Arc-swift: A Novel Transition System for Dependency Parsing

Peng Qi **Christopher D. Manning**
Computer Science Department
Stanford University
{pengqi, manning}@cs.stanford.edu

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, *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, *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, *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 *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).

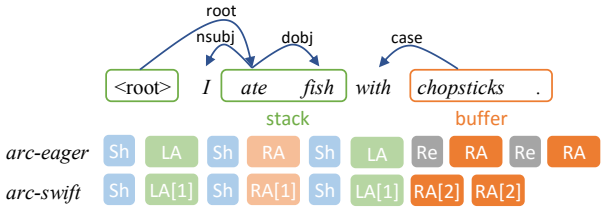

Figure 1: An example of the state of a transition-based dependency parser, and the transition sequences used by *arc-eager* and *arc-swift* to induce the correct parse. The state shown is generated by the first six transitions of both systems.

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., *ate*→*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, *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 *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-

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 110–117
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-2018

arc-standard

Shift $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
LArc $(\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j \rightarrow i)\})$
RArc $(\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i \rightarrow j)\})$

arc-hybrid

Shift $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
LArc $(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j \rightarrow i)\})$
RArc $(\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i \rightarrow j)\})$

arc-eager

Shift $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
LArc $(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j \rightarrow i)\})$
RArc $(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i \rightarrow j)\})$
Reduce $(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$

arc-swift

Shift $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
LArc$[k]$ $(\sigma|i_k|\ldots|i_1, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j \rightarrow i_k)\})$
RArc$[k]$ $(\sigma|i_k|\ldots|i_1, j|\beta, A) \Rightarrow (\sigma|i_k|j, \beta, A \cup \{(i_k \rightarrow j)\})$

Figure 2: Transitions defined by different transition systems.

structions. 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 $\sigma|i$ 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 ate fish with ketchup.
I 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 $\star$ 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 "$\star$".

[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.

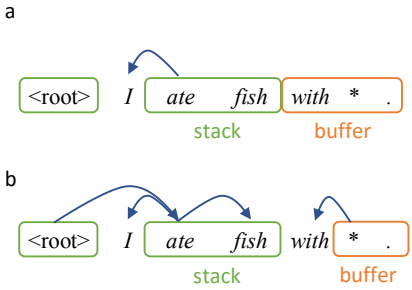

Figure 3: Parser states that present difficult transition decisions in traditional systems. In these states, parsers would need to incorporate context about "ate", "fish", and "⋆" to make the correct local transition.

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.[4] 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]This is easy to show because in *arc-eager*, all Reduce transitions can be viewed as preparing for a later LArc or RArc transition. We also note that similar to *arc-eager* transitions, *arc-swift* transitions must also satisfy certain pre-conditions. Specifically, an RArc$[k]$ transition requires that the top $k-1$ elements in the stack are already attached; LArc$[k]$ additionally requires that the k-th element is unattached, resulting in no more than one feasible LArc candidate for any parser state.

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.[5] 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

[5]In the interest of space, we abbreviate all transition systems (TS) as follows in tables: *asw* for *arc-swift*, *asd* for *arc-standard*, *aeS/R* for *arc-eager-S/R*, and *ah* for *arc-hybrid*.

Model	TS	PTB-SD		EN-UD	
		UAS	LAS	UAS	LAS
This work	*asd*	94.0	91.7	85.6	81.5
This work	*aeS*	94.0	91.8	85.4	81.4
This work	*aeR*	93.8	91.7	85.2	81.2
This work	*ah*	93.9	91.7	85.4	81.3
This work	*asw*	94.3	92.2	**86.1**	**82.2**
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]

	aeS	*asd*	*ah*	*aeR*
asw	***/***	***/***	***/***	***/***
aeS		-/-	-/-	*/-
asd			-/-	*/-
ah				-/-

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*.

[6] https://github.com/tensorflow/models/blob/master/syntaxnet/g3doc/universal.md

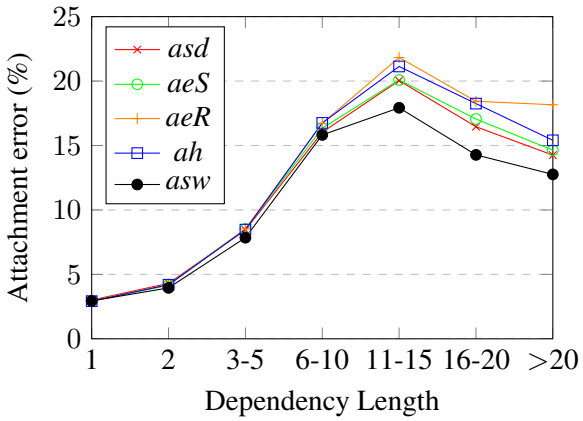

Figure 4: Parser attachment errors on PTB-SD binned by the length of the gold dependency.

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.

	asw	*aeS*	*asd*
PP attachment	545	569	571
Noun/Adjective confusion	221	230	221
Conjunction attachment	156	170	164
Adverbial attachment	123	122	143
Total Errors	3884	4100	4106

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.

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

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. https://www.aclweb.org/anthology/P16-1231.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*. http://www.aclweb.org/anthology/P15-1034.

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 166–170. http://www.aclweb.org/anthology/W06-2922.

Bernd Bohnet, Ryan McDonald, Emily Pitler, and Ji Ma. 2016. Generalized transition-based dependency parsing via control parameters. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. volume 1, pages 150–160. https://www.aclweb.org/anthology/P16-1015.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*. http://www.aclweb.org/anthology/P16-1139.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*. pages 740–750. http://www.aclweb.org/anthology/D14-1082.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. The Stanford typed dependencies representation. In *COLING 2008: Proceedings of the Workshop on Cross-framework and Cross-domain Parser Evaluation*. pages 1–8. http://www.aclweb.org/anthology/W08-1301.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2012. Improving transition-based dependency parsing with buffer transitions. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. pages 308–319. http://www.aclweb.org/anthology/D12-1029.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*. pages 959–976. http://www.aclweb.org/anthology/C12-1059.

Carlos Gómez-Rodríguez and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics* 39(4):799–845.

Joseph Gubbins and Andreas Vlachos. 2013. Dependency language models for sentence completion. In *EMNLP*. volume 13, pages 1405–1410. http://www.aclweb.org/anthology/D13-1143.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics (TACL)* https://aclweb.org/anthology/Q16-1023.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. pages 673–682. http://www.aclweb.org/anthology/P11-1068.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. pages 523–530. http://www.aclweb.org/anthology/H05-1066.

Ryan T McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*. pages 122–131. http://www.aclweb.org/anthology/D07-1013.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*. pages 149–160. http://stp.lingfil.uu.se/ nivre/docs/iwpt03.pdf.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. pages 50–57. https://www.aclweb.org/anthology/W04-0308.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. pages 1659–1666.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *Association for Computational Linguistics*. pages 135–144. http://www.aclweb.org/anthology/P13-1014.

Michael Schmitz, Robert Bart, Stephen Soderland, Oren Etzioni, et al. 2012. Open language learning for information extraction. In *Proceedings*

of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. pages 523–534. https://www.aclweb.org/anthology/D12-1048.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*. http://www.aclweb.org/anthology/P15-1032.

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).[8] 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_{head} and dependent representation v_{dep}, and perform the following biaffine combination

$$\begin{aligned} v_{\text{feat},i} &= [f(v_{\text{head}}, v_{\text{dep}})]_i \\ &= \text{ReLU}\big(v_{\text{head}}^\top W_i v_{\text{dep}} + b_i^\top v_{\text{head}} \\ &\qquad + c_i^\top v_{\text{dep}} + d_i\big) \end{aligned} \tag{1}$$

where $W_i \in \mathbb{R}^{32\times32}$, $b_i, c_i \in \mathbb{R}^{32}$, and d_i is a scalar for $i = 1, \ldots, 32$. The resulting 32-dimensional features are concatenated as the input

[8] We also kept the vectors of the top 400k words trained on Wikipedia and English Gigaword for a broader coverage of unseen words.

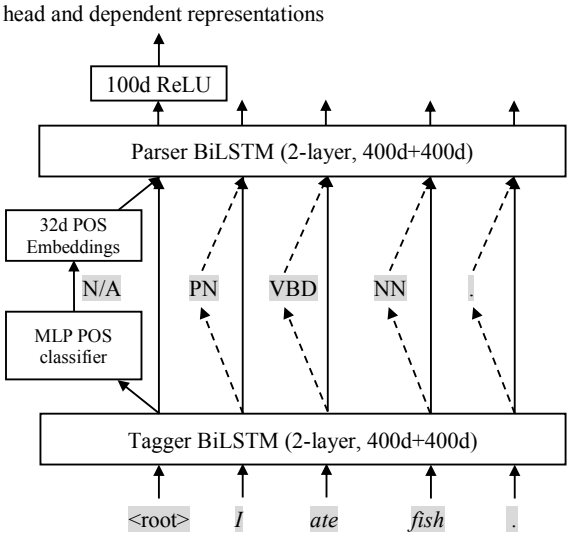

Figure 5: Illustration of the model.

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_{head} of the head token and v_{dep} of the dependent token of the arc to be induced. For Shift, we simply combine v_{head} and v_{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.