Efficient Second-Order TreeCRF for Neural Dependency Parsing

Yu Zhang, Zhenghua Li*, Min Zhang
School of Computer Science and Technology, Soochow University, China
yzhang.cs@outlook.com
{zhli13,minzhang}@suda.edu.cn

Abstract

In the deep learning (DL) era, parsing models are extremely simplified with little hurt on performance, thanks to the remarkable capability in context representation of multi-layer BiLSTM encoders. As the most popular graph-based dependency parser due to its high efficiency and performance, the biaffine parser directly scores single dependencies under the arc-factorization assumption, and adopts a very simple local token-wise cross-entropy training loss. This paper for the first time presents a second-order TreeCRF extension to the biaffine parser. Through experiments and analysis on 15 datasets from 13 languages, we endeavour to answer the question: are techniques developed before the DL era such as structural learning and high-order modeling still useful, and if so, how? A key weakness that hinders the popularity of TreeCRF is the complexity and inefficiency of the inside-outside algorithm. Therefore, the second question addressed in this work is: can we batchify the TreeCRF loss computation via direct large matrix operation on GPU? For both questions, this paper presents very interesting and positive results.

1 Introduction

As a fundamental task in NLP, dependency parsing has attracted a lot of research interest due to its simplicity and multi-lingual applicability in capturing both syntactic and semantic information (Nivre et al., 2016). Given an input sentence $\boldsymbol{x} = w_0 w_1 \dots w_n$, a dependency tree, as depicted in Figure 1, is defined as $\boldsymbol{y} = \{(i, j, l), 0 \leq i \leq n, 1 \leq j \leq n, l \in \mathcal{L}\}$, where (i, j, l) is a dependency from the head word w_i to the modifier word w_j with the relation label $l \in \mathcal{L}$. Between two mainstream approaches, this work focuses on the graph-based paradigm (vs. transition-based).

*Corresponding author

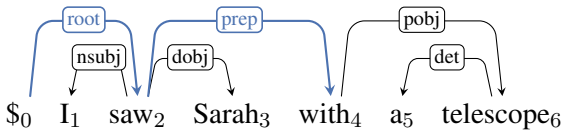

Figure 1: An example full dependency tree. In the case of partial annotation, only a part of dependencies are annotated, for example the two thick (blue) arcs.

Before the deep learning (DL) era, graph-based parsing relies on many hand-crafted features and differs from its neural counterpart in two major aspects. First, structural learning, i.e., explicit awareness of tree structure constraints during training, is indispensable. Most non-neural graph-based parsing adopt the max-margin training algorithm, which first predicts a highest-scoring *tree* with the current model, and then updates feature weights so that the correct tree has a higher score than the predicted tree.

Second, high-order modeling brings significant accuracy gains. The basic first-order model factors the score of a tree into independent scores of single dependencies (McDonald et al., 2005). Second-order models were soon proposed to incorporate scores of dependency pairs, such as adjacent-siblings (McDonald and Pereira, 2006) and grand-parent-child (Carreras, 2007; Koo and Collins, 2010), showing significant accuracy improvement yet with the cost of lower efficiency and more complex decoding algorithms.[1]

In contrast, neural graph-based dependency parsing exhibits an opposite development trend. Pei et al. (2015) proposed to use feed-forward neural networks for automatically learning combinations of dozens of atomic features similar to Chen and Manning (2014), and for computing subtree

[1]Third-order and fourth-order models shows little accuracy improvement probably due to the feature sparseness problem (Koo and Collins, 2010; Ma and Zhao, 2012).

scores. They showed that incorporating second-order scores of adjacent-sibling subtrees significantly improved performance. Then, both Wang and Chang (2016) and Kiperwasser and Goldberg (2016) proposed to utilize BiLSTM as an encoder and use minimal feature sets for scoring single dependencies in a first-order parser. These three representative works all employ global max-margin training. Dozat and Manning (2017) proposed a strong and efficient biaffine parser and obtained state-of-the-art accuracy on a variety of datasets and languages. The biaffine parser is also first-order and employs simpler and more efficient non-structural training via local head selection for each token (Zhang et al., 2017).

Observing such contrasting development, we try to make a connection between pre-DL and DL techniques for graph-based parsing. Specifically, **the first question** to be addressed in this work is: *can previously useful techniques such as structural learning and high-order modeling further improve the state-of-the-art*[2] *biaffine parser, and if so, in which aspects are they helpful?*

For structural learning, we focus on the more complex and less popular TreeCRF instead of max-margin training. The reason is two-fold. First, estimating probability distribution is the core issue in modern data-driven NLP methods (Le and Zuidema, 2014). The probability of a tree ($p(\boldsymbol{y}|\boldsymbol{x})$) is potentially more useful than an unbounded score ($s(\boldsymbol{x}, \boldsymbol{y})$) for high-level NLP tasks when utilizing parsing outputs. Second, as a theoretically sound way to measure model confidence of subtrees, marginal probabilities can support Minimum Bayes Risk (MBR) decoding (Smith and Smith, 2007), and are also proven to be crucial for the important research line of token-level active learning based on partial trees (Li et al., 2016).

One probable reason for the less popularity of TreeCRF, despite its usefulness, is due to the complexity and inefficiency of the inside-outside algorithm, especially the outside algorithm. As far as we know, all existing works compute the inside and outside algorithms on CPUs. The inefficiency issue becomes more severe in the DL era, due to the unmatched speed of CPU and GPU computations. This leads to **the second question**: *can we batchify the inside-outside algorithm and perform computation directly on GPUs?* In that case, we can employ efficient TreeCRF as a built-in component in DL toolkits such as PyTorch for wider applications (Cai et al., 2017; Le and Zuidema, 2014).

Overall, targeted at the above two questions, this work makes the following contributions.

- We for the first time propose second-order TreeCRF for neural dependency parsing. We also propose an efficient and effective triaffine operation for scoring second-order subtrees.
- We propose to batchify the inside algorithm via direct large tensor computation on GPUs, leading to very efficient TreeCRF loss computation. We show that the complex outside algorithm is no longer needed for the computation of gradients and marginal probabilities, and can be replaced by the equally efficient back-propagation process. We will release codes at `https://github.com/yzhangcs/crfpar`.
- We conduct experiments on 15 datasets from 13 languages. The results and analysis show that both structural learning and high-order modeling are still beneficial to the state-of-the-art biaffine parser in many ways in the DL era.

2 The Basic Biaffine Parser

We re-implement the state-of-the-art biaffine parser (Dozat and Manning, 2017) with two modifications, i.e., using CharLSTM word representation vectors instead of POS tag embeddings, and the first-order Eisner algorithm (Eisner, 2000) for projective decoding instead of the non-projective MST algorithm.

Scoring architecture. Figure 2 shows the scoring architecture, composing of four layers.

Input vectors. The ith input vector is composed of two parts: the word embedding and the CharLSTM word representation vector of w_i.

$$\mathbf{e}_i = \mathrm{emb}(w_i) \oplus \mathrm{CharLSTM}(w_i) \quad (1)$$

where $\mathrm{CharLSTM}(w_i)$ is obtained by feeding w_i into a BiLSTM and then concatenating the two last hidden vectors (Lample et al., 2016). We find that replacing POS tag embeddings with $\mathrm{CharLSTM}(w_i)$ leads to consistent improvement, and also simplifies the multilingual experiments by avoiding POS tag generation (especially n-fold jackknifing on training data).

[2]Though many recent works report higher performance with extra resources, for example contextualized word representations learned from large-scale unlabeled texts under language model loss, they either adopt the same architecture or achieve similar performance under fair comparison.

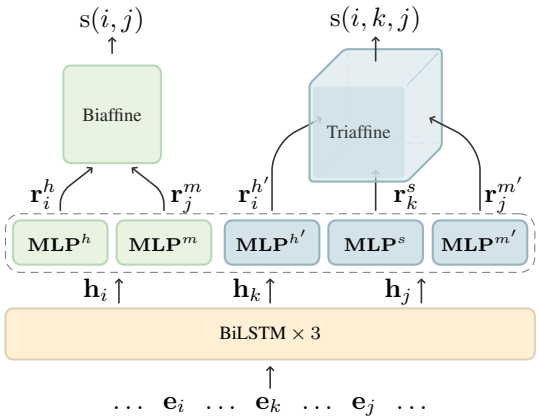

Figure 2: Scoring architecture with second-order extension.

BiLSTM encoder. To encode the sentential contexts, the parser applies three BiLSTM layers over $\mathbf{e}_0 \dots \mathbf{e}_n$. The output vector of the top-layer BiLSTM for the ith word is denoted as $\mathbf{h}_i$.

MLP feature extraction. Two shared MLPs are applied to each $\mathbf{h}_i$, obtaining two lower-dimensional vectors that detain only syntax-related features:

$$\mathbf{r}_i^h; \mathbf{r}_i^m = \mathbf{MLP}^{h/m}(\mathbf{h}_i) \tag{2}$$

where $\mathbf{r}_i^h$ and $\mathbf{r}_i^m$ are the representation vector of w_i as a head word and a modifier word respectively.

Biaffine scorer. Dozat and Manning (2017) for the first time propose to compute the score of a dependency $i \rightarrow j$ via biaffine attention:

$$s(i,j) = \begin{bmatrix} \mathbf{r}_j^m \\ 1 \end{bmatrix}^{\mathrm{T}} \mathbf{W}^{biaffine} \mathbf{r}_i^h \tag{3}$$

where $\mathbf{W}^{biaffine} \in \mathbb{R}^{d \times d}$. The computation is extremely efficient on GPUs.

Local token-wise training loss. The biaffine parser adopts a simple non-structural training loss, trying to independently maximize the local probability of the correct head word for each word. For a gold-standard head-modifier pair (w_i, w_j) in a training instance, the cross-entropy loss is

$$L(i,j) = -\log \frac{e^{s(i,j)}}{\sum_{0 \le k \le n} e^{s(k,j)}} \tag{4}$$

In other words, the model is trained based on simple head selection, without considering the tree structure at all, and losses of all words in a mini-batch are accumulated.

Decoding. Having scores of all dependencies, we adopt the first-order Eisner algorithm of $O(n^3)$ time complexity to find the highest-scoring tree.

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}} \left[s(\boldsymbol{x}, \boldsymbol{y}) \equiv \sum_{i \rightarrow j \in \boldsymbol{y}} s(i,j) \right] \tag{5}$$

Handling dependency labels. The biaffine parser treats as two independent (training phase) and cascaded (parsing phase) tasks skeletal tree searching and labeling. This work follows this same strategy for simplicity. Please refer to Dozat and Manning (2017) for details.

3 Second-order TreeCRF

This work substantially extends the biaffine parser in two closely related aspects: using probabilistic TreeCRF for structural training and explicitly incorporating high-order subtree scores. Specifically, we further incorporate adjacent-sibling subtree scores into the basic first-order model:[3]

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i \rightarrow j \in \boldsymbol{y}} s(i,j) + \sum_{i \rightarrow \{k,j\} \in \boldsymbol{y}} s(i,k,j) \tag{6}$$

where k and j are two adjacent modifiers of i and satisfy either $i < k < j$ or $j < k < i$.

As a probabilistic model, TreeCRF computes the conditional probability of a tree as

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{s(\boldsymbol{x},\boldsymbol{y})}}{Z(\boldsymbol{x}) \equiv \sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{x})} e^{s(\boldsymbol{x},\boldsymbol{y'})}} \tag{7}$$

where $\mathcal{Y}(\boldsymbol{x})$ is the set of all legal (projective) trees for $\boldsymbol{x}$, and $Z(\boldsymbol{x})$ is commonly referred to as the normalization (or partition) term.

During training, TreeCRF employs the following structural training loss to maximize the conditional probability of the gold-standard tree $\boldsymbol{y}$ given $\boldsymbol{x}$.

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{y}) &= -\log p(\boldsymbol{y}|\boldsymbol{x}) \\ &= -s(\boldsymbol{x}, \boldsymbol{y}) + \log Z(\boldsymbol{x}) \end{aligned} \tag{8}$$

3.1 Scoring Second-order Subtrees

To avoid major modification to the original scoring architecture, we take a straightforward extension to obtain scores of adjacent-sibling subtrees. First,

[3] This work can be further extended to incorporate grand-parent-modifier subtree scores based on the viterbi algorithm of $O(n^4)$ time complexity proposed by Koo and Collins (2010), which we leave for future work.

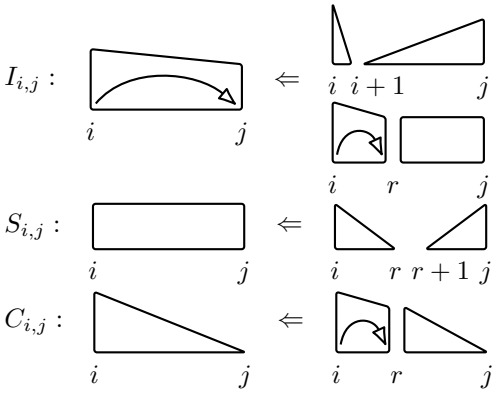

Figure 3: Diagrams of the second-order inside algorithm based on bottom-up dynamic programming.

we employ three extra MLPs to perform similar feature extraction.

$$\mathbf{r}_i^{h'}; \mathbf{r}_i^{s}; \mathbf{r}_i^{m'} = \mathbf{MLP}^{h'/s/m'}(\mathbf{h}_i) \tag{9}$$

where $\mathbf{r}_i^{h'}; \mathbf{r}_i^{s}; \mathbf{r}_i^{m'}$ are the representation vectors of w_i as head, sibling, and modifier respectively.[4]

Then, we propose a natural extension to the biaffine equation, and employ triaffine for score computation over three vectors.[5]

$$s(i,k,j) = \begin{bmatrix} \mathbf{r}_k^{s} \\ 1 \end{bmatrix}^{\mathrm{T}} \mathbf{r}_i^{h'\mathrm{T}} \mathbf{W}^{triaffine} \begin{bmatrix} \mathbf{r}_j^{m'} \\ 1 \end{bmatrix} \tag{10}$$

where $\mathbf{W}^{triaffine} \in \mathbb{R}^{d' \times d' \times d'}$ is a three-way tensor. The triaffine computation can be quite efficiently performed with the einsum function on PyTorch.

3.2 Computing TreeCRF Loss Efficiently

The key to TreeCRF loss is how to efficiently compute $\log Z(\boldsymbol{x})$, as shown in Equation 8. This problem has been well solved long before the DL era for non-neural dependency parsing. Straightforwardly, we can directly extend the viterbi decoding algorithm by replacing max product with sum product, and naturally obtain $\log Z(\boldsymbol{x})$ in the same polynomial time complexity. However, it is not enough to solely perform the inside algorithm for non-neural parsing, due to inapplicability of the automatic back-propagation mechanism, which is an obvious advantage of DL as shown in this work.

[4]Another way is to use one extra MLP for sibling representation, and re-use head and modifier representation from the basic first-order components, which however leads to inferior performance in our preliminary experiments.

[5]We have also tried the approximate method of Wang et al. (2019), which uses three biaffine operations to simulate the interactions of three input vectors, but observed inferior performance. We omit the results due to space limitation.

Algorithm 1 Second-order Inside Algorithm.

define: $I, S, C \in \mathbb{R}^{n \times n \times B}$ ▷ B is batch size
initialize: $C_{i,i} = \log e^0 = 0, 0 \le i \le n$
for $w = 1$ **to** n **do** ▷ span width
  Batchify: $0 \le i; j = i + w \le n$ (also for B)
  $I_{i,j} = \log \left(e^{C_{i,i}+C_{j,i+1}} + \sum_{i<r<j} e^{I_{i,r}+S_{r,j}+s(i,r,j)} \right)$ + s(i, j)
  $S_{i,j} = \log \sum_{i \le r < j} e^{C_{i,r}+C_{j,r+1}}$
  $C_{i,j} = \log \sum_{i < r \le j} e^{I_{i,r}+C_{r,j}}$
end for ▷ refer to Figure 3
return $C_{0,n} \equiv \log Z$

In order to obtain marginal probabilities and then feature weight gradients, we have to realize the more complex outside algorithm, which is usually at least twice slower than the inside algorithm. This may be the major reason for the less popularity of TreeCRF (vs. max-margin training) before the DL era.

As far as we know, all previous works on neural TreeCRF parsing explicitly implement the inside-outside algorithm for gradient computation (Zhang et al., 2019; Jiang et al., 2018). To improve efficiency, computation is transferred from GPUs to CPUs with Cython programming.

This work shows that the inside algorithm can be effectively batchified to fully utilize the power of GPUs. Figure 3 and Algorithm 1 together illustrate the batchified version of the second-order inside algorithm, which is a direct extension of the second-order Eisner algorithm in McDonald and Pereira (2006) by replacing max product with sum product. We omit the generations of incomplete, complete, and sibling spans in the opposite direction from j to i for brevity.

Basically, we first collect into large tensors scores of same-width spans at different positions (i, j) for all B sentences in the data batch. Then we can do computation and aggregation simultaneously on GPUs via efficient large tensor operation. Meanwhile, elaborately masking is needed to skip illegal operations not allowed in Figure 3.

Similarly, we also batchify the decoding algorithm. Due to space limitation, we omit the details.

It is noteworthy that techniques described here are also applicable to other grammar formulations such as CKY-style constituent parsing (Finkel et al., 2008; Drozdov et al., 2019).

3.3 Outside via Back-propagation

Eisner (2016) propose a theoretical proof on the

equivalence between the back-propagation mechanism and the outside algorithm in the case of constituent (phrase-structure) parsing. This work empirically verifies this equivalence for dependency parsing. We do find that gradient tensors are identical after running the two versions of codes.

Moreover, we also find that marginal probabilities $p(i \rightarrow j|\boldsymbol{x})$ directly correspond to gradients after back-propagation setting loss to $\log Z(\boldsymbol{x})$:

$$\frac{\partial \log Z}{\partial \mathrm{s}(i,j)} = \sum_{\boldsymbol{y}:(i,j)\in\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) = p(i \rightarrow j|\boldsymbol{x}) \quad (11)$$

which can be easily proved. For TreeCRF parsers, we perform MBR decoding (Smith and Smith, 2007) by replacing scores with marginal probabilities in the decoding algorithm, leading to slight but consistent accuracy increase.

3.4 Handling Partial Annotation

As an attractive research direction, studies show that it is more effective to construct or even collect partially labeled data (Nivre et al., 2014; Hwa, 1999; Pereira and Schabes, 1992), where a sentence may correspond to a partial tree $|\boldsymbol{y}^p| < n$ in the case of dependency parsing. Partial annotation can be very powerful when combined with active learning, because annotation cost can be greatly reduced if annotators only need to annotate sub-structures that are difficult for models. Li et al. (2016) presented a detailed survey on this topic. Moreover, Peng et al. (2019) recently released a partially labeled multi-domain Chinese dependency treebank based on this idea.

Then, the question is how to train models on partially labeled data. Li et al. (2016) propose to extend TreeCRF for this purpose and obtain promising results in the case of non-neural dependency parsing. This work applies their approach to the neural biaffine parser. We are particularly concerned at the influence of structural learning and high-order modeling on the utilization of partially labeled training data.

For the basic biaffine parser based on first-order local training, it seems the only choice is omitting losses of unannotated words. In contrast, tree constraints allow annotated dependencies to influence the probability distributions of unannotated words, and high-order modeling further help by promoting inter-token interaction. Therefore, both structural learning and high-order modeling are intuitively very helpful.

Under partial annotation, we follow Li et al. (2016) and define the training loss as:

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{y}^p) &= -\log \sum_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x});\boldsymbol{y}\supseteq\boldsymbol{y}^p} p(\boldsymbol{y}|\boldsymbol{x}) \\ Z(\boldsymbol{x}, \boldsymbol{y}^p) &\equiv \sum_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x});\boldsymbol{y}\supseteq\boldsymbol{y}^p} e^{s(\boldsymbol{x},\boldsymbol{y})} \\ &= -\log \frac{Z(\boldsymbol{x}, \boldsymbol{y}^p)}{Z(\boldsymbol{x})} \end{aligned} \quad (12)$$

where $Z(\boldsymbol{x}, \boldsymbol{y}^p)$ only considers all legal trees that are compatible with the given partial tree and can be efficiently computed via the inside algorithm with more masking (analogous to $Z(\boldsymbol{x})$).

4 Experiments

Data. We conduct experiments and analysis on several 15 datasets from 13 languages, including two widely used datasets: English Penn Treebank (PTB) data with Stanford dependencies (Chen and Manning, 2014), and the Chinese data at the CoNLL09 shared task (Hajič et al., 2009).

We also adopt the multi-domain Chinese dataset released at the NLPCC19 cross-domain dependency parsing shared task (Peng et al., 2019), containing one source domain and three target domains. For simplicity, we directly merge the train/dev/test data of the four domains into larger ones respectively. One characteristic of the data is that most sentences are partially annotated based on active learning. Finally, we follow Ji et al. (2019) and select 12 representative Universal Dependencies (v2.2) treebanks at the CoNLL-2018 shared task (CoNLL18) (Zeman et al., 2018).

Evaluation Metrics. We use unlabeled and labeled attachment score (UAS/LAS) as the main metrics. Punctuations are omitted for PTB. For the partially labeled NLPCC19 data, we adopt the official evaluation script, which simply omits the words without gold-standard heads to accommodate partial annotation. We adopt Dan Bikel's randomized parsing evaluation comparator for significance test (Noreen, 1989).

Hyper-parameters. We directly adopt most hyper-parameters of Dozat and Manning (2017), including dropout and initialization strategies. For CharLSTM, the dimension of input char embeddings is 50, and the dimension of output vector is 100, following Lample et al. (2016). For the second-order model, we set the dimensions of

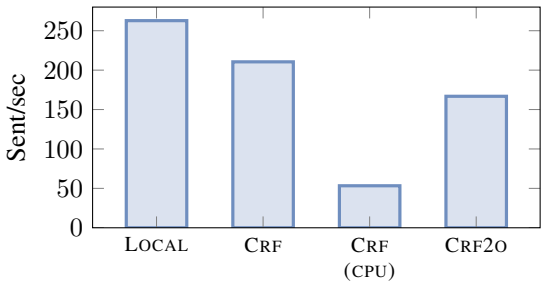

Figure 4: Parsing speed comparison on PTB-test.

$\mathbf{r}_i^{h'/s/m'}$ to 100, and find little accuracy improvement when increasing to 300. We trained each model for at most 1,000 iterations, and stop training if the peak performance on the dev data does not increase in 100 consecutive epochs.

4.1 Efficiency Comparison

Figure 4 compares parsing speed of different models on PTB-test. For fair comparison, we run all models on the same machine with Intel Xeon CPU (E5-2650v4, 2.20GHz) and GeForce GTX 1080 Ti GPU. "CRF (CPU)" refers to the model that explicitly performs the inside and outside algorithms using Cython on CPUs. Multi-threading is employed since sentences are mutually independent. However, we find using more than 4 threads does not further improve speed.

We can see that the efficiency of TreeCRF is greatly improved by batchifying the inside algorithm and implicitly realizing the outside algorithm by back-propagation on GPUs. For the first-order CRF model, our implementation is four time faster than multi-thread Cython. Our CRF and CRF2O models are only about 20% and 40% slower than the basic model with local training loss.

4.2 Main Results

Table 1 lists the main results on the dev and test data, where the trends on dev and test are mostly consistent. For fair comparison with previous works, we only consider those without using extra resources such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019). We can see that our baseline local model achieves the best performance on both PTB and CoNLL09.

On PTB, both CRF and CRF2O fail to further improve parsing accuracy, probably because the performance is already very high. However, as shown by further analysis in Section 4.3, positive effect is actually introduced by structural learning and high-order modeling.

	Dev		Test	
	UAS	LAS	UAS	LAS
PTB				
Biaffine17	-	-	95.74	94.08
F&K19	-	-	-	91.59
Li19	95.76	93.97	95.93	94.19
Ji19	95.88	93.94	95.97	94.31
Zhang19	-	-	-	93.96
LOCAL	95.82	93.99	96.08	**94.47**
CRF	95.76	93.99	96.02	94.33
CRF2O	**95.90**	**94.12**	**96.11**	94.46
CoNLL09				
Biaffine17	-	-	88.90	85.38
Li19	88.68	85.47	88.77	85.58
LOCAL	89.07	86.10	89.15	85.98
CRF	89.12	86.12	89.28	86.18†
CRF2O	**89.44**	**86.37**	**89.63**‡	**86.52**‡
NLPCC19				
LOCAL	77.01	71.14	76.92	71.04
CRF	77.34	71.62	77.53‡	71.89‡
CRF2O	**78.08**	**72.32**	**78.02**‡	**72.33**‡

Table 1: Main results. We perform significance test against LOCAL on the test data, where "†" means $p < 0.05$ and "‡" means $p < 0.005$. Biaffine17: Dozat and Manning (2017); F&K19: Falenska and Kuhn (2019); Li19: Li et al. (2019); Ji19: Ji et al. (2019); Zhang19: Zhang et al. (2019).

On CoNLL09, CRF significantly outperforms the basic local model, and CRF2O can further improve parsing accuracy.

On the partially annotated NLPCC19 data, CRF outperforms the basic local model by very large margin, indicating the usefulness of structural learning in the scenario of partial annotation. CRF2O further improves parsing performance by explicitly modeling second-order subtree features. These results confirm our intuitions discussed in Section 3.4. Please notice that the parsing accuracy looks very low because the partially annotated tokens are usually difficult for models.

4.3 Analysis

Convergence behaviour. Figure 5 compares the convergence curves. For clarity, we plot one data point corresponding to the peak LAS every 20 epochs. We can clearly see that both structural learning and high-order modeling consistently improve the model. The CRF2O model achieves steadily higher accuracy for the same epochs and

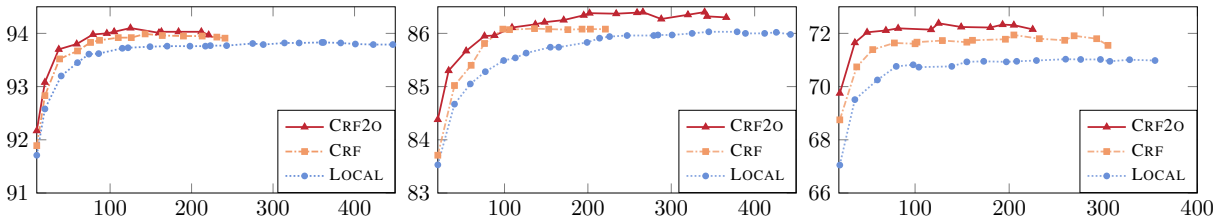

Figure 5: Convergence curves (LAS vs. training epochs) on dev data of PTB, CoNLL09, and NLPCC19.

	SIB			UCM	LCM
	P	R	F		
			PTB		
LOCAL	91.16	90.80	90.98	61.59	50.66
CRF	91.24	90.92	91.08	61.92	50.33
CRF2O	**91.56**	**91.11**	**91.33**	**63.08**	**50.99**
			CoNLL09		
LOCAL	79.20	79.02	79.11	40.10	28.91
CRF	79.17	79.55	79.36	40.61	29.38
CRF2O	**81.00**	**80.63**	**80.82**	**42.53**	**30.09**

Table 2: Sub- and full-tree performance on test data.

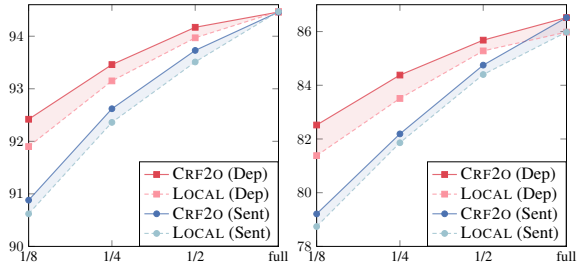

Figure 6: LAS on PTB (left) and CoNLL09-test (right) regarding the amount of training data (dependencies vs. sentences).

converges much faster than the basic local model.

Performance at sub- and full-tree levels. Beyond the dependency-wise accuracy (UAS/LAS), we would like to evaluate the models regarding performance at sub-tree and full-tree levels. Table 2 shows the results. We skip the partially labeled NLPCC19 data. UCM means unlabeled complete matching rate, i.e., the percent of sentences obtaining whole correct skeletal trees, while LCM further requires all labels are also correct. For SIB, we evaluate the model regarding adjacent-sibling subtrees (unlabeled).

We can clearly see that by modeling adjacent-sibling subtree scores, the SIB performance obtains larger improvement than both CRF and LOCAL, and this further contributes to the large improvement on full-tree matching rates (UCM/LCM).

Capability to learn from partial trees. To better understand why CRF2O performs very well on partially annotated NLPCC19, we design more comparison experiments by retaining either a proportion of random training sentences (full trees), or a proportion of random dependencies for each sentence (partial trees). Figure 6 shows the results.

We can see that the performance gap is quite steady when we gradually reduce the number of training sentences. In contrast, the gap clearly becomes larger when each training sentence has less annotated dependencies. This shows that CRF2O is superior to the basic LOCAL in utilizing partial annotated data for model training.

4.4 Results on CoNLL18 Data

Table 3 compares different models on CoNLL18 datasets, which contains a lot of non-projective trees. We adopt the multilingual pre-trained word embeddings kindly shared by Ji et al. (2019). To simplify comparison and significance test, we adopt the gold tokenization setting.

We adopt the pseudo-projective approach (Nivre and Nilsson, 2005) for handling the ubiquitous non-projective trees of most languages. Basically, the idea is to transform non-projective trees into projective ones using more complex labels for post-processing recovery.

$\text{LOCAL}_{\text{MST}}$ denotes the basic local model that directly produces non-projective tree based on MST decoding (Dozat and Manning, 2017). We can see that for the basic local parsers, the pseudo-projective LOCAL and the direct non-projective $\text{LOCAL}_{\text{MST}}$ achieve very similar performance.

More importantly, both CRF and CRF2O produce visible improvement over the baseline on many languages. Our proposed CRF2O model achieves the highest accuracy for 10 languages among 12, and obtain significant improvement on 5 languages. Overall, the averaged improvement is 0.33, which is also significant at $p < 0.005$.

	LOCAL$_{\text{MST}}$	LOCAL	CRF	CRF2O
bg	90.32	90.37	90.50	**90.66**
ca	90.77	90.97	91.16†	**91.39**‡
cs	90.80	90.87	**91.16**	91.01
de	80.48	80.23	80.44	**80.52**
en	86.87	86.93	86.84	**87.07**
es	90.63	90.54	90.72†	**91.03**‡
fr	87.66	87.72	87.92	**88.36**†
it	91.85	**91.90**	91.89	91.83
nl	87.81	87.90	88.74‡	**89.04**‡
no	90.51	90.79	90.61	**90.99**
ro	86.39	86.41	86.42	**86.52**
ru	93.11	93.10	93.04	**93.33**‡
Avg.	88.93	88.98	89.12†	**89.31**‡

Table 3: LAS on CoNLL18-test datasets. Again, † and ‡ means significance level at $p < 0.05$ and $p < 0.005$ respectively against the LOCAL parser.

5 Related Works

Due to space limitation, here we discuss in detail three recent works that are most closely related with this work. Please see the performance comparison in Table 1.

Falenska and Kuhn (2019) presented a nice analytical work on dependency parsing, similar to Gaddy et al. (2018) on constituent parsing. By extending the first-order graph-based[6] K&G parser (Kiperwasser and Goldberg, 2016) into second-order, they try to find out how much structural context is implicitly captured by the BiLSTM encoder. They concatenated three BiLSTM output vectors (i, k, j) for scoring adjacent-sibling subtrees, and adopted max-margin loss and the second-order Eisner decoding algorithm (McDonald and Pereira, 2006). Based on their negative results and analysis, they drew the conclusion that high-order modeling is redundant because BiLSTM can implicitly and effectively encode enough structural context. They also presented a nice survey on the relationship between RNNs and syntax. In this work, we use a much stronger basic parser and observe more significant UAS/LAS improvement than theirs. Particularly, we present in-depth analysis showing that explicitly high-order modeling certainly helps the parsing model and thus is complementary to the BiLSTM encoder.

[6]They have also experimented with the transition-based parser, but obtained even worse results.

Ji et al. (2019) employ graph neural network to implicitly incorporate high-order structural information into the biaffine parser. They add a three-layer graph attention network (GAT) component (Veličković et al., 2018) between the MLP and Biaffine layers. The first GAT layer takes $\mathbf{r}_i^h$ and $\mathbf{r}_i^m$ from MLPs as inputs and produces new representation $\mathbf{r}_i^{h1}$ and $\mathbf{r}_i^{m1}$ by aggregating neighboring nodes. Similarly, the second GAT layer operates on $\mathbf{r}_i^{h1}$ and $\mathbf{r}_i^{m1}$, and produces $\mathbf{r}_i^{h2}$ and $\mathbf{r}_i^{m2}$. In this way, a node gradually collects multi-hop high-order information as global evidence for scoring single dependencies. They followed the original local head-selection training loss. In contrast, this work adopts global TreeCRF loss and explicitly incorporates high-order scores into the biaffine parser.

Zhang et al. (2019) investigated the usefulness of structural training for the first-order biaffine parser. They compared performance of local head-selection loss, global max-margin loss, and TreeCRF loss on multilingual datasets. They showed that TreeCRF loss is overall slightly superior to max-margin loss, and LAS improvement from structural learning is modest but significant for some languages. They also showed that structural learning (especially TreeCRF) substantially improves sentence-level complete matching rate, which is consistent with our findings. Moreover, they explicitly computed the inside and outside algorithms on CPUs via Cython. In contrast, this work proposes an efficient second-order TreeCRF extension to the biaffine parser, and presents much more in-depth analysis to show the effect of both structural learning and high-order modeling.

6 Conclusions

This paper for the first time presents second-order TreeCRF for neural dependency parsing using triaffine for explicitly scoring second-order subtrees. We propose to batchify the inside algorithm to accommodate GPUs. We also empirically verify that the complex outside algorithm can be implicitly performed via efficient back-propagation, which naturally produces gradients and marginal probabilities. We conduct experiments and detailed analysis on 15 datasets from 13 languages, and find that structural learning and high-order modeling can further enhance the state-of-the-art biaffine parser in various aspects: 1) better convergence behaviour; 2) higher performance on sub- and full-tree levels; 3) better utilization of partially annotated data.

Acknowledgments

We thank for the helpness of Wei Jiang, Yahui Liu, Haoping Yang, Houquan Zhou and Minyue Zhou on charts and paper writing. We also thank for Tao Ji for kind share of the data and suggestions on our experiments.

References

Jiong Cai, Yong Jiang, and Kewei Tu. 2017. CRF autoencoder for unsupervised dependency parsing. In *Proceedings of EMNLP*.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of EMNLP*.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR*.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of NAACL*.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of WS*.

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of ACL*.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL*.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's going on in neural constituency parsers? an analysis. In *Proceedings of NAACL-HLT*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL*.

Rebecca Hwa. 1999. Supervised grammar induction using training data with limited constituent information. In *Proceedings of ACL*.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of ACL*.

Xinzhou Jiang, Zhenghua Li, Bo Zhang, Min Zhang, Sheng Li, and Luo Si. 2018. Supervised treebank conversion: Data and approaches. In *Proceedings of ACL*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of ACL*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of EMNLP*.

Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. 2019. Self-attentive biaffine dependency parsing. In *Proceedings of IJCAI*.

Zhenghua Li, Min Zhang, Yue Zhang, Zhanyi Liu, Wenliang Chen, Hua Wu, and Haifeng Wang. 2016. Active learning for dependency parsing with partial annotation. In *Proceedings of ACL*.

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of COLING*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the EACL*.

Joakim Nivre, Yoav Goldberg, and Ryan McDonald. 2014. Squibs: Constrained arc-eager dependency parsing. *CL*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of LREC*.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of ACL*.

Eric W Noreen. 1989. *Computer-intensive methods for testing hypotheses*.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proceedings of ACL-IJCNLP*.

Xue Peng, Zhenghua Li, Min Zhang, Rui Wang, Yue Zhang, and Luo Si. 2019. Overview of the nlpcc 2019 shared task: cross-domain dependency parsing. In *Proceedings of NLPCC*.

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of ACL*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*.

David A. Smith and Noah A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of EMNLP*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of ICLR*.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of ACL*.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of ACL*.

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 CoNLL*.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of EACL*.

Zhisong Zhang, Xuezhe Ma, and Eduard Hovy. 2019. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of ACL*.