The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

A Neural Transition-Based Approach for Semantic Dependency Graph Parsing

Yuxuan Wang, Wanxiang Che,* Jiang Guo, Ting Liu
Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, Harbin, China
{yxwang, car, jguo, tliu}@ir.hit.edu.cn

Abstract

Semantic dependency graph has been recently proposed as an extension of tree-structured syntactic or semantic representation for natural language sentences. It particularly features the structural property of *multi-head*, which allows nodes to have multiple heads, resulting in a *directed acyclic graph* (DAG) parsing problem. Yet most statistical parsers focused exclusively on shallow bi-lexical tree structures, DAG parsing remains under-explored. In this paper, we propose a neural transition-based parser, using a variant of list-based arc-eager transition algorithm for dependency graph parsing. Particularly, two non-trivial improvements are proposed for representing the key components of the transition system, to better capture the semantics of segments and internal sub-graph structures. We test our parser on the SemEval-2016 Task 9 dataset (Chinese) and the SemEval-2015 Task 18 dataset (English). On both benchmark datasets, we obtain superior or comparable results to the best performing systems. Our parser can be further improved with a simple ensemble mechanism, resulting in the state-of-the-art performance.

Introduction

Tree-structured bi-lexical dependencies, either syntactic or semantic, have been extensively studied in the past decade. Various algorithms have been proposed for parsing natural language sentences into such tree structures and obtained great success, especially when equipped with deep neural models. However, tree structures become inadequate when moving from syntactic or shallow semantic level to deep semantic level. In the deep semantic representation, a word can be the argument of multiple predicates, and vacuous words might be unattached, which results in a directed acyclic graph (DAG) structure, where a node may have multiple or no incoming arcs. To cope with these challenges, several semantic-oriented dependencies have been recently proposed, typically including the *Chinese Semantic Dependency Graph* (Che et al. 2016) and the *Broad-Coverage Semantic Dependency Graph* with three kinds of formalisms (Oepen et al. 2015), which will be our main focus in this paper.

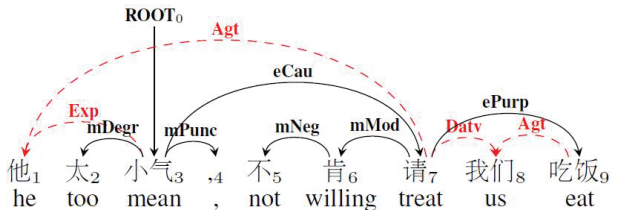
Most existing parsing algorithms were proposed exclusively for tree structures, and the problem of dependency graph parsing (more generally, DAG parsing) remains under-explored. Developing elegant, efficient and effective algorithms for DAG parsing is still an important challenge of great significance. One recent success towards this goal is the model of Peng, Thomson, and Smith (2017), which achieves state-of-the-art performance in the SemEval-2015 Task 18 dataset. Their model is a graph-based one and decodes with the AD^3 algorithm (Martins et al. 2011). Ding et al. (2014) instead uses a two-stage approach, by first producing a semantic dependency tree with a structured prediction model, followed with a classification procedure to recover the *non-local dependencies* (Sun et al. 2014) (i.e. arcs pointing to words with multiple heads). A related approach was studied by Sagae and Tsujii (2008), which projectivize the dependency graphs first, and then learn a projective dependency graph parser. This kind of approaches, however, require either elaborately-designed linguistic rules or complex pre- and post-processing, and suffer from error propagation.

In this paper, we propose a neural transition-based alternative for parsing semantic dependency graphs. We introduce a novel transition algorithm which is a variant of the list-based arc-eager algorithm described originally for non-projective tree parsing (Choi and McCallum 2013). We show how the algorithm can be effectively adapted to parsing DAG structures. Furthermore, the main challenge for a transition-based parsing system to success is the representation of parsing states (i.e. configurations), based on which the transition actions are predicted. To cope with this challenge, we borrow the successful idea from the Stack-LSTM neural parsing model (Dyer et al. 2015), and present two non-trivial improvements, namely *Bi-LSTM Subtraction* and *Incremental Tree-LSTM*, to better capture the semantics of the segments (i.e. spans) and the partially derived sub-graph structures.

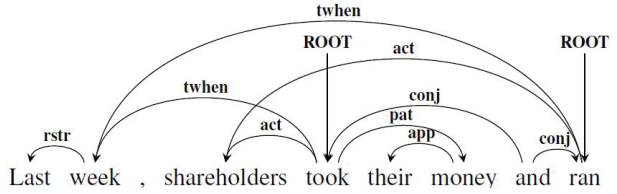
We use SemEval-2016 Task 9: *Chinese Semantic Dependency Parsing* (Che et al. 2016)[1] (Chinese) and SemEval-2015 Task 18: *Broad-Coverage Semantic Dependency Parsing* (Oepen et al. 2015)[2] (English) as our testbed. Experimental results are promising on both datasets. For Chinese,

*Email corresponding

[1] alt.qcri.org/semeval2016/task9/
[2] alt.qcri.org/semeval2015/task18/



(a) An example of CSDG. The dashed arcs indicate non-local dependencies. ("He is too mean to treat us to a meal.")



(b) An example sentence annotated with PSD semantic formalism in BCSDG.

Figure 1: Examples of CSDG and BCSDG.

we outperform all participating systems by a substantial margin, and for English, we obtain superior or comparable results to the participating systems. An ensemble of 3 models using a simple mechanism (Che et al., 2017) leads to further significant improvements, outperforming the state-of-the-art system of Peng, Thomson, and Smith (2017) on the macro-averages of three graph formalisms. Our system will be publicly available at https://github.com/HITalexwang/lstm-sdparser.

Background

Semantic Dependency Graph

Here we briefly introduce and compare the different semantic dependency graph formalisms presented respectively in SemEval-2016 Task 9 and SemEval-2015 Task 18.

Chinese Semantic Dependency Graph (CSDG) Figure 1a shows an example of a Chinese semantic dependency graph presented in SemEval-2016 Task 9. We can see that "he" has two head words, i.e., "mean" with the *Experiencer* (Exp) relation and "treat" with the *Agent* (Agt) relation, and every word is attached. More details of the corpora will be described in experiment section.

Broad-Coverage Semantic Dependency Graph (BCSDG) SemEval-2015 Task 18 provides three graph formalisms: *DM*, *PAS* and *PSD* which have different dependency annotations. For simplicity, we present here an example of *PSD* representation in Figure 1b to show the general structure.

In general, CSDG is similar with BCSDG. Main differences between CSDG and BCSDG are three-fold:

1. The ROOT node can only have one child in CSDG, but in BCSDG it may have more than one (e.g. in Figure 1b ROOT has two children *took* and *ran*).

Transitions	Change of State
LEFT$_l$-REDUCE	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{(\sigma,\ \delta,\ [j\|\beta],\ A \cup \{(i \overset{l}{\leftarrow} j)\})}$
RIGHT$_l$-SHIFT	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{([\sigma\|i\|\delta\|j],\ [\,],\ \beta,\ A \cup \{(i \overset{l}{\rightarrow} j)\})}$
NO-SHIFT	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{([\sigma\|i\|\delta\|j],\ [\,],\ \beta,\ A)}$
NO-REDUCE	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{(\sigma,\ \delta,\ [j\|\beta],\ A)}$
LEFT$_l$-PASS	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{(\sigma,\ [i\|\delta],\ [j\|\beta],\ A \cup \{(i \overset{l}{\leftarrow} j)\})}$
RIGHT$_l$-PASS	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{(\sigma,\ [i\|\delta],\ [j\|\beta],\ A \cup \{(i \overset{l}{\rightarrow} j)\})}$
NO-PASS	$\frac{([\sigma\|i],\ \delta,\ [j\|\beta],\ A)}{(\sigma,\ [i\|\delta],\ [j\|\beta],\ A)}$

Table 1: Transitions defined in the list-based arc-eager algorithm (Choi and McCallum 2013).

2. In CSDG the child of ROOT node is not allowed to have other heads, while in BCSDG it is acceptable (e.g. in Figure 1b the child *took* of ROOT has another head *and*).
3. The restriction in CSDG that each word has to have at least one head does not hold in BCSDG (e.g. in Figure 1b *and* has no head).

Comparison with SRL and AMR Compared with the dependency-based semantic role labeling (SRL) representation (Surdeanu et al. 2008; Hajič et al. 2009) which only considers dependencies between predicates and their arguments, semantic dependency graphs assume relation between every possible word pair, making the task much more challenging.

Semantic dependency graph is also distinct from Abstract Meaning Representation (AMR) presented by Banarescu et al. (2013), in which the vertexes are abstract concepts with no explicit alignment to tokens in the sentence.

Transition-Based Dependency Parsing

Transition-based dependency parsers (Nivre 2008) generate a dependency structure by predicting a transition action sequence. Typically, a transition system consists of a stack σ containing words being processed, a buffer β containing words to be processed and a memory A that holds the generated dependency arcs which form the partially constructed trees. A sequence of predefined transitions are incrementally produced to construct the dependency tree or graph of one sentence. At each parsing state, the next transition to be taken is either decided according to the gold structure while training or predicted by a classifier at test time.

Choi and McCallum (2013) proposed a transition-based dependency parsing algorithm for non-projective trees,

Transitions	Preconditions of Transitions in List-Based Arc-Eager Algorithm	
	Tree-Parsing	**Graph-Parsing**
LEFT$_l$-$*$	$[i \neq 0] \wedge \neg[(i \rightarrow^* j) \in A] \wedge \neg[\exists k.(i \leftarrow k) \in A]$	$[i \neq 0] \wedge \neg[(i \rightarrow^* j) \in A]$
RIGHT$_l$-$*$	$\neg[(j \rightarrow^* i) \in A] \wedge \neg[\exists k.(k \rightarrow j) \in A]$	$\neg[(j \rightarrow^* i) \in A]$
$*$-REDUCE	$[\exists h.(h \rightarrow i) \in A] \wedge \neg[\exists k \in \beta.(i \rightarrow k)]$	$[\exists h.(h \rightarrow i) \in A] \wedge \neg[\exists k \in \beta.(i \rightarrow k) \vee (i \leftarrow k)]$
$*$-SHIFT	$\neg[\exists k \in \sigma.(k \neq i) \wedge ((k \rightarrow j) \vee (k \leftarrow j))]$	

Table 2: Comparisons in terms of the preconditions of Left$_l$-$*$, Right$_l$-$*$ and $*$-Reduce in our graph-parsing algorithm and the tree-parsing algorithm.

State	Transition	σ	δ	β	A
0	Initialization	$[0]$	$[\,]$	$[1,\ldots,9]$	$\emptyset$
1	NO-SHIFT	$[0,1]$	$[\,]$	$[2,\ldots,9]$	
2	NO-SHIFT	$[0,1,2]$	$[\,]$	$[3,\ldots,9]$	
3	LEFT-REDUCE	$[0,1]$	$[\,]$	$[3,\ldots,9]$	$A \cup \{2 \leftarrow \text{mDegr} - 3\}$
4	LEFT-PASS	$[0]$	$[1]$	$[3,\ldots,9]$	$A \cup \{1 \leftarrow \text{Exp} - 3\}$
5	RIGHT-SHIFT	$[0,1,3]$	$[\,]$	$[4,\ldots,9]$	$A \cup \{0 - \text{ROOT} \rightarrow 3\}$
6	RIGHT-SHIFT	$[0,1,3,4]$	$[\,]$	$[5,\ldots,9]$	$A \cup \{3 - \text{mPunc} \rightarrow 4\}$
7	NO-REDUCE	$[0,1,3]$	$[\,]$	$[5,\ldots,9]$	
8	NO-SHIFT	$[0,1,3,5]$	$[\,]$	$[6,\ldots,9]$	
9	LEFT-REDUCE	$[0,1,3]$	$[\,]$	$[6,\ldots,9]$	$A \cup \{5 \leftarrow \text{mNeg} - 6\}$
10	NO-SHIFT	$[0,1,3,6]$	$[\,]$	$[7,8,9]$	
11	LEFT-REDUCE	$[0,1,3]$	$[\,]$	$[7,8,9]$	$A \cup \{6 \leftarrow \text{mMod} - 7\}$
12	RIGHT-PASS	$[0,1]$	$[3]$	$[7,8,9]$	$A \cup \{3 - \text{eCau} \rightarrow 7\}$
13	LEFT-REDUCE	$[0]$	$[3]$	$[7,8,9]$	$A \cup \{1 \leftarrow \text{Agt} - 7\}$
14	NO-SHIFT	$[0,3,7]$	$[\,]$	$[8,9]$	
15	RIGHT-SHIFT	$[0,3,7,8]$	$[\,]$	$[9]$	$A \cup \{7 - \text{Datv} \rightarrow 8\}$
16	LEFT-REDUCE	$[0,3,7]$	$[\,]$	$[9]$	$A \cup \{8 \leftarrow \text{Agt} - 9\}$
17	RIGHT-SHIFT	$[0,3,7,9]$	$[\,]$	$[\,]$	$A \cup \{7 - \text{ePurp} \rightarrow 9\}$

Table 3: Transition sequence for the dependency graph in Figure 1a generated by list-based arc-eager graph-parsing algorithm with gold-standard oracles.

which is a hybrid between Nivre's arc-eager and list-based algorithms (Nivre 2003; 2008).[3] In this algorithm, a tuple $(\sigma, \delta, \beta, A)$ is used to represent each parsing state, where σ is a stack holding processed words, δ is a deque holding words popped out of σ that will be pushed back in the future, and β is a buffer holding unprocessed words. A is a set of labeled dependency arcs. We use index i to represent word w_i, and index 0 to represent the pseudo root of the graph w_0. The initial state is $([0], [\,], [1, \cdots, n], \emptyset)$, while the terminal state is $(\sigma, \delta, [\,], A)$. During parsing, arcs will only be generated between the top element of σ (w_i) and of β (w_j).

The transition actions of the list-based arc-eager algorithm are listed in Table 1. LEFT$_l$-$*$ and RIGHT$_l$-$*$ add an arc with label l betweem w_j amd w_i. They are performed only when one of w_i and w_j is the head of the other. Otherwise, NO-$*$ will be applied. $*$-SHIFT is performed when no dependency exists between w_j and any word in σ other than w_i, which pushes all words in δ and w_j into σ. $*$-REDUCE is performed only when w_i has head and is not the head or child of any word in β, which pops w_i out of σ. $*$-PASS is performed when neither $*$-SHIFT nor $*$-REDUCE can be performed, which moves w_i to the front of δ. $(i \xrightarrow{l} j)$ is used to denote an arc from w_i to w_j with label l. $(i \rightarrow j)$ and $(i \rightarrow^* j)$ indicate that w_i is a head and an ancestor of w_j respectively.

[3]We refer to the system proposed by Choi and McCallum (2013) as *list-based arc-eager* algorithm in this paper.

Neural Networks for Parsing

Recent years have seen great success in applying neural networks to transition-based dependency parsing (Chen and Manning 2014; Dyer et al. 2015; Andor et al. 2016). The use of neural networks not only avoids the heavy labour of feature designing, but also captures much more information including those that may have not been considered by human experts. Additionally, it also saves a lot of time that was wasted in feature combination. Our work is mainly inspired by the Stack-LSTM model proposed by Dyer et al. (2015), which depended on an arc-standard algorithm (Nivre 2008). Three Stack-LSTMs are utilized to incrementally obtain the representations of the buffer, the stack and the transition action sequence. Stack LSTM is an unidirectional LSTM augmented with a stack pointer. When computing a new memory cell, the current location of this stack pointer determines which cell in the LSTM provides the information of previous time step. In addition, a dependency-based Recursive Neural Network (RecNN) is used to compute the partially constructed tree representations. In this way, full information of each parsing state is expected to be captured, alleviating the labour-intensive feature engineering.

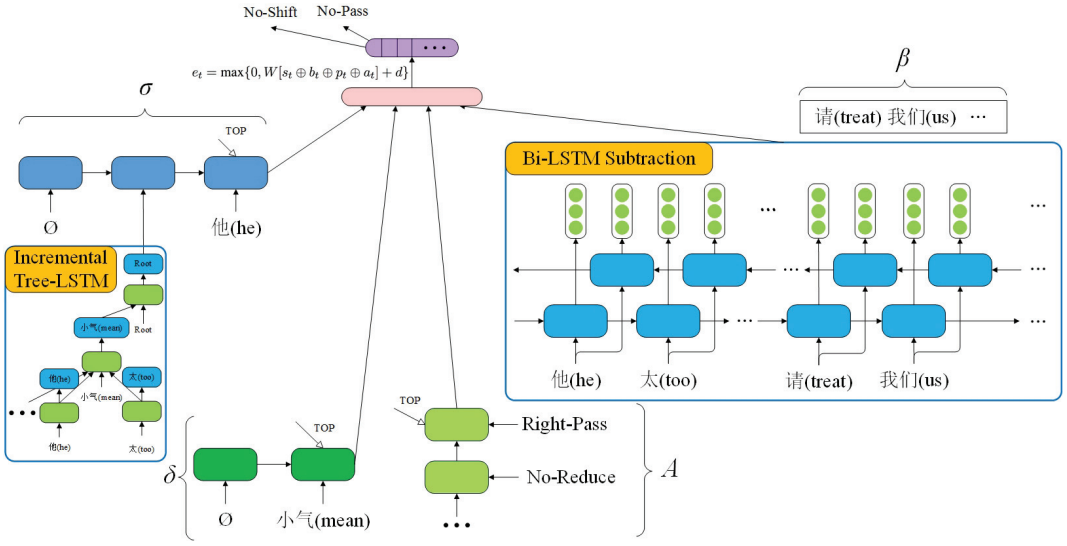


Figure 2: Example transition state representation based on LSTMs. The buffer β is represented by Bi-LSTM Subtraction, the sub-graphs are computed by Incremental Tree-LSTM.

Neural Transition-based DAG Parser

Transition System

The key challenge in dependency graph parsing is the generation of non-local dependencies. To produce these dependencies, we modify the preconditions of LEFT$_l$-$*$ and RIGHT$_l$-$*$ so that the dependency between a head and its modifier can be generated even if the modifier already has a head. Table 2 shows the specific differences between the original tree-parsing algorithm and the proposed variant. Besides, we want to confirm that all of a word's heads and children are found before it is reduced. So we modify the head confirmation in the precondition of $*$-REDUCE to make sure no extra head of w_i is in the buffer β. While w_i is only validated to have one head in the tree-parsing algorithm since each word has only one head in the tree structure.

Table 3 illustrates the transition sequence for the dependency graph in Figure 1a using the modified algorithm. In state 4, w_1 is moved from σ to δ because it has another head w_7 in β, and the arc between them will be generated in the future (state 13). In state 12, the transition is RIGHT-PASS rather than RIGHT-SHIFT because w_7 still has a child w_1 in σ, and w_3 is moved to δ so that the arc between w_7 and w_1 can be generated (state 13).

LSTM-Based State Representation

Compared with the arc-standard algorithm (Nivre 2004) used by Dyer et al. (2015), our transition system has an extra component in each configuration, i.e., the deque δ. So we use an additional Stack-LSTM to learn the representation of δ. More importantly, we introduce two LSTM-based techniques, namely *Bi-LSTM Subtraction* and *Incremental Tree-LSTM* (explained below) for modeling the buffer and sub-graph representations in our system.

Representations of the four components are then concatenated and passed through a hidden layer to obtain the repre-

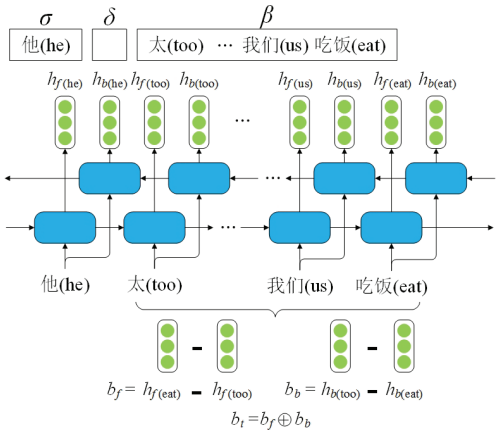


Figure 3: Illustration of Bi-LSTM Subtraction for buffer representation learning. $h_f(*)$ and $h_b(*)$ indicate the hidden vectors of forward and backward LSTM respectively. b_t is the resulting buffer representation.

sentation of the parsing state at time t:

$$e_t = \max\{0, W[s_t \oplus b_t \oplus p_t \oplus a_t] + d\}$$

where s_t, b_t, p_t and a_t are the representation of σ, β, δ and A respectively. d is the bias. $\oplus$ is the concatenation operator. e_t is finally used to compute the probability distribution of possible transition actions at time t through a softmax layer. Figure 2 shows the architecture.

Bi-LSTM Subtraction In Dyer et al. (2015)'s work, the buffer is simply represented by the last hidden vector of a right-to-left unidirectional LSTM encoding words in the buffer, which has the risk of losing information from words out of the buffer (which have been shifted or reduced)

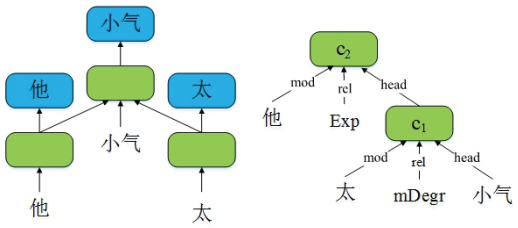


Figure 4: Representations of a dependency sub-graph (above) computed by Tree-LSTM (left) and dependency-based RecNN (right).

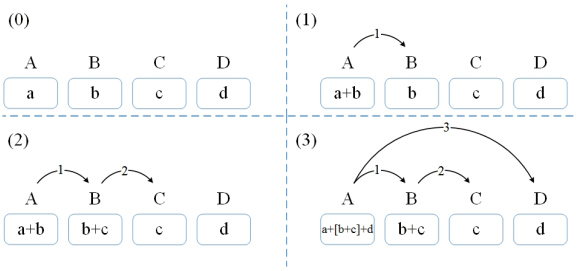


Figure 5: Illustration of the incremental construction process of a sub-graph. a, b, c and d indicate the embeddings of A, B, C and D. a+b indicates the combination of A and B.

and from the left-to-right direction. Besides, a single hidden vector is not substantial for representing the whole buffer. Here, we regard the buffer as a *segment* and use the subtraction between LSTM hidden vectors of the segment head and tail as its representation. Similar idea was also explored in Wang and Chang (2016) and Cross and Huang (2016b). To include the information of words out of the buffer, we apply subtraction on bidirectional LSTM representations over the whole sentence (Wang et al. 2016; Kiperwasser and Goldberg 2016; Cross and Huang 2016a), thus called *Bi-LSTM Subtraction*. The forward and backward subtractions are calculated independently, i.e., $b_f = h_f(l) - h_f(f)$ and $b_b = b_b(f) - b_b(l)$, where $h_f(f)$ and $h_f(l)$ are the hidden vectors of the first and the last words in the forward LSTM, $h_b(f)$ and $h_b(l)$ are the hidden vectors of the first and the last words in the backward LSTM. Then b_f and b_b are concatenated as the buffer representation. As illustrated in Figure 3, the forward and backward subtractions for the buffer are $b_f = h_f(eat) - h_f(too)$ and $b_b = h_b(too) - h_b(eat)$ respectively.

Incremental Tree-LSTM Dyer et al. (2015) have used a dependency-based Recursive Neural Network (RecNN) to compute the subtree representations, which has a risk of suffering from gradient vanishing when dealing with deep subtrees. Furthermore, we also believe that the architecture used to model the sub-structures requires the capability to memorize important information and forget unimportant one. So we use a Tree-LSTM (Tai, Socher, and Manning 2015) instead of RecNN in our architecture. The example in Figure 4 shows the differences between RecNN and Tree-LSTM. In RecNN, the representation of a sub-graph is computed by recursively combining head-modifier pairs, whereas in Tree-LSTM, a head is combined with all of its modifiers simultaneously in each LSTM unit.

However, due to the particularities of graph parsing, our implementation of Tree-LSTM is different from the conventional one. First of all, the partially constructed structures generated in our parsing procedure are not strictly trees, since some of the words may have more than one head. Fortunately, the structures of dependency graphs are constrained to be directed acyclic, which makes the Tree-LSTM still applicable. More importantly, unlike traditional bottom-up Tree-LSTMs in which each head and all of its modifiers are combined simultaneously, the modifiers are found incrementally during our parsing procedure. Therefore, we propose *Incremental Tree-LSTM*, which obtains sub-graph representations incrementally. To be more specific, each time a dependency arc is generated, we collect representations of all the found modifiers of the head and combine them along with the embedding of the head as the representation of the sub-graph. The original embedding rather than the current representation of the head is utilized to avoid the reusage of modifier information, since the current representation of the head contains information of its modifiers found previously.

For example, in the sub-graph depicted in Figure 5, the sub-graph is constructed as follows: (1) embeddings of A and B are combined and take the place A;[4] (2) embeddings of B and C are combined and take the place of B; (3) embeddings of A and D along with current representation of B (combination of B and C) are combined as the representation of the final sub-graph which updates A. Note that in RecNN, after the generation of the third arc, the representation of A (combination of A and B) and the newly found modifier D will be combined as the final representation of the sub-graph. Thus the information of dependency relation between B and C will be lost.[5]

Although our Incremental Tree-LSTM can cover more situations than RecNN does, it still has limitations when dealing with certain structures. For example, in Figure 5, if the dependency between A and D is not generated, then the final representation of the sub-graph will be the combination of embeddings of A and B without C. An ideal solution which we have tried is to update all of the ancestors of the head whenever an arc is generated. However, preliminary experiment results show that updating all the ancestors can barely help to improve the final performance in both CSDG and BCSDG. We believe it is because that updating all the ancestors will not only improve the performance of the Tree-LSTM but also amplify the structural errors. Thus we eventually choose the simplified version.

In addition, on the contrary to RecNN, our Tree-LSTM does not model the dependency relations. We conduct an experiment in a basic system with the same architecture of Dyer et al. (2015) and found that neglecting relations in

[4]The embeddings are combined through a Tree-LSTM unit.

[5]It is important to note that the problem does not exist in the arc-standard algorithm used by Dyer et al. (2015) since the trees are constructed in a bottom-up manner.

RecNN has only marginal effect on the results.

Experiments

Data and Settings

For Chinese, we use the SemEval-2016 Task 9 as our testbed. The shared task provides two distinguished corpora in the domain of NEWS and TEXTBOOKS. We follow the official evaluation setup (Che et al. 2016) and use the labeled F-score (**LF**) and unlabeled F-score (**UF**) at the dependency arcs level as the metrics. In particular, non-local dependencies are evaluated separately, i.e., **NLF** and **NUF**.

For English, we conduct experiments on the English part of SemEval-2015 Task 18 closed track (Oepen et al. 2015). We use the same data split as previous work (Almeida and Martins 2015; Du et al. 2015), with 33,964 training sentences (WSJ §00-19), 1,692 development sentences (§20), 1,410 in-domain testing sentences (§21) and 1,849 out-of-domain testing sentences from the Brown Corpus. We utilize the official evaluation tool to evaluate our results.

DyNet (Neubig et al. 2017) is used to implement our neural models.[6] Similar to Dyer et al. (2015), we utilize three types of atomic features to represent the tokens: pre-trained word embeddings, POS tags and randomly initialized word embeddings updated while training. For Chinese we use 100-dimensional word embeddings trained by word2vec (Mikolov et al. 2013) with Xinhua portion of the Chinese Gigawords.[7] For English we use 50-dimensional SENNA embeddings.[8]

The Stack-LSTMs and Bi-LSTM have two layers while the Tree-LSTM has one. The input and hidden dimensions of Stack-LSTM, Bi-LSTM and Tree-LSTM are 200. The learned word embedding size $d_{wt} = 100$, POS tag, relation and transition embedding size are all 50.

Results and Discussions

We first explore the effect of Bi-LSTM Subtraction (BS) and Incremental Tree-LSTM (IT) with the CSDG dataset. Results are shown in Table 4. We can see that either module can improve the performance of the **basic system**, which uses unidirectional LSTM to represent the buffer and RecNN to compute the sub-graphs (Dyer et al. 2015), and the system that uses both yields the best results. The speed comparison in Table 5 shows that both modules slow down the basic system by about 10%. And updating all ancestors in Tree-LSTM also slows down the BS-IT system by about 10%.

The overall results on CSDG are presented in Table 6. To further show the effectiveness of our method, we compare our system with a strong baseline approach proposed by Ding et al. (2014), where a conventional transition-based tree parser is utilized to produce dependency trees first, then an SVM classifier is used to recover the additional dependency arcs by selecting arcs from a set of candidates generated by manually designed linguistic rules. In the work of Ding et al. (2014), they used structured perceptron (Collins 2002) to train the classifier. Here we instead use our proposed neural network architecture to exclude the effect of transition classification and focus on the differences between our one-stage approach and their two-stage one. Results show that our implementation of the two-stage system achieves better performance than other participating systems[9] in all evaluation metrics except for NLF, indicating that the two-stage approach is still weak in predicting non-local dependencies. However, our proposed BS-IT system significantly outperforms it and all the other participating systems. Besides, our BS-IT System achieves an absolute improvement of more than 10% in NLF against the two-stage system on both copora, showing the superiority of our proposed method in dealing with non-local dependencies.

We further evaluate our system (BS-IT) on the BCSDG dataset (English). Since the restriction in our transition system that each word must have at least one head does not hold in this task, we pre-process these sentences by attaching the words that have no head to ROOT with an extra NULL label. At test time, arcs with NULL labels will be removed.

Table 7 compares our system with the best published results on SemEval-2015 Task 18 English test sets. Here Du et al. (2015)'s parser is a hybrid of transition-based and graph-based parsing approaches, and also utilizes model ensembling. A&M, 2015 (Almeida and Martins 2015) is a second-order graph-based parser. Peng et al.,2017 (Peng, Thomson, and Smith 2017) is also a graph-based parser and yields state-of-the-art performance on the English part of this task. Both A&M and Peng et al.'s models used AD^3 algorithm for decoding, while Peng et al. used bidirectional LSTMs composed with a multi-layer perceptron to score dependency arcs. They also explored multi-task learning approaches to combine the three different annotations (*DM*, *PAS*, *PSD*), but we only compare with their basic models here.

Corpus	System	LF	UF	NLF	NUF
NEWS	Basic	62.23	80.42	49.18	63.90
	+BS	62.59	80.61	49.83	65.80
	+IT	62.62	80.50	49.25	65.13
	+BS&IT	**63.30**	**81.14**	**51.16**	**66.92**
TEXT	Basic	71.51	84.95	59.70	71.63
	+BS	72.29	85.41	59.98	71.68
	+IT	72.20	85.33	60.61	71.58
	+BS&IT	**72.92**	**85.71**	**61.91**	**72.74**

Table 4: Effect of the Bi-LSTM Subtraction (BS) module for buffer representation and the Incremental Tree-LSTM (IT) module for sub-graph representation using the one-stage system on Chinese test set (CSDG).

Basic	+BS	+IT	+BS&IT	+BS&IT (Anc)
335.0	296.9	302.6	258.1	229.7

Table 5: Speed (tokens/s) comparison on CSDG TEXTBOOK set. **Anc** means update all ancestors in Tree-LSTM.

[6] github.com/clab/dynet

[7] https://catalog.ldc.upenn.edu/LDC2003T09

[8] https://ronan.collobert.com/senna/

[9] Refer to Che et al. (2016) for more details.

System	NEWS				TEXTBOOKS			
	LF	UF	NLF	NUF	LF	UF	NLF	NUF
IHS-RD-Belarus	59.06	77.64	40.84	60.20	68.59	82.41	50.57	64.58
OCLSP (lbpg)	57.22	74.93	45.57	58.03	65.54	79.39	51.75	63.21
OCLSP (lbpgs)	57.81	75.54	41.56	54.34	66.21	79.85	47.79	55.51
OCLSP (lbpg75)	57.78	75.40	48.89	58.28	66.38	79.91	57.51	63.87
OSU_CHGCG	55.69	73.72	49.23	60.71	65.17	78.83	54.70	65.71
2-stage (Ding et al. 2014)	62.29	80.56	39.93	64.29	71.94	85.24	50.67	69.97
BS-IT	**63.30**	**81.14**	**51.16**	**66.92**	**72.92**	**85.71**	**61.91**	**72.74**

Table 6: Results of our systems (the bottom one), our implementation of Ding et al. (2014)'s 2-stage system (the middle one) and other participating systems (the top five) on Chinese test set (CSDG).

System	DM	PAS	PSD	Avg.
IN-DOMAIN				
Du et al., 2015 (ensemble)	89.1	91.3	75.7	85.4
A&M, 2015 (single)	88.2	90.9	76.4	85.2
Peng et al., 2017 (single)	89.4	**92.2**	77.6	86.4
BS-IT (single)	89.3	91.4	76.1	85.6
BS-IT (ensemble)	**90.3**	91.7	**78.6**	**86.9**
OUT-OF-DOMAIN				
Du et al., 2015 (ensemble)	81.8	87.2	73.3	80.8
A&M, 2015 (single)	81.8	86.9	74.8	81.2
Peng et al., 2017 (single)	84.5	**88.3**	75.3	82.7
BS-IT (single)	83.2	87.2	73.2	81.2
BS-IT (ensemble)	**84.9**	87.6	**75.9**	**82.8**

Table 7: Labeled F_1 score on the BCSDG English test set. *DM*, *PAS* and *PSD* are the three types of annotations on the same dataset. The last column shows the macro-average over the three tasks. McNemar's test shows that our ensemble model outperforms Peng et al.'s model in DM and PSD annotations with p-value < 0.001.

Results of our single model are either comparable with or better than that of Du et al. (2015) and Almeida and Martins (2015)'s systems. Since Du et al. (2015)'s model is an ensembled one of 14 models, we also utilize a simple model ensemble mechanism similar to Che et al. (2017), which trains three models separately with different random seeds and while predicting, use the sum of scores predicted by the three models to decide the next transition at each state. With this simple mechanism, we achieve better results than the state-of-the-art one (Peng, Thomson, and Smith 2017) in 4 settings out of 6, and also outperform them on the macro-averages of both in-domain and out-of-domain datasets.

Related Work

Deep dependency structures beyond surface tree representation are traditionally obtained as a by-product of grammar-guided parsers grounded by CCG, LFG and HPSG (Clark, Hockenmaier, and Steedman 2002; Miyao and Tsujii 2005; Sagae, Miyao, and Tsujii 2007). However, most of the previous work on dependency parsing focused on tree structures, whereas recent work has proved that transition-based approaches are also applicable to generate more informative dependency graphs. Sagae and Tsujii (2008) presented a transition-based approach to generate HPSG-style predicate-argument structures using pseudo-projective transformations (Nivre et al. 2006). Following their step, Tokgöz and Eryiğit (2015) presented a DAG parser with dynamic oracle, which still requires pseudo-projective transformation and detransformation. Recently, Zhang et al. (2016) presented two new transition systems to generate arbitrary directed graphs in an incremental manner with structured perceptron algorithm (Collins 2002) for transition classification. Except for the differences between our and their transition systems, we also make good use of neural networks in our model.

Conclusion

This paper presents a neural transition-based approach for semantic dependency graph parsing. We describe a variant of list-based arc-eager transition algorithm that is capable of producing DAG structures. Furthermore, two improved LSTM modules are proposed for representing the key components in our transition system. Experimental results on SemEval-2016 Task 9 and SemEval-2015 Task 18 which defined different kinds of graph formalisms in different languages (i.e. Chinese and English) demonstrate the effectiveness of our approach.

Acknowledgments

We thank the anonymous reviewers for their valuable suggestions. This work was supported by the National Key Basic Research Program of China via grant 2014CB340503 and the National Natural Science Foundation of China (NSFC) via grant 61370164 and 61632011.

References

Almeida, M. S. C., and Martins, A. F. T. 2015. Lisbon: Evaluating turbosemanticparser on multiple languages and out-of-domain data. In *Proc. of SemEval*, 970–973.

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 *Proc. of ACL*, 2442–2452.

Banarescu, L.; Bonial, C.; Cai, S.; Georgescu, M.; Griffitt, K.; Hermjakob, U.; Knight, K.; Koehn, P.; Palmer, M.; and Schneider, N. 2013. Abstract meaning representation for sembanking. In *Proc. of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 178–186.

Che, W.; Shao, Y.; Liu, T.; and Ding, Y. 2016. Semeval-2016 task 9: Chinese semantic dependency parsing. In *Proc. of SemEval*, 1074–1080.

Che, W.; Guo, J.; Wang, Y.; Zheng, B.; Zhao, H.; Liu, Y.; Teng, D.; and Liu, T. 2017. The hit-scir system for end-to-end parsing of universal dependencies. In *Proc. of CoNLL*, 52–62.

Chen, D., and Manning, C. 2014. A fast and accurate dependency parser using neural networks. In *Proc. of EMNLP*, 740–750.

Choi, J. D., and McCallum, A. 2013. Transition-based dependency parsing with selectional branching. In *Proc. of ACL*, 1052–1062.

Clark, S.; Hockenmaier, J.; and Steedman, M. 2002. Building deep dependency structures with a wide-coverage ccg parser. In *Proc. of ACL*, 327–334.

Collins, M. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, 1–8.

Cross, J., and Huang, L. 2016a. Incremental parsing with minimal features using bi-directional lstm. In *Proc. of ACL*, 32–37.

Cross, J., and Huang, L. 2016b. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proc. of EMNLP*, 1–11.

Ding, Y.; Shao, Y.; Che, W.; and Liu, T. 2014. Dependency graph based chinese semantic parsing. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*. Springer. 58–69.

Du, Y.; Zhang, F.; Zhang, X.; Sun, W.; and Wan, X. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proc. of SemEval*, 927–931.

Dyer, C.; Ballesteros, M.; Ling, W.; Matthews, A.; and Smith, N. A. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. of ACL and IJCNLP*, 334–343.

Hajič, J.; Ciaramita, M.; Johansson, R.; Kawahara, D.; Martí, M. A.; Màrquez, L.; Meyers, A.; Nivre, J.; Padó, S.; Štěpánek, J.; Straňák, P.; Surdeanu, M.; Xue, N.; and Zhang, Y. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proc. of CoNLL*, 1–18.

Kiperwasser, E., and Goldberg, Y. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *TACL* 4:313–327.

Martins, A. F. T.; Smith, N. A.; Aguiar, P. M. Q.; and Figueiredo, M. A. T. 2011. Dual decomposition with many overlapping components. In *Proc. of EMNLP*, 238–249.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *ICLR Workshop*.

Miyao, Y., and Tsujii, J. 2005. Probabilistic disambiguation models for wide-coverage hpsg parsing. In *Proc. of ACL*, 83–90.

Neubig, G.; Dyer, C.; Goldberg, Y.; Matthews, A.; Ammar, W.; Anastasopoulos, A.; Ballesteros, M.; Chiang, D.; Clothiaux, D.; Cohn, T.; Duh, K.; Faruqui, M.; Gan, C.; Garrette, D.; Ji, Y.; Kong, L.; Kuncoro, A.; Kumar, G.; Malaviya, C.; Michel, P.; Oda, Y.; Richardson, M.; Saphra, N.; Swayamdipta, S.; and Yin, P. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Nivre, J.; Hall, J.; Nilsson, J.; Eryigit, G.; and Marinov, S. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc. of CoNLL*, 221–225.

Nivre, J. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, 149–160.

Nivre, J. 2004. Incrementality in deterministic dependency parsing. In *Proc. of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, 50–57.

Nivre, J. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553.

Oepen, S.; Kuhlmann, M.; Miyao, Y.; Zeman, D.; Cinkova, S.; Flickinger, D.; Hajic, J.; and Uresova, Z. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, 915–926.

Peng, H.; Thomson, S.; and Smith, N. A. 2017. Deep multitask learning for semantic dependency parsing. In *Proc. of ACL*, 2037–2048.

Sagae, K., and Tsujii, J. 2008. Shift-reduce dependency dag parsing. In *Proc. of ICCL*, 753–760.

Sagae, K.; Miyao, Y.; and Tsujii, J. 2007. Hpsg parsing with shallow dependency constraints. In *Proc. of ACL*, 624–631.

Sun, W.; Du, Y.; Kou, X.; Ding, S.; and Wan, X. 2014. Grammatical relations in chinese: Gb-ground extraction and data-driven parsing. In *Proc. of ACL*, 436–456.

Surdeanu, M.; Johansson, R.; Meyers, A.; Màrquez, L.; and Nivre, J. 2008. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proc. of CoNLL*, 159–177.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. of ACL and IJCNLP*, 1556–1566.

Tokgöz, A., and Eryiğit, G. 2015. Transition-based dependency dag parsing using dynamic oracles. In *Proc. of ACL-IJCNLP 2015 Student Research Workshop*, 22–27.

Wang, W., and Chang, B. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proc. of ACL*, 2306–2315.

Wang, P.; Qian, Y.; Soong, F. K.; He, L.; and Zhao, H. 2016. Learning distributed word representations for bidirectional lstm recurrent neural network. In *Proc. of HLT-NAACL*.

Zhang, X.; Du, Y.; Sun, W.; and Wan, X. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics* 42(3):353–389.