A Re-ranking Model for Dependency Parser with Recursive Convolutional Neural Network

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

Abstract

In this work, we address the problem to model all the nodes (words or phrases) in a dependency tree with the dense representations. We propose 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, we introduce 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. Based on RCNN, we use a discriminative model to re-rank a k-best list of candidate dependency parsing trees. The experiments show that RCNN is very effective to improve the state-of-the-art dependency parsing on both English and Chinese datasets.

1 Introduction

Feature-based discriminative supervised models have achieved much progress in dependency parsing (Nivre, 2004; Yamada and Matsumoto, 2003; McDonald et al., 2005), which typically use millions of discrete binary features generated from a limited size training data. However, the ability of these models is restricted by the design of features. The number of features could be so large that the result models are too complicated for practical use and prone to overfit on training corpus due to data sparseness.

Recently, many methods are proposed to learn various distributed representations on both syntax and semantics levels. These distributed representations have been extensively applied on many natural language processing (NLP) tasks, such as syntax (Turian et al., 2010; Mikolov et al., 2010; Collobert et al., 2011; Chen and Manning, 2014) and semantics (Huang et al., 2012; Mikolov et al., 2013). Distributed representations are to represent words (or phrase) by the dense, low-dimensional and real-valued vectors, which help address the curse of dimensionality and have better generalization than discrete representations.

[*]Corresponding author.

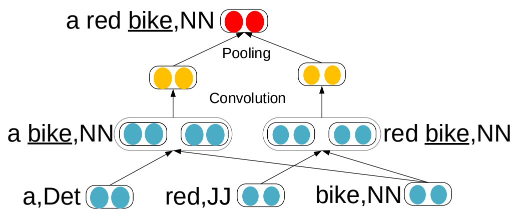


Figure 1: Illustration of a RCNN unit.

For 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. These embeddings are pre-trained and keep unchanged in the training phase of parsing model, which cannot be optimized for the specific tasks.

Besides, it is also important to represent the (unseen) phrases with dense vector in dependency parsing. Since the dependency tree is also in recursive structure, it is intuitive to use the recursive neural network (RNN), which is used for constituent parsing (Socher et al., 2013a). However, recursive neural network can only process the binary combination and is not suitable for dependency parsing, since a parent node may have two or more child nodes in dependency tree.

In this work, we address the problem to represent all level nodes (words or phrases) with dense representations in a dependency tree. We propose a recursive convolutional neural network (RCNN) architecture to capture syntactic and compositional-semantic representations of phrases and words. RCNN is a general architecture and can deal with k-ary parsing tree, therefore it is very suitable for dependency parsing. For each node in a given dependency tree, we first use a RCNN unit to model the interactions between it and each of its children and choose the most informative features by a pooling layer. Thus, we can apply the RCNN unit recursively to get the vector representation of the whole dependency tree. The output of each RCNN unit is used as the input of the RCNN unit of its parent node, until it outputs a single fixed-length vector at root node. Figure 1 illustrates an example how a RCNN unit represents the phrases “a red bike” as continuous vectors.

The contributions of this paper can be summarized as follows.

- RCNN is a general architecture to model the distributed representations of a phrase or sentence with its dependency tree. Although RCNN is just used for the re-ranking of the dependency parser in this paper, it can be regarded as semantic modelling of text sequences and handle the input sequences of varying length into a fixed-length vector. The parameters in RCNN can be learned jointly with some other NLP tasks, such as text classification.

- Each RCNN unit can model the complicated interactions of the head word and its children. Combined with a specific task, RCNN can capture the most useful semantic and structure information by the convolution and pooling layers.

- When applied to the re-ranking model for parsing, RCNN improve the accuracy of base parser to make accurate parsing decisions. The experiments on two benchmark datasets show that RCNN outperforms the state-of-the-art models.

2 Recursive Neural Network

In this section, we briefly describe the recursive neural network architecture of (Socher et al., 2013a).

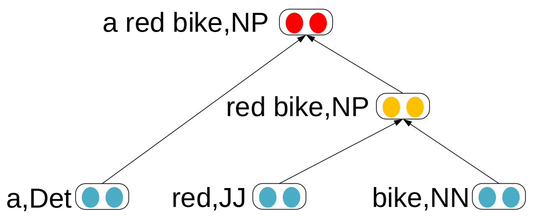


Figure 2: Illustration of a RNN unit.

The idea of recursive neural networks (RNN) for natural language processing (NLP) is to train a deep learning model that can be applied to phrases and sentences, which have a grammatical structure (Pollack, 1990; Socher et al., 2013c). RNN can be also regarded as a general structure to model sentence. At every node in the tree, the contexts at the left and right children of the node are combined by a classical layer. The weights of the layer are shared across all nodes in the tree. The layer computed at the top node gives a representation for the whole sentence.

Following the binary tree structure, RNN can assign a fixed-length vector to each word at the leaves of the tree, and combine word and phrase pairs recursively to create intermediate node vectors of the same length, eventually having one final vector representing the whole sentence. Multiple recursive combination functions have been explored, from linear transformation matrices to tensor products (Socher et al., 2013c). Figure 2 illustrates the architecture of RNN.

The binary tree can be represented in the form of branching triplets ($p \rightarrow c_1 c_2$). Each such triplet denotes that a parent node p has two children and each c_k can be either a word or a non-terminal node in the tree.

Given a labeled binary parse tree, $((p_2 \rightarrow a p_1), (p_1 \rightarrow bc))$, the node representations are computed by

$$\mathbf{p}_1 = f(\mathbf{W}\begin{bmatrix}\mathbf{b}\\ \mathbf{c}\end{bmatrix}), \mathbf{p}_2 = f(\mathbf{W}\begin{bmatrix}\mathbf{a}\\ \mathbf{p}_1\end{bmatrix}), \quad (1)$$

where $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{a}, \mathbf{b}, \mathbf{c})$ are the vector representation of (p_1, p_2, a, b, c) respectively, which are denoted by lowercase bold font letters; $\mathbf{W}$ is a matrix of parameters of the RNN.

Based on RNN, Socher et al. (2013a) introduced a compositional vector grammar, which uses the syntactically untied weights $\mathbf{W}$ to learn the syntactic-semantic, compositional vector representations. In order to compute the score of how plausible of a syntactic constituent a parent is, RNN uses a single-unit linear layer for all p_i:

$$s(p_i) = \mathbf{v} \cdot \mathbf{p}_i, \qquad (2)$$

where $\mathbf{v}$ is a vector of parameters that need to be trained. This score will be used to find the highest scoring tree. For more details on how standard RNN can be used for parsing, see (Socher et al., 2011).

Costa et al. (2003) applied recursive neural networks to re-rank possible phrase attachments in an incremental constituency parser. Their work is the first to show that RNNs can capture enough information to make the correct parsing decisions. Menchetti et al. (2005) used RNNs to re-rank different constituency parses. For their results on full sentence parsing, they re-ranked candidate trees created by the Collins parser (Collins, 2003).

3 Recursive Convolutional Neural Network

The dependency grammar is a widely used syntactic structure, which directly reflects relationships among the words in a sentence. In a dependency tree, all nodes are terminal (words) and each node may have more than two children. Therefore, the standard RNN architecture is not suitable for dependency grammar since it is based on the binary tree. In this section, we propose a more general architecture, called **recursive convolutional neural network** (RCNN), which borrows the idea of convolutional neural network (CNN) and can deal with to k-ary tree.

3.1 RCNN Unit

For ease of exposition, we first describe the basic unit of RCNN. A RCNN unit is to model a head word and its children. Different from the constituent tree, the dependency tree does not have non-terminal nodes. Each node consists of a word and its POS tags. Each node should have a different interaction with its head node.

Word Embeddings Given a word dictionary $\mathcal{W}$, each word $w \in \mathcal{W}$ is represented as a real-valued vector (word embedding) $\mathbf{w} \in \mathbb{R}^m$ where m is the dimensionality of the vector space. The word embeddings are then stacked into a embedding matrix $M \in \mathbb{R}^{m|\mathcal{W}|}$. For a word $w \in \mathcal{W}$, its corresponding word embedding $Embed(w) \in \mathbb{R}^m$ is retrieved by the lookup table layer. The matrix M

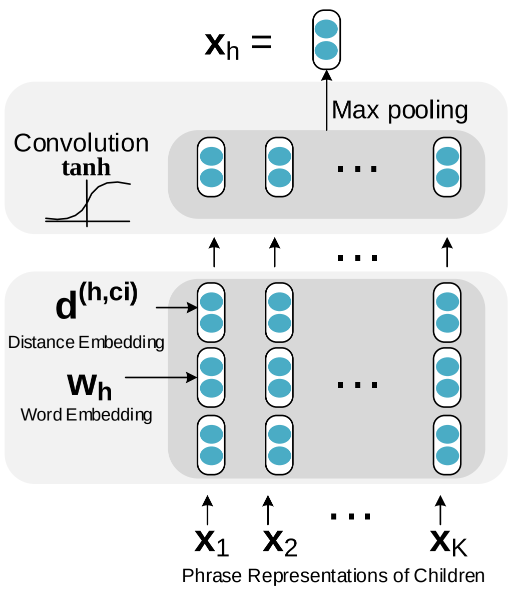


Figure 3: Architecture of a RCNN unit.

is initialized with pre-training embeddings and updated by back-propagation.

Distance Embeddings Besides word embeddings, we also use distributed vector to represent the relative distance of a head word h and one of its children c. For example, as shown in Figure 1, the relative distances of "bike" to "a" and "red" are -2 and -1, respectively. The relative distances also are mapped to a vector of dimension m_d (a hyperparameter); this vector is randomly initialized. Distance embedding is a usual way to encode the distance information in neural model, which has been proven effectively in several tasks. Our experimental results also show that the distance embedding gives more benefits than the traditional representation. The relative distance can encode the structure information of a subtree.

Convolution The word and distance embeddings are subsequently fed into the convolution component to model the interactions between two linked nodes.

Different with standard RNN, there are no non-terminal nodes in dependency tree. Each node h in dependency tree has two associated distributed representations:

1. word embedding $\mathbf{w}_h \in \mathbb{R}^m$, which is denoted as its own information according to its word form;

2. phrase representation $\mathbf{x}_h \in \mathbb{R}^m$, which is denoted as the joint representation of the whole subtree rooted at h. In particular, when h is leaf node, $\mathbf{x}_h = \mathbf{w}_h$.

Given a subtree rooted at h in dependency tree, we define $c_i, 0 < i \leq L$ as the i-th child node of h, where L represents the number of children.

For each pair (h, c_i), we use a convolutional hidden layer to compute their combination representation $\mathbf{z}_i$.

$$\mathbf{z}_i = \tanh(\mathbf{W}^{(h,c_i)}\mathbf{p}_i), 0 < i \leq K, \tag{3}$$

where $\mathbf{W}^{(h,c_i)} \in \mathbb{R}^{m \times n}$ is the linear **composition matrix**, which depends on the POS tags of h and c_i; $\mathbf{p}_i \in \mathbb{R}^n$ is the concatenated representation of h and the i-th child, which consists of the head word embeddings $\mathbf{w}_h$, the child phrase representation $\mathbf{x}_{c_i}$ and the distance embeddings $\mathbf{d}^{h,c_i}$ of h and c_i,

$$\mathbf{p}_i = \mathbf{x}_h \oplus \mathbf{x}_{c_i} \oplus \mathbf{d}^{(h,c_i)}, \tag{4}$$

where $\oplus$ represents the concatenation operation.

The distances d_{h,c_i} is the relative distance of h and c_i in a given sentence. Then, the relative distances also are mapped to m-dimensional vectors. Different from constituent tree, the combination should consider the order or position of each child in dependency tree.

In our model, we do not use the POS tags embeddings directly. Since the composition matrix varies on the different pair of POS tags of h and c_i, it can capture the different syntactic combinations. For example, the combination of adjective and noun should be different with that of verb and noun.

After the composition operations, we use **tanh** as the non-linear activation function to get a hidden representation $\mathbf{z}$.

Max Pooling After convolution, we get $\mathbf{Z}^{(h)} = [\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_K]$, where K is dynamic and depends on the number of children of h. To transform $\mathbf{Z}$ to a fixed length and determine the most useful semantic and structure information, we perform a max pooling operation to $\mathbf{Z}$ on rows.

$$\mathbf{x}_j^{(h)} = \max_i \mathbf{Z}_{j,i}^{(h)}, 0 < j \leq m. \tag{5}$$

Thus, we obtain the vector representation $\mathbf{x}_h \in \mathbb{R}^m$ of the whole subtree rooted at node h.

Figure 3 shows the architecture of our proposed RCNN unit.

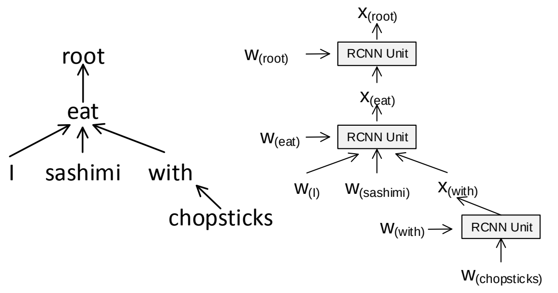


Figure 4: Example of a RCNN unit

Given a whole dependency tree, we can apply the RCNN unit recursively to get the vector representation of the whole sentence. The output of each RCNN unit is used as the input of the RCNN unit of its parent node.

Thus, RCNN can be used to model the distributed representations of a phrase or sentence with its dependency tree and applied to many NLP tasks. The parameters in RCNN can be learned jointly with the specific NLP tasks. Each RCNN unit can model the complicated interactions of the head word and its children. Combined with a specific task, RCNN can select the useful semantic and structure information by the convolution and max pooling layers.

Figure 4 shows an example of RCNN to model the sentence "*I eat sashimi with chopsitcks*".

4 Parsing

In order to measure the plausibility of a subtree rooted at h in dependency tree, we use a single-unit linear layer neural network to compute the score of its RCNN unit.

For constituent parsing, the representation of a non-terminal node only depends on its two children. The combination is relative simple and its correctness can be measured with the final representation of the non-terminal node (Socher et al., 2013a).

However for dependency parsing, all combinations of the head h and its children $c_i (0 < i \leq K)$ are important to measure the correctness of the subtree. Therefore, our score function $s(h)$ is computed on all of hidden layers $\mathbf{z}_i (0 < i \leq K)$:

$$s(h) = \sum_{i=1}^{K} \mathbf{v}^{(h,c_i)} \cdot \mathbf{z}_i, \tag{6}$$

where $\mathbf{v}^{(h,c_i)} \in \mathbb{R}^{m \times 1}$ is the **score vector**, which also depends on the POS tags of h and c_i.

Given a sentence x and its dependency tree y, the goodness of a complete tree is measured by summing the scores of all the RCNN units.

$$s(x, y, \Theta) = \sum_{h \in y} s(h), \tag{7}$$

where $h \in y$ is the node in tree y; $\Theta = \{\Theta_{\mathbf{W}}, \Theta_{\mathbf{v}}, \Theta_{\mathbf{w}}, \Theta_{\mathbf{d}}\}$ including the combination matrix set $\Theta_{\mathbf{W}}$, the score vector set $\Theta_{\mathbf{v}}$, the word embeddings $\Theta_{\mathbf{w}}$ and distance embeddings $\Theta_{\mathbf{d}}$.

Finally, we can predict dependency tree $\hat{y}$ with highest score for sentence x.

$$\hat{y} = \arg\max_{y \in \mathbf{gen}(x)} s(x, y, \Theta), \tag{8}$$

where $\mathbf{gen}(x)$ is defined as the set of all possible trees for sentence x. When applied in re-ranking, $\mathbf{gen}(x)$ is the set of the k-best outputs of a base parser.

5 Training

For a given training instance (x_i, y_i), we use the max-margin criterion to train our model. We first predict the dependency tree $\hat{y_i}$ with the highest score for each x_i and define a structured margin loss $\Delta(y_i, \hat{y_i})$ between the predicted tree $\hat{y_i}$ and the given correct tree y_i. $\Delta(y_i, \hat{y_i})$ is measured by counting the number of nodes y_i with an incorrect span (or label) in the proposed tree (Goodman, 1998).

$$\Delta(y_i, \hat{y_i}) = \sum_{d \in \hat{y_i}} \kappa \mathbf{1}\{d \notin y_i\} \tag{9}$$

where κ is a discount parameter and d represents the nodes in trees.

Given a set of training dependency parses $\mathcal{D}$, the final training objective is to minimize the loss function $J(\Theta)$, plus a l_2-regulation term:

$$J(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} r_i(\Theta) + \frac{\lambda}{2} \|\Theta\|_2^2, \tag{10}$$

where

$$r_i(\Theta) = \max_{\hat{y_i} \in Y(x_i)} \big(0, s_t(x_i, \hat{y_i}, \Theta) + \Delta(y_i, \hat{y_i}) - s_t(x_i, y_i, \Theta) \big). \tag{11}$$

By minimizing this object, the score of the correct tree y_i is increased and the score of the highest scoring incorrect tree $\hat{y_i}$ is decreased.

We use a generalization of gradient descent called subgradient method (Ratliff et al., 2007) which computes a gradient-like direction. The subgradient of equation is:

$$\frac{\partial J}{\partial \Theta} = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} \Big(\frac{\partial s_t(x_i, \hat{y_i}, \Theta)}{\partial \Theta} - \frac{\partial s_t(x_i, y_i, \Theta)}{\partial \Theta} \Big) + \lambda \Theta. \tag{12}$$

To minimize the objective, we use the diagonal variant of AdaGrad (Duchi et al., 2011). The parameter update for the i-th parameter $\Theta_{t,i}$ at time step t is as follows:

$$\Theta_{t,i} = \Theta_{t-1,i} - \frac{\rho}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}, \tag{13}$$

where ρ is the initial learning rate and $g_\tau \in \mathbb{R}^{|\theta_i|}$ is the subgradient at time step τ for parameter θ_i.

6 Re-rankers

Re-ranking k-best lists was introduced by Collins and Koo (2005) and Charniak and Johnson (2005). They used discriminative methods to re-rank the constituent parsing. In the dependency parsing, Sangati et al. (2009) used a third-order generative model for re-ranking k-best lists of base parser. Hayashi et al. (2013) used a discriminative forest re-ranking algorithm for dependency parsing. These re-ranking models achieved a substantial raise on the parsing performances.

Given $\mathcal{T}(x)$, the set of k-best trees of a sentence x from a base parser, we use the popular mixture re-ranking strategy (Hayashi et al., 2013; Le and Mikolov, 2014), which is a combination of the our model and the base parser.

$$\hat{y_i} = \arg\max_{y \in \mathcal{T}(x_i)} \alpha s_t(x_i, y, \Theta) + (1 - \alpha) s_b(x_i, y) \tag{14}$$

where $\alpha \in [0, 1]$ is a hyperparameter; $s_t(x_i, y, \Theta)$ and $s_b(x_i, y)$ are the scores given by RCNN and the base parser respectively.

To apply RCNN into re-ranking model, we first get the k-best outputs of all sentences in train set with a base parser. Thus, we can train the RCNN in a discriminative way and optimize the re-ranking strategy for a particular base parser.

Note that the role of RCNN is not fully valued when applied in re-ranking model since that the $\mathbf{gen}(x)$ in Eq.(8) is just the k-best outputs of a base parser, not the set of all possible trees for sentence x. The parameters of RCNN could overfit to k-best outputs of training set.

7 Experiments

7.1 Datasets

To empirically demonstrate the effectiveness of our approach, we use two datasets in different languages (English and Chinese) in our experimental evaluation and compare our model against the other state-of-the-art methods using the unlabeled attachment score (UAS) metric ignoring punctuation.

English For English dataset, we follow the standard splits of Penn Treebank (PTB), using sections 2-21 for training, section 22 as development set and section 23 as test set. We tag the development and test sets using an automatic POS tagger (at 97.2% accuracy), and tag the training set using four-way jackknifing similar to (Collins and Koo, 2005).

Chinese For Chinese dataset, we follow the same split of the Penn Chinese Treeban (CTB5) as described in (Zhang and Clark, 2008) and use sections 001-815, 1001-1136 as training set, sections 886-931, 1148- 1151 as development set, and sections 816-885, 1137-1147 as test set. Dependencies are converted by using the Penn2Malt tool with the head-finding rules of (Zhang and Clark, 2008). And following (Zhang and Clark, 2008) (Zhang and Nivre, 2011), we use gold segmentation and POS tags for the input.

We use the linear-time incremental parser (Huang and Sagae, 2010) as our base parser and calculate the 64-best parses at the top cell of the chart. Note that we optimize the training settings for base parser and the results are slightly improved on (Huang and Sagae, 2010). Then we use max-margin criterion to train RCNN. Finally, we use the mixture strategy to re-rank the top 64-best parses.

For initialization of parameters, we train word2vec embeddings (Mikolov et al., 2013) on Wikipedia corpus for English and Chinese respectively. For the combination matrices and score vectors, we use the random initialization within $(0.01, 0.01)$. The parameters which achieve the best unlabeled attachment score on the development set will be chosen for the final evaluation.

7.2 English Dataset

We first evaluate the performances of the RCNN and re-ranker (Eq. (14)) on the development set. Figure 5 shows UASs of different models with varying k. The base parser achieves 92.45%. When $k = 64$, the oracle best of base parser achieves 97.34%, while the oracle worst achieves 73.30% (-19.15%) . RCNN achieves the maximum improvement of 93.00%(+0.55%) when $k = 6$. When $k > 6$, the performance of RCNN declines with the increase of k but is still higher than baseline (92.45%). The reason behind this is that RCNN could require more negative samples to avoid overfitting when k is large. Since the negative samples are limited in the k-best outputs of a base parser, the learnt parameters could easily overfits to the training set.

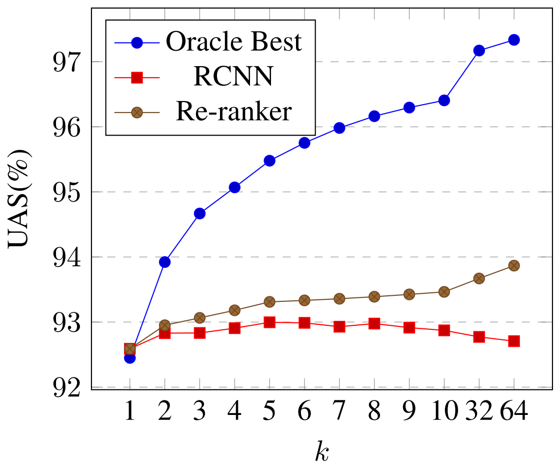
The mixture re-ranker achieves the maximum improvement of 93.50%(+1.05%) when $k = 64$. In mixture re-ranker, α is optimised by searching with the step-size 0.005.

Therefore, we use the mixture re-ranker in the following experiments since it can take the advantages of both the RCNN and base models.

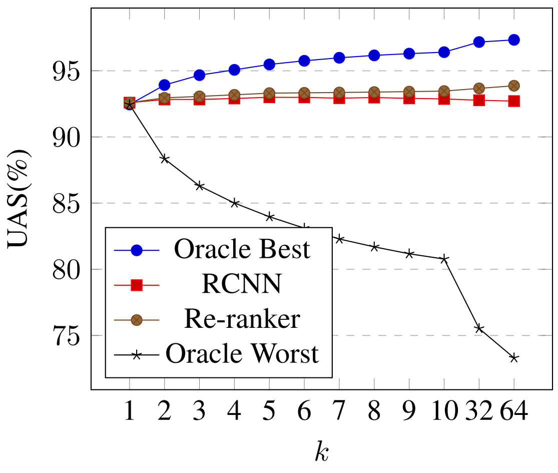
Figure 6 shows the accuracies on the top ten POS tags of the modifier words with the largest improvements. We can see that our re-ranker can improve the accuracies of *CC* and *IN*, and therefore may indirectly result in rising the the well-known coordinating conjunction and PP-attachment problems.

The final experimental results on test set are shown in Table 1. The hyperparameters of our model are set as in Table 2. Our re-ranker achieves the maximum improvement of 93.83%(+1.48%) on test set. Our system performs slightly better than many state-of-the-art systems such as Zhang and Clark (2008) and Huang and Sagae (2010). It outperforms Hayashi et al. (2013) and Le and Zuidema (2014), which also use the mixture re-ranking strategy.

Since the result of ranker is conditioned to k-best results of base parser, we also do an experiment to avoid this limitation by adding the oracle to k-best candidates. With including oracle, the re-ranker can achieve 94.16% on UAS, which is shown in the last line ("our re-ranker (with oracle)") of Table 1.



(a) without the oracle worst result



(b) with the oracle worst result

Figure 5: UAS with varying k on the development set. Oracle best: always choosing the best result in the k-best of base parser; Oracle worst: always choosing the worst result in the k-best of base parser; RCNN: choosing the most probable candidate according to the score of RCNN; Re-ranker: a combination of the RCNN and base parser.

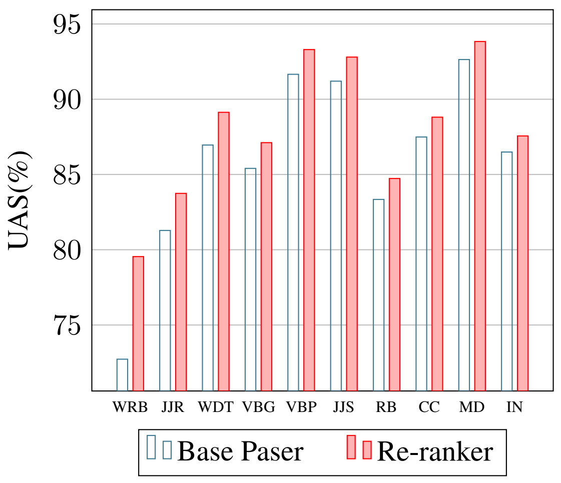


Figure 6: Accuracies on the top ten POS tags of the modifier words with the largest improvements on the development set.

	UAS
Traditional Methods	
Zhang and Clark (2008)	91.4
Huang and Sagae (2010)	92.1
Distributed Representations	
Stenetorp (2013)	86.25
Chen et al. (2014)	93.74
Chen and Manning (2014)	92.0
Re-rankers	
Hayashi et al. (2013)	93.12
Le and Zuidema (2014)	93.12
Our baseline	92.35
Our re-ranker	93.83(+1.48)
Our re-ranker (with oracle)	94.16

Table 1: Accuracy on English test set. Our baseline is the result of base parser; our re-ranker uses the mixture strategy on the 64-best outputs of base parser; our re-ranker(with oracle) is to add the oracle to k-best outputs of base parser.

7.3 Chinese Dataset

We also make experiments on the Penn Chinese Treebank (CTB5). The hyperparameters is the same as the previous experiment on English except that α is optimised by searching with the step-size 0.005.

The final experimental results on the test set are shown in Table 3. Our re-ranker achieves the performance of 85.71%(+0.25%) on the test set, which also outperforms the previous state-of-the-art methods. With adding oracle, the re-ranker can achieve 87.43% on UAS, which is shown in the last line ("our re-ranker (with oracle)") of Table 3.

Compared with the re-ranking model of Hayashi et al. (2013), that use a large number of handcrafted features, our model can achieve a competitive performance with the minimal feature engineering.

7.4 Discussions

The performance of the re-ranking model is affected by the base parser. The small divergence of the dependency trees in the output list also results to overfitting in training phase. Although our re-

Word embedding size	$m = 25$
Distance embedding size	$m_d = 25$
Initial learning rate	$\rho = 0.1$
Margin loss discount	$\kappa = 2.0$
Regularization	$\lambda = 10^{-4}$
k-best	$k = 64$

Table 2: Hyperparameters of our model

	UAS
Traditional Methods	
Zhang and Clark (2008)	84.33
Huang and Sagae (2010)	85.20
Distributed Representations	
Chen et al. (2014)	82.94
Chen and Manning (2014)	83.9
Re-rankers	
Hayashi et al. (2013)	85.9
Our baseline	85.46
Our re-ranker	85.71(+0.25)
Our re-ranker (with oracle)	87.43

Table 3: Accuracy on Chinese test set.

ranker outperforms the state-of-the-art methods, it can also benefit from improving the quality of the candidate results. It was also reported in other re-ranking works that a larger k (eg. $k > 64$) results the worse performance. We think the reason is that the oracle best increases when k is larger, but the oracle worst decrease with larger degree. The error types increase greatly. The re-ranking model requires more negative samples to avoid overfitting. When k is larger, the number of negative samples also needs to multiply increase for training. However, we just can obtain at most k negative samples from the k-best outputs of the base parser.

The experiments also show that the our model can achieves significant improvements by adding the oracles into the output lists of the base parser. This indicates that our model can be boosted by a better set of the candidate results, which can be implemented by combining the RCNN in the decoding algorithm.

8 Related Work

There have been several works to use neural networks and distributed representation for dependency parsing.

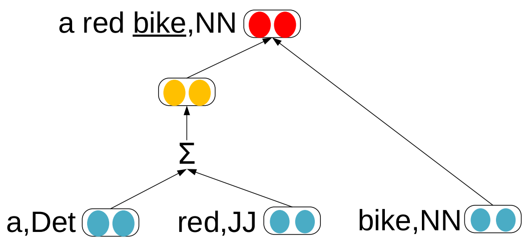


Figure 7: Example of a DT-RNN unit

Stenetorp (2013) attempted to build recursive neural networks for transition-based dependency parsing, however the empirical performance of his model is still unsatisfactory. 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 methods aim to transition-based parsing and can not model the sentence in semantic vector space for other NLP tasks.

Socher et al. (2013b) proposed a compositional vectors computed by dependency tree RNN (DT-RNN) to map sentences and images into a common embedding space. However, there are two major differences as follows. 1) They first summed up all child nodes into a dense vector $\mathbf{v}_c$ and then composed subtree representation from $\mathbf{v}_c$ and vector parent node. In contrast, our model first combine the parent and each child and then choose the most informative features with a pooling layer. 2) We represent the relative position of each child and its parent with distributed representation (position embeddings), which is very useful for convolutional layer. Figure 7 shows an example of DTRNN to illustrates how RCNN represents phrases as continuous vectors.

Specific to the re-ranking model, 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). Unlike IORNN, our proposed RCNN is a discriminative model and can optimize the re-ranking strategy for a particular base parser. Another difference is that RCNN computes the score of tree in a recursive way, which is more natural for the hierarchical structure of natural language. Besides, the RCNN can not only be used for the re-ranking, but also be regarded as general model to represent sentence with its dependency tree.

9 Conclusion

In this work, we address the problem to represent all level nodes (words or phrases) with dense representations in a dependency tree. We propose a recursive convolutional neural network (RCNN) architecture to capture the syntactic and compositional-semantic representations of phrases and words. RCNN is a general architecture and can deal with k-ary parsing tree, therefore RCNN is very suitable for many NLP tasks to minimize the effort in feature engineering with a external dependency parser. Although RCNN is just used for the re-ranking of the dependency parser in this paper, it can be regarded as semantic modelling of text sequences and handle the input sequences of varying length into a fixed-length vector. The parameters in RCNN can be learned jointly with some other NLP tasks, such as text classification.

For the future research, we will develop an integrated parser to combine RCNN with a decoding algorithm. We believe that the integrated parser can achieve better performance without the limitation of base parser. Moreover, we also wish to investigate the ability of our model for 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), the National High Technology Research and Development Program of China (2015AA015408), Shanghai Science and Technology Development Funds (14ZR1403200), Shanghai Leading Academic Discipline Project (B114).

References

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*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June. 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. Dublin City University and Association for Computational Linguistics.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

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.

Fabrizio Costa, Paolo Frasconi, Vincenzo Lombardo, and Giovanni Soda. 2003. Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*, 19(1-2):9–25.

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.

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

Joshua Goodman. 1998. Parsing inside-out. *arXiv preprint cmp-lg/9805007*.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *TACL*, 1:139–150.

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. Association for Computational Linguistics.

Eric Huang, Richard Socher, Christopher Manning, and Andrew Ng. 2012. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 873–882, Jeju Island, Korea, July. Association for Computational Linguistics.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of ICML*.

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.

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.

Sauro Menchetti, Fabrizio Costa, Paolo Frasconi, and Massimiliano Pontil. 2005. Wide coverage natural language processing using kernel methods and neural networks for structured data. *Pattern Recognition Letters*, 26(12):1896–1906.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*.

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.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. 2007. (online) subgradient methods for structured prediction. In *Eleventh International Conference on Artificial Intelligence and Statistics (AIStats)*.

Federico Sangati, Willem Zuidema, and Rens Bod. 2009. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 238–241. Association for Computational Linguistics.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 129–136.

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

Richard Socher, Q Le, C Manning, and A Ng. 2013b. Grounded compositional semantics for finding and describing images with sentences. In *NIPS Deep Learning Workshop*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013c. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

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 *ACL*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, volume 3.

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. Association for Computational Linguistics.

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. Association for Computational Linguistics.