Enhancing Structure-aware Encoder with Extremely Limited Data for Graph-based Dependency Parsing

Yuanhe Tian\textsuperscript{\textcopyright}, Yan Song\textsuperscript{†}, Fei Xia\textsuperscript{\textcopyright}
\textsuperscript{\textcopyright}University of Washington \textsuperscript{†}University of Science and Technology of China
\textsuperscript{\textcopyright}\{yhtian, fxia\}@uw.edu \textsuperscript{†}clksong@gmail.com

Abstract

Dependency parsing is an important fundamental natural language processing task which analyzes the syntactic structure of an input sentence by illustrating the syntactic relations between words. To improve dependency parsing, leveraging existing dependency parsers and extra data (e.g., through semi-supervised learning) has been demonstrated to be effective, even though the final parsers are trained on inaccurate (but massive) data. In this paper, we propose a frustratingly easy approach to improve graph-based dependency parsing, where a structure-aware encoder is pre-trained on auto-parsed data by predicting the word dependencies and then fine-tuned on gold dependency trees, which differs from the usual pre-training process that aims to predict the context words along dependency paths. Experimental results and analyses demonstrate the effectiveness and robustness of our approach to benefit from the data (even with noise) processed by different parsers, where our approach outperforms strong baselines under different settings with different dependency standards and model architectures used in pre-training and fine-tuning. More importantly, further analyses find that only 2K auto-parsed sentences are required to obtain improvement when pre-training vanilla BERT-large based parser without requiring extra parameters.\textsuperscript{1}

1 Introduction

Dependency parsing aims to produce the syntactic structure of a sentence by illustrating the syntactic relations between words, where the words with dependency relations are connected by directed and labeled arcs. It is an important fundamental natural language processing (NLP) task that is widely used to enhance downstream NLP tasks (Cai et al., 2009; Strubell et al., 2018; Huang and Carley, 2019; Zhang et al., 2019; Guo et al., 2019; Nie et al., 2020; Zhou et al., 2020b; Chen et al., 2020; Tian et al., 2022) such as coreference resolution, relation extraction, and sentiment analysis.

To produce the dependency structure of a sentence, the contextual information is of great importance to achieve good model performance. Thus, most recent studies (Dozat and Manning, 2017; Zhou and Zhao, 2019; Zhou et al., 2020a,b; Mrini et al., 2020; Zhang et al., 2021) leverage advanced encoders (e.g., bi-LSTM, Transformer (Vaswani et al., 2017)) to model the contextual information of the input and obtain outstanding performance. In addition, because leveraging different models to obtain better results is an important technique for many NLP tasks (Juraska et al., 2018; Kobayashi, 2018; Kuwabara et al., 2020; Qin et al., 2021), many previous studies apply this technique to dependency parsing to further improve model performance. Under this paradigm, many studies utilize semi-supervised methods (e.g., self-training) to benefit from auto-processed extra data which is used to extract useful features (Smith and Eisner, 2007; Koo et al., 2008; Bansal and Klein, 2011; Ma and Xia, 2013; Kiperwasser and Goldberg, 2015; Yu and Bohnet, 2017) or training data (Spreyer and Kuhn, 2009; Rybak and Wróblewska, 2018; Rotman and Reichart, 2019). However, since the auto-generated parse tree is not always accurate, semi-supervised methods need to handle the noise with care to achieve better performance (Søgaard and Rishøj, 2010; Chen et al., 2018). To address the noise issue, in this paper, we propose to apply pre-training and fine-tuning to enhance dependency parsing with the auto-parsed data generated by existing parsers. Although the effectiveness of pre-training and fine-tuning paradigm has been demonstrated to leverage extra data in many NLP tasks, it is still worth studying whether this paradigm works well for dependency
2 Training the Dependency Parser

In this study, we use neural graph-based dependency parsers, because they have achieved state-of-the-art performance on this task (Dozat and Manning, 2017; Zhou and Zhao, 2019; Mrini et al., 2020). Specifically, training our graph-based parser follows a two-stage procedure with pre-training and fine-tuning, where pre-training is performed on auto-parsed data with the same object to train a dependency parser, and the fine-tuning is conducted on the gold dependency trees with the pre-trained encoder (other modules in the final parser are freshly learned in fine-tuning).

Figure 1 shows the model architecture and the two-stage procedure to train our final dependency parser. In the following text, we firstly introduce the neural graph-based dependency parser and then illustrate the process to pre-train the structure-aware encoder.

2.1 Neural Graph-based Dependency Parser

Given the input sentence \( \mathcal{X} = x_1, x_2, \ldots, x_i, \ldots, x_n \) (\( x_i \) is the \( i \)-th word), conventional neural graph-based dependency parsers firstly obtain the hidden vector \( \mathbf{h}_i \) for each word \( x_i \) from the encoder. Then, based on \( \mathbf{h}_i \), for each word pair \((x_i, x_j)\), the decoder of the parser computes \( s^\text{arc}_{i,j} \) and \( s^\text{rel}_{i,j} \) indicating the score for the directional dependency connection (arc) between \( x_i \) and \( x_j \) and the score for the dependency relation type \( \text{rel} \in \mathcal{R} \) (\( \mathcal{R} \) is the dependency type set) between them, respectively.

Next, the parser applies the Eisner algorithm\(^3\) (Eisner, 1996) to all \( s^\text{arc}_{i,j} \) to predict the dependency tree \( \hat{T}_0 \) and assigns the connection between \( x_i \) and its

---

\(^3\)The Eisner algorithm is only applied in inference. In training, the parser is optimized by comparing \( s^\text{arc}_{i,j} \) and \( s^\text{rel}_{i,j} \) with the gold standards using the cross-entropy loss function.
head $x_j$ with the dependency type $\tilde{r}_{i,j}$ having the highest score $s_{i,j}^{rel}$.

It is worth noting that there are many ways to obtain the dependency arc scores $s_{i,j}^{arc}$ and the dependency relation scores $s_{i,j}^{rel}$. In doing so, bi-affine attentions (Dozat and Manning, 2017) (as illustrated in Figure 1(b)) is the most common and effective way to obtain $s_{i,j}^{arc}$ and $s_{i,j}^{rel}$. Specifically, for $s_{i,j}^{arc}$, it is computed by

$$h_{i}^{arc-d} = \text{MLP} \cdot h_{i}$$  \hspace{1cm} (1)$$

$$h_{j}^{arc-h} = \text{MLP} \cdot h_{j}$$  \hspace{1cm} (2)$$

$$s_{i,j}^{arc} = (h_{i}^{arc-d} \oplus [1])^\top \text{W}^{arc}(h_{j}^{arc-h} \oplus [1])$$  \hspace{1cm} (3)$$

where MLP $\text{arc-h}$ and MLP $\text{arc-d}$ denote multi-layer perceptrons for the head and dependent representations, respectively; $\text{W}^{arc}$ is a trainable matrix; $\oplus$ is the vector concatenation operation; $[1]$ is a one-dimensional unit vector which serves as a bias term for $h_{i}^{arc-d}$ and $h_{j}^{arc-h}$. Following the aforementioned process, $s_{i,j}^{rel}$ for a particular dependency type $rel \in \mathcal{R}$ is computed in a similar way.

2.2 Pre-training Structure-aware Encoder

To leverage existing parsers and unlabeled data, we replace the original encoder with a structure-aware encoder. The encoder is pre-trained with dependency trees generated from an existing parser, following the same but simplified procedure (as shown in Figure 1(a) without using bi-affine attentions) of training a parser. That is, in the pre-training procedure, we use

$$s_{i,j}^{arc} = h_{i}^\top \text{W}^{arc}h_{j}, \quad s_{i,j}^{rel} = \text{W}^{rel}(h_{i} \oplus h_{j})$$  \hspace{1cm} (4)$$

to compute the arc score $s_{i,j}^{arc}$ and the relation score vector $s_{i,j}^{rel}$ over all dependency relation types. Herein, each dimension of $s_{i,j}^{rel}$ corresponds to a particular dependency relation type in $\mathcal{R}$ and $\text{W}^{arc}$ and $\text{W}^{rel}$ denote two trainable matrices.

Once the model is pre-trained, we get rid of the $\text{W}^{arc}$ and $\text{W}^{rel}$ and combine the resulting encoder with a new randomly initialized bi-affine attention module to construct our final dependency parser (illustrated in Figure 1(b)) for fine-tuning.

Through pre-training, the encoder is able to learn dependency information from the auto-parsed data (with noise). Meanwhile, because the decoder (i.e., the bi-affine attentions) of the final parser is changed and randomly initialized without using the decoder parameters (i.e., $\text{W}^{arc}$ and $\text{W}^{rel}$ in Eq. (4)) obtained from pre-training, our final parser is able to optimize its parameters based on the gold standard trees. Therefore, by using the auto-parsed and the gold training data in different stages (i.e., pre-training and fine-tuning, respectively), the noise in the auto-parsed data is carefully addressed: errors learnt from the pre-training stage can be “fixed” in the fine-tuning stage. In contrast, many existing semi-supervised approaches train the final parser on the combination of auto-parsed and gold training data, which could be risky.

3 Experiments

3.1 Datasets

In the experiments, we use English Wiki (with 92M sentences and 2,380M tokens) as the raw data for pre-training. We follow previous studies (Dozat and Manning, 2017; Zhou and Zhao, 2019; Mrini et al., 2020) to use English Penn Treebank (PTB)\(^4\) (Marcus et al., 1993) converted by version 3.3.0 of the Stanford Dependency converter\(^5\) as the benchmark dataset, which is further split into train/dev/test sets. In addition, we use Brown corpus (Marcus et al., 1993) and the English Web Treebank of Universal Dependencies (UD)\(^6\) (Nivre et al., 2016). Herein, the dependency parses of Brown are obtained in the same process as PTB and the dependency parsing standard used in PTB and Brown is the Stanford typed dependencies (the Stanford standard) (De Marneffe and Manning, 2008); while UD follows a different standard named the UD dependency parsing standard\(^7\). The statistics (i.e., the number of sentences, tokens, and

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Sent. #</th>
<th>Token #</th>
<th>ASL</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTB</td>
<td>Train</td>
<td>40K</td>
<td>950K</td>
</tr>
<tr>
<td></td>
<td>Dev</td>
<td>2K</td>
<td>40K</td>
</tr>
<tr>
<td></td>
<td>Test</td>
<td>2K</td>
<td>57K</td>
</tr>
<tr>
<td>UD</td>
<td>Train</td>
<td>13K</td>
<td>205K</td>
</tr>
<tr>
<td></td>
<td>Dev</td>
<td>2K</td>
<td>25K</td>
</tr>
<tr>
<td></td>
<td>Test</td>
<td>2K</td>
<td>25K</td>
</tr>
<tr>
<td>Brown (Full)</td>
<td></td>
<td>24K</td>
<td>458K</td>
</tr>
<tr>
<td>English Wiki</td>
<td></td>
<td>92M</td>
<td>2,380M</td>
</tr>
</tbody>
</table>

Table 1: The number of sentences, tokens, and the averaged sentence length (ASL) of PTB, UD, Brown, and English Wiki used in our experiments.

\(^4\)https://catalog.ldc.upenn.edu/LDC99T42.
\(^5\)https://stanfordnlp.github.io/CoreNLP.
\(^6\)We use the version 2.9 of UD obtained from https://universaldependencies.org/
\(^7\)https://universaldependencies.org/u/overview/syntax.html
3.2 Obtaining the Auto-parsed Data

In the experiments, we propose to use existing NLP toolkits to obtain the auto-parsed Wiki data, because it not only allows us to benefit from existing toolkits, but also is a good approximate of real-world applications where we want to build a good parser with existing toolkits. In addition, given PTB is one of the most widely used benchmark datasets for English dependency parsing, we want the auto-parsed data to follow exactly the same dependency standard as PTB, so that we can explore the effect of our approach when there is no gap between the standards in the auto-parsed and training data. However, many well-known existing dependency parsers (e.g., Stanford CoreNLP Toolkit (SCT) (Manning et al., 2014) and SpaCy\(^8\)) follow a different standard.\(^9\) Therefore, in the experiments, we employ a parsing-conversion process to obtain the dependency trees: we first use Berkeley Neural Parser\(^10\) (Kitaev and Klein, 2018) to obtain the constituency trees of the Wiki text; then we convert them into dependency trees following the same process to obtain PTB (we denote this process as BNP-SD). Since the Berkeley Neural Parser is trained on the training set of PTB, this process ensures that the off-the-shelf dependency parser does not see the test data of PTB in training and the auto-parsed dependency trees follow the same dependency parsing standard as PTB.

\(^8\)https://spacy.io/

\(^9\)Specifically, the dependency parser of SCT is trained on UD and follows the UD dependency parsing standard; SpaCy is trained on OntoNotes 5 with the dependency trees converted from the corresponding constituency trees through ClearNLP, which does not exactly follow the Stanford standard.

\(^10\)We use the model named “benepar_en2”, which is downloaded from https://github.com/nikitakit/self-attentive-parser.

3.3 Settings

Table 2 summarizes the datasets used in pre-training, fine-tuning, and testing, where the dependency parsing standards for them are also illustrated in parentheses (SD and UD stand for the Stanford standard and the UD dependency parsing standard, respectively). Herein, we denote the experiments using Brown and UD in testing as cross-domain and cross-standard experiments, respectively, because Brown (for testing) and PTB (for fine-tuning) come from different domains whereas UD (for testing) and Wiki (for pre-training) use different dependency standards. Intuitively, the cross-standard setting with UD as the test set is most challenging as the auto-parsed Wiki data used for pre-training and the UD data used for fine-tuning and testing follow different dependency standards.

In addition, since our system design requires the final parser to use a new randomly initialized decoder before fine-tuning, it is interesting to explore the impact of the choice of the final parser decoder. Therefore, in addition to our final parser with bi-affine attentions (BF) following the architecture in Figure 1(b) (we denote the final parsers as “+BF”), we also try final parsers without BF and following the architecture in Figure 1(a) (we denote it as “-BF”). It is worth noting that, the architectures used in pre-training and fine-tuning are different under “+BF” whereas they are the same under “-BF” (the parser used in pre-training does not use BF). Intuitively, “+BF” setting is more challenging because the patterns learned by the encoder from pre-training may not fit into the new architecture (i.e., the BF module) in the final parser and thus result in noise in the final parser.

3.4 Implementation Details

Since pre-trained language models have achieved outstanding performance in many NLP tasks (Devlin et al., 2019; Wu et al., 2019; Yang et al., 2019; Raffel et al., 2019; Chen et al., 2020; Tian et al., 2019), we also use pre-trained language models to initialize our model. The pre-trained language model we use is 110M parameter BERT. 110M parameter BERT is trained on a massive data set and it has a very strong language understanding ability and rich language knowledge. Since the dependency parser is closely related to language understanding, we think that the pre-trained language model can provide a very helpful initial parameter. The pre-trained language model is fine-tuned using both the dataset and the dependency standard as our experiments. The hyper-parameters used in fine-tuning are presented in Table 3.

Table 2: The datasets used in pre-training, fine-tuning, and testing. The dependency standard used in the datasets is illustrated in parentheses with SD and UD referring to the Stanford dependency standard and the UD standard, respectively.

<table>
<thead>
<tr>
<th>Pre-training</th>
<th>Fine-tuning</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki (SD)</td>
<td>PTB Training (SD)</td>
<td>PTB Test (SD)</td>
</tr>
<tr>
<td></td>
<td>Brown (SD)</td>
<td>UD Training (UD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UD Test (UD)</td>
</tr>
</tbody>
</table>

Table 3: The hyper-parameters tested in tuning our models. The best ones used in our final experiments are highlighted in boldface.

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>5e-6, 1e-5, 3e-5</td>
</tr>
<tr>
<td>Warmup Rate</td>
<td>0.1, 0.2</td>
</tr>
<tr>
<td>Dropout Rate</td>
<td>0.33</td>
</tr>
<tr>
<td>Batch Size</td>
<td>16, 32</td>
</tr>
</tbody>
</table>
Table 4: The performance of different dependency parsers obtained after pre-training, without fine-tuning. We only report UAS (which does not evaluate the relation types associated with the dependency connections) on UD because UD and the auto-parsed data use different dependency parsing standard.

<table>
<thead>
<tr>
<th>Models</th>
<th>PTB</th>
<th>Brown</th>
<th>UD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
</tr>
<tr>
<td>BERT-base</td>
<td>96.43</td>
<td>94.74</td>
<td>94.25</td>
</tr>
<tr>
<td>BERT-large</td>
<td>96.70</td>
<td>94.96</td>
<td>94.60</td>
</tr>
<tr>
<td>XLNet-base</td>
<td>96.81</td>
<td>95.02</td>
<td>94.74</td>
</tr>
<tr>
<td>XLNet-large</td>
<td>96.97</td>
<td>95.15</td>
<td>95.01</td>
</tr>
</tbody>
</table>

Overall, results on PTB, Brown, and UD demonstrate the effectiveness of our approach under different configurations (i.e., using the base and large versions of BERT and XLNet encoders, with and without BF), where consistent improvements are observed in most cases, even though BERT and XLNet baselines have already achieved good performance. Particularly, it is promising to observe that our approach works well on UD (i.e., the cross-standard setting), where the pre-trained models has rather low performance (e.g., according to Table 4, BERT-large achieves 65.31% UAS on UD test set after pre-training) before they are fine-tuned on the gold standard. This observation demonstrates the effectiveness of our approach in cases where other approaches (e.g., training on the combination of the auto-parsed data and the gold training data) may not work well owing to the poor quality of auto-parsed data. Besides, our final parser with BF also works well in most cases with its architecture differing from the one used in pre-training. It shows the effectiveness and robustness of our approach to leverage the structure-aware encoder obtained from a parser with a different architecture, where the representation obtained from such encoder may not fit into the final parser due to the differences of the architectures in pre-training and fine-tuning.

In addition, we compare our best performing model using BERT-large and XLNet-large and BF with previous studies on the PTB test set and report the results in Table 6. Overall, our approach outperforms all previous graph-based approaches (i.e., the ones without the “†” mark) except Mrini et al. (2020) that leverage auto-generated part-of-speech (POS) tags. Particularly, our model outperforms Zhou and Zhao (2019) and Zhou et al. (2020a) that perform constituency and dependency parsing at the same time through head-driven phrase structure grammar (HPSG) parsing. Besides, compared with Mrini et al. (2020) who proposed label attention layer to enhance the study of Zhou and Zhao (2019) on HPSG parsing, our approach obtain inferior performance because we do not use the label attention layer or the auto-generated POS tags. Given that our approach out-performs Zhou and Zhao (2019) on the test set of PTB, the effectiveness of our approach for dependency parsing is still valid and promising.
4.2 The Effect of System Design

Our parser is different from many of the previous studies in two ways: (1) the auto-parsed data is used for pre-training only, and (2) the fine-tuning step uses a different decoder from the one used in pre-training, whose weights are initialized randomly.

To determine the impact of those decisions, we build three more parsers for comparison. The first one uses the architecture in Figure 1(a) and is trained with the union of the auto-parsed and gold standard data (we denote this approach as “Union”) without the fine-tuning step. The second parser (“Fine-tuning”) is pre-trained with the auto-parsed data and fine-tuned with gold dependency trees, but the two stages use the same decoder (as in Figure 1(a)) and the decoder’s weights for fine-tuning are initialized with the weights from pre-training. The third one (“Randomize”) is the same as the second one but the weights of the decoder derived from pre-training are thrown away before fine-tuning. The “Randomize” system differs from our final parser only in that our final parser uses a different decoder in the fine-tuning stage.

All aforementioned three approaches use BERT-base encoder. For auto-parsed data, we randomly select sentences from English Wiki where the number of selected sentences equals to the number of sentences in the training set of different datasets (i.e., 40K auto-parsed sentences for PTB and Brown, and 13K auto-parsed sentences for UD).

For each approach, we run it five times with different random data and report the average results (LAS for PTB and Brown, and UAS for UD) of them, as well as the average results of BERT-base baseline and our final parser, in Table 7.

It is observed that “Randomize” consistently outperforms the other two approaches on the test set of all datasets. Particularly, for cross-standard settings on UD, because the auto-parsed data and the UD data use different dependency standards, the quality of the auto-parsed data can be considered relatively low with respect to the UD standard (this can be confirmed by the low model performance of the parser pre-trained on auto-parsed data on the test set of UD (see Table 4)). Under this setting, the “Union” and “Fine-tuning” even achieve inferior results (the results are underlined) compared with the BERT-base baseline, because they suffer from the gap between the dependency standards of auto-parsed data and gold data. On the contrary,

We also tested other numbers of selected auto-parsed sentences and obtain similar observations.

We report UAS (where the dependency type are ignored in evaluation) for UD because the auto-parsed data and the UD use different dependency standards.
Approach  |  PTB  |  Brown  |  UD  
---|---|---|---
BERT-base  |  94.65  |  91.26  |  92.86  
+ Union  |  94.69  |  91.30  |  83.51  
+ Fine-tuning  |  94.76  |  91.38  |  92.14  
+ Randomize  |  **94.94**  |  **91.47**  |  **93.10**  
Our Final Parser  |  **95.19**  |  **91.60**  |  **92.98**  

Table 7: The average performance (LAS for PTB and Brown, UAS for UD) of different approaches using BERT-base encoder with the scores lower than the baseline highlighted by underlines. “Union” refers to the model that is trained on the union of auto-parsed and gold data; “Fine-tuning” denotes the model that is pre-trained on auto-parsed data and then further fine-tuned on the gold data; “Randomize” is the same as “Fine-tuning” except that the weights of the decoder is randomly initialized before fine-tuning. The best scores among “Union”, “Fine-tuning”, and “Randomize” are highlighted by boldface. All three models and the baseline use the architecture in Figure 1(a) and the only difference between “Randomize” and our final parser is the architecture of decoder in fine-tuning.

“Randomize” is able to carefully address the noise in the auto-parsed data by using the auto-parsed and the gold standard data in different stages and getting rid of the pre-trained decoder (which may learn the noise) and randomly initializing a new one before fine-tuning.

### 4.3 The Size of Auto-parsed Data

An essential question for evaluating our approach is that how many data (auto-parsed with noise) are required to improve the parsers with various model sizes. To answer its question, we pre-train four models with variant sizes, i.e., 6-layer BERT-base (55M parameters), 12-layer BERT-base (110M parameters), 18-layer BERT-large (252M parameters), and 24-layer BERT-large (336M parameters), on different amount of randomly selected auto-parsed Wiki data. Figure 2 illustrates the averaged improvement (LAS) of four different models over their corresponding baselines on three test sets: (a) PTB, (b) Brown, and (c) UD, with respect to the ratio of the pre-training sentence number to the model size. It is interesting that, for the in-domain setting (i.e., when both training and test sets are from the PTB), the zero points of different curves for all models are roughly the same, i.e., around 0.3, which means that the pre-training only needs a little bit over 300 sentences for every 50M parameters in a parser to ensure an improvement (e.g., for 24-layer BERT-large, it only requires two thousand auto-parsed sentences to obtain a better-than-original parser).

This finding is highly encouraging since it only needs a small amount of auto-parsed data (compared to 92M sentences in English Wiki) to improve a large model. Similar observations can be drawn for cross-domain and cross-standard settings on Brown and UD datasets, where more (since there are gaps between the pre-training and the fine-tuning data) but still limited auto-parsed data is required to ensure that improvement. Particularly, for each dataset, we found there exists a rather stable ratio for different models, e.g., 0.3 for PTB, 0.6 for Brown, etc., which is a meaningful guidance to improve parsers’ performance regarding to their parameters. An explanation to this observation is that structural data is useful to update representation models (Gubbins and Vlachos, 2013; Levy and Goldberg, 2014; Zhou et al., 2020b) so that a limited amount (w.r.t. model size) could greatly affect model performance especially when they are applied on structure-prediction tasks such as parsing.

### 4.4 The Choice of Existing Parsers

Another factor that may affect the performance of our systems is the off-the-shelf parser used to produce auto-parsed data. To assess the effect of the parser on the performance of the final systems, we experimented with two more parsers from our baselines, i.e., the ones using BERT-base (Parser I) and XLNet-base (Parser II) trained on PTB, in addition to BNP-SD as described in Section 3.2: Table 8 reports the LAS of models with or without pre-training. While pre-training improves the performance with auto-parsed data from all four

<table>
<thead>
<tr>
<th>Parsers</th>
<th>PTB</th>
<th>Brown</th>
<th>UD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNP-SD (94.03)</td>
<td>95.37</td>
<td>91.83</td>
<td>90.48</td>
</tr>
<tr>
<td>Parser I (94.78)</td>
<td>94.83</td>
<td>91.58</td>
<td>90.42</td>
</tr>
<tr>
<td>Parser II (95.26)</td>
<td>95.38</td>
<td>91.84</td>
<td>91.70</td>
</tr>
</tbody>
</table>

Table 8: Performance (LAS) comparison of a model (using BERT-base with BF) pre-trained on auto-parsed English Wiki from different parsers. N/A refers to no pre-training. Parser I and Parser II are the parsers (with BF) trained on PTB using BERT-base and XLNet-base, respectively. The LAS from all parsers on the PTB test set are reported in parentheses for reference.
Figure 2: The improvement (LAS) of four models over their baselines on three test sets. The X-axis is the number of auto-parsed sentences used for pre-training divided by the number of model parameters, and then multiplied by 50,000 (to make the scale more readable).

<table>
<thead>
<tr>
<th>Models</th>
<th>PTB</th>
<th>Brown</th>
<th>UD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-base</td>
<td>94.70</td>
<td>91.46</td>
<td>89.09</td>
</tr>
<tr>
<td>+ Dep. Wiki (-BF)</td>
<td>95.30</td>
<td>91.76</td>
<td>90.39</td>
</tr>
<tr>
<td>+ Dep. Wiki (+BF)</td>
<td><strong>95.35</strong></td>
<td><strong>91.80</strong></td>
<td><strong>90.43</strong></td>
</tr>
<tr>
<td>XLNet-base</td>
<td>95.19</td>
<td>91.88</td>
<td>91.50</td>
</tr>
<tr>
<td>+ Dep. Wiki (-BF)</td>
<td>95.50</td>
<td>92.38</td>
<td>91.39</td>
</tr>
<tr>
<td>+ Dep. Wiki (+BF)</td>
<td><strong>95.54</strong></td>
<td><strong>92.40</strong></td>
<td><strong>91.97</strong></td>
</tr>
</tbody>
</table>

Table 9: The average LAS of final parsers (with BF) using BERT-base and XLNet-base encoders, with (+) and without (-) using BF in pre-training.

In the main experiments, we pre-train the parser without using BF. To explore its effect, we conduct an ablation study where BF is used in pre-training. Table 9 reports the average LAS of different final parsers (with BF) using BERT-base and XLNet-base encoders, where BF is used (i.e., “+BF”) or not used (i.e., “-BF”) in pre-training. It is observed that using BF in pre-training results in similar performance compared with the settings where BF is not used. It demonstrates the robustness of our approach where the architecture of the final parser (with BF) does not need necessary to be identical to the one (without BF) in pre-training to obtain promising improvement over the baselines. In addition, it is worth-noting that for experiments on UD with XLNet-base, “+ Dep. Wiki (+BF)” outperforms the baseline model whereas “+ Dep. Wiki (-BF)” fails to do so. The explanation could be the following. For “+ Dep. Wiki (+BF)”, the only gap between pre-training and fine-tuning is the dependency standard, whereas “+ Dep. Wiki (-BF)” faces an additional gap that the architectures used in pre-training and fine-tuning are different. Therefore, “+ Dep. Wiki (-BF)” fails to overcome the two gaps and thus results in inferior results compared with the XLNet baseline. On the other hand, when BF is also used in pre-training, the gap between the architectures does not exist, which allows our final parser to obtain a higher performance.

5 Related Work

Recent studies for dependency parsing use advanced architectures (e.g., bi-LSTM, BERT) to capture contextual information so as to achieve outstanding performance (Shen et al., 2021; Zhang et al., 2021; Yang and Tu, 2021; Li et al., 2021). To further improve dependency parsing, approaches
such as bi-affine attentions (Dozat and Manning, 2017; Attardi et al., 2021; Xu and Koehn, 2021), HPSG parsing (Zhou and Zhao, 2019; Zhou et al., 2020a; Mrini et al., 2020), TreeCRF (Zhang et al., 2020) are further applied. Besides, to improve model performance, there are studies that use existing dependency parsers and auto-parsed data through model ensemble (Attardi and Dell’Orletta, 2009; Surdeanu and Manning, 2010; Che et al., 2018) or semi-supervised approaches (Sagae and Lavie, 2006; Chen et al., 2009; Prokopidis and Papaioannou, 2014; Yu and Bohnet, 2017; Zhang et al., 2021; Wagner and Foster, 2021).

Compared to previous studies that use auto-parsed data, our approach differs in several ways. First, our encoder is structure-aware as it is pre-trained with dependency trees. Second, because auto-parsed data is noisy and may use dependency standard different from that of the test data (in the cross-standard setting), it is used in pre-training only. In contrast, training a parser on the union of auto-parsed data and gold data would not work well, especially in the cross-standard setting, as shown in Table 7. Third, the decoder of the fine-tuning stage starts with randomly initialized weights, instead of with the weights learned from the pre-training stage, thus ensuring that the decoder in the final parser will not be affected by the noisy auto-parsed data.

6 Conclusion

In this study, we propose a simple and effective solution to improve dependency parser through pre-training on auto-parsed data. In doing so, the encoder is able to learn structural information from the auto-parsed data in pre-training. During fine-tuning, a different decoder is used and its weights initialized randomly, thus reducing the impact of errors in the auto-parsed data. We have run a large number of experiments under different settings (e.g., cross-domain vs. cross-standard, -BF vs. +BF, different parsers used to parse Wiki) and shown that our approach outperforms strong baselines and many previous studies under those settings. Furthermore, pre-training needs only a small amount of auto-parsed data (e.g., 2K sentences for a BERT-large based parser on the PTB test set) to ensure improvement over strong baselines.

References


