High-order Graph-based Neural Dependency Parsing

Zhisong Zhang[1,2] **and Hai Zhao**[1,2,*,†]
[1]Center for Brain-Like Computing and Machine Intelligence,
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@sjtu.edu.cn, zhaohai@cs.sjtu.edu.cn

Abstract

In this work, we present a novel way of using neural network for graph-based dependency parsing, which fits the neural network into a simple probabilistic model and can be furthermore generalized to high-order parsing. Instead of the sparse features used in traditional methods, we utilize distributed dense feature representations for neural network, which give better feature representations. The proposed parsers are evaluated on English and Chinese Penn Treebanks. Compared to existing work, our parsers give competitive performance with much more efficient inference.

1 Introduction

There have been two classes of typical approaches for dependency parsing: transition-based parsing and graph-based parsing. The former parses sentences by making a series of shift-reduce decisions (Yamada and Matsumoto, 2003; Nivre, 2003), while the latter searches for a tree through graph algorithms by decomposing trees into factors. This paper will focus on graph-based methods, which are based on dynamic programming strategies (Eisner, 1996; McDonald et al., 2005; McDonald and Pereira, 2006). In this recent decade, extensions have been made to use high-order factors (Carreras, 2007; Koo and Collins, 2010) in graph models and the highest one considers fourth-order (Ma and Zhao, 2012). However, all those methods usually use sparse indicator features as inputs and linear models to get the scores for later inference process. They are easy to suffer from the problem of sparsity, and linear models can be insufficient to effectively integrate all the sparse features in spite of various rich context that can be potentially exploited.

Distributed representations and neural network provide a way to alleviate such a drawback (Bengio et al., 2003; Collobert et al., 2011). Instead of high-dimensional sparse indicator feature vectors, distributed representations use low-dimensional dense vectors (also known as embeddings) to represent the features, and then they are usually used in a neural network. For example, in the traditional methods, a word is usually expressed by a one-hot vector; while distributed representations use a dense vector. By appropriate representation learning (usually by back-propagations in neural network), these embeddings can replace traditional sparse features and perform quite well together with neural network.

In recent years, using distributed representations and neural network has gradually gained popularity in natural language processing (NLP) since the pioneer work of (Bengio et al., 2003). Several neural network language models have reported exciting results for the tasks of machine translation and speech recognition (Schwenk, 2007; Mikolov et al., 2010;

[*]Correspondence author.

[†]This work was partially supported by the National Natural Science Foundation of China (No. 61170114, and No. 61272248), the National Basic Research Program of China (No. 2013CB329401), the Science and Technology Commission of Shanghai Municipality (No. 13511500200), the European Union Seventh Framework Program (No. 247619), the Cai Yuanpei Program (CSC fund 201304490199 and 201304490171), and the art and science interdiscipline funds of Shanghai Jiao Tong University, No. 14X190040031, and the Key Project of National Society Science Foundation of China, No. 15-ZDA041.

Wang et al., 2013; Wang et al., 2014; Wang et al., 2015). Many other tasks of NLP have also been reconsidered using neural network, the SENNA system[1] (Collobert et al., 2011) solved the tasks of part-of-speech (POS) tagging, chunking, named entity recognition and semantic role labeling.

In this work, we utilize neural network for first-order, second-order and third-order graph-based dependency parsing, with the help of the existing graph-based parsing algorithms. For high-order parsing, it is performed after the first-order parser prunes unlikely parts of the parsing tree. We use neural network to learn dense representations for word, POS and distance information, and predict how likely the dependency relationships are for a sub-tree factor in the dependency tree. For unlabeled projective dependency parsing, we have put a free distribution of our implementation on the Internet[2].

The remainder of the paper is organized as follows: Section 2 discusses related work, Section 3 gives the background for graph-based dependency parsing, Section 4 describes our neural network model and how we utilize it with graph-based parsing and Section 5 presents our experiments, results and some discussions. We summarize this paper in Section 6.

2 Related Work

There has been a few of attempts to parse with neural network. For dependency parsing, (Chen and Manning, 2014) uses neural network for greedy transition-based dependency parsing. We explore graph-based methods in this work, which might be difficultly utilized with neural network. (Le and Zuidema, 2014) implements a generative dependency model with a recursive neural network, but the model is used for re-ranking which needs k-best candidates.

For constituency parsing, (Collobert, 2011) uses a convolutional neural network and solves the problem with a hierarchical tagging process. (Socher et al., 2010) and (Socher et al., 2013) use recursive neural network to model phrase-based parse trees, but their methods might be unlikely generalized to dependency parsing because a dependency parse tree has no non-terminal nodes while constituency parse trees are derived from the phrases structure.

Semi-supervised methods usually incorporate word representations as the embeddings for words in the projection layer in neural network; they usually make use of lots of unlabeled data to find the patterns in natural languages. If we utilize pre-trained word vectors (see in Section 5.1), our models can be regarded as semi-supervised to some extent. (Koo et al., 2008) uses Brown clustering algorithm to obtain word representations, but then transforms them into sparse features as additional features and again uses the traditional methods; while in neural network models including this work, the embeddings directly replace sparse features for inputs.

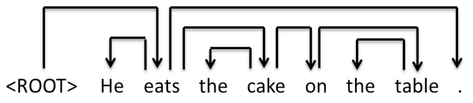

Figure 1: An example dependency tree.

3 Graph-based Dependency Parsing

3.1 Background of Dependency Parsing

Syntax information is important for many other tasks (Zhang and Zhao, 2013; Chen et al., 2015). As a classic syntactic problem, dependency parsing aims to predict a dependency tree, which directly represents head-modifier relationships between words in a sentence. Figure 1 shows a dependency tree, in which all the links connect head-modifier pairs. By enforcing that all the nodes must have one and only one parent and the resulting graphs should be acyclic and connected, we can get a directed dependency tree for a sentence (we usually add a dummy node $\langle root \rangle$ for the sentence as the highest level node).

Labels or dependency category can also be defined for the links in the dependency tree, however, this work will focus on unlabeled dependency parsing, because once the parsing tree has been built, labeling can be very effectively performed. Most dependency trees for most treebanks follow a useful constrain that is called projectiveness, i.e., no cross links exist in the tree. In treebanks for major languages such as English, nearly all sentences are pro-

[1]http://ronan.collobert.com/senna/

[2]https://github.com/zzsfornlp/nngdparser

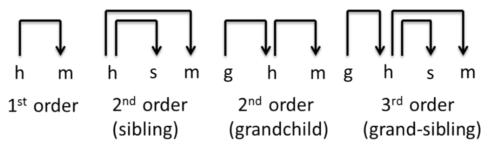

Figure 2: The decompositions of factors.

jective. Therefore this work also considers projective dependency parsing only.

3.2 Graph-based Methods and their Decompositions

In graph-based methods, dependency trees are decomposed into specific factors that do not influence with each other. Each factor, usually represented by a sub-tree, is given a score individually based on its features. The score for a whole dependency tree T is the summation of the scores of all the factors:

$$Score_{tree}(T) = \sum_{p \in factors(T)} Score_{factor}(p)$$

According to the sub-tree size of the factors, we can define the order of the graph model, some of the decomposition methods are shown in Figure 2. As the simplest case, the first-order model just considers sub-tree factor of single edge and its score is obtained by adding all the scores of the edges. For second-order models, another node is added into the factor, which can be either sibling or grandparent. For third-order models, the simplest form is the grand-sibling decomposition, which adds both sibling and grandparent nodes. Existing work also applied various decompositions, such as third-order tri-sibling (Koo and Collins, 2010) which considers two siblings of the modifier and fourth-order grand-tri-sibling (Ma and Zhao, 2012) which adds a grandparent node on tri-siblings.

For the sake of simplicity and the convenient use of neural network, we only consider four models discussed above (the sub-tree patterns of their factors are also shown in Figure 2). The notations for the four models are defined as follows:

- $o1$, first-order model
- $o2sib$, second-order model with sibling nodes
- $o2g$, second-order model with grandparent nodes
- $o3g$, third-order model with both sibling and grandparent nodes

3.3 Parsing Algorithms

Graph-based methods usually need to use dynamic programming based parsing algorithms, which make use of the scores of sub-trees for larger sub-trees in a bottom-up way. These algorithms solve the inference problem, that is, how to get an optimal tree given the scores for the parts. Our proposed parsers also take these algorithms as backbones and use them for inference.

In the traditional methods, scores are usually obtained directly from a linear model. In the learning phase, parameter estimation methods for structured linear models may adopt averaged perceptron (Collins, 2002; Collins and Roark, 2004) and max-margin methods (Taskar et al., 2004).

Still using all the existing parsing algorithms, this work focuses on improving scoring for the factors. In detail, our work uses neural network to determine the scores. Nevertheless the traditional methods might be difficultly extended to neural network because of the non-linearity. Therefore, we do not directly obtain scores from neural network. Instead we utilize a probabilistic model and obtain scores by some transformations, and then use these existing parsing algorithms for inference.

4 Neural Network Parsers

4.1 The Probabilistic Model

For graph-based dependency parsing, it is not straightforward to extend the linear models to the more powerful nonlinear neural network, because we need to figure out the scores for the factors of the tree, which are not specified in the original treebank. That is, we only know which factors are in the correct parsing tree, but there are no natural ways to indicate how they are scored; the only intuition is to give high scores to the right factors and low scores to the wrong ones.

In this work, a simple probabilistic model is adopted for the neural network parsers. It is one of Eisner's models (Eisner, 1996). Precisely, Eisner's model A is chosen and slightly modified for scoring. The model describes bi-gram lexical affinities, and it gives each possible link an affinity probability. The final probability of drawing a parsing tree for a sentence is the product of all the affinity probabilities. The original model also considers probabilities of words and tags and its formula is given as follows:

$$\begin{aligned}&Pr(words, tags, links)\\ &= Pr(words, tags)\cdot Pr(links\ present\ or\ not|words, tags)\\ &\approx Pr(words, tags)\cdot \prod_{1\le h,m\le n} Pr(L_{hm}|tword(h), tword(m))\end{aligned}$$

Unlike the original model, we determine only the probability for the parsing tree (the existence of the links):

$$Pr(T|S) = \prod_{\substack{0\le h\le length(S)\\ 0<m\le length(S)}} Pr(L_{hm}|context(h,m))$$

Here L_{hm} is a binary variable with Bernoulli distribution which means whether node h is the head of node m and $context(h,m)$ means the context of the two nodes which includes words, POS tags and distance.

When looking for the best tree, we simply find the tree with highest probability (we use logarithmic form for more convenient computations). Considering the single-headed constrain for dependency tree construction, if we assign 1 to L_{Hm}, which makes H the parent of m, we must assign 0 to all other L_{hm}, h means all the nodes that are not equal to H. The logarithmic probability can be rewritten as follows:

$$\begin{aligned}\log(Pr(T|S)) = &\sum_{0<m\le length(S)} \Big(\log\big(Pr(L_{Hm}=1)\big)\\ &+ \sum_{\substack{0\le h\le length(S)\\ h\ne H, h\ne m}} \log\big(Pr(L_{hm}=0)\big)\Big)\end{aligned}$$

Here H represents the real parent node of node m. The formula is in the form of summation of the factor scores, which are defined as:

$$\begin{aligned}Score(H,m) = &\log\big(Pr(L_{Hm}=1)\big)\\ &+ \sum_{\substack{0\le h\le length(S)\\ h\ne H, h\ne m}} \log\big(Pr(L_{hm}=0)\big)\end{aligned}$$

After defining the score of each dependency factor, we can apply the scores to the existing parsing algorithms (Eisner, 1996; McDonald et al., 2005).

4.2 High-Order Parsing

We now generalize the model to high-order parsing. In the first-order model, we define probabilities for the head-modifier pair, which is the factor for first-order parsing. Naturally, we can define probabilities for high-order factors. The probability of a parse tree is the product of all its factors (either existing ones or wrong ones), the probability for one factor is again a binary value which means whether the factor exists in the dependency tree.

Using single-headed constraint again, for all the factors with the same node as the children, only one can exist in a legal parsing tree. The similar transformations can be performed and then again we will take the transformed scores as inputs to the corresponding parsing algorithms.

We describe the high-order extension by taking the $o2g$ model as an example and other models can be handled in a similar way. We will use the similar notations: L_{ghm} is the binary variable that indicates the factor with node g as grandparent, node h as head and node m as modifier exists in the parse tree. We continuously use H as the parent of m and G as its grandparent so that L_{GHm} is 1 (representing an existing factor) in the parser tree. The logarithmic probability can be given by the following equation:

$$\begin{aligned}\log(Pr(T|S)) &= \sum_{g,h,m} Pr(L_{ghm}|context(g,h,m))\\ &= \sum_{0<m\le length(S)} \Big(\log\big(Pr(L_{GHm}=1)\big)\\ &\quad + \sum_{\substack{h,g\\ h\ne H, g\ne G\\ h\ne m, g\ne m}} \log\big(Pr(L_{ghm}=0)\big)\Big)\end{aligned}$$

4.3 Neural Network Model

Now we adopt feed-forward neural network to learn and compute the probability for a factor. The inputs for the network are features for a factor such as word forms, POS tags and distance, and the output will be the probability that the factor exists in the parse tree. Figure 3 shows the structure of our neural network for the $o2g$ model, the networks for other models will be similar.

For the architecture of the neural network, as usual, the first layer is the projection layer or the embedding layer, which performs the concatenation for the embeddings. All features are treated equally and mapped to embeddings of the same dimension. So, the embedding or projection matrix $E \in \mathbb{R}^{d\times N}$ in-

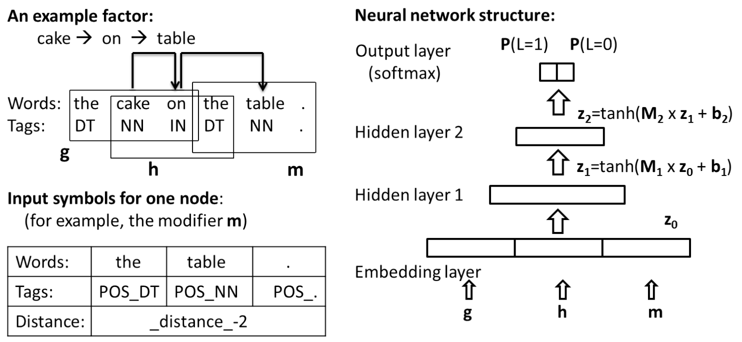

Figure 3: The structure of neural network for *o*2*g* model for an example input factor "*cake* → *on* → *table*". Here we only demonstrate the case of a three-word window.

cludes the embeddings for features, where d is the dimension for the embedding, N is total number of possible features.

For the rest of the network, it can be viewed as a fully-connected feed-forward neural network with two hidden layers and a probabilistic output layer (we use a two-way softmax output unit to compute the probability). For the hidden layers, we use hyperbolic tangent as activation function.

The training objective is to maximize the logarithmic probability of parse trees with an L2-regularization term to avoid over-fitting, which equals to minimizing the cross-entropy loss with L2-regularization:

$$L(\theta) = -\sum_{S} \log\big(Pr(T|S)\big) + \frac{\lambda}{2} \cdot \|\theta\|^2$$

Here θ means parameters of the neural network and λ is the hyper-parameter for weight decay. We initialize all the weights with random values and use mini-batch stochastic gradient descent for training.

4.4 Feature Sets

We utilize three kinds of features:

- Word forms (inside a specified sized window)
- POS tags (for each word)
- Distance (to the node's parent in the factor)

Using embeddings and neural network, we only need to provide unigram features, which will be mapped to embeddings in the neural network. The connections between features will be exploited by the non-linear computations of the neural network. Those three kinds of features are treated in the same way as strings in the vocabulary, and special prefix strings are added to POS and distance features to differ them from word features ("POS_" and "_distance_" respectively).

Features for the head node: w_{h-1}, w_h, w_{h+1}; t_{h-1}, t_h, t_{h+1}; $d_{g,h}$
Features for the modifier node: w_{m-1}, w_m, w_{m+1}; t_{m-1}, t_m, t_{m+1}; $d_{h,m}$
Features for the grandparent node: w_{g-1}, w_g, w_{g+1}; t_{g-1}, t_g, t_{g+1}

Table 1: Features for the *o*2*g* model (with three-word windows). w: words, t: POS tags, d: distance. +1 and -1 means neighboring indexes.

Again, take the situation for *o*2*g* model as an example, there are three nodes in a factor: g for grandparent, h for head and m for modifier. We show the features in Table 1 when considering three-word window, there will be three word forms and three tags for each node, h and m both have one distance feature while g does not have one because its parent is unknown at this time. In fact, larger-sized context can be included and a seven-word window is actually considered for later experiments.

4.5 Integrating Lower-order Models for Higher-order Parsing

Following standard practice for high-order models (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010), we integrate the lower-order scores into the higher order parsing for better performance. For *o*2*sib* and *o*2*g* models in this work, we integrate the scores computed from the first-order model into second order factors. And for *o*3*g* model, two lower-order scores are integrated. Specifically, the score for the factor (g, h, s, m) will include the lower-order scores of $o1$ and $o2sib$ in addition to the third-order score $o3gScore(g, h, s, m)$ from $o3g$ model. The integration of the scores can be shown by the following equation:

$$\begin{aligned} Score(g,h,s,m) &= o1Score(h,m) \\ &+ o2sibScore(h,s,m) \\ &+ o3gScore(g,h,s,m) \end{aligned}$$

More importantly, we may let the first-order model to serve as an edge-filter for high-order parsing. This type of pruning has been used by many graph-based models (Koo and Collins, 2010; Rush and Petrov, 2012) to avoid too expensive operations in high-order parsing. For our model, we utilize our own first-order neural network model which will produce the probabilities for all the edges in the graph. We simply set a pruning threshold so that all edges whose probabilities are under the threshold will be discarded for high-order parsing.

4.6 Efficient Neural Network Computation

This subsection introduces two techniques to speed up neural network computation.

Efficient computation strategies have been explored extensively for neural network language models (Morin and Bengio, 2005; Mnih and Hinton, 2008; Vaswani et al., 2013). These models consider speeding up the output softmax layer which contains thousands of neurons. However, it is not the case for our neural network as the output layer of our network only has two neurons. Main computation cost in our network is from the first hidden layer, which needs matrix multiplications and the hyperbolic tangent activation calculations for the hidden neurons.

Similar to some previous work (Devlin et al., 2014; Chen and Manning, 2014), we apply the pre-calculation strategy to speed up the most concerned computation. This can be implemented as calculating a lookup table for the first hidden layer (values before computing activation function), which can replace the operations of the looking-up for embedding layer and the matrix multiplication for second layer (first hidden layer after). With the pre-calculation table, we only need to look up the corresponding matrix multiplication results for each position's input and add them together to get the values for the first hidden layer.

#Number of sentences			
Corpus	Train	Dev	Test
PTB	39832	1700	2416
CTB	16091	803	1910
#Number of tokens			
Corpus	Train	Dev	Test
PTB	950348	40121	56702
CTB	437990	20454	50315

Table 2: Statistics for the data sets for dependency parsing.

Another technique is to pre-calculate a hyperbolic tangent table, which will replace the computation for the activation function with a table looking-up process.

5 Experiments and Discussions

The proposed parsers are evaluated on English Penn Treebank (PTB3.0) and Chinese Penn Treebank(CTB7.0). For all the results, we report unlabeled attachment scores (UAS) excluding punctuations[3] as in previous work (Koo and Collins, 2010; Zhang and Clark, 2008). In Table 2, we show statistics of both treebanks.

For English, we follow the splitting conventions, using sections 2-21 for training, 22 for developing and 23 for test. We patch the Treebank using Vadas' NP bracketing[4] (Vadas and Curran, 2007) and use the LTH Converter[5] (Johansson and Nugues, 2007) to get the dependency treebank. 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 follow the convention described in (Zhang and Clark, 2008). The dependencies are converted with Penn2Malt tool[6]. As in previous work, we use gold segmentation and POS tags.

For both treebanks, all the graph-based parsers

[3]Punctuations are the tokens whose gold POS tag is one of {“ ” : , .} for PTB and PU for CTB.

[4]http://sydney.edu.au/engineering/it/∼dvadas1

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

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

Initialize	Source	UAS
random	–	91.79
SENNA[7]	Collobert et al. (2011)	91.75
GloVe[8]	Pennington et al. (2014)	91.73
word2vec[9]	Mikolov et al. (2013)	91.81

Table 3: Accuracies for different initializations, with first-order models on dev set.

run on the same machine with Intel Xeon 3.47GHz CPU using single core.

5.1 Different Embedding Initializations

We initialize the embedding matrix (only the parts for the embeddings of words) with some trained word embeddings or word vectors as shown in Table 3. Compared to the random initialization method, using pre-trained embeddings does not bring too significant improvements. We contribute this mostly to already large enough training set. In fact, the number of the training samples fed to the network is over 20 million. Another possible reason is that the embedding initialization only works for word form features and other features such as POS tags and distance will have to be initialized with random values. Those two types of initializations existing in the same space may cause possible inconsistence. Based on the above empirical results and comparison, we will only use random initialization for our parsers.

5.2 Pruning

For high-order models, their full training can be computationally expensive or even impossible, so we must prune unlikely dependencies as we stated before in Section 4.5. We use a simple strategy by setting a fixed probability threshold and the results of different thresholds are shown in Table 4. In this table, the notations are defined as the following:

- W_t = %edges wrongly pruned in training set
- W_d = %edges wrongly pruned in dev set
- *#inst* = number of instances for one iteration
- *Time* = time for one iteration
- *Acc.* = UAS on dev set

With a large threshold, we might prune some correct dependencies, but if the threshold is set smaller, more incorrect dependencies will remain and the training will be more expensive. Even though those wrongly pruned dependencies are allowed, their scores are also too low to influence the inference. A threshold of 0.001 is finally chosen for other experiments in this work.

Threshold	W_t	W_d	*#inst(M)*	*Time(min.)*	*Acc.*
0.01	0.47	1.41	315	29	92.41
0.001	0.13	0.58	764	65	92.47
0.0001	0.02	0.13	2591	220	92.43

Table 4: Effects of pruning methods with different thresholds (on English dev set with the *o*2*sib* model).

5.3 Main Results

As for detailed neural network setting, we use embeddings of 50 dimensions, and the size of the two hidden layers are 200 and 40, respectively. We initialize the learning rate as 0.1. After each iteration, the parser is tested on the development set and if the accuracy decreases, the learning rate will be halved. We train the models for 10 iterations and select the ones that perform best on the development set.

For the inputs, we consider a seven-word window. Notice that only with distributed representations, can we incorporate such very-long-context features. We ignore the words that occur less than 3 times in the training treebank and use a default token to represent unknown words.

Our evaluations will follow the setting in (Chen and Manning, 2014), which reported results of the transition-based neural network parser. For graph-based parsers, in order to get exact comparisons between traditional methods and neural network methods, we run the traditional graph-based parsers under the same executing environment as our parsers. In detail, MSTParser[10] for *o*1 and *o*2*sib* models and MaxParser[11] (Ma and Zhao, 2012) for *o*2*g* and *o*3*g* models are respectively used for comparison. Notice that in recent years, there have been plenty of graph-based parsers which utilize various techniques and obtain state-of-art results (Rush and Petrov, 2012; Zhang and McDonald, 2012), however, they will not be included in the comparisons for the reason that we only concern about basic graph-based parsing al-

[10]http://sourceforge.net/projects/mstparser/

[11]http://sourceforge.net/projects/maxparser/, this is a C++ implementation for several high-order graph-based parsers

Parser	UAS	Root	CM	Speed
$o1$-nn	91.77	96.61	35.89	150
$o2sib$-nn	92.35	96.40	39.86	109
$o2g$-nn	92.18	96.85	38.45	89
$o3g$-nn	92.52	96.81	41.10	38
$o1$-Mst	91.31	95.12	36.67	18
$o2sib$-Mst	91.99	95.90	39.74	14
$o2g$-Max	92.12	96.03	40.11	2
$o3g$-Max	92.60	96.31	42.63	0.3
$transition$	92.0	–	–	1013

Table 5: Results on PTB, the English treebank.

Parser	UAS	Root	CM	Speed
$o1$-nn	83.59	76.86	26.60	112
$o2sib$-nn	86.00	77.59	31.94	70
$o2g$-nn	84.13	77.75	27.59	49
$o3g$-nn	86.01	78.06	31.88	11
$o1$-Mst	83.31	71.57	27.49	9
$o2sib$-Mst	85.34	75.60	32.98	8
$o2g$-Max	84.96	76.32	31.94	1
$o3g$-Max	86.41	78.22	34.82	0.1
$transition$	83.9	–	–	936

Table 6: Results on CTB, the Chinese treebank.

gorithms.

We report three accuracy metrics, UAS, Root (percentage of the root words correctly identified), CM (complete rate, percentage of sentences for which the whole tree is correct) and Speed (number of sentences per second). For Chinese, the UAS and CM both consider root words.

Tables 5 and 6 show the results for PTB and CTB. As for name suffix in the tables, nn means our neural network graph-based parsers, Mst means MstParser, Max means MaxParser, $transition$ means the transition-based neural network parser (Chen and Manning, 2014).

From the results, we can see that our parsers can get similar or even better results compared to the traditional graph-based models of the corresponding orders. In addition, our speed is faster (notice that even our $o3g$ parser is faster than the traditional first-order graph-based parser). Compared to the transition-based neural network parser, although our parsers are not that fast (transition-based parsers usually have O(n) time complexity), they give better performance in accuracies.

5.4 Discussions

We find that integrating lower-order models into high-order parsing leads to better results. Although the high-order factors already include the lower-order parts, it might be hard for the neural network to decide whether the whole factor is correct. During training, we specify a factor as a positive sample only if all the dependencies in it are correct because we only do a binary classification. This might be the limitation for our high-order model and might explain the reason why some of our high-order parsers do not surpass traditional ones in accuracy, we might need more appropriate object functions to improve its learning.

Compared to the features of traditional methods, the only information beyond the proposed feature set is the words that fall out of the windows between the nodes in the factor (previously called in-between features) because so far we only use fixed-size inputs for the feed-forward neural network. Extra operations for embedding vectors (like adding embedding vectors) and other forms of neural networks (such as convolutional neural network which can consider the context of a whole sentence) might be explored in the future.

6 Conclusions

In this paper, we show a way to use neural network for graph-based dependency parsing and the method is also suitable for high-order parsing. We show that using distributed representations for neural network to replace traditional sparse features in traditional graph models can be suitable for dependency parsing, even though only using a feed-forward network. From the evaluation results and comparison with existing models, we show that the proposed parsers get good results with quite efficient inference even though graph-based models usually need at least cubic-time for inference.

References

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

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the*

CoNLL Shared Task Session of EMNLP-CoNLL 2007, pages 957–961, Prague, Czech Republic, June. Association for Computational Linguistics.

Danqi Chen and Christopher 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, Doha, Qatar, October. Association for Computational Linguistics.

Changge Chen, Peilu Wang, and Hai Zhao. 2015. Shallow discourse parsing using constituent parsing tree. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning - Shared Task*, pages 37–41, Beijing, China, July. Association for Computational Linguistics.

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. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

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

Ronan Collobert. 2011. Deep learning for efficient discriminative parsing. In *AISTATS*.

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 *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland, June. Association for Computational Linguistics.

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.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of NODALIDA 2007*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June. Association for Computational Linguistics.

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.

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

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the ACL (EACL 2006)*, pages 81–88. 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 of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of INTERSPEECH-2010*, pages 1045–1048.

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

Andriy Mnih and Geoffrey Hinton. 2008. A scalable hierarchical distributed language model. In *NIPS*.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS05*, pages 246–252.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160, April.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October. Association for Computational Linguistics.

Alexander Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Lin-*

guistics: Human Language Technologies, pages 498–507, Montréal, Canada, June. Association for Computational Linguistics.

Holger Schwenk. 2007. Continuous space language models. *Computer Speech and Language*, 21(3):492–518.

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 the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 1–8, Barcelona, Spain, July. Association for Computational Linguistics.

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 HLT-NAACL*, pages 252–259.

David Vadas and James Curran. 2007. Adding noun phrase structure to the penn treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 240–247, Prague, Czech Republic, June. Association for Computational Linguistics.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proceedings of EMNLP-2013*, pages 1387–1392, Seattle, Washington, USA, October. Association for Computational Linguistics.

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 the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 845–850, Seattle, Washington, USA, October. Association for Computational Linguistics.

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 the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 189–195, Doha, Qatar, October. Association for Computational Linguistics.

Rui Wang, Hai Zhao, Bao-Liang Lu, Masao Utiyama, and Eiichiro Sumita. 2015. Bilingual continuous-space language model growing for statistical machine translation. *IEEE/ACM Transactions on Audio, Speech, and Languange Processing*, 23:1209–1220.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206, April.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea, July. Association for Computational Linguistics.

Jingyi Zhang and Hai Zhao. 2013. Improving function word alignment with frequency and syntactic information. In *IJCAI-2013*, pages 2211–2217, Beijing, China, August.