Graph-based Dependency Parsing with Recursive Neural Network

Pingping Huang , Baobao Chang

Key Laboratory of Computational Linguistics, Ministry of Education
School of Software and Microelectronics
School of Electronics Engineering and Computer Science, Peking University
Collaborative Innovation Center for Language Ability, Xuzhou 221009 China
pinghpp@pku.edu.cn, chbb@pku.edu.cn

Abstract. Graph-based dependency parsing models have achieved state-of-the-art performance, yet their defect in feature representation is obvious: these models enforce strong independence assumptions upon tree components, thus restricting themselves to local, shallow features with limited context information. Besides, they rely heavily on hand-crafted feature templates. In this paper, we extend recursive neural network into dependency parsing. This allows us to efficiently represent the whole sub-tree context and rich structural information for each node. We propose a heuristic search procedure for decoding. Our model can also be used in the reranking framework. With words and pos-tags as the only input features, it gains significant improvement over the baseline models, and shows advantages in capturing long distance dependencies.

Keywords: dependency parsing, recursive neural network, weighted-sum pooling

1 Instruction

Dependency parsing is a fundamental NLP task in which the syntactic structure of a sentence is depicted by the dependency relations between words. It it also wildly used in other applications that rely on syntactic trees.

Current supervised models mainly fall into two fundamentally different categories: transition-based models (see Yamada and Matsumoto, 2003; Nivre et al., 2006) and graph-based models (see Eisner,1996; McDonald et al.,2005). The former resorts to a set of transition actions, thus turning the search for the best tree structure into the search for the optimal choices of transition sequence. By contrast, graph-based models explicitly parametrize the dependencies between words, and search over all possible trees which span the whole sentence for the optimal structure. With carefully designed features and proper search procedures, both of these models achieve state-of-the-art performance.

In the two types of models, graph-based models usually decompose the whole tree structure into small sub-graphs and enforce strong independence assumption among them. When scoring an individual sub-graph, the features are constrained inside this small part. The drawbacks of these models are obvious:

1. The independence assumption is not justified linguistically, because the sentence is an organized whole item, and there are complicated interactions among different sub-graphs.
2. The parsing performance relies heavily on hand-crafted features, which demands much expertise knowledge. Besides, dealing with the enormous amount of features are time-consuming in parsing, and increases the risk of over-fitting.

In face of these drawbacks, we propose a recursive neural network model for graph-based dependency parsing. The main advantages of our model are as follows:

1. Much wider contexts are taken into account. In our model, each node in the dependency tree is represented with the whole sub-tree context into consideration.
2. Our model does not rely on complicated feature templates. Instead, it only needs words and pos-tags as input features, the representation and selection of features are automatically learned during the training.
3. Our model can also be used as a reranking model.

We evaluate our model on the English Penn Treebank. Experiment results show that our models gains significant improvement over the baseline system. Especially when used as a reranking model, the improvement is more exciting. Experiments also demonstrate that our model exhibits advantages in capturing long distance dependencies.

The remaining part of this paper is organized as follows. Section 2 describes the motivation behind our model. Section 3 elaborates on our recursive neural networking model for dependency trees. The parsing algorithm is described in Section 4. In Section 5, we describe how our model can be used as a reranking model. The training method is given in Section 6. Experiments, results and analysis are given in Section 7. Section 8 summarizes related work. Section 9 draws the conclusions.

2 Motivation

In graph-based models, the parsing process is the search for the highest scored tree structure that spans all the words of the sentence and roots at an artificial node ”ROOT”. A fundamental property of graph-based parsing systems is that, for input sentence x, the score of a structure y is assumed to factor through the scores of independent sub-graphs:

$$score(x, y) = \sum_{g \in y} h(g) = \sum_{g \in y} \boldsymbol{w} \cdot \boldsymbol{\psi}_g \tag{1}$$

where g is the decomposed sub-graph in tree y, h is the score function, and is often a liner model as we use it here. $\boldsymbol{\psi}_g$ is the high dimensional feature vector defined on g, and $\boldsymbol{w}$ is the corresponding weight vector.

According to the number of edges each decomposed sub-graph contains, the order of a system is thus defined. The first-order model, which is also called edge-factored model (see Eisner 2000; McDonald et al., 2005), assumes that the score of a tree structure is the sum scores of independent edges. In second-order models, each sub-graph contains a pair of adjacent edges, like the same-side sibling edges (see McDonald and Pereira, 2006), or head-modifier-grandchild edges (see Carreras, 2007; Koo and Collins,

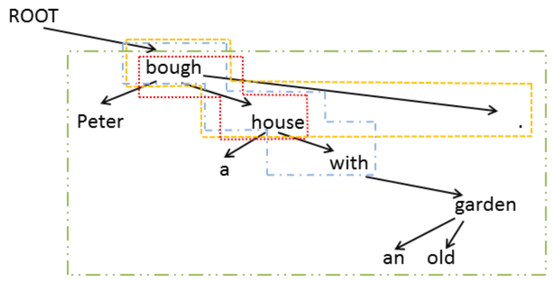

Fig. 1. Illustration of the context ranges for different order models when scoring the dependency between *bought* and *house*. The red block indicates the context for the first-order model, the yellow block the second-order model, and the blue block the third-order model. The green block is the sub-tree context in our model for node *bought*.

2010). In third-order models , the size of sub-graph further expands to certain patterns of three edges, like grand-siblings or tri-siblings structures (see Koo and Collins, 2010; Hayashi et al., 2011). A crucial limitation of these models is that the associated sub-graphs are typically small, losing much of the contextual information and interactions with other components, as is illustrated in Figure 1. When scoring the dependencies for *bought*, even the third-order model can only reaches context in a very limited local region. For a similar sentence: *"Peter bought a house with an old friend."*, the above mentioned context information will be quite insufficient to decide whether the head of *with* should be *bought* or *house*.

Correspondingly, the features that the model can draw on is also limited inside each sub-graph. Commonly used features are like the words and pos-tags of parent and child, the arity of the parent, the distance between them, the sibling's words and pos-tag, and most importantly, the various conjunctions for two of three of these atomic features. The weights for these high dimensional sparse features will be poorly estimated in a linear model due to data sparsity. What's worse, those local and lexical features are incapable to catch various interactions in a wide context.

In this paper, we propose a model that scores the tree according to all its sub-tree structures rooted at each nodes, and this score function will move beyond the limitation of previous models to employ more global and structure context. At the same time, our model learns to select useful features all by itself instead of hand-crafted feature templates.

Different from traditional graph-based models that score on edges, we score the tree as the sum over all the nodes:

$$score(x, y) = \sum_{n \in y} s(n) \tag{2}$$

where n is a node in y, yet this node is not scored in isolation. The score $s(n)$ given by our model judges how likely this whole sub-tree rooted at node n is the right structure for the corresponding spanned words.

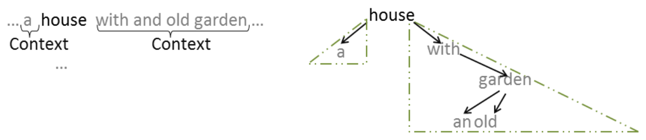

Fig. 2. Equivalence relation between the descendants for node *house* in the dependency tree and its context words in the sentence.

Our assumption is that the syntactic role for a word is decided by two factors: the word's interior meaning and its surrounding context. In a sub-tree structure, the root node serves as the head word for the corresponding span and carries the most important information. Therefore, we need to consider the root node's input as the interior information.

We notice that in a projective dependency tree, descendant nodes for a root is equal to the context that surrounds this word in the sentence, as shown in Figure 2. Therefore we also need to represent the information carried by the descendants nodes in the sub-tree for this root. We have further observed that dependency words before and after a head word usually play different roles. Therefore, we use two features, $leftContext$ and $rightContext$, to represent descendant nodes from this two different directions.

In this way, each root node in the tree contains three parts: context from left descendants, the information carried by the node itself, and the context from right descendants. Compared to traditional models, we incorporates much larger context and more patterns of interactions. Then the question remains: how can we represent and score each node with all these information in consideration?

3 Recursive Neural Network for Dependency Trees

The idea of recursive neural networks (see Socheret al., 2010; Coller and Kuchler, 1996) is to learn hierarchical feature representations by applying the same neural network recursively in a tree structure. But the standard network architecture can only be applied to a fixed-tree structure. We adapt this network for dependency trees and propose the $content\text{-}context$ vectorial representation for each node, this node representation compresses rich context of the whole sub-tree information.

3.1 Node and Context Representation

One of the revolutionary changes coming with the rise of Deep Neural Network(DNN) is the idea of representation learning and automatic feature selection. The most convincing example is word and feature embedding learning that shows great superiority (see Bengio et al., 2003; Mikolov et al., 2010; Mikolov et al.,2013).

Inspired by these success, we use word and pos-tag as the only input information and leave it to the model to learn the vectorial representations, and feature selection and combination. For each node in the tree, its word and pos-tag are indexed through lookup tables into low-dimensional dense valued embeddings. Then we concatenate these two embeddings and get the *content* embedding for a node:

$$content_i = f[I_{word_i} \cdot L_{words}; I_{pos_i} \cdot L_{poses}] \tag{3}$$

where f is the nonlinear transformation function, I_{word} is the one-hot high dimensional vector for word w_i, and I_{pos} is the one-hot vector for the pos-tag p_i. L_{words} and L_{poses} are lookup tables of word and pos-tag embeddings, which are learned during the training. Each child's *content* embedding will be propagated forward into the parent node's *context* embedding. And there will be specific null embeddings to indicate the context for leaf nodes.

3.2 Weighted-sum Pooling

The standard recursive neural networks requires a fixed local structure for each tree node. But for dependency trees, the number of children in each node varies. Thus we need a way to deal with this structural speciality so that all nodes can share the weights when propagating information in the tree.

We introduce weighted sum pooling strategy that assigns a weight to each child node. This weight indicates how important this child is among its siblings, and correlates to the sub-tree size spanned by this node. The more words this sub-structure spans, the more important role it will play in parent's context embedding. The weight w_j for node j is computed as follows:

$$cnt_i = \begin{cases} 1, \text{ if } i \text{ is a leaf} \\ 1 + \sum_{(i,j) \in y} cnt_j, \text{ otherwise} \end{cases} \tag{4}$$

$$w_j = Cnt_j / Cnt_i, \text{ for } (i, j) \in \text{y} \tag{5}$$

That is, the weight of a child node is the ratio of this node's number of spanned words in that of its parent's. With this pooling strategy, we can propagate all the information from arbitrary number of children into parent while sharing one set of parameter.

3.3 Network Architecture

When a tree structure y is given, we propagate the content of children into the context of parent node in a bottom-up direction as follows:

$$context_{left}(i) = f(W1 \cdot \sum_{(i,j) \in y, j<i} w_j \cdot content_j + b_l) \tag{6}$$

$$context_{right}(i) = f(W2 \cdot \sum_{(i,j) \in y, j>i} w_j \cdot content_j + b_r) \tag{7}$$

where w_j is the weight associated with each node, $W1$ and $W2$ are the weight matrix that combine incoming children from two different directions into the parent. f is the non-linear transformation function in the neural network.

Till now, we have filled the $content\text{-}contex$ embeddings for each node. Then we concatenate these vectorial representations as features and directly score on it:

$$s(i) = W3 \cdot [context_{left}(i); content(i); context_{right}(i)] \tag{8}$$

where $W3$ is the weight matrix that is used to score on these representations. The higher the score is, the more likely this given sub-tree structure rooted at the current node is the correct structure.

The network structure and the forward propagation process is shown in Figure 3. As the tree is built in the bottom-up fashion, each node's content will appear in parent's context, and then the grand node and so on. In this way, even the far-most leaf's information will be propagated into the root's context, thus the context information for each node we can use is the whole sub-tree. Once the forward propagation is done we get the score for each node, and the score of a given tree is the sum score of them. In our model,

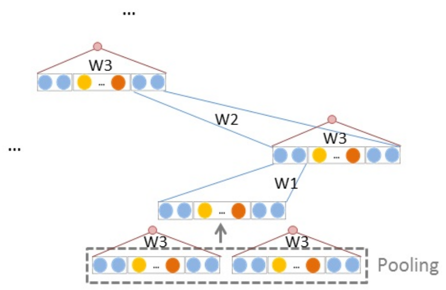

Fig. 3. Model architecture and the propagation process in a dependency tree

the only feature we use is the word and pos-tag for each node. Instead of carefully designing the feature templates to capture certain kinds of interactions among nodes, the model learns to select useful information and interactions by tuning the weight matrix.

4 Parsing

Parsing is the search for the highest scored tree structure that spans all the words in a sentence. As recursive neural network scores each node in a bottom-up way, the Eisner algorithm for first order projective parsing launches the parsing in a similar direction (see Eisner, 2000; McDonald et al., 2005). With the edge independence assumption, this algorithm employs dynamic programming for exact search.

But as we score each node with the whole sub-tree structure in consideration, which breaks the dynamic programming condition and thus the model can only use inexact search. We extends Esiner algorithm with beam search strategy, and use the same chart table and parsing structures as in the original algorithm. The only difference is that in each chart cell, we keep B highest scored candidates instead of just one, and call this list an *agenda*. When combing two chart cells to form a bigger structure, we try all the available combinations coming from this two agendas. When all the chart cells are filled, we get the highest scored tree structure in $E[0][n][1][0]$. This is the target structure coving all words with ”ROOT” as the root node.

But when we take a further look into the chart cell combination process, each trial combination involves several matrix operations and a non-linear transformation to update the newly combined candidate structure's score. This will be time-consuming in practise. Therefore we use the prune heuristic to filter out unlikely combined candidates. We follow the similar strategy in (Socher et al., 2013): we use the k-best list from a baseline system, arcs that never appeared in any of the k-best is pruned directly. Note that though this prune strategy relies on a k-best list, it differs from re-ranking in two main aspects: first each node will just get access to the local information that whether another node is a possible child, instead of the whole tree or any other global information. Second, we can generate tree structures that is not in the k-best list. This prune allows the second pass to be very fast yet still keep the exploration space large enough.

5 Used as a Reranking Model

As our model can give scores of a whole tree structure, it can also be directly used in the reranking phase. The max margin training criterion expects the score of the correct tree to be higher than that for an incorrect tree by a margin, therefore it can discriminate good structure from bad ones. Besides, as the parsing algorithm is heuristic search, reranking eliminates the risk of search errors thus gives a more directly measure to the model's ability to discriminate good structures from bad ones. Therefore we also experiments in the reranking framework, without any extra training once we get the model trained for parsing.

6 Training

For model training, we use the Max-Margin criterion. Given a training instance (x_i, y_i), we search for the dependency tree with the highest score. The structured margin loss between the predicted structure and the given correct tree is defined on the discrepancy between trees. It is measured by counting the number of nodes $N(y_i)$ with incorrect head in the predicted tree:

$$(y_i, \hat{y}_i) = \sum_{j}^{n} \kappa 1\{h(y_i, x_i) \neq h(\hat{y}_i, x_i)\} \tag{9}$$

where n is the length for the sentence x_i. We set the parameter $\kappa = 0.1$ throughout the experiment. The loss increases the more incorrect the proposed parse tree is.

The object of Max-Margin training is that the highest scoring tree is the correct one $y_i^* = y_i$ and its score will be larger up to a margin to other possible tree $\hat{y}_i \in Y(x_i)$:

$$score(x_i, y_i; \theta) \geq score(x_i, \hat{y}_i; \theta) + \mathit{\Delta}(y_i, \hat{y}_i) \tag{10}$$

Then the regularized objective function for m training examples is thus defined:

$$\begin{aligned} &J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(l_i(\theta) + \frac{\lambda}{2}||\theta||), \text{where} \\ &l_i(\theta) = max_{\hat{y}\in Y(x_i)}\left(score\left(x_i, \hat{y}; \theta\right) + \varDelta\left(y_i, \hat{y}_i\right)\right) - score\left(x_i, y_i; \theta\right) \end{aligned} \tag{11}$$

We train the network with Back Propagation Through Structure(BPTS) (see Goller and Kuchler, 1996). As the objective function is not differentiable due to the hinge loss, we follow Socher et al. (2013) to use generalized gradient descent via the sub-gradient method (see Ratliff et al.,2007) which computes a gradient-like direction.

$$\frac{\partial J}{\partial \theta} = \sum_i \frac{\partial s(x_i, \hat{y}_{max})}{\partial \theta} - \frac{\partial s(x_i, y_i)}{\partial \theta} + \lambda\theta \tag{12}$$

where $\hat{y}_i$ is the tree with highest score. We use the diagonal variant of AdaGrad (see Duchi et al., 2011) with minibatchs for optimization.

7 Experiment

7.1 Data and Setup

We experiment on the English Penn Treebank (PTB3.0) and use the head rules of (see Yamada and Matsumoto, 2003) and Penn2malt[1] tool to extract dependency trees. We follow the standard splits of PTB3, using section 2-21 for training, section 22 as development set and 23 as test set. The pos-tag is labelled by The Stanford POS Tagger (see Toutanova et al., 2003). We first tag the development and test data trained on the whole training data. The training itself is tagged with the ten-way jackknifing strategy. The overall pos-tagging accuracy is around 97.2%.

We use our implementation of beam search based arc-standard system to generate the k-best list for pruning and for reranking. The model is trained with the structured perceptron learning with early update strategy (see Collins and Roark, 2004). Rich feature templates are used following (see Huang and Sagae, 2010; Zhang and Nivre, 2011; Huang et al., 2009).

7.2 Parameter Setting

The parameters of our neural network include:

$$\theta = \{W_1, W_2, W3, b_r, b_l, L_{words}, L_{pos}\}$$

We set the dimension of word and context embeddings to be 50, pos tag embedding of dimension 20. We initialize the word embeddings using wor2vec (see Mikolov et at., 2013) by pre-training on Gigaword corpus (see Graff et al., 2003). Pos tag embeddins are random initialized as $L_{pos} \sim U[-1, 1]$. We use $tanh$ as the non-linear transformation function throughout the experiments. Weight matrix takes Xavie initialization (see Glorot and Bengio, 2010): $r = \sqrt{6/(fanIn + fanOut)}$, $W \sim U[-r, r]$ [2]. We fix the regularization parameter $\lambda = 10^{-3}$ for all parameters. The minibatch size was set as 20. We also cross-validated on AdaGrad's learning rate which was eventually set to 0.04.

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

[2] $fanIn$ is the number of node from incoming layer and $fanout$ is the number for the next layer

Table 1. Experiment result of UAS without punctuations.

Model	Dev	Test
OurModel-Parsing	91.56	91.41
OurModel-Rerank	**92.84**	**92.46**
Beam Arc-Standard	90.79	90.63

7.3 Result

As our neural network scorer judges whether a bare tree structure is reasonable, we measure its performance in UAS without punctuation[3]. We examine it in both parsing and in a re-ranking framework. The result is shown in Table 1. We use our implementation, the Beam Arc-Standard as the comparing baseline. Early trial experiments show that the beam size in the training phase has a direct effect on the parsing performance and training speed. To trade off training time and accuracy, we chose a beam size of 60 for model training. But during the parsing, we cross-validated on the development data and used a larger beam size of 200 to explore more search space.

When used in parsing, we achieve the an UAS of 91.56 on the development data, and 91.41 on the test data. We then directly applies this trained model in a reranking framework. The top-60 candidate trees are generated by the our baseline, the Beam Arc-Standard system. Then our model scores each tree and chooses the highest scored candidate as the final output. This yields a more promising result: we get an UAS of 92.84 and 92.46 on development and test set, which is an improvement of 2.1 and 1.83 over the baseline.

7.4 Analysis

Ability to Capture Long Distance Dependencies: Our main motivation for introducing recursive neural network into dependency parsing is to make use of richer context and thus better capturing dependencies that lie beyond traditional models' reach. Therefore we compare our model with the baseline in view of the the ability to discriminate dependency relations at different distances. We plot the comparison result in Figure 4 for both decoding and reranking setup. [4] The smooth dash lines are polynomial trends for each comparison setup.

We can see that our model wins over the baseline in most cases, especially when the distance get larger than a certain threshold. What's more, there is an ascending trend for both decoding and reranking. Especially in the re-ranking setup, our model shows continuous superiority. This shows that our sub-tree based context does capture the long distance better as we expected.

[3] UAS: Unlabelled Attachment Score. Following previous work, we excludes tokens with pos tags of {" " , ; .}

[4] The win-over ratio is defined as: r = (the number of dependencies our model gets right $-$ the number of dependencies the baseline gets right) / total number of dependencies at this distance. $r > 0$ indicates that our model performs better than baseline at this distance, the higher the ratio is, the bigger advantage we gains.

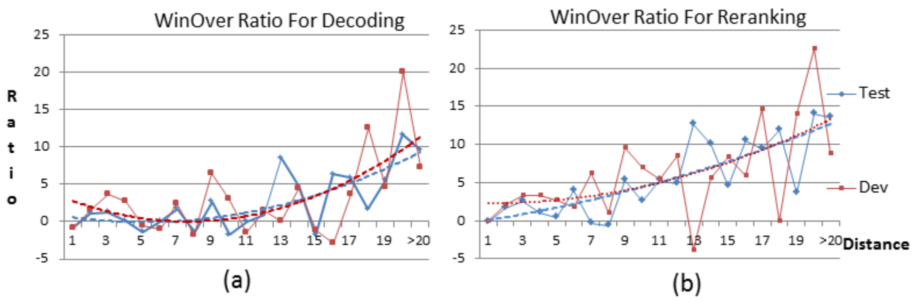

Fig. 4. The win-over ratio of our model versus baseline at different distances. (a) is the result of decoding, (b) is the result at re-ranking. The smoothed lines are binomial trend lines.

Ability of Embedding Learning: To see what kind of information and interactions our model has learned, we probe into the learned embedding representations for pos-tags. Figure 5 is the visualization result.

We can see that, on one hand, our model captures the syntactic similarities among pos-tags quite well. As shown in Figure 5(b)/(c), pos-tags for verbs/nous are clustered closely. On the other hand, these embeddings also capture the relatedness among tags that usually form dependencies, as in Figure 5(b) IN and MD are clustered with verbs. And in Figure 5(c) adjective tags scatters closely among noun tags. This is quiet a desirable representation.

8 Related Work

Models for dependency parsing have been studied with considerable effort in the NLP community. Among them, we only focus on the graph-based models here. Traditional models decompose the tree into small sub-graphs and scores independently. McDonaldet.al (2005) proposed the first-order model which is also know as arc-factored model. Pereira (2006) further extend the first-order model to second-order model where sibling information is available during decoding. Carreras (2007) proposed a more powerful second-order model that can score both sibling and grandchild parts. To exploit more structure information, Koo and Collins (2010) proposed three third-order models that further include grand-siblings and tri-siblings structures into consideration. Compared with these models, our system considers richer more structural context information, and we do not rely on hand-crafted feature templates.

Recently, neural network models have been increasingly focused on. Chen and Manning (2014) uses a neural network to substitute the classifier in a transition-based models with a few atomic features. Recursive neural network for tree parsing was first introduced by Socher et al.(2010) into phrase structure parsing. And was extended for dependency trees in Le and Zuidema(2014), but this generative model can only be used during the reranking, while our model can be used in parsing.

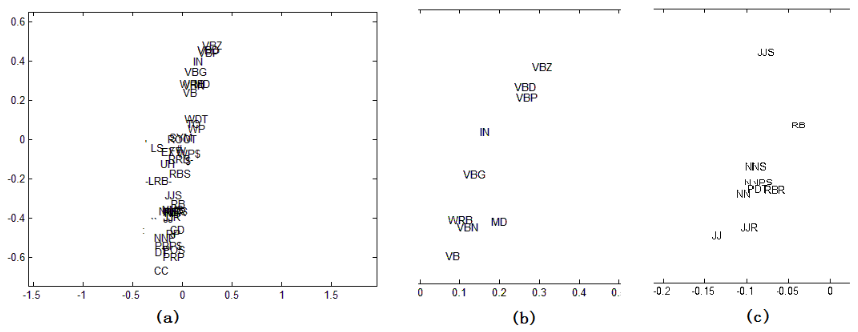

Fig. 5. Visualization of pos embeddings, (b) and (c) are the amplified local for (a).

9 Conclusion

In this paper, we adapts the recursive neural network for dependency trees to employ richer context and more structural information. Each node is represented by a content and two context embeddings so that the sub-tree structure is scored as a whole. We use the weighted sum pooling strategy so that nodes with arbitrary number of children can share one set of parameters to forward-propagate in the tree. Our model can be used both in parsing and in reranking phase. And both achieve competitive results and show an advantage in discriminating long-distance dependencies.

10 Acknowledgments

This work is supported by National Key Basic Research Program of China(2014CB340504) and National Natural Science Foundation of China(61273318).

References

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

Xavier Carreras: Experiments with a Higher-Order Projective Dependency Parser. EMNLP-CoNLL. 957–961 (2007)

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

Michael Collins and Brian Roark: Incremental parsing with the perceptron algorithm. Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics. 111 (2004)

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

Jason M Eisner: Three new probabilistic models for dependency parsing: An exploration. Proceedings of the 16th conference on Computational linguistics. 1, 340–345 (1996)

Jason Eisner: Bilexical grammars and their cubic-time parsing algorithms. Advances in probabilistic and other parsing technologies. 29–61 (2000)

Xavier Glorot and Yoshua Bengio: Understanding the difficulty of training deep feedforward neural networks. International conference on artificial intelligence and statistics. 249–256 (2010)

Christoph Goller and Andreas Kuchler: Learning task-dependent distributed representations by backpropagation through structure. Neural Networks, 1996., IEEE International Conference on. 1, 347–352 (1996)

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto: Third-order variational reranking on packed-shared dependency forests. Proceedings of the Conference on Empirical Methods in Natural Language Processing. 1479–1488 (2011)

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

Liang Huang, Wenbin Jiang, and Qun Liu: Bilingually-constrained (monolingual) shift-reduce parsing. Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing. 3, 1222–1231 (2009)

Terry Koo and Michael Collins: Efficient third-order dependency parsers. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. 1–11 (2010)

Phong Le and Willem Zuidema: The inside-outside recursive neural network model for dependency parsing. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 729–739 (2014)

Ryan T McDonald and Fernando CN Pereira: Online Learning of Approximate Dependency Parsing Algorithms. EACL. (2006)

Ryan McDonald, Koby Crammer, and Fernando Pereira: Online large-margin training of dependency parsers. Proceedings of the 43rd annual meeting on association for computational linguistics. 91–98 (2005)

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur: Recurrent neural network based language model.. INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010. 1045–1048 (2010)

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

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryigit, and Svetoslav Marinov: Labeled pseudo-projective dependency parsing with support vector machines. Proceedings of the Tenth Conference on Computational Natural Language Learning. 221–225 (2006)

Richard Socher, Christopher D Manning, and Andrew Y Ng: Learning continuous phrase representations and syntactic parsing with recursive neural networks. Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop. 1–9 (2010)

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

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

Hiroyasu Yamada and Yuji Matsumoto: Statistical dependency analysis with support vector machines. Proceedings of IWPT. 3, 195–206 (2003)

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