Making Dependency Labeling Simple, Fast and Accurate

Tianxiao Shen[1] **Tao Lei**[2] **Regina Barzilay**[2]

[1]The Institute for Theoretical Computer Science (ITCS)
Institute for Interdisciplinary Information Sciences
Tsinghua University

[2]MIT CSAIL

[1]shentianxiao0831@gmail.com
[2]{taolei, regina}@csail.mit.edu

Abstract

This work addresses the task of dependency labeling—assigning labels to an (unlabeled) dependency tree. We employ and extend a feature representation learning approach, optimizing it for both high speed and accuracy. We apply our labeling model on top of state-of-the-art parsers and evaluate its performance on standard benchmarks including the CoNLL-2009 and the English PTB datasets. Our model processes over 1,700 English sentences per second, which is 30 times faster than the sparse-feature method. It improves labeling accuracy over the outputs of top parsers, achieving the best LAS on 5 out of 7 datasets[1].

1 Introduction

Traditionally in dependency parsing, the tasks of finding the tree structure and labeling the dependency arcs are coupled in a joint achitecture. While it has potential to eliminate errors propogated through a separated procedure, joint decoding introduces other sources of issues that can also lead to non-optimal labeling assignments. One of the issues arises from inexact algorithms adopted in order to solve the hard joint search problem. For instance, many parsers (Nivre et al., 2007; Titov and Henderson, 2007; Zhang et al., 2013; Dyer et al., 2015; Weiss et al., 2015) adopt greedy decoding such as beam search, which may prune away the correct labeling hypothesis in an early decoding stage. Another issue is caused by the absence of rich label features. Adding dependency labels to the combinatorial space significantly slows down the search procedure. As a trade-off, many parsers such as MSTParser, TurboParser and RBGParser (McDonald et al., 2005; Martins et al., 2010; Zhang et al., 2014) incorporate only single-arc label features to reduce the processing time. This restriction greatly limits the labeling accuracy.

In this work, we explore an alternative approach where the dependency labeling is applied as a separate procedure, alleviating the issues described above. The potential of this approach has been explored in early work. For instance, McDonald et al. (2006) applied a separate labeling step on top of the first-order MSTParser. The benefit of such approach is two-fold. First, finding the optimal labeling assignment (once the tree structure is produced) can be solved via an *exact* dynamic programming algorithm. Second, it becomes relatively cheap to add *rich label features* given a fixed tree, and the exact algorithm still applies when high-order label features are included. However, due to performance issues, such approach has not been adopted by the top performing parsers. In this work, we show that the labeling procedure, when optimized with recent advanced techniques in parsing, can achieve very high speed and accuracy.

Specifically, our approach employs the recent distributional representation learning technique for parsing. We apply and extend the low-rank tensor factorization method (Lei et al., 2014) to the second-order case to learn a joint scoring function over grand-head, head, modifier and their labels. Unlike the prior work which additionally requires

[1]Our code is available at https://github.com/shentianxiao/RBGParser/tree/labeling.

Proceedings of NAACL-HLT 2016, pages 1089–1094,
San Diego, California, June 12-17, 2016.

traditional sparse features to achieve state-of-the-art performance, our extention alone delivers the same level of accuracy, while being substantially faster. As a consequence, the labeling model can be applied either as a refinement (re-labeling) step on top of existing parsers with negligible cost of computation, or as a part of a decoupled procedure to simplify and speed up the dependency parsing decoding.

We evaluate on all datasets in the CoNLL-2009 shared task as well as the English Penn Treebank dataset, applying our labeling model on top of state-of-the-art dependency parsers. Our labeling model processes over 1,700 English sentences per second, which is 30 times faster than the sparse-feature method. As a refinement (re-labeling) model, it achieves the best LAS on 5 out of 7 datasets.

2 Method

2.1 Task Formulation

Given an unlabeled dependency parsing tree $\mathbf{y}$ of sentence $\mathbf{x}$, where $\mathbf{y}$ can be obtained using existing (non-labeling) parsers, we classify each head-modifier dependency arc $h \rightarrow m \in \mathbf{y}$ with a particular label $l_{h\rightarrow m}$. Let $\mathbf{l} = \bigcup_{h\rightarrow m\in\mathbf{y}}\{l_{h\rightarrow m}\}$, our goal is to find the assignment with the highest score:

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} \ \mathrm{S}(\mathbf{x}, \mathbf{y}, \mathbf{l})$$

For simplicity, we omit $\mathbf{x}, \mathbf{y}$ in the following discussion, which remain the same during the labeling process. We assume that the score $\mathrm{S}(\mathbf{l})$ decomposes into a sum of local scores of single arcs or pairs of arcs (in the form of grand-head–head–modifier), i.e.

$$\mathrm{S}(\mathbf{l}) = \sum_{h\rightarrow m\in\mathbf{y}} \mathrm{s}_1(h \xrightarrow{q} m) + \sum_{g\rightarrow h\rightarrow m\in\mathbf{y}} \mathrm{s}_2(g \xrightarrow{p} h \xrightarrow{q} m)$$

where $p = l_{g\rightarrow h}$, $q = l_{h\rightarrow m}$.

Parameterizing the scoring function $\mathrm{s}_1(h \xrightarrow{q} m)$ and $\mathrm{s}_2(g \xrightarrow{p} h \xrightarrow{q} m)$ is a key challenge. We follow Lei et al. (2014) to learn dense representations of features, which have been shown to better generalize the scoring function.

2.2 Scoring

The representation-based approach requires little feature engineering. Concretely, let $\phi_g, \phi_h, \phi_m \in \mathbb{R}^n$ be the atomic feature vector of the grand-head, head and modifier word respectively, and

Unigram features:		
form	form-p	form-n
lemma	lemma-p	lemma-n
POS	POS-p	POS-n
morph	bias	
Bigram features:		
POS-p, POS	POS, POS-n	
form, POS	lemma, POS	
lemma, POS-p	lemma, POS-n	
Trigram features:		
POS-p, POS, POS-n		

Table 1: Word atomic features used by our model. *POS*, *form*, *lemma* and *morph* stand for the POS tag, word form, word lemma and morphology features respectively. The suffix *-p* refers to the previous token, and *-n* refers to the next.

$\phi_{g\rightarrow h,p}, \phi_{h\rightarrow m,q} \in \mathbb{R}^d$ be the atomic feature vector of the two dependency arcs respectively. It is easy to define and compute these vectors. For instance, ϕ_g (as well as ϕ_h and ϕ_m) can incorperate binary features which indicate the word and POS tag of the current token (and its local context), while $\phi_{g\rightarrow h,p}$ (and $\phi_{h\rightarrow m,q}$) can indicate the label, direction and length of the arc between the two words.

The scores of the arcs are computed by (1) projecting the atomic vectors into low-dimensional spaces; and (2) summing up the element-wise products of the resulting dense vectors:

$$\mathrm{s}_1(h \xrightarrow{q} m) = \sum_{i=1}^{r_1} [U_1\phi_h]_i [V_1\phi_m]_i [W_1\phi_{h\rightarrow m,q}]_i$$

where r_1 is a hyper-parameter denoting the dimension after projection, and $U_1, V_1 \in \mathbb{R}^{r_1\times n}$, $W_1 \in \mathbb{R}^{r_1\times d}$ are projection matrices to be learned.

The above formulation can be shown equivalent to factorizing a huge score table $T_1(\cdot,\cdot,\cdot)$ into the product of three matrices U_1, V_1 and W_1, where T_1 is a 3-way array (tensor) storing feature weights of all possible features involving three components—the head, modifier and the arc between the two. Accordingly, the formula to calculate $\mathrm{s}_1(\cdot)$ is equivalent to summing up all feature weights (from T_1) over the structure $h \xrightarrow{q} m$.[2]

We depart from the prior work in the following aspects. First, we naturally extend the factorization approach to score second-order structures of grand-

[2]We refer readers to the original work (Lei et al., 2014) for the derivation and more details.

head, head and modifier,

$$\mathrm{s}_2(g \xrightarrow{p} h \xrightarrow{q} m) = \sum_{i=1}^{r_2} [U_2\phi_g]_i [V_2\phi_h]_i [W_2\phi_m]_i \\ [X_2\phi_{g\to h,p}]_i [Y_2\phi_{h\to m,q}]_i$$

Here r_2 is a hyper-parameter denoting the dimension, and $U_2, V_2, W_2 \in \mathbb{R}^{r_2\times n}$, $X_2, Y_2 \in \mathbb{R}^{r_2\times d}$ are additional parameter matrices to be learned. Second, in order to achieve state-of-the-art parsing accuracy, prior work combines the single-arc score $\mathrm{s}_1(h \xrightarrow{q} m)$ with an extensive set of sparse features which go beyond single-arc structures. However, we find this combination is a huge impediment to decoding speed. Since our extention already captures high-order structures, it readily delivers state-of-the-art accuracy without the combination. This change results in a speed-up of an order of magnitude (see section 2.4 for a further discussion).

2.3 Viterbi Labeling

We use a dynamic programming algorithm to find the labeling assignment with the highest score. Suppose h is any node apart from the root, and g is h's parent. Let $\mathrm{f}(h,p)$ denote the highest score of subtree h with $l_{g\to h}$ fixed to be p. Then we can compute $\mathrm{f}(\cdot,\cdot)$ using a bottom-up method, from leaves to the root, by transition function

$$\mathrm{f}(h,p) = \sum_{h\to m\in\mathbf{y}} \max_q \Big\{ \mathrm{f}(m,q) + \mathrm{s}_1(h \xrightarrow{q} m) \\ + \mathrm{s}_2(g \xrightarrow{p} h \xrightarrow{q} m) \Big\}$$

And the highest score of the whole tree is

$$\mathrm{f}(root) = \sum_{root\to m\in\mathbf{y}} \max_q \mathrm{f}(m,q) + \mathrm{s}_1(root \xrightarrow{q} m)$$

Once we get $\mathrm{f}(\cdot,\cdot)$, we can determine the labels backward, in a top-down manner. The time complexity of our algorithm is $\mathrm{O}(NL^2 \cdot T)$, where N is the number of words in a sentence, L is the number of total labels, and T is the time of computing features and scores.

2.4 Speed-up

In this section, we discuss two simple but effective strategies to speed up the labeling procedure.

Pruning We prune unlikely labels by simply exploiting the part-of-speech (POS) tags of the head and the modifier. Specifically, let $\mathbf{1}(\mathrm{pos}_h, \mathrm{pos}_m, l)$ denote whether there is an arc $h \xrightarrow{l} m$ in the training data such that h's POS tag is pos_h and m's POS tag is pos_m. In the labeling process, we only consider the possible labels that occur with the corresponding POS tags. Let K be the average number of possible labels per arc, then the time complexity is dropped to $\mathrm{O}(NK^2 \cdot T)$ approximately. In practice, $K \approx L/4$. Hence this pruning step makes our labeler 16 times faster.

Using Representation-based Scoring Only The time to compute scores, i.e. T, consists of building the features and fetching the corresponding feature weights. For traditional methods, this requires enumerating feature templates, constructing feature ID and searching the feature weight in a look-up table. For representation-based scoring, the dense word representations (e.g. $U_1\phi_h$) can be pre-computed, and the scores are obtained by simple inner products of small vectors. We choose to use representation-based scoring only, therefore reducing the time to $\mathrm{O}(NK^2 \cdot (r_1 + r_2) + NT')$. In practice, we find the labeling process becomes about 30 times faster.

2.5 Learning

Let $\mathbf{D} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{l}_i)\}_{i=1}^M$ be the collection of M training samples. Our goal is to learn the values of the set of parameters $\Theta = \{U_1, V_1, W_1, U_2, V_2, W_2, X_2, Y_2\}$ based on $\mathbf{D}$. Following standard practice, we optimize the parameter values in an online maximum soft-margin framework, minimizing the structural hinge loss:

$$\mathrm{loss}(\Theta) = \max_{\hat{\mathbf{l}}} \Big\{ \mathrm{S}(\hat{\mathbf{l}}) + \|\mathbf{l}_i - \hat{\mathbf{l}}\|_1 \Big\} - \mathrm{S}(\mathbf{l}_i)$$

where $\|\mathbf{l}_i - \hat{\mathbf{l}}\|_1$ is the number of different labels between $\mathbf{l}_i$ and $\hat{\mathbf{l}}$. We adjust parameters Θ by $\Delta\Theta$ via passive-aggressive update:

$$\Delta\Theta = \max\left\{ C, \frac{\mathrm{loss}(\Theta)}{\|\delta\Theta\|^2} \right\} \cdot \delta\Theta$$

where $\delta\Theta = \frac{\mathrm{dloss}(\Theta)}{\mathrm{d}\Theta}$ denotes the derivatives and C is a regularization hyper-parameter controlling the maximum step size of each update.

To counteract over-fitting, we follow the common practice of averaging parameters over all iterations.

Model	Catalan		Chinese		Czech		English		German		Japanese		Spanish	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Best Shared Task	-	87.86	-	79.17	-	80.38	-	89.88	-	87.48	-	92.57	-	87.64
Bohnet (2010)	-	87.45	-	76.99	-	80.96	-	90.33	-	88.06	-	92.47	-	88.13
Zhang and McDonald (2014)	91.41	87.91	82.87	78.57	86.62	80.59	92.69	90.01	89.88	87.38	92.82	91.87	90.82	87.34
Alberti et al. (2015)	**92.31**	**89.17**	**83.34**	**79.50**	88.35	83.50	92.37	90.21	90.12	87.79	93.99	93.10	**91.71**	88.68
RBG	91.37	87.31	82.16	77.24	**88.88**	81.90	**92.75**	90.04	**90.88**	87.91	**94.18**	93.38	91.50	87.69
+ our labeling		88.29		77.12		**84.04**		**90.38**		**88.68**		**93.59**		**88.71**

Table 2: Pipelined Results on CoNLL-2009.

3 Results

Experimental Setup We test our model on the CoNLL-2009 shared task benchmark with 7 different languages as well as the English Penn Treebank dataset. Whenever available, we use the predicted POS tags, word lemmas and morphological information provided in the datasets as atomic features. Following standard practice, we use unlabeled attachment scores (UAS) and labeled attachment scores (LAS) as evaluation measure[3]. In order to compare with previous reported numbers, we exclude punctuations for PTB in the evaluation, and include punctuations for CoNLL-2009 for consistency.

We use RBGParser[4], a state-of-the-art graph-based parser for predicting dependency trees, and then apply our labeling model to obtain the dependency label assignments. To demonstrate the effectiveness of our model on other systems, we also apply it on two additional parsers – Stanford Neural Shift-reduce Parser (Chen and Manning, 2014)[5] and TurboParser (Martins et al., 2010)[6]. In all reported experiments, we use the default suggested settings to run these parsers. The hyper-parameters of our labeling model are set as follows: $r_1 = 50$, $r_2 = 30$, $C = 0.01$.

Labeling Performance To test the performance of our labeling method, we first train our model using the gold unlabeled dependency trees and evaluate the labeling accuray on CoNLL-2009. Table 3 presents the results. For comparison, we implement a combined system which adds a rich set of traditional, sparse features into the scoring function and jointly train the feature weights. As shown in the table, using our representation-based method alone is super fast, being 30 times faster than the implementation with traditional feature computation and able to process over 1,700 English sentences per second. It does not affect the LAS accuracy except for Chinese.

	Ours + Sparse Features		Ours only	
	LAS	Speed	LAS	Speed
Catalan	96.33	30	96.42	1070
Chinese	94.16	38	93.16	1304
Czech	95.54	71	95.60	2065
English	97.00	62	96.88	1751
German	96.93	113	96.89	1042
Japanese	98.92	305	98.95	2778
Spanish	96.53	43	96.68	1142

Table 3: LAS and parsing speed (sentence per second) based on unlabeled golden trees.

UAS		Labeled	Unlabeled
	RBG	93.48	93.33
LAS		Before	After
	Stanford NN	89.37	89.55
	Turbo	90.22	90.65
	RBG	91.00	91.43
Runtime		Joint	Two-step
	Stanford NN	4.4	3.3
	Turbo	182.1	119.4
	RBG	365.4	305.7

Table 4: Joint vs. Separate analysis on PTB.

PTB Results Table 4 shows the performance on the English PTB dataset. We use RBGParser to predict both labeled and unlabeled trees, and there is no significant difference between their UAS. This finding lays the foundation for a separate procedure, as the tree structure does not vary much comparing to the joint procedure, and we can exploit rich label features and sophisticated algorithms to improve the LAS. Our re-labeling model improves over the predictions generated by the three different parsers, ranging from 0.2% to 0.4% LAS gain. Moreover, the labeling procedure runs in only 1.5 seconds on the test set. If we use the existing parsers to only predict

[3] We use the official evaluation script from CoNLL-X: http://ilk.uvt.nl/conll/software.html

[4] https://github.com/taolei87/RBGParser

[5] http://nlp.stanford.edu/software/nndep.shtml

[6] http://www.cs.cmu.edu/~ark/TurboParser/

unlabeled trees, we also obtain speed improvement, even for the highly speed-optimzed Stanford Neural Parser.

CoNLL-2009 Results In Table 2, we compare our model with the best systems[7] of the CoNLL-2009 shared task, Bohnet (2010), Zhang and McDonald (2014) as well as the most recent neural network parser (Alberti et al., 2015). Despite the simplicity of the decoupled parsing procedure, our labeling model achieves LAS performance on par with the state-of-the-art neural network parser. Specifically, our model obtains the best LAS on 5 out of 7 languages, while the neural parser outperforms ours on Catalan and Chinese.

4 Conclusion

The most common method for dependency parsing couples the structure search and label search. We demonstrate that decoupling these two steps yields both computational gains and improvement in labeling accuracy. Specifically, we demonstrate that our labeling model can be used as a post-processing step to improve the accuracy of state-of-the-art parsers. Moreover, by employing dense feature representations and a simple pruning strategy, we can significantly speed up the labeling procedure and reduce the total decoding time of dependency parsing.

Acknowledgments

We thank Yuan Zhang for his help on the experiments, and Danqi Chen for answering questions about their parser. We also thank the MIT NLP group and the reviewers for their comments. This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61361136003.

References

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Bernd Bohnet. 2009. Efficient parsing of syntactic and semantic dependency structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*.

Wanxiang Che, Zhenghua Li, Yongqiang Li, Yuhang Guo, Bing Qin, and Ting Liu. 2009. Multilingual dependency-based syntactic and semantic parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Andrea Gesmundo, James Henderson, Paola Merlo, and Ivan Titov. 2009. A latent variable model of synchronous syntactic-semantic parsing for multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics.

[7] Winners include Bohnet (2009), Che et al. (2009), Gesmundo et al. (2009) and Ren et al. (2009).

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*.

Han Ren, Donghong Ji, Jing Wan, and Mingyao Zhang. 2009. Parsing syntactic and semantic dependencies for multiple languages with a pipeline approach. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of 10th International Conference on Parsing Technologies (IWPT)*.

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

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Hao Zhang, Liang Zhao, Kai Huang, and Ryan McDonald. 2013. Online learning for inexact hypergraph search. In *Proceedings of EMNLP*.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.