# Graph-based Dependency Parsing with Bidirectional LSTM

**Wenhui Wang**  **Baobao Chang**
Key Laboratory of Computational Linguistics, Ministry of Education.
School of Electronics Engineering and Computer Science, Peking University,
No.5 Yiheyuan Road, Haidian District, Beijing, 100871, China
Collaborative Innovation Center for Language Ability, Xuzhou, 221009, China.
{wangwenhui,chbb}@pku.edu.cn

## Abstract

In this paper, we propose a neural network model for graph-based dependency parsing which utilizes Bidirectional LSTM (BLSTM) to capture richer contextual information instead of using high-order factorization, and enable our model to use much fewer features than previous work. In addition, we propose an effective way to learn sentence segment embedding on sentence-level based on an extra forward LSTM network. Although our model uses only first-order factorization, experiments on English Peen Treebank and Chinese Penn Treebank show that our model could be competitive with previous higher-order graph-based dependency parsing models and state-of-the-art models.

## 1 Introduction

Dependency parsing is a fundamental task for language processing which has been investigated for decades. It has been applied in a wide range of applications such as information extraction and machine translation. Among a variety of dependency parsing models, graph-based models are attractive for their ability of scoring the parsing decisions on a whole-tree basis. Typical graph-based models factor the dependency tree into subgraphs, including single arcs (McDonald et al., 2005), sibling or grandparent arcs (McDonald and Pereira, 2006; Carreras, 2007) or higher-order substructures (Koo and Collins, 2010; Ma and Zhao, 2012) and then score the whole tree by summing scores of the subgraphs. In these models, subgraphs are usually represented as high-dimensional feature vectors which are then fed into a linear model to learn the feature weights.

However, conventional graph-based models heavily rely on feature engineering and their performance is restricted by the design of features. In addition, standard decoding algorithm (Eisner, 2000) only works for the first-order model which limits the scope of feature selection. To incorporate high-order features, Eisner algorithm must be somehow extended or modified, which is usually done at high cost in terms of efficiency. The fourth-order graph-based model (Ma and Zhao, 2012), which seems the highest-order model so far to our knowledge, requires $O(n^5)$ time and $O(n^4)$ space. Due to the high computational cost, high-order models are normally restricted to producing only unlabeled parses to avoid extra cost introduced by inclusion of arc-labels into the parse trees.

To alleviate the burden of feature engineering, Pei et al. (2015) presented an effective neural network model for graph-based dependency parsing. They only use atomic features such as word unigrams and POS tag unigrams and leave the model to automatically learn the feature combinations. However, their model requires many atomic features and still relies on high-order factorization strategy to further improve the accuracy.

Different from previous work, we propose an LSTM-based dependency parsing model in this paper and aim to use LSTM network to capture richer contextual information to support parsing decisions, instead of adopting a high-order factorization. The main advantages of our model are as follows:

- By introducing Bidirectional LSTM, our model shows strong ability to capture potential long range contextual information and exhibits improved accuracy in recovering long distance dependencies. It is different to previous work in which a similar effect is usually achieved by high-order factorization. Moreover, our model also eliminates the need for setting feature selection windows and reduces the number of features to a minimum level.

- We propose an LSTM-based sentence segment embedding method named LSTM-Minus, in which distributed representation of sentence segment is learned by using subtraction between LSTM hidden vectors. Experiment shows this further enhances our model's ability to access to sentence-level information.
- Last but important, our model is a first-order model using standard Eisner algorithm for decoding, the computational cost remains at the lowest level among graph-based models. Our model does not trade-off efficiency for accuracy.

We evaluate our model on the English Penn Treebank and Chinese Penn Treebank, experiments show that our model achieves competitive parsing accuracy compared with conventional high-order models, however, with a much lower computational cost.

## 2 Graph-based dependency parsing

In dependency parsing, syntactic relationships are represented as directed arcs between head words and their modifier words. Each word in a sentence modifies exactly one head, but can have any number of modifiers itself. The whole sentence is rooted at a designated special symbol *ROOT*, thus the dependency graph for a sentence is constrained to be a rooted, directed tree.

For a sentence $x$, graph-based dependency parsing model searches for the highest-scoring tree of $x$:

$$y^*(x) = \arg\max_{\hat{y} \in Y(x)} Score(x, \hat{y}; \theta) \tag{1}$$

Here $y^*(x)$ is the tree with the highest score, $Y(x)$ is the set of all valid dependency trees for $x$ and $Score(x, \hat{y}; \theta)$ measures how likely the tree $\hat{y}$ is the correct analysis of the sentence $x$, $\theta$ are the model parameters. However, the size of $Y(x)$ grows exponentially with respect to the length of the sentence, directly solving equation (1) is impractical.

The common strategy adopted in the graph-based model is to factor the dependency tree $\hat{y}$ into a set of subgraph $c$ which can be scored in isolation, and score the whole tree $\hat{y}$ by summing score of each subgraph:

$$Score(x, \hat{y}; \theta) = \sum_{c \in \hat{y}} ScoreC(x, c; \theta) \tag{2}$$

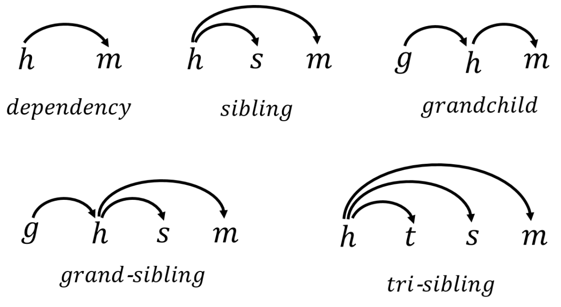

Figure 1: First-order, Second-order and Third-order factorization strategy. Here $h$ stands for head word, $m$ stands for modifier word, $s$ and $t$ stand for the sibling of $m$. $g$ stands for the grandparent of $m$.

Figure 1 shows several factorization strategies. The order of the factorization is defined according to the number of dependencies that subgraph contains. The simplest first-order factorization (McDonald et al., 2005) decomposes a dependency tree into single dependency arcs. Based on the first-order factorization, second-order factorization (McDonald and Pereira, 2006; Carreras, 2007) brings sibling and grandparent information into their model. Third-order factorization (Koo and Collins, 2010) further incorporates richer contextual information by utilizing grand-sibling and tri-sibling parts.

Conventional graph-based models (McDonald et al., 2005; McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012) score subgraph by a linear model, which heavily depends on feature engineering. The neural network model proposed by Pei et al. (2015) alleviates the dependence on feature engineering to a large extent, but not completely. We follow Pei et al. (2015) to score dependency arcs using neural network model. However, different from their work, we introduce a Bidirectional LSTM to capture long range contextual information and an extra forward LSTM to better represent segments of the sentence separated by the head and modifier. These make our model more accurate in recovering long-distance dependencies and further decrease the number of atomic features.

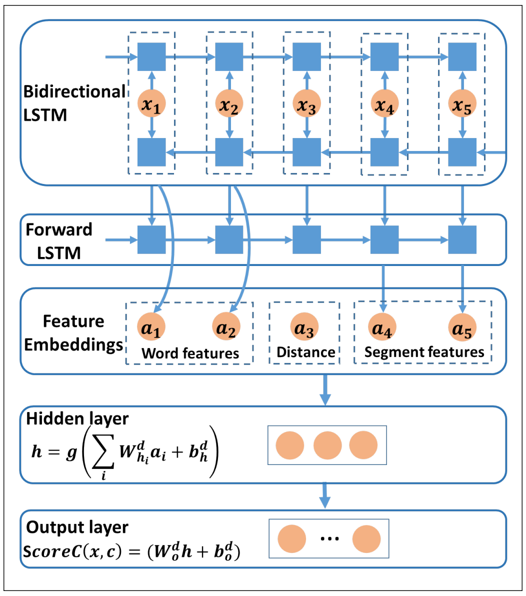

Figure 2: Architecture of the Neural Network. $x_1$ to $x_5$ stand for the input token of Bidirectional LSTM. $a_1$ to $a_5$ stand for the feature embeddings used in our model.

## 3 Neural Network Model

In this section, we describe the architecture of our neural network model in detail, which is summarized in Figure 2.

### 3.1 Input layer

In our neural network model, the words, POS tags are mapped into distributed embeddings. We represent each input token $x_i$ which is the input of Bidirectional LSTM by concatenating POS tag embedding $e_{p_i} \in \mathbb{R}^{d_e}$ and word embedding $e_{w_i} \in \mathbb{R}^{d_e}$, $d_e$ is the the dimensionality of embedding, then a linear transformation $w_e$ is performed and passed though an element-wise activation function $g$:

$$x_i = g(w_e[e_{w_i}; e_{p_i}] + b_e) \tag{3}$$

where $x_i \in \mathbb{R}^{d_e}$, $w_e \in \mathbb{R}^{d_e \times 2d_e}$ is weight matrix, $b_e \in \mathbb{R}^{d_e}$ is bias term. the dimensionality of input token $x_i$ is equal to the dimensionality of word and POS tag embeddings in our experiment, ReLU is used as our activation function $g$.

### 3.2 Bidirectional LSTM

Given an input sequence $x = (x_1, \ldots, x_n)$, where $n$ stands for the number of words in a sentence, a standard LSTM recurrent network computes the hidden vector sequence $h = (h_1, \ldots, h_n)$ in one direction.

Bidirectional LSTM processes the data in both directions with two separate hidden layers, which are then fed to the same output layer. It computes the forward hidden sequence $\overrightarrow{h}$, the backward hidden sequence $\overleftarrow{h}$ and the output sequence $v$ by iterating the forward layer from $t = 1$ to $n$, the backward layer from $t = n$ to 1 and then updating the output layer:

$$v_t = \overrightarrow{h}_t + \overleftarrow{h}_t \tag{4}$$

where $v_t \in \mathbb{R}^{d_l}$ is the output vector of Bidirectional LSTM for input $x_t$, $\overrightarrow{h}_t \in \mathbb{R}^{d_l}$, $\overleftarrow{h}_t \in \mathbb{R}^{d_l}$, $d_l$ is the dimensionality of LSTM hidden vector. We simply add the forward hidden vector $\overrightarrow{h}_t$ and the backward hidden vector $\overleftarrow{h}_t$ together, which gets similar experiment result as concatenating them together with a faster speed.

The output vectors of Bidirectional LSTM are used as word feature embeddings. In addition, they are also fed into a forward LSTM network to learn segment embedding.

### 3.3 Segment Embedding

Contextual information of word pairs[1] has been widely utilized in previous work (McDonald et al., 2005; McDonald and Pereira, 2006; Pei et al., 2015). For a dependency pair $(h, m)$, previous work divides a sentence into three parts (*prefix*, *infix* and *suffix*) by head word $h$ and modifier word $m$. These parts which we call *segments* in our work make up the context of the dependency pair $(h, m)$.

Due to the problem of data sparseness, conventional graph-based models can only capture contextual information of word pairs by using bigrams or tri-grams features. Unlike conventional models, Pei et al. (2015) use distributed representations obtained by averaging word embeddings in segments to represent contextual information of the word pair, which could capture richer syntactic and semantic information. However, their method is restricted to segment-level since their segment embedding only consider the word information within the segment. Besides, averaging operation simply treats all the words in segment equally. However, some words might carry more

[1]A word pair is limited to the dependency pair $(h, m)$ in our work since we use only first-order factorization. In previous work, word pair could be any pair with particular relation (e.g., sibling pair $(s, m)$ in Figure 1).

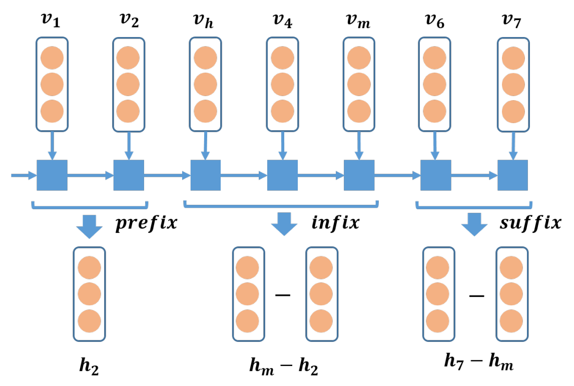

Figure 3: Illustration for learning segment embeddings based on an extra forward LSTM network, $v_h$, $v_m$ and $v_1$ to $v_7$ indicate the output vectors of Bidirectional LSTM for head word $h$, modifier word $m$ and other words in sentence, $h_h$, $h_m$ and $h_1$ to $h_7$ indicate the hidden vectors of the forward LSTM corresponding to $v_h$, $v_m$ and $v_1$ to $v_7$.

salient syntactic or semantic information and they are expected to be given more attention.

A useful property of forward LSTM is that it could keep previous useful information in their memory cell by exploiting input, output and forget gates to decide how to utilize and update the memory of previous information. Given an input sequence $v = (v_1, \ldots, v_n)$, previous work (Sutskever et al., 2014; Vinyals et al., 2014) often uses the last hidden vector $h_n$ of the forward LSTM to represent the whole sequence. Each hidden vector $h_t$ $(1 \le t \le n)$ can capture useful information before and including $v_t$.

Inspired by this, we propose a method named **LSTM-Minus** to learn segment embedding. We utilize subtraction between LSTM hidden vectors to represent segment's information. As illustrated in Figure 3, the segment *infix* can be described as $h_m - h_2$, $h_m$ and $h_2$ are hidden vector of the forward LSTM network. The segment embedding of *suffix* can also be obtained by subtraction between the last LSTM hidden vector of the sequence ($h_7$) and the last LSTM hidden vector in *infix* ($h_m$). For *prefix*, we directly use the last LSTM hidden vector in *prefix* to represent it, which equals to subtract a zero embedding. When no *prefix* or *suffix* exists, the corresponding embedding is set to zero.

Specifically, we place an extra forward LSTM layer on top of the Bidirectional LSTM layer and learn segment embeddings using LSTM-Minus based on this forward LSTM. LSTM-minus enables our model to learn segment embeddings from information both outside and inside the segments and thus enhances our model's ability to access to sentence-level information.

### 3.4 Hidden layer and output layer

As illustrated in Figure 2, we map all the feature embeddings to a hidden layer. Following Pei et al. (2015), we use direction-specific transformation to model edge direction and tanh-cube as our activation function:

$$h = g\Big(\sum_i W_{h_i}^d a_i + b_h^d\Big) \qquad (5)$$

where $a_i \in \mathbb{R}^{d_{a_i}}$ is the feature embedding, $d_{a_i}$ indicates the dimensionality of feature embedding $a_i$, $W_{h_i}^d \in \mathbb{R}^{d_h \times d_{a_i}}$ is weight matrices which corresponding to $a_i$, $d_h$ indicates the dimensionality of hidden layer vector, $b_h^d \in \mathbb{R}^{d_h}$ is bias term. $W_{h_i}^d$ and $b_h^d$ are bound with index $d \in \{0, 1\}$ which indicates the direction between head and modifier.

A output layer is finally added on the top of the hidden layer for scoring dependency arcs:

$$ScoreC(x, c) = W_o^d h + b_o^d \qquad (6)$$

Where $W_o^d \in \mathbb{R}^{L \times d_h}$ is weight matrices, $b_o^d \in \mathbb{R}^L$ is bias term, $ScoreC(x, c) \in \mathbb{R}^L$ is the output vector, $L$ is the number of dependency types. Each dimension of the output vector is the score for each kind of dependency type of head-modifier pair.

### 3.5 Features in our model

Previous neural network models (Pei et al., 2015; Pei et al., 2014; Zheng et al., 2013) normally set context window around a word and extract atomic features within the window to represent the contextual information. However, context window limits their ability in detecting long-distance information. Simply increasing the context window size to get more contextual information puts their model in the risk of overfitting and heavily slows down the speed.

Unlike previous work, we apply Bidirectional LSTM to capture long range contextual information and eliminate the need for context windows, avoiding the limit of the window-based feature selection approach. Compared with Pei et al. (2015), the cancellation of the context window allows our model to use much fewer features. Moreover, by combining a word's atomic features (word form and POS tag) together, our model further decreases the number of features.

|  |  |
|---|---|
| Pei et al. (2015) | $h_{-2}.w$, $h_{-1}.w$, $h.w$, $h_1.w$, $h_2.w$<br>$h_{-2}.p$, $h_{-1}.p$, $h.p$, $h_1.p$, $h_2.p$<br>$m_{-2}.w$, $m_{-1}.w$, $m.w$, $m_1.w$, $m_2.w$<br>$m_{-2}.p$, $m_{-1}.p$, $m.p$, $m_1.p$, $m_2.p$<br>*dis(h, m)* |
| Our basic model | $v_h$, $v_m$<br>*dis(h, m)* |

Table 1: Atomic features in our basic model and Pei's 1st-order atomic model. $w$ is short for word and $p$ for POS tag. $h$ indicates head and $m$ indicates modifier. The subscript represents the relative position to the center word. $dis(h, m)$ is the distance between head and modifier. $v_h$ and $v_m$ indicate the outputs of Bidirectional LSTM for head word and modifier word.

Table 1 lists the atomic features used in 1st-order atomic model of Pei et al. (2015) and atomic features used in our basic model. Our basic model only uses the outputs of Bidirectional LSTM for head word and modifier word, and the distance between them as features. Distance features are encoded as randomly initialized embeddings. As we can see, our basic model reduces the number of atomic features to a minimum level, making our model run with a faster speed. Based on our basic model, we incorporate additional segment information (*prefix*, *infix* and *suffix*), which further improves the effect of our model.

## 4 Neural Training

In this section, we provide details about training the neural network.

### 4.1 Max-Margin Training

We use the Max-Margin criterion to train our model. Given a training instance $(x^{(i)}, y^{(i)})$, we use $Y(x^{(i)})$ to denote the set of all possible dependency trees and $y^{(i)}$ is the correct dependency tree for sentence $x^{(i)}$. The goal of Max Margin training is to find parameters $\theta$ such that the difference in score of the correct tree $y^{(i)}$ from an incorrect tree $\hat{y} \in Y(x^{(i)})$ is at least $\triangle(y^{(i)}, \hat{y})$.

$$Score(x^{(i)},y^{(i)};\theta) \geq Score(x^{(i)},\hat{y};\theta) + \triangle(y^{(i)},\hat{y})$$

The structured margin loss $\triangle(y^{(i)}, \hat{y})$ is defined as:

$$\triangle(y^{(i)}, \hat{y}) = \sum_{j}^{n} \kappa \mathbf{1}\{h(y^{(i)}, x_j^{(i)}) \neq h(\hat{y}, x_j^{(i)})\}$$

where $n$ is the length of sentence $x$, $h(y^{(i)}, x_j^{(i)})$ is the head (with type) for the $j$-th word of $x^{(i)}$ in tree $y^{(i)}$ and $\kappa$ is a discount parameter. The loss is proportional to the number of word with an incorrect head and edge type in the proposed tree.

Given a training set with size $m$, The regularized objective function is the loss function $J(\theta)$ including a $l_2$-norm term:

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} l_i(\theta) + \frac{\lambda}{2}||\theta||^2 \\ l_i(\theta) &= \max_{\hat{y} \in Y(x^{(i)})} (Score(x^{(i)},\hat{y};\theta) + \triangle(y^{(i)},\hat{y})) \\ &\quad - Score(x^{(i)},y^{(i)};\theta) \end{aligned} \tag{7}$$

By minimizing this objective, the score of the correct tree is increased and score of the highest scoring incorrect tree is decreased.

### 4.2 Optimization Algorithm

Parameter optimization is performed with the diagonal variant of AdaGrad (Duchi et al., 2011) with minibatchs (batch size = 20) . The parameter update for the $i$-th parameter $\theta_{t,i}$ at time step $t$ is as follows:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i} \tag{8}$$

where $\alpha$ is the initial learning rate ($\alpha = 0.2$ in our experiment) and $g_\tau \in \mathbb{R}^{|\theta_i|}$ is the subgradient at time step $\tau$ for parameter $\theta_i$.

To mitigate overfitting, dropout (Hinton et al., 2012) is used to regularize our model. we apply dropout on the hidden layer with 0.2 rate.

### 4.3 Model Initialization&Hyperparameters

The following hyper-parameters are used in all experiments: word embedding size = 100, POS tag embedding size = 100, hidden layer size = 200, LSTM hidden vector size = 100, Bidirectional LSTM layers = 2, regularization parameter $\lambda = 10^{-4}$.

We initialized the parameters using pretrained word embeddings. Following Dyer et al. (2015), we use a variant of the skip n-gram model introduced by Ling et al. (2015) on Gigaword corpus (Graff et al., 2003). We also experimented with randomly initialized embeddings, where embeddings are uniformly sampled from range $[-0.3, 0.3]$. All other parameters are uniformly sampled from range $[-0.05, 0.05]$.

|  | Models | UAS | LAS | Speed(sent/s) |
|---|---|---|---|---|
| First-order | MSTParser | 91.60 | 90.39 | 20 |
|  | 1st-order atomic (Pei et al., 2015) | 92.14 | 90.92 | 55 |
|  | 1st-order phrase (Pei et al., 2015) | 92.59 | 91.37 | 26 |
|  | **Our basic model** | 93.09 | 92.03 | **61** |
|  | **Our basic model + segment** | **93.51** | **92.45** | 26 |
| Second-order | MSTParser | 92.30 | 91.06 | 14 |
|  | 2nd-order phrase (Pei et al., 2015) | 93.29 | 92.13 | 10 |
| Third-order | (Koo and Collins, 2010) | 93.04 | N/A | N/A |
| Fourth-order | (Ma and Zhao, 2012) | 93.4 | N/A | N/A |
| Unlimited-order | (Zhang and McDonald, 2012) | 93.06 | 91.86 | N/A |
|  | (Zhang et al., 2013) | 93.50 | 92.41 | N/A |
|  | **(Zhang and McDonald, 2014)** | **93.57** | **92.48** | N/A |

Table 2: Comparison with previous graph-based models on Penn-YM.

## 5 Experiments

In this section, we present our experimental setup and the main result of our work.

### 5.1 Experiments Setup

We conduct our experiments on the English Penn Treebank (PTB) and the Chinese Penn Treebank (CTB) datasets.

For English, we follow the standard splits of PTB3. Using section 2-21 for training, section 22 as development set and 23 as test set. We conduct experiments on two different constituency-to-dependency-converted Penn Treebank data sets. The first one, **Penn-YM**, was created by the Penn2Malt tool[2] based on Yamada and Matsumoto (2003) head rules. The second one, **Penn-SD**, use Stanford Basic Dependencies (Marneffe et al., 2006) and was converted by version 3.3.0[3] of the Stanford parser. The Stanford POS Tagger (Toutanova et al., 2003) with ten-way jackknifing of the training data is used for assigning POS tags (accuracy $\approx$ 97.2%).

For Chinese, we adopt the same split of CTB5 as described in (Zhang and Clark, 2008). Following (Zhang and Clark, 2008; Dyer et al., 2015; Chen and Manning, 2014), we use gold segmentation and POS tags for the input.

### 5.2 Experiments Results

We first make comparisons with previous graph-based models of different orders as shown in Table 2. We use MSTParser [4] for conventional first-order model (McDonald et al., 2005) and second-order model (McDonald and Pereira, 2006). We also include the results of conventional high-order models (Koo and Collins, 2010; Ma and Zhao, 2012; Zhang and McDonald, 2012; Zhang et al., 2013; Zhang and McDonald, 2014) and the neural network model of Pei et al. (2015). Different from typical high-order models (Koo and Collins, 2010; Ma and Zhao, 2012), which need to extend their decoding algorithm to score new types of higher-order dependencies. Zhang and McDonald (2012) generalized the Eisner algorithm to handle arbitrary features over higher-order dependencies and controlled complexity via approximate decoding with cube pruning. They further improve their work by using perceptron update strategies for inexact hypergraph search (Zhang et al., 2013) and forcing inference to maintain both label and structural ambiguity through a secondary beam (Zhang and McDonald, 2014).

Following previous work, UAS (unlabeled attachment scores) and LAS (labeled attachment scores) are calculated by excluding punctuation[5]. The parsing speeds are measured on a workstation with Intel Xeon 3.4GHz CPU and 32GB RAM which is same to Pei et al. (2015). We measure the parsing speeds of Pei et al. (2015) according to their codes[6] and parameters.

On accuracy, as shown in table 2, our

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

[3]`http://nlp.stanford.edu/software/lex-parser.shtml`

[4]`http://sourceforge.net/projects/mstparser`

[5]Following previous work, a token is a punctuation if its POS tag is {“ ” : , .}

[6]`https://github.com/Williammed/DeepParser`

| Method | Penn-YM | | Penn-SD | | CTB5 | |
|---|---|---|---|---|---|---|
| | UAS | LAS | UAS | LAS | UAS | LAS |
| (Zhang and Nivre, 2011) | 92.9 | 91.8 | - | - | 86.0 | 84.4 |
| (Bernd Bohnet, 2012) | 93.39 | 92.38 | - | - | 87.5 | 85.9 |
| (Zhang and McDonald, 2014) | **93.57** | **92.48** | 93.01 | 90.64 | **87.96** | **86.34** |
| (Dyer et al., 2015) | - | - | 93.1 | 90.9 | 87.2 | 85.7 |
| (Weiss et al., 2015) | - | - | 93.99 | **92.05** | - | - |
| **Our basic model + segment** | 93.51 | 92.45 | **94.08** | 91.82 | 87.55 | 86.23 |

Table 3: Comparison with previous state-of-the-art models on Penn-YM, Penn-SD and CTB5.

basic model outperforms previous first-order graph-based models by a substantial margin, even outperforms Zhang and McDonald (2012)'s unlimited-order model. Moreover, incorporating segment information further improves our model's accuracy, which shows that segment embeddings do capture richer contextual information. By using segment embeddings, our improved model could be comparable to high-order graph-based models[7].

With regard to parsing speed, our model also shows advantage of efficiency. Our model uses only first-order factorization and requires $O(n^3)$ time to decode. Third-order model requires $O(n^4)$ time and fourth-order model requires $O(n^5)$ time. By using approximate decoding, the unlimited-order model of Zhang and McDonald (2012) requires $O(k \cdot log(k) \cdot n^3)$ time, where $k$ is the beam size. The computational cost of our model is the lowest among graph-based models. Moreover, although using LSTM requires much computational cost. However, compared with Pei's 1st-order model, our model decreases the number of atomic features from 21 to 3, this allows our model to require a much smaller matrix computation in the scoring model, which cancels out the extra computation cost introduced by the LSTM computation. Our basic model is the fastest among first-order and second-order models. Incorporating segment information slows down the parsing speed while it is still slightly faster than conventional first-order model. To compare with conventional high-order models on practical parsing speed, we can make an indirect comparison according to Zhang and McDonald (2012). Conventional first-order model is about 10 times faster than Zhang and McDonald (2012)'s unlimited-order model and about 40 times faster than conventional third-order model, while our model is faster than conventional first-order model. Our model should be much faster than conventional high-order models.

[7]Note that our model can't be strictly comparable with third-order model (Koo and Collins, 2010) and fourth-order model (Ma and Zhao, 2012) since they are unlabeled model. However, our model is comparable with all the three unlimited-order models presented in (Zhang and McDonald, 2012), (Zhang et al., 2013) and (Zhang and McDonald, 2014), since they all are labeled models as ours.

| Method | Peen-YM | Peen-SD | CTB5 |
|---|---|---|---|
| Average | 93.23 | 93.83 | 87.24 |
| **LSTM-Minus** | **93.51** | **94.08** | **87.55** |

Table 4: Model performance of different way to learn segment embeddings.

We further compare our model with previous state-of-the-art systems for English and Chinese. Table 3 lists the performances of our model as well as previous state-of-the-art systems on on Penn-YM, Penn-SD and CTB5. We compare to conventional state-of-the-art graph-based model (Zhang and McDonald, 2014), conventional state-of-the-art transition-based model using beam search (Zhang and Nivre, 2011), transition-based model combining graph-based approach (Bernd Bohnet, 2012) , transition-based neural network model using stack LSTM (Dyer et al., 2015) and transition-based neural network model using beam search (Weiss et al., 2015). Overall, our model achieves competitive accuracy on all three datasets. Although our model is slightly lower in accuarcy than unlimited-order double beam model (Zhang and McDonald, 2014) on Penn-YM and CTB5, our model outperforms their model on Penn-SD. It seems that our model performs better on data sets with larger label sets, given the number of labels used in Penn-SD data set is almost four times more than Penn-YM and CTB5 data sets.

To show the effectiveness of our segment embedding method LSTM-Minus, we compare with averaging method proposed by Pei et al. (2015). We get segment embeddings by averaging the output vectors of Bidirectional LSTM in segments.

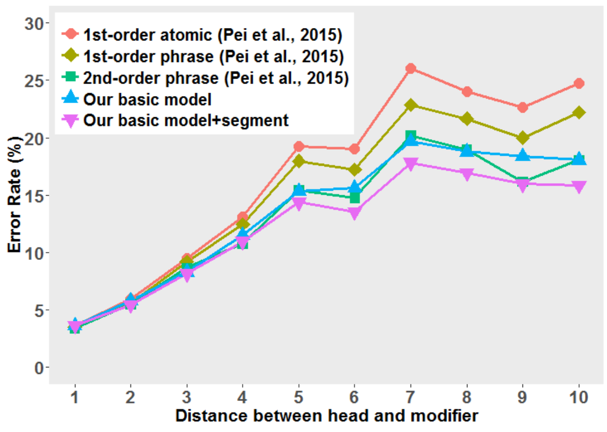

Figure 4: Error rates of different distance between head and modifier on Peen-YM.

To make comparison as fair as possible, we let two models have almost the same number parameters. Table 4 lists the UAS of two methods on test set. As we can see, LSTM-Minus shows better performance because our method further incorporates more sentence-level information into our model.

### 5.3 Impact of Network Structure

In this part, we investigate the impact of the components of our approach.

#### LSTM Recurrent Network

To evaluate the impact of LSTM, we make error analysis on Penn-YM. We compare our model with Pei et al. (2015) on error rates of different distance between head and modifier.

As we can see, the five models do not show much difference for short dependencies whose distance less than three. For long dependencies, both our two models show better performance compared with the 1st-order model of Pei et al. (2015), which proves that LSTM can effectively capture long-distance dependencies. Moreover, our models and Pei's 2nd-order phrase model both improve accuracy on long dependencies compared with Pei's 1st-order model, which is in line with our expectations. Using LSTM shows the same effect as high-order factorization strategy. Compared with 2nd-order phrase model of Pei et al. (2015), our basic model occasionally performs worse in recovering long distant dependencies. However, this should not be a surprise since higher order models are also motivated to recover long-distance dependencies. Nevertheless, with the introduction of LSTM-minus segment embeddings, our model consistently outperforms the 2nd-order phrase model of Pei et al. (2015) in accuracies of all long dependencies. We carried out significance test on the difference between our and Pei's models. Our basic model performs significantly better than all 1st-order models of Pei et al. (2015) (t-test with $p<0.001$) and our basic+segment model (still a 1st-order model) performs significantly better than their 2nd-order phrase model (t-test with $p<0.001$) in recovering long-distance dependencies.

#### Initialization of pre-trained word embeddings

We further analyze the influence of using pre-trained word embeddings for initialization. without using pretrained word embeddings, our improved model achieves 92.94% UAS / 91.83% LAS on Penn-YM, 93.46% UAS / 91.19% LAS on Penn-SD and 86.5% UAS / 85.0% LAS on CTB5. Using pre-trained word embeddings can obtain around 0.5%∼1.0% improvement.

## 6 Related work

Dependency parsing has gained widespread interest in the computational linguistics community. There are a lot of approaches to solve it. Among them, we will mainly focus on graph-based dependency parsing model here. Dependency tree factorization and decoding algorithm are necessary for graph-based models. McDonald et al. (2005) proposed the first-order model which decomposes a dependency tree into its individual edges and use a effective dynamic programming algorithm (Eisner, 2000) to decode. Based on first-order model, higher-order models(McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010; Ma and Zhao, 2012) factor a dependency tree into a set of high-order dependencies which bring interactions between head, modifier, siblings and (or) grandparent into their model. However, for above models, scoring new types of higher-order dependencies requires extensions of the underlying decoding algorithm, which also requires higher computational cost. Unlike above models, unlimited-order models (Zhang and McDonald, 2012; Zhang et al., 2013; Zhang and McDonald, 2014) could handle arbitrary features over higher-order dependencies by generalizing the Eisner algorithm.

In contrast to conventional methods, neural network model shows their ability to reduce the effort in feature engineering. Pei et al. (2015) proposed a model to automatically learn high-order feature combinations via a novel activation function, allowing their model to use a set of atomic features instead of millions of hand-crafted features.

Different from previous work, which is sensitive to local state and accesses to larger context by higher-order factorization. Our model makes parsing decisions on a global perspective with first-order factorization, avoiding the expensive computational cost introduced by high-order factorization.

LSTM network is heavily utilized in our model. LSTM network has already been explored in transition-based dependency parsing. Dyer et al. (2015) presented stack LSTMs with push and pop operations and used them to implement a state-of-the-art transition-based dependency parser. Ballesteros et al. (2015) replaced lookup-based word representations with character-based representations obtained by Bidirectional LSTM in the continuous-state parser of Dyer et al. (2015), which was proved experimentally to be useful for morphologically rich languages.

## 7 Conclusion

In this paper, we propose an LSTM-based neural network model for graph-based dependency parsing. Utilizing Bidirectional LSTM and segment embeddings learned by LSTM-Minus allows our model access to sentence-level information, making our model more accurate in recovering long-distance dependencies with only first-order factorization. Experiments on PTB and CTB show that our model could be competitive with conventional high-order models with a faster speed.

## Acknowledgments

This work is supported by National Key Basic Research Program of China under Grant No.2014CB340504 and National Natural Science Foundation of China under Grant No.61273318. The Corresponding author of this paper is Baobao Chang.

## References

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 349–359.

Jonas Kuhn Bernd Bohnet. 2012. The best of both worlds: a graph-based completion model for transition-based parsers. *Conference of the European Chapter of the Association for Computational Linguistics.*

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *EMNLP-CoNLL*, pages 957–961.

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

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

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 334–343.

Jason Eisner. 2000. *Bilexical Grammars and their Cubic-Time Parsing Algorithms*. Springer Netherlands.

David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2003. English gigaword. *Linguistic Data Consortium, Philadelphia.*

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*

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. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.*

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Posters, 8-15 December 2012, Mumbai, India*, pages 785–796.

Marie Catherine De Marneffe, Bill Maccartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. *Lrec*, pages 449–454.

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 293–303.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 313–322.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing System*, pages 3104–3112.

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. Association for Computational Linguistics.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2014. Grammar as a foreign language. *CoRR*, abs/1412.7449.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*.

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

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

Hao Zhang and Ryan T. 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.

Hao Zhang and Ryan T. McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 656–661.

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

Hao Zhang, Liang Huang Kai Zhao, and Ryan Mcdonald. 2013. Online learning for inexact hypergraph search. *Proceedings of Emnlp*.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 647–657, October.