Combining (Second-Order) Graph-Based and Headed-Span-Based Projective Dependency Parsing

Anonymous ACL submission

Abstract

Graph-based methods are popular in dependency parsing for decades, which decompose the score of a dependency tree into scores of dependency arcs. Recently, Yang and Tu (2021) propose a headed-span-based method that decomposes the score of a dependency tree into scores of headed spans. In this paper, we combine the two types of methods by considering both arc scores and headed-span scores, designing three scoring methods and the corresponding dynamic programming algorithms for joint inference. Experiments show the effectiveness of our proposed methods.

1 Introduction

Dependency parsing is an important task in natural language processing. There are many methods to tackle projective dependency parsing. In this paper, we focus on two kinds of global methods: graph-based methods and headed-span-based methods. They both score all parse trees and globally find the highest scoring tree. The difference between the two is how they score dependency trees. The simplest first-order graph-based methods (McDonald et al., 2005) decompose the score of a dependency tree into the scores of dependency arcs. Second-order graph-based methods (McDonald and Pereira, 2006) additionally score adjacent siblings, i.e., pairs of adjacent arcs with a shared head. There are many other higher-order graph-based methods (Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012). In contrast, headed-span-based method (Yang and Tu, 2021) decomposes the score of a dependency tree into the scores of headed-spans: in a projective tree, a headed-span is a word-span pair such that the subtree rooted at the word covers the span in the surface order. Figure 1 shows an example projective dependency parse tree with all its headed-spans.

Figure 1: An example projective dependency parse tree with all its headed-spans.

2 Model

2.1 Scoring

Given an input sentences $x_1, \ldots, x_n$, we add <bos> (beginning of sentence) and <eos> (end of sen-
tence) as \(x_0\) and \(x_{n+1}\). We apply mean-pooling at the last layer of BERT (Devlin et al., 2019) (i.e., averaging all subwords embeddings) to obtain the word-level embeddings \(e_i\). Then we feed \(e_0, \ldots, e_{n+1}\) into a three-layer BiLSTM (Hochreiter and Schmidhuber, 1997) network to get \(h_k = [f_k, b_{k+1}]\) and \(h_j = [f_j, b_{j+1}]\) to represent the \(k\)th boundary lying between \(x_k\) and \(x_{k+1}\), and use \(e_{i,j} = h_j - h_{i-1}\) to represent span \((i, j)\) from position \(i\) to \(j\) inclusive where \(1 \leq i \leq j \leq n\). Then we compute:

- \(s_{i,j,k}^\text{arc}\) (for arc \(x_i \rightarrow x_j\), used in all three models) by feeding \(c_i, c_j\) into a deep biaffine function (Dozat and Manning, 2017).
- \(s_{i,j,k}^\text{span}\) (for headed-span \((i, j, k)\) where \(x_k\) is the headword of span \((i, j)\), used in §3.1) by feeding \(e_{i,j}, c_k\) to a deep biaffine function.
- \(s_{i,j}^\text{left}\) and \(s_{i,j}^\text{right}\) (for headed-span \((i, j, k)\), used in §3.2 and §3.3) by feeding \(c_k, h_{i-1}\) and \(c_k, h_j\) into two different deep biaffine functions.
- \(s_{i,j,k}^\text{sub}\) (for adjacent siblings \(x_i \rightarrow \{x_j, x_k\}\) with \(k < j < i\) or \(i < j < k\), used in §3.3) by feeding \(e_i, c_k, c_j\) into a deep triaffine function (Zhang et al., 2020).

### 2.2 Learning

We decompose the training loss \(L\) into \(L_{\text{parse}} + L_{\text{label}}\). For \(L_{\text{parse}}\), we use the max-margin loss (Taskar et al., 2004):

\[
L_{\text{parse}} = \max(0, \max_{y' \neq y} (s(y') + \Delta(y', y) - s(y))
\]

where \(\Delta\) measures the difference between the incorrect tree and gold tree \(y\). Here we let \(\Delta\) to be the Hamming distance (i.e., the total number of mismatches of arcs, sibling pairs, and (split) headed-spans depending on the setting). We use the same label loss \(L_{\text{label}}\) in Dozat and Manning (2017).

### 3 Parsing

We use the parsing-as-deduction framework (Pereira and Warren, 1983) to describe the parsing algorithms of our proposed models.

1For some datasets requiring the use of gold POS tags, we additionally concatenate the POS tag embedding to obtain \(e_i\).

### 3.1 \(O(n^4)\) modified Eisner-Satta algorithm

In this case, we combine first-order graph-based parsing and headed-span-based parsing. The score of a dependency tree \(y\) is defined as:

\[
s(y) = \sum_{(x_i \rightarrow x_j) \in y} s_{i,j,k}^\text{arc} + \sum_{(l_i, r_i, x_i) \in y} s_{i,r,i}^\text{span}
\]

We design a dynamic programming algorithm adapted from the Eisner-Satta algorithm, which uses the hook trick to accelerate biaffine context-free parsing. The axiom items are

\[
\begin{array}{c}
\hline
\text{with initial score 0 and the deduction} \\
\hline
\end{array}
\]

With initial score 0 and the deduction rules are listed in Figure 2. Unlike the original Eisner-Satta algorithm, we distinguish between “finished” spans and “unfinished” spans. An “unfinished” span can absorb a child span to form a larger span, while in a “finished” span, the headword has already generated all its children, so it cannot expand anymore and corresponds to a headed-span for the given headword. By explicitly distinguishing between “unfinished” spans and “finished” spans, we can incorporate headed-span scores \(s_{i,r,i}^\text{span}\) into parsing via the newly introduced rule \(\text{FINISH}\). We then modify the rule \(L\)-\text{LINK} and \(R\)-\text{LINK} accordingly as only a “finished” span can be attached.

### 3.2 \(O(n^3)\) modified Eisner algorithm

In order to decrease the inference time complexity from \(O(n^4)\) to \(O(n^3)\), we decompose \(s_{i,r,i}^\text{span}\) into two terms:

\[
s(y) = \sum_{(x_i \rightarrow x_j) \in y} s_{i,j,k}^\text{arc} + \sum_{(l_i, r_i, x_i) \in y} (s_{i,j}^\text{left} + s_{i,j}^\text{right})
\]

and modify the Eisner algorithm accordingly. The axiom items are \(\bigtriangleup\) and \(\bigtriangledown\) with initial score 0 and the deduction rules are shown in the first two rows of Figure 3. Similar to the case in the previous subsection, the original Eisner algorithm does not distinguish between “finished” complete spans and “unfinished” complete spans. An “unfinished” complete span can absorb another complete span to form a larger incomplete span, while a “finished” complete span has no more child in the given direction and thus cannot expand anymore. We introduce new rules \(L\)-\text{FINISH} and \(R\)-\text{FINISH} to incorporate the left or right span boundary scores respectively, and adjust other rules accordingly.
### 3.3 \(O(n^3)\) modified second-order Eisner algorithm

We further enhance the model with adjacent sibling information:

\[
s(y) = \sum_{(x_i \rightarrow x_j) \in y} s^\text{arc}_{i,j} + \sum_{(x_i \rightarrow \{x_j, x_k\}) \in y} s^\text{sib}_{i,j,k} + \sum_{(l_i, r_i, x_i) \in y} (s^\text{left}_{l_i} + s^\text{right}_{r_i})
\]

where for each adjacent sibling part \(x_i \rightarrow \{x_j, x_k\}\), \(x_j\) and \(x_k\) are two adjacent dependents of \(x_i\).

Similarly, we modify the second-order extension of the Eisner algorithm (McDonald and Pereira, 2006) by distinguishing between “unfinished” and “finished” complete spans. The additional deductive rules for second-order parsing are shown in the last row of Figure 3 and the length of the “unfinished” complete span is forced to be 1 in the rule L-LINK and R-LINK.
Table 1: Labeled Attachment Score (LAS) on twelve languages in UD 2.2. We use ISO 639-1 codes to represent languages. † means reported by Yang and Tu (2021). MFVI2O: Wang and Tu (2020). Span: Yang and Tu (2021).

|                | bg   | ca   | cs   | de   | en   | es   | fr   | it   | nl   | no   | ro   | ru   | Avg  |
|----------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| **+BERT**      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Biaffine+MM†   | 90.30| 94.49| 92.65| 85.98| 91.13| 93.78| 91.77| 94.72| 91.04| 94.21| 87.24| 94.53| 91.82|
| Span           | 91.10| 94.46| 92.57| 85.87| **91.32**| 93.84| 91.69| 94.78| 91.65| 94.28| 87.48| 94.45| 91.96|
| 1O+Span        | 91.44| 94.54| 92.68| 85.75| 91.23| 93.84| 91.67| **94.97**| 91.81| 94.35| 87.17| 94.49| 91.99|
| 1O+Span+Headsplit | 91.46| 94.53| 92.63| 85.78| 91.25| 93.77| **91.91**| 94.88| 91.59| 94.18| 87.45| 94.47| 91.99|
| Biaffine+2O+MM | 91.58| 94.48| 92.69| 85.72| 91.28| 93.80| 91.89| 94.23| 91.77| 94.00| 87.24| 94.53| 91.82|
| 2O+Span+Headsplit | 91.82| 94.58| 92.59| 85.85| 91.28| **93.86**| 91.80| 94.75| 91.65| 94.28| 87.48| 94.45| 91.96|

For reference

<table>
<thead>
<tr>
<th></th>
<th>PTB</th>
<th></th>
<th>CTB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
<td>LAS</td>
</tr>
<tr>
<td><strong>+BERT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Biaffine+MM†</td>
<td>97.22</td>
<td>95.71</td>
<td>93.18</td>
<td>92.10</td>
</tr>
<tr>
<td>Span</td>
<td>97.24</td>
<td>95.73</td>
<td>93.33</td>
<td>92.30</td>
</tr>
<tr>
<td>1O+Span</td>
<td>97.26</td>
<td>95.68</td>
<td>93.56</td>
<td><strong>92.49</strong></td>
</tr>
<tr>
<td>1O+Span+Headsplit</td>
<td>97.30</td>
<td>95.77</td>
<td>93.46</td>
<td>92.42</td>
</tr>
<tr>
<td>Biaffine+2O+MM</td>
<td>97.28</td>
<td>95.73</td>
<td>93.42</td>
<td>92.34</td>
</tr>
<tr>
<td>2O+Span+Headsplit</td>
<td>97.23</td>
<td>95.69</td>
<td>93.57</td>
<td>92.47</td>
</tr>
</tbody>
</table>

For reference

<table>
<thead>
<tr>
<th></th>
<th>PTB</th>
<th></th>
<th>CTB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
<td>LAS</td>
</tr>
<tr>
<td><strong>+BERT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MFVI2O</td>
<td>91.30</td>
<td>93.60</td>
<td>92.09</td>
<td>82.00</td>
</tr>
<tr>
<td>HierPtr</td>
<td>97.01</td>
<td>95.48</td>
<td>92.65</td>
<td>91.47</td>
</tr>
<tr>
<td>HPSG♭</td>
<td>97.20</td>
<td>95.72</td>
<td>93.72</td>
<td><strong>93.57</strong></td>
</tr>
<tr>
<td>HPSG+LAL♭</td>
<td>97.20</td>
<td>95.72</td>
<td>93.72</td>
<td><strong>93.57</strong></td>
</tr>
</tbody>
</table>

Table 2: Results on PTB and CTB. ♭ denotes use of additional constituency tree data and thus not comparable to our work. † denotes results reported by Yang and Tu (2021). MFVI2O: Wang and Tu (2020). Span: Yang and Tu (2021).

4 Experiments

4.1 Setup

We conduct experiments on in Penn Treebank (PTB) 3.0 (Marcus et al., 1993), Chinese Treebank (CTB) 5.1 (Xue et al., 2005) and 12 languages on Universal Dependencies (UD) 2.2. We use the same data processing, evaluation methods, and hyperparameters as Yang and Tu (2021) for fair comparison and we refer readers to their paper for details due to the limit of space. We set the hidden size of the Triaffine function to 300 additionally. The reported results are averaged over three runs with different random seeds.

4.2 Main result

Table 1 and 2 show the results on UD, PTB and CTB respectively. We additionally reimplement Biaffine+2O+MM by replacing the TreeCRF loss of Zhang et al. (2020) with the max-margin loss for fair comparison. We refer to our proposed models as 1O+Span (§3.1), 1O+Span+Headsplit (§3.2), and 2O+Span+Headsplit (§3.3) respectively.

We draw the following observations:

- Second-order information is still helpful even with powerful encoders (i.e., BERT). Biaffine+2O+MM outperforms Biaffine+MM in almost all cases.
- Combining first-order graph-based and headed-span-based methods is effective. Both 1O+Span and 1O+Span+Headsplit beat Biaffine+MM and Span in almost all cases. However, when incorporating span information into the second-order model, the improvement is slight: only +0.04 average LAS on UD, +0.13 LAS on CTB, and worse performance (-0.04 LAS) on PTB. We speculate that the utility of adjacent sibling information and span information is overlapping.
- Decomposing the headed-span scores is effective. 1O+Span+Headsplit has a comparable performance to 1O+Span while manages to decrease the time complexity from $O(n^4)$ to $O(n^3)$. We speculate that powerful encoders mitigate the issue of independent scoring.

5 Conclusion

In this paper we have presented three scoring and decoding methods to combine graph-based and headed-span-based methods for projective dependency parsing. Experiments show the effectiveness of our proposed methods.
References


