A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing

Hao Zhou†* **Yue Zhang**‡ **Shujian Huang**† **Jiajun Chen**†
†State Key Laboratory for Novel Software Technology, Nanjing University, China
‡Singapore University of Technology and Design, Singapore
{zhouh, huangsj, chenjj}@nlp.nju.edu.cn, yue_zhang@sutd.edu.sg

Abstract

Neural probabilistic parsers are attractive for their capability of automatic feature combination and small data sizes. A transition-based greedy neural parser has given better accuracies over its linear counterpart. We propose a neural probabilistic structured-prediction model for transition-based dependency parsing, which integrates search and learning. Beam search is used for decoding, and contrastive learning is performed for maximizing the sentence-level log-likelihood. In standard Penn Treebank experiments, the structured neural parser achieves a 1.8% accuracy improvement upon a competitive greedy neural parser baseline, giving performance comparable to the best linear parser.

1 Introduction

Transition-based methods have given competitive accuracies and efficiencies for dependency parsing (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Zhang and Clark, 2008; Huang and Sagae, 2010; Zhang and Nivre, 2011; Goldberg and Nivre, 2013). These parsers construct dependency trees by using a sequence of transition actions, such as SHIFT and REDUCE, over input sentences. High accuracies are achieved by using a linear model and millions of binary indicator features. Recently, Chen and Manning (2014) propose an alternative dependency parser using a neural network, which represents atomic features as dense vectors, and obtains feature combination automatically other than devising high-order features manually.

The greedy neural parser of Chen and Manning (2014) gives higher accuracies compared to the greedy linear MaltParser (Nivre and Scholz, 2004), but lags behind state-of-the-art linear systems with sparse features (Zhang and Nivre, 2011), which adopt global learning and beam search decoding (Zhang and Nivre, 2012). The key difference is that Chen and Manning (2014) is a *local classifier* that greedily optimizes each action. In contrast, Zhang and Nivre (2011) leverage a *structured-prediction* model to optimize whole sequences of actions, which correspond to tree structures.

In this paper, we propose a novel framework for *structured* neural probabilistic dependency parsing, which maximizes the likelihood of action sequences instead of individual actions. Following Zhang and Clark (2011), beam search is applied to decoding, and global structured learning is integrated with beam search using early-update (Collins and Roark, 2004). Designing such a framework is challenging for two main reasons:

First, applying global structured learning to transition-based neural parsing is non-trivial. A direct adaptation of the framework of Zhang and Clark (2011) under the neural probabilistic model setting does not yield good results. The main reason is that the parameter space of a neural network is much denser compared to that of a linear model such as the structured perceptron (Collins, 2002). Due to the dense parameter space, for neural models, the scores of actions in a sequence are relatively more dependent than that in the linear models. As a result, the log probability of an action sequence can not be modeled just as the sum of log probabilities of each action in the sequence, which is the case of structured linear model. We address the challenge by using a *softmax* function to directly model the distribution of action sequences.

Second, for the structured model above, maximum-likelihood training is computationally intractable, requiring summing over all possible action sequences, which is difficult for transition-

*Work done while the first author was visiting SUTD.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 1213–1222, Beijing, China, July 26-31, 2015.

based parsing. To address this challenge, we take a contrastive learning approach (Hinton, 2002; LeCun and Huang, 2005; Liang and Jordan, 2008; Vickrey et al., 2010; Liu and Sun, 2014). Using the sum of log probabilities over the action sequences in the beam to approximate that over all possible action sequences.

In standard PennTreebank (Marcus et al., 1993) evaluations, our parser achieves a significant accuracy improvement (+1.8%) over the greedy neural parser of Chen and Manning (2014), and gives the best reported accuracy by shift-reduce parsers. The incremental neural probabilistic framework with global contrastive learning and beam search could be used in other structured prediction tasks.

2 Background

2.1 Arc-standard Parsing

Transition-based dependency parsers scan an input sentence from left to right, and perform a sequence of transition actions to predict its parse tree (Nivre, 2008). In this paper, 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 input 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. The process repeats until the input queue is empty and stack contains only one dependency tree.

Formally, a parsing state is denoted as $\langle j, S, L\rangle$, where S is a stack of subtrees $[\dots s_2,\ s_1,\ s_0]$, 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. At each step, the parser chooses one of the following actions:

- SHIFT: move the front word w_j from the queue onto the stacks.
- LEFT-ARC(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-ARC(l): 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.

The arc-standard parser can be summarized as the deductive system in Figure 1, where k denotes the current parsing step. For a sentence with size n, parsing stops after performing exactly $2n-1$ actions.

input : $w_0 \dots w_{n-1}$
axiom : $0 : \langle 0, \phi, \phi, 0\rangle$
goal : $2n-1 : \langle n, s_0, L\rangle$

$$\text{SHIFT}\quad \frac{k : \langle j, S, L\rangle}{k+1 : \langle j+1, S|w_j, L\rangle}$$

$$\text{LEFT-ARC}(l)\quad \frac{k : \langle j, S|s_1|s_0, L\rangle}{k+1 : \langle j, S|s_0, L \cup \{s_1 \overset{l}{\leftarrow} s_0\}\rangle}$$

$$\text{RIGHT-ARC}(l)\quad \frac{k : \langle j, S|s_1|s_0, L\rangle}{k+1 : \langle j, S|s_1, L \cup \{s_1 \overset{l}{\rightarrow} s_0\}\rangle}$$

Figure 1: The deductive system for arc-standard dependency parsing.

MaltParser uses an SVM classifier for deterministic arc-standard parsing. At each step, MaltParser generates a set of successor states according to the current state, and deterministically selects the highest-scored one as the next state.

2.2 Global Learning and Beam Search

The drawback of deterministic parsing is error propagation. An incorrect action will have a negative influence to its subsequent actions, leading to an incorrect output parse tree.

To address this issue, global learning and beam search (Zhang and Clark, 2011; Bohnet and Nivre, 2012; Choi and McCallum, 2013) are used. Given an input x, the goal of decoding is to find the highest-scored action sequence *globally*.

$$y = \underset{y' \in \text{GEN}(x)}{\arg\max}\ score(y') \qquad (1)$$

Where $\text{GEN}(x)$ denotes all possible action sequences on x, which correspond to all possible parse trees. The score of an action sequence y is:

$$score(y) = \sum_{a \in y} \theta \cdot \Phi(a) \qquad (2)$$

Here a is an action in the action sequence y, Φ is a feature function for a, and θ is the parameter vector of the linear model. The score of an action sequence is the linear sum of the scores of each action. During training, action sequence scores are globally learned.

The parser of Zhang and Nivre (2011) is developed using this framework. The structured perceptron (Collins, 2002) with early update (Collins and Roark, 2004) is applied for training. By utilizing rich manual features, it gives state-of-the-art accuracies in standard Penn Treebank evaluation. We take this method as one baseline.

2.3 Greedy Neural Network Model

Chen and Manning (2014) build a greedy neural arc-standard parser. The model can be regarded as an alternative implementation of MaltParser, using a feedforward neural network to replace the SVM classifier for deterministic parsing.

2.3.1 Model

The greedy neural model extracts n atomic features from a parsing state, which consists of words, POS-tags and dependency labels from the stack ans queue. *Embeddings* are used to represent word, POS and dependency label atomic features. Each embedding is represented as a d-dimensional vector $e_i \in \mathbb{R}$. Therefore, the full embedding matrix is $E \in \mathbb{R}^{d \times V}$, where V is the number of distinct features. A *projection layer* is used to concatenate the n input embeddings into a vector $x = [e_1; e_2 \dots e_n]$, where $x \in \mathbb{R}^{d \cdot n}$. The purpose of this layer is to fine-tune the embedding features. Then x is mapped to a d_h-dimensional *hidden layer* by a mapping matrix $W_1 \in \mathbb{R}^{d_h \times d \cdot n}$ and a cube activation function:

$$h = (W_1 x + b_1)^3 \tag{3}$$

Finally, h is mapped into a *softmax output layer* for modeling the probabilistic distribution of candidate shift-reduce actions:

$$p = softmax(o) \tag{4}$$

where

$$o = W_2 h \tag{5}$$

$W_2 \in \mathbb{R}^{d_o \times d_h}$ and d_o is the number of shift-reduce actions.

2.3.2 Features

One advantage of Chen and Manning (2014) is that the neural network parser achieves feature combination automatically. Their atomic features are defined by following Zhang and Nivre (2011). As shown in Table 1, the features are categorized into three types: F^w, F^t, F^l, which represents word features, POS-tag features and dependency label features, respectively.

	Templates
F^w	$s_0w, s_2w, q_0w, q_1w, q_2w, lc_1(s_0)w, lc_2(s_0)w$ $s_1w, rc_2(s_0)w, lc_1(s_1)w, lc_2(s_1)w, rc_2(s_1)w$ $rc_1(s_0)w, rc_1(s_1)w, lc_1(lc_1(s_0))w, lc_1(lc_1(s_1))w$ $rc_1(rc_1(s_1))w, rc_1(rc_1(s_0))w$
F^t	$s_0t, q_0t, q_1t, q_2t, rc_1(s_0)t, lc_1(s_0)t, lc_2(s_0)t$ $s_1t, s_2t, lc_1(s_1)t, lc_2(s_1)t, rc_1(s_1)t, rc_2(s_0)t$ $rc_2(s_1)t, lc_1(lc_1(s_0))t, lc_1(lc_1(s_1))t$ $rc_1(rc_1(s_0))t, rc_1(rc_1(s_1))t$
F^l	$rc_1(s_0)l, lc_1(s_0)l, lc_2(s_0)l, lc_1(s_1)l, lc_2(s_1)l$ $rc_1(s_1)l, rc_2(s_0)l, rc_2(s_1)l, lc_1(lc_1(s_0))l$ $lc_1(lc_1(s_1))l, rc_1(rc_1(s_0))l, rc_1(rc_1(s_1))l$

Table 1: Feature templates.

For example, s_0w and q_0w represent the first word on the stack and queue, respectively; $lc_1(s_0)w$ and $rc_1(s_0)w$ represent the leftmost and rightmost child of s_0, respectively. Similarly, $lc_1(s_0)t$ and $lc_1(s_0)l$ represent the POS-tag and dependency label of the leftmost child of s_0, respectively.

Chen and Manning (2014) find that the cube activation function in Equation (3) is highly effective in capturing feature interaction, which is a novel contribution of their work. The cube function achieves linear combination between atomic word, POS and label features via the product of three element combinations. Empirically, it works better compared to a sigmoid activation function.

2.4 Training

Given a set of training examples, the training objective of the greedy neural parser is to minimize the cross-entropy loss, plus a l_2-regularization term:

$$L(\theta) = -\sum_{i \in A} \log p_i + \frac{\lambda}{2} \parallel \theta \parallel^2 \tag{6}$$

θ is the set of all parameters (i.e. W_1, W_2, b, E), and A is the set of all gold actions in the training data. AdaGrad (Duchi et al., 2011) with mini-batch is adopted for optimization. We take the greedy neural parser of Chen and Manning (2014) as a second baseline.

3 Structured Neural Network Model

We propose a neural structured-prediction model that scores whole sequences of transition actions, rather than individual actions. As shown in Table 2, the model can be seen as a neural probabilistic alternative of Zhang and Nivre (2011),

	local classifier	structured prediction
linear sparse	Section 2.1 (Nivre et al., 2007)	Section 2.2 (Zhang and Nivre, 2011)
neural dense	Section 2.3 (Chen and Manning, 2014)	this work

Table 2: Correlation between different parsers.

or a structured-prediction alternative of Chen and Manning (2014). It combines the advantages of both Zhang and Nivre (2011) and Chen and Manning (2014) over the greedy linear MaltParser.

3.1 Neural Probabilistic Ranking

Given the baseline system in Section 2.2, the most intuitive structured neural dependency parser is to replace the linear scoring model with a neural probabilistic model. Following Equation 1, the score of an action sequence y, which corresponds to its log probability, is sum of log probability scores of each action in the sequence.

$$s(y) = \sum_{a \in y} \log p_a \tag{7}$$

where p_a is defined by the baseline neural model of Section 2.3 (Equation 4). The training objective is to maximize the score margin between the gold action sequences (y_g) and these of incorrectly predicated action sequences (y_p):

$$L(\theta) = max(0, \delta - s(y_g) + s(y_p)) + \frac{\lambda}{2} \parallel \theta \parallel^2 \tag{8}$$

With this ranking model, beam search and early-update are used. Given a training instance, the negative example is the incorrectly predicted output with largest score (Zhang and Nivre, 2011).

However, we find that the ranking model works poorly. One explanation is that the actions in a sequence is probabilistically dependent on each other, and therefore using the total log probabilities of each action to compute the log probability of an action sequence (Equation 7) is inaccurate. Linear models do not suffer from this problem, because the parameter space of linear models is much more sparse than that of neural models. For neural networks, the dense parameter space is shared by all the actions in a sequence. Increasing the likelihood of a gold action may also change the likelihood of incorrect actions through the shared parameters. As a result, increasing the scores of a gold action sequence and simultaneously reducing the scores of an incorrect action sequence does not work well for neural models.

3.2 Sentence-Level Log-Likelihood

To overcome the above limitation, we try to directly model the probabilistic distribution of whole action sequences. Given a sentence x and neural networks parameter θ, the probability of the action sequence y_i is given by the *softmax* function:

$$p(y_i \mid x, \theta) = \frac{e^{f(x,\,\theta)_i}}{\sum\limits_{y_j \in \mathrm{GEN}(x)} e^{f(x,\,\theta)_j}} \tag{9}$$

where

$$f(x,\,\theta)_i = \sum_{a_k \in y_i} o(x,\,y_i,\,k,\,a_k) \tag{10}$$

Here $\mathrm{GEN}(s)$ is the set of all possible valid action sequences for a sentence x; $o(x,\,y_i,\,k,\,a_k)$ denotes the neural network score for the action a_k given x and y_i. We use the same sub network as Chen and Manning (2014) to calculate $o(x,\,y_i,\,k,\,a_k)$ (Equation 5). The same features in Table 1 are used.

Given the training data as $(X,\,Y)$, our training objective is to minimize the negative log-likelihood:

$$L(\theta) = -\sum_{(x_i,\,y_i) \in (X,Y)} \log p(y_i \mid x_i, \theta) \tag{11}$$

$$= -\sum_{(x_i,\,y_i) \in (X,Y)} \log \frac{e^{f(x_i,\theta)_i}}{Z(x_i,\theta)} \tag{12}$$

$$= \sum_{(x_i,\,y_i) \in (X,Y)} \log Z(x_i,\theta) - f(x_i,\theta)_i \tag{13}$$

where

$$Z(x,\theta) = \sum_{y_j \in \mathrm{GEN}(x)} e^{f(x,\,\theta)_j} \tag{14}$$

Here, $Z(x,\theta)$ is called the *partition function*. Following Chen and Manning(2014), we apply l_2-regularization for training.

For optimization, we need to compute gradients for $L(\theta)$, which includes gradients of exponential

numbers of negative examples in *partition function* $Z(x, \theta)$. However, beam search is used for transition-based parsing, and no efficient optimal dynamic program is available to estimate $Z(x, \theta)$ accurately. We adopt a novel contrastive learning approach to approximately compute $Z(x, \theta)$.

3.3 Contrastive Learning

As an alternative to maximize the likelihood on some observed data, contrastive learning (Hinton, 2002; LeCun and Huang, 2005; Liang and Jordan, 2008; Vickrey et al., 2010; Liu and Sun, 2014) is an approach that assigns higher probabilities to observed data and lower probabilities to noisy data.

We adopt the contrastive learning approach, assigning higher probabilities to the gold action sequence compared to incorrect action sequences in the beam. Intuitively, this method only penalizes incorrect action sequences with high probabilities. Our new training objective is approximated as:

$$L'(\theta) = -\sum_{(x_i, y_i) \in (X,Y)} \log p'(y_i \mid x_i, \theta) \quad (15)$$

$$= -\sum_{(x_i, y_i) \in (X,Y)} \log \frac{e^{f(x_i,\theta)_i}}{Z'(x_i, \theta)} \quad (16)$$

$$= \sum_{(x_i, y_i) \in (X,Y)} \log Z'(x_i, \theta) - f(x_i, \theta)_i \quad (17)$$

where

$$Z'(x, \theta) = \sum_{y_j \in \text{BEAM}(x)} e^{f(x,\,\theta)_j} \quad (18)$$

$p'(y_i \mid x, \theta)$ is the relative probability of the action sequence y_i, computed over only the action sequences in the beam. $Z'(x, \theta)$ is the contrastive approximation of $Z(x, \theta)$. $\text{BEAM}(x)$ returns the predicated action sequences in the beam and the gold action sequence.

We assume that the probability mass concentrates on a relatively small number of action sequences, which allows the use of a limited number of probable sequences to approximate the full set of action sequences. The concentration may be enlarged dramatically with an exponential activation function of the neural network (i.e. $a > b \Rightarrow e^a \gg e^b$).

3.4 The Neural Probabilistic Structured-Prediction Framework

We follow Zhang and Clark (2011) to integrate search and learning. Our search and learning framework for dependency parsing is shown as Algorithm 1. In every training iteration i, we randomly sample the training instances, and perform online learning with early update (Collins and Roark, 2004). In particular, given a training example, we use beam-search to decode the sentence. At any step, if the gold action sequence falls out of the beam, we take all the incorrect action sequences in the beam as negative examples, and the current gold sequence as a positive example for parameter update, using the training algorithm of Section 3.3. AdaGrad algorithm (Duchi et al., 2011) with mini-batch is adopted for optimization.

Algorithm 1: Training Algorithm for Structured Neural Parsing

```
Input: training examples (X, Y)
Output: θ
θ ← pretrained embedding
for i ← 1 to N do
    x, y = RANDOMSAMPLE(X, Y)
    δ = 0
    foreach x_j, y_j ∈ x, y do
        beam = φ
        goldState = null
        terminate = false
        beamGold = true
        while beamGold and not terminate do
            beam = DECODE(beam, x_j, y_j)
            goldState = GOLDMOVE(goldState, x_j, y_j)
            if not ISGOLD(beam) then
                beamGold = false
            if ITEMSCOMPLETE(beam) then
                terminate = true;
        δ = δ + UPDATE(goldState, beam)
    θ = θ + delta
```

In this way, the distribution of ot only full action sequences (i.e. complete parse trees), but also partial action sequences (i.e. partial outputs) are modeled, which makes training more challenging. The advantage of early update is that training is used to guide search, minimizing search errors.

4 Experiments

4.1 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 testing and section 23 for final testing. For comparison with previous work, we use Penn2Malt[1] to convert constituent trees to dependency trees. We use the POS-tagger of Collins (2002) to assign POS automatically. 10-fold jackknifing is performed for tagging the training data.

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

4.2 WSJ Experiments

4.2.1 Development experiments

We set the following hyper-parameters according to the baseline greedy neural parser (Chen and Manning, 2014): embedding size d = 50, hidden layer size d_h = 200, regularization parameter λ = 10^{-8}, initial learning rate of Adagrad α = 0.01. For the structured neural parser, beam size and mini-batch size are important to the parsing performance. We tune them on the development set.

Beam size. Beam search enlarges the search space. More importantly, the larger the beam is, the more accurate our training algorithm is. the Contrastive learning approximates the exact probabilities over exponential many action sequences by computing the relative probabilities over action sequences in the beam (Equation 18). Therefore, the larger the beam is, the more accurate the relative probability is.

The first column of Table 3 shows the accuracies of the structured neural parser on the development set with different beam sizes, which improves as the beam size increases. We set the final beam size as 100 according to the accuracies on development set.

The effect of integrating search and learning. We also conduct experiments on the parser of Chen and Manning (2014) with beam search decoding. The score of a whole action sequence is computed by the sum of log action probabilities (Equation 7). As shown in the second column of Table 3, beam search can improve parsing slightly. When the beam size increases beyond 16, however, accuracy improvements stop. In contrast, by integrating beam search and global learning, our parsing performance benefits from large beam sizes much more significantly. With a beam size as 16, the structured neural parser gives an accuracy close to that of baseline greedy parser[3]. When the beam size is 100, the structured neural parser outperforms baseline by 1.6%.

Zhang and Nivre (2012) find that global learning and beam search should be used jointly for improving parsing using a *linear* transition-based model. In particular, increasing the beam size, the accuracy of ZPar (Zhang and Nivre, 2011) increases significantly, but that of MaltParser does not. For structured *neural* parsing, our finding is similar: integrating search and learning is much more effective than using beam search only in decoding.

Our results in Table 3 are obtained by using the same beam sizes for both training and testing. Zhang and Nivre (2012) also find that for their lin-

Description	UAS	
Baseline	91.63	
	structured	greedy
beam = 1	74.90	91.63
beam = 4	84.64	91.92
beam = 16	91.53	91.90
beam = 64	93.12	91.84
beam = 100	93.23	91.81

Table 3: Accuracies of structured neural parsing and local neural classification parsing with different beam sizes.

Description	UAS
greedy neural parser	91.47
ranking model	89.08
beam contrastive learning	93.28

Table 4: Comparison between sentence-level log-likelihood and ranking model.

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

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

[3] Our baseline accuracy is a little lower than accuracy reported in baseline paper (Chen and Manning, 2014), because we use Penn2Malt to convert the Penn Treebank, and they use LTH Conversion.

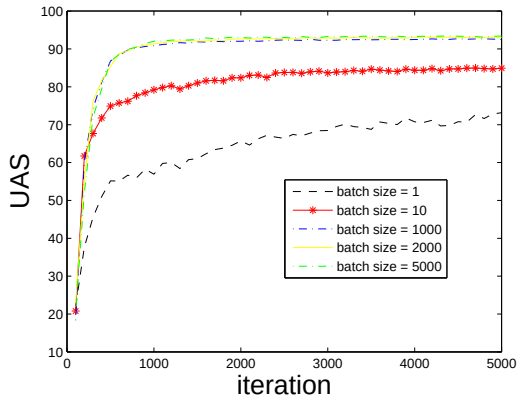

Figure 2: Parsing performance with different training batch sizes.

System	UAS	LAS	Speed
baseline greedy parser	91.47	90.43	0.001
Huang and Sagae (2010)	92.10		0.04
Zhang and Nivre (2011)	92.90	91.80	0.03
Choi and McCallum (2013)	92.96	91.93	0.009
Ma et al. (2014)	93.06		
Bohnet and Nivre (2012)†‡	93.67	92.68	0.4
Suzuki et al. (2009)†	93.79		
Koo et al. (2008)†	93.16		
Chen et al. (2014)†	93.77		
beam size			
training / decoding			
100 / 100	**93.28**	**92.35**	0.07
100 / 64	93.20	92.27	0.04
100 / 16	92.40	91.95	0.01

Table 5: Results on WSJ. Speed: sentences per second. †: semi-supervised learning. ‡: joint POS-tagging and dependency parsing models.

ear model, the best results are achieved by using the same beam sizes during training and testing. We find that this observation does not apply to our neural parser. In our case, a large training beam always leads to better results. This is likely because a large beam improves contrastive learning. As a result, our training beam size is set to 100 for the final test.

Batch size. Parsing performance using neural networks is highly sensitive to the batch size of training. In greedy neural parsing (Chen and Manning, 2014), the accuracy on the development data improves from 85% to 91% by setting the batch size to 10 and 100000, respectively. In structured neural parsing, we fix the beam size as 100 and draw the accuracies on the development set by the training iteration.

As shown in Figure 2, in 5000 training iterations, the parsing accuracies improve as the iteration grows, yet different batch sizes result in different convergence accuracies. With a batch size of 5000, the parsing accuracy is about 25% higher than with a batch size of 1 (i.e. SGD). For the remaining experiments, we set batch size to 5000, which achieves the best accuracies on development testing.

4.2.2 Sentence-level maximum likelihood vs. ranking model

We compare parsing accuracies of the sentence-level log-likelihood + beam contrastive learning (Section 3.2), and the structured neural parser with probabilistic ranking (Section 3.1). As shown in Table 4, performance of global learning with ranking model is weaker than the baseline greedy parser. In contrast, structured neural parsing with sentence-level log-likelihood and contrastive learning gives a 1.8% accuracy improvement upon the baseline greedy parser.

As mentioned in Section 3.1, a likely reason for the poor performance of the structured neural ranking model may be that, the likelihoods of action sequences are highly influenced by each other, due to the dense parameter space of neural networks. To maximize likelihood of gold action sequence, we need to decrease the likelihoods of more than one incorrect action sequences.

4.2.3 Final Results

Table 5 shows the results of our final parser and a line of transition-based parsers on the test set. Our structured neural parser achieves an accuracy of 93.28%, 0.38% higher than Zhang and Nivre (2011), which employees millions of high-order binary indicator features in parsing. The model size of ZPar (Zhang and Nivre, 2011) is over 250 MB on disk. In contrast, the model size of our structured neural parser is only 25 MB. To our knowledge, the result is the best reported result achieved by shift-reduce parsers on this data set.

Bohnet and Nivre (2012) obtain an accuracy of 93.67%, which is higher than our parser. However, their parser is a joint model of parsing and POS-tagging, and they use external data in parsing. We also list the result of Chen et al. (2014), Koo et al. (2008) and Suzuki et al. (2009) in Table 5, which make use of large-scale unannotated text to improve parsing accuracies. The input embeddings of our parser are also trained

over large raw text, and in this perspective our model is correlated with the semi-supervised models. However, because we fine-tune the word embeddings in supervised training, the embeddings of in-vocabulary words become systematically different from these of out-of-vocabulary words after training, and the effect of pre-trained out-of-vocabulary embeddings become uncertain. In this sense, our model can also be regarded as an almost fully supervised model. The same applies to the models of Chen and Manning (2014).

We also compare the speed of the structured neural parser on an Intel Core i7 3.40GHz CPU with 16GB RAM. The structured neural parser runs about as fast as Zhang and Nivre (Zhang and Nivre, 2011) and Huang and Sagae (Huang and Sagae, 2010). The results show that our parser combines the benefits of structured models and neural probabilistic models, offering high accuracies, fast speed and slim model size.

5 Related Work

Parsing with neural networks. A line of work has been proposed to explore the effect of neural network models for constituent parsing (Henderson, 2004; Mayberry III and Miikkulainen, 2005; Collobert, 2011; Socher et al., 2013; Legrand and Collobert, 2014). Performances of most of these methods are still well below the state-of-the-art, except for Socher et al.(2013), who propose a neural reranker based on a PCFG parser. For transition-based dependency parsing, Stenetorp (2013) applies a compositional vector method (Socher et al., 2013), and Chen and Manning (2014) propose a feed-forward neural parser. The performances of these neural parsers lag behind the state-of-the-art.

More recently, Dyer et al. (2015) propose a greedy transition-based dependency parser, using three stack LSTMs to represent the input, the stack of partial syntactic trees and the history of parse actions, respectively. By modeling more history, the parser gives significant better accuracies compared to the greedy neural parser of Chen and Manning (2014).

Structured neural models. Collobert et al. (2011) presents a unified neural network architecture for various natural language processing (NLP) tasks. They propose to use sentence-level log-likelihood to enhance a neural probabilistic model, which inspires our model. Sequence labeling is used for graph-based decoding. Using the Viterbi algorithm, they can compute the exponential partition function in linear time without approximation. However, with a dynamic programming decoder, their sequence labeling model can only extract local features. In contrast, our integrated approximated search and learning framework allows rich global features.

Weiss et al. (2015) also propose a structured neural transition-based parser by adopting beam search and early updates. Their model is close in spirit to ours in performing structured prediction using a neural network. The main difference is that their structured neural parser uses a greedy parsing process for pre-training, and fine-tunes an additional perceptron layer consisting of the pre-trained hidden and output layers using structured perceptron updates. Their structured neural parser achieves an accuracy of 93.36% on Stanford conversion of the PTB, which is significant higher than the baseline parser of Chen and Manning (2014). Their results are not directly comparable with ours due to different dependency conversions.

6 Conclusion

We built a structured neural dependency parsing model. Compared to the greedy neural parser of Chen and Manning (2014), our parser integrates beam search and global contrastive learning. In standard PTB evaluation, our parser achieved a 1.8% accuracy improvement over the parser of Chen and Manning (2014), which shows the effect of combining search and learning. To our knowledge, the structured neural parser is the first neural parser that outperforms the best linear shift-reduce dependency parsers. The structured neural probabilistic framework can be used in other incremental structured prediction tasks.

7 Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work was partially founded by the Natural Science Foundation of China (61170181, 61300158), the Jiangsu Provincial Research Foundation for Basic Research (BK20130580), Singapore Ministratry of Education Tier 2 Grant T2MOE201301 and SRGISTD2012038 from Singapore University of Technology and Design.

References

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465. 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)*.

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proceedings of the international conference on Computational linguistics*. Association for Computational Linguistics.

Jinho D Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *ACL (1)*, pages 1052–1062.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.

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.

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

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 ACL conference*. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 95. Association for Computational Linguistics.

Geoffrey Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

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

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing.

Yann LeCun and F Huang. 2005. Loss functions for discriminative training of energybased models. AIStats.

J. Legrand and R. Collobert. 2014. Recurrent greedy parsing with neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*.

Percy Liang and Michael I Jordan. 2008. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proceedings of the 25th international conference on Machine learning*, pages 584–591. ACM.

Yang Liu and Maosong Sun. 2014. Contrastive unsupervised word alignment with non-local features. *arXiv preprint arXiv:1410.2082*.

Ji Ma, Yue Zhang, and Jingbo Zhu. 2014. Punctuation processing for projective dependency parsing. In *Proc. of ACL*.

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.

Marshall R Mayberry III and Risto Miikkulainen. 2005. Broad-coverage parsing with neural networks. *Neural Processing Letters*, 21(2):121–132.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 64–70.

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.

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.

Pontus Stenetorp. 2013. Transition-based dependency parsing using recursive neural networks. In *NIPS Workshop on Deep Learning*.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 551–560. Association for Computational Linguistics.

David Vickrey, Cliff C Lin, and Daphne Koller. 2010. Non-local contrastive objectives. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1103–1110.

David Weiss, Christopher Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the ACL conference*. Association for Computational Linguistics.

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

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

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

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

Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*, pages 1391–1400.