A Search-Based Dynamic Reranking Model for Dependency Parsing

Hao Zhou† Yue Zhang‡ Shujian Huang† Junsheng Zhou*
Xin-Yu Dai† Jiajun Chen†
†State Key Laboratory for Novel Software Technology, Nanjing University, China
‡Singapore University of Technology and Design, Singapore
* Nanjing Normal University, China
zhouh@nlp.nju.edu.cn, yue_zhang@sutd.edu.sg
{huangshujian,daixinyu,chenjj}@nju.edu.cn, zhoujs@njnu.edu.cn

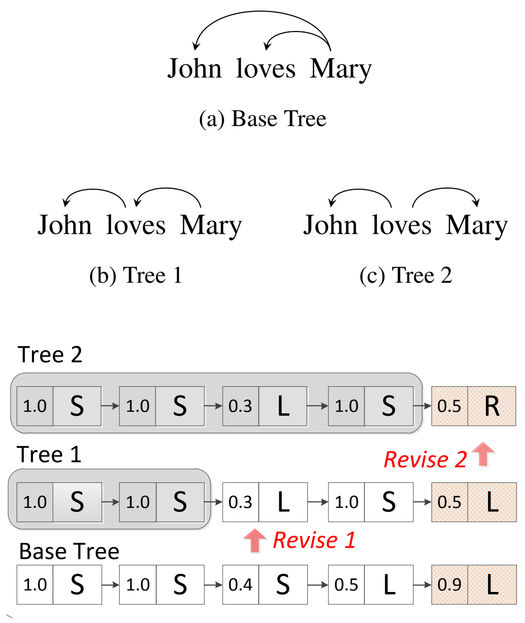
Abstract

We propose a novel reranking method to extend a deterministic neural dependency parser. Different to conventional k-best reranking, the proposed model integrates search and learning by utilizing a dynamic action revising process, using the reranking model to guide modification for the base outputs and to rerank the candidates. The dynamic reranking model achieves an absolute 1.78% accuracy improvement over the deterministic baseline parser on PTB, which is the highest improvement by neural rerankers in the literature.

1 Introduction

Neural network models have recently been exploited for dependency parsing. Chen and Manning (2014) built a seminal model by replacing the SVM classifier at the transition-based MaltParser (Nivre et al., 2007) with a feed-forward neural network, achieving significantly higher accuracies and faster speed. As a local and greedy neural baseline, it does not outperform the best discrete-feature parsers, but nevertheless demonstrates strong potentials for neural network models in transition-based dependency parsing.

Subsequent work aimed to improve the model of Chen and Manning (2014) in two main directions. First, global optimization learning and beam search inference have been exploited to reduce error propagation (Weiss et al., 2015; Zhou et al., 2015). Second, recurrent neural network models have been used to extend the range of neural features beyond a local window (Dyer et al., 2015; Ballesteros et al., 2015). These methods give accuracies that are competitive to the best results in the literature.



(d) 2-step action revising process for sentence "John loves Mary". Numbers before actions are the probabilities for that action.

Figure 1: Example action revising process. S, L, R stand for the SHIFT, LEFT, RIGHT actions, respectively (Section 2).

Another direction to extend a baseline parser is reranking (Collins and Koo, 2000; Charniak and Johnson, 2005; Huang, 2008). Recently, neural network models have been used to constituent (Socher et al., 2013; Le et al., 2013) and dependency (Le and Zuidema, 2014; Zhu et al., 2015) parsing reranking. Compared with rerankers that rely on discrete manual features, neural network rerankers can potentially capture more global information over whole parse trees.

Traditional rerankers are based on chart parsers, which can yield exact k-best lists and forests. For reranking, this is infeasible for the transition-based neural parser and neural reranker, which have rather weak feature locality. In addition, k-best lists from the baseline parser are not necessarily the best candidates for a reranker. Our preliminary results show that reranking candidates can be constructed by modifying unconfident actions in the baseline parser output, and letting the baseline parser re-decode the sentence from the modified action. In particular, revising two incorrect actions of the baseline parser yields oracle with 97.79% UAS, which increases to 99.74% by revising five actions. Accordingly, we design a novel search-based dynamic reranking algorithm by revising baseline parser outputs.

For example, the sentence: "*John loves Mary*", the baseline parser generates a *base tree* (Figure 1a) using 5 shift-reduce actions (Figure 1d) of Section 2. The gold parse tree can be obtained by a 2-step action revising process:

$$\text{Base Tree} \xrightarrow{\text{Revise 1}} \text{Tree 1} \xrightarrow{\text{Revise 2}} \text{Tree 2}$$

As shown in Figure 1d, we first revise the least confident action S of the *base tree*, running the baseline parser again from the revised action to obtain *tree 1*. This corrects the *John* $\curvearrowleft$ *loves* dependency arc. Then we obtain the gold parsing tree (*tree 2*) by further revising the least confident action in *tree 1* on the second action sequence.

Rather than relying on the baseline model scores alone for deciding the action to revise (*static search*), we build a neural network model to guide which actions to revise, as well as to rerank the output trees (*dynamic search*). The resulting model integrates search and learning, yielding the minimum amount of candidates for the best accuracies. Given the extensively fast speed of the baseline parser, the reranker can be executed with high efficiency.

Our dynamic search reranker has two main advantages over the static one: the first is *training diversity*, the dynamic reranker searches over more different structurally diverse candidate trees, which allows the reranker to distinguish candidates more easily; the second is *reranking oracle*, with the guidance of the reranking model, the dynamic reranker has a better reranking oracle compared to the static reranker.

On WSJ, our dynamic reranker achieved 94.08% and 93.61% UAS on the development and test sets, respectively, at a speed of 16.1 sentences per second. It yields a 0.44% accuracy improvement (+1.78%) from the same number of candidates, compared to a static reranker (+1.34%), obtaining the largest accuracy improvement among related neural rerankers.

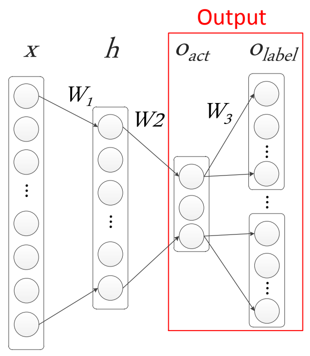


Figure 2: Hierarchical neural parsing model.

2 Baseline Dependency Parser

Transition-based dependency parsers scan an input sentence from left to right, performing a sequence of transition actions to predict its parse tree (Nivre, 2008). We employ the *arc-standard* system (Nivre et al., 2007), which maintains partially-constructed outputs using a *stack*, and orders the incoming words in the sentence in a *queue*. Parsing starts with an empty stack and a queue consisting of the whole input sentence. At each step, a *transition action* is taken to consume the input and construct the output.

Formally, a *parsing state* is denoted as $\langle j, S, L\rangle$, where S is a stack of subtrees $[\ldots s_2, s_1, s_0]$, j is the head of the queue (i.e. $[\, q_0 = w_j, q_1 = w_{j+1} \cdots]$), and L is a set of dependency arcs that has been built. At each step, the parser chooses one of the following actions:

- SHIFT (S): move the front word w_j from the queue onto the stacks.
- LEFT-l (L): add an arc with label l between the top two trees on the stack ($s_1 \leftarrow s_0$), and remove s_1 from the stack.
- RIGHT-l (R): add an arc with label l between the top two trees on the stack ($s_1 \rightarrow s_0$), and remove s_0 from the stack.

Given the sentence "*John loves Mary*", the gold standard action sequence is S, S, L, S, R.

2.1 Model

Chen and Manning (2014) proposed a deterministic neural dependency parser, which rely on dense embeddings to predict the optimal actions at each step. We propose a variation of Chen and Manning

(2014), which splits the output layer into two hierarchical layers: the action layer and dependency label layer. The hierarchical parser determines a action in two steps, first deciding the *action type*, and then the *dependency label* (Figure 2).

At each step of deterministic parsing, the neural model extracts n atomic features from the parsing state. We adopt the feature templates of Chen and Manning (2014). Every atomic feature is represented by a feature embedding $e_i \in \mathbb{R}^d$, An *input layer* is used to concatenate the n feature embeddings into a vector $x = [e_1; e_2 \dots e_n]$, where $x \in \mathbb{R}^{d \cdot n}$. Then x is mapped to a d_h-dimensional *hidden layer h* by a mapping matrix $W_1 \in \mathbb{R}^{d_h \times d \cdot n}$ and a cube activation function for feature combination:

$$h = (W_1 x + b_1)^3 \quad (1)$$

Our method is different from Chen and Manning (2014) in the output layer. Given the hidden layer h, the action type output layer o_{act} and the label output layer $o_{label}(a_i)$ of the action type a_i are computed as

$$o_{act} = W_2 h \quad (2)$$

$$o_{label}(a_i) = W_3^i h \ , \quad (3)$$

Where $W_2 \in \mathbb{R}^{d_a \times d_h}$ is the mapping matrix from the hidden layer to the action layer, and d_a is the number of action types. $W_3^i \in \mathbb{R}^{d_{label} \times d_h}$ is the mapping matrix from the hidden layer to the corresponding label layer, d_{label} is the number of dependency labels.

The probability of a labeled action $y_{i,j}$ given its history $Acts$ and input x is computed as:

$$\begin{aligned} & p(y_{i,j} \mid x, Acts) \\ = \ & p(a_i \mid x, Acts) \times p(l_j \mid x, Acts, a_i) \end{aligned} \quad (4)$$

where

$$p(a_i \mid x, Acts) = \frac{e^{o_{act}^i}}{\sum_{k=1}^{d_a} e^{o_{act}^k}} \quad (5)$$

$$p(l_j \mid x, Acts, a_i) = \frac{e^{o_{label}^j(a_i)}}{\sum_{k=1}^{d_{label}} e^{o_{label}^k(a_i)}} \ , \quad (6)$$

Here a_i is the i_{th} action in the action layer, and l_j is the j_{th} label in the label layer for a_i.

In training, we use the cross-entropy loss to maximum the probability of training data A:

$$L(\theta) = -\sum_{y_{i,j} \in A} \log p(y_{i,j} \mid x, Acts) \quad (7)$$

Experiments show that our hierarchical neural parser is both faster and slightly accurate than the original neural parser.

3 Reranking Scorer

We adopt the recursive convolutional neural network (RCNN) of Zhu et al. (2015) for scoring full trees. Given a dependency subtree rooted at h, c_i $(0 < i \leq L)$ is the i_{th} child of h. The dependency arc (h, c_i) is represented by:

$$z_i = tanh(W^{(h,c_i)} p_i) \ , \quad (8)$$

where

$$p_i = w_h \oplus x_{c_i} \oplus d^{(h,c_i)} \quad (9)$$

Here $p_i \in \mathbb{R}^n$ is the concatenation of head word embedding w_h, child phrase representation x_{c_i} and the distance embeddings $d^{(h,c_i)}$. $W^{(h,c_i)} \in \mathbb{R}^{m \times n}$ is a linear composition matrix, which depends on the POS tags of h and c_i. The subtree phrase representation x^h are computed using a max-pooling function on rows, over the matrix of arc representations Z^h.

$$Z^h = [z_1, z_2, \dots, z_L] \quad (10)$$

$$x_j^h = \max_i Z_{j,i}^h, 0 < j < m \quad (11)$$

The subtree with the head h is scored by:

$$score(h) = \sum_{i=1}^{L} v^{h,c_i} z_i \quad (12)$$

Here, v^{h,c_i} is the score vector, which is a vector of parameters that need to be trained. The score of the whole dependency tree y is computed as:

$$s_t(x, y, \Theta) = \sum_{w \in y} score(w), \quad (13)$$

where w is the node in tree y and Θ denotes the set of parameters in the network.

4 Search-based Dynamic Reranking for Dependency Parsing

Using the hierarchical parser of Section 2 as the baseline parser, we propose a search-based dynamic reranking model, which integrates search and learning by searching the reranking candidates dynamically, instead of limiting the scope to a fixed k-best list. The efficiency of the reranking model is guaranteed by 3 properties of the baseline parser, namely *revising efficiency*, *probability diversity* and *search efficiency*.

Revising Depth	UAS	LAS
0	92.28	91.15
1	95.76	94.42
2	97.79	96.63
3	98.77	97.55
4	99.39	98.15
5	99.74	98.47

Table 1: Oracle of the baseline parser after revising actions. Revising depth is the maximum number of revised actions for one sentence.

Action Type		Num	Average Probability
Gold	Shift	39194	99.38%
	Right	19477	98.90%
	Left	19556	99.61%
Incorrect	Shift	968	84.96%
	Right	746	85.88%
	Left	338	85.03%

Table 2: Average action probabilities.

4.1 Properties of the Baseline Parser

To demonstrate the above three properties, we give some preliminary results for the baseline. To parse the 1,695 sentences in Section 22 of WSJ, our baseline parser needs to perform 78,227 shift-reduce actions. During the process, if we correct every encountered incorrectly determined action and let the baseline parser re-decode the sentence from the point, we need to revise 2,052 actions, averaging 1.2 actions per sentence. In other words, the baseline parser can parse the 1,695 sentences correctly with 2,052 action being revised.

Note that the revise operation is required to change the action type (i.e. S, L). After revising the action type, the optimal dependency label will be chosen for parsing by the hierarchical baseline parser. We only modify the action type in the revising process. Thus the modified trees are always structurally different instead of only with different dependency labels compared to the original one, which guarantees structured diversity.

Revising Efficiency It can be seen from Table 1 that revising one incorrect action results in 3.5% accuracy improvement. We obtain a 99.74% UAS after a maximum 5 depth revising. Although we only revise the action type, the LAS goes up with the UAS. The property of *revising efficiency* suggests that high quality tree candidates can be found with a small number of changes.

Probability Diversity Actions with lower probabilities are more likely to be incorrect. We compute the average probabilities of gold and incorrect actions in parsing the section 22 of WSJ (Table 2), finding that most gold actions have very high probabilities. The average probabilities of the gold actions is much higher than that of the incorrectly predicted ones, indicating that revising actions with lower probabilities can lead to better trees.

Search Efficiency The fast speed of the baseline parser allows the reranker to search a large number of tree candidates efficiently. With the graph stack trick (Goldberg et al., 2013), the reranker only needs to perform partial parsing to obtain new trees. This enables a fast reranker in theory.

4.2 Search Strategy

Given an output sequence of actions by the baseline parser, we revise the action with the lowest *probability margin*, and start a new branch by taking a new action at this point. The probability margin of an action a is computed as: $p(a_{max}) - p(a)$, where a_{max} is the action taken by the baseline, which has the highest model probability. a is taken instead of a_{max} for this branch, and the baseline parser is executed deterministically until parsing finishes, thus yielding a new dependency tree. We require that the action type must change in the revision and the most probable dependency label among all for the revised action type will be used.

Multiple strategies can be used to search for the revised reranking process. For example, one intuitive strategy is best-first, which modifies the action with the lowest probability margin among all sequences of actions constructed so far. Starting from the original output of the baseline parser, modifying the action with the lowest probability margin results in a new tree. According to the best-first strategy, the action with the lowest probability margin in the two outputs will be revised next to yield the third output. The search repeats until k candidates are obtained, which are used as candidates for reranking.

The best-first strategy, however, does not consider the quality of the output, which is like a greedy process. A better candidate (with higher F1 score) is more likely to take us to the gold tree. With the best-first strategy, we revise one tree at each time. If the selected tree is not the optimal one, the revised tree will be less likely the gold one. Revising a worse output is less likely to generate the gold parse tree compared with revising a relatively better output. Our preliminary experiments confirms this intuition. As a result, we take a beam search strategy, which uses a beam to hold b outputs to modify.

For each tree in beam search, most f actions with the lowest probability margin are modified, leading to $b \times f$ new trees. Here, b is the beam size, f is the revising factor. From these trees, the b best are put to the beam for the next step. Search starts with the beam containing only the original base parse, and repeats for l steps, where l is called the revising depth. The best tree will be selected from all the trees constructed. The search process for example in Figure 1 is illustrated in Figure 3, in which b = 1, f = 3 and l = 2.

At each iteration, the b best candidates can be decided by the baseline parser score alone, which is the product of the probability of each action. We call this the *static search* reranking. As mentioned in the introduction, the baseline model score might not be the optimal criteria to select candidates for reranking, since they may not reflect the best oracle or diversity. We introduce a *dynamic search* strategy instead, using the reranking model to calculate heuristic scores for guiding the search.

4.3 Search-Based Dynamic Reranking

Doppa et al. (2013) propose that structured-prediction by learning guide search should maintain two different scoring functions, a heuristic function for guiding search and a cost function for obtaining the best output. Following Doppa et al. (2013), we use the RCNN in Section 3 to yield two different scores, namely a heuristic score $s_t(x, y, \Theta_h)$ to guide the search of revising, and a cost score $s_t(x, y, \Theta_c)$ to select the best tree output.

Denote $b(i)$ as the beam at i-th step of search, k-best candidates in the beam of $i+1$ step is:

$$b(i+1) = \underset{c \in c(i)}{\arg K}(s_t(x, c, \Theta_h) + s_b(x, c)), \quad (14)$$

where $c(i)$ denotes the set of newly constructed trees by revising trees in $b(i)$, $s_b(x, c)$ is the baseline model score and $\arg K$ leaves the k best candidate trees to the next beam. Finally, the output tree y_i of reranking is selected from all searched trees C in the revising process

$$y_i = \arg\max_{c \in C}(s_t(x, c, \Theta_c) + s_b(x, c)) \quad (15)$$

Interpolated Reranker In testing, we also adopt the popular mixture reranking strategy (Hayashi et al., 2013; Le and Mikolov, 2014), which obtains better reranking performance by a linear combination of the reranking score and the baseline model score.

$$y_i = \arg\max_{y \in \tau(x_i)} (\beta(s_t(x_i, y, \Theta_c) + s_t(x, y, \Theta_h)) + (1-\beta)s_b(x_i, y)) \quad (16)$$

Here y_i is the final output tree for a sentence x_i; $\tau(x_i)$ returns all the trees candidates of the dynamic reranking; $\beta \in$[0, 1] is a hyper-parameter.

Algorithm 1: Training Algorithm for the Search-Based Dynamic Reranking.

```
Input: Sentence x, Gold Trees y
Output: $\Theta_h$, $\Theta_c$
for iter ← 1 to N do
    $D_h$ = [];
    $D_k$ = [];
    foreach (x, y) ∈ (x, y) do
        bestHScoreT = null;
        bestCScoreT = null;
        bestUAST = null;
        initTree = BaselineParse(x);
        $b_1$ = [initTree];
        $b_2$ = [];
        for d ← 1 to depth do
            foreach $t \in b_1$ do
                revisedActs = Seek (t);
                revisedTrees = Revise (t, revisedActs);
                bestK = Sort (revisedTrees, $\Theta_h$ )
                $b_2$.Add (bestK);
                bestHScoreT = MaxScore (bestHScoreT, revisedTrees, $\Theta_h$);
                bestCScoreT = MaxScore (bestCScoreT, revisedTrees, $\Theta_c$);
                bestUAST = MaxUAS (bestUAST, revisedTrees, y)
            $b_1$ = $b_2$;
            $b_2$ = [];
        $D_h$.Add (x, bestUAST, bestTScoreT);
        $D_c$.Add (x, y, bestCScoreT);
    Update($D_h$, $\Theta_h$);
    Update($D_c$, $\Theta_c$);
```

4.4 Training

As k-best neural rerankers (Socher et al., 2013; Zhu et al., 2015), we use the max-margin criterion to train our model in a stage-wise manner (Doppa et al., 2013). Given training data D_c = $(x_i, y_i, \hat{y_i})_{i=1}^{N}$, where x_i is the sentence, $\hat{y_i}$ is the output tree with highest cost score and y_i is the corresponding gold tree, the final training objective is to minimize the loss function $J(\Theta_c)$, plus a

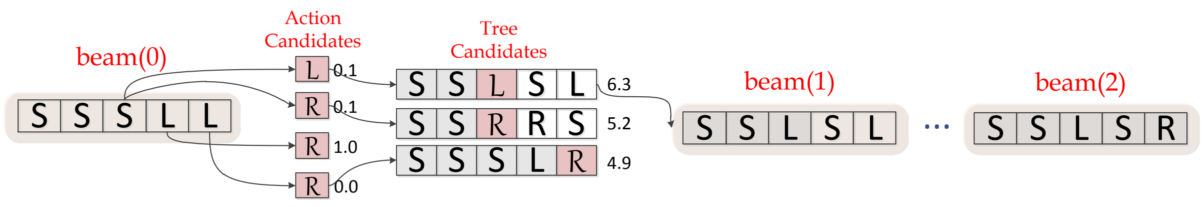


Figure 3: The beam search revising process of the example in Figure 1 with $b = 1$, $f = 3$ and $l = 2$

l_2-regularization:

$$J(\Theta_c) = \frac{1}{|D_c|} \sum_{(x_i,y_i,\hat{y}_i)\in D_c} r_i(\Theta_c) + \frac{\lambda}{2}||\Theta_c|| \tag{17}$$

$$r_i(\Theta_c) = \max(0, s_t(x_i, \hat{y}_i, \Theta_c) + \Delta(y_i, \hat{y}_i) - s_t(x_i, y_i, \Theta_c)) \tag{18}$$

Here, Θ_c is the model, $s_t(x_i, y_i, \Theta_c)$ is the cost reranking score for y_i.

$$\Delta(y_i, \hat{y}_i) = \sum_{d\in\hat{y}_i} \kappa \mathbf{1}\{d \notin y_i\} \tag{19}$$

$\Delta(y_i, \hat{y}_i)$ is the structured margin loss between y_i and $\hat{y}_i$, measured by counting the number of incorrect dependency arcs in the tree (Goodman, 1998; Zhu et al., 2015).

Given training data $D_h = (x_i, y_i', \hat{y}_i')_{i=1}^N$ for the heuristic score model, the training objective is to minimize the loss between the tree with the best UAS y_i' and the tree with the best heuristic reranking score $\hat{y}_i'$.

$$J(\Theta_h) = \frac{1}{|D_h|} \sum_{(x_i,y_i',\hat{y}_i')\in D_h} r_i(\Theta_h) + \frac{\lambda}{2}||\Theta_h|| \tag{20}$$

$$r_i(\Theta_h) = \max(0, s_t(x_i, \hat{y}_i', \Theta_h)) - s_t(x_i, y_i', \Theta_h) \tag{21}$$

The detailed training algorithm is given by Algorithm 1. AdaGrad (Duchi et al., 2011) updating with subgradient (Ratliff et al., 2007) and minibatch is adopted for optimization.

5 Experiments

5.1 Set-up

Our experiments are performed using the English Penn Treebank (PTB; Marcus et al., (1993)). We follow the standard splits of PTB3, using sections 2-21 for training, section 22 for development and section 23 for final testing. Following prior work on reranking, we use Penn2Malt[1] to convert constituent trees to dependency trees. Ten-fold POS jackknifing is used in the training of the baseline parser. We use the POS-tagger of Collins (2002) to assign POS automatically. Because our reranking model is a dynamic reranking model, which generates training instances during search, we train 10 baseline parsing models on the 10-fold jackknifing data, and load the baseline parser model dynamically for reranking training .

We follow Chen and Manning (2014), using the set of pre-trained word embeddings with a dictionary size of 13,000[2] from Collobert et al. (2011). The word embeddings were trained on the entire English Wikipedia, which contains about 631 million words.

5.2 Hyper-parameters

There are two different networks in our system, namely a hierarchical feed-forward neural network for the baseline parsing and a recursive convolution network for dynamic reranking. The hyper-parameters of the hierarchical parser are set as described by Chen and Manning (2014), with the embedding size $d = 50$, the hidden layer size $d_h = 300$, the regularization parameter $\lambda = 10^{-8}$, the initial learning rate of Adagrad $\alpha = 0.01$ and the batch size $b = 100{,}000$. We set the hyper-parameters of the RCNN as follows: word embedding size $d^w_{rnn} = 25$, distance embedding size $d^d_{rnn} = 25$, initial learning rate of Adagrad $\alpha_{rnn} = 0.1$, regularization parameter $\lambda_{rnn} = 10^{-4}$, margin loss discount $\kappa = 0.1$ and revising factor $f = 8$.

5.3 The Hierarchical Neural Parser

Shown in Table 3, the proposed hierarchical base parser is 1.3 times faster, and obtains a slight accuracy improvement (Table 3) upon the parser of Chen and Manning (2014). The reason for the

[1]http://stp.lingfil.uu.se/ nivre/research/Penn2Malt.html
[2]http://ronan.collobert.com/senna/

Parser	dev		test		Speed
	UAS	LAS	UAS	LAS	
hiero	92.28	91.15	91.83	90.76	884.7
original	92.00	90.89	91.67	90.62	682.3

Table 3: Performance comparison between the hierarchical and original neural parsers. Speed: sentences per second.

Beam Size	1	2	4	8
UAS	93.38	93.45	93.81	93.51
Oracle	96.95	97.29	97.80	97.81
K	22.57	37.16	65.8	118.7

Table 4: Accuracies of the revising reranker with different beam sizes on the development set.

Revising Depth	1	2	3	4
UAS	93.22	93.50	93.81	93.53
Oracle	96.31	97.57	97.80	97.81
K	8.87	38.45	65.8	90.28

Table 5: Accuracies of the revised reranker with different revising depths on development set.

Search Type	UAS	+UAS	Oracle
Dynamic	93.81	+1.53	97.80
Static	93.29	+1.01	97.61

Table 6: Comparing dynamic and the static search.

speed gain is that smaller output layer leads to less computation of mapping from the hidden layer to the output layer in neural networks (Morin and Bengio, 2005; Mnih and Hinton, 2009).

5.4 Development Tests

For the beam search dynamic reranking model, the selection of beam size b and revising depth l affect the accuracy and efficiency of the reranker. We tune the values on the development set.

Beam Size A proper beam size balances efficiency and accuracy in the search process. The reranking accuracies with different beam sizes are listed in Table 4. Here, the oracle is the best UAS among searched trees during reranking. K is the number of searched candidate trees in testing. The UAS and parsing oracle both go up with increasing the beam size. Reranking with beam size = 4 gives the best development performance. We set the final beam size as 4 in the next experiments.

Revising Depth As shown in Table 5, with revising depth increasing from 1 to 3, the reranker obtains better parsing oracle. The depth of 3 gives the best UAS 93.81% on the development set. The parsing oracle stops improving with deeper revised search. This may because in the fourth search step, the high quality trees begin to fall out the beam, resulting in worse output candidates, which make the revising step yield less oracle gains. We set the search depth as 3 in the next experiments.

Integrating Search and Learning Shown in Table 6, the dynamic and static rerankers both achieve significant accuracy improvements over the baseline parser. The dynamic reranker gives much better improvement, although the oracle of dynamic reranker is only 0.2% higher than the static one. This demostrates the benefit of diversity. The candidates are always the same for static search, but the dynamic reranker searches more diverse tree candidates in different iterations of training.

To further explore the impact of training diversity to dynamic reranking, we also compare the dynamic search reranker of training and testing with different revising depth. In Table 7, *origin* is the results by training and testing with the same depth d. Results of *ts* is obtained by training with d = 3, and testing with a smaller d. For example, a reranker with training d = 3 and testing d = 2 achieves better performance than with training d = 2 and testing d = 2. The testing oracle of the former reranker is lower than the later, yet the former learns more from the training instance, obtaining better parsing accuracies. This again indicates that training diversity is very important besides the oracle accuracy.

Interpolated Reranker Finally, we mix the baseline model score and the reranking score by following Hayashi et al. (2013) and Zhu et al. (2015), and the mixture parameter β is optimized by searching with the step size of 0.005. With the mixture reranking trick, the dynamic reranker obtains an accuracy of 94.08% (Table 8), with an improvement of 0.28% on the development set.

5.5 Final Results

Comparison with Dependency Rerankers In Table 9, we compare the search-based dynamic rerankers with a list of dependency rerankers. The reranking models of Hayashi et al. (2013) and Hayashi et al. (2011) are forest reranking models. Le and Zuidema (2014) and Zhu et al. (2015) are neural k-best reranking models. Our dynamic

Depth		1	2	3
ordinary	UAS	93.22	93.50	93.81
	oracle	96.31	97.57	97.80
ts	UAS	93.59	93.79	93.81
	oracle	96.29	93.42	97.80

Table 7: Accuracies of the revised reranker with different revising depths on the development set.

Type	static	dynamic
w/o mixture	93.29	93.81
w/ mixture	93.53	94.08

Table 8: Effects of interpolated reranking.

reranking model achieves the highest accuracy improvement over the baseline parser on both the development and test sets. We obtain the best performance on the development set. Zhu et al. (2015) achieved higher accuracy on the test set, but they adopted a better baseline parser than ours, which could not be used in our dynamic reranker because it is not fast enough and will make our reranker slow in practice.

Comparing with Neural Dependency Parsers We also compare parsing accuracies and speeds with a number of neural network dependency parsers. Dyer et al. (2015) proposed a dependency parser with stack LSTM; Zhou et al. (2015) applied the beam search for structured dependency parsing. Both achieved significant accuracy improvements over the deterministic neural parser of Chen and Manning (2014). Our dynamic search reranker obtains a 93.61% UAS on the test set, which is higher than most of the neural parsers except Weiss et al. (2015), who employ a structured prediction model upon the neural greedy baseline, achieving very high parsing accuracy.

5.6 Results on Stanford dependencies

We also evaluate the proposed static and dynamic rerankers on Staford dependency treebank. The main results are consistent with CoNLL dependency treebank with the dynamic reranker achieving a 0.41% accuracy improvement upon the static reranker on test data. But the parsing accuracy on Stanford dependency is not the state-of-the-art. We speculate that there may be two reasons. First, the baseline parsing accuracy on Stanford dependencies is lower than CoNLL. Second, all the hyper-parameters are tuned on the CoNLL data.

Reranker		UAS	
		dev	test
Hayashi et al. (2011)		N/A	92.87 (+0.97)
Hayashi et al. (2013)		N/A	93.12 (+0.62)
Le and Zuidema (2014)		N/A	93.12 (+1.09)
(Zhu et al., 2015)	baseline	92.45	92.35
	reranking	93.50 (+1.05)	**93.83** (+1.48)
This work (CoNLL)	baseline	92.28	91.83
	dynamic	**94.08** (**+1.80**)	93.61 (**+1.78**)
	static	93.53 (+1.25)	93.17 (+1.34)

Table 9: Comparison of dependency rerankers.

6 Related Work

Neural Networks Reranking A line of work has been proposed to explore reranking using neural networks. Socher et al. (2013) first proposed a neural reranker using a recursive neural network for constituent parsing. Le and Zuidema (2014) extended the neural reranker to dependency parsing using a inside-outside recursive neural network (IORNN), which can process trees both bottom-up and top-down. Zhu et al. (2015) proposed a RCNN method, which solved the problem of modeling k-ary parsing tree in dependency parsing. The neural rerankers are capable of capturing global syntax features across the tree. In contrast, the most non-local neural parser with LSTM (Dyer et al., 2015) cannot exploit global features. Different to previous neural rerankers, our work in this paper contributes on integrating search and learning for reranking, instead of proposing a new neural model.

Forest Reranking Forest reranking (Huang, 2008; Hayashi et al., 2013) offers a different way to extend the coverage of reranking candidates, with computing the reranking score in the trees forests by decomposing non-local features with cube-pruning (Huang and Chiang, 2005). In contrast, the neural reranking score encodes the whole dependency tree, which cannot be decomposed for forest reranking efficiently and accurately.

HC-Search Doppa et al. (2013) proposed a structured prediction model with HC-Search strategy and imitation learning, which is closely related to our work in spirit. They used the complete space search (Doppa et al., 2012) for sequence labeling tasks, and the whole search process halts after a specific time bound. Different from them, we propose a dynamic parsing reranking model based on the action revising process, which is a multi-step process by revising the least confident

Type	System	UAS	Speed
Neural	Zhou et al. (2015)	93.28	14.3
	Dyer et al. (2015)‡	93.30	105
	Weiss et al. (2015)†	**93.99**	N/A
	Weiss et al. (2015) semi †	**94.26**	N/A
	Pei et al. (2015)	93.29	N/A
	Chen et al. (2015)	92.60	2.7
	Chen and Manning (2014)	92.00	1013
This work	dynamic	**93.61**	16.1

Table 10: Comparison with neural parsers. Speed: sentences per second. †: results are reported on Stanford dependencies. ‡: results are run by ourself using their codes.

System	UAS	
	dev	test
baseline	91.80	91.41
dynamic	93.44 (+1.64)	92.95 (+1.57)
static	93.09 (+1.29)	92.57 (+1.16)

Table 11: Dynamic reranking results on Stanford dependencies.

actions from the base output and the search stops in a given revising depth. The dynamic reranking model concentrates on extending the training diversity and testing oracle for parsing reranking, which is built on the transition-based parsing framework.

7 Conclusion

In this paper, we proposed a search-based dynamic reranking model using a hierarchical neural base parser and a recursive convolutional neural score model. The dynamic model is the first reranker integrating search and learning for dependency parsing. It achieves significant accuracy improvement (+1.78%) upon the baseline deterministic parser. With the dynamic search process, our reranker obtains a 0.44% accuracy improvement upon the static reranker. The code of this paper can be downloaded from `http://github.com/zhouh/dynamic-reranker`.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. Xin-Yu Dai is the corresponding author of this paper. This work was supported by the Natural Science Foundation of China (61472183, 6130158, 61472191), the 863 program via 2015AA015406 and Singapore Ministratry of Education Tier 2 Grant T2MOE201301.

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*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Xinchi Chen, Yaqian Zhou, Chenxi Zhu, Xipeng Qiu, and Xuanjing Huang. 2015. Transition-based dependency parsing using two heterogeneous gated recursive neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Michael Collins and Terry Koo. 2000. Discriminative reranking for natural language parsing. *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 175–182.

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

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

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2012. Output space search for structured prediction. In *ICML*.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2013. Hc-search: Learning heuristics and cost functions for structured prediction. In *AAAI*, volume 2, page 4.

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

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 53th Annual Meeting of the Association for Computational Linguistics*.

Yoav Goldberg, Kai Zhao, and Liang Huang. 2013. Efficient implementation of beam-search incremental parsers. In *ACL (2)*, pages 628–633.

Joshua Goodman. 1998. Parsing inside-out. *arXiv preprint cmp-lg/9805007*.

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto. 2011. Third-order variational reranking on packed-shared dependency forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1479–1488. Association for Computational Linguistics.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1:139–150.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Quoc V Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*.

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

Phong Le, Willem Zuidema, and Remko Scha, 2013. *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, chapter Learning from errors: Using vector-based compositional semantics for parse reranking, pages 11–19. Association for Computational Linguistics.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252. Citeseer.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-based dependency parsing. In *Proc. of ACL*.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. 2007. (online) subgradient methods for structured prediction.

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

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

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 and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July. Association for Computational Linguistics.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. 2015. A re-ranking model for dependency parser with recursive convolutional neural network. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.