# Probabilistic Graph-based Dependency Parsing with Convolutional Neural Network

**Zhisong Zhang**[1,2], **Hai Zhao**[1,2,*], **Lianhui Qin**[1,2]
[1]Department of Computer Science and Engineering,
Shanghai Jiao Tong University, Shanghai, 200240, China
[2]Key Laboratory of Shanghai Education Commission for Intelligent Interaction
and Cognitive Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China
{zzs2011,qinlianhui}@sjtu.edu.cn,zhaohai@cs.sjtu.edu.cn

## Abstract

This paper presents neural probabilistic parsing models which explore up to third-order graph-based parsing with maximum likelihood training criteria. Two neural network extensions are exploited for performance improvement. Firstly, a convolutional layer that absorbs the influences of all words in a sentence is used so that sentence-level information can be effectively captured. Secondly, a linear layer is added to integrate different order neural models and trained with perceptron method. The proposed parsers are evaluated on English and Chinese Penn Treebanks and obtain competitive accuracies.

## 1 Introduction

Neural network methods have shown great promise in the field of parsing and other related natural language processing tasks, exploiting more complex features with distributed representation and non-linear neural network (Wang et al., 2013; Wang et al., 2014; Cai and Zhao, 2016; Wang et al., 2016). In transition-based dependency parsing, neural models that can represent the partial or whole parsing histories have been explored (Weiss et al., 2015; Dyer et al., 2015). While for graph-based parsing, on which we focus in this work, Pei et al. (2015) also show the effectiveness of neural methods.

*Corresponding author. This paper was partially supported by Cai Yuanpei Program (CSC No. 201304490199 and No. 201304490171), National Natural Science Foundation of China (No. 61170114 and No. 61272248), National Basic Research Program of China (No. 2013CB329401), Major Basic Research Program of Shanghai Science and Technology Committee (No. 15JC1400103), Art and Science Interdisciplinary Funds of Shanghai Jiao Tong University (No. 14JCRZ04), and Key Project of National Society Science Foundation of China (No. 15-ZDA041).

The graph-based parser generally consists of two components: one is the parsing algorithm for inference or searching the most likely parse tree, the other is the parameter estimation approach for the machine learning models. For the former, classical dynamic programming algorithms are usually adopted, while for the latter, there are various solutions. Like some previous neural methods (Socher et al., 2010; Socher et al., 2013), to tackle the structure prediction problems, Pei et al. (2015) utilize a max-margin training criterion, which does not include probabilistic explanations. Re-visiting the traditional probabilistic criteria in log-linear models, this work utilizes maximum likelihood for neural network training. Durrett and Klein (2015) adopt this method for constituency parsing, which scores the anchored rules with neural models and formalizes the probabilities with tree-structured random fields. Motivated by this work, we utilize the probabilistic treatment for dependency parsing: scoring the edges or high-order sub-trees with a neural model and calculating the gradients according to probabilistic criteria. Although scores are computed by a neural network, the existing dynamic programming algorithms for gradient calculation remain the same as those in log-linear models.

Graph-based methods search globally through the whole space for trees and get the highest-scored one, however, the scores for the sub-trees are usually locally decided, considering only surrounding words within a limited-sized window. Convolutional neural network (CNN) provides a natural way to model a whole sentence. By introducing a distance-aware convolutional layer, sentence-level representation can be exploited for parsing. We will especially verify the effectiveness of such representation incorporated with window-based representation.

Graph-based parsing has a natural extension



*Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1382–1392,
Berlin, Germany, August 7-12, 2016. 

through raising its order and higher-order parsers usually perform better. In previous work on high-order graph-parsing, the scores of high-order sub-trees usually include the lower-order parts in their high-order factorizations. In traditional linear models, combining scores can be implemented by including low-order features. However, for neural models, this is not that straightforward because of nonlinearity. A straightforward strategy is simply adding up all the scores, which in fact works well; another way is stacking a linear layer on the top of the representation from various already-trained neural parsing models of different orders.

This paper presents neural probabilistic models for graph-based projective dependency parsing, and explores up to third-order models. Here are the three highlights of the proposed methods:

- Probabilistic criteria for neural network training. (Section 2.2)
- Sentence-level representation learned from a convolutional layer. (Section 3.2)
- Ensemble models with a stacked linear output layer. (Section 3.3)

Our main contribution is exploring sub-tree scoring models which combine local features with a window-based neural network and global features from a distance-aware convolutional neural network. A free distribution of our implementation is publicly available[1].

The remainder of the paper is organized as follows: Section 2 explains the probabilistic model for graph-based parsing, Section 3 describes our neural network models, Section 4 presents our experiments and Section 5 discusses related work, we summarize this paper in Section 6.

## 2 Probabilistic Graph-based Dependency Parsing

### 2.1 Graph-based Dependency Parsing

Dependency parsing aims to predict a dependency tree, in which all the edges connect head-modifier pairs. In graph-based methods, a dependency tree is factored into sub-trees, from single edge to multiple edges with different patterns; we will call these specified sub-trees *factors* in this paper. According to the sub-tree size of the factors, we can define the order of the graph model. Three different ordered factorizations considered in this work and their sub-tree patterns are shown in Figure 1.

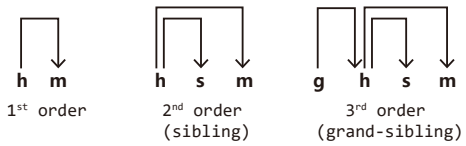


Figure 1: The decompositions of factors.

The score for a dependency tree ($T$) is defined as the sum of the scores of all its factors ($p$):

$$Score(T) = \sum_{p \in T} Score(p)$$

In this way, the dependency parsing task is to find a max-scoring tree. For projective dependency parsing considered in this work, this searching problem is conquered by dynamic programming algorithms with the key assumption that the factors are scored independently. Previous work (Eisner, 1996; McDonald et al., 2005; McDonald and Pereira, 2006; Koo and Collins, 2010; Ma and Zhao, 2012) explores ingenious algorithms for decoding ranging from first-order to higher-orders. Our proposed parsers also take these algorithms as backbones and use them for inference.

### 2.2 Probabilistic Model

With the graph factorization and inference, the remaining problems are how to obtain the scores and how to train the scoring model. For the scoring models, traditional linear methods utilize manually specified features and linear scoring models, while we adopt neural network models, which may exploit better feature representations.

For the training methods, in recent neural graph-based parsers, non-probabilistic margin-based methods are usually used. However, following the maximum likelihood criteria in traditional log-linear models, we can treat it in a probabilistic way. In fact, the probabilistic treatment still utilizes the scores of sub-tree factors in graph models. As in log-linear models like Conditional Random Field (CRF) (Lafferty et al., 2001), the exponentials of scores are taken before re-normalizing, and the probability distribution over trees condi-

[1] https://github.com/zzsfornlp/nnpgdparser

tioned on a sentence $X$ is defined as follows:

$$Pr(T|X,\theta) = \frac{1}{Z(X)}\exp(Score(T|\theta))$$
$$Z(X) = \sum_{T'}\exp(Score(T'|\theta))$$

where $\theta$ represents the parameters and $Z(X)$ is the re-normalization partition function. The intuition is that the higher the score is, the more potential or mass it will get, leading to higher probability.

The training criteria will be log-likelihood in the classical setting of maximum likelihood estimation, and we define the loss for a parse tree as negative log-likelihood:

$$\begin{aligned} L(\theta) &= -\log Pr(T_g|X,\theta) \\ &= -Score(T_g|\theta) + \log(Z(X)) \end{aligned}$$

where $T_g$ stands for the golden parse tree. Now we need to calculate the gradients of $\theta$ according to gradient-based optimization. Focusing on the second term, we have (some conditions are left out for simplicity):

$$\begin{aligned} \frac{\partial \log(Z(X))}{\partial\theta} &= \sum_{T'} Pr(T') \sum_{p\in T'} \frac{\partial Score(p)}{\partial\theta} \\ &= \sum_{p} \frac{\partial Score(p)}{\partial\theta} \sum_{T'\in T(p)} Pr(T') \end{aligned}$$

Here, $T(p)$ is the set of trees that contain the factor $p$, and the inner summation is defined as the marginal probability $m(p)$:

$$m(p) = \sum_{T'\in T(p)} Pr(T')$$

which can be viewed as the mass of all the trees containing the specified factor $p$. The calculation of $m(p)$ (Paskin, 2001; Ma and Zhao, 2015) is solved by a variant of inside-outside algorithm, which is of the same complexity compared with the corresponding inference algorithms. Finally, the gradients can be represented as:

$$\frac{\partial L(\theta)}{\partial\theta} = \sum_{p} \frac{\partial Score(p)}{\partial\theta}\Big(-\big[p\in T_g\big] + m(p)\Big)$$

where $[p \in T_g]$ is a binary value which indicates whether $p$ is in tree $T_g$.

Traditional models usually utilize linear functions for the *Score* function, which might need carefully feature engineering such as (Zhao et al., 2009a; Zhao et al., 2009b; Zhao et al., 2009c; Zhao, 2009; Zhao et al., 2013), while we adopt neural models with the probabilistic training criteria unchanged.

### 2.3 Training Criteria

We take a further look between the maximum-likelihood criteria and the max-margin criteria. For the max-margin method, the loss is the difference between the scores of the golden tree and a predicted tree, and its sub-gradient can be written in a similar form:

$$\frac{\partial L_m(\theta)}{\partial\theta} = \sum_{p} \frac{\partial Score(p)}{\partial\theta}\Big(-\big[p\in T_g\big] + \big[p\in T_b\big]\Big)$$

Here, the predicted tree $T_b$ is the best-scored tree with a structured margin loss in the score.

Comparing the derivatives, we can see that the one of probabilistic criteria can be viewed as a soft version of the max-margin criteria, and all the possible factors are considered when calculating gradients for the probabilistic way, while only wrongly predicted factors have non-zero sub-gradients for max-margin training. This observation is not new and Gimpel and Smith (2010) provide a good review of several training criteria. It might be interesting to explore the impacts of different training criteria on the parsing performance, and we will leave it for future research.

### 2.4 Labeled Parsing

In a dependency tree, each edge can be given a label indicating the type of the dependency relation, this labeling procedure can be integrated directly into the parsing task, instead of a second pass after obtaining the structure.

For the probabilistic model, integrating labeled parsing only needs some extensions for the inference procedure and marginal probability calculations. For the simplicity, we only consider a single label for each factor (even for high-order ones) which corresponds to Model 1 in (Ma and Hovy, 2015): the label of the edge between head and modifier word, which will only multiply $O(l)$ to the complexity. We find this direct approach not only achieves labeled parsing in one pass, but also improves unlabeled attachment accuracies (see Section 4.3), which may benefit from the joint learning with the labels.

## 3 Neural Model

The task for the neural models is computing the labeled scores of the factors. The inputs are the words in a factor with contexts, and the outputs are the scores for this factor to be valid in the dependency tree. We propose neural models to in-

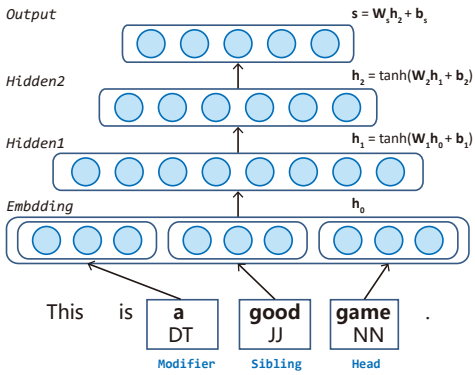


Figure 2: The architecture for the basic model (second order parsing).

tegrate features from both local word-neighboring windows and the entire sentence, and furthermore explore ensemble models with different orders.

### 3.1 Basic Local Model

**Architecture** The basic model uses a window-based approach, which includes only surrounding words for the contexts. Figure 2 illustrates a second-order sibling model and models of other orders adopt similar structures. It is simply a standard feed-forward neural network with two hidden layers ($\mathbf{h}_1$ and $\mathbf{h}_2$) above the embedding layer ($\mathbf{h}_0$), the hidden layers all adopt $tanh$ activation function, and the output layer (noted as $\mathbf{s}$) directly represents the scores for different labels.

**Feature Sets** All the features representing the input factor are atomic and projected to embeddings, then the embedding layer is formed by concatenating them. There are three categories of features: word forms, POS (part-of-speech) tags and distances. For each node in the factor, word forms and POS tags of the surrounding words in a specified window are also considered. Special tokens for start or end of sentences, root node and unknown words are added for both word forms and POS tags. Distances can be negative or positive to represent the relative positions between the factor nodes in surface string. Take the situation for the second-order model as an example, there are three nodes in a factor: $h$ for head, $m$ for modifier and $s$ for sibling. When considering three-word windows, there will be three word forms and three tags for each node and its surrounding context. $m$ and $s$ both have one distance feature while $h$ does not have one as its parent does not exist in the factor.

**Training** As stated in Section 2.2, we use the maximum likelihood criteria. Moreover, we add two L2-regularizations: one is for all the weights $\theta'$ (biases and embeddings not included) to avoid over-fitting and another is for preventing the final output scores from growing too large. The former is common practice for neural network, while the latter is to set soft limits for the norms of the scores. Although the second term is not usually adopted, it directly puts soft constraints on the scores and improves the accuracies (about 0.1% for UAS/LAS overall) according to our primary experiments. So the final loss function will be:

$$L'(\theta) = \sum_p \Big( Score(p) \cdot \big( - \big[ p \in T_g \big] + m(p) \big) + \lambda_s \cdot Score(p)^2 \Big) + \lambda_m \cdot \|\theta'\|^2$$

where $\lambda_m$ and $\lambda_s$ respectively represent regularization parameters for model and scores. The training process utilizes a mini-batched stochastic gradient descent method with momentum.

**Comparisons** Our basic model resembles the one of Pei et al. (2015), but with some major differences: probabilistic training criteria are adopted, the structures of the proposed networks are different and direction information is encoded in distance features. Moreover, they simply average embeddings in specified regions for phrase-embedding, while we will include sentence-embedding in convolutional model as follows.

### 3.2 Convolutional Model

To encode sentence-level information and obtain sentence embeddings, a convolutional layer of the whole sentence followed by a max-pooling layer is adopted. However, we intend to score a factor in a sentence and the position of the nodes should also be encoded. The scheme is to use the distance embedding for the whole convolution window as the position feature.

We will take the second-order model as an example to introduce the related operations. Figure 3 shows the convolution operation for a convolution window, the input atomic features are the word forms and POS tags for each word inside the window, and the distances of only the center word (assuming an odd-sized window) to specified nodes in the factor are adopted as position features. In the example, "*game-good-a*" is to be scored as a second-order sibling factor, and for a

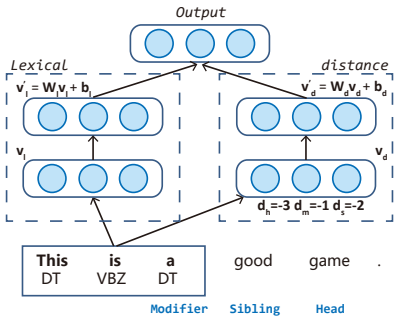


Figure 3: The operations for one convolution window (second order parsing).

convolution window of "*This is a*", word forms and corresponding POS tags are projected to embeddings and concatenated as the lexical vector $\mathbf{v}_l$, the distances of the center word "*is*" to all the three nodes in the factor are also projected to embeddings and concatenated as the distance vector $\mathbf{v}_d$, then these two vectors go through difference linear transformations into the same dimension and are combined together through element-wise addition or multiplication.

In general, assuming after the projection layer, embeddings of the word forms and POS tags of the sentence are represented as $[\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_{n-1}]$ and $[\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{n-1}]$. Those embeddings in the basic model may be reused here by sharing the embedding look-up table. The second-order sibling factor to be scored has nodes with indexes of $m$ (modifier), $h$ (head) and $s$ (sibling). The distance embeddings are denoted by $\mathbf{d}$, which can be either negative or positive. These distance embeddings are different from the ones in the basic model, because here we measure the distances between the convolution window (its center word) and factor nodes, while the distances between nodes inside the factors are measured in the basic model.

For a specified window $[i : j]$, always assuming an odd number sized window, and the center token is indexed to $c = \frac{i+j}{2}$, the $\mathbf{v}_l$ and $\mathbf{v}_d$ are obtained through simple concatenation:

$$\mathbf{v}_l = [\mathbf{w}_i, \mathbf{p}_i, \mathbf{w}_{i+1}, \mathbf{p}_{i+1}, ..., \mathbf{w}_j, \mathbf{p}_j]$$
$$\mathbf{v}_d = [\mathbf{d}_{c-h}, \mathbf{d}_{c-m}, \mathbf{d}_{c-s}]$$

then $\mathbf{v}_l$ and $\mathbf{v}_d$ go through difference linear transformations into same dimension space: $\mathbf{v'}_l, \mathbf{v'}_d \in \mathbb{R}^n$, where $n$ is also the dimension of the output vector $\mathbf{v}_o$ for the window. The linear operations can be expressed as:

$$\mathbf{v'}_l = \mathbf{W}_l \cdot \mathbf{v}_l + \mathbf{b}_l$$
$$\mathbf{v'}_d = \mathbf{W}_d \cdot \mathbf{v}_d + \mathbf{b}_d$$

The final vector $\mathbf{v}_o$ is obtained by element-wise operations of $\mathbf{v'}_l$ and $\mathbf{v'}_d$. We consider two strategies: (1) $add$: simple element-wise addition, (2) $mul$: element-wise multiplication with $\mathbf{v'}_d$ activated by $tanh$. They can be formalized as:

$$\mathbf{v}_{o\text{-}add} = \mathbf{v'}_l \oplus \mathbf{v'}_d$$
$$\mathbf{v}_{o\text{-}mul} = \mathbf{v'}_l \odot tanh(\mathbf{v'}_d)$$

All the windows whose center-located word is valid (exists) in the sentence are considered and we will get a sequence of convolution outputs whose number is the same as the sentence length. The convolution outputs (all $\mathbf{v}_o$) are collapsed into one global vector $\mathbf{v}_g$ using a standard max-pooling operation. Finally, for utilizing the sentence-level representation in the basic model, we can either replace the original first hidden layer $\mathbf{h}_1$ with $\mathbf{v}_g$ or concatenate $\mathbf{v}_g$ to $\mathbf{h}_1$ for combining local and global features.

### 3.3 Ensemble Models

For higher-order dependency parsing, it is a standard practice to include the impact of lower-order parts in the scoring of higher-order factors, which actually is an ensemble method of different order models for scoring.

A simple adding scheme is often used. For nonlinear neural models, we use an explicit adding method. For example, in third-order parsing, the final score for the factor $(g, h, m, s)$ will be:

$$\mathbf{s}_{add}(g, h, m, s) = \mathbf{s}_{o3}(g, h, m, s) + \mathbf{s}_{o2}(h, m, s) + \mathbf{s}_{o1}(h, m)$$

Here, $g$, $h$, $m$ and $s$ represent the grandparent, head, modifier and sibling nodes in the grand-sibling third-order factor; $\mathbf{s}_{o1}$, $\mathbf{s}_{o2}$ and $\mathbf{s}_{o3}$ stand for the corresponding lower-order scores from first, second and third order models, respectively.

We notice that ensemble or stacking methods for dependency parsing have explored in previous work (Nivre and McDonald, 2008; Torres Martins et al., 2008). Recently, Weiss et al. (2015) stack a linear layer for the final scoring in a single model, and we extend this method to combine multiple models by stacking a linear layer on their output and hidden layers. The simple adding scheme can

be viewed as adopting a final layer with specially fixed weights.

For each model to be combined, we concatenate the output layer and all hidden layers (except embedding layer $\mathbf{h}_0$):

$$\mathbf{v}_{all} = [\mathbf{s}, \mathbf{h}_1, \mathbf{h}_2]$$

All $\mathbf{v}_{all}$ from different models are again concatenated to form the input for the final linear layer and the final scores are obtained through a linear transformation (no bias adding):

$$\mathbf{v}_{combine} = [\mathbf{v}_{all\text{-}o1}, \mathbf{v}_{all\text{-}o2}, \mathbf{v}_{all\text{-}o3}]$$
$$\mathbf{s}_{combine} = \mathbf{W}_{combine} \cdot \mathbf{v}_{combine}$$

We no longer update weights for the underlying neural models, and the learning of the final layer is equally training a linear model, for which structured average perceptron (Collins, 2002; Collins and Roark, 2004) is adopted for simplicity.

This ensemble scheme can be extended in several ways which might be explored in future work: (1) feed-forward network can be stacked rather than a single linear layer, (2) traditional sparse features can also be concatenated to $\mathbf{v}_{combine}$ to combine manually specified representations with distributed neural representations as in (Zhang and Zhang, 2015).

## 4 Experiments

The proposed parsers are evaluated on English Penn Treebank (PTB) and Chinese Penn Treebank (CTB). Unlabeled attachment scores (UAS), labeled attachment scores (LAS) and unlabeled complete matches (CM) are the metrics. Punctuations[2] are ignored as in previous work (Koo and Collins, 2010; Zhang and Clark, 2008).

For English, we follow the splitting convention for *PTB3*: sections 2-21 for training, 22 for developing and 23 for test. We prepare three datasets of PTB, using different conversion tools: (1) Penn2Malt[3] and the head rules of Yamada and Matsumoto (2003), noted as *PTB-Y&M*; (2) dependency converter in Stanford parser v3.3.0 with Stanford Basic Dependencies (De Marneffe et al., 2006), noted as *PTB-SD*; (3) LTH Constituent-to-Dependency Conversion Tool[4] (Johansson and Nugues, 2007), noted as *PTB-LTH*. We use Stanford POS tagger (Toutanova et al., 2003) to get predicted POS tags for development and test sets, and the accuracies for their tags are 97.2% and 97.4%, respectively.

For Chinese, we adopt the splitting convention for *CTB5* described in (Zhang and Clark, 2008). The dependencies (noted as *CTB*), are converted with the Penn2Malt converter. Gold segmentation and POS tags are used as in previous work.

### 4.1 Settings

Settings of our models will be described in this sub-section, including pre-processing and initializations, hyper-parameters, and training details.

We ignore the words that occur less than 3 times in the training treebank and use a special token to replace them. For English parsing, we initialize word embeddings with word vectors trained on Wikipedia using *word2vec* (Mikolov et al., 2013); all other weights and biases are initialized randomly with uniform distribution.

For the structures of neural models, all the embeddings (word, POS and distances) have dimensions of 50. For basic local models, $\mathbf{h}_1$ and $\mathbf{h}_2$ are set to 200 and 100, and the local window size is set to 7. For convolutional models, a three-word-sized window for convolution is specified, and convolution output dimension (number of filters) is 100. When concatenating the convolution vector (after pooling) to $\mathbf{h}_1$, it will make the first hidden layer's dimension 300.

For the training of neural network, we set the initial learning rate to 0.1 and the momentum to 0.6. After each iteration, the parser is tested on the development set and if the accuracy decreases, the learning rate will be halved. The learning rate will also be halved if no decreases of the accuracy for three epochs. We train the neural models for 12 epochs and select the one that performs best on the development set. The regularization parameters $\lambda_m$ and $\lambda_s$ are set to 0.0001 and 0.001. For the perceptron training of the ensemble model, only one epoch is enough based on the results of the development set.

The runtime of the model is influenced by the hyper-parameter setting. According to our experiments, using dual-core on 3.0 GHz i7 CPU, the training costs 6 to 15 hours for different-order models and the testing is comparably efficient as recent neural graph-parsers. The calculation of the

[2]Tokens whose gold POS tags are one of {“ ” : , .} for PTB or *PU* for CTB.

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

[4]http://nlp.cs.lth.se/software/treebank_converter

| Method | UAS | LAS | CM |
|---|---|---|---|
| *Basic (first-order)* | | | |
| Unlabeled | 91.53 | – | 42.82 |
| Labeled | 92.13 | 89.60 | 45.06 |
| Labeled+pre-training | 92.19 | 89.73 | 45.18 |
| *Convolutional (first-order)* | | | |
| replace-add | 92.26 | 89.83 | 44.76 |
| replace-mul | 92.02 | 89.61 | 44.24 |
| concatenate-add | 92.63 | 90.20 | 46.18 |
| concatenate-mul | 92.33 | 89.83 | 44.94 |
| *Higher-orders* | | | |
| o2-nope | 92.85 | 90.51 | 49.65 |
| o2-adding | 93.47 | 91.13 | 51.41 |
| o2-perceptron | 93.63 | 91.39 | 51.53 |
| o3-nope | 92.47 | 90.01 | 49.06 |
| o3-adding | 93.70 | 91.37 | 53.53 |
| o3-perceptron | 93.51 | 91.20 | 51.76 |

Table 1: Effects of the components, on *PTB-SD* development set.

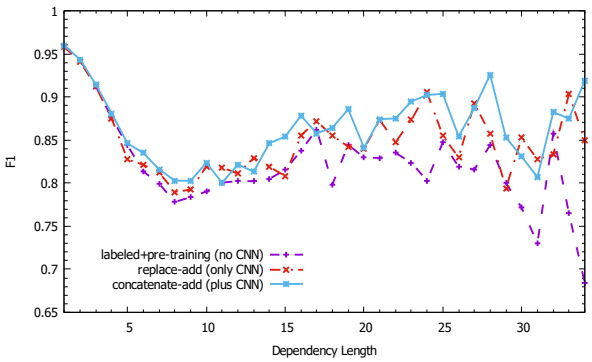


Figure 4: F1 measure of different dependency lengths, on *PTB-SD* development set.

convolution model approximately takes up 40% of all computations. The convolution operation indeed costs more, but the lexical parts $\mathbf{v}'_l$ of the convolution do not concern the factors and are computed only once for one sentence, which makes it less computationally expensive.

### 4.2 Pruning

For high-order parsing, the computation cost rises in proportion to the length of the sentence, and it will be too expensive to calculate scores for all the factors. Fortunately, many edges are quite unlikely to be valid and can be pruned away using low-order models. We follow the method of Koo and Collins (2010) and directly use the first-order probabilistic neural parser for pruning. We compute the marginal probability $m(h, m)$ for each edge and prune away the edges whose marginal probability is below $\epsilon \times \max_{h'} m(h', m)$. $\epsilon$ means the pruning threshold that is set to 0.0001 for second-order. For third-order parsing, considering the computational cost, we set it to 0.001.

### 4.3 Model Analysis

This section presents experiments to verify the effectiveness of the proposed methods and only the *PTB-SD* development set will be used in these experiments, which fall into three groups concerning basic models, convolutional models and ensemble ones, as shown in Table 1.

The first group focuses on the basic local models of first order. The first two, *Unlabeled* and *Labeled*, do not use pre-training vectors for initialization, while the third, *Labeled+pre-training*, utilizes them. The *Unlabeled* does not utilize the labels in training set and its model only gives one dependency score (we do not train a second stage labeling model, so the LAS of the unlabeled one is not available) and the *Labeled* directly predicts the scores for all labels. We can see that labeled parsing not only demonstrates the convenience of outputting dependency relations and labels for once, but also obtains better parsing performances. Also, we observe that pre-trained word vectors bring slight improvements. Pre-trained initialization and labeled parsing will be adopted for the next two groups and the rest experiments.

Next, we explore the effectiveness of the CNN enhancement. In the four entries of this group, *concatenate* or *replace* means whether to concatenate the sentence-level vector $\mathbf{v}_g$ to the first hidden layer $\mathbf{h}_1$ or just replace it (just throw away the representation from basic models), *add* or *mul* means to use which way for attaching distance information. Surprisingly, simple adding method surpasses the more complex multiplication-with-activation method, which might indicate that the direct activation operation may not be suitable for encoding distance information. With no surprises, the concatenating method works better because it combines both the local window-based and global sentence-level information. We also explore the influences of the convolution operations on dependencies of different lengths, as shown in Figure 4, the convolutional methods help the decisions of long-range dependencies generally. For the high-order parsing in the rest of this paper, we will all adopt the *concatenate-add* setting.

In the third group, we can see that high-order parsing brings significant performance improvement. For high-order parsing, three ensemble schemes are examined: no combination, adding

| | *PTB-Y&M* | | | *PTB-SD* | | | *PTB-LTH* | | | *CTB* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | UAS | LAS | CM | UAS | LAS | CM | UAS | LAS | CM | UAS | LAS | CM |
| *Graph-NN:proposed* | | | | | | | | | | | | |
| o3-adding | 93.20 | 92.12 | 48.92 | 93.42 | 91.29 | 50.37 | **93.14** | 90.07 | 43.38 | 87.55 | **86.19** | 35.65 |
| o3-perceptron | **93.31** | 92.23 | 50.00 | 93.42 | 91.26 | 49.92 | 93.12 | 89.53 | 43.83 | **87.65** | 86.17 | 36.07 |
| *Graph-NN:others* | | | | | | | | | | | | |
| Pei et al. (2015) | 93.29 | 92.13 | – | – | – | – | – | – | – | – | – | – |
| Fonseca and Aluísio (2015) | – | – | – | – | – | – | 91.6– | 88.9– | – | – | – | – |
| Zhang and Zhao (2015) | – | – | – | – | – | – | 92.52 | – | 41.10 | 86.01 | – | 31.88 |
| *Graph-Linear* | | | | | | | | | | | | |
| Koo and Collins (2010) | 93.04 | – | – | – | – | – | – | – | – | – | – | – |
| Martins et al. (2013) | 93.07 | – | – | 92.82 | – | – | – | – | – | – | – | – |
| Ma and Zhao (2015) | 93.0– | – | 48.8– | – | – | – | – | – | – | 87.2– | – | 37.0– |
| *Transition-NN* | | | | | | | | | | | | |
| Chen and Manning (2014) | – | – | – | 91.8– | 89.6– | – | 92.0– | **90.7–** | – | 83.9– | 82.4– | – |
| Dyer et al. (2015) | – | – | – | 93.1– | 90.9– | – | – | – | – | 87.2– | 85.7– | – |
| Weiss et al. (2015) | – | – | – | **93.99** | **92.05** | – | – | – | – | – | – | – |
| Zhou et al. (2015) | 93.28 | **92.35** | – | – | – | – | – | – | – | – | – | – |

Table 2: Comparisons of results on the test sets.

and stacking another linear perceptron layer (with the suffixes of *-nope*, *-adding* and *-perceptron* respectively). The results show that model ensemble improves the accuracies quite a few. For third-order parsing, the no-combination method performs quite poorly compared to the others, which may be caused by the relative strict setting of the pruning threshold. Nevertheless, with model ensemble, the third-order models perform better than the second-order ones. Though the perceptron strategy does not work well for third-order parsing in this dataset, it is still more general than the simple adding method, since the latter can be seen as a special parameter setting of the former.

### 4.4 Results

We show the results of two of the best proposed parsers: third-order adding (*o3-adding*) and third-order perceptron (*o3-perceptron*) methods, and compare with the reported results of some previous work in Table 2. We compare with three categories of models: other *Graph*-based *NN* (neural network) models, traditional *Graph*-based *Linear* models and *Transition*-based *NN* models. For PTB, there have been several different dependency converters which lead to different sets of dependencies and we choose three of the most popular ones for more comprehensive comparisons. Since not all work report results on all of these dependencies, some of the entries might be not available.

From the comparison, we see that the proposed parser has output competitive performance for different dependency conversion conventions and treebanks. Compared with traditional graph-based linear models, neural models may benefit from better feature representations and more general non-linear transformations.

The results and comparisons in Table 2 demonstrate the proposed models can obtain comparable accuracies, which show the effectiveness of combining local and global features through window-based and convolutional neural networks.

## 5 Related Work

CNN has been explored in recent work of relation classification (Zeng et al., 2014; Chen et al., 2015), which resembles the task of deciding dependency relations in parsing. However, relation classification usually involves labeling for given arguments and seldom needs to consider the global structure. Parsing is more complex for it needs to predict structures and the use of CNN should be incorporated with the searching algorithms.

Neural network methods have been proved effective for graph-based parsing. Lei et al. (2014) explore a tensor scoring method, however, it needs to combine scores from linear models and we are not able to compare with it because of different datasets (they take datasets from CoNLL shared task). Zhang and Zhao (2015) also explore a probabilistic treatment, but its model may give mass to illegal trees or non-trees. Fonseca and Aluísio (2015) utilize CNN for scoring edges, though only explore first-order parsing. Its model is based on head selection for each modifier and might be difficult to be extended to high-order parsing. Recently, several neural re-ranking models, like Inside-Outside Recursive Neural Network

(Le and Zuidema, 2014) and Recursive CNN (Zhu et al., 2015), are utilized for capturing features with more contexts. However, re-ranking models depend on the underlying base parsers, which might already miss the correct trees. Generally, the re-ranking techniques play a role of additional enhancement for basic parsing models, and therefore they are not included in our comparisons.

The conditional log-likelihood probabilistic criterion utilized in this work is actually a (conditioned) Markov Random Field for tree structures, and it has been applied to parsing since long time ago. Johnson et al. (1999) utilize the Markov Random Fields for stochastic grammars and gradient based methods are adopted for parameter estimations, and Geman and Johnson (2002) extend this with dynamic programming algorithms for inference and marginal-probability calculation. Collins (2000) uses the same probabilistic treatment for re-ranking and the denominator only includes the candidate trees which can be seen as an approximation for the whole space of trees. Finkel et al. (2008) utilize it for feature-based parsing. The probabilistic training criterion for linear graph-based dependency models have been also explored in (Li et al., 2014; Ma and Zhao, 2015). However, these previous methods usually exploit log-linear models utilizing sparse features for input representations and linear models for score calculations, which are replaced by more sophisticated distributed representations and neural models, as shown in this work.

## 6 Conclusions

This work presents neural probabilistic graph-based models for dependency parsing, together with a convolutional part which could capture the sentence-level information. With distributed vectors for representations and complex non-linear neural network for calculations, the model can effectively capture more complex features when deciding the scores for sub-tree factors and experiments on standard treebanks show that the proposed techniques improve parsing accuracies.

## References


Deng Cai and Hai Zhao. 2016. Neural word segmentation learning for Chinese. In *Proceedings of ACL*, Berlin, Germany, August.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*, pages 740–750, Doha, Qatar, October.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of ACL*, pages 167–176, Beijing, China, July.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 25–70.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Greg Durrett and Dan Klein. 2015. Neural crf parsing. In *Proceedings of ACL*, pages 302–312, Beijing, China, July.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL*, pages 334–343, Beijing, China, July.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 340–345, Copenhagen, August.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL*, pages 959–967, Columbus, Ohio, June.

Erick Fonseca and Sandra Aluísio. 2015. A deep architecture for non-projective dependency parsing. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 56–61, Denver, Colorado, June.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 279–286, Philadelphia, Pennsylvania, USA, July.

Kevin Gimpel and Noah A. Smith. 2010. Softmax-margin crfs: Training log-linear models with cost functions. In *Proceedings of NAACL*, pages 733–736, Los Angeles, California, June.

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.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 535–541, College Park, Maryland, USA, June.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*, pages 1–11, Uppsala, Sweden, July.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of EMNLP*, pages 729–739, Doha, Qatar, October.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL*, pages 1381–1391, Baltimore, Maryland, June.

Zhenghua Li, Min Zhang, and Wenliang Chen. 2014. Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proceedings of ACL*, pages 457–467, Baltimore, Maryland, June.

Xuezhe Ma and Eduard Hovy. 2015. Efficient inner-to-outer greedy algorithm for higher-order labeled dependency parsing. In *Proceedings of EMNLP*, pages 1322–1328, Lisbon, Portugal, September.

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of COLING*, pages 785–796, Mumbai, India, December.

Xuezhe Ma and Hai Zhao. 2015. Probabilistic models for high-order projective dependency parsing. *arXiv preprint arXiv:1502.04174*.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of ACL*, pages 617–622, Sofia, Bulgaria, August.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, Ann Arbor, Michigan, June.

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*, pages 3111–3119.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL*, pages 950–958, Columbus, Ohio, June.

Mark A Paskin. 2001. Cubic-time parsing and learning algorithms for grammatical bigram models. Technical report.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proceedings of ACL*, pages 313–322, Beijing, China, July.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with compositional vector grammars. In *Proceedings of ACL*, pages 455–465, Sofia, Bulgaria, August.

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. In *Proceedings of ENNLP*, pages 157–166, Honolulu, Hawaii, October.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180.

Rui Wang, Masao Utiyama, Isao Goto, Eiichro Sumita, Hai Zhao, and Bao-Liang Lu. 2013. Converting continuous-space language models into n-gram language models for statistical machine translation. In *Proceedings of EMNLP*, pages 845–850, Seattle, Washington, USA, October.

Rui Wang, Hai Zhao, Bao-Liang Lu, Masao Utiyama, and Eiichiro Sumita. 2014. Neural network based bilingual language model growing for statistical machine translation. In *Proceedings of EMNLP*, pages 189–195, Doha, Qatar, October.

Peilu Wang, Yao Qian, Frank Soong, Lei He, and Hai Zhao. 2016. Learning distributed word representations for bidirectional lstm recurrent neural network. In *Proceedings of NAACL*, June.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL*, pages 323–333, Beijing, China, July.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proceedings of COLING*, pages 2335–2344, Dublin, Ireland, August.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*, pages 562–571, Honolulu, Hawaii, October.

Meishan Zhang and Yue Zhang. 2015. Combining discrete and continuous features for deterministic transition-based dependency parsing. In *Proceedings of EMNLP*, pages 1316–1321, Lisbon, Portugal, September.

Zhisong Zhang and Hai Zhao. 2015. High-order graph-based neural dependency parsing. In *Proceedings of the 29th Pacific Asia Conference on Language, Information, and Computation*, pages 114–123, Shanghai, China, October.

Hai Zhao, Wenliang Chen, and Chunyu Kit. 2009a. Semantic dependency parsing of NomBank and PropBank: An efficient integrated approach via a large-scale feature selection. In *Proceedings of EMNLP*, pages 30–39, Singapore, August.

Hai Zhao, Wenliang Chen, Chunyu Kity, and Guodong Zhou. 2009b. Multilingual dependency learning: A huge feature engineering method to semantic dependency parsing. In *Proceedings of CoNLL*, pages 55–60, Boulder, Colorado, June.

Hai Zhao, Yan Song, Chunyu Kit, and Guodong Zhou. 2009c. Cross language dependency parsing using a bilingual lexicon. In *Proceedings of ACL*, pages 55–63, Suntec, Singapore, August.

Hai Zhao, Xiaotian Zhang, and Chunyu Kit. 2013. Integrative semantic dependency parsing via efficient large-scale feature selection. *Journal of Artificial Intelligence Research*, 46:203–233.

Hai Zhao. 2009. Character-level dependencies in chinese: Usefulness and learning. In *Proceedings of EACL*, pages 879–887, Athens, Greece, March.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of ACL*, pages 1213–1222, Beijing, China, July.

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 ACL*, pages 1159–1168, Beijing, China, July.