Our fully factorized basic system already substantially outperforms Peng et al.'s single-task baseline and also beats out their much more complex multi-task approach. Simply adding in either a character-level word embedding model (similar to Dozat et al.'s (2017)) or a lemma embedding matrix likewise improves performance quite a bit, and including both together generally pushes performance even higher. Many infrequent words were excluded from the frequent token embedding matrix, so it makes sense that the system should improve when provided more lexical information that's harder to overfit on.

Surprisingly, the PAS dataset seems not to benefit substantially from lemma or character embeddings. It has been noted that PAS is the easiest of the three datasets to achieve good performance for;

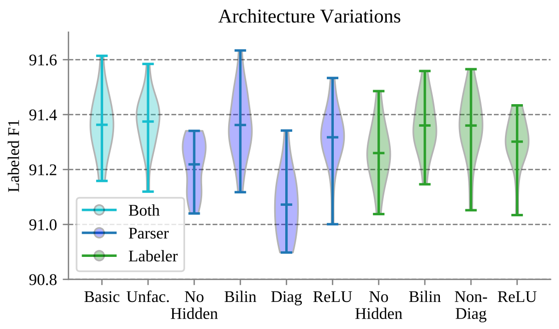


Figure 3: Performance of architecture variations: our basic system; unfactorized (labeler-only); ommitting the hidden layers (Eqs. 7–6); with bilinear classifiers (Eq. 3); with nondiagonal tensors in the labeler or diagonal tensors in the parser; with the ReLU nonlinearity.

so one possible explanation is that 94% LF1 may simply be near the ceiling of what can be achieved for the dataset. Alternatively, the main difference bewteen PAS as DM/PSD is that PAS includes semantically vacuous function words in its representation. Because function words are extremely frequent, it's possible that they are being disproportionately represented in the loss or LF1 score. Using a hinge loss (like Peng et al. (2017)) instead of a cross-entropy loss might help, since the system would stop focusing on potentially "easy" functional predicates once it learned to predict their argument structures confidently, allowing it to put more resources into modeling more challenging phenomena.

4.3 Variations

We also consider the impact that slight variations on basic architecture have on final performance in Figure 3. We train twenty models on the DM treebank for each variation we consider, reducing the number of training steps but keeping all other hyperparameters constant. Rank-sum tests (Lehmann et al., 1975) reveal that the basic system outperforms variants with no hidden layers in the edge classifier (W=339; p<.001) or the label classifier (W=307; p<.01). Using a diagonal tensor $\mathbf{U}$ in the unlabeled parser also significantly hurts performance (W=388; p<.001), likely being too underpowered. While the other variations (especially the unfactorized and ReLU systems) appeared to make a difference during hyperparameter tuning, they were not significant here.

The improved performance of deeper systems (replicating Dozat and Manning (2017)) likely justifies the added complexity. On the other hand, the choice between biaffine and bilinear classifiers comes down largely to aesthetics. This is perhaps unsurprising since the change from biaffine to bilinear represents only a small decrease in overall power. Unusually, using no nonlinearity in the hidden layers in Eqs. (7–6) works as well as ReLU—in fact, using ReLU in the unlabeled parser marginally reduced performance (W=269; p=.063). Overall, the parser displayed considerable invariance to architecture changes. Since our system is significantly larger and more heavily regularized than the systems we compare against, this suggests that unglamorous, low-level hyperparameters—such as hidden sizes and dropout rates—are more critical to system performance than high-level architecture enhancements.

5 Discussion

We minimally extended a simple syntactic dependency parser to produce graph-structured dependencies. Without any further augmentations, our carefully-tuned system achieves state-of-the-art performance, highlighting the importance of finding the best hyperparameter configuration (and by extention, building fast systems that can be trained quickly). Additionally, we can see that a multitask system relying on a complex decoding algorithm to prune away invalid graph structures isn't necessary for achieving the level of parsing performance a simple system can achieve (though it could push performance even higher). We also find easier or independently motivated ways to improve accuracy—taking advantage of provided lemma or subtoken information provides a boost comparable to one found by drastically increasing system complexity.

Further, we observe a high-performing graph-based parser can be adapted to different types of dependency graphs (projective tree, non-projective tree, directed graph) with only small changes without obviously hurting accuracy. By contrast, transition-based parsers—which were originally designed for parsing projective constituency trees (Nivre, 2003; Aho and Ullman, 1972)—require whole new transition sets or even data structures to generate arbitrary graphs. We feel that this points to graph-based parsers being the most natural way to produce dependency graphs with different structural restrictions.

References

Alfred V Aho and Jeffrey D Ullman. 1972. *The theory of parsing, translation, and compiling*, volume 1. Prentice Hall.

Mariana SC Almeida and André FT Martins. 2015. Lisbon: Evaluating turbosemanticparser on multiple languages and out-of-domain data. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. pages 970–973.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. 2016. Training with exploration improves a greedy stack-LSTM parser. *Proceedings of the conference on empirical methods in natural language processing* .

Marie Candito, Bruno Guillaume, Guy Perrier, and Djamé Seddah. 2017. Enhanced ud dependencies with neutralized diathesis alternation. In *Depling 2017-Fourth International Conference on Dependency Linguistics*.

Jinho D Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 1052–1062.

Hal Daumé III. 2007. Frustratingly easy domain adaptation. *ACL 2007* page 256.

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

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. *ICLR 2017* .

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *CoNLL* pages 20–30.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. pages 927–931.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *Proceedings of the conference on empirical methods in natural language processing* .

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank. a dynamically annotated treebank of the wall street journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*. pages 85–96.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning* .

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational linguistics* 28(3):245–288.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondrej Bojar, Silvie Cinková, Eva Fucíková, Marie Mikulová, Petr Pajas, Jan Popelka, et al. 2012. Announcing prague czech-english dependency treebank 2.0. In *LREC*. pages 3153–3160.

Young-Bum Kim, Karl Stratos, and Ruhi Sarikaya. 2016. Frustratingly easy neural domain adaptation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. pages 387–396.

Erich Leo Lehmann, HJM D'Abrera, et al. 1975. *Nonparametrics*. Holden-Day.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. *ACL* .

André FT Martins, Noah A Smith, Pedro MQ Aguiar, and Mário AT Figueiredo. 2011. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 238–249.

Yusuke Miyao and Jun'ichi Tsujii. 2004. Deep linguistic analysis for the accurate identification of predicate-argument relations. In *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, page 1392.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation*. pages 915–926.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In

Proceedings of the 8th International Workshop on Semantic Evaluation. pages 63–72.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based mrs banking. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*. pages 1250–1255.

Hao Peng, Sam Thomson, and Noah A Smith. 2017. Deep multitask learning for semantic dependency parsing. In *ACL*. volume 1, pages 2037–2048.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)* 12.

Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. *arXiv preprint arXiv:1702.03196* .

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Sebastian Schuster, Éric Villemonte de la Clergerie, Marie Candito, Benoît Sagot, Christopher D. Manning, and Djamé Seddah. 2017. Paris and Stanford at EPE 2017: Downstream evaluation of graph-based dependency representations. In *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation (EPE 2017)*.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing. *AAAI* .