Bidirectional Transition-Based Dependency Parsing

Yunzhe Yuan, Yong Jiang, Kewei Tu
School of Information Science and Technology
ShanghaiTech University
{yuanyzh,jiangyong,tukw}@shanghaitech.edu.cn

Abstract

Transition-based dependency parsing is a fast and effective approach for dependency parsing. Traditionally, a transition-based dependency parser processes an input sentence and predicts a sequence of parsing actions in a left-to-right manner. During this process, an early prediction error may negatively impact the prediction of subsequent actions. In this paper, we propose a simple framework for bidirectional transition-based parsing. During training, we learn a left-to-right parser and a right-to-left parser separately. To parse a sentence, we perform joint decoding with the two parsers. We propose three joint decoding algorithms that are based on joint scoring, dual decomposition, and dynamic oracle respectively. Empirical results show that our methods lead to competitive parsing accuracy and our method based on dynamic oracle consistently achieves the best performance.

Introduction

Dependency parsing is a task with a long history in natural language processing. The goal of dependency parsing is to discover syntactic relations among words in a sentence. Dependency trees have been shown to provide effective syntactic information in many semantic tasks, such as semantic role labeling (Lei et al. 2015), machine translation (Chen et al. 2017) and sentence classification (Ma et al. 2015). Generally speaking, there are two types of models for dependency parsing, graph-based models (McDonald 2006) and transition-based models (Yamada and Matsumoto 2003; Nivre 2003). Known for its efficiency in training and decoding, transition-based parsing has been very popular for many languages. A transition-based parser processes the input sentence in a sequential manner and outputs a series of actions to incrementally construct a parse tree. During this process, each action is typically predicted using a discriminative classifier, such as a SVM (Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2006), a stack LSTM (Dyer et al. 2015) or a BiLSTM (Kiperwasser and Goldberg 2016).

Traditionally, transition-based dependency parsing processes the sentence in a left-to-right manner. During this process, a prediction mistake, especially one in the early stage, may negatively impact many subsequent predictions. Motivated by bidirectional neural machine translation (Liu et al. 2016), we propose to alleviate this problem and improve parsing accuracy with a simple framework of bidirectional transition-based parsing. During training, in addition to the traditional left-to-right parser, we also learn a right-to-left parser by reversing the word order of all the training sentences. For parsing, we propose three joint decoding algorithms that can be used to parse a new sentence based on both of the learned parsers. The first algorithm runs the two parsers and then selects one of the two parse trees that has a higher summed score given by the two parsers. The second algorithm is based on the dual decomposition technique (Rush and Collins 2012) which is an iterative algorithm that encourages the two parsers to gradually reach consensus. The last algorithm follows the same idea, but during each iteration it guides the two parsers with dynamical oracles (Goldberg and Nivre 2012). To the best of our knowledge, we are the first to employ the dynamical oracle mechanism in the decoding period rather than the training period.

Our framework is quite general and can be applied to any transition-based dependency parsing model with well-defined dynamic oracles. In our experiments, we apply our framework to the transition-based parser of Kiperwasser and Goldberg (2016). We evaluate our model on two popular treebanks (PTB and CTB) as well as eight additional treebanks from the Universal Dependency dataset. Experimental results show that our proposed algorithms significantly outperform the unidirectional baselines and our third algorithm based on dynamic oracles has the highest accuracy.

Related Work

Transition-based Dependency Parsing

A transition-based dependency parser incrementally builds a dependency tree by a sequence of actions. Some of the early transition-based parsers are list-based (Covington 2001), but more recently stack-based parsers (Nivre 2003), also known as shift-reduced parsers, are more widely used. There are two basic transition systems: the arc-standard system (Nivre 2004) and the arc-eager system (Nivre 2003), which can be combined to form the arc-hybrid system (Kuhlmann, Gómez-Rodríguez, and Satta 2011; Kiperwasser and Goldberg 2016).

Many traditional machine learning methods can be used to train a transition-based parser, such as SVM (Yamada

and Matsumoto 2003), maximum entropy classification (Attardi 2006), memory based learning (Attardi 2006), multi-class averaged perceptron (Attardi et al. 2007) and so on. With the rise of deep learning, neural networks have recently been used to train dependency parsers (Stenetorp 2013; Chen and Manning 2014; Dyer et al. 2015; Kiperwasser and Goldberg 2016), which leads to state-of-the-art dependency parsing accuracies. For decoding, the simplest way is Athe greedy algorithm that always performs the action with the highest local score given each parsing configuration, but there also exist other methods such as beam search (Zhang and Clark 2008) and dynamic programming (Huang and Sagae 2010).

Parser Ensemble

Our work is related to parser ensemble, which aims at combining multiple different parses of the same sentence. Sagae and Tsujii (2007) combine the parse trees produced by a left-to-right parser and a right-to-left parser with a maximum spanning tree voting scheme in the CoNLL Shared Task 2007 (Nivre et al. 2007). Zhang and Clark (2008) trained a graph-based parser and a transition-based parser, combined the scoring of the two parsers, and then used the transition-based parser for decoding. Surdeanu and Manning (2010) systematically studied different ensemble methods based on seven different parsers from MSTParser (McDonald et al. 2005) and Malt Parser (Nivre, Hall, and Nilsson 2006). While our framework can be seen as combining two unidirectional parsers, most of the ensemble methods cannot be directly applied to our setting.

Oracles in Dependency Parsing

To effectively train a dependency parser, one key component is an oracle, which is employed to generate optimal action sequences from gold trees. The oracle is used to generate training targets to learn a classifier or used to determine whether to perform updates in training a beam search classifier.

There are two kinds of oracles: static oracles and dynamic oracles. Static oracles refer to producing a single static sequence of transitions that is supposed to be followed in its entirety. However, at test time, the parsers classifier may be faced with configurations which were not observed in training. This phenomenon may lead to error propagation and significantly decrease the parsing accuracy. Rather than defining a deterministic transition sequence, dynamic oracles permit configurations which are not a part of a gold derivation. While dynamic oracles have been widely employed for training transition-based models (Goldberg and Nivre 2012; 2013; Goldberg, Sartorio, and Satta 2014), they have never been used in the decoding period.

Transition-Based Dependency Parsing

Given a sentence $\mathbf{x}$ that consists of a word sequence, a transition-based parser produces a sequence of actions that gradually construct an output parse tree $\mathbf{y}$. We choose the parser of Kiperwasser and Goldberg (2016) as our baseline.

Algorithm 1 Greedy Transition-Based Parsing

Input: a sentence $\mathbf{x}$
Output: a dependency parse tree $\mathbf{y}$

$c \leftarrow \text{Initial}(\mathbf{x})$
while not $\text{Terminal}(c)$ **do**
  $\hat{a} \leftarrow \arg\max_{a \in \text{Legal}(c)} f(c, a)$
  $c \leftarrow \hat{a}(c)$
$\mathbf{y} \leftarrow c.T$
return $\mathbf{y}$

Model

The transition system uses an abstract machine that generates an action given a parsing configuration. Each configuration c consists of a stack σ, a buffer β and an arc set T. The initial configuration c_0 contains an empty stack, a buffer containing a special index ROOT followed by all the words of the sentence, and an empty arc set.

We use the arc-hybrid transition system (Kuhlmann, Gómez-Rodríguez, and Satta 2011), which defines three types of actions named SHIFT, LEFT and RIGHT. The SHIFT action pushes the first item b_0 of the buffer to the stack. The LEFT_l action (l is a dependency label) pops the stack, let the modifier of the poped item s_0 be the first item of the buffer b_0, and add the arc with label l to the arc set. The RIGHT_l action pops the stack, let the modifier of the poped item s_0 be the new top item of the stack s_1, and add the arc with label l to the arc set. We use $\sigma|s_1|s_0$ to denote the stack σ with top items s_0 and s_1, $b_0|\beta$ to denote the buffer β with the first item b_0.

$$\text{SHIFT}[(\sigma, b_0|\beta, T)] = (\sigma|b_0, \beta, T)$$
$$\text{LEFT}_l[(\sigma|s_1|s_0, b_0|\beta, T)] = (\sigma|s_1, b_0|\beta, T \cup \{(b_0, s_0, l)\})$$
$$\text{RIGHT}_l[(\sigma|s_1|s_0, \beta, T)] = (\sigma|s_1, \beta, T \cup \{(s_1, s_0, l)\})$$

We use the greedy algorithm to parse sentences. Algorithm 1 shows the procedure. The abstract machine starts with the initial configuration c_0. For each configuration c, one needs to choose an action a to transfer to the next configuration. We define a score function f for actions and select the action $\hat{a}$ that leads to the largest score $f(c, \hat{a})$. The transition ends when the buffer of c becomes empty. The arc set T then contains all the dependency arcs that constitute the output parse tree $\mathbf{y}$.

The action score for each configuration $f(c, a)$ is computed as follows. We run a BiLSTM on the input sentence and produce a vector $\mathbf{v}_i$ for each word $\mathbf{x}_i$ in the sentence.

$$\mathbf{v}_i = \text{BiLSTM}(\mathbf{x}_{1:n}, i)$$

The vector representation of a configuration $c = (\sigma, \beta, T)$ is the concatenation of the vectors of the top three items of σ and the first item of β.

$$\boldsymbol{\phi}(c) = \mathbf{v}_{s_2} \circ \mathbf{v}_{s_1} \circ \mathbf{v}_{s_0} \circ \mathbf{v}_{b_0}$$

The score function $f(c, a)$ is computed by a multi-layer perceptron. The input to the multi-layer perceptron is the vector representation of configuration c, and the output is the scores of all the valid actions under configuration c.

$$f(c, a) = MLP(\phi(c))[a]$$

$$
\begin{aligned}
\mathcal{C}\big(\text{SHIFT}; c=(\sigma|s_1|s_0,b|\beta,T), T_g\big) &= \Big|\{(k,b,l)\in T_g | k\in s_1\cup\sigma\}\cup\{(b,k,l)\in T_g | k\in s_0\cup s_1\cup\sigma\}\Big| \\
\mathcal{C}\big(\text{LEFT}_l; c=(\sigma|s_1|s_0,b|\beta,T), T_g\big) &= \Big|\{(k,s_0,l)\in T_g | k\in s_1\cup\beta\}\cup\{(s_0,k,l)\in T_g | k\in b\cup\beta\}\Big| \\
\mathcal{C}\big(\text{RIGHT}_l; c=(\sigma|s_1|s_0,b|\beta,T), T_g\big) &= \Big|\{(k,s_0,l)\in T_g | k\in b\cup\beta\}\cup\{(s_0,k,l)\in T_g | k\in b\cup\beta\}\Big|
\end{aligned}
$$

Table 1: Transition costs for the arc-hybrid system

Training

Our training algorithm follows Kiperwasser and Goldberg (2016). Given a training sentence, we convert its gold dependency tree into a set[1] of sequences of transition actions. For each action, we use the following hinge loss.

$$\max(0, 1 - \max_{a\in G} MLP(\phi(c))[a] + \max_{a\in A\setminus G} MLP(\phi(c))[a])$$

where A is the set of all possible actions and G is the set of gold (correct) actions. The total loss is the summation of the loss for each action.

Bidirectional Transition-Based Parsing

A traditional transition-based parser processes a sentence from left to right. During this process, prediction mistakes in the early stage can negatively impact future predictions, leading to a series of subsequent mistakes. We get inspiration from bidirectional neural machine translation (Liu et al. 2016) and propose bidirectional transition-based parsing. Specifically, we train a left-to-right parser as well as a right-to-left parser. The right-to-left parser is trained by reversing the word order of all the training sentences. During parsing, we perform joint decoding using both parsers. Below we discuss three algorithms for joint decoding.

Vanilla Joint Scoring

The simplest method is to design a joint scoring function of parse trees $s(\mathbf{t})$, based on which we select from the parse trees $\{\mathbf{y}, \mathbf{z}\}$ produced by the two unidirectional parsers, i.e., we output $\arg\max_{\mathbf{t}\in\{\mathbf{y},\mathbf{z}\}} s(\mathbf{t})$. Here we simply use a scoring function that is the summation of the scores computed by the two unidirectional parsers:

$$s(\mathbf{t}) = F(\mathbf{t}) + G(\mathbf{t})$$

where $\mathbf{t}$ is a parse tree, F and G are the parse tree scoring functions of the two parsers.

Joint Decoding with Dual Decomposition

The joint scoring method is rather limited in that it cannot produce any new parse trees. A better method would be to encourage each unidirectional parser to gradually modify its parse tree in order to satisfy the other parser, in hope that the two parsers would eventually reach agreement on a single new parse tree. Here we use the dual decomposition technique (Dantzig and Wolfe 1960; Rush and Collins 2012) for this purpose.

[1]The gold action at a configuration may not be unique in some transition systems.

We use $\mathcal{Y}, \mathcal{Z}$ to denote the sets of all possible parse trees produced by the two unidirectional parsers. Typically we have $\mathcal{Y} = \mathcal{Z}$. We represent a parse tree $\mathbf{y}$ by a matrix: if the modifier of the i-th word of the sentence is the j-th word, then $\mathbf{y}(i,j) = 1$; otherwise $\mathbf{y}(i,j) = 0$. Our goal is to maximize $F(\mathbf{y}) + G(\mathbf{z})$ under the constraint that $\mathbf{y} = \mathbf{z}$. By using Lagrangian relaxation, we can optimize the two parsers $\mathbf{y}$ and $\mathbf{z}$ respectively by adding and subtracting the same term to $F(\mathbf{y})$ and $G(\mathbf{z})$. The summation of the two parts is called the Lagrangian dual $L(\mathbf{u})$.

$$
\begin{aligned}
L(\mathbf{u}) = &\max_{\mathbf{y}\in\mathcal{Y}}(F(\mathbf{y}) + \sum_{i,j}\mathbf{u}(i,j)\mathbf{y}(i,j)) + \\
&\max_{\mathbf{z}\in\mathcal{Z}}(G(\mathbf{z}) - \sum_{i,j}\mathbf{u}(i,j)\mathbf{z}(i,j))
\end{aligned}
$$

It can be proved that $L(\mathbf{u})$ is an upper bound of our goal, and we can iteratively update $\mathbf{u}$ trying to reach the tightest bound. In the case where the two parsers cannot reach agreement after the maximal iterations, we simply return one of them. Note that the greedy decoding algorithm (Algorithm 1) cannot solve the $\arg\max$ exactly and therefore we can only approximately optimize the dual decomposition objective using Algorithm 2.

Joint Decoding Guided by Dynamic Oracle

Dynamic oracles (DO) (Goldberg and Nivre 2012) are a useful technique in training a transition-based parser. When the current configuration of the parser deviates from a given gold parse, the dynamic oracle suggests one or more actions that help the parser return to the gold parse in the fastest possible way.

Goldberg and Nivre (2012) derived the dynamic oracle function for the arc-eager transition system. Here we briefly describe the dynamic oracle function proposed by Kiperwasser and Goldberg (2016) for the arc-hybrid system.

$$o(a; c, T) = \begin{cases} \text{true}, & \text{if } \mathcal{C}(a; c, T) = 0 \\ \text{false}, & \text{otherwise} \end{cases}$$

The oracle function o is a Boolean function, whose value is true if and only if the cost function $\mathcal{C}$ is equal to 0. $\mathcal{C}$ is a function of the input configuration c, action a and gold tree T_g as shown in Table 1.

During joint decoding, we hope to encourage the two unidirectional parsers to reach an agreement. We follow an iterative procedure similar to the dual decomposition method during which the two parsers produce increasingly similar parse trees. In each iteration, our idea is to regard the parse tree produced by one parser as the gold parse and use a dynamic oracle to bias the other parser so that it produces a

Algorithm 2 Decoding with Dual Decomposition

Input: a sentence $\mathbf{x}$, the number of iteration K
Output: a dependency parse tree $\mathbf{y}$

$\mathbf{u} \leftarrow \mathbf{0}$
$\mathbf{y}^{(0)} \leftarrow \mathbf{0}$
$\mathbf{z}^{(0)} \leftarrow \mathbf{0}$
for $k = 1$ to K **do**
    $\mathbf{y}^{(k)} \leftarrow \textsc{Decode}(\mathbf{x}, \mathbf{u}, \mathbf{y}^{(k-1)}, 1)$
    $\mathbf{z}^{(k)} \leftarrow \textsc{Decode}(\mathbf{x}, \mathbf{u}, \mathbf{z}^{(k-1)}, -1)$
    if $\mathbf{y}^{(k)} = \mathbf{z}^{(k)}$ **then**
        return $\mathbf{y}^{(k)}$
    else
        $\mathbf{u} \leftarrow \mathbf{u} - \alpha_k(\mathbf{y}^{(k)} - \mathbf{z}^{(k)})$
return $\mathbf{y}^{(K)}$

function $\textsc{Decode}(\mathbf{x}, \mathbf{u}, \mathbf{y}, sgn)$
    $c \leftarrow \text{Initial}(\mathbf{x})$
    while not $\text{Terminal}(c)$ **do**
        $\mathbf{v} \leftarrow \mathbf{0}$
        for a in $\text{Legal}(c)$ **do**
            if a **is not** SHIFT **then**
                $arc \leftarrow a(c).T \backslash c.T$
                $\mathbf{v}(a) \leftarrow sgn \cdot \mathbf{u}(arc)\mathbf{y}(arc)$
        $\hat{a} \leftarrow \arg\max_{a \in \text{Legal}(c)} f(c, a) + \mathbf{v}(a)$
        $c \leftarrow \hat{a}(c)$
    $\mathbf{y} \leftarrow c.T$
    return $\mathbf{y}$

parse tree closer to the gold parse than it would without the dynamic oracle. More concretely, at each step of parsing with the second parser, we add a reward to the scores of the actions suggested by the dynamic oracle so as to bias the action choice.

The algorithm is shown in Algorithm 3. First, we obtain two parse trees $\mathbf{y}^{(0)}$ and $\mathbf{z}^{(0)}$ from the two unidirectional parsers. Then, at the k-th iteration, we regard $\mathbf{z}^{(k-1)}$ as the gold tree to guide the first parser to produce $\mathbf{y}^{(k)}$, and likewise we regard $\mathbf{y}^{(k-1)}$ as the gold tree to guide the second parser to produce $\mathbf{z}^{(k)}$. With a proper reward value (c_{do}), the parse trees produced by the two parsers would become increasingly similar.

While Algorithms 2 and 3 have very similar forms, they differ in the set of actions whose scores are modified in the decoding subroutine. In the dual decomposition algorithm, only the LEFT and RIGHT actions that produce arcs upon which the two parsers disagree may have their scores modified. In the algorithm guided by dynamic oracles, at least one action in each valid configuration would have its score modified. Therefore, dynamic oracles lead to significantly more change to the scoring function, which potentially leads to much faster convergence of the parses produced by the two parsers.

Finally, because Algorithms 2 and 3 have similar forms, we can combine these two methods by adding up their modifications to the scoring functions.

Algorithm 3 Decoding Guided by Dynamic Oracle

Input: a sentence $\mathbf{x}$, a constant reward c_{do}, the number of iteration K
Output: a dependency parse tree $\mathbf{y}$

$\mathbf{y}^{(0)} = \text{None}$
$\mathbf{z}^{(0)} = \text{None}$
for $k = 1$ to K **do**
    $\mathbf{y}^{(k)} \leftarrow \textsc{Decode}(\mathbf{x}, z^{(k-1)}, c_{do})$
    $\mathbf{z}^{(k)} \leftarrow \textsc{Decode}(\mathbf{x}, y^{(k-1)}, c_{do})$
    if $\mathbf{y}^{(k)} = \mathbf{z}^{(k)}$ **then**
        return $\mathbf{y}^{(k)}$
return $\mathbf{y}^{(K)}$

function $\textsc{Decode}(\mathbf{x}, \mathbf{y_g}, c_{do})$
    $c \leftarrow \text{Initial}(\mathbf{x})$
    while not $\text{Terminal}(c)$ **do**
        $\mathbf{v} \leftarrow \mathbf{0}$
        if $\mathbf{y_g}$ **is not** None **then**
            for $a \in \text{Legal}(c)$ **and** $o(a, c, \mathbf{y_g})$ **do**
                $\mathbf{v}(a) = c_{do}$
        $\hat{a} \leftarrow \arg\max_{a \in \text{Legal}(c)} f(c, a) + \mathbf{v}(a)$
        $c \leftarrow \hat{a}(c)$
    $\mathbf{y} \leftarrow c.T$
    return $\mathbf{y}$

Dataset	#Train	#Dev	#Test
PTB	39832	1700	2416
CTB	16091	803	1910
UD-de	13814	799	977
UD-en	12543	2002	2077
UD-es	14187	1400	426
UD-fr	14554	1478	416
UD-it	13121	564	482
UD-nl	12269	718	596
UD-pl	13774	1745	1727
UD-zh	3997	500	500

Table 2: Statistics of the ten treebank datasets

Experiments

Data

We evaluate our framework on the Wall Street Journal corpus of Penn Treebank (PTB), Chinese Treebank 5 (CTB), and eight additional treebanks of different languages from Universal Dependencies 2.2 (UD)[2].

For PTB, we use Stanford Dependencies (Silveira et al. 2014) to convert the original treebank to the dependency version and use the standard 2-21/22/23 split for training, development and testing.

For CTB, we use Penn2Malt[3] to convert the original treebank to the dependency version with the heading rules of (Zhang and Clark 2008). We use the training, development

[2] http://universaldependencies.org/

[3] http://stp.lingfil.uu.se/ñivre/research/Penn2Malt.html

Method	DE		EN		ES		FR	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
L2R	81.62	76.14	88.87	86.79	86.52	82.90	87.33	83.17
R2L	81.54	76.03	89.13	87.10	86.78	83.05	87.63	83.57
Vanilla	82.62	76.90	90.20	88.02	87.49	83.60	88.25	84.04
DD	82.64	77.12	90.23	88.24	87.52	83.78	88.30	84.77
DO	**83.02**	**79.58**	**90.56**	**89.48**	**87.83**	**85.69**	**88.81**	**87.82**
Method	**IT**		**NL**		**PL**		**ZH**	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
L2R	91.41	89.25	87.07	83.43	94.77	92.98	85.16	82.64
R2L	91.46	89.33	87.74	84.44	95.39	93.81	86.01	83.26
Vanilla	92.19	89.90	88.56	84.72	95.94	93.96	87.04	84.24
DD	92.22	90.61	88.58	85.04	95.96	94.47	87.06	84.38
DO	**92.31**	**91.58**	**89.41**	**87.41**	**96.10**	**94.62**	**87.75**	**86.46**

Table 3: Results on UD

Method	PTB		CTB	
	UAS	LAS	UAS	LAS
L2R	93.54± 0.12	92.22± 0.17	86.21± 0.14	85.02± 0.13
R2L	93.56± 0.18	93.27± 0.25	86.44± 0.07	85.22± 0.07
Vanilla	94.35± 0.05	92.91± 0.11	87.36± 0.07	86.07± 0.06
DD	94.35± 0.05	93.01± 0.09	87.41± 0.09	86.18± 0.09
DO	**94.60± 0.04**	**94.02± 0.13**	88.07± 0.07	**87.54± 0.14**
DD + DO	**94.60± 0.04**	**94.02± 0.13**	88.09± 0.08	87.52± 0.10
C&M14	91.80	89.60	83.90	82.40
Dyer15	93.10	90.90	87.20	85.70
Weiss15	93.99	92.05	-	-
Andor16	**94.61**	92.79	-	-
Ballesteros16	93.56	91.42	87.65	86.21
K&G16	93.90	91.90	87.60	86.10
Zhang16	94.10	91.90	87.84	86.15
Shi17	94.53± 0.05	-	**88.62± 0.09**	-

Table 4: Results on PTB and CTB

Hyperparameter	Value
Word embedding dimension	100
POS tag dimension	25
BiLSTM layers	2
LSTM dimensions	200/200
MLP units	100

Table 5: Hyperparameters used in the experiments

and testing split of the dataset following (Zhang and Clark 2008).

We choose eight additional treebanks from UD, including Chinese (zh-GSD), Dutch (nl-Alpino), English (en-EWT), French (fr-GSD), German (de-GSD), Italian (it-ISDT), Polish (pl-LFG) and Spanish (es-GSD). See Table 2 for statistics of the datasets.

For PTB, we use predicted POS tags generated by Stanford Tagger (Toutanova et al. 2003) with an accuracy of 98.0%. For CTB and UD, we use the gold POS tags.

We use pretrained word embeddings from GloVe[4] (Pennington, Socher, and Manning 2014) in our experiments on PTB and pretrained word embeddings from fastText[5] (Bojanowski et al. 2017) in our experiments on UD. We use word2vec (Mikolov et al. 2013) to train Chinese word embeddings on Wikipedia for our experiments on CTB.

Setup

For dual decomposition, we decrease the update rate α_k over time as recommended by (Rush and Collins 2012). Specifically, we set $\alpha_k = \frac{c_{dd}}{t+1}$ where t is the number of times the dual value increases and c_{dd} is a hyperparameter. For our method based on dynamic oracle, we also decrease the reward c_{do} over time in a similar way to make the decoding process stabler.

Since convergence is not guaranteed for both dual decomposition and our method based on dynamic oracle, we set a maximum iteration $K = 100$. If no agreement is reached

[4]https://nlp.stanford.edu/projects/glove/

[5]https://fasttext.cc/docs/en/crawl-vectors.html

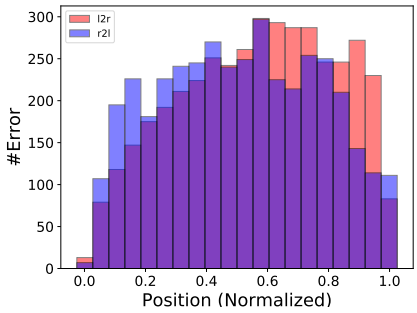

Figure 1: Comparisons between results from the two bidirectional models

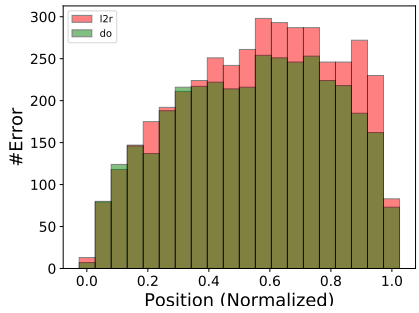

Figure 2: Comparisons between results from the left-to-right model and the DO method

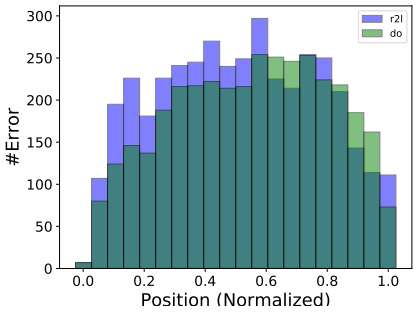

Figure 3: Comparisons between results from the right-to-left model and the DO method

after K iterations, we collect all the $2K$ parse trees produced during decoding and select the best parse following joint scoring.

We compare six methods: the left-to-right parser (L2R), the right-to-left parser (R2L), vanilla joint scoring (Vanilla), joint decoding with dual decomposition (DD), joint decoding guided by dynamic oracles (DO), and the combination of DD and DO (DD+DO).

For each dataset, we train each unidirectional parser for 20 epochs, collect all the models after each of the 20 epochs, and choose the best model based on the UAS metric on the development set. We then fix the unidirectional parsers and tune the hyperparameters of our joint decoding algorithms ($c_{dd} \in \{0, 0.005, 0.01\}$ and $c_{do} \in \{0, 1, 2, 3, 4\}$) based on the UAS on the development set. We repeat the process for five times with different random seeds and report the average accuracy and the standard deviation on the test set.

Table 5 shows all the other hyperparameters used in our experiments.

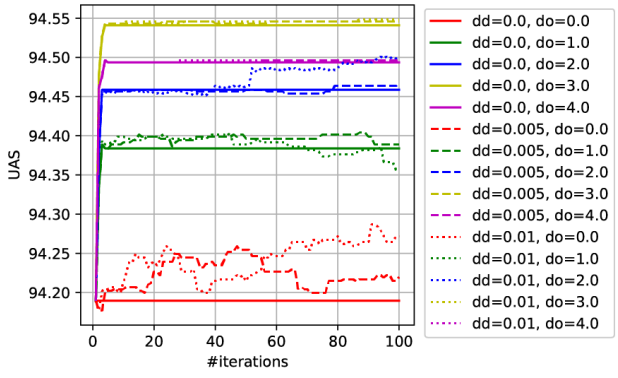

Figure 4: Accuracy vs. iteration for DD (corresponding to do=0), DO (corresponding to dd=0), and DD+DO

Results

We compare our methods with the following neural transition-based parsers on PTB and CTB: C&M14 (Chen and Manning 2014), Dyer15 (Dyer et al. 2015), Weiss15 (Weiss et al. 2015), Ballesteros16 (Ballesteros et al. 2016), Andor16 (Andor et al. 2016), K&G16(Kiperwasser and Goldberg 2016), Zhang16 (Zhang, Cheng, and Lapata 2017), and Shi17 (Shi, Huang, and Lee 2017). As can be seen from Table 4, our methods are very competitive compared with previous methods and our DO-based algorithm produces the highest accuracies on PTB.

It can also be seen that DD only slightly outperforms Vanilla, while DO achieves significant improvement over both Vanilla and DD. Combining DD with DO does not lead to any improvement over DO.

We also run our methods without hyperparameter tuning on the eight UD datasets. As can be seen from Table 3, our three joint decoding algorithms consistently outperform the unidirectional baselines and the DO-based algorithm consistently achieves the highest accuracy.

Figure 4 plots the change of accuracy during our iterative joint decoding algorithms (DD, DO and DD+DO) on the WSJ development set. It can be seen that the DO-based method not only leads to higher accuracy, but also converges very fast (within 4 iterations). In comparison, DD does not converge within 100 iterations. It can also be seen that when c_{do} is small, combining DD into the DO-based method is helpful; but when c_{do} is large, adding DD to the DO-based method has little impact on the accuracy.

Analysis

Statistics We count the number of action errors appeared in the relative position of parse trees from the left-to-right parser, the right-to-left parser and results from the DO method. We plot the graph in Figure 1, 2 and 3. According to the Figure, there are some interesting findings.

For the two unidirectional parsers, we also analyze the distribution of erroneous actions with respect to the relative position (normalized to the range between 0 and 1) of the action in the complete action sequence produced by the parser (Figure 1). For the right-to-left parser, we reverse the

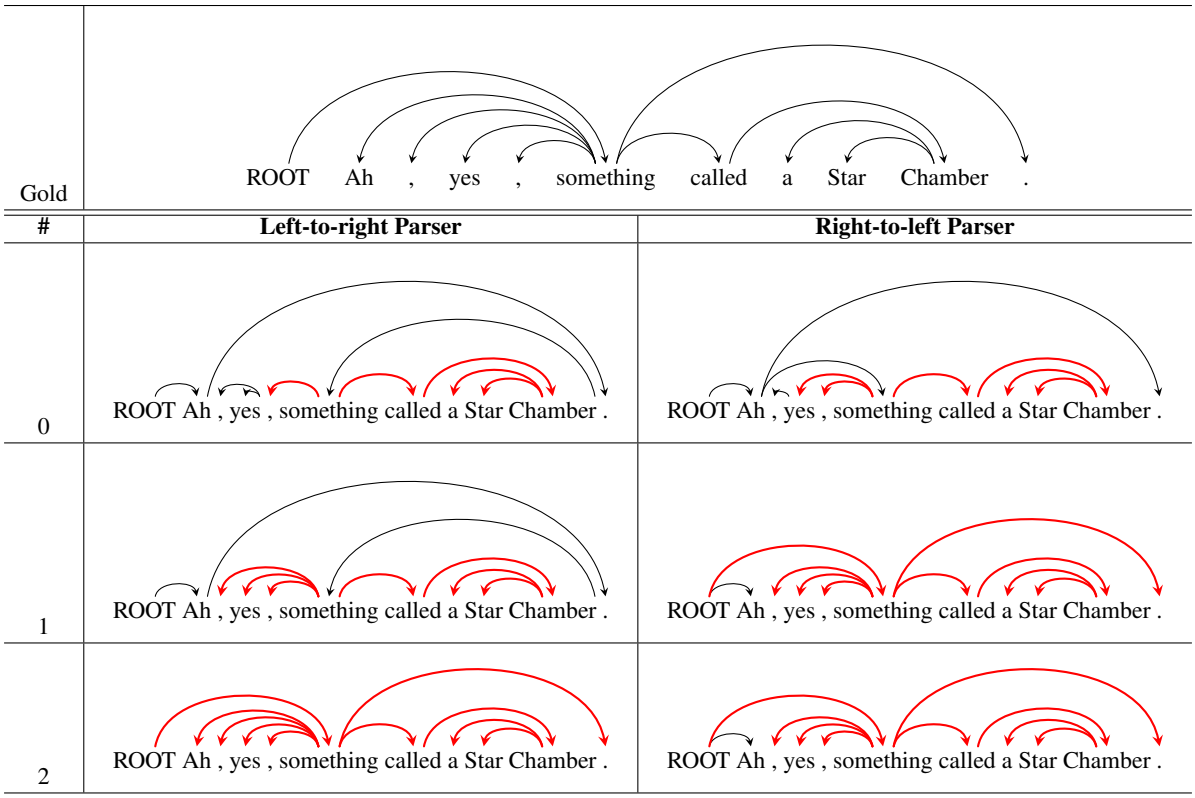

Figure 5: Example of joint decoding guided by dynamic oracles. The red arcs denote correct dependency arcs.

direction of the x-axis to align the histogram with that of the left-to-right parser. It can be seen that both parsers are more likely to make mistakes at the later stage of parsing: the left-to-right parser makes significantly more mistakes in the range of $[0.6, 0.9]$, while the right-to-left parser makes significantly more mistakes in the range of roughly $[0.1, 0.4]$ (i.e., $[0.6, 0.9]$ in the right-to-left order). This phenomenon suggests that error propagation exists in transition-based dependency parsers.

We then analyze the error distribution of the DO-based method. The comparisons of the error distributions of the two unidirectional parsers and the DO-based method are shown in Figure 2 and 3. We can see that overall the DO-based method makes less errors than the two unidirectional parsers and its errors are also more evenly distributed.

Case Study

Figure 5 shows the the intermediate parsing results during the iterative process of the DO-based method on the input sentence *Ah, yes, something called a Star Chamber*.

At the beginning of the algorithm, we use the left-to-right parser and the right-to-left parser to produce two parse trees, which have 5 and 6 correct arcs respectively. After the first iteration, we get two new parse trees using the left-to-right parser and the right-to-left parser, which have 7 and 9 correct arcs respectively. After the second iteration, the two parse trees have 10 and 9 correct arcs respectively. From this example we can see that with more iterations, the parse trees produced by the two parsers become increasingly similar as well as increasingly close to the gold tree.

Conclusion

In this paper, we propose a framework for bidirectional transition based dependency parsing. During training, we learn two unidirectional parsers separately. During parsing, we perform joint decoding using the two parsers with three algorithms based on joint scoring, dual decomposition, and dynamic oracles respectively. We evaluate our methods on ten datasets and observe that our methods lead to competitive parsing accuracy and our method based on dynamic oracles consistently achieves the best performance. Our code is based on Kiperwasser and Goldberg (2016) and is available at `https://github.com/yuanyunzhe/bi-trans-parser`.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (61503248) and Major Program of Science and Technology Commission Shanghai Municipal (17JC1404102).

References

Andor, D.; Alberti, C.; Weiss, D.; Severyn, A.; Presta, A.; Ganchev, K.; Petrov, S.; and Collins, M. 2016. Globally

normalized transition-based neural networks. In *Proceedings of ACL*.

Attardi, G.; DellOrletta, F.; Simi, M.; Chanev, A.; and Ciaramita, M. 2007. Multilingual dependency parsing and domain adaptation using desr. In *Proceedings of EMNLP*.

Attardi, G. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*.

Ballesteros, M.; Goldberg, Y.; Dyer, C.; and Smith, N. A. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of EMNLP*.

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching word vectors with subword information. *TACL*.

Chen, D., and Manning, C. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Chen, K.; Wang, R.; Utiyama, M.; Liu, L.; Tamura, A.; Sumita, E.; and Zhao, T. 2017. Neural machine translation with source dependency representation. In *Proceedings of EMNLP*.

Covington, M. A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, 95–102.

Dantzig, G. B., and Wolfe, P. 1960. Decomposition principle for linear programs. *Operations research*.

Dyer, C.; Ballesteros, M.; Ling, W.; Matthews, A.; and Smith, N. A. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL*.

Goldberg, Y., and Nivre, J. 2012. A dynamic oracle for arc-eager dependency parsing. *Proceedings of COLING*.

Goldberg, Y., and Nivre, J. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*.

Goldberg, Y.; Sartorio, F.; and Satta, G. 2014. A tabular method for dynamic oracles in transition-based parsing. *TACL*.

Huang, L., and Sagae, K. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*.

Kiperwasser, E., and Goldberg, Y. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *TACL*.

Kuhlmann, M.; Gómez-Rodríguez, C.; and Satta, G. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of ACL*.

Lei, T.; Zhang, Y.; Màrquez, L.; Moschitti, A.; and Barzilay, R. 2015. High-order low-rank tensors for semantic role labeling. In *Proceedings of NAACL*.

Liu, L.; Utiyama, M.; Finch, A.; and Sumita, E. 2016. Agreement on target-bidirectional neural machine translation. In *Proceedings of NAACL*.

Ma, M.; Huang, L.; Xiang, B.; and Zhou, B. 2015. Dependency-based convolutional neural networks for sentence embedding. *arXiv preprint arXiv:1507.01839*.

McDonald, R.; Pereira, F.; Ribarov, K.; and Hajič, J. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of EMNLP*.

McDonald, R. 2006. Discriminative training and spanning tree algorithms for dependency parsing. *University of Pennsylvania, PhD Thesis*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.

Nivre, J.; Hall, J.; Kübler, S.; McDonald, R.; Nilsson, J.; Riedel, S.; and Yuret, D. 2007. The conll 2007 shared task on dependency parsing. In *Proceedings of EMNLP*.

Nivre, J.; Hall, J.; and Nilsson, J. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*.

Nivre, J. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*.

Nivre, J. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*.

Rush, A. M., and Collins, M. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *JAIR* 45:305–362.

Sagae, K., and Tsujii, J. 2007. Dependency parsing and domain adaptation with lr models and parser ensembles. In *Proceedings of EMNLP*.

Shi, T.; Huang, L.; and Lee, L. 2017. Fast (er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of EMNLP*.

Silveira, N.; Dozat, T.; De Marneffe, M.-C.; Bowman, S. R.; Connor, M.; Bauer, J.; and Manning, C. D. 2014. A gold standard dependency corpus for english. In *LREC*.

Stenetorp, P. 2013. Transition-based dependency parsing using recursive neural networks. In *NIPS Workshop on Deep Learning*. Citeseer.

Surdeanu, M., and Manning, C. D. 2010. Ensemble models for dependency parsing: cheap and good? In *Proceedings of NAACL*.

Toutanova, K.; Klein, D.; Manning, C. D.; and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL*.

Weiss, D.; Alberti, C.; Collins, M.; and Petrov, S. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL*.

Yamada, H., and Matsumoto, Y. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, 195–206. Nancy, France.

Zhang, Y., and Clark, S. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*.

Zhang, X.; Cheng, J.; and Lapata, M. 2017. Dependency parsing as head selection. In *Proceedings of EACL*.