1. Introduction

There exists a trend in syntactic dependency parsing towards simpler and simpler approaches. For example, early dependency parsing algorithms had a runtime complexity of $O(n^3)$ (Eisner, 1996). These were followed by maximum spanning tree algorithms with a complexity of $O(n^2)$ (McDonald et al., 2005), and shift-reduce methods, which are linear in sentence length ($O(n)$) (Nivre, 2003). This quest for simplicity continued with the recent works of Ma et al. (2018) and Fernández-González and Gómez-Rodríguez (2019), which replaced the shift-reduce algorithm with pointer networks, which are also linear but are arguably simpler than the shift-reduce algorithm.

This evolutionary trend begs the question: can we simplify dependency parsing further, without losing considerable performance? This paper indicates that the answer is yes. We propose an extremely simple method for dependency parsing that treats parsing as tagging (PaT). That is, our approach addresses the parsing of dependency trees with a sequence model implemented with a bidirectional LSTM over BERT embeddings, where the “tag” to be predicted at each token position is the relative position of the corresponding head. For example, for the sentence *John eats cake*, the tag to be predicted for the token *cake* is $-1$ because its head (*eats*) occurs one token to the left. Despite its simplicity, our approach performs well. For example, our approach outperforms the state-of-the-art method of Fernández-González and Gómez-Rodríguez (2019) on Universal Dependencies (UD) by 1.76% unlabeled attachment score (UAS) for English, 1.98% UAS for French, and 1.16% UAS for German. On average, on 15 UD languages, our method with minimal tuning performs comparably with this state-of-the-art approach, being only 0.16% UAS, and 0.82% LAS behind.

Keywords: dependency parsing, sequence methods

(1) We propose a simple approach for dependency parsing that operates in three stages, which are all widely used in other natural language processing (NLP) approaches. First, we encode the input tokens using a combination of randomly initialized word embeddings, contextualized embeddings (Devlin et al., 2018), character-level embeddings generated using a convolutional neural network (CNN), and part-of-speech (POS) embeddings. Second, these representations are fed into a BiLSTM. Finally, using the BiLSTM’s hidden states, we predict the relative position of the head for each token in the sentence and the label of the corresponding dependency.

(2) Despite its simplicity, we show that our approach performs well. For example, our approach outperforms the state-of-the-art method of Fernández-González and Gómez-Rodríguez (2019) on Universal Dependencies (UD) by 1.76% unlabeled attachment score (UAS) and 1.26% labeled attachment score (LAS) for English, 1.98% UAS and 1.65% LAS for French, and 1.16% UAS and 0.45% LAS for German. On average, on 15 UD languages, our method performs near this state-of-the-art approach, being only 0.16% UAS and 0.82% LAS below.

(3) An ablation analysis indicates that the BERT contextualized embeddings and the word-level BiLSTM have considerable contributions to performance, confirming earlier work that indicated the importance of contextualized embeddings for many NLP tasks (Devlin et al., 2018), and that LSTMs capture grammatical structure (Linzen et al., 2016; Kuncoro et al., 2018). As removing either of these components impacts performance negatively, this suggests that we are approaching the limits of simplicity for this task.

2. Related Work

Early algorithms for automatic dependency parsing such as the dynamic programming approach proposed by Eisner (1996) had a complexity of $O(n^3)$. Currently, the two primary approaches, i.e., the graph-based maximum spanning tree approach proposed by McDonald et al. (2005) and the transition based approach formalized by Nivre (2003), have complexities of $O(n^2)$ and $O(n)$, respectively. Many variants of these lower-complexity transition-based approaches have been proposed. Yamada and Matsumoto (2003) trained a support vector machine to direct a shift-reduce parser. Chen and Manning (2014) encode features as embeddings and feed them to a multi-layer perceptron.
Kiperwasser and Goldberg (2016) use a BiLSTM to learn feature representations, and use these to encode the parser state. Xipeng (2009) proposed a simpler method, where the dependency parsing task is transformed into a sequence labeling problem using conditional random fields. More recently, Ma et al. (2018) proposed a stack-pointer network, which uses information from the sentence as a whole. Fernández-González and Gómez-Rodríguez (2019) introduced a left-to-right parsing approach with a pointer network, reducing the number of transitions required by Ma et al. (2018) from $2n-1$ to $n$. Our approach continues this simplification trend with a method that reduces parsing to a simple sequence modeling task, similar with the approaches of (Strzyn et al., 2019) and (Li et al., 2018), albeit with a much simpler architecture and better overall performance. Li et al. (2018) propose an encoder-decoder architecture with an attention layer; Strzyz et al. (2019) propose an encoder-decoder architecture with a more complex encoding scheme. We propose a much simpler architecture, consisting of only a BiLSTM operating on top of contextualized embeddings. Similar to our direction, Hewitt and Manning (2019) showed that syntax trees are embedded in a linear transformation of the BERT and ELMo embeddings. However, they did not consider labels and the evaluation was done only on undirected unlabeled attachment score (UUAS) after generating a minimum spanning tree on the predicted distance graph.

In the pool of exciting methods that simplify the parsing task, our approach is closer in spirit to the approaches of Zhang et al. (2017) for dependency parsing, and Marcheggiani et al. (2017) for semantic role labeling. Both these works also encode the tokens in a sentence using an LSTM. However, they operate over pairs of words to predict a syntactic dependency between a modifier and head (Zhang et al., 2017), or a token’s semantic role given a predicate (Marcheggiani et al., 2017). Further, Marcheggiani et al. (2017) separately encode the sentence with an LSTM for each predicate. Our approach is simpler and faster, i.e., we encode the sentence just once with the LSTM, and we predict the relative position of head words rather than relying on all possible pairs of words.

3. PaT: Parsing as Tagging

Our approach casts the task of dependency parsing to one of sequence tagging. Since in the dependency tree formalism, each token has a single head, which may be another token in the sentence or ROOT, we can naturally map the parsing task into a tagging problem. That is, the “tag” of each token becomes the relative position of its head (relospos), computed as follows:

$$relpos = \begin{cases} 0, & \text{if head is ROOT} \\ head - id, & \text{otherwise} \end{cases}$$  \hspace{1cm} (1)

where $id$ is the absolute position of the current token (starting at 1) (Buchholz and Marsi, 2006), and head is the absolute position of the corresponding head word. As an example, consider the sentence in Figure 1. The relative position of the head of cake is $-1$ because its head (eats) is one word to the left. A relative position of 0 indicates that the token is headed by the ROOT. To predict the dependency parse, our architecture straightforwardly encodes the sentence, and for each token predicts these relative positions, as well as the label of the directed edge between them.

Because we framed this task as tagging, we need to constrain prediction to a finite number of possible relative positions. We empirically chose a range of $(-50, 50)$, which accounts for 99.9% of the English dependencies in the Universal Dependencies training dataset. We use the same range for all languages.

3.1. Token Representations

Given an input sentence $s$, consisting of $n$ tokens $t_1, \ldots, t_n$, we represent each token $t_i$ by a vector $e_i$, which is the concatenation of (a) the pretrained BERT representation of Devin et al. (2018) of the token; (b) the word embedding (we) for the token; (c) the character CNN encoding (ce) of the token; and (d) its part-of-speech (POS) embedding (pos):

$$e_i = [i_{bert}^t, i_{we}^t, i_{ce}^t, i_{pos}^t]$$  \hspace{1cm} (2)

For the character-level encoding, we use a character CNN with max pooling, which has been shown to be useful by Lample et al. (2016). We learned the embeddings for part-of-speech tags and words, which were initialized using Xavier initialization (Glorot and Bengio, 2010).

3.2. Architecture

Our proposed architecture, as shown in Figure 1, consists of a BiLSTM (Hochreiter and Schmidhuber, 1997) that operates over our token representations $e_i$, producing the corresponding hidden state $h_i$. For each token $t_i$ we use $h_i$ in two ways: to predict the relative position of the head, and to predict the label of the corresponding dependency relation. For the relative position, we pass $h_i$ into a multi-layer perceptron (MLP), followed by a linear layer and a softmax to get a distribution over the possible relative positions. For the corresponding dependency label, we concatenate the MLP outputs corresponding to $t_i$ and its predicted head $h_{head}$ and pass them to another linear layer and softmax to produce a distribution over the dependency labels.

The objective function of the model is calculated by the cross entropy (Nasr et al., 2002) between predicted and observed values of both the dependency labels and relative positions.

3.3. Cycle Detection

Note that there is nothing in the above architecture that prevents our approach from generating cycles. To control for this, we explore three post-processing options for handling cycles:

(1) No cycle removal, leaving the output of the model unchanged.

(2) Greedy cycle removal, where we globally sort the predicted dependencies based on weight. Then, we incrementally add to the output tree the dependency with the highest weight.

(3) 2We used pre-trained “BERT-Base Uncased” model for English and “BERT-Base Multilingual Cased (new)” model for other languages. We do not fine-tune.
(3) Optimal cycle removal, which finds a maximum spanning tree (MST) using the Chu-Liu-Edmonds algorithm (Edmonds, 1967).

4. Experiments and Results

We used the same datasets as Fernández-González and Gómez-Rodríguez (2019) and Ma et al. (2018). We tested our model on the Stanford Dependencies (de Marneffe and Manning, 2008) (SD) version of the Penn Treebank (Marcus et al., 1993) and on 15 languages from the Universal Dependencies (UD) Treebank. For SD we used the standard splits and the predicted part-of-speech tags. For UD we used the same set up as Fernández-González and Gómez-Rodríguez (2019) with the addition of Arabic, Estonian, and Japanese.

Table 1 lists the unlabeled (UAS) and labeled (LAS) accuracies of our model (averaged over 3 runs) for 15 languages from the UD Treebank, compared against the state-of-the-art method of Fernández-González and Gómez-Rodríguez (2019). The hyper parameters were tuned on English UD, by exploring 100 configurations. The same hyper parameters were used for all languages in this table.

In Table 2 we show the performance of our method after tuning the hyper parameters for each language. For each language, we selected the best performing model from a set of 15 configurations. The same search space was used for all languages.

In our experiments, we used early stopping to halt training if there is no improvement for 3 epochs. In some instances, the training stops too early, resulting in a bigger standard deviation in some cases (e.g., Italian), or sub-optimal performance in others. For future experiments, we recommend either to use a warm-up phase of 10–15 epochs, or to increase the early stopping threshold to 5.

Despite the simple architecture, the two tables show that PaT outperforms the method of Fernández-González et al. on six languages for UAS, and on four for LAS; on average, on the 15 languages, its performance is 0.16% UAS and 0.82% LAS behind this state-of-the-art parser.

Even though PaT does not enforce the production of acyclic structures, relatively few parse trees produced by PaT contain cycles: between 0.47% for Japanese and 17.20% for Arabic, for an average of 5.11% ± 1.11. Further, we observe that the simple, greedy cycle removal performs similarly to the optimal MST algorithm, which is probably due
Table 1: Accuracy comparison between PaT and the best performing models of Fernández-González et al. on the test partitions of 15 languages from the Universal Dependencies Treebank. The bold font indicates the best performing model on each language. PaT's reported results are the average and the standard deviation (stdev) over three runs. The *same parameters, tuned for English*, were used for all languages.

<table>
<thead>
<tr>
<th>Language</th>
<th>UAS</th>
<th>LAS</th>
<th>Without cycle removal</th>
<th>With greedy cycle removal</th>
<th>With optimal cycle removal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>92.98</td>
<td>83.40</td>
<td>92.21</td>
<td>90.80</td>
<td>89.50</td>
</tr>
<tr>
<td>es</td>
<td>93.42</td>
<td>83.73</td>
<td>92.79</td>
<td>91.94</td>
<td>90.62</td>
</tr>
<tr>
<td>fr</td>
<td>93.97</td>
<td>85.07</td>
<td>93.33</td>
<td>92.57</td>
<td>91.26</td>
</tr>
<tr>
<td>it</td>
<td>94.28</td>
<td>87.28</td>
<td>93.77</td>
<td>93.04</td>
<td>91.71</td>
</tr>
<tr>
<td>ja</td>
<td>92.98</td>
<td>84.28</td>
<td>92.45</td>
<td>91.73</td>
<td>90.40</td>
</tr>
<tr>
<td>nl</td>
<td>93.77</td>
<td>86.50</td>
<td>93.22</td>
<td>92.50</td>
<td>91.18</td>
</tr>
<tr>
<td>no</td>
<td>92.21</td>
<td>83.40</td>
<td>91.93</td>
<td>91.24</td>
<td>90.93</td>
</tr>
<tr>
<td>ru</td>
<td>94.57</td>
<td>87.38</td>
<td>94.21</td>
<td>93.52</td>
<td>92.23</td>
</tr>
</tbody>
</table>

Table 2: Accuracy comparison between tuned PaT and the best performing models of Fernández-González et al. on the test partitions of 15 languages from the Universal Dependencies Treebank. The bold font indicates the best performing model on each language. PaT's reported results are the average and the standard deviation (stdev) over three runs. For each language, we select the *best performing configuration* from a set of 12 possible configurations.

<table>
<thead>
<tr>
<th>Language</th>
<th>UAS</th>
<th>LAS</th>
<th>Without cycle removal</th>
<th>With greedy cycle removal</th>
<th>With optimal cycle removal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>92.98</td>
<td>83.40</td>
<td>92.21</td>
<td>90.80</td>
<td>89.50</td>
</tr>
<tr>
<td>es</td>
<td>93.42</td>
<td>83.73</td>
<td>92.79</td>
<td>91.94</td>
<td>90.62</td>
</tr>
<tr>
<td>fr</td>
<td>93.97</td>
<td>85.07</td>
<td>93.33</td>
<td>92.57</td>
<td>91.26</td>
</tr>
<tr>
<td>it</td>
<td>94.28</td>
<td>87.28</td>
<td>93.77</td>
<td>93.04</td>
<td>91.71</td>
</tr>
<tr>
<td>ja</td>
<td>92.98</td>
<td>84.28</td>
<td>92.45</td>
<td>91.73</td>
<td>90.40</td>
</tr>
<tr>
<td>nl</td>
<td>93.77</td>
<td>86.50</td>
<td>93.22</td>
<td>92.50</td>
<td>91.18</td>
</tr>
<tr>
<td>no</td>
<td>92.21</td>
<td>83.40</td>
<td>91.93</td>
<td>91.24</td>
<td>90.93</td>
</tr>
<tr>
<td>ru</td>
<td>94.57</td>
<td>87.38</td>
<td>94.21</td>
<td>93.52</td>
<td>92.23</td>
</tr>
</tbody>
</table>

Table 3 shows the UAS error rates for English UD based on distance between the dependent and its head. PaT's error rate is small for heads that are within three tokens to their dependents, the most frequent situation, and increases with the distance.

Table 4 shows an ablation analysis performed on the English UD dataset. As shown, eliminating the BiLSTM has the biggest impact on performance, followed by BERT, and POS embeddings. This confirms previous observations that LSTMs capture grammatical structure [Linzen et al., 2016; Kuncoro et al., 2018], and are the critical ingredient to de-

to the above-mentioned fact that only a small percentage of sentences yield structures with cycles.
We proposed an approach that reframes dependency parsing as a sequence tagging task that relies solely on surface information. Specifically, for each token in a given sentence, we predict the relative position to that token’s head, as well as the corresponding dependency label. Our approach achieves state-of-the-art performance on three Universal Dependencies languages and strong performance on nine others. This work suggests that parsing as tagging can serve as a new, simple, yet strong baseline for dependency parsing.

For reproducibility, we release the code behind this work as open source. The software, together with all hyper parameters used, is available at this URL: https://github.com/clulab/releases/tree/master/lrec2020-pat

6. Acknowledgments

This work was funded by the Bill and Melinda Gates Foundation HBGDI Initiative. Marco Valenzuela-Escárciga and Mihai Surdeanu declare a financial interest in lum.ai. This interest has been properly disclosed to the University of Arizona Institutional Review Committee and is managed in accordance with its conflict of interest policies.

7. Bibliographical References


