Evaluating a Deterministic Shift-Reduce Neural Parser for Constituent Parsing

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

Abstract

Greedy transition-based parsers are appealing for their very fast speed, with reasonably high accuracies. In this paper, we build a fast shift-reduce neural constituent parser by using a neural network to make local decisions. One challenge to the parsing speed is the large hidden and output layer sizes caused by the number of constituent labels and branching options. We speed up the parser by using a hierarchical output layer, inspired by the hierarchical log-bilinear neural language model. In standard WSJ experiments, the neural parser achieves an almost 2.4 time speed up (320 sen/sec) compared to a non-hierarchical baseline without significant accuracy loss (89.06 vs 89.13 F-score).

Keywords: Hierarchical Model, Neural Networks, Parsing

1. Introduction

With the growth of demands for processing massive amounts of web data, the speed of NLP systems has become a key practical performance.

Transition-based parsers build outputs by using a sequence of shift-reduce actions (Sagae and Lavie, 2005; Zhang and Nivre, 2011; Bohnet and Nivre, 2012; Zhou et al., 2015). They are attractive by their fast speed, offering linear time complexity with deterministic decoding algorithms. Linear classifiers have traditionally been used to build very fast deterministic parsers (Nivre, 2008; Goldberg and Nivre, 2013).

Recently, Chen and Manning (2014) proposed a transition-based dependency parser, which adopts a neural network as the classifier. It has two advantages compared to the traditional linear model (Nivre, 2008), namely fast speed and automatic feature combination. In particular, the speed of a neural dependency parser can be 1000 sentences per second on a single CPU, which is significant faster than traditional parsers.

In this paper, we explore the method of Chen and Manning (2014) for fast deterministic transition-based *constituent* parsing. This is a more challenging task compared with neural dependency parsing (Table 1). First, due to unary actions and temporary nodes caused by binarization, the number of actions in a transition-based constituent parser is much larger than in a dependency parser, resulting in a large output layer for the neural network. Second, we find empirically that the neural constituent parser needs a much larger hidden layer (i.e. 800) to obtain reasonable performance compared to the neural dependency parser (i.e. 200). As a result, the speed of neural constituent parser is much slower than the dependency counterpart.

Motivated by hierarchical log-bilinear neural language models (Morin and Bengio, 2005; Mnih and Hinton, 2009), we propose a hierarchical-output neural model to speed up the parser. A hierarchical neural language model divides a huge vocabulary into a hierarchy of clusters, and constructs a binary tree upon the word clusters. Each word corresponds to a leaf node, and can be uniquely specified by the path from the root.

	Hidden	Action	Computation
Dependency	200	27	5,400
Constituent	800	131	104,800

Table 1: Comparison between neural constituent and dependency parsing. Computation: the product of hidden size and action number.

The actions of constituent parsing are natively hierarchical based on the action structure. We decompose the output layer of the neural network into a two-layer hierarchical softmax, the action layer and the label layer. In deterministic parsing, we first find the optimal shift-reduce action type, which resolves structural ambiguities, and then find the optimal constituent label given the action type. With this hierarchical neural network, the computation cost from the hidden layer to the output layer is effectively reduced.

In our experiments, the hierarchical neural constituent parser achieves almost 2.4 times the speed of the baseline non-hierarchical neural parser, without significant accuracy loss. Our final parser runs at 320 sentences per second on a Intel(R) Core(TM) i7-4790 CPU, giving an accuracy of 89.06 on the standard Penn Treebank benchmark.

2. Shift-Reduce Constituent Parsing

Transition-based parsers parse a sentence by performing a sequence of shift-reduce actions. At each step, the action to be taken is determined by a statistical classifier. We adopt the model of Sagae and Lavie (2005), which uses a stack to hold partial outputs, and a queue to maintain the next incoming words in the input sentence.

Formally, a parsing state is $\langle S, Q \rangle$, where S is the stack of partial outputs $[\dots, s_2, s_1, s_0]$ and Q is the queue of incoming words $[q_0, q_1, q_2, \dots]$. A shift-reduce constituent parser starts with an initial state and makes transitions from one state to another by taking actions. Parsing stops when Q is empty and S contains only one tree. Given an input sentence W, the initial state is $\langle \phi, W \rangle$ and the final state is $\langle [s_0], \phi \rangle$. At each step, one of the following actions is applied:

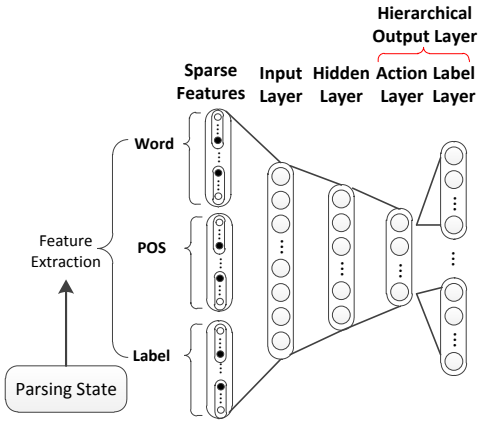

Figure 1: Hierarchical softmax neural network parser.

- SHIFT: remove the first word q_0 from the queue and push it onto the stack.
- UNARY-X: pop the top node s_0 off the stack; generate a unary-branching node with constituent label X whose child is s_0; push the new node back onto the stack.
- LEFT/RIGHT-X: pop the top two nodes s_1, s_0 off the stack; generate a binary-branching node with constituent label X whose left child is s_1 and right child is s_0, with the left (LEFT)/right (RIGHT) child as its head; push the new node back to the stack.

The shift-reduce actions only build binarized trees. As a result, a binarization process is necessary to convert the Penn Treebank into binarized trees. During this process, temporary nodes are constructed. To accommodate for binarization, we follow Zhang and Clark (2009), adding counterparts to LEFT/RIGHT-X for temporary nodes, namely LEFT/RIGHT-TEMP-X.

3. Vanilla Neural Network Parsing

Following Chen and Manning (2014), we use a three-layer feed-forward neural network for deterministic parsing. At each step of deterministic parsing, the neural model extracts n atomic features from a parsing state, which consists of head words, POS-tags and constituent labels from the stack and queue. *Embeddings* are used to represent atomic features. Each embedding is 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.

Once each sparse feature is extracted, we retrieves its feature embedding e_i from E by index i. An *input layer* is used to concatenate the n feature embeddings into a vector $x = [e_{i1}; e_{i2} \dots e_{in}]$, where $x \in \mathbb{R}^{d \cdot n}$. 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 for feature combination:

$$h = (W_1 x + b_1)^3 \tag{1}$$

Finally, h is mapped into a output layer:

$$o = W_2 h \tag{2}$$

	Templates
F^w	$s_0w, s_1w, s_2w, s_3w, s_0lw, s_0rw, s_0uw$ $s_1lw, s_1rw, s_1uw, q_0w, q_1w, q_2w, q_3w$
F^t	$s_0t, s_1t, s_2t, s_3t, q_0t, q_1t, q_2t, q_3t$
F^c	$s_0c, s_1c, s_2c, s_3c, s_0lc, s_0rc, s_0uc, s_1lc$ s_1rc, s_1uc

Table 2: Feature templates.

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

Given the neural networks parameter θ, the probability of each candidate shift-reduce action a_i is calculated by a *softmax function* over the output layer o:

$$p(a_i \mid \theta) = \frac{e^{o^i}}{\sum_{a_j \in \mathrm{GEN}(Acts)} e^{o^j}} \tag{3}$$

Where o^i is the i_{th} dimension of output o, GEN$(Acts)$ returns the valid actions according to the history actions $Acts$, θ is the set of all parameters.

We employ the features used in Zhang and Clark (2009), but remove the features about brackets and separators (Table 2). Here s_i is the i^{th} node on top of the stack S and q_j is the j^{th} node in the queue Q. s_0l, s_0r represent the left and right child for a binary branching s_0, and s_0u represents the single child for a unary branching s_0. w, c and t represent the lexical head token, constituent label and lexical head's POS tag for a node, respectively.

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

$$L(\theta) = -\sum_{a \in A} \log p(a \mid \theta) + \frac{\lambda}{2} \parallel \theta \parallel^2 \tag{4}$$

Here A is the set of all gold labeled actions in the training data. Mini-bached AdaGrad (Duchi et al., 2011) and dropout (Srivastava et al., 2014) are used for optimization.

4. Hierarchical Output Neural Network

Due to its large hidden and output layer sizes, the vanilla neural constituent parser is much slower than the dependency counterpart. The main computation cost is the mapping from hidden layer to output layer.

Motivated by the hierarchical neural language model (Mnih and Hinton, 2009), we adjust the neural constituent parser by using a hierarchical output layer.

The structure is shown in Figure 1. We decompose the output layer of the baseline in two layers: the action layer and the label layer. The action layer is a six-way class between SHIFT, UNARY, LEFT, RIGHT, LEFT-TEMP and RIGHT-TEMP. A label layer selects constituent labels, such as *NP*, for the LEFT-NP, LEFT-TEMP-NP, RIGHT-NP and RIGHT-TEMP-NP actions. Except SHIFT, different action types have different label output layers.

Given the hidden layer h (Equation 1), the action type output layer o_{act} and the label output layer $o_{label}(a_i)$ of action type a_i are computed as

$$o_{act} = W_2 h \tag{5}$$

$$o_{label}(a_i) = W_3^i h \tag{6}$$

	200	500	800
baseline	85.38	86.91	87.28
hierarchical	84.70	86.82	87.23

Table 3: Results against hidden layer sizes.

$W_2 \in \mathbb{R}^{d_a \times d_h}$ is the mapping matrix from 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 label layer.

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

$$\begin{aligned} & p(y_{i,j} \mid x, Acts) \\ = \;& p(a_i \mid x, Acts) \times p(l_j \mid x, Acts, a_i) \end{aligned} \quad (7)$$

where

$$p(a_i \mid x, Acts) = \frac{e^{o^i_{act}}}{\sum_{a_k \in \mathrm{GEN}(Acts)} e^{o^k_{act}}} \quad (8)$$

$$p(l_j \mid x, Acts, a_i) = \frac{e^{o^j_{label}(a_i)}}{\sum_{l_k \in \mathrm{GEN}(Acts)} e^{o^k_{label}(a_i)}} \quad (9)$$

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

We adopt a greedy decoding strategy in the hierarchical parsing process. In each parsing step, the action type a_i with the highest probability is first selected, and then the constituent label l_j with the highest probability is selected given the optimal action type a_i.

As in the baseline parser, we adopt the cross-entropy loss as our training objective:

$$L(\theta) = -\sum_{y_{i,j} \in A} \log p(y_{i,j} \mid x, Acts) + \frac{\lambda}{2} \parallel \theta \parallel^2 \quad (10)$$

5. Experiments

5.1. Set-up

We conduct our experiments on the Wall Street Journal (WSJ) corpus of the Penn Treebank (Marcus et al., 1993). Following the standard splits of WSJ, sections 2–21 are used as the labeled training data, section 24 is used as the development data and section 23 is used as the evaluation data. Ten-fold jackknifing (Collins, 2000) is used to automatically assign POS tags to the training data. The SVM-Tool is used as the POS-tagger[1].

5.2. Parameters

We carry out a development experiment to measure the correlation between hidden layer size and constituent parsing accuracies. From Table 3, we can see that both the baseline neural parser and the hierarchical neural parser achieve higher parsing accuracies with increasing hidden larger sizes. The two neural parsers obtain converging performances with the hidden size of 800, which is larger than 200 in the dependency case.

[1] http://www.lsi.upc.edu/?nlp/SVMTool/

	Model	F1	Speed
Linear	Collins (1999)	88.2	3.5
	Charniak (2000)	89.6	5.7
	Charniak and Johnson (2005)‡	91.1	
	Huang (2008)‡	91.7	
	McClosky et al. (2006)‡	92.1	
	Shindo et al. (2012)	92.4	
	Sagae and Lavie (2005)	86.0	3.7
	Petrov and Klein (2007)	90.1	6.2
	Carreras et al. (2008)	91.1	
	Zhu et al. (2013)	89.9	100.7
	Zhu et al. (2013) + *padding*	90.4	89.5
Neural	Henderson (2004)‡	90.1	
	Titov and Henderson (2007)	90.0	
	Collobert (2011)	87.9	31.7
	Billingsley and Curran (2012)	84.9	
	Socher et al. (2013)‡	90.4	6.1
	Legrand and Collobert (2014)	88.3	22.0
	Watanabe and Sumita (2015)	90.7	1.8
	This Work	89.13	133.6
	This Work + *hierarchical*	89.06	**320.2**

Table 4: Comparisons with previous work. ‡: reranking model. Speed: sentences per second.

We also tune the other hyper parameters on the development data. For the final test, we set the parameters as follows: embedding size d = 50, regularization parameter λ = 10^{-8}, initial learning rate of Adagrad α = 0.01, batch size b = 10000.

5.3. Final Results

The final parsing result is shown in Table 4, which are grouped into 2 classes, the traditional linear parsers (**Linear**) and the neural parsers (**Neural**).

Our deterministic neural constituent parser obtains an accuracy of 89.13. Besides the reranking neural parsers, accuracy of our parser falls behind the parser of Titov and Henderson (2007), who give a high parsing accuracy by using a dynamic sigmoid belief network, and the parser of Watanabe and Sumita (2015), who use a beam search decoding. However, their parsers are much more time consuming than our parser.

The baseline neural parser runs at a speed of 133 sentences per second. The hierarchical neural constituent parser achieves a 2.4 times parsing speed up (320 sen/sec) without significant accuracy loss.

6. Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments.

This work was partially founded by the Natural Science Foundation of China (61223003, 61300158), the Jiangsu Provincial Research Foundation for Basic Research (BK20130580).

7. Conclusion

We proposed a shift-reduce neural constituent parser, using a hierarchical softmax model to improve the parsing speed.

The hierarchical neural parser runs at a speed of 320 sentences per second on a single CPU, with a reasonable accuracy. To our knowledge, this is the best speed result in the literature for constituent parsing.

The source code of our system could be found at: `github.com/zhouh/StructNNConParser`.

8. Copyrights

9. Bibliographical References

Billingsley, R. J. and Curran, J. R. (2012). Improvements to training an rnn parser.

Bohnet, B. and Nivre, J. (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.

Carreras, X., Collins, M., and Koo, T. (2008). Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16. Association for Computational Linguistics.

Charniak, E. and Johnson, M. (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.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics.

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

Collins, M. (1999). *HEAD-DRIVEN STATISTICAL MODELS FOR NATURAL LANGUAGE PARSING*. Ph.D. thesis, University of Pennsylvania.

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

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

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

Henderson, J. (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.

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

Legrand, J. and Collobert, R. (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)*.

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

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics.

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

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

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

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *HLT-NAACL*, pages 404–411.

Sagae, K. and Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

Shindo, H., Miyao, Y., Fujino, A., and Nagata, M. (2012). Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 440–448. Association for Computational Linguistics.

Socher, R., Bauer, J., Manning, C. D., and Ng, A. Y. (2013). Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Titov, I. and Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In

ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, volume 45, page 632.

Watanabe, T. and Sumita, E. (2015). Transition-based neural constituent 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 1169–1179, Beijing, China, July. Association for Computational Linguistics.

Zhang, Y. and Clark, S. (2009). Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171. Association for Computational Linguistics.

Zhang, Y. and Nivre, J. (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.

Zhou, H., Zhang, Y., Huang, S., and Chen, J. (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.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., and Zhu, J. (2013). Fast and accurate shift-reduce constituent parsing. In *51st Annual Meeting of the Association for Computational Linguistics*.