Sequential Graph Dependency Parser

Sean Welleck
New York University
wellecks@nyu.edu

Kyunghyun Cho
New York University
CIFAR Azrieli Global Scholar
Facebook AI Research
kyunghyun.cho@nyu.edu

Abstract

We propose a method for non-projective dependency parsing by incrementally predicting a set of edges. Since the edges do not have a pre-specified order, we propose a set-based learning method. Our method blends graph, transition, and easy-first parsing, including a prior state of the parser as a special case. The proposed transition-based method successfully parses near the state of the art on both projective and non-projective languages, without assuming a certain parsing order.

1 Introduction

Dependency parsing methods can be categorized as graph-based and transition-based. Typical graph-based methods support non-projective parsing, but introduce independence assumptions and rely on external decoding algorithms. Conversely, transition-based methods model joint dependencies, but without modification are typically limited to projective parses.

There are two recent exceptions of interest here. (Ma et al., 2018) developed the Stack-Pointer parser, a transition-based, non-projective parser that maintains a stack populated in a top-down, depth-first manner, and uses a pointer network to determine the dependent of the stack's top node, resulting in a transition sequence of length $2n-1$. Recently, (Fernández-González and Gómez-Rodrguez, 2019) developed a variant of the Stack-Pointer parser which parses in n steps by traversing the sentence left-to-right, selecting the *head* of the current node in the traversal, while incrementally checking for, and prohibiting, cycles.

We take inspiration from both graph-based and transition-based approaches by viewing parsing as sequential graph generation. In this view, a graph is incrementally built by adding edges to an edge set. No distinction between projective and non-projective trees is necessary. Since edges do not have a pre-specified order, we propose a set-based learning method. Like (Fernández-González and Gómez-Rodrguez, 2019), our parser runs in n steps. However, our learning method and transitions do not impose a left-to-right parsing order, allowing easy-first (Tsuruoka and Tsujii, 2005; Goldberg and Elhadad, 2010) behavior. Experimentally, we find that the proposed method can yield a sequential parser with preferred, input-dependent generation orders and performance gains over strong one-step methods.[1]

2 Graph Dependency Parser

Given a sentence $x = x_1, \ldots, x_N$, a dependency parser constructs a graph $G = (V, E)$ with $V = (x_0, x_1, \ldots, x_N)$ and $E = \{(i,j)_1, \ldots (i,j)_N\}$, where x_0 is a special root node, and $E \subset \mathcal{E}$ forms a dependency tree.[2]

We describe a family of sequential graph-based dependency parsers. A parser in this family generates a *sequence* of graphs where V is fixed and $E = \bigcup_{t=1}^{T} E_t$:

$$H^{\text{enc}} = f_{\text{enc}}(x_0, \ldots x_N) \quad (1)$$

$$H_t^{\text{head}}, H_t^{\text{dep}} = f_{\text{V}}(H^{\text{enc}}, E_{<t}, h_{t-1}) \quad (2)$$

$$S_t = f_{\text{E}}(H_t^{\text{head}}, H_t^{\text{dep}}, S_{t-1}) \quad (3)$$

$$E_t = f_{\text{dec}}(S_t, E_{<t}). \quad (4)$$

Steps (2-4) run for $T \le N$ time-steps. At each time-step, first f_V generates head and dependent

[1] Code will be made available at https://github.com/wellecks/nonmonotonic_parsing.

[2] See properties (1-5) in Appendix A.

representations for each vertex, $H_t^{\cdot} \in \mathbb{R}^{V \times d_H}$, based on vertex representations $H^{\text{enc}} \in \mathbb{R}^{V \times d}$, previously predicted edges $E_{<t}$, and a recurrent state $h_{t-1} \in \mathbb{R}^d$. Then f_E computes a score for every possible edge, $S_t \in \mathbb{R}^{V \times V}$, and the scores are used by f_{dec} to predict a set of edges E_t.

This general sequential family includes the biaffine parser of (Dozat and Manning, 2017) as a one-step special case, as well as a recurrent variant which we discuss below.

2.1 Biaffine One-Step

The Biaffine parser of (Dozat and Manning, 2017) is a one-step variant, implementing steps (1-4) using a bidirectional LSTM, head and dependent neural networks, a biaffine scorer, and a maximum spanning tree decoder, respectively:

$$\begin{aligned} H^{\text{enc}} &= \text{BiLSTM}(x_1, \ldots, x_N) \\ H^{\text{head}}, H^{\text{dep}} &= \text{MLP}^{\text{h}}(H^{\text{enc}}), \text{MLP}^{\text{d}}(H^{\text{enc}}) \\ S &= \text{BiAffine}(H^{\text{head}}, H^{\text{dep}}) \\ E &= \text{MST}(S), \end{aligned}$$

where each row of scores $S^{(i)}$ is interpreted as a distribution over i's potential head nodes:

$$p((j \to i)|x) \propto \text{softmax}_j(S^{(i)}),$$

and $\text{MST}(\cdot)$ is an off-the-shelf maximum-spanning-tree algorithm. This model assumes conditional independence of the edges.

2.2 Recurrent Weight

We propose a variant which iteratively adjusts a distribution over edges at each step, based on the predictions so far. A recurrent function generates a weight matrix W which is used to form vertex embeddings and in turn adjust edge scores.

Specifically, we first obtain an initial score matrix S_0 using the biaffine one-step parser (2.1), and initialize a recurrent hidden state h_0 using a linear transformation of f_{enc}'s final hidden state. Then f_V is defined as:

$$\begin{aligned} W, h_t &= \text{LSTM}(f_{\text{emb}}(E_{t-1}), h_{t-1}) \\ H_t^{\text{head}} &= \text{emb}_h(0, \ldots, N)W \\ H_t^{\text{dep}} &= \text{emb}_d(0, \ldots, N)W, \end{aligned}$$

and $f_{\text{E}}(H_t^{\text{head}}, H_t^{\text{dep}}, S_{t-1})$ is defined as:

$$\begin{aligned} S_t^{\Delta} &= \text{BiAffine}(H_t^{\text{head}}, H_t^{\text{dep}}) \\ S_t &= S_{t-1} + S_t^{\Delta}, \end{aligned}$$

where t ranges from 1 to N, $W \in \mathbb{R}^{d_{\text{emb}} \times d_H}$, and each $\text{emb}_{(\cdot)} : \mathbb{N} \to \mathbb{R}^{d_{\text{emb}}}$ is a learned embedding layer, yielding $\text{emb}_{(\cdot)}(0, \ldots, N)$ in $\mathbb{R}^{V \times d_{\text{emb}}}$. We use a bidirectional LSTM as f_{enc}.

The scores at each step yield a distribution over all $V \times V$ edges, which we denote by π:

$$\pi((i \to j)|E_{<t}, x) \propto \text{softmax}(\text{flatten}(S_t)). \quad (5)$$

Unlike the one-step model, this recurrent model can predict edges based on past predictions.

Inference We must ensure the incrementally decoded edges $E = \bigcup_{t=1}^{T} E_t$ form a valid dependency tree. To do so, we choose f_{dec} to be a decoder which greedily selects valid edges,

$$E_t = f_{\text{valid}}(S_t, E_{<t}),$$

which we refer to as the **valid decoder**, detailed in Appendix A. We only predict one edge per step ($|E_t| = 1$), leaving the setting of multiple predictions per step as future work.

Embedding Edges We embed a predicted edge $E_t = \{(\hat{i, j})\}$ as:

$$\begin{aligned} f_{\text{emb}}(E_t) &= e_{\text{edge}}; e_{\text{head}}; e_{\text{dependent}} \\ e_{\text{edge}} &= W_e H_{(i)}^{\text{enc}} - W_e H_{(j)}^{\text{enc}} \\ e_{\text{head}} &= \text{emb}_h(i) \\ e_{\text{dependent}} &= \text{emb}_d(j), \end{aligned}$$

where $H_{(\cdot)}^{\text{enc}} \in \mathbb{R}^d$ are row vectors, $W_e \in \mathbb{R}^{d_e \times d}$ is a learned weight matrix, $\text{emb}_{(\cdot)}$ are learned embedding layers, and ; is concatenation.

Future Work The proposed method does not specifically require a BiLSTM encoder, LSTM, or the BiAffine function. For instance, f_{V} could use a Transformer (Vaswani et al., 2017) to output states that are linearly transformed into H^{head} and H^{dep}. Additionally, partial graphs $(V, E_{<t})$ might be embedded using neural networks specifically designed for graphs (Gilmer et al., 2017). Finally, predicting edge sets of size greater than 1 could potentially be achieved using a partially-autoregressive model, trained with a 'masked

edges' objective, similar to recent work in machine translation with conditional masked language models (Ghazvininejad et al., 2019). Each call to f_V would involve a separate forward pass which calls a Transformer f_{enc}. The partial tree is encoded via non-masked inputs to f_{enc}. f_{E} corresponds to having V outputs, each a distribution over V edges. The multi-step decoder (Appendix A) might be used at test time.

3 Learning

In this paper, we restrict to the case of predicting a single edge $(\widehat{i,j})$ per step, so that the recurrent weight model generates a sequence of edges with the goal of matching a target edge set, i.e. $\bigcup_{t=1}^{N} (\widehat{i,j})_t = E$. Since the target edges E are a set, the model's generation order is not determined *a priori*. As a result, we propose to use a learning method that does not require a pre-specified generation order and allows the model to learn input-dependent orderings.

Our proposed method is based on the multiset loss (Welleck et al., 2018) and its recent extensions for non-monotonic generation (Welleck et al., 2019). The method is motivated from the perspective of learning-to-search (Daumé III et al., 2009; Chang et al., 2015), which involves learning a *policy* π_θ that mimics an *oracle policy* π^*. The policy maps *states* to distributions over *actions*.

For the proposed graph parser, an action is an edge $(i,j) \in \mathcal{E}$, and a state s_t is an input sentence x along with the edges predicted so far, $\hat{E}_{<t}$. The policy is a conditional distribution over $\mathcal{E}$,

$$\pi_\theta((i,j)|\hat{E}_{<t}, x),$$

such as the distribution in equation (5).

Learning consists of minimizing a cost, computed by first sampling states from a *roll-in* policy π^{in}, then using a *roll-out policy* π^{out} to estimate cost-to-go for all actions at the sampled states. Formally, we minimize the following objective with respect to θ:

$$\mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{s_1,\ldots,s_{|x|}\sim\pi^{\text{in}}}\mathcal{C}(\pi_\theta, \pi^{\text{out}}, s_t). \tag{6}$$

This objective involves sampling a sentence x from a dataset, sampling a sequence of edges from the roll-in policy, then computing a cost $\mathcal{C}$ at each of the resulting states. We now describe our choices of $\mathcal{C}$, π^{out}, π^*, and π^{in}, and evaluate them later in the experiments (4).

3.1 Cost Function and Roll-Out

Following (Welleck et al., 2018, 2019) we use a KL-divergence cost:

$$\mathcal{C}(\pi_\theta, \pi^{\text{out}}, s) = D_{\text{KL}}(\pi^{\text{out}}(\cdot|s)||\pi_\theta(\cdot|s)). \tag{7}$$

We use the oracle π^* as the roll-out π^{out}.

3.2 Oracle

Based on the free labels set in (Welleck et al., 2018), we first define a *free edge set* containing the un-predicted target edges at time t:

$$E^t_{\text{free}} = E \setminus \bigcup_{t'=1}^{t-1} (\widehat{i,j})_{t'}, \tag{8}$$

where $E^0_{\text{free}} = E$. We then construct a family of oracle policies that place non-zero probability mass only on free edges:

$$\pi^*((i,j)|E^t_{\text{free}}) = \begin{cases} p_{ij} & (i,j) \in E^t_{\text{free}} \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

We now describe several oracles by varying how p_{ij} is defined.

Uniform This oracle treats each permutation of the target edge set as equally likely by assigning a uniform probability to each free edge:

$$\pi^*_{\text{unif}}((i,j)|E^t_{\text{free}}) = \begin{cases} \frac{1}{|E^t_{\text{free}}|} & (i,j) \in E^t_{\text{free}} \\ 0 & \text{otherwise.} \end{cases}$$

Coaching Motivated by (He et al., 2012; Welleck et al., 2019), we define a coaching oracle which weights free edges by π_θ:

$$\pi^*_{\text{coaching}}((i,j)|E^t_{\text{free}}) \propto \pi^*_{\text{unif}}(\cdot|E^t_{\text{free}})\pi_\theta(\cdot|E_{<t}, X).$$

This oracle prefers certain edge permutations over others, reinforcing π_θ's preferences. The coaching and uniform oracles can be mixed to ensure each free edge receives probability mass:

$$\beta\pi^*_{\text{unif}} + (1-\beta)\pi^*_{\text{coaching}}, \tag{10}$$

where $\beta \in [0, 1]$.

Annealed Coaching This oracle begins with the uniform oracle, then anneals towards the coaching oracle as training progresses by annealing the β term in (10). This may prevent the coaching oracle from reinforcing sub-optimal permutations early in training.

Linearized This oracle uses a deterministic function to linearize an edge set E into a sequence E_{seq}. The oracle selects the t'th element of E_{seq} at time t with probability 1. We linearize an edge set in increasing edge-index order: (i_1, j_1) precedes (i_2, j_2) if $(i_1, j_1) < (i_2, j_2)$. This oracle serves as a baseline that is analogous to the fixed generation orders used in conventional parsers.

3.3 Roll-In

The roll-in policy determines the state distribution that π_θ is trained on, which can address the mismatch between training and testing state distributions (Ross et al., 2011; Chang et al., 2015) or narrow the set of training trajectories. We evaluate several alternatives:

1. **uniform** $(i, j) \sim \pi^*_{\text{unif}}$
2. **coaching** $(i, j) \sim \pi_\theta \odot \pi^*_{\text{unif}}$
3. **valid-policy** $(i, j) \sim \text{valid}(\pi_\theta)$

where valid(π_θ) is the set of edges that keeps the predicted tree as a valid dependency tree. The coaching and valid-policy roll-ins choose edge permutations that are preferred by the policy, with valid-policy resembling test-time behavior.

4 Experiments

In Experiments 4.1 and 4.2 we evaluate on English, German, Chinese, and Ancient Greek since they vary with respect to projectivity, size, and performance in (Qi et al., 2018). Based on these development set results, we then test our strongest model on a large suite of languages (4.3).

Experimental Setup Experiments are done using datasets from the CoNLL 2018 Shared Task (Zeman et al., 2018). We build our implementation from the open-source version of (Qi et al., 2018)[3], and use their experimental setup (e.g.

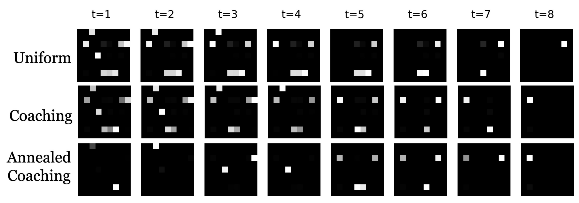

Figure 1: Per-step edge distributions from recurrent weight models trained with the given oracle.

pre-processing, data-loading, pre-trained vectors, evaluation) which follows the shared task setup. Our model uses the same encoder from (Qi et al., 2018). For the (Qi et al., 2018) baseline, we use their pretrained models[4] and evaluation script. For the (Dozat and Manning, 2017) baseline, we use the (Qi et al., 2018) implementation with auxiliary outputs and losses disabled, and train with the default hyper-parameters and training script. For our models only, we changed the learning rate schedule (and model-specific hyper-parameters), after observing diverging loss in preliminary experiments with the default learning rate. Our models did not require the additional AMSGrad technique used in (Qi et al., 2018). We evaluate validation UAS every 2k steps (vs. 100 for the baseline). Models are trained for up to 100k steps, and the model with the highest validation unlabeled attachment score (UAS) is saved.

4.1 Multi-Step Learning

In this experiment we evaluate the sequential aspect of the proposed recurrent model by comparing it with one-step baselines. We compare against a baseline ('One-Step') that simply uses the first step's score matrix S_0 from the recurrent weight model and minimizes (6) for one time-step using a uniform oracle. At test time the valid decoder uses S_0 for all timesteps. We also compare against the biaffine one-step model of (Dozat and Manning, 2017) which uses Chu-Liu-Edmonds maximum spanning tree decoding instead of valid decoding. Since we only evaluate UAS, we disable its edge label output and loss. Finally, we compare against (Qi et al., 2018) which is based on (Dozat and Manning, 2017) plus auxiliary losses for length and linearization prediction.

Results are shown in Table 1, including results for

[3]`https://github.com/stanfordnlp/stanfordnlp`.

[4]`https://stanfordnlp.github.io/stanfordnlp/installation_download.html`.

	En	De	Grc	Zh
D & M (2017)	91.14	90.38	78.99	86.50
Qi et al. (2018)	92.11	89.46	81.35	86.73
One-Step	91.74	91.07	79.60	86.61
Recurrent (U)	91.92	91.02	79.15	86.69
Recurrent (C)	91.99	91.19	79.93	86.77

Table 1: Development set UAS for single vs. multi-step methods. (U) is uniform oracle and roll-in, (C) is coaching with greedy valid roll-in ($\beta = 0.5$). D & M (2017) is an abbreviation for (Dozat and Manning, 2017).

a recurrent model trained with coaching ('Recurrent (C)') using a mixture (eq. 10) with $\beta = 0.5$. The one-step baseline is strong, even outperforming the uniform recurrent variant on some languages. The recurrent weight model with coaching, however, outperforms the one-step and (Dozat and Manning, 2017) baselines on all four languages. Adding in auxiliary losses to the (Dozat and Manning, 2017) model yields improved UAS as seen in the (Qi et al., 2018) performance, suggesting that our proposed recurrent model might be improved further with auxiliary losses.

Temporal Distribution Adjustment Figure 1 shows per-step edge distributions on an eight-edge example. The recurrent weight variants learned to adjust their distributions over time based on past predictions. The model trained with the uniform oracle has a decreasing number of high probability edges per step since it aims to place equal mass on each free edge $(i, j) \in \hat{E}^t_{\text{free}}$. The model trained with coaching learned to prefer certain free edges over others, but with $\beta = 0.5$ the uniform term in the loss still encourages placing mass on multiple edges per step. By annealing β, however, the coaching model exhibits vastly different behavior than the uniform-trained policy. The low entropy distributions at early steps followed by higher entropy distributions later on (e.g. $t \in \{5, 6\}$) may indicate easy-first behavior.

4.2 Oracle and Roll-In Choice

In this experiment, we study the effects of varying the oracle and roll-in distributions. Table (2) shows results on German, analyzed below. Models trained with coaching (C) use a mixture with $\beta = 0.5$, after observing lower UAS in preliminary experiments with lower β. The π^*_{coach} and $\pi^{\text{valid}}_{\theta}$ roll-ins use a mixture with $\beta = 0.5$ and greedy decoding, which generally outperformed stochastic sampling.

Oracle	Roll-in	UAS	Loss
Linear	π^*_{linear}	81.03	0.04
U	π^*_{unif}	91.02	0.35
C	π^*_{unif}	91.04	0.17
U	π^*_{coach}	90.93	0.45
C	π^*_{coach}	91.17	0.33
CA	π^*_{coach}	90.89	0.34
U	$\pi^{\text{valid}}_{\theta}$	90.99	0.51
C	$\pi^{\text{valid}}_{\theta}$	**91.19**	0.31
CA	$\pi^{\text{valid}}_{\theta}$	90.91	0.30

Table 2: Varying oracle and roll-in policies on German. (U), (C), (A) refer to uniform, coaching, and annealing, respectively. The π^*_{coach} and $\pi^{\text{valid}}_{\theta}$ roll-ins are mixtures with a uniform oracle, with $\beta = 0.5$ for coaching (C), and β linearly annealed by 0.02 every 2000 steps for annealing (CA).

Set-Based Learning The model trained with the linearized oracle (UAS 81.03), which teaches the model to adhere to a pre-specified generation order, significantly under-performs the set-based models (UAS $\geq$ 90.89), which do not have a pre-specified generation order and can in principle learn strategies such as easy-first.

Coaching Models trained with coaching (C, UAS $\geq$ 91.04) had higher UAS and lower loss than models trained with the uniform oracle (U, UAS $\leq$ 91.02), for all roll-in methods. This suggests that for the proposed model, weighting free edges in the loss based on the model's distribution is more effective than a uniform weighting.

Annealing the β parameter generally did not further improve UAS (CA vs. C), possibly due to the annealing schedule or overfitting; despite lower losses with annealing, eventually validation UAS *decreased* as training progressed.

Roll-In With the coaching oracle (C), the choice of roll-in impacted UAS, with coaching roll-in (π^*_{coach}, 91.17) and valid roll-in ($\pi^{\text{valid}}_{\theta}, 91.19$) achieving higher UAS than uniform oracle roll-in ($\pi^*_{\text{unif}}, 91.04$). This suggests that when using coaching, narrowing the set of training trajectories

	Ours	Qi et al. (2018)
AR	88.22	**88.35**
CA	**94.13**	**94.13**
CS (CAC)	**93.53**	93.22
CS (PDT)	**93.80**	93.21
DE	**88.39**	87.21
EN (EWT)	**91.28**	91.21
ES	**93.70**	93.38
ET	**89.56**	89.40
FR (GSD)	**91.07**	90.90
GRC (Perseus)	80.90	**82.77**
HI	**96.78**	**96.78**
IT (ISDT)	94.06	**94.24**
KO (KAIST)	**91.02**	90.55
LA (ITTB)	**93.66**	93.00
NO (Bokmaal)	**94.63**	94.27
NO (Nynorsk)	**94.44**	94.02
PT	91.22	**91.67**
RU (SynTagRus)	**94.57**	94.42
ZH	87.31	**88.49**

Table 3: Test set results (UAS) on datasets from the CoNLL 2018 shared task with greater than 200k examples, plus the Ancient Greek (GRC) and Chinese (ZH) datasets. Bold denotes the highest UAS on each dataset.

to those preferred by the policy may be more effective than sampling uniformly from the set of all correct trajectories. Based on these results, we use the coaching oracle and valid roll-in for training our final model in the next experiment.

4.3 CoNLL 2018 Comparison

In this experiment, we evaluate our best model on a diverse set of multi-lingual datasets. We use the CoNLL 2018 shared task datasets that have at least 200k examples, along with the four datasets used in the previous experiments. We train a recurrent weight model for each dataset using the coaching oracle and valid roll-in. We compare against (Qi et al., 2018) which placed highly in the CoNLL 2018 competition, reporting test UAS evaluated using their pre-trained models.

Table 3 shows the results on the 19 datasets from 17 different languages. The proposed model trained with coaching achieves a higher UAS than the Qi et al. (2018) model on 12 of the 19 datasets, plus two ties.

5 Related Work

Transition-based dependency parsing has a rich history, with methods generally varying by the choice of transition system and feature representation. Traditional stack-based arc-standard and arc-eager (Yamada and Matsumoto, 2003; Nivre, 2003) transition systems only parse projectively, requiring additional operations for pseudo-non-projectivity (Gómez-Rodríguez et al., 2014) or projectivity (Nivre, 2009), while list-based non-projective systems have been developed (Nivre, 2008). Recent variations assume a generation order such as top-down (Ma et al., 2018) or left-to-right (Fernández-González and Gómez-Rodrguez, 2019). Other recent models focus on unsupervised settings (Kim et al., 2019). Our focus here is a non-projective transition system and learning method which does not assume a particular generation order.

A separate thread of research in sequential modeling has demonstrated that generation order can affect performance (Vinyals et al., 2015), both in tasks with set-structured outputs such as objects (Welleck et al., 2017, 2018) or graphs (Li et al., 2018), and in sequential tasks such as language modeling (Ford et al., 2018). Developing models with relaxed or learned generation orders has picked up recent interest (Welleck et al., 2018, 2019; Gu et al., 2019; Stern et al., 2019). We investigate this for dependency parsing, framing the problem as sequential set generation without a pre-specified order.

Finally, our work is inspired by techniques for improving upon maximum likelihood training through error exploration and dynamic oracles (Goldberg and Nivre, 2012, 2013), and related techniques in imitation learning for structured prediction (Daumé III et al., 2009; Ross et al., 2011; He et al., 2012; Goodman et al., 2016). In particular, our formulation is closely related to the framework of (Chang et al., 2015), where our oracle can be seen as an optimal roll-out policy which computes action costs without explicit roll-outs.

6 Conclusion

We described a family of dependency parsers which construct a dependency tree by generating a sequence of edge sets, and a learning method that does not presuppose a generation order. Experimentally, we found that a ‘coaching’ method, which weights actions in the loss according to the model, improves parsing accuracy compared to a uniform weighting and allows the parser to learn preferred, input-dependent generation orders. The model’s sequential aspect, along with the coaching method and training on a state distribution which resembles the model’s own behavior, yielded improvements in unlabeled dependency parsing over strong one-step baselines.

References

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based Structured Prediction. Technical report.

Timothy Dozat and Christopher D Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *International Conference on Learning Representations (ICLR)*.

Daniel Fernández-González and Carlos Gómez-Rodrguez. 2019. Left-to-right dependency parsing with pointer networks.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. 2018. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Constant-time machine translation with conditional masked language models.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural Message Passing for Quantum Chemistry.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. Technical report.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927. Association for Computational Linguistics.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–11, Berlin, Germany. Association for Computational Linguistics.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019. Insertion-based decoding with automatically inferred generation order.

He He, Jason Eisner, and Hal Daume. 2012. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised Recurrent Neural Network Grammars.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. In *ICML 2018*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-Pointer Networks for Dependency Parsing. Technical report.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT*, pages 149–160, Nancy, France.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359. Association for Computational Linguistics.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 467–474. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order Matters: Sequence to sequence for sets.

Sean Welleck, Kiant Brantley, Hal Daum III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation.

Sean Welleck, Kyunghyun Cho, and Zheng Zhang. 2017. Saliency-based sequential image attention with multiset prediction. In *Advances in neural information processing systems*.

Sean Welleck, Zixin Yao, Yu Gai, Jialin Mao, Zheng Zhang, and Kyunghyun Cho. 2018. Loss functions for multiset prediction. In *Advances in Neural Information Processing Systems*, pages 5788–5797.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

A Sequential Valid Decoder

We wish to sequentially sample $E_1, E_2, \ldots, E_T$ from score matrices $S_1, S_2, \ldots, S_T$, respectively, such that $E = \bigcup_t E_t$ is a dependency tree. A dependency tree must satisfy:

1. The root node has no incoming edges.
2. Each non-root node has exactly one incoming edge.
3. There are no duplicate edges.
4. There are no self-loops.
5. There are no cycles.

We first consider predicting one edge per step $|E_t| = 1$, then address the case $|E_t| \geq 1$.

One Edge Per Step Let $x = x_0, x_1, \ldots, x_N$ where x_0 is a root node. We define a function $f_{\text{valid}}(S_t, E_{<t}) \rightarrow (i, j)$ which chooses the highest scoring edge (i, j) such that $E_{<t} \cup \{(i, j)\}$ is a dependency tree, given edges $E_{<t}$ and scores S_t. We represent $E_{<t}$ as an adjacency matrix $A_{<t}$, and implement $f_{\text{valid}}(S_t, A_{<t})$ by masking S_t to yield scores $\tilde{S}$ that satisfy (1-5) as follows:

1. $\tilde{S}_{\cdot,0} = -\infty$
2. $A_{i,j} = 1$ implies $\tilde{S}_{\cdot,j} = -\infty$
3. $A_{i,j} = 1$ implies $\tilde{S}_{i,j} = -\infty$
4. $\tilde{S}_{i,i} = -\infty$ for all i
5. $R_{i,j} = 1$ implies $\tilde{S}_{j,i} = -\infty$, where $R \in \{0,1\}^{N \times N}$ is the reachability matrix (transitive closure) of A. That is, $R_{i,j} = 1$ when there is a directed path from i to j. [5]

The selected edge is then $\arg\max_{(i,j)} \tilde{S}_{i,j}$.

A full tree is decoded by calling f_{valid} for T steps, using the current step scores S_t and an adjacency matrix $A_{<t} = \bigcup_{t'=1}^{t-1} \{(i, j)_{t'}\}$.

Multiple Edges Per Step To decode multiple edges per step, i.e. $|E_t| \geq 1$, we propose to repeatedly call f_{valid}, adding the returned edge to the adjacency matrix after each call, and stopping once the returned edge's score is below a pre-defined threshold τ.

[5]The reachability matrix R can be computed with batched matrix multiplication as $\sum_{k=1}^{t} A^k$ where t is the maximum path length; other methods could potentially improve speed.