Transition-based Dependency Parsing Using Two Heterogeneous Gated Recursive Neural Networks

Xinchi Chen, Yaqian Zhou, Chenxi Zhu, Xipeng Qiu, Xuanjing Huang
Shanghai Key Laboratory of Intelligent Information Processing, Fudan University
School of Computer Science, Fudan University
825 Zhangheng Road, Shanghai, China
{xinchichen13,zhouyaqian,czhu13,xpqiu,xjhuang}@fudan.edu.cn

Abstract

Recently, neural network based dependency parsing has attracted much interest, which can effectively alleviate the problems of data sparsity and feature engineering by using the dense features. However, it is still a challenge problem to sufficiently model the complicated syntactic and semantic compositions of the dense features in neural network based methods. In this paper, we propose two heterogeneous gated recursive neural networks: tree structured gated recursive neural network (Tree-GRNN) and directed acyclic graph structured gated recursive neural network (DAG-GRNN). Then we integrate them to automatically learn the compositions of the dense features for transition-based dependency parsing. Specifically, Tree-GRNN models the feature combinations for the trees in stack, which already have partial dependency structures. DAG-GRNN models the feature combinations of the nodes whose dependency relations have not been built yet. Experiment results on two prevalent benchmark datasets (PTB3 and CTB5) show the effectiveness of our proposed model.

1 Introduction

Transition-based dependency parsing is a core task in natural language processing, which has been studied with considerable efforts in the NLP community. The traditional discriminative dependency parsing methods have achieved great success (Koo et al., 2008; He et al., 2013; Bohnet, 2010; Huang and Sagae, 2010; Zhang and Nivre, 2011; Martins et al., 2009; McDonald et al., 2005; Nivre et al., 2006; Kübler et al., 2009; Goldberg and Nivre, 2013; Choi and McCallum, 2013; Ballesteros and Bohnet, 2014). However, these methods are based on discrete features and suffer from the problems of data sparsity and feature engineering (Chen and Manning, 2014).

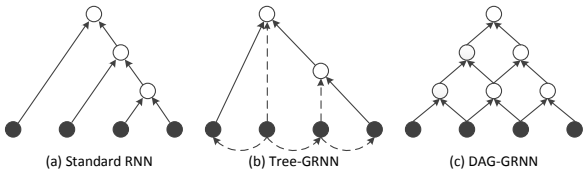

Figure 1: Sketch of three recursive neural networks (RNN). (a) is the standard RNN for constituent tree; (b) is Tree-GRNN for dependency tree, in which the dashed arcs indicate the dependency relations between the nodes; (c) is DAG-GRNN for the nodes without given topological structure.

Recently, distributed representations have been widely used in a variety of natural language processing (NLP) tasks (Collobert et al., 2011; Devlin et al., 2014; Socher et al., 2013; Turian et al., 2010; Mikolov et al., 2013b; Bengio et al., 2003). Specific to the transition-based parsing, the neural network based methods have also been increasingly focused on due to their ability to minimize the efforts in feature engineering and the boosted performance (Le and Zuidema, 2014; Stenetorp, 2013; Bansal et al., 2014; Chen and Manning, 2014; Zhu et al., 2015).

However, most of the existing neural network based methods still need some efforts in feature engineering. For example, most methods often select the first and second leftmost/rightmost children of the top nodes in stack, which could miss some valuable information hidden in the unchosen nodes. Besides, the features of the selected nodes are just simply concatenated and then fed into neural network. Since the concatenation operation is relatively simple, it is difficult to model the com-

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1879–1889,
Lisbon, Portugal, 17-21 September 2015.

plicated feature combinations which can be manually designed in the traditional discrete feature based methods.

To tackle these problems, we use two heterogeneous gated recursive neural networks, tree structured gated recursive neural network (Tree-GRNN) and directed acyclic graph gated structured recursive neural network (DAG-GRNN), to model each configuration during transition based dependency parsing. The two proposed GRNNs introduce the gate mechanism (Chung et al., 2014) to improve the standard recursive neural network (RNN) (Socher et al., 2013; Socher et al., 2014), and can model the syntactic and semantic compositions of the nodes during parsing.

Figure 1 gives a rough sketch for the standard RNN, Tree-GRNN and DAG-GRNN. Tree-GRNN is applied to the partial-constructed trees in stack, which have already been constructed according to the previous transition actions. DAG-GRNN is applied to model the feature composition of nodes in stack and buffer which have not been labeled their dependency relations yet. Intuitively, Tree-GRNN selects and merges features recursively from children nodes into their parent according to their dependency structures, while DAG-GRNN further models the complicated combinations of extracted features and explicitly exploits features in different levels of granularity.

To evaluate our approach, we experiment on two prevalent benchmark datasets: English Penn Treebank 3 (PTB3) and Chinese Penn Treebank 5 (CTB5) datasets. Experiment results show the effectiveness of our proposed method. Compared to the parser of Chen and Manning (2014), we receive 0.6% (UAS) and 0.9% (LAS) improvement on PTB3 test set, while we receive 0.8% (UAS) and 1.3% (LAS) improvement on CTB5 test set.

2 Neural Network Based Transition Dependency Parsing

2.1 Transition Dependency Parsing

In this paper, we employ the arc-standard transition systems (Nivre, 2004) and examine only greedy parsing for its efficiency. Figure 2 gives an example of arc-standard transition dependency parsing.

In transition-based dependency parsing, the consecutive configurations of parsing process can be defined as $c^{(i)} = (s^{(i)}, b^{(i)}, A^{(i)})$ which consists of a stack s, a buffer b, and a set of dependency arcs A. Then, the greedy parsing process consecutively predicts the actions based on the features extracted from the corresponding configurations. For a given sentence $w_1, \dots, w_n$, parsing process starts from a initial configuration $c^{(0)} = ([ROOT], [w_1, \dots, w_n], \emptyset)$, and terminates at some configuration $c^{(2n)} = ([ROOT], \emptyset, A^{(2n)}]$, where n is the length of the given sentence $w_{1:n}$. As a result, we derive the parse tree of the sentence $w_{1:n}$ according to the arcs set $A^{(2n)}$.

In arc-standard system, there are three types of actions: **Left-Arc**, **Right-Arc** and **Shift**. Denoting $s_j(j = 1, 2, \dots)$ as the j^{th} top element of the stack, and $b_j(j = 1, 2, \dots)$ as the j^{th} front element of the buffer, we can formalize the three actions of arc-standard system as:

- **Left-Arc**(l) adds an arc $s_2 \leftarrow s_1$ with label l and removes s_2 from the stack, resulting a new arc $l(s_1, s_2)$. Precondition: $|s| \geq 3$ (The ROOT node cannot be child node).
- **Right-Arc**(l) adds an arc $s_2 \rightarrow s_1$ with label l and removes s_1 from the stack, resulting a new arc $l(s_2, s_1)$. Precondition: $|s| \geq 2$.
- **Shift** removes b_1 from the buffer, and adds it to the stack. Precondition: $|b| \geq 1$.

The greedy parser aims to predict the correct transition action for a given configuration. There are two versions of parsing: unlabeled and labeled versions. The set of possible action candidates $\mathcal{T} = 2n_l + 1$ in the labeled version of parsing, and $\mathcal{T} = 3$ in the unlabeled version, where n_l is number of different types of arc labels.

2.2 Neural Network Based Parser

In neural network architecture, the words, POS tags and arc labels are mapped into distributed vectors (embeddings). Specifically, given the word embedding matrix $\mathbf{E}^w \in \mathbb{R}^{d_e \times n_w}$, each word w_i is mapped into its corresponding column $\mathbf{e}^w_{w_i} \in \mathbb{R}^{d_e}$ of $\mathbf{E}^w$ according to its index in the dictionary, where d_e is the dimensionality of embeddings and n_w is the dictionary size. Likewise, The POS and arc labels are also mapped into embeddings by the POS embedding matrix $\mathbf{E}^t \in \mathbb{R}^{d_e \times n_t}$ and arc label embedding matrix $\mathbf{E}^l \in \mathbb{R}^{d_e \times n_l}$ respectively, where n_t and n_l are the numbers of distinct POS tags and arc labels respectively. Correspondingly, embeddings of each POS tag t_i and each arc label l_i are $\mathbf{e}^t_{t_i} \in \mathbb{R}^{d_e}$ and $\mathbf{e}^l_{l_i} \in \mathbb{R}^{d_e}$ extracted from $\mathbf{E}^t$ and $\mathbf{E}^l$ respectively.

Configurations				Gold Actions
ID	Stack	Buffer	A	
0	[ROOT]	[He likes story books .]	$\emptyset$	
1	[ROOT He]	[likes story books .]		Shift
2	[ROOT He likes]	[story books .]		Shift
3	[ROOT likes]	[story books .]	A $\cup$ SUB(likes,He)	Left_Arc(SUB)
4	[ROOT likes story]	[books .]		Shift
5	[ROOT likes story books]	[.]		Shift
6	[ROOT likes books]	[.]	A$\cup$NMOD(books,story)	Left_Arc(NMOD)
7	[ROOT likes]	[.]	A$\cup$OBJ(likes,books)	Right_Arc(OBJ)
8	[ROOT likes .]	$\emptyset$		Shift
9	[ROOT likes]	$\emptyset$	A$\cup$P(likes,.)	Right_Arc(P)
10	[ROOT]	$\emptyset$	A$\cup$ROOT(ROOT,likes)	Right_Arc(ROOT)

Figure 2: An example of arc-standard transition dependency parsing.

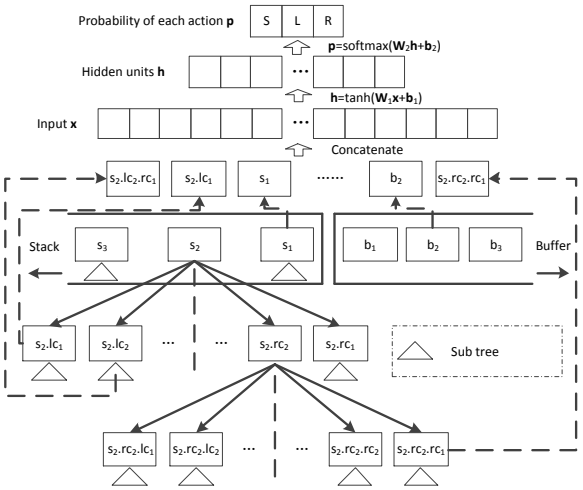

Figure 3: Architecture of neural network based transition dependency parsing.

Figure 3 gives the architecture of neural network based parser. Following Chen and Manning (2014), a set of elements S from stack and buffer (e.g. $S = \{s_2.lc_2.rc_1, s_2.lc_1, s_1, b_2, s_2.rc_2.rc_1, \dots\}$) is chosen as input. Specifically, the information (word, POS or label) of each element in the set S (e.g. $\{s_2.lc_2.rc_1.t, s_2.lc_1.l, s_1.w, s_1.t, b_2.w, \dots\}$) are extracted and mapped into their corresponding embeddings. Then these embeddings are concatenated as the input vector $\mathbf{x} \in \mathbb{R}^{\hat{d}}$. A special token NULL is used to represent a non-existent element.

We perform a standard neural network using one hidden layer with d_h hidden units followed by a softmax layer as:

$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \quad (1)$$

$$\mathbf{p} = \text{softmax}(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2), \quad (2)$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_h \times \hat{d}}$, $\mathbf{b}_1 \in \mathbb{R}^{d_h}$, $\mathbf{W}_2 \in \mathbb{R}^{|\mathcal{T}| \times d_h}$, $\mathbf{b}_2 \in \mathbb{R}^{|\mathcal{T}|}$.Here, g is a non-linear function which can be hyperbolic tangent, sigmoid, cube (Chen and Manning, 2014), etc.

3 Recursive Neural Network

Recursive neural network (RNN) is one of classical neural networks, which performs the same set of parameters recursively on a given structure (e.g. syntactic tree) in topological order (Pollack, 1990; Socher et al., 2013).

In the simplest case, children nodes are combined into their parent node using a weight matrix $\mathbf{W}$ which is shared across the whole network, followed by a non-linear function $g(\cdot)$. Specifically, given the left child node vector $\mathbf{h}_L \in \mathbb{R}^d$ and right child node vector $\mathbf{h}_R \in \mathbb{R}^d$, their parent node vector $\mathbf{h}_P \in \mathbb{R}^d$ will be formalized as:

$$\mathbf{h}_P = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_L \\ \mathbf{h}_R\end{bmatrix}\right), \quad (3)$$

where $\mathbf{W} \in \mathbb{R}^{d \times 2d}$ and g is a non-linear function as mentioned above.

4 Architecture of Two Heterogeneous Gated Recursive Neural Networks for Transition-based Dependency Parsing

In this paper, we apply the idea of recursive neural network (RNN) to dependency parsing task. RNN needs a pre-defined topological structure. However, in each configuration during parsing, just partial dependency relations have been constructed, while the remains are still unknown. Besides, the standard RNN can just deal with the binary tree. Therefore we cannot apply the standard RNN directly.

Here, we propose two heterogeneous recursive neural networks: tree structured gated recursive neural network (Tree-GRNN) and directed acyclic graph structured gated recursive neural network

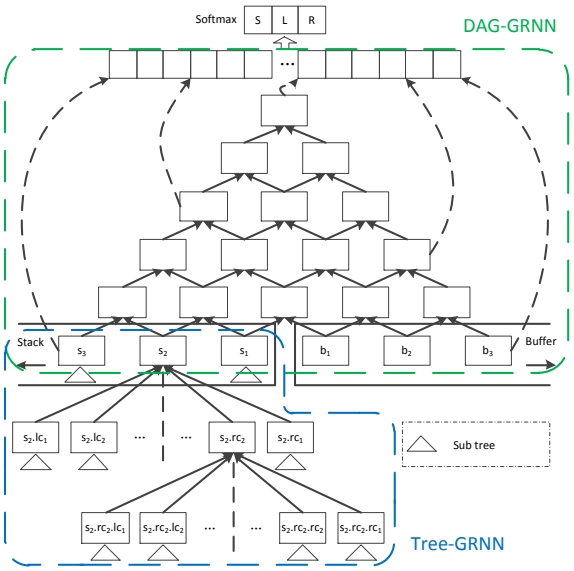

Figure 4: Architecture of our proposed dependency parser using two heterogeneous gated recursive neural networks.

(DAG-GRNN). Tree-GRNN is applied to the subtrees with partial dependency relations in stack which have already been constructed according to the previous transition actions. DAG-GRNN is employed to model the feature composition of nodes in stack and buffer which have not been labeled their dependency relations yet.

Figure 4 shows the whole architecture of our model, which integrates two different GRNNs to predict the action for each parsing configuration. The detailed descriptions of two GRNNs will be discussed in the following two subsections.

4.1 Tree Structured Gated Recursive Neural Network

It is a natural way to merge the information from children nodes into their parent node recursively according to the given tree structures in stack. Although the dependency relations have been built, it is still hard to apply the recursive neural network (as Eq. 3) directly for the uncertain number of children of each node in stack. By averaging operation on children nodes (Socher et al., 2014), the parent node cannot well capture the crucial features from the mixed information of its children nodes. Here, we propose tree structured gated recursive neural network (Tree-GRNN) incorporating the gate mechanism (Cho et al., 2014; Chung et al., 2014; Chen et al., 2015a; Chen et al., 2015b), which can selectively choose the

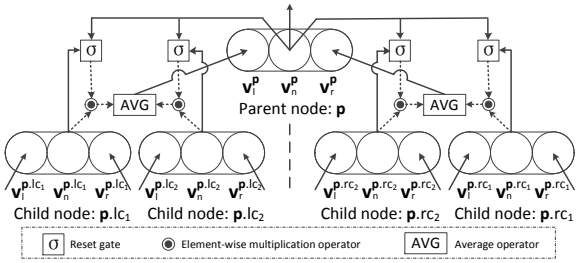

Figure 5: Minimal structure of tree structured gated recursive neural network (Tree-GRNN). The solid arrow denotes that there is a weight matrix on the link, while the dashed one denotes none.

crucial features according to the gate state. Figure 5 shows the minimal structure of Tree-GRNN model.

In Tree-GRNN, each node $\mathbf{p}$ of trees in stack is composed of three components: state vector of left children nodes $\mathbf{v}_l^{\mathbf{p}} \in \mathbb{R}^{d_c}$, state vector of current node $\mathbf{v}_n^{\mathbf{p}} \in \mathbb{R}^{d_n}$ and state vector of right children nodes $\mathbf{v}_r^{\mathbf{p}} \in \mathbb{R}^{d_c}$, where d_n and d_c indicate the corresponding vector dimensionalities. Particularly, we represent information of node $\mathbf{p}$ as a vector

$$\mathbf{v}^{\mathbf{p}} = \begin{bmatrix} \mathbf{v}_l^{\mathbf{p}} \\ \mathbf{v}_n^{\mathbf{p}} \\ \mathbf{v}_r^{\mathbf{p}} \end{bmatrix}, \tag{4}$$

where $\mathbf{v}^{\mathbf{p}} \in \mathbb{R}^q$ and $q = 2d_c + d_n$. Specifically, $\mathbf{v}_n^{\mathbf{p}}$ contains the information of current node including its word form $\mathbf{p}.w$, pos tag $\mathbf{p}.t$ and label type $\mathbf{p}.l$ as shown in Eq. 5, and $\mathbf{v}_l^{\mathbf{p}}$ and $\mathbf{v}_r^{\mathbf{p}}$ are initialized by zero vectors $\mathbf{0} \in \mathbb{R}^{d_c}$, then update as Eq. 6.

$$\mathbf{v}_n^{\mathbf{p}} = \tanh\left(\begin{bmatrix} \mathbf{e}^w_{\mathbf{p}.w} \\ \mathbf{e}^t_{\mathbf{p}.t} \\ \mathbf{e}^l_{\mathbf{p}.l} \end{bmatrix}\right), \tag{5}$$

where word embedding $\mathbf{e}^w_{\mathbf{p}.w} \in \mathbb{R}^{d_e}$, pos embedding $\mathbf{e}^t_{\mathbf{p}.t} \in \mathbb{R}^{d_e}$ and label embedding $\mathbf{e}^l_{\mathbf{p}.l} \in \mathbb{R}^{d_e}$ are extracted from embedding matrices $\mathbf{E}^w$, $\mathbf{E}^t$ and $\mathbf{E}^l$ according to the indices of the corresponding word $\mathbf{p}.w$, pos $\mathbf{p}.t$ and label $\mathbf{p}.l$ respectively. Specifically, in the case of unlabeled attachment parsing, we ignore the last term $\mathbf{e}^l_{\mathbf{p}.l}$ in Eq. 5. Thus, the dimensionality d_n of $\mathbf{v}_n^{\mathbf{p}}$ varies. In labeled attachment parsing case, we set a special token NULL to represent label $\mathbf{p}.l$ if not available (e.g. $\mathbf{p}$ is the node in stack or buffer).

By given node $\mathbf{p}$ and its left children nodes $\mathbf{p}.lc_i$ and right children nodes $\mathbf{p}.rc_i$, we update the left

children information $\mathbf{v}_l^{\mathbf{p}}$ and right children information $\mathbf{v}_r^{\mathbf{p}}$ as

$$\begin{aligned}\mathbf{v}_l^{\mathbf{p}} &= \tanh(\mathbf{W}_l \frac{1}{N_L(\mathbf{p})} \sum_i \mathbf{o}^{\mathbf{p}.lc_i} \odot \mathbf{v}^{\mathbf{p}.lc_i} + \mathbf{b}_l), \\ \mathbf{v}_r^{\mathbf{p}} &= \tanh(\mathbf{W}_r \frac{1}{N_R(\mathbf{p})} \sum_i \mathbf{o}^{\mathbf{p}.rc_i} \odot \mathbf{v}^{\mathbf{p}.rc_i} + \mathbf{b}_r),\end{aligned} \tag{6}$$

where $\mathbf{o}^{\mathbf{p}.lc_i}$ and $\mathbf{o}^{\mathbf{p}.rc_i}$ are the reset gates of the nodes $\mathbf{p}.lc_i$ and $\mathbf{p}.rc_i$ respectively as shown in Eq. 7. In addition, functions $N_L(\mathbf{p})$ and $N_R(\mathbf{p})$ result the numbers of left and right children nodes of node $\mathbf{p}$ respectively. The operator $\odot$ indicates element multiplication here. $\mathbf{W}_l \in \mathbb{R}^{d_c \times q}$ and $\mathbf{W}_r \in \mathbb{R}^{d_c \times q}$ are weight matrices. $\mathbf{b}_l \in \mathbb{R}^{d_c}$ and $\mathbf{b}_r \in \mathbb{R}^{d_c}$ are bias terms.

The reset gates $\mathbf{o}^{\mathbf{p}.lc_i}$ and $\mathbf{o}^{\mathbf{p}.rc_i}$ can be formalized as

$$\begin{aligned}\mathbf{o}^{\mathbf{p}.lc_i} &= \sigma(\mathbf{W}_o \begin{bmatrix} \mathbf{v}^{\mathbf{p}.lc_i} \\ \mathbf{v}_n^{\mathbf{p}} \end{bmatrix} + \mathbf{b}_o), \\ \mathbf{o}^{\mathbf{p}.rc_i} &= \sigma(\mathbf{W}_o \begin{bmatrix} \mathbf{v}^{\mathbf{p}.rc_i} \\ \mathbf{v}_n^{\mathbf{p}} \end{bmatrix} + \mathbf{b}_o),\end{aligned} \tag{7}$$

where σ indicates the sigmoid function, $\mathbf{W}_o \in \mathbb{R}^{q \times (q+d_n)}$ and $\mathbf{b}_o \in \mathbb{R}^q$.

By the mechanism above, we can summarize the whole information into the stack recursively from children nodes to their parent using the partial-built tree structure. Intuitively, the gate mechanism can selectively choose the crucial features of a child node according to the gate state which is derived from the current child node and its parent.

4.2 Directed Acyclic Graph Structured Gated Recursive Neural Network

Previous neural based parsing works feed the extracted features into a standard neural network with one hidden layer. Then, the hidden units are fed into a softmax layer, outputting the probability vector of available actions. Actually, it cannot well model the complicated combinations of extracted features. As for the nodes, whose dependency relations are still unknown, we propose another recursive neural network namely directed acyclic graph structured gated recursive neural network (DAG-GRNN) to better model the interactions of features.

Intuitively, the DAG structure models the combinations of features by recursively mixing the information from the bottom layer to the top layer as shown in Figure 4. The concatenation operation can be regraded as a mix of features in different levels of granularity. Each node in the directed acyclic graph can be seen as a complicated feature composition of its governed nodes.

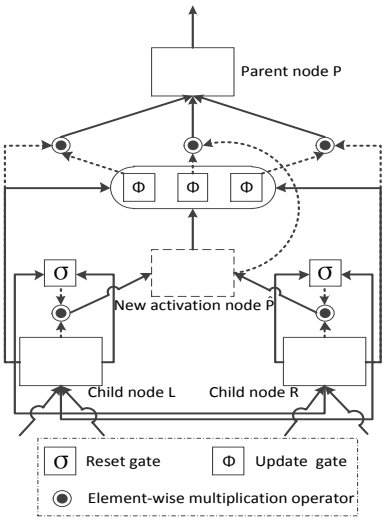

Figure 6: Minimal structure of directed acyclic graph structured gated recursive neural network (DAG-GRNN). The solid arrow denotes that there is a weight matrix on the link, while the dashed one denotes none.

Moreover, we also use the gate mechanism to better model the feature combinations by introducing two kinds of gates, namely "reset gate" and "update gate". Intuitively, each node in the network seems to preserve all the information of its governed notes without gates, and the gate mechanism similarly plays a role of filter which decides how to selectively exploit the information of its children nodes, discovering and preserving the crucial features.

DAG-GRNN structure consists of minimal structures as shown in Figure 6. Vectors $\mathbf{h}^P$, $\mathbf{h}^L$, $\mathbf{h}^R$ and $\mathbf{h}^{\hat{P}} \in \mathbb{R}^q$ denote the value of the parent node P, left child node L, right child node R and new activation node $\hat{P}$ respectively. The value of parent node $\mathbf{h}^P \in \mathbb{R}^q$ is computed as:

$$\mathbf{h}^P = \mathbf{z}_{\hat{P}} \odot \mathbf{h}^{\hat{P}} + \mathbf{z}_L \odot \mathbf{h}^L + \mathbf{z}_R \odot \mathbf{h}^R, \tag{8}$$

where $\mathbf{z}_{\hat{P}}$, $\mathbf{z}_L$ and $\mathbf{z}_R \in \mathbb{R}^q$ are update gates for new activation node $\hat{P}$, left child node L and right child node R respectively. Operator $\odot$ indicates element-wise multiplication.

The update gates $\mathbf{z}$ can be formalized as:

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_{\hat{P}} \\ \mathbf{z}_L \\ \mathbf{z}_R \end{bmatrix} \propto \exp(\mathbf{W}_z \begin{bmatrix} \mathbf{h}^{\hat{P}} \\ \mathbf{h}^L \\ \mathbf{h}^R \end{bmatrix}), \quad (9)$$

which are constrained by:

$$\begin{cases} [\mathbf{z}_{\hat{P}}]_k + [\mathbf{z}_L]_k + [\mathbf{z}_R]_k = 1, & 1 \le k \le q, \\ [\mathbf{z}_{\hat{P}}]_k \ge 0, & 1 \le k \le q, \\ [\mathbf{z}_L]_k \ge 0, & 1 \le k \le q, \\ [\mathbf{z}_R]_k \ge 0, & 1 \le k \le q, \end{cases} \quad (10)$$

where $\mathbf{W}_z \in \mathbb{R}^{3q \times 3q}$ is the coefficient of update gates.

The value of new activation node $\mathbf{h}_{\hat{P}}$ is computed as:

$$\mathbf{h}_{\hat{P}} = \tanh(\mathbf{W}_{\hat{P}} \begin{bmatrix} \mathbf{r}_L \odot \mathbf{h}_L \\ \mathbf{r}_R \odot \mathbf{h}_R \end{bmatrix}), \quad (11)$$

where $\mathbf{W}_{\hat{P}} \in \mathbb{R}^{q \times 2q}$, $\mathbf{r}_L \in \mathbb{R}^q$, $\mathbf{r}_R \in \mathbb{R}^q$. $\mathbf{r}_L$ and $\mathbf{r}_R$ are the reset gates for left child node L and right child node R respectively, which can be formalized as:

$$\mathbf{r} = \begin{bmatrix} \mathbf{r}_L \\ \mathbf{r}_R \end{bmatrix} = \sigma(\mathbf{W}_r \begin{bmatrix} \mathbf{h}_L \\ \mathbf{h}_R \end{bmatrix}), \quad (12)$$

where $\mathbf{W}_r \in \mathbb{R}^{2q \times 2q}$ is the coefficient of two reset gates and σ indicates the sigmoid function.

Intuitively, the reset gates $\mathbf{r}$ partially read the information from the left and right children, outputting a new activation node $\mathbf{h}_{\hat{P}}$, while the update gates $\mathbf{z}$ selectively choosing the information among the the new activation node $\hat{P}$, the left child node L and the right child node R. This gate mechanism is effective to model the combinations of features.

Finally, we concatenate all the nodes in the DAG-GRNN structure as input $\mathbf{x}$ of the architecture described in Section 2.2, resulting the probability vector for all available actions.

4.3 Inference

We use greedy decoding in parsing. At each step, we apply our two GRNNs on the current configuration to extract the features. After softmax operation, we choose the feasible transition with the highest possibility, and perform the chosen transition on the current configuration to get the next configuration state.

In practice, we do not need calculate the Tree-GRNN over the all trees in the stack on the current configuration. Instead, we preserve the representations of trees in the stack. When we need apply a new transition on the configuration, we update the relative representations using Tree-GRNN.

5 Training

We use the maximum likelihood (ML) criterion to train our model. By extracting training set (x_i, y_i) from gold parse trees using a shortest stack oracle which always prefers Left-Arc(l) or Right-Arc(l) action over Shift, the goal of our model is to minimize the loss function with the parameter set θ:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \log \mathbf{p}(y_i|x_i;\theta) + \frac{\lambda}{2m}\|\theta\|_2^2, \quad (13)$$

where m is number of extracted training examples which is as same as the number of all configurations.

Following Socher et al. (2013), we use the diagonal variant of AdaGrad (Duchi et al., 2011) with minibatch strategy to minimize the objective. We also employ dropout strategy to avoid overfitting.

In practice, we perform DAG-GRNN with two hidden layers, which gets the best performance. We use the approximated gradient for Tree-GRNN, which only performs gradient back propagation on the first two layers.

6 Experiments

6.1 Datasets

To evaluate our proposed model, we experiment on two prevalent datasets: English Penn Treebank 3 (PTB3) and Chinese Penn Treebank 5 (CTB5) datasets.

- **English** For English Penn Treebank 3 (PTB3) dataset, we use sections 2-21 for training, section 22 and section 23 as development set and test set respectively. We adopt CoNLL Syntactic Dependencies (CD) (Johansson and Nugues, 2007) using the LTH Constituent-to-Dependency Conversion Tool.
- **Chinese** For Chinese Penn Treebank 5 (CTB5) dataset, we follow the same split as described in (Zhang and Clark, 2008). Dependencies are converted by the Penn2Malt tool with the head-finding rules of (Zhang and Clark, 2008).

Embedding size	$d_e = 50$
Dimensionality of child node vector	$d_c = 50$
Initial learning rate	$\alpha = 0.05$
Regularization	$\lambda = 10^{-8}$
Dropout rate	$p = 20\%$

Table 1: Hyper-parameter settings.

6.2 Experimental Settings

For parameter initialization, we use random initialization within (-0.01, 0.01) for all parameters except the word embedding matrix $\mathbf{E}^w$. Specifically, we adopt pre-trained English word embeddings from (Collobert et al., 2011). And we pre-train the Chinese word embeddings on a huge unlabeled data, the Chinese Wikipedia corpus, with word2vec toolkit (Mikolov et al., 2013a).

Table 1 gives the details of hyper-parameter settings of our approach. In addition, we set mini-batch size to 20. In all experiments, we only take s_1, s_2, s_3 nodes in stack and b_1, b_2, b_3 nodes in buffer into account. We also apply dropout strategy here, and only dropout at the nodes in stack and buffer with probability $p = 20\%$.

6.3 Results

The experiment results on PTB3 and CTB5 datasets are list in Table 2 and 3 respectively. On all datasets, we report unlabeled attachment scores (UAS) and labeled attachment scores (LAS). Conventionally, punctuations are excluded in all evaluation metrics.

To evaluate the effectiveness of our approach, we compare our parsers with feature-based parser and neural-based parser. For feature-based parser, we compare our models with two prevalent parsers: MaltParser (Nivre et al., 2006) and MSTParser (McDonald and Pereira, 2006). For neural-based parser, we compare our results with parser of Chen and Manning (2014). Compared with parser of Chen and Manning (2014), our parser with two heterogeneous gated recursive neural networks (Tree-GRNN+DAG-GRNN) receives 0.6% (UAS) and 0.9% (LAS) improvement on PTB3 test set, and receives 0.8% (UAS) and 1.3% (LAS) improvement on CTB5 test set.

Since that speed of algorithm is not the focus of our paper, we do not optimize the speed a lot. On CTB (UAS), it takes about 2 days to train Tree-GRNN+DAG-GRNN model with CPU only. The testing speed is about 2.7 sentences per second. All implementation is based on Python.

6.4 Effects of Gate Mechanisms

We adopt five different models: plain parser, Tree-RNN parser, Tree-GRNN parser, Tree-RNN+DAG-GRNN parser, and Tree-GRNN+DAG-GRNN parser. The experiment results show the effectiveness of our proposed two heterogeneous gated recursive neural networks.

Specifically, plain parser is as same as parser of Chen and Manning (2014). The difference between them is that plain parser only takes the nodes in stack and buffer into account, which uses a simpler feature template than parser of Chen and Manning (2014). As plain parser omits all children nodes of trees in stack, it performs poorly compared with parser of Chen and Manning (2014). In addition, we find plain parser outperforms MaltParser (standard) on PTB3 dataset making about 1% progress, while it performs poorer than MaltParser (standard) on CTB5 dataset. It shows that the children nodes of trees in stack is of great importance, especially for Chinese. Moreover, it also shows the effectiveness of neural network based model which could represent complicated features as compacted embeddings. Tree-RNN parser additionally exploits all the children nodes of trees in stack, which is a simplified version of Tree-GRNN without incorporating the gate mechanism described in Section 4.1. In anther word, Tree-RNN omits the gate terms $\mathbf{o}^{\mathbf{p}.lc_i}$ and $\mathbf{o}^{\mathbf{p}.rc_i}$ in Eq. 6. As we can see, the results are significantly boosted by utilizing the all information in stack, which again shows the importance of children nodes of trees in stack. Although the results of Tree-RNN are comparable to results of Chen and Manning (2014), it not outperforms parser of Chen and Manning (2014) in all cases (e.g. UAS on CTB5), which implies that exploiting all information without selection might lead to incorporate noise features. Moreover, Tree-GRNN parser further boosts the performance by incorporating the gate mechanism. Intuitively, Tree-RNN who exploits all the information of stack without selection cannot well capture the crucial features, while Tree-GRNN with gate mechanism could selectively choose and preserve the effective features by adapting the current gate state.

We also experiment on parsers using two heterogeneous gated recursive neural networks: Tree-RNN+DAG-GRNN parser and Tree-GRNN+DAG-GRNN parser. The similarity of

models	Dev		Test	
	UAS	LAS	UAS	LAS
Malt:standard	90.0	88.8	89.9	88.5
Malt:eager	90.1	88.9	90.1	88.7
MSTParser	92.1	90.8	92.0	90.5
Chen's Parser	92.2	91.0	92.0	90.7
Plain	91.1	90.0	91.2	89.7
Tree-RNN	92.4	91.0	92.1	90.8
Tree-GRNN	92.6	91.1	92.4	91.0
Tree-RNN+DAG-GRNN	**92.8**	**91.9**	92.4	91.5
Tree-GRNN+DAG-GRNN	92.6	**91.9**	**92.6**	**91.6**

Table 2: Performance of different models on PTB3 dataset. UAS: unlabeled attachment score. LAS: labeled attachment score.

models	Dev		Test	
	UAS	LAS	UAS	LAS
Malt:standard	82.4	80.5	82.4	80.6
Malt:eager	91.2	79.3	80.2	78.4
MSTParser	84.0	82.1	83.0	81.2
Chen's Parser	84.0	82.4	83.9	82.4
Plain	81.6	79.3	81.1	78.8
Tree-RNN	83.5	82.5	83.8	82.7
Tree-GRNN	84.2	82.5	84.3	83.1
Tree-RNN+DAG-GRNN	84.5	83.3	84.5	83.1
Tree-GRNN+DAG-GRNN	**84.6**	**83.6**	**84.7**	**83.7**

Table 3: Performance of different models on CTB5 dataset. UAS: unlabeled attachment score. LAS: labeled attachment score.

two parsers is that they all employ the DAG structured recursive neural network with gate mechanism to model the combination of features extracted from stack and buffer. The difference between them is the former one employs the Tree-RNN without gate mechanism to model the features of stack, while the later one employs the gated version (Tree-GRNN). Again, the performance of these two parsers is further boosted, which shows DAG-GRNN can well model the combinations of features which is summarized by Tree-(G)RNN structure. In addition, we find the performance does not drop a lot in almost cases by turning off the gate mechanism of Tree-GRNN, which implies that the DAG-GRNN can help selecting the information from trees in stack, even it has not been selected by gate mechanism of Tree-GRNN yet.

6.5 Convergency Speed

To further analyze the convergency speed of our approach, we compare the UAS results on development sets of two datasets for first ten epoches as shown in Figure 7 and 8. As plain parser only take the nodes in stack and buffer into account, the performance is much poorer than the rest parsers. Moreover, Tree-GRNN converges slower than Tree-RNN, which shows that it might be more difficult to learn this gate mechanism. By introducing the DAG-GRNN, both Tree-RNN and Tree-GRNN parsers become faster to converge, which shows that the DAG-GRNN is of great help in boosting the convergency speed.

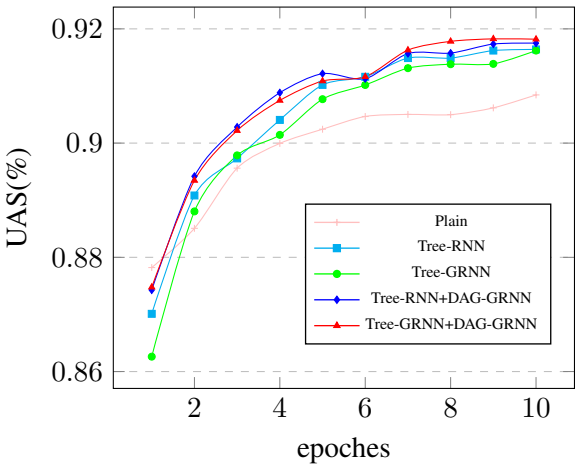

Figure 7: Performance of different models on PTB3 development set. UAS: unlabeled attachment score.

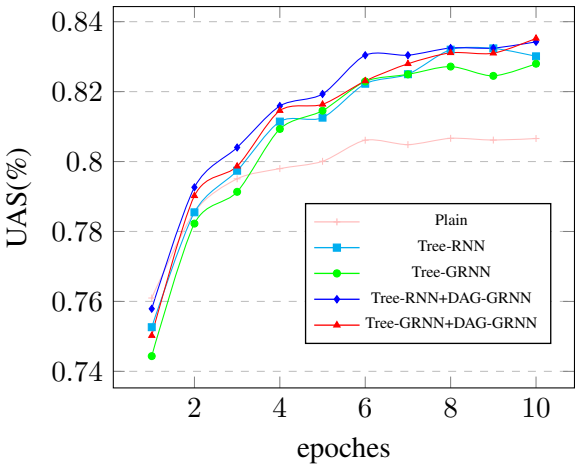

Figure 8: Performance of different models on CTB5 development set. UAS: unlabeled attachment score.

7 Related Work

Many neural network based methods have been used for transition based dependency parsing.

Chen et al. (2014) and Bansal et al. (2014) used the dense vectors (embeddings) to represent words or features and found these representations are

complementary to the traditional discrete feature representation. However, these two methods only focus on the dense representations (embeddings) of words or features.

Stenetorp (2013) first used RNN for transition based dependency parsing. He followed the standard RNN and used the binary combination to model the representation of two linked words. But his model does not achieve the performance of the traditional method.

Le and Zuidema (2014) proposed a generative re-ranking model with Inside-Outside Recursive Neural Network (IORNN), which can process trees both bottom-up and top-down. However, IORNN works in generative way and just estimates the probability of a given tree, so IORNN cannot fully utilize the incorrect trees in k-best candidate results. Besides, IORNN treats dependency tree as a sequence, which can be regarded as a generalization of simple recurrent neural network (SRNN) (Elman, 1990).

Although the two methods also used RNN, they just deal with the binary combination, which is unnatural for dependency tree.

Zhu et al. (2015) proposed a recursive convolutional neural network (RCNN) architecture to capture syntactic and compositional-semantic representations of phrases and words in a dependency tree. Different with the original recursive neural network, they introduced the convolution and pooling layers, which can model a variety of compositions by the feature maps and choose the most informative compositions by the pooling layers.

Chen and Manning (2014) improved the transition-based dependency parsing by representing all words, POS tags and arc labels as dense vectors, and modeled their interactions with neural network to make predictions of actions. Their method only relies on dense features, and is not able to automatically learn the most useful feature conjunctions to predict the transition action.

Compared with (Chen and Manning, 2014), our method can fully exploit the information of all the descendants of a node in stack with Tree-GRNN. Then DAG-GRNN automatically learns the complicated combination of all the features, while the traditional discrete feature based methods need manually design them.

Dyer et al. (2015) improved the transition-based dependency parsing using stack long short term memory neural network and received significant improvement on performance. They focused on exploiting the long distance dependencies and information, while we aims to automatically model the complicated feature combination.

8 Conclusion

In this paper, we pay attention to the syntactic and semantic composition of the dense features for transition-based dependency parsing. We propose two heterogeneous gated recursive neural networks, Tree-GRNN and DAG-GRNN. Each hidden neuron in two proposed GRNNs can be regarded as a different combination of the input features. Thus, the whole model has an ability to simulate the design of the sophisticated feature combinations in the traditional discrete feature based methods.

Although the two proposed GRNNs are only used for the greedy parsing based on arc-standard transition system in this paper, it is easy to generalize them to other transition systems and graph based parsing. In future work, we would also like to extend our GRNNs for the other NLP tasks.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work was partially funded by the National Natural Science Foundation of China (61472088, 61473092), National High Technology Research and Development Program of China (2015AA015408), Shanghai Science and Technology Development Funds (14ZR1403200).

References

Miguel Ballesteros and Bernd Bohnet. 2014. Automatic feature selection for agenda-based dependency parsing. In *Proceedings of the 25th International Conference on Computational Linguistics*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Com-*

putational Linguistics, pages 89–97. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 816–826, Dublin, Ireland, August.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuanjing Huang. 2015a. Gated recursive neural network for Chinese word segmentation. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015b. Sentence modeling with gated recursive neural network. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of EMNLP*.

Jinho D Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *ACL (1)*, pages 1052–1062.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, June*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*, 1:403–414.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *EMNLP*, pages 1455–1464.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *16th Nordic Conference of Computational Linguistics*, pages 105–112. University of Tartu.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *ACL*.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739, Doha, Qatar, October. Association for Computational Linguistics.

André FT Martins, Noah A Smith, and Eric P Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 342–350. Association for Computational Linguistics.

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 91–98.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Jordan B Pollack. 1990. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. 2014. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218.

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

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. 2015. A re-ranking model for dependency parser with recursive convolutional neural network. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*.