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 architecture. While it has potential to eliminate errors propagated 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 MST-Parser, 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.
traditional sparse features to achieve state-of-the-art performance, our extension 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 approach to score second-order structures of grand-head, head and modifier word respectively, and

\[ \phi_{g \rightarrow h, p, \phi_{h \rightarrow m, q}} \in \mathbb{R}^d \]

be the atomic feature vector of the two dependency arcs respectively. It is easy to define and compute these vectors. For instance, \( \phi_g \) (as well as \( \phi_h \) and \( \phi_m \)) can incorporate 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:

\[ s_1(h \nrightarrow m) = \sum_{i=1}^{r_1} [U_1 \phi_h]_i [V_1 \phi_m]_i [W_1 \phi_{h \nrightarrow m, q}]_i, \]

where \( r_1 \) is a hyper-parameter denoting the dimension after projection, and \( U_1, V_1, W_1 \in \mathbb{R}^{r_1 \times d} \) 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 \( s_1(\cdot, \cdot) \) is equivalent to summing up all feature weights (from \( T_1 \)) over the structure \( h \nrightarrow m \).

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

\[ s_2(g \xrightarrow{h} m) = \sum_{i=1}^{r_2} [U_2\phi_g], [V_2\phi_h], [W_2\phi_m], [X_2\phi_{g\rightarrow h}, p], [Y_2\phi_{h\rightarrow m}, q]_i \]

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 \( s_1(h \xrightarrow{g} m) \) 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 extension already captures high-order structures, it readily delivers state-of-the-art parsing accuracy, prior work combines the single-arc score 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 extension 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 \( f(h, p) \) denote the highest score of subtree \( h \) with \( l_{g\rightarrow h} \) fixed to be \( p \). Then we can compute \( f(\cdot, \cdot) \) using a bottom-up method, from leaves to the root, by transition function

\[
 f(h, p) = \max_{h \rightarrow m \in \gamma} \left\{ f(m, q) + s_1(h \xrightarrow{g} m) + s_2(g \xrightarrow{h} m) \right\}
\]

And the highest score of the whole tree is

\[
 f(\text{root}) = \max_{\text{root} \rightarrow m \in \gamma} \left\{ f(m, q) + s_1(\text{root} \xrightarrow{g} m) \right\}
\]

Once we get \( f(\cdot, \cdot) \), we can determine the labels backward, in a top-down manner. The time complexity of our algorithm is \( O(NL^2 \cdot T) \), 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 \( 1(\text{pos}_h, \text{pos}_m, l) \) denote whether there is an arc \( h \rightarrow m \) in the training data such that \( h \)'s POS tag is \( \text{pos}_h \) and \( m \)'s POS tag is \( \text{pos}_m \). 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 \( O(NK^2 \cdot T) \) 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 \( O(NK^2 \cdot (r_1 + r_2) + NT') \). In practice, we find the labeling process becomes about 30 times faster.

2.5 Learning

Let \( D = \{(x_i, y_i, l_i)\}_{i=1}^M \) 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 \( D \). Following standard practice, we optimize the parameter values in an online maximum soft-margin framework, minimizing the structural hinge loss:

\[
 \text{loss}(\Theta) = \max_i \left\{ S(l_i) + \|l_i - \hat{l}_i\|_1 \right\} - S(l_i)
\]

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

\[
 \Delta \Theta = \max \left\{ C, \frac{\text{loss}(\Theta)}{\|d\Theta\|^2} \right\} \cdot d\Theta
\]

where \( d\Theta = \frac{\text{dloss}(\Theta)}{\text{d}\Theta} \) 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.
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. 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, 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) and TurboParser (Martins et al., 2010). 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 accuracy 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.

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
unlabeled trees, we also obtain speed improvement, even for the highly speed-optimized 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


