Incremental Graph-based Neural Dependency Parsing

Xiaoqing Zheng
School of Computer Science, Fudan University, Shanghai, China
Shanghai Key Laboratory of Intelligent Information Processing
zhengxq@fudan.edu.cn

Abstract

Very recently, some studies on neural dependency parsers have shown advantage over the traditional ones on a wide variety of languages. However, for graph-based neural dependency parsing systems, they either count on the long-term memory and attention mechanism to implicitly capture the high-order features or give up the global exhaustive inference algorithms in order to harness the features over a rich history of parsing decisions. The former might miss out the important features for specific headword predictions without the help of the explicit structural information, and the latter may suffer from the error propagation as false early structural constraints are used to create features when making future predictions. We explore the feasibility of explicitly taking high-order features into account while remaining the main advantage of global inference and learning for graph-based parsing. The proposed parser first forms an initial parse tree by head-modifier predictions based on the first-order factorization. High-order features (such as grandparent, sibling, and uncle) then can be defined over the initial tree, and used to refine the parse tree in an iterative fashion. Experimental results showed that our model (called INDP) archived competitive performance to existing benchmark parsers on both English and Chinese datasets.

1 Introduction and Motivation

The rise of machine learning methods in natural language processing (NLP) coupled with the availability of treebanks (Buchholz and Marsi, 2006) for a wide variety of languages has led to a rapid increase in research on data-driven dependency parsing. Two predominant paradigms for the data-driven dependency parsing are often called graph-based and transition-based dependency parsing (McDonald and Nivre, 2007, 2011). The first category learns the parameters to score correct dependency subgraphs over incorrect ones, typically by factoring the graphs into their component directed arcs, and performs parsing by searching the highest-scoring graph for a given sentence. The second category of parsing systems instead learns to predict one transition from one parse state to the next given a parse history, and performs parsing by taking the predicted transitions at each parse state until a complete dependency graph is derived.

Empirical studies show that the graph-based and transition-based models exhibit no statistically significant difference in accuracy on a variety of languages, although they are very different theoretically (McDonald and Nivre, 2011). Graph-based models are usually trained by maximizing the difference in score between the entire correct dependency graph and all incorrect ones for every training sentence. However, exhaustive inference is generally NP-hard when the score is factored over any extended scope of the dependency subgraph beyond a single arc (McDonald and Satta, 2007), which is the primary shortcoming of the graph-based systems. In transition-based parsing, the feature representations are not restricted to a small number of arcs in the graph but can be derived from all the dependency subgraphs built so far, while the main disadvantage of these models is that the local greedy parsing strategy may lead to the error propagation because false early predictions can eliminate valid parse trees.

With a few exceptions (Zeman and Žabokrtskỳ, 2005; Zhang and Clark, 2008; Zhang et al., 2014), the graph-based parsers usually require global

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1654–1664
Copenhagen, Denmark, September 7–11, 2017.

learning and inference, but define features over a limited scope of the dependency graph, while the transition-based ones typically use local, greedy training and inference, but introduce a rich feature space based on the history of parsing decisions.

Many approaches have been proposed to overcome the weaknesses of traditional graph-based or transition-based models. There are at least three ways for potential improvement: ensemble—weighting the predictions of multiple parsing systems (Sagae and Lavie, 2006; Hall et al., 2007), feature integration—combining the two models by allowing the output of one model to define features for the other (Martins et al., 2008; Nivre and McDonald, 2008; McDonald and Nivre, 2011), and novel approaches—changing the underlying model structure directly by constructing globally trained transition-based parsers (Zhang and Clark, 2008; Huang and Sagae, 2010) or graph-based parsers with rich features (Riedel and Clarke, 2006; Nakagawa, 2007; Smith and Eisner, 2008; Martins et al., 2009).

Very recently, some studies on the deep architectures have shown advantage over the shallow ones on a wide variety of dependency parsing benchmarks. Deep neural networks were used to replace the classifiers for predicting optimal transitions in transition-based parers (Chen and Manning, 2014) or the scoring functions for ranking the subgraphs in graph-based rivals (Kiperwasser and Goldberg, 2016a,b). There are several recent developments in neural dependency parsing (Weiss et al., 2015; Zhou et al., 2015; Dyer et al., 2015), which can be viewed as targeting the weaknesses of locally greedy algorithms in transition-based models by using the beam search and conditional random field loss objective, although using the beam search instead of strictly deterministic parsing can to some extent alleviate the error propagation problem but does not eliminate it.

For graph-based neural dependency parsing systems, they either count on the long-term memory and neural attention to implicitly capture the high-order features (Kiperwasser and Goldberg, 2016b; Cheng et al., 2016; Dozat and Manning, 2017) or give up the global inference algorithms in order to introduce features over a rich history of parsing decisions by a greedy, bottom-up method (Kiperwasser and Goldberg, 2016a). The former might miss out the important information for specific headword predictions without the help of the structural features derived from the entire parse tree, while the latter may suffer from the error propagation as false structural constraints are used to create features when making future predictions. In this study, we explore the feasibility of explicitly taking advantage of high-order features while remaining the strength of global exhaustive inference and learning as a graph-based parser.

The proposed parser first encodes each word in a sentence by distributed embeddings using a convolutional neural network and constructs an initial parse graph by head-modifier predictions with a maximum directed spanning tree algorithm based on the first-order features (i.e. the score is factored over the arcs in a graph). Once an initial parse graph is built, the high-order features (such as grandparent, sibling, and uncle) can be defined, and used to refine the structure of the parse tree in an iterative way. Theoretically, the refinement will continue until no change is made in the iteration. But experimental results demonstrated that pretty good performance can be achieved with no more than twice updates because many dependencies are determined by independent arc prediction and a few head-modifier pairs need to be re-estimated after one update (i.e. only a few changes above and beyond the dominant first-order scores). We call this proposed model an incremental neural dependency parsing (INDP)[1].

2 Incremental Neural Dependency Parser

Given an input sentence x, we denote the set of all valid dependency parse trees that can be constructed from x as $\mathcal{Y}(x)$. Assuming there exists a graph scoring function s, the dependency parsing problem can be formulated as finding the highest scoring directed spanning tree for the sentence x.

$$y^*(x) = \underset{\hat{y} \in \mathcal{Y}(x)}{\operatorname{argmax}}\, s(x, \hat{y}; \theta) \tag{1}$$

where $y^*(x)$ is the parse tree with the highest score, and θ is a set of the parameters used to compute the scores. To make the search tractable, the score of a graph is usually factorized into the sum of its arc (head-modifier) scores (McDonald et al., 2005a).

$$s(x, \hat{y}; \theta) = \sum_{(h,m) \in A(\hat{y})} s(h, m; \theta) \tag{2}$$

[1]The source code is available at http://homepage.fudan.edu.cn/zhengxq/deeplearning/

where $A(\hat{y})$ represents a set of directed arcs in the parse tree $\hat{y}$. The score of an arc (h, m) represents the likelihood of creating a dependency from head h to modifier (or dependent) m in a dependency tree. If each arc score is estimated independently, we call it a first-order factorization. When the scoring is based on two or more arcs, second- or high-order factorizations are applied.

In traditional approaches, this score is commonly defined to be the product of a high dimensional feature representation of the arc and a learned weighting parameter vector. The performance of those systems is heavily dependent on the choice of features. For that reason, much effort in designing such systems goes into the feature engineering, which is important but labor-intensive, mainly first based on human ingenuity and linguistic intuition, and then confirmed or refined by empirical analyses. In this study, a neural network is designed instead to estimate the arc scores using the high-order features. In the following, we first describe how the word representations are produced. Then, the key components of the INDP, direction-specific scoring with special normalization and incremental refinement with high-order features, are discussed in detail. Finally, we present the entire parsing algorithm of the INDP.

2.1 Word Feature Representations

In graph-based neural dependency parsing work, such as (Kiperwasser and Goldberg, 2016a,b; Dozat and Manning, 2017), recurrent neural network (RNN) is a popular statistical learner used to produce the continuous vector representations for each word in a sentence due to its ability to bridge long time lags between relevant inputs. We chose to use one-dimensional convolution instead as a building block because it is good enough to capture the interactions of word feature representations in a context window with less computational cost. Such a design makes the parameters of our first-order parser to be optimized efficiently, which will be augmented with the high-order features (i.e. long distance dependencies) at incremental refinement stages.

The words are fed into the network as indices that are used by a lookup operation to transform words into their feature vectors. We consider a fixed-sized word dictionary $\mathcal{D}$[2]. The vector representations are stored in a word embedding matrix $E^{word} \in \mathbb{R}^{d \times |\mathcal{D}|}$, where d is the dimensionality of the vector space (a hyper-parameter to be chosen) and $|\mathcal{D}|$ is the size of the dictionary. Like (Chen and Manning, 2014; Dyer et al., 2015; Weiss et al., 2015; Cheng et al., 2016), we also map part-of-speech (POS) tags to another q-dimensional vector space, and provide POS type features for words. Formally, assume we are given a sentence $x_{[1:n]}$ that is a sequence of n words $x_i, 1 \leq i \leq n$. For each word $x_i \in \mathcal{D}$ that has an associated index k_i into the column of the matrix E^{word}, and is labeled as a POS tag of type l_i, its feature representation is obtained by concatenating both word and POS tag embeddings as:

$$E(x_i) = E^{word} e_{k_i} \oplus E^{pos} e_{l_i} \quad (3)$$

where $E^{pos} \in \mathbb{R}^{q \times |\mathcal{P}|}$ is a POS tag embedding matrix and $|\mathcal{P}|$ is the size of POS tag set $\mathcal{P}$ (fine-grained POS tags are used if available). Binary e_{k_i} and e_{l_i} are one-hot encoding vectors for the ith word in the sentence.

The lookup table layer extracts features for each single word, but the meaning of a word is strongly related to its surrounding words. Given a word, we consider a fixed size window w (another hyper-parameter) of words around it. More precisely, given an input sentence $x_{[1:n]}$, the feature window produced by the first lookup table layer at position x_i can be written as:

$$f^{win}_{x_i} = (E(x_{i-w/2}) \cdots E(x_i) \cdots E(x_{i+w/2})) \quad (4)$$

where the word feature window is a matrix $f^{win} \in \mathbb{R}^{(d+q) \times w}$, and each column of the matrix is the word feature vector in the context window. A one-dimensional convolution is used to yield another feature vector by taking the dot product of filter vectors with the rows of the matrix f^{win} at the same dimension. After each row of f^{win} is convolved with the corresponding column of a filter matrix W^1, some non-linear function $\phi(\cdot)$ will be applied as:

$$f^{con} = \phi(f^{win} \odot W^1) \quad (5)$$

where the weights in the matrix $W^1 \in \mathbb{R}^{w \times (d+q)}$ are the parameters to be trained, and the output $f^{con} \in \mathbb{R}^{(d+q)}$ is a vector. We choose a hyperbolic tangent as the non-linear function ϕ. The word feature vectors from a window of text can be computed efficiently thanks to the speed advantage of the one-dimensional convolution (Kalchbrenner et al., 2014).

[2]Unless otherwise specified, the word dictionary is extracted from the training set. Unknown words are mapped to a special symbol that is not used elsewhere.

2.2 Direction-Specific Scoring

For the same head-modifier arc (h, m), the head word h may occur on the left size of m (i.e. left-arc) in some sentences while it also can appear on the right size of m (i.e. right-arc) in other ones. Considering two English sentences excerpted from the Penn Treebank (Buchholz and Marsi, 2006): "A group of workers exposed to it.", and "Mr. Vinken is chairman of Elsevier, the Dutch publishing group.", they have the same (group, of) head-modifier arc, but those two words occur in different orders. This would not be problem in the traditional models, such as (McDonald et al., 2005a; Nivre and McDonald, 2008), in which the arc directions are directly used as features by their structured learning algorithms. However, it is hard to train a single neural network that gives a higher score to the left-arc case than the right-arc one in some situations while reverses in others because of the symmetries in weight space (Note that we cannot tell which case is correct in advance, and both cases need to be scored). It would be more serious when the first-order factorization is applied due to the lack of context information.

Based on the above observations, we use a multi-layer perceptron (MLP) to score the left-arc cases, and another MLP to score the right-arc ones. Those two MLPs share the word and POS tag embeddings, and can update them when necessary during the training process. Formally, if a MLP with one hidden layer is used, the score of each possible head-modifier arc is computed as:

$$s(h, m; \theta) = W^3(\phi(W^2(f_h^{con} \oplus f_m^{con} \oplus f_{h,m}^{dis}) + b^2)) \quad (6)$$

where the convolutional outputs of the head and dependent words are concatenated with a bucketed distance between the head and modifier, denoted by $f_{h,m}^{dis}$, in buckets of 0 (root), 1, 2, 3-5, and 6+, and feed into the MLP for scoring. The weights in the hidden and output layers are denoted by W^2 and W^3 respectively, and the corresponding bias by b^2. Once every possible arc is scored, we obtain a matrix like Figure 1, in which the element at the row i and column j is the score for (x_i, x_j) arc, denoted by $s(i, j)$. An artificial word, x_0, has been inserted at the beginning of a sentence that will always serve as the single root of the graph and is primarily a means to simplify computation. The scores at the lower (or upper) triangular are computed by the left-arc (or right-arc) MLP, and the shaded elements do not need to be calculated.

We can treat $s(i, j)$ as a score of the corresponding arc and then search for the highest scoring directed spanning tree to form a dependency parse tree as proposed in (McDonald et al., 2005b). This problem can be solved using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967), which can be implemented in $O(n^2)$.

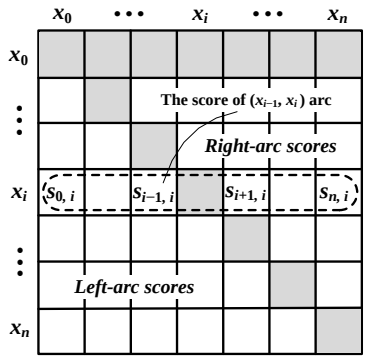

Figure 1: Scoring matrix for possible head and modifier arcs, in which the element at the row i and column j is the score for (x_i, x_j) arc, denoted by $s(i, j)$. A dependency tree can be formed by finding the highest scoring directed spanning tree over the scoring matrix.

The left-arc and right-arc MLPs should carefully collaborate with each other; otherwise, one MLP would be overwhelmed by another (i.e. the maximum score produced by one MLP is less than the minimum by another). To overcome this bias problem, we use the partition function by summing over the elements in each row of the scoring matrix, namely the scores/probabilities are normalized across the two MLPs. The conditional probability of arc (x_i, x_j) given a sentence $x_{[1:n]}$ is defined as:

$$p((x_i, x_j)|x_{[1:n]}; \theta) = \frac{\exp s(x_i, x_j; \theta)}{Z_i(x_{[1:n]}; \theta)}$$
$$\text{where } Z_i(x_{[1:n]}; \theta) = \sum_{i \in \{0 \ldots n\}, i \neq j} \exp s(x_i, x_j; \theta) \quad (7)$$

Each $Z_i(x_{[1:n]}; \theta)$ is a normalization term used to predict x_i's head word.

2.3 Incremental Refinement with High-order Features

Given an input sentence, once the initial dependency tree is built using the first-order factorization, we can define the high-order features over the resulting tree. For each head-modifier arc, the modifier's left sibling, right sibling, leftmost child, and rightmost child vector representations are concatenated with the inputs of Equation (6), which are then feed into two new left-arc and right-arc

MLPs to update the scoring matrix. Like the head and modifier, those additional feature representations are added as the results produced by the convolution layer. As shown in Figure 2, commonly-used high-order features have been take into account, such as consecutive sibling (H, B, S), tri-siblings (B, M, S), and grandparent (H, M, R). The missing feature vectors are replaced by one of four special vectors, namely "left-sibling", "right-sibling", "leftmost-child", and "rightmost-child" according to their relations to the modifier word.

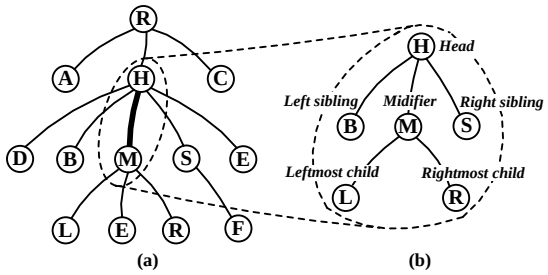

Figure 2: High-order features. (a) An example dependency parse tree. (b) The subgraph used to define high-order features for the head-modifier arc (H, M).

Although high-order features are used, the highest scoring parse tree still can be founded efficiently in $O(n^2)$ by the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). The main rationale is that, even in the presence of high-order features, the resulting scores remain based on single head-modifier arcs. The higher-order features are derived from the parse tree obtained with first-order inference, and because that tree is already pretty good, these higher-order features end up being a good approximation, and such approximation can be further improved by incremental refinements upon the parse tree. Thus, the high-order features used by the scoring MLPs can offer deliberate refinement above and beyond the first-order results. Theoretically, the refinement can be made until there is no update in the scoring matrix. However, experimental results show that comparable performance can be achieved with no more than twice high-order refinements (see Section 3).

We add a softmax layer to the network (after removing the last scoring layer) to predict syntactic labels for each arc. Labeling is trained by minimizing the cross-entropy error of the softmax layer using backpropagation. The network performs the structure prediction and labeling jointly. The two tasks shared the several layers (from the input to convolutional layers) of the network. When minimizing the cross-entropy error of the softmax layer, the error will also backpropagate and influence both the network parameters and the embeddings. We list our incremental neural dependency parsing algorithm in Figure 3. Staring with an initial tree formed using the first-order features, the algorithm makes changes to the parse tree with the high-order refinements in an attempt to climb the objective function.

Inputs:
θ: neural network parameters.
x: an input sentence.
T: maximum number of iterations.
Output: optimal dependency tree y^*.
Algorithm:
form an initial tree using the first-order features;
$t = 0$;
repeat
  update the scoring matrix using the high-order features;
  find the highest scoring tree y by Chu-Liu-Edmonds algorithm;
  $t = t + 1$;
until no change in this iteration **or** $t \geq T$;
predict syntactic labels based on the parse tree y;
return $y^* = y$;

Figure 3: Incremental neural dependency parsing (INDP) algorithm.

2.4 Training

Given a training example (x, y), we defined a structured margin $\Delta(x, y, \hat{y})$ loss for proposing a parse $\hat{y}$ for sentence x when y is the true parse. This penalty is proportional to the number of unlabeled arcs on which the two parse trees do not agree. In general, $\Delta(x, y, \hat{y})$ is equal to 0 if $y = \hat{y}$. The loss function is defined as a penalization of incorrect arcs:

$$\Delta(x, y, \hat{y}) = \sum_{(h,m) \in A(\hat{y})} \kappa \mathbf{1}\{(h, m) \notin A(y)\} \quad (8)$$

where κ is a penalization term to each incorrect arc, and $A(y)$ is a set of arcs in the true parse y.

For a training set, we seek a function with small expected loss on unseen sentences. The function we consider take the following form as Equation (1). The score of a tree $\hat{y}$ is higher if the algorithm is more confident that the structure of the tree is correct. In the max-margin estimation framework, we want to ensure that the highest scoring tree is the true parse for all training instances $(x^i, y^i), i = 1, \cdots, h$, and it's score to be larger up to a margin defined by the loss. For all i in the training data:

$$s(\theta, x^i, y^i) \geq s(\theta, x^i, \hat{y}) + \Delta(x^i, y^i, \hat{y}) \quad (9)$$

These lead us to minimize the following regularized objective for h training instances:

$$\begin{aligned} J(\theta) &= \frac{1}{h}\sum_{i=1}^{h} E^i(\theta) + \frac{\lambda}{2}||\theta||^2, \text{where} \\ E_i(\theta) &= \max_{\hat{y}\in\mathcal{Y}(x^i)} (0, (s(\theta, x^i, \hat{y}) + \Delta(x^i, y^i, \hat{y})) \\ &\quad - s(\theta, x^i, y^i)) \end{aligned} \tag{10}$$

where the coefficient λ governs the relative importance of the regularization term compared with the error. The trees are penalized more by the loss when they deviate from the correct one. Minimizing this objective maximizes the score of the correct tree, and minimizes that of the highest scoring but incorrect parse tree. The objective is not differentiable due to the hinge loss. We use the subgradient method to compute a gradient-like direction for minimizing the objective function.

3 Experiments

We conducted three sets of experiments. The first one is to test several variants of the INDP on the development set, to gain some understanding of how the choice of hyper-parameters impacts upon the performance. The goal of the second one is to see how well the incremental approach enhanced with the high-order features to improve the first-order results by analysing parsing errors relative to sentence length. In the third set, we compared the performance of the INDP with existing state-of-the-art models on both English and Chinese datasets. We report unlabeled attachment scores (UAS) and labeled attachment scores (LAS) with punctuations being omitted from evaluation.

3.1 Datasets

We show test results for the proposed model on the English Penn Treebank (PTB), converted into Stanford dependencies using version 3.3.0 of the Stanford dependency converter, and the Chinese Penn Treebank (CTB). We follow the standard splits of PTB, using section 2-21 for training, section 22 as development set and 23 as test set. We use POS tags generated from the Stanford POS tagger (Toutanova et al., 2003); for the Chinese PTB dataset, we use gold word segmentation and POS tags.

3.2 Training Strategy

Previous work demonstrated that the performance can be improved by using word embeddings learned from large-scale unlabeled data in many NLP tasks both in English (Collobert et al., 2011; Socher et al., 2011) and Chinese (Zheng et al., 2013). Unsupervised pretraining guides the learning towards basins of attraction of minima that support better generalization (Erhan et al., 2010). We leveraged large unlabeled corpus to learn word embeddings, and then used these improved embeddings to initialize the word embedding matrices of the neural networks. English and Chinese Wikipedia documents were used to train the word embeddings by Word2Vec tool[3] proposed in (Mikolov et al., 2013).

Previous studies show that a joint solution (i.e., performing several tasks at the same time) usually leads to the improvement in accuracy over pipelined systems because the error propagation is avoided and the various information normally used in the different steps of pipelined systems can be integrated. The INDP networks are also trained in a joint way, but adopting three-step strategy. The parameters of the parsing neural network using the first-order factorization are first learned, and when its unlabeled parsing accuracy exceeds a given threshold (e.g. 85%), we start to train the high-order parsing network. The weights already trained in the first step will remain unchanged for the first several epochs, and they are in fact used to generate the high-order features. After the parsing accuracy reaches another threshold (e.g. 90%), all the parameters for the first-order, and high-order predictions as well as labeling are trained jointly.

3.3 Hyper-parameter Choices

Hyper-parameters was tuned with the PTB 3.3.0 development set by trying only a few different networks. Generally, the dimensionality of the embeddings, and the numbers of hidden units, provided they are large enough, have a limited impact on the generalization performance. In the following experiments, the window size was set to 5, the learning rate to 0.02, and the number of hidden layer to 300. The embedding size of words was set to 50, and that of tags to 30, which achieved a good trade-off between speed and performance. All experiments were run on a computer equipped with an Intel Xeon processor working at 2.2GHz, with 16GB RAM and a NVIDIA Titan GPU. The parsing speed of the INDP is around 250-300 sents/sec in average on the PTB dataset.

[3] Available at http://code.google.com/p/word2vec/

3.4 Sentence Length Factors

It is well known that dependency parsers tend to have lower accuracies for longer sentences because the increased presence of complex syntactic structures. In order to get a better understanding of how well the incremental strategy and high-order features benefit the models, Figure 4 shows the accuracy of our neural dependency parser using the first-order features only (indicated with "NDP + First-order") and INDP with at most twice high-order refinements (indicated with "INDP + High-order + M2") on the English PTB develop set. For simplicity, the experiments report unlabeled parsing accuracy, and identical experiments using labeled parsing accuracy did not reveal any additional information.

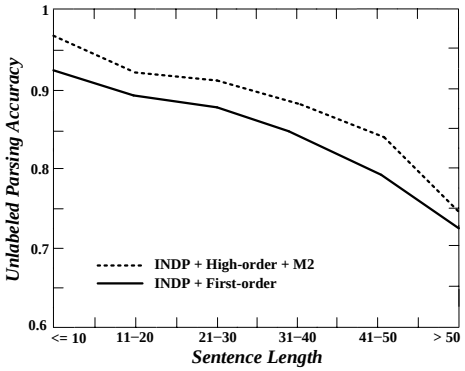

Figure 4: Accuracy relative to sentence length.

The INDP with the high-order refinements is more precise than the parser using only the first-order features. Due to the fact that longer dependencies are typically harder to parse, there is still a degradation in performance for our INDP. However, the accuracy curve for INDP is slightly flatter than its reduced version in which the high-order features and incremental recipe are not applied when the sentence length is within 11-50. This behavior can be explained by the reasons that the feature representations are not restricted to a limited number of graph arcs, but can take into account with the (almost) entire dependency graph built so far at the refinement stages of the INDP, and it do offer substantial refinements.

3.5 Results

We report the experimental results on the English PTB and Chinese CTB datasets in Table 1 and 2 respectively, in which our networks are denoted by "INDP". The "M1" indicates that the results are obtained by the INDP with just one refinement over the parse graphs built using the first-order features, and similarly, the "M2" indicates the results are achieved by the INDP with at most twice high-order refinements, while the "UNC" in the last row indicates that the refinements will continue until no change is made in the structure predictions (see the algorithm listed in Figure 3). All compared transition-based parsing systems are indicated by a "‡", and graph-based ones by "§".

Table 1: Results on the English PTB dataset.

Model	UAS	LAS
Zhou et al (2015)‡	93.28	92.35
Weiss et al (2015)‡	94.26	92.41
Ballesteros et al (2016)‡	93.56	91.42
Kiperwasser and Goldberg (2016b)‡	93.90	91.90
Andor et al (2016)‡	94.61	92.79
Kuncoro et al (2016)‡	**95.80**	**94.60**
Kiperwasser and Goldberg (2016a)§	93.00	90.90
Cheng et al (2016)§	94.10	91.49
Hashimoto et al (2016)§	94.67	92.90
Dozat and Manning (2017)§	95.74	94.08
NDP + First-order	90.88	88.93
INDP + High-order + M1	93.31	91.51
INDP + High-order + M2	94.76	93.12
INDP + High-order + UNC	95.53	93.94

From these numbers, a handful of trends are readily apparent. Firstly, we note that the "full-fledged" INDP (indicated with "UNC") is superior to that without the high-order refinements by a fairly significant margin (5.01% for English and 6.55% for Chinese in LAS). Another striking result of these experiments is that comparable performance can be obtained by no more than twice refinements with high-order features, and "INDP + High-order + M2" achieves a good trade-off between the performance and parsing complexity.

Table 2: Results on the Chinese CTB dataset.

Model	UAS	LAS
Ballesteros et al (2016)‡	87.65	86.21
Kiperwasser and Goldberg (2016b)‡	87.60	86.10
Kiperwasser and Goldberg (2016a)§	87.10	85.50
Cheng et al (2016)§	88.10	85.70
Dozat and Manning (2017)§	89.30	**88.23**
NDP + First-order	82.97	81.39
INDP + High-order + M1	87.35	85.82
INDP + High-order + M2	88.78	87.28
INDP + High-order + UNC	**89.42**	87.94

Our INDP gets nearly the same performance on the English PTB as the current models of (Kuncoro et al., 2016) and (Dozat and Manning, 2017)

in spite of its simpler architectures, and gets state-of-the-art UAS accuracy on the Chinese CTB. The INDP lags behind in LAS, indicating one of a few possibilities. Firstly, we tried only a few different network configurations, and there are many ways (such as using deeper architectures, and recruiting bi-directional recurrent neural networks to produce word feature representations) that we could improve it further. Secondly, the model of (Kuncoro et al., 2016) is particularly designed to capture phrase compositionality, and thus, another possible improvement is to capture such compositionality by optimizing the network architectures, which may also lead to a better label score.

4 Related Work

Dependency-based syntactic representations of sentences have been found to be useful for various NLP tasks, especially for those involving natural language understanding in some way. We briefly review prior work both on graph-based and transition-based neural dependency parsers.

In transition-based parsing, we learn a model for scoring transitions from one state to the next, conditioned on the parse history, and parse a sentence by taking the highest-scoring transition out of every state until a complete dependency graph has been derived. Chen and Manning (2014) made the first successful attempt at introducing deep learning into a transition-based dependency parser. At each step, the feed-forward neural network assigns a probability to every action the parse can take from certain state (words on the stack and buffer). Some researchers have attempted to address the limitations of (Chen and Manning, 2014) by augmenting it with additional complexity.

A beam search and a conditional random field loss function were incorporated into the transition-based neural network models (Weiss et al., 2015; Zhou et al., 2015; Andor et al., 2016), which allow the parsers to keep the top-k partial parse trees and revoke previous actions once it finds evidence that they may have been incorrect by locally greedy choices. Dyer et al (2015) used three LSTMs to represent the buffer, stack, and parsing history, getting state-of-the-art results on Chinese and English dependency parsing tasks.

Graph-based parsers use machine learning for scoring each possible edge for a given sentence, typically by factoring the graphs into their component arcs, and constructing the parse tree with the highest score from these weighted edges. Kiperwasser and Goldberg (2016b) presented a neural graph-based parser in which the bi-directional LSTM's recurrent output vector for each word is concatenated with each possible head's vector (also produced by the same biLSTM), and the result is used as input to a multi-layer perceptron (MLP) for scoring this modifier-head pair. Given the scores of the arcs, the highest scoring tree is constructed using Eisner's decoding algorithm (Eisner, 1996). Labels are predicted similarly, with each word's recurrent output vector and its head's vector being used in a multi-class MLP.

Kiperwasser and Goldberg (2016a) also proposed a hierarchical tree LSTM to model the dependency tree structures in which each word is represented by the concatenation of its left and right modifier (child) vectors, and the modifier vectors are generated by two (leftward or rightward) recurrent neural networks. The tree representations were produced in a bottom-up recursive way with the (greedy) easy-first parsing algorithm (Goldberg and Elhadad, 2010). Similarly, Cheng et al (2016) proposed a graph-based neural dependency parser that is able to predict the scores for the next arc, conditioning on previous parsing decisions. In addition to using one bi-directional recurrent network that produces a recurrent vector for each word, they also have uni-directional recurrent neural networks (left-to-right and right-to-left) that keep track of the probabilities of each previous parsing actions.

In their many-task neural model, Hashimoto et al (2016) included a graph-based dependency parse in which the traditional MLP-based method that Kiperwasser and Goldberg (2016b) used was replaced with a bilinear one. Dozat and Manning (2017) modified the neural graph-based approach of (Kiperwasser and Goldberg, 2016b) in a few ways to improve the performance. In addition to building a network that is larger and uses more regularization, they replace the traditional MLP-based attention mechanism and affine label classifier with biaffine ones.

This work is most closely related to the graph-based parsing approaches with multiple high-order refinements (Rush and Petrov, 2012; Zhang et al., 2014), although the neural networks were not used in their parsers. Rush and Petrov (2012) proposed a multi-pass coarse-to-fine approach in which a coarse model was used to prune the search space

in order to make the inference with up to third-order features practical. They start with a linear-time vine pruning pass and build up to high-order models. Zhang et al (2014) introduced a randomized greedy algorithm for dependency parsing in which they begin with a tree drawn from the uniform distribution and use hill-climbing strategy to find the optimal parse tree. Although they reported that drawing the initial tree randomly results in the same performance as when initialized from a trained first-order distribution, but multiple random restarts are required to avoid getting stuck in a locally optimal solution. Their greedy algorithm breaks the parsing into a sequence of local steps, which correspond to choosing the head for each modifier word (one arc at a time) in the bottom-up order relative to the current tree. In contrast, we employed the global inference algorithm to change the entire tree (all at a time) in each refinement step, which makes the improvement more efficient.

5 Conclusion

Graph-based parsers cannot easily condition on any extended scope of the dependency parse tree beyond a single arc, which is their primary shortcoming relative to transition-based competitors. We have shown that a simple, generally applicable incremental neural dependency parsing algorithm can deliver close to state-of-the-art parsing performance, which allows the high-order features to be taken into account without hurting the advantage of global exhaustive inference and learning as a member of graph-based parsing systems. Future work will involve exploring ways of augmenting the parser with a more innovative architecture than the relatively simple one used in current neural graph-based parsers.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments. This work was partly supported by a grant from Shanghai Municipal Natural Science Foundation (No. 15511104303).

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL'16)*.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack-LSTM parser. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'16)*.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the International Conference on Computational Natural Language Learning (CoNLL'06)*.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *arXiv: 1608.02076*.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

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*, 12:2493–2537.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency paring with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL'15)*.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71:233–240.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, and Pascal Vincent. 2010. Why does unsupervised pre-training help deep learning. *Journal of Machine Learning Research*, 11:625–660.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'10)*.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryiğit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single Malt or blended? a study in multilingual parser optimization. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*.

Kazuma Hashimoto, Yoshimasa Tsuruoka Caiming Xiong, and Richard Socher. 2016. A joint many-task model: Growing a neural network for multiple NLP tasks. In *arXiv: 1611.01587*.

Liang Huang and Kenjie Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*.

Nal Kalchbrenner, Edward Grefenstette, and Fernando Pereira. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14)*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

Eliyahu Kiperwasser and Yoav Goldberg. 2016b. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Adhiguna Kuncoro, Miguel Ballesteros, Chris Dyer Lingpeng Kong, Graham Neibig, and Noah A. Smith. 2016. What do recurrent neural network grammars learn about syntax? In *arXiv: 1611.05774*.

André F. T. Martins, Dipanian Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'08)*.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer liear programming formulations for dependency parsing. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing (ACL-IJCNLP'09)*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Lingustics*, 37:197–230.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hanjič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conferece and the International Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP'05)*.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT'07)*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Tetsuji Nakagawa. 2007. Multilingual depdendency parsing using global features. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL'08:HLT)*.

Sebastian Riedel and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'06)*.

Alexander M. Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL'12)*.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL'06)*.

David Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'08)*.

Richard Socher, Cliff C-Y. Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the International Conference on Machine learning (ICML'11)*.

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 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'03)*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Stuctured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL'15)*.

Daniel Zeman and Zdeněk Žabokrtskỳ. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the International Workshop on Parsing Technologies (IWPT'05)*.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*.

Yue Zhang and Stephen Clark. 2008. A tale of two parses: Investigating and bombining graph-based and transition-based dependency parsing. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'08)*.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and pos tagging. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP'13)*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL'15)*.