Abstract

Transition-based dependency parsers often need sequences of local shift and reduce operations to produce certain attachments. Correct individual decisions hence require global information about the sentence context and mistakes cause error propagation. This paper proposes a novel transition system, \textit{arc-swift}, that enables direct attachments between tokens farther apart with a single transition. This allows the parser to leverage lexical information more directly in transition decisions. Hence, \textit{arc-swift} can achieve significantly better performance with a very small beam size. Our parsers reduce error by 3.7–7.6\% relative to those using existing transition systems on the Penn Treebank dependency parsing task and English Universal Dependencies.

1 Introduction

Dependency parsing is a longstanding natural language processing task, with its outputs crucial to various downstream tasks including relation extraction (Schmitz et al., 2012; Angeli et al., 2015), language modeling (Gubbins and Vlachos, 2013), and natural logic inference (Bowman et al., 2016).

Attractive for their linear time complexity and amenability to conventional classification methods, \textit{transition-based} dependency parsers have sparked much research interest recently. A transition-based parser makes sequential predictions of transitions between states under the restrictions of a \textit{transition system} (Nivre, 2003). Transition-based parsers have been shown to excel at parsing shorter-range dependency structures, as well as languages where non-projective parses are less pervasive (McDonald and Nivre, 2007).

However, the transition systems employed in state-of-the-art dependency parsers usually define very local transitions. At each step, only one or two words are affected, with very local attachments made. As a result, distant attachments require long and not immediately obvious transition sequences (e.g., \textit{ate}→\textit{chopsticks} in Figure 1, which requires two transitions). This is further aggravated by the usually local lexical information leveraged to make transition predictions (Chen and Manning, 2014; Andor et al., 2016).

In this paper, we introduce a novel transition system, \textit{arc-swift}, which defines non-local transitions that directly induce attachments of distance up to $n$ ($n$ = the number of tokens in the sentence). Such an approach is connected to \textit{graph-based} dependency parsing, in that it leverages pairwise scores between tokens in making parsing decisions (McDonald et al., 2005).

We make two main contributions in this paper. Firstly, we introduce a novel transition system for dependency parsing, which alleviates the difficulty of distant attachments in previous systems by allowing direct attachments anywhere in the stack. Secondly, we compare parsers by the number of mistakes they make in common linguistic con-
instructions. We show that arc-swift parsers reduce errors in attaching prepositional phrases and conjunctions compared to parsers using existing transition systems.

2 Transition-based Dependency Parsing

Transition-based dependency parsing is performed by predicting transitions between states (see Figure 1 for an example). Parser states are usually written as \((\sigma, i, j, \beta, A)\), where \(\sigma\) denotes the stack with token \(i\) on the top, \(j, \beta\) denotes the buffer with token \(j\) at its leftmost, and \(A\) the set of dependency arcs. Given a state, the goal of a dependency parser is to predict a transition to a new state that would lead to the correct parse. A transition system defines a set of transitions that are sound and complete for parsers, that is, every transition sequence would derive a well-formed parse tree, and every possible parse tree can also be derived from some transition sequence.\(^1\)

Arc-standard (Nivre, 2004) is one of the first transition systems proposed for dependency parsing. It defines three transitions: shift, left arc (LArc), and right arc (RArc) (see Figure 2 for definitions, same for the following transition systems), where all arc-inducing transitions operate on the stack. This system builds the parse bottom-up, i.e., a constituent is only attached to its head after it has received all of its dependents. A potential drawback is that during parsing, it is difficult to predict if a constituent has consumed all of its right dependents. Arc-eager (Nivre, 2003) remedies this drawback by defining arc-inducing transitions that operate between the stack and the buffer. As a result, a constituent no longer needs to be complete before it can be attached to its head to the left, as a right arc doesn’t prevent the attached dependent from taking further dependents of its own.\(^2\)

Kuhlmann et al. (2011) propose a hybrid system derived from a tabular parsing scheme, which they have shown both arc-standard and arc-eager can be derived from. Arc-hybrid combines LArc from arc-eager and RArc from arc-standard to build dependencies bottom-up.

3 Non-local Transitions with arc-swift

The traditional transition systems discussed in Section 2 only allow very local transitions affecting one or two words, which makes long-distance dependencies difficult to predict. To illustrate the limitation of local transitions, consider parsing the following sentences:

\[I\text{ ate fish with ketchup.}\]

\[I\text{ ate fish with chopsticks.}\]

The two sentences have almost identical structures, with the notable difference that the prepositional phrase is complementing the direct object in the first case, and the main verb in the second.

For arc-standard and arc-hybrid, the parser would have to decide between Shift and RArc when the parser state is as shown in Figure 3a, where * stands for either “ketchup” or “chopsticks”.\(^3\) Similarly, an arc-eager parser would deal with the state shown in Figure 3b. Making the correct transition requires information about context words “ate” and “fish”, as well as “*”.

---

\(^1\) We only focus on projective parses for the scope of this paper.

\(^2\) A side-effect of arc-eager is that there is sometimes spurious ambiguity between Shift and Reduce transitions. For the example in Figure 1, the first Reduce can be inserted before the third Shift without changing the correctness of the resulting parse, i.e., both are feasible at that time.

\(^3\) For this example, we assume that the sentence is being parsed into Universal Dependencies.
Parsers employing traditional transition systems would usually incorporate more features about the context in the transition decision, or employ beam search during parsing (Chen and Manning, 2014; Andor et al., 2016).

In contrast, inspired by graph-based parsers, we propose arc-swift, which defines non-local transitions as shown in Figure 2. This allows direct comparison of different attachment points, and provides a direct solution to parsing the two example sentences. When the arc-swift parser encounters a state identical to Figure 3b, it could directly compare transitions RArc[1] and RArc[2] instead of evaluating between local transitions. This results in a direct attachment much like that in a graph-based parser, informed by lexical information about affinity of the pairs of words.

Arc-swift also bears much resemblance to arc-eager. In fact, an LArc[k] transition can be viewed as k – 1 Reduce operations followed by one LArc in arc-eager, and similarly for RArc[k]. Reduce is no longer needed in arc-swift as it becomes part of LArc[k] and RArc[k], removing the ambiguity in derived transitions in arc-eager. arc-swift is also equivalent to arc-eager in terms of soundness and completeness. A caveat is that the worst-case time complexity of arc-swift is $O(n^2)$ instead of $O(n)$, which existing transition-based parsers enjoy. However, in practice the runtime is nearly linear, thanks to the usually small number of reducible tokens in the stack.

4 Experiments

4.1 Data and Model

We use the Wall Street Journal portion of Penn Treebank with standard parsing splits (PTB-SD), along with Universal Dependencies v1.3 (Nivre et al., 2016) (EN-UD). PTB-SD is converted to Stanford Dependencies (De Marneffe and Manning, 2008) with CoreNLP 3.3.0 (Manning et al., 2014) following previous work. We report labelled and unlabelled attachment scores (LAS/UAS), removing punctuation from all evaluations.

Our model is very similar to that of (Kiperwasser and Goldberg, 2016), where features are extracted from tokens with bidirectional LSTMs, and concatenated for classification. For the three traditional transition systems, features of the top 3 tokens on the stack and the leftmost token in the buffer are concatenated as classifier input. For arc-swift, features of the head and dependent tokens for each arc-inducing transition are concatenated to compute scores for classification, and features of the leftmost buffer token is used for Shift. For other details we defer to Appendix A. The full specification of the model can also be found in our released code online at https://github.com/qipeng/arc-swift.

4.2 Results

We use static oracles for all transition systems, and for arc-eager we implement oracles that always Shift/Reduce when ambiguity is present (arc-eager-S/R). We evaluate our parsers with greedy parsing (i.e., beam size 1). The results are shown in Table 1. Note that K&G 2016 is trained with a dynamic oracle (Goldberg and Nivre, 2012), Andor 2016 with a CRF-like loss, and both Andor 2016 and Weiss 2015 employed beam search (with sizes 32 and 8, respectively).

For each pair of the systems we implemented, we studied the statistical significance of their difference by performing a paired test with 10,000 bootstrap samples on PTB-SD. The resulting p-values are analyzed with a 10-group Bonferroni-Holm test, with results shown in Table 2. We note
<table>
<thead>
<tr>
<th>Model</th>
<th>TS</th>
<th>PTB-SD UAS</th>
<th>PTB-SD LAS</th>
<th>EN-UD UAS</th>
<th>EN-UD LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>asd</td>
<td>94.0</td>
<td>91.7</td>
<td>85.6</td>
<td>81.5</td>
</tr>
<tr>
<td>This work</td>
<td>aeS</td>
<td>94.0</td>
<td>91.8</td>
<td>85.4</td>
<td>81.4</td>
</tr>
<tr>
<td>This work</td>
<td>aeR</td>
<td>93.8</td>
<td>91.7</td>
<td>85.2</td>
<td>81.2</td>
</tr>
<tr>
<td>This work</td>
<td>ah</td>
<td>93.9</td>
<td>91.7</td>
<td>85.4</td>
<td>81.3</td>
</tr>
<tr>
<td>This work</td>
<td>asw</td>
<td>94.3</td>
<td>92.2</td>
<td>86.1</td>
<td>82.2</td>
</tr>
</tbody>
</table>

Andor 2016  asd  94.6  92.8  84.8*  80.4*
K&G 2016    ah   93.9  91.9
Weiss 2015  asd  94.0  92.1
C&M 2014    asd  91.8  89.6

Table 1: Performance of parsers using different transition systems on the Penn Treebank dataset. *: Obtained from their published results online.\(^6\)

<table>
<thead>
<tr>
<th></th>
<th>aeS</th>
<th>asd</th>
<th>ah</th>
<th>aeR</th>
</tr>
</thead>
<tbody>
<tr>
<td>asw</td>
<td><em><strong>/</strong></em></td>
<td><em><strong>/</strong></em></td>
<td><em><strong>/</strong></em></td>
<td><em><strong>/</strong></em></td>
</tr>
<tr>
<td>aeS</td>
<td>-/-</td>
<td>-/-</td>
<td>*/-</td>
<td>-/-</td>
</tr>
<tr>
<td>asd</td>
<td>-/-</td>
<td>*/-</td>
<td>-/-</td>
<td>-/-</td>
</tr>
<tr>
<td>ah</td>
<td>-/-</td>
<td>*/-</td>
<td>-/-</td>
<td>-/-</td>
</tr>
</tbody>
</table>

Table 2: Significance test for transition systems. Each grid shows adjusted test result for UAS and LAS, respectively, showing whether the system on the row is significantly better than that on the column. "***" stands for \(p < 0.001\), "**" \(p < 0.01\), "*" \(p < 0.05\), and "-" \(p \geq 0.05\).

that with almost the same implementation, arc-swift parsers significantly outperform those using traditional transition systems. We also analyzed the performance of parsers on attachments of different distances. As shown in Figure 4, arc-swift is equally accurate as existing systems for short dependencies, but is more robust for longer ones.

While arc-swift introduces direct long-distance transitions, it also shortens the overall sequence necessary to induce the same parse. A parser could potentially benefit from both factors: direct attachments could make an easier classification task, and shorter sequences limit the effect of error propagation. However, since the two effects are correlated in a transition system, precise attribution of the gain is out of the scope of this paper.

**Computational efficiency.** We study the computational efficiency of the arc-swift parser by comparing it to an arc-eager parser. On the PTB-SD development set, the average transition sequence length per sentence of arc-swift is 77.5% of that of arc-eager. At each step of parsing, arc-swift needs to evaluate only about 1.24 times the number of transition candidates as arc-eager, which results in very similar runtime. In contrast, beam search with beam size 2 for arc-eager requires evaluating 4 times the number of transition candidates compared to greedy parsing, which results in a UAS 0.14% worse and LAS 0.22% worse for arc-eager compared to greedily decoded arc-swift.

### 4.3 Linguistic Analysis

We automatically extracted all labelled attachment errors by error type (incorrect attachment or relation), and categorized a few top parser errors by hand into linguistic constructions. Results on PTB-SD are shown in Table 3.\(^7\) We note that the arc-swift parser improves accuracy on prepositional phrase (PP) and conjunction attachments, while it remains comparable to other parsers on other common errors. Analysis on EN-UD shows a similar trend. As shown in the table, there are still many parser errors unaccounted for in our analysis. We leave this to future work.

\(^7\)We notice that for some examples the parsers predicted a ccomp (complement clause) attachment to verbs “says” and “said”, where the CoreNLP output simply labelled the relation as dep (unspecified). For other examples the relation between the prepositions in “out of” is labelled as prep (preposition) instead of pcomp (prepositional complement). We suspect this is due to the converter’s inability to handle certain corner cases, but further study is warranted.
Table 3: Common parser errors on PTB-SD. The top 4 common errors are categorized and shown in this table. Errors not shown are in a long-tail distribution and warrant analyses in future work.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>asw</th>
<th>asd</th>
<th>aeS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP attachment</td>
<td>545</td>
<td>571</td>
<td>569</td>
</tr>
<tr>
<td>Noun/Adjective confusion</td>
<td>221</td>
<td>221</td>
<td>230</td>
</tr>
<tr>
<td>Conjunction attachment</td>
<td>156</td>
<td>164</td>
<td>170</td>
</tr>
<tr>
<td>Adverbial attachment</td>
<td>123</td>
<td>143</td>
<td>122</td>
</tr>
<tr>
<td>Total Errors</td>
<td>3884</td>
<td>4106</td>
<td>4100</td>
</tr>
</tbody>
</table>

5 Related Work

Previous work has also explored augmenting transition systems to facilitate longer-range attachments. Attardi (2006) extended the arc-standard system for non-projective parsing, with arc-inducing transitions that are very similar to those in arc-swift. A notable difference is that their transitions retain tokens between the head and dependent. Fernández-González and Gómez-Rodríguez (2012) augmented the arc-eager system with transitions that operate on the buffer, which shorten the transition sequence by reducing the number of Shift transitions needed. However, limited by the sparse feature-based classifiers used, both of these parsers just mentioned only allow direct attachments of distance up to 3 and 2, respectively. More recently, Sartorio et al. (2013) extended arc-standard with transitions that directly attach to left and right “spines” of the top two nodes in the stack. While this work shares very similar motivations as arc-swift, it requires additional data structures to keep track of the left and right spines of nodes. This transition system also introduces spurious ambiguity where multiple transition sequences could lead to the same correct parse, which necessitates easy-first training to achieve a more noticeable improvement over arc-standard. In contrast, arc-swift can be easily implemented given the parser state alone, and does not give rise to spurious ambiguity.

For a comprehensive study of transition systems for dependency parsing, we refer the reader to (Bohnet et al., 2016), which proposed a generalized framework that could derive all of the traditional transition systems we described by configuring the size of the active token set and the maximum arc length, among other control parameters. However, this framework does not cover arc-swift in its original form, as the authors limit each of their transitions to reduce at most one token from the active token set (the buffer). On the other hand, the framework presented in (Gómez-Rodríguez and Nivre, 2013) does not explicitly make this constraint, and therefore generalizes to arc-swift. However, we note that arc-swift still falls out of the scope of existing discussions in that work, by introducing multiple Reduces in a single transition.

6 Conclusion

In this paper, we introduced arc-swift, a novel transition system for dependency parsing. We also performed linguistic analyses on parser outputs and showed arc-swift parsers reduce errors in conjunction and adverbial attachments compared to parsers using traditional transition systems.

Acknowledgments

We thank Timothy Dozat, Arun Chaganty, Danqi Chen, and the anonymous reviewers for helpful discussions. Stanford University gratefully acknowledges the support of the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under Air Force Research Laboratory (AFRL) contract No. FA8750-13-2-0040. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, or the US government.

References


A Model and Training Details

Our model setup is similar to that of (Kiperwasser and Goldberg, 2016) (See Figure 5). We employ two blocks of bidirectional long short-term memory (BiLSTM) networks (Hochreiter and Schmidhuber, 1997) that share very similar structures, one for part-of-speech (POS) tagging, the other for parsing. Both BiLSTMs have 400 hidden units in each direction, and the output of both are concatenated and fed into a dense layer of rectified linear units (ReLU) before 32-dimensional representations are derived as classification features. As the input to the tagger BiLSTM, we represent words with 100-dimensional word embeddings, initialized with GloVe vectors (Pennington et al., 2014). The output distribution of the tagger classifier is used to compute a weighted sum of 32-dimensional POS embeddings, which is then concatenated with the output of the tagger BiLSTM (800-dimensional per token) as the input to the parser BiLSTM. For the parser BiLSTM, we use two separate sets of dense layers to derive a “head” and a “dependent” representation for each token. These representations are later merged according to the parser state to make transition predictions.

For traditional transition systems, we follow (Kiperwasser and Goldberg, 2016) by featurizing the top 3 tokens on the stack and the leftmost token in the buffer. To derive features for each token, we take its head representation $v_{\text{head}}$ and dependent representation $v_{\text{dep}}$, and perform the following biaffine combination

$$v_{\text{feat},i} = [f(v_{\text{head}}, v_{\text{dep}})]_i = \text{ReLU}(v_{\text{head}}^\top W_i v_{\text{dep}} + b_i^\top v_{\text{head}} + c_i^\top v_{\text{dep}} + d_i)$$

where $W_i \in \mathbb{R}^{32 \times 32}$, $b_i, c_i \in \mathbb{R}^{32}$, and $d_i$ is a scalar for $i = 1, \ldots, 32$. The resulting 32-dimensional features are concatenated as the input to a fixed-dimensional softmax classifier for transition decisions.

For arc-swift, we featurize for each arc-inducing transition with the same composition function in Equation (1) with $v_{\text{head}}$ of the head token and $v_{\text{dep}}$ of the dependent token of the arc to be induced. For Shift, we simply combine $v_{\text{head}}$ and $v_{\text{dep}}$ of the leftmost token in the buffer with the biaffine combination, and obtain its score by computing the inner-product of the feature and a vector. At each step, the scores of all feasible transitions are normalized to a probability distribution by a softmax function.

In all of our experiments, the parsers are trained to maximize the log likelihood of the desired transition sequence, along with the tagger being trained to maximize the log likelihood of the correct POS tag for each token.

To train the parsers, we use the ADAM optimizer (Kingma and Ba, 2014), with $\beta_2 = 0.9$, an initial learning rate of 0.001, and minibatches of size 32 sentences. Parsers are trained for 10 passes through the dataset on PTB-SD. We also find that annealing the learning rate by a factor of 0.5 for every pass after the 5th helped improve performance. For EN-UD, we train for 30 passes, and anneal the learning rate for every 3 passes after the 15th due to the smaller size of the dataset. For all of the biaffine combination layers and dense layers, we dropout their units with a small probability of 5%. Also during training time, we randomly replace 10% of the input words by an artificial UNK token, which is then used to replace
all unseen words in the development and test sets. Finally, we repeat each experiment with 3 independent random initializations, and use the average result for reporting and statistical significance tests.

The code for the full specification of our models and aforementioned training details are available at https://github.com/qipeng/arc-swift.