DEEP BIAFFINE ATTENTION FOR NEURAL DEPENDENCY PARSING

Timothy Dozat
Stanford University
tdozat@stanford.edu

Christopher D. Manning
Stanford University
manning@stanford.edu

ABSTRACT

This paper builds off recent work from Kiperwasser & Goldberg (2016) using neural attention in a simple graph-based dependency parser. We use a larger but more thoroughly regularized parser than other recent BiLSTM-based approaches, with biaffine classifiers to predict arcs and labels. Our parser gets state of the art or near state of the art performance on standard treebanks for six different languages, achieving 95.7% UAS and 94.1% LAS on the most popular English PTB dataset. This makes it the highest-performing graph-based parser on this benchmark—outperforming Kiperwasser & Goldberg (2016) by 1.8% and 2.2%—and comparable to the highest performing transition-based parser (Kuncoro et al., 2016), which achieves 95.8% UAS and 94.6% LAS. We also show which hyperparameter choices had a significant effect on parsing accuracy, allowing us to achieve large gains over other graph-based approaches.

1 INTRODUCTION

Dependency parsers—which annotate sentences in a way designed to be easy for humans and computers alike to understand—have been found to be extremely useful for a sizable number of NLP tasks, especially those involving natural language understanding in some way (Bowman et al., 2016; Angeli et al., 2015; Levy & Goldberg, 2014; Toutanova et al., 2016; Parikh et al., 2015). However, frequent incorrect parses can severely inhibit final performance, so improving the quality of dependency parsers is needed for the improvement and success of these downstream tasks.

The current state-of-the-art transition-based neural dependency parser (Kuncoro et al., 2016) substantially outperforms many much simpler neural graph-based parsers. We modify the neural graph-based approach first proposed by Kiperwasser & Goldberg (2016) in a few ways to achieve competitive performance: we build a network that's larger but uses more regularization; we replace the traditional MLP-based attention mechanism and affine label classifier with biaffine ones; and rather than using the top recurrent states of the LSTM in the biaffine transformations, we first put them through MLP operations that reduce their dimensionality. Furthermore, we compare models trained with different architectures and hyperparameters to motivate our approach empirically. The resulting parser maintains most of the simplicity of neural graph-based approaches while approaching the performance of the SOTA transition-based one.

2 BACKGROUND AND RELATED WORK

Transition-based parsers—such as shift-reduce parsers—parse sentences from left to right, maintaining a "buffer" of words that have not yet been parsed and a "stack" of words whose head has not been seen or whose dependents have not all been fully parsed. At each step, transition-based parsers can access and manipulate the stack and buffer and assign arcs from one word to another. One can then train any multi-class machine learning classifier on features extracted from the stack, buffer, and previous arc actions in order to predict the next action.

Chen & Manning (2014) make the first successful attempt at incorporating deep learning into a transition-based dependency parser. At each step, the (feedforward) network assigns a probability to each action the parser can take based on word, tag, and label embeddings from certain words on the

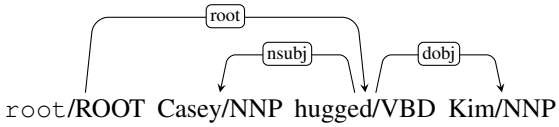

Figure 1: A dependency tree parse for *Casey hugged Kim*, including part-of-speech tags and a special `root` token. Directed edges (or arcs) with labels (or relations) connect the verb to the root and the arguments to the verb head.

stack and buffer. A number of other researchers have attempted to address some limitations of Chen & Manning's Chen & Manning parser by augmenting it with additional complexity: Weiss et al. (2015) and Andor et al. (2016) augment it with a beam search and a conditional random field loss objective to allow the parser to "undo" previous actions once it finds evidence that they may have been incorrect; and Dyer et al. (2015) and (Kuncoro et al., 2016) instead use LSTMs to represent the stack and buffer, getting state-of-the-art performance by building in a way of composing parsed phrases together.

Transition-based parsing processes a sentence sequentially to build up a parse tree one arc at a time. Consequently, these parsers don't use machine learning for directly predicting edges; they use it for predicting the operations of the transition algorithm. Graph-based parsers, by contrast, use machine learning to assign a weight or probability to each possible edge and then construct a maximum spaning tree (MST) from these weighted edges. Kiperwasser & Goldberg (2016) present a neural graph-based parser (in addition to a transition-based one) that uses the same kind of attention mechanism as Bahdanau et al. (2014) for machine translation. In Kiperwasser & Goldberg's 2016 model, the (bidirectional) LSTM's recurrent output vector for each word is concatenated with each possible head's recurrent vector, and the result is used as input to an MLP that scores each resulting arc. The predicted tree structure at training time is the one where each word depends on its highest-scoring head. Labels are generated analogously, with each word's recurrent output vector and its gold or predicted head word's recurrent vector being used in a multi-class MLP.

Similarly, Hashimoto et al. (2016) include a graph-based dependency parser in their multi-task neural model. In addition to training the model with multiple distinct objectives, they replace the traditional MLP-based attention mechanism that Kiperwasser & Goldberg (2016) use with a bilinear one (but still using an MLP label classifier). This makes it analogous to Luong et al.'s 2015 proposed attention mechanism for neural machine translation. Cheng et al. (2016) likewise propose a graph-based neural dependency parser, but in a way that attempts to circumvent the limitation of other neural graph-based parsers being unable to condition the scores of each possible arc on previous parsing decisions. In addition to having one bidirectional recurrent network that computes a recurrent hidden vector for each word, they have additional, unidirectional recurrent networks (left-to-right and right-to-left) that keep track of the probabilities of each previous arc, and use these together to predict the scores for the next arc.

3 Proposed Dependency Parser

3.1 Deep biaffine attention

We make a few modifications to the graph-based architectures of Kiperwasser & Goldberg (2016), Hashimoto et al. (2016), and Cheng et al. (2016), shown in Figure 2: we use biaffine attention instead of bilinear or traditional MLP-based attention; we use a biaffine dependency label classifier; and we apply dimension-reducing MLPs to each recurrent output vector $\mathbf{r}_i$ before applying the biaffine transformation.[1] The choice of biaffine rather than bilinear or MLP mechanisms makes the classifiers in our model analogous to traditional affine classifiers, which use an affine transformation over a single LSTM output state $\mathbf{r}_i$ (or other vector input) to predict the vector of scores $\mathbf{s}_i$ for all classes (1). We can think of the proposed biaffine attention mechanism as being a traditional affine

[1] In this paper we follow the convention of using lowercase italic letters for scalars and indices, lowercase bold letters for vectors, uppercase italic letters for matrices, uppercase bold letters for higher order tensors. We also maintain this notation when indexing; so row i of matrix R would be represented as $\mathbf{r}_i$.

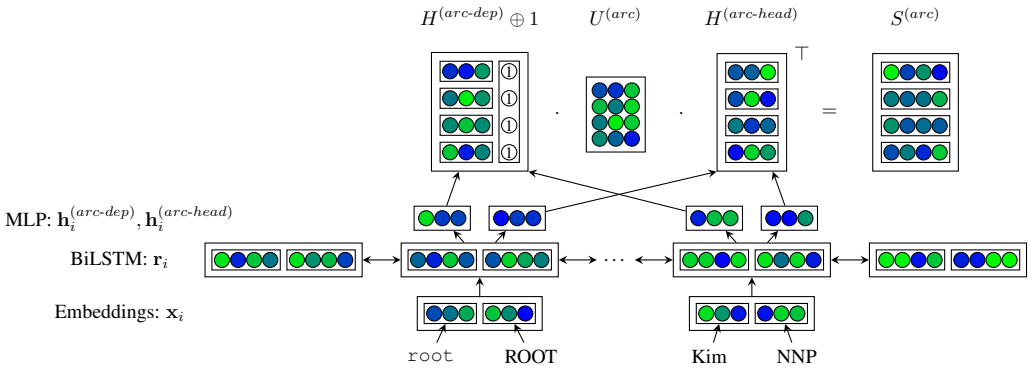

Figure 2: BiLSTM with deep biaffine attention to score each possible head for each dependent, applied to the sentence "Casey hugged Kim". We reverse the order of the biaffine transformation here for clarity.

classifier, but using a $(d \times d)$ linear transformation of the stacked LSTM output $RU^{(1)}$ in place of the weight matrix W and a $(d \times 1)$ transformation $R\mathbf{u}^{(2)}$ for the bias term $\mathbf{b}$ (2).

$$\mathbf{s}_i = W\mathbf{r}_i + \mathbf{b} \qquad \text{Fixed-class affine classifier} \qquad (1)$$

$$\mathbf{s}_i^{(arc)} = \left(RU^{(1)}\right)\mathbf{r}_i + \left(R\mathbf{u}^{(2)}\right) \qquad \text{Variable-class biaffine classifier} \qquad (2)$$

In addition to being arguably simpler than the MLP-based approach (involving one bilinear layer rather than two linear layers and a nonlinearity), this has the conceptual advantage of directly modeling both the prior probability of a word j receiving any dependents in the term $\mathbf{r}_j^\top \mathbf{u}^{(2)}$ and the likelihood of j receiving a specific dependent i in the term $\mathbf{r}_j^\top U^{(1)}\mathbf{r}_i$. Analogously, we also use a biaffine classifier to predict dependency labels given the gold or predicted head y_i (3).

$$\mathbf{s}_i^{(label)} = \mathbf{r}_{y_i}^\top \mathbf{U}^{(1)}\mathbf{r}_i + (\mathbf{r}_{y_i} \oplus \mathbf{r}_i)^\top U^{(2)} + \mathbf{b} \qquad \text{Fixed-class biaffine classifier} \qquad (3)$$

This likewise directly models each of the prior probability of each class, the likelihood of a class given just word i (how probable a word is to take a particular label), the likelihood of a class given just the head word y_i (how probable a word is to take dependents with a particular label), and the likelihood of a class given both word i and its head (how probable a word is to take a particular label given that word's head).

Applying smaller MLPs to the recurrent output states before the biaffine classifier has the advantage of stripping away information not relevant to the current decision. That is, every top recurrent state $\mathbf{r}_i$ will need to carry enough information to identify word i's head, find all its dependents, exclude all its non-dependents, assign itself the correct label, and assign all its dependents their correct labels, as well as transfer any relevant information to the recurrent states of words before and after it. Thus $\mathbf{r}_i$ necessarily contains significantly more information than is needed to compute any individual score, and training on this superfluous information needlessly reduces parsing speed and increases the risk of overfitting. Reducing dimensionality and applying a nonlinearity (4 - 6) addresses both of these problems. We call this a *deep* bilinear attention mechanism, as opposed to *shallow* bilinear attention, which uses the recurrent states directly.

$$\mathbf{h}_i^{(arc\text{-}dep)} = \text{MLP}^{(arc\text{-}dep)}(\mathbf{r}_i) \qquad (4)$$

$$\mathbf{h}_j^{(arc\text{-}head)} = \text{MLP}^{(arc\text{-}head)}(\mathbf{r}_j) \qquad (5)$$

$$\mathbf{s}_i^{(arc)} = H^{(arc\text{-}head)}U^{(1)}\mathbf{h}_i^{(arc\text{-}dep)} + H^{(arc\text{-}head)}\mathbf{u}^{(2)} \qquad (6)$$

We apply MLPs to the recurrent states before using them in the label classifier as well. As with other graph-based models, the predicted tree at training time is the one where each word is a dependent of its highest scoring head (although at test time we ensure that the parse is a well-formed tree via the MST algorithm).

3.2 HYPERPARAMETER CONFIGURATION

Param	Value	Param	Value
Embedding size	100	Embedding dropout	33%
LSTM size	400	LSTM dropout	33%
Arc MLP size	500	Arc MLP dropout	33%
Label MLP size	100	Label MLP dropout	33%
LSTM depth	3	MLP depth	1
α	$2e^{-3}$	β_1,β_2	.9
Annealing	$.75^{\frac{t}{5000}}$	t_{max}	50,000

Table 1: Model hyperparameters

Aside from architectural differences between ours and the other graph-based parsers, we make a number of hyperparameter choices that allow us to outperform theirs, laid out in Table 1. We use 100-dimensional uncased word vectors[2] and POS tag vectors; three BiLSTM layers (400 dimensions in each direction); and 500- and 100-dimensional ReLU MLP layers. We also apply dropout at every stage of the model: we drop words and tags (independently); we drop nodes in the LSTM layers (input and recurrent connections), applying the same dropout mask at every recurrent timestep (cf. the Bayesian dropout of Gal & Ghahramani (2015)); and we drop nodes in the MLP layers and classifiers, likewise applying the same dropout mask at every timestep. We optimize the network with annealed Adam (Kingma & Ba, 2014) for about 50,000 steps, rounded up to the nearest epoch.

4 EXPERIMENTS & RESULTS

4.1 DATASETS

We show test results for the proposed model on the English Penn Treebank, converted into Stanford Dependencies using both version 3.3.0 and version 3.5.0 of the Stanford Dependency converter (PTB-SD 3.3.0 and PTB-SD 3.5.0); the Chinese Penn Treebank; and the CoNLL 09 shared task dataset,[3] following standard practices for each dataset. We omit punctuation from evaluation only for the PTB-SD and CTB. For the English PTB-SD datasets, we use POS tags generated from the Stanford POS tagger (Toutanova et al., 2003); for the Chinese PTB dataset we use gold tags; and for the CoNLL 09 dataset we use the provided predicted tags. Our hyperparameter search was done with the PTB-SD 3.5.0 validation dataset in order to minimize overfitting to the more popular PTB-SD 3.3.0 benchmark, and in our hyperparameter analysis in the following section we report performance on the PTB-SD 3.5.0 test set, shown in Tables 2 and 3.

4.2 HYPERPARAMETER CHOICES

4.2.1 ATTENTION MECHANISM

We examined the effect of different classifier architectures on accuracy and performance. What we see is that the deep bilinear model outperforms the others with respect to both speed and accuracy. The model with shallow bilinear arc and label classifiers gets the same unlabeled performance as the deep model with the same settings, but because the label classifier is much larger ($(801 \times c \times 801)$ as opposed to $(101 \times c \times 101)$), it runs much slower and overfits. One way to decrease this overfitting is by increasing the MLP dropout, but that of course doesn't change parsing speed; another way is to decrease the recurrent size to 300, but this hinders unlabeled accuracy without increasing parsing speed up to the same levels as our deeper model. We also implemented the MLP-based approach to attention and classification used in Kiperwasser & Goldberg (2016).[4] We found this version to

[2]We compute a "trained" embedding matrix composed of words that occur at least twice in the training dataset and add these embeddings to their corresponding pretrained embeddings. Any words that don't occur in either embedding matrix are replaced with a separate OOV token.

[3]We exclude the Japanese dataset from our evaluation because we do not have access to it.

[4]In the version of TensorFlow we used, the model's memory requirements during training exceeded the available memory on a single GPU when default settings were used, so we reduced the MLP hidden size to 200

Classifier				Size			
Model	**UAS**	**LAS**	**Sents/sec**	**Model**	**UAS**	**LAS**	**Sents/sec**
Deep	**95.75**	**94.22**	**410.91**	3 layers, 400d	95.75	94.22	410.91
Shallow	95.74	94.00*	298.99	3 layers, 300d	95.82	**94.24**	460.01
Shallow, 50% drop	95.73	94.05*	300.04	3 layers, 200d	95.55*	93.89*	469.45
Shallow, 300d	95.63*	93.86*	373.24	2 layers, 400d	95.62*	93.98*	**497.99**
MLP	95.53*	93.91*	367.44	4 layers, 400d	**95.83**	94.22	362.09

Recurrent Cell			
Model	**UAS**	**LAS**	**Sents/sec**
LSTM	**95.75**	**94.22**	410.91
GRU	93.18*	91.08*	435.32
Cif-LSTM	95.67	94.06*	**463.25**

Table 2: Test accuracy and speed on PTB-SD 3.5.0. Statistically significant differences are marked with an asterisk.

Input Dropout			Adam		
Model	**UAS**	**LAS**	**Model**	**UAS**	**LAS**
Default	**95.75**	**94.22**	$\beta_2 = .9$	**95.75**	**94.22**
No word dropout	95.74	94.08*	$\beta_2 = .999$	95.53*	93.91*
No tag dropout	95.28*	93.60*			
No tags	95.77	93.91*			

Table 3: Test Accuracy on PTB-SD 3.5.0. Statistically significant differences are marked with an asterisk.

likewise be somewhat slower and significantly underperform the deep biaffine approach in both labeled and unlabeled accuracy.

4.2.2 NETWORK SIZE

We also examine more closely how network size influences speed and accuracy. In Kiperwasser & Goldberg's 2016 model, the network uses 2 layers of 125-dimensional bidirectional LSTMs; in Hashimoto et al.'s 2016 model, it has one layer of 100-dimensional bidirectional LSTMs dedicated to parsing (two lower layers are also trained on other objectives); and Cheng et al.'s 2016 model has one layer of 368-dimensional GRU cells. We find that using three or four layers gets significantly better performance than two layers, and increasing the LSTM sizes from 200 to 300 or 400 dimensions likewise signficantly improves performance.[5]

4.2.3 RECURRENT CELL

GRU cells have been promoted as a faster and simpler alternative to LSTM cells, and are used in the approach of Cheng et al. (2016); however, in our model they drastically underperformed LSTM cells. We also implemented the coupled input-forget gate LSTM cells (Cif-LSTM) suggested by Greff et al. (2015),[6] finding that while the resulting model still slightly underperforms the more popular LSTM cells, the difference between the two is much smaller. Additionally, because the gate and candidate cell activations can be computed simultaneously with one matrix multiplication, the Cif-LSTM model is faster than the GRU version even though they have the same number of parameters. We hypothesize that the output gate in the Cif-LSTM model allows it to maintain a sparse recurrent output state, which helps it adapt to the high levels of dropout needed to prevent overfitting in a way that GRU cells are unable to do.

[5]The model with 400-dimensional recurrent states significantly outperforms the 300-dimensional one on the validation set, but not on the test set

[6]In addition to using a coupled input-forget gate, we remove the first tanh nonlinearity, which is no longer needed when using a coupled gate

Type	Model	English PTB-SD 3.3.0 UAS	English PTB-SD 3.3.0 LAS	Chinese PTB 5.1 UAS	Chinese PTB 5.1 LAS
Transition	Ballesteros et al. (2016)	93.56	91.42	87.65	86.21
	Andor et al. (2016)	94.61	92.79	–	–
	Kuncoro et al. (2016)	**95.8**	**94.6**	–	–
Graph	Kiperwasser & Goldberg (2016)	93.9	91.9	87.6	86.1
	Cheng et al. (2016)	94.10	91.49	88.1	85.7
	Hashimoto et al. (2016)	94.67	92.90	–	–
	Deep Biaffine	95.74	94.08	**89.30**	**88.23**

Table 4: Results on the English PTB and Chinese PTB parsing datasets

	Catalan		Chinese		Czech	
Model	**UAS**	**LAS**	**UAS**	**LAS**	**UAS**	**LAS**
Andor et al.	92.67	89.83	84.72	80.85	88.94	84.56
Deep Biaffine	**94.69**	**92.02**	**88.90**	**85.38**	**92.08**	**87.38**

	English		German		Spanish	
Model	**UAS**	**LAS**	**UAS**	**LAS**	**UAS**	**LAS**
Andor et al.	93.22	91.23	90.91	89.15	92.62	89.95
Deep Biaffine	**95.21**	**93.20**	**93.46**	**91.44**	**94.34**	**91.65**

Table 5: Results on the CoNLL '09 shared task datasets

4.2.4 EMBEDDING DROPOUT

Because we increase the parser's power, we also have to increase its regularization. In addition to using relatively extreme dropout in the recurrent and MLP layers mentioned in Table 1, we also regularize the input layer. We drop 33% of words and 33% of tags during training: when one is dropped the other is scaled by a factor of two to compensate, and when both are dropped together, the model simply gets an input of zeros. Models trained with only word or tag dropout but not both wind up signficantly overfitting, hindering label accuracy and—in the latter case—attachment accuracy. Interestingly, not using any tags at all actually results in better performance than using tags without dropout.

4.2.5 OPTIMIZER

We choose to optimize with Adam (Kingma & Ba, 2014), which (among other things) keeps a moving average of the L_2 norm of the gradient for each parameter throughout training and divides the gradient for each parameter by this moving average, ensuring that the magnitude of the gradients will on average be close to one. However, we find that the value for β_2 recommended by Kingma & Ba—which controls the decay rate for this moving average—is too high for this task (and we suspect more generally). When this value is very large, the magnitude of the current update is heavily influenced by the larger magnitude of gradients very far in the past, with the effect that the optimizer can't adapt quickly to recent changes in the model. Thus we find that setting β_2 to .9 instead of .999 makes a large positive impact on final performance.

4.3 RESULTS

Our model gets nearly the same UAS performance on PTB-SD 3.3.0 as the current SOTA model from Kuncoro et al. (2016) in spite of its substantially simpler architecture, and gets SOTA UAS performance on CTB 5.1[7] as well as SOTA performance on all CoNLL 09 languages. It is worth noting that the CoNLL 09 datasets contain many non-projective dependencies, which are difficult or impossible for transition-based—but not graph-based—parsers to predict. This may account for some of the large, consistent difference between our model and Andor et al.'s 2016 transition-based model applied to these datasets.

[7]We'd like to thank Zhiyang Teng for finding a bug in the original code that affected the CTB 5.1 dataset

Where our model appears to lag behind the SOTA model is in LAS, indicating one of a few possibilities. Firstly, it may be the result of inefficiencies or errors in the GloVe embeddings or POS tagger, in which case using alternative pretrained embeddings or a more accurate tagger might improve label classification. Secondly, the SOTA model is specifically designed to capture phrasal compositionality; so another possibility is that ours doesn't capture this compositionality as effectively, and that this results in a worse label score. Similarly, it may be the result of a more general limitation of graph-based parsers, which have access to less explicit syntactic information than transition-based parsers when making decisions. Addressing these latter two limitations would require a more innovative architecture than the relatively simple one used in current neural graph-based parsers.

5 Conclusion

In this paper we proposed using a modified version of bilinear attention in a neural dependency parser that increases parsing speed without hurting performance. We showed that our larger but more regularized network outperforms other neural graph-based parsers and gets comparable performance to the current SOTA transition-based parser. We also provided empirical motivation for the proposed architecture and configuration over similar ones in the existing literature. Future work will involve exploring ways of bridging the gap between labeled and unlabeled accuracy and augment the parser with a smarter way of handling out-of-vocabulary tokens for morphologically richer languages.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Association for Computational Linguistics*, 2016. URL `https://arxiv.org/abs/1603.06042`.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2014.

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

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. *ACL 2016*, 2016.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the conference on empirical methods in natural language processing*, pp. 740–750, 2014.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. *arXiv preprint arXiv:1608.02076*, 2016.

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

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning*, 2015.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. What do recurrent neural network grammars learn about syntax? *CoRR*, abs/1611.05774, 2016. URL `http://arxiv.org/abs/1611.05774`.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *ACL 2014*, pp. 302–308, 2014.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *Empirical Methods in Natural Language Processing*, 2015.

Ankur P Parikh, Hoifung Poon, and Kristina Toutanova. Grounded semantic parsing for complex knowledge extraction. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, pp. 756–766, 2015.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 173–180. Association for Computational Linguistics, 2003.

Kristina Toutanova, Xi Victoria Lin, and Wen-tau Yih. Compositional learning of embeddings for relation paths in knowledge bases and text. In *ACL*, 2016.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *Annual Meeting of the Association for Computational Linguistics*, 2015.