Shift-Reduce Constituent Parsing with Neural Lookahead Features

Jiangming Liu and **Yue Zhang**
Singapore University of Technology and Design,
8 Somapah Road, Singapore, 487372
{jiangming_liu, yue_zhang}@sutd.edu.sg

Abstract

Transition-based models can be fast and accurate for constituent parsing. Compared with chart-based models, they leverage richer features by extracting history information from a parser stack, which consists of a sequence of non-local constituents. On the other hand, during incremental parsing, constituent information on the right hand side of the current word is not utilized, which is a relative weakness of shift-reduce parsing. To address this limitation, we leverage a fast neural model to extract lookahead features. In particular, we build a bidirectional LSTM model, which leverages full sentence information to predict the hierarchy of constituents that each word starts and ends. The results are then passed to a strong transition-based constituent parser as lookahead features. The resulting parser gives 1.3% absolute improvement in WSJ and 2.3% in CTB compared to the baseline, giving the highest reported accuracies for fully-supervised parsing.

1 Introduction

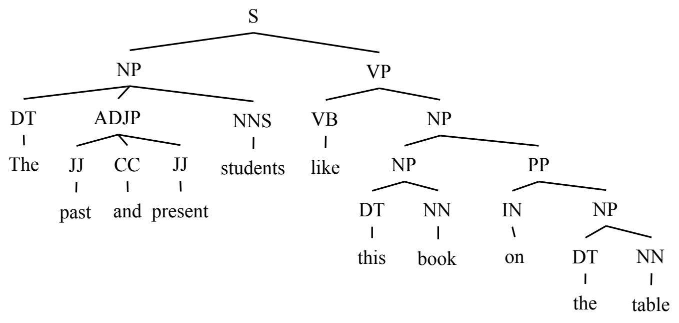
Transition-based constituent parsers are fast and accurate, performing incremental parsing using a sequence of state transitions in linear time. Pioneering models rely on a classifier to make local decisions, searching greedily for local transitions to build a parse tree (Sagae and Lavie, 2005). Zhu et al. (2013) use a beam search framework, which preserves linear time complexity of greedy search, while alleviating the disadvantage of error propagation. The model gives state-of-the-art accuracies at a speed of 89 sentences per second on the standard WSJ benchmark (Marcus et al., 1993).

Zhu et al. (2013) exploit rich features by extracting history information from a parser stack, which consists of a sequence of non-local constituents. However, due to the incremental nature of shift-reduce parsing, the right-hand side constituents of the current word cannot be used to guide the action at each step. Such lookahead features (Tsuruoka et al., 2011) correspond to the outside scores in chart parsing (Goodman, 1998), which has been effective for obtaining improved accuracies.

To leverage such information for improving shift-reduce parsing, we propose a novel neural model to predict the constituent hierarchy related to each word before parsing. Our idea is inspired by the work of Roark and Hollingshead (2009) and Zhang et al. (2010b), which shows that shallow syntactic information gathered over the word sequence can be utilized for pruning chart parsers, improving chart parsing speed without sacrificing accuracies. For example, Roark and Hollingshead (2009) predict constituent boundary information on words as a pre-processing step, and use such information to prune the chart. Since such information is much lighter-weight compared to full parsing, it can be predicted relatively accurately using sequence labellers.

Different from Roark and Hollingshead (2009), we collect *lookahead* constituent information for *shift-reduce* parsing, rather than *pruning* information for *chart* parsing. Our main concern is improving the *accuracy* rather than improving the *speed*. Accordingly, our model should predict the constituent hierarchy for each word rather than simple boundary information. For example, in Figure 1(a), the constituent hierarchy that the word "*The*" starts is "$S \rightarrow NP$", and the constituent hierarchy that the word "table" ends is "$S \rightarrow VP \rightarrow NP \rightarrow PP \rightarrow NP$".

Transactions of the Association for Computational Linguistics, vol. 5, pp. 45–58, 2017. Action Editor: Brian Roark.
Submission batch: 5/2016; Revision batch: 9/2016; Published 1/2017.



(a)

Word	*s*-type	*e*-type
The	[*s*: *S*→*NP*]	[*e*: Ø]
past	[*s*: *ADJP*]	[*e*: Ø]
and	[*s*: Ø]	[*e*: Ø]
present	[*s*: Ø]	[*e*: *ADJP*]
students	[*s*: Ø]	[*e*: *NP*]
like	[*s*: *VP*]	[*e*: Ø]
this	[*s*: *NP*→*NP*]	[*e*: Ø]
book	[*s*: Ø]	[*e*: *NP*]
on	[*s*: *PP*]	[*e*: Ø]
the	[*s*: *NP*]	[*e*: Ø]
table	[*s*: Ø	[*e*: *S*→*VP*→*NP*→*PP*→*NP*]

(b)

Figure 1: Example constituent hierarchies for the sentence "The past and present students like this book on the table". (a) parse tree; (b) constituent hierarchies on words.

For each word, we predict both the constituent hierarchy it starts and the constituent hierarchy it ends, using them as lookahead features.

The task is challenging. First, it is significantly more difficult compared to simple sequence labelling, since two sequences of constituent hierarchies must be predicted for each word in the input sequence. Second, for high accuracies, global features from the full sentence are necessary since constituent hierarchies contain rich structural information. Third, to retain high speed for shift-reduce parsing, lookahead feature prediction must be executed efficiently. It is highly difficult to build such a model using manual discrete features and structured search.

Fortunately, sequential recurrent neural networks (RNNs) are remarkably effective models to encode the full input sentence. We leverage RNNs for building our constituent hierarchy predictor. In particular, an LSTM (Hochreiter and Schmidhuber, 1997) is used to learn global features automatically from the input words. For each word, a second LSTM is then used to generate the constituent hierarchies greedily using features from the hidden layer of the first LSTM, in the same way a neural language model decoder generates output sentences for machine translation (Bahdanau et al., 2015). The resulting model solves all three challenges raised above. For fully-supervised learning, we learn word embeddings as part of the model parameters.

In the standard WSJ (Marcus et al., 1993) and CTB 5.1 tests (Xue et al., 2005), our parser gives 1.3 F_1 and 2.3 F_1 improvement, respectively, over the baseline of Zhu et al. (2013), resulting in a accuracy of 91.7 F_1 for English and 85.5 F_1 for Chinese, which are the best for fully-supervised models in the literature. We release our code, based on ZPar (Zhang and Clark, 2011; Zhu et al., 2013), at https://github.com/SUTDNLP/LookAheadConparser.

Initial State	$[\phi, 0, false, 0]$
Final State	$[S, n, true, m : 2n <= m <= 4n]$

Induction Rules:

SHIFT	$\dfrac{[S,i,false,k]}{[S\|w,i{+}1,false,k{+}1]}$
REDUCE-L/R-X	$\dfrac{[S\|s_1 s_0,i,false,k]}{[S\|X,i,false,k{+}1]}$
UNARY-X	$\dfrac{[S\|s_0,i,false,k]}{[S\|X,i,false,k{+}1]}$
FINISH	$\dfrac{[S,n,false,k]}{[S,n,true,k{+}1]}$
IDLE	$\dfrac{[S,n,true,k]}{[S,n,true,k{+}1]}$

Figure 2: Deduction system for the baseline shift-reduce parsing process.

2 Baseline System

We adopt the parser of Zhu et al. (2013) for a baseline, which is based on the shift-reduce process of Sagae and Lavie (2005) and the beam search strategy of Zhang and Clark (2011) with global perceptron training.

2.1 The Shift-Reduce System

Shift-reduce parsers process an input sentence incrementally from left to right. A stack is used to maintain partial phrase-structures, while the incoming words are ordered in a buffer. At each step, a transition action is applied to consume an input word or construct a new phrase-structure. The set of transition actions are

- SHIFT: pop the front word off the buffer, and push it onto the stack.
- REDUCE-L/R-X: pop the top two constituents off the stack (L/R means that the head is the left constituent or the right constituent, respectively), combine them into a new constituent with label X, and push the new constituent onto the stack.
- UNARY-X: pop the top constituent off the stack, raise it to a new constituent X, and push the new constituent onto the stack.
- FINISH: pop the root node off the stack and end parsing.
- IDLE: no-effect action on a completed state without changing items on the stack or buffer, used to ensure that the same number of actions are in each item in beam search (Zhu et al., 2013).

The deduction system for the process is shown in Figure 2, where a state is represented as [*stack*, *buffer front index*, *completion mark*, *action index*], and n is the number of words in the input. For example, given the sentence "They like apples", the action sequence "SHIFT, SHIFT, SHIFT, REDUCE-L-VP, REDUCE-R-S" gives its syntax "(S They (VP like apples))".

2.2 Search and Training

Beam-search is used for decoding with the k best state items at each step being kept in the agenda. During initialization, the agenda contains only the initial state $[\phi, 0, false, 0]$. At each step, each state in the agenda is popped and expanded by applying all valid transition actions, and the top k resulting states are put back onto the agenda (Zhu et al., 2013). The process repeats until the agenda is empty, and the best completed state is taken as output.

The score of a state is the total score of the transition actions that have been applied to build it:

$$C(\alpha) = \sum_{i=1}^{N} \Phi(\alpha_i) \cdot \vec{\theta} \qquad (1)$$

Here $\Phi(\alpha_i)$ represents the feature vector for the ith action α_i in the state item α. N is the total number of actions in α.

The model parameter vector $\vec{\theta}$ is trained online using the averaged perceptron algorithm with the early-update strategy (Collins and Roark, 2004).

2.3 Baseline Features

Our baseline features are taken from Zhu et al. (2013). As shown in Table 1, they include the UNIGRAM, BIGRAM, TRIGRAM features of Zhang and Clark (2009) and the extended features of Zhu et al. (2013).

Description	Templates
UNIGRAM	$s_0tc, s_0wc, s_1tc, s_1wc, s_2tc$
	$s_2wc, s_3tc, s_3wc, q_0wt, q_1wt$
	$q_2wt, q_3wt, s_0lwc, s_0rwc$
	$s_0uwc, s_1lwc, s_1rwc, s_1uwc$
BIGRAM	$s_0ws_1w, s_0ws_1c, s_0cs_1w, s_0cs_1c$
	$s_0wq_0w, s_0wq_0t, s_0cq_0w, s_0cq_0t$
	$q_0wq_1w, q_0wq_1t, q_0tq_1w, q_0tq_1t$
	$s_1wq_0w, s_1wq_0t, s_1cq_0w, s_1cq_0t$
TRIGRAM	$s_0cs_1cs_2c, s_0ws_1cs_2c, s_0cs_1wq_0t$
	$s_0cs_1cs_2w, s_0cs_1cq_0t, s_0ws_1cq_0t$
	$s_0cs_1wq_0t, s_0cs_1cq_0w$
Extended	$s_0llwc, s_0lrwc, s_0luwc$
	$s_0rlwc, s_0rrwc, s_0ruwc$
	$s_0ulwc, s_0urwc, s_0uuwc$
	$s_1llwc, s_1lrwc, s_1luwc$
	$s_1rlwc, s_1rrwc, s_1ruwc$

Table 1: Baseline feature templates, where s_i represents the ith item on the top of the stack and q_i denotes the ith item in the front of the buffer. The symbol w denotes the lexical head of an item; the symbol c denotes the constituent label of an item; the symbol t is the POS of a lexical head; u denotes unary child; s_ill denotes the left child of s_i's left child.

Templates
$s_0g_s, s_0g_e, s_1g_s, s_1g_e$
$q_0g_s, q_0g_e, q_1g_s, q_1g_e$

Table 2: Lookahead feature templates, where s_i represents the ith item on the top of the stack and q_i denotes the ith item in the front end of the buffer. The symbol g_s and g_e denote the next level constituent in the s-type hierarchy and e-type hierarchy, respectively.

3 Global Lookahead Features

The baseline features suffer two limitations, as mentioned in the introduction. First, they are relatively local to the state, considering only the neighbouring nodes of s_0 (top of stack) and q_0 (front of buffer). Second, they do not consider lookahead information beyond s_3, or the syntactic structure of the buffer and sequence. We use an LSTM to capture full sentential information in linear time, representing such global information that is fed into the baseline parser as a constituent hierarchy for each word. *Lookahead features* are extracted from the constituent hierarchy to provide top-down guidance for bottom-up parsing.

3.1 Constituent Hierarchy

In a constituency tree, each word can start or end a constituent hierarchy. As shown in Figure 1, the word "*The*" starts a constituent hierarchy "*S* → *NP*". In particular, it starts a constituent *S* in the top level, dominating a constituent *NP*. The word "*table*" ends a constituent hierarchy "*S* → *VP* → *NP* → *PP* → *NP*". In particular, it ends a constituent hierarchy, with a constituent *S* on the top level, dominating a *VP* (starting from the word "*like*"), and then an *NP* (starting from the noun phrase "*this book*"), and then a *PP* (starting from the word "*in*"), and finally an *NP* (starting from the word "*the*"). The extraction of constituent hierarchies for each word is based on *unbinarized* grammars, reflecting the *unbinarized* trees that the word starts or ends. The constituent hierarchy is *empty* (denoted as ϕ) if the corresponding word does not start or end a constituent. The constituent hierarchies are added into the shift-reduce parser as soft features (section 3.2).

Formally, a constituent hierarchy is defined as

$$[type : c_1 \rightarrow c_2 \rightarrow ... \rightarrow c_z],$$

where c is a constituent label (e.g. *NP*), "→" represents the top-down hierarchy, and $type$ can be s or e, denoting that the current word starts or ends the constituent hierarchy, respectively, as shown in Figure 1. Compared with full parsing, the constituent hierarchies associated with each word have no forced structural dependencies between each other, and therefore can be modelled more easily, for each word individually. Being soft lookahead features rather than hard constraints, inter-dependencies are not crucial for the main parser.

3.2 Lookahead Features

The lookahead feature templates are defined in Table 2. In order to ensure parsing efficiency, only simple feature templates are taken into consideration. The lookahead features of a state are instantiated for the top two items on the stack (i.e., s_0 and s_1) and buffer (i.e., q_0 and q_1). The new function Φ' is defined to output the lookahead features vector. The scoring of a state in our model is based on Formula (1) but with a new term $\Phi'(\alpha_i) \cdot \vec{\theta'}$:

$$C'(\alpha) = \sum_{i=1}^{N} \Phi(\alpha_i) \cdot \vec{\theta} + \Phi'(\alpha_i) \cdot \vec{\theta'}$$

For each word, the lookahead feature represents the next level constituent in the top-down hierarchy, which can guide bottom-up parsing.

For example, Figure 3 shows two intermediate states during parsing. In Figure 3(a), the s-type and e-type lookahead features of s_1 (i.e., the word "*The*" are extracted from the constituent hierarchy in the bottom level, namely *NP* and NULL, respectively. On the other hand, in Figure 3(b), the s-type lookahead feature of s_1 is extracted from the s-type constituent hierarchy of same word "*The*", but it is *S* based on current hierarchical level. The e-type lookahead feature, on the other hand, is extracted from the e-type constituent hierarchy of end word "*students*" of the *VP* constituent, which is NULL in the next level. Lookahead features for items on the buffer are extracted in the same way.

The lookahead features are useful for guiding shift-reduce decisions given the current state. For

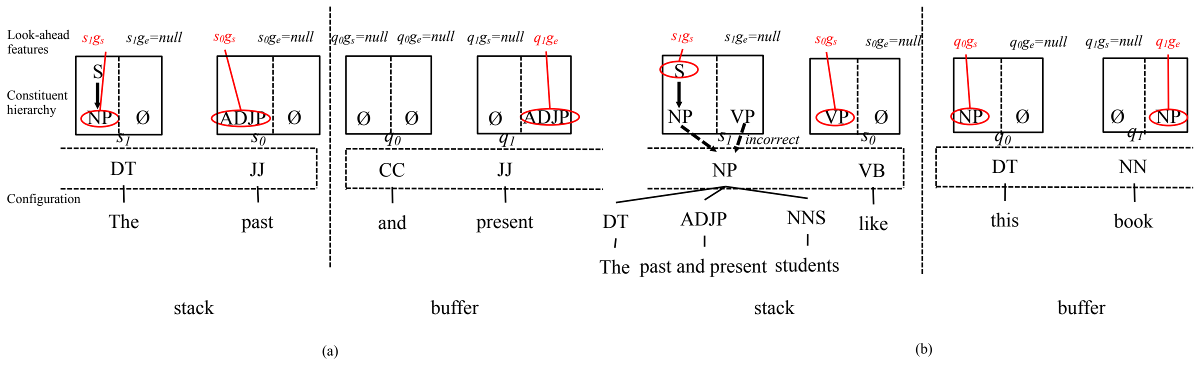


Figure 3: Two intermediate states for parsing on the sentence "The past and present students like this book on the table". Each item on the stack or buffer has two constituent hierarchies: s-type (left) and e-type (right), respectively, in the corresponding box. Note that the e-type constituent hierarchy of the word "students" is incorrectly predicted, yet used as soft constraints (i.e., features) in our model.

example, given the intermediate state in Figure 3(a), s_0 has a s-type lookahead feature *ADJP*, and q_1 in the buffer has e-type lookahead feature *ADJP*. This indicates that the two items are likely reduced into the same constituent. Further, s_0 cannot end a constituent because of the empty e-type constituent hierarchy. As a result, the final shift-reduce parser will assign a higher score to the SHIFT decision.

4 Constituent Hierarchy Prediction

We propose a novel neural model for constituent hierarchy prediction. Inspired by the encoder-decoder framework for neural machine translation (Bahdanau et al., 2015; Cho et al., 2014), we use an LSTM to capture full sentence features, and another LSTM to generate the constituent hierarchies for each word. Compared with a CRF-based sequence labelling model (Roark and Hollingshead, 2009), the proposed model has three advantages. First, the global features can be automatically represented. Second, it can avoid the exponentially large number of labels if constituent hierarchies are treated as unique labels. Third, the model size is relatively small, and does not have a large effect on the final parser model.

As shown in Figure 4, the neural network consists of three main layers, namely the *input layer*, the *encoder layer* and the *decoder layer*. The input layer represents each word using its characters and token information; the encoder hidden layer uses a bidirectional recurrent neural network structure to learn global features from the sentence; and the decoder layer predicts constituent hierarchies according to the encoder layer features, by using the attention mechanism (Bahdanau et al., 2015) to compute the contribution of each hidden unit of the encoder.

4.1 Input Layer

The input layer generates a dense vector representation of each input word. We use character embeddings to alleviate OOV problems in word embeddings (Ballesteros et al., 2015; Santos and Zadrozny, 2014; Kim et al., 2016), concatenating character-embeddings of a word with its word embedding. Formally, the input representation x_i of the word w_i is computed by:

$$x_i = [x_{w_i}; c_{i\_att}]$$
$$c_{i\_att} = \sum_j \alpha_{ij} c'_{ij},$$

where x_{w_i} is a word embedding vector of the word w_i according to a embedding lookup table, c_{i_att} is a character embedding form of the word w_i, c_{ij} is the embedding of the jth character in w_i, c'_{ij} is the character window representation centered at c_{ij}, and α_{ij} is the contribution of the c'_{ij} to c_{i_att}, which is computed by:

$$\alpha_{ij} = \frac{e^{f(x_{w_i}, c'_{ij})}}{\sum_k e^{f(x_{w_i}, c'_{ik})}}$$

f is a non-linear transformation function.

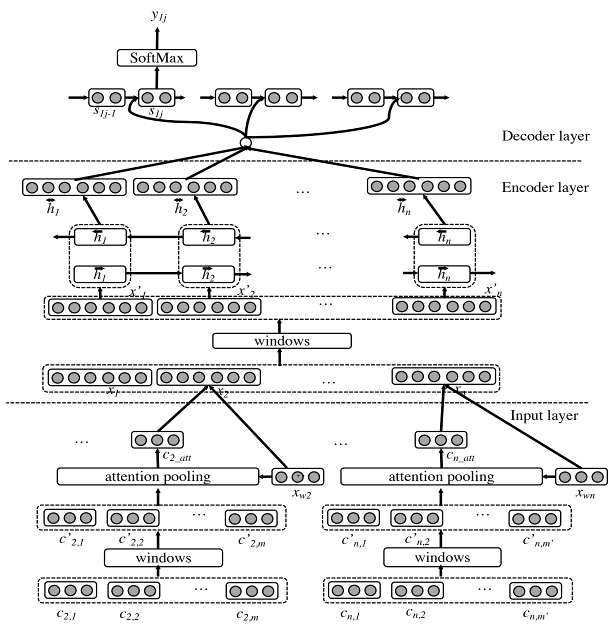


Figure 4: Structure of the constituent hierarchy prediction model. $\overrightarrow{h_i}$ denotes the left-to-right encoder hidden units; $\overleftarrow{h_i}$ denotes the right-to-left encoder hidden units; s denotes the decoder hidden state vector; and y_{ij} is the jth label of the word w_i.

4.2 Encoder Layer

The encoder first uses a window strategy to represent input nodes with their corresponding local context nodes. Formally, a word window representation takes the form

$$x'_i = [x_{i-win}; ...; x_i; ...; x_{i+win}].$$

Second, the encoder scans the input sentence and generates hidden units for each input word using a recurrent neural network (RNN), which represents features of the word from the global sequence. Formally, given the windowed input nodes x'_1, x'_2, ..., x'_n for the sentence w_1, w_2, ..., w_n, the RNN layer calculates a hidden node sequence h_1, h_2, ..., h_n.

Long Short-Term Memory (LSTM) mitigates the vanishing gradient problem in RNN training, by introducing gates (i.e., input i, forget f and output o) and a cell memory vector c. We use the variation of Graves and Schmidhuber (2008). Formally, the values in the LSTM hidden layers are computed as follows:

$$\begin{aligned}
i_i &= \sigma(W_1 x'_i + W_2 h_{i-1} + W_3 \odot c_{i-1} + b_1)\\
f_i &= 1 - i_i\\
\tilde{c}_i &= tanh(W_4 x'_i + W_5 h_{i-1} + b_2)\\
c_i &= f_i \odot c_{i-1} + i_i \odot \tilde{c}_i\\
o_i &= \sigma(W_6 x'_i + W_7 h_{i-1} + W_8 \odot c_i + b_3)\\
h_i &= o_i \odot tanh(c_i),
\end{aligned}$$

where $\odot$ is pair-wise multiplication. Further, in order to collect features for x_i from both x'_1, .., x'_{i-1} and x'_{i+1}, ... x'_n, we use a bidirectional variation (Schuster and Paliwal, 1997; Graves et al., 2013). As shown in Figure 4, the hidden units are generated by concatenating the corresponding hidden layers of a left-to-right LSTM $\overrightarrow{h_i}$ and a right-to-left LSTM $\overleftarrow{h_i}$, where $\overleftrightarrow{h_i} = [\overrightarrow{h_i}; \overleftarrow{h_i}]$ for each word w_i.

4.3 Decoder Layer

The decoder hidden layer uses two different LSTMs to generate the s-type and e-type sequences of constituent labels from each encoder hidden output, respectively, as shown in Figure 4. Each constituent hierarchy is generated bottom-up recurrently. In particular, a sequence of state vectors is generated recurrently, with each state yielding a output constituent label. The process starts with a $\vec{0}$ state vector and ends when a NULL constituent is generated. The recurrent state transition process is achieved using an LSTM model with the hidden vectors of the encoder layer being used for context features.

Formally, for word w_i, the value of the jth state unit s_{ij} of the LSTM is computed by:

$$s_{ij} = f(s_{ij-1}, a_{ij}, \overleftrightarrow{h_i})^1,$$

where the context a_{ij} is computed by:

$$a_{ij} = \sum_k \beta_{ijk} \overleftrightarrow{h_k}$$

$$\beta_{ijk} = \frac{e^{f(s_{ij-1}, \overleftrightarrow{h_k})}}{\sum_{k'} e^{f(s_{ij-1}, \overleftrightarrow{h_{k'}})}}$$

[1] Here, different from typical MT models (Bahdanau et al., 2015), the chain is predicted sequentially in a feed-forward way with no feedback of the prediction made. We found that this fast alternative gives similar results.

Here $\overleftrightarrow{h_k}$ refers to the encoder hidden vector for w_k. The weights of contribution β_{ijk} are computed using the attention mechanism (Bahdanau et al., 2015).

The constituent labels are generated from each state unit s_{ij}, where each constituent label y_{ij} is the output of a SOFTMAX function,

$$p(y_{ij} = l) = \frac{e^{s_{ij}^{\top} W_l}}{\sum_k e^{s_{ij}^{\top} W_k}}$$

$y_{ij} = l$ denotes that the jth label of the ith word is $l (l \in L)$.

As shown in Figure 4, the SOFTMAX functions are applied to the state units of the decoder, generating hierarchical labels bottom-up, until the default label NULL is predicted.

4.4 Training

We use two separate models to assign the s-type and e-type labels, respectively. For training each constituent hierarchy predictor, we minimize the following training objective:

$$L(\theta) = -\sum_{i}^{T} \sum_{j}^{Z_i} log\, p_{ijo} + \frac{\lambda}{2} ||\theta||^2,$$

where T is the length of the sentence, Z_i is the depth of the constituent hierarchy of the word w_i, and p_{ijo} stands for $p(y_{ij} = o)$, which is given by the SOFTMAX function, and o is the gold label.

We apply back-propagation, using momentum stochastic gradient descent (Sutskever et al., 2013) with a learning rate of $\eta = 0.01$ for optimization and regularization parameter $\lambda = 10^{-6}$.

5 Experiments

5.1 Experiment Settings

Our English data are taken from the Wall Street Journal (WSJ) sections of the Penn Treebank (Marcus et al., 1993). We use sections 2-21 for training, section 24 for system development, and section 23 for final performance evaluation. Our Chinese data are taken from the version 5.1 of the Penn Chinese Treebank (CTB) (Xue et al., 2005). We use articles 001- 270 and 440-1151 for training, articles 301-325 for system development, and articles 271-300 for final performance evaluation. For both English and Chinese data, we adopt ZPar[2] for POS tagging, and use ten-fold jackknifing to assign POS tags automatically to the training data. In addition, we use ten-fold jackknifing to assign constituent hierarchies automatically to the training data for training the parser using the constituent hierarchy predictor.

hyper-parameters	value
Word embedding size	50
Word window size	2
Character embedding size	30
Character window size	2
LSTM hidden layer size	100
Character hidden layer size	60

Table 3: Hyper-parameter settings

	s-type	e-type	parser
1-layer	93.39	81.50	90.43
2-layer	93.76	83.37	90.72
3-layer	93.84	83.42	90.80

Table 4: Performance of the constituent hierarchy predictor and the corresponding parser on the WSJ dev dataset. n-layer denotes an LSTM model with n hidden layers.

We use F_1 score to evaluate constituent hierarchy prediction. For example, if the prediction is "*S* $\rightarrow$ *S* $\rightarrow$ *VP* $\rightarrow$ *NP*" and the gold is "*S* $\rightarrow$ *NP* $\rightarrow$ *NP*", the evaluation process matches the two hierarchies bottom-up. The precision is 2/4 = 0.5, the recall is 2/3 = 0.66 and the $F1$ score is 0.57. A label is counted as correct if and only if it occurs at the correct position.

We use EVALB to evaluate parsing performance, including labelled precision (LP), labelled recall (LR), and bracketing F_1.[3]

5.2 Model Settings

For training the constituent hierarchy prediction model, gold constituent labels are derived from labelled constituency trees in the training data. The hyper-parameters are chosen according to development tests, and the values are shown in Table 3.

For the shift-reduce constituency parser, we set the beam size to 16 for both training and decoding, which achieves a good tradeoff between efficiency

[2]https://github.com/SUTDNLP/ZPar

[3]http://nlp.cs.nyu.edu/evalb

	s-type	e-type	parser
all	93.76	83.37	90.72
all w/o wins	93.62	83.34	90.58
all w/o chars	93.51	83.21	90.33
all w/o chars & wins	93.12	82.36	89.18

Table 5: Performance of the constituent hierarchy predictor and the corresponding parser on the WSJ dev dataset. *all* denotes the proposed model without ablation. *wins* denotes input windows. *chars* denotes character-based attention.

and accuracy (Zhu et al., 2013). The optimal training iteration number is determined on the development sets.

5.3 Results of Constituent Hierarchy Prediction

Table 4 shows the results of constituent hierarchy prediction, where word and character embeddings are randomly initialized, and fine-tuned during training. The third column shows the development parsing accuracies when the labels are used for lookahead features. As Table 4 shows, when the number of hidden layers increases, both s-type and e-type constituent hierarchy prediction improve. The accuracy of e-type prediction is relatively lower due to right-branching in the treebank, which makes e-type hierarchies longer than s-type hierarchies. In addition, a 3-layer LSTM does not give significant improvements compared to a 2-layer LSTM. For better tradeoff between efficiency and accuracy, we choose the 2-layer LSTM as our constituent hierarchy predictor.

Table 5 shows ablation results for constituent hierarchy prediction given by different reduced architectures, which include an architecture without character embeddings and an architecture with neither character embeddings nor input windows. We find that the original architecture achieves the highest performance on constituent hierarchy prediction, compared to the two baselines. The baseline only without character embeddings has relatively small influence on constituent hierarchy prediction. On the other hand, the baseline only without input word windows has relatively smaller influence on constituent hierarchy prediction. Nevertheless, both of these two ablation architectures lead to lower parsing accuracies. The baseline removing both the character embeddings and the input word windows has a relatively low F-score.

Parser	LR	LP	F_1
Fully-supervised			
Ratnaparkhi (1997)	86.3	87.5	86.9
Charniak (2000)	89.5	89.9	89.5
Collins (2003)	88.1	88.3	88.2
Sagae and Lavie (2005)†	86.1	86.0	86.0
Sagae and Lavie (2006)†	87.8	88.1	87.9
Petrov and Klein (2007)	90.1	90.2	90.1
Carreras et al. (2008)	90.7	91.4	91.1
Shindo et al. (2012)	N/A	N/A	91.1
Zhu et al. (2013)†	90.2	90.7	90.4
Socher et al. (2013)*	N/A	N/A	90.4
Vinyals et al. (2015)*	N/A	N/A	88.3
Cross and Huang (2016)*†	N/A	N/A	91.3
Dyer et al. (2016)*†	N/A	N/A	91.2
This work	**91.3**	**92.1**	**91.7**
Ensemble			
Shindo et al. (2012)	N/A	N/A	92.4
Vinyals et al. (2015)*	N/A	N/A	90.5
Rerank			
Charniak and Johnson (2005)	91.2	91.8	91.5
Huang (2008)	92.2	91.2	91.7
Dyer et al. (2016)*†	N/A	N/A	93.3
Semi-supervised			
McClosky et al. (2006)	92.1	92.5	92.3
Huang and Harper (2009)	91.1	91.6	91.3
Huang et al. (2010)	91.4	91.8	91.6
Zhu et al. (2013)†	91.1	91.5	91.3
Durrett and Klein (2015)*	N/A	N/A	91.1

Table 6: Comparison of related work on the WSJ test set. * denotes neural parsing; † denotes methods using a shift-reduce framework.

5.4 Final Results

For English, we compare the final results with previous related work on the WSJ test sets. As shown in Table 6[4], our model achieves 1.3% F_1 improvement compared to the baseline parser with fully-supervised learning (Zhu et al., 2013). Our model outperforms the state-of-the-art fully-supervised system (Carreras et al., 2008; Shindo et al., 2012) by 0.6% F_1. In addition, our fully-supervised model also catches up with many state-of-the-art semi-supervised models (Zhu et al., 2013;

[4]We treat the methods as semi-supervised if they use pre-trained word embeddings, word clusters (e.g., Brown clusters) or extra resources.

Parser	LR	LP	F_1
Fully-supervised			
Charniak (2000)	79.6	82.1	80.8
Bikel (2004)	79.3	82.0	80.6
Petrov and Klein (2007)	81.9	84.8	83.3
Zhu et al. (2013)†	82.1	84.3	83.2
Wang et al. (2015)‡	N/A	N/A	83.2
Dyer et al. (2016)*†	N/A	N/A	84.6
This work	**85.2**	**85.9**	**85.5**
Rerank			
Charniak and Johnson (2005)	80.8	83.8	82.3
Dyer et al. (2016)*†	N/A	N/A	86.9
Semi-supervised			
Zhu et al. (2013)†	84.4	86.8	85.6
Wang and Xue (2014)‡	N/A	N/A	86.3
Wang et al. (2015)‡	N/A	N/A	86.6

Table 7: Comparison of related work on the CTB5.1 test set. * denotes neural parsing; † denotes methods using a shift-reduce framework; ‡ denotes joint POS tagging and parsing.

Huang and Harper, 2009; Huang et al., 2010; Durrett and Klein, 2015) by achieving 91.7% F_1 on WSJ test set. The size of our model is much smaller than the semi-supervised model of Zhu et al. (2013), which contains rich features from a large automatically parsed corpus. In contrast, our model is about the same in size compared to the baseline parser.

We carry out Chinese experiments with the same models, and compare the final results with previous related work on the CTB test set. As shown in Table 7, our model achieves 2.3% F_1 improvement compared to the state-of-the-art baseline system with fully-supervised learning (Zhu et al., 2013), which is by far the best result in the literature. In addition, our fully-supervised model is also comparable to many state-of-the-art semi-supervised models (Zhu et al., 2013; Wang and Xue, 2014; Wang et al., 2015; Dyer et al., 2016) by achieving 85.5% F_1 on the CTB test set. Wang and Xue (2014) and Wang et al. (2015) do joint POS tagging and parsing.

5.5 Comparison of Speed

Table 8 shows the running times of various parsers on test sets on a Intel 2.2 GHz processor with 16G memory. Our parsers are much faster than the related parser with the same shift-reduce framework (Sagae and Lavie, 2005; Sagae and Lavie, 2006). Compared to the baseline parser, our parser gives

Parser	#Sent/Second
Ratnaparkhi (1997)	Unk
Collins (2003)	3.5
Charniak (2000)	5.7
Sagae and Lavie (2005)	3.7
Sagae and Lavie (2006)	2.2
Petrov and Klein (2007)	6.2
Carreras et al. (2008)	Unk
Zhu et al. (2013)	89.5
This work	79.2

Table 8: Comparison of running times on the test set, where the time for loading models is excluded. The running times of related parsers are taken from Zhu et al. (2013).

significant improvement on accuracies (90.4% to 91.7% F_1) at the speed of 79.2 sentences per second[5], in contrast to 89.5 sentences per second on the standard WSJ benchmark.

6 Error Analysis

We conduct error analysis by measuring parsing accuracies against: different phrase types, constituents of different span lengths, and different sentence lengths.

6.1 Phrase Type

Table 9 shows the accuracies of the baseline and the final parsers with lookahead features on 9 common phrase types. As the results show, while the parser with lookahead features achieves improvements on all of the frequent phrase types, there are relatively higher improvements on *VP*, *S*, *SBAR* and *WHNP*.

The constituent hierarchy predictor has relatively better performance on s-type labels for the constituents *VP*, *WHNP* and *PP*, which are prone to errors by the baseline system. The constituent hierarchy can give guidance to the constituent parser for tackling the issue. Compared to the s-type constituent hierarchy, the e-type constituent hierarchy

[5]The constituent hierarchy prediction is excluded, which processes an average of 150 sentences per second on a single CPU. The cost of this step is far less than the cost of parsing, and can be essentially eliminated by pipelining the constituent hierarchy prediction and the shift-reduce decoder, by launching the constituent hierarchy predictor first, and then starting parsing in parallel as soon as the lookahead output is available for the first sentence, since the lookahead will outpace the parsing from that point forward.

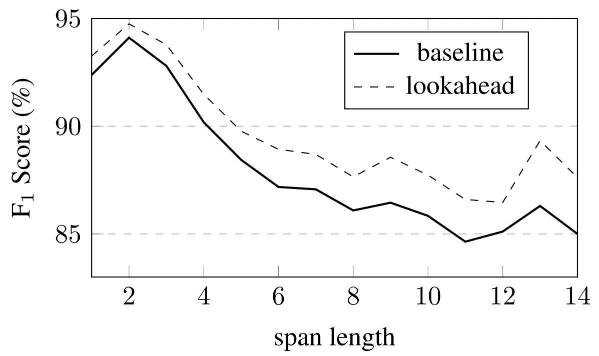


Figure 5: Comparison with the baseline on spans of different lengths.

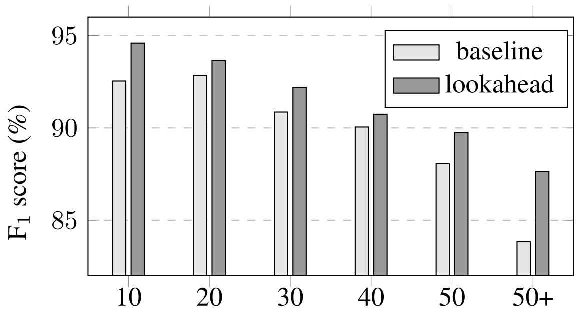


Figure 6: Comparison with the baseline on sentences of different lengths. Sentences with length [0, 10) fall in the bin 10.

is relatively more difficult to predict, particularly for the constituents with long spans such as *VP*, *S* and *SBAR*. Despite this, the e-type constituent hierarchies with relatively low accuracies also benefit prediction of constituents with long spans.

6.2 Span Length

Figure 5 shows the F1-scores of the two parsers on constituents with different span lengths. As the results show, lookahead features are helpful on both large spans and small spans, and the performance gap between the two parsers is larger as the size of span increases. This reflects the usefulness of long-range information captured by the constituent hierarchy predictor and lookahead features.

6.3 Sentence Length

Figure 6 shows the F1-scores of the two parsers on sentences of different lengths. As the results show, the parser with lookahead features outperforms the baseline system on both short sentences and long sentences. Also, the performance gap between the two parsers is larger as the length of sentence increases.

The constituent hierarchy predictors generate hierarchical constituents for each input word using global information. For longer sentences, the predictors yield deeper constituent hierarchies, offering corresponding lookahead features. As a result, compared to the baseline parser, the performance of the parser with lookahead features decreases more slowly as the length of the sentences increases.

7 Related Work

Our lookahead features are similar in spirit to the pruners of Roark and Hollingshead (2009) and Zhang et al. (2010b), which infer the maximum length of constituents that a particular word can start or end. However, our method is different in three main ways. First, rather than using a CRF with sparse local word window features, a neural network is used for dense global features on the sentence. Second, not only the size of constituents but also the constituent hierarchy is identified for each word. Third, the results are added into a transition-based parser as soft features, rather then being used as hard constraints to a chart parser.

Our concept of *constituent hierarchies* is similar to *supertags* in the sense that both are shallow parses. For lexicalized grammars such as Combinatory Categorial Grammar (CCG), Tree-Adjoining Grammar (TAG) and Head-Driven Phrase Structure Grammar (HPSG), each word in the input sentence is assigned one or more supertags, which are used to identify the syntactic role of the word to constrain parsing (Clark, 2002; Clark and Curran, 2004; Carreras et al., 2008; Ninomiya et al., 2006; Dridan et al., 2008; Faleńska et al., 2015). For a lexicalized grammar, *supertagging* can benefit the parsing in both accuracy and efficiency by offering *almost-parsing* information. In particular, Carreras et al. (2008) used the concept of *spine* for TAG (Schabes, 1992; Vijay-Shanker and Joshi, 1988), which is similar to our *constituent hierarchy*. However, there are three differences. First, the *spine* is defined to describe the main syntactic tree structure with a series

		NP	VP	S	PP	SBAR	ADVP	ADJP	WHNP	QP
baseline		92.06	90.63	90.28	87.93	86.93	84.83	74.12	95.03	89.32
with lookahead feature		93.10	92.45	91.78	88.84	88.59	85.64	74.50	96.18	89.63
improvement		+1.04	+1.82	+1.50	+0.91	+1.66	+0.81	+0.38	+1.15	+0.31
constituent hierarchy	s-type	95.18	97.51	93.37	98.01	92.14	88.94	79.88	96.18	91.70
	e-type	91.98	76.82	80.72	84.80	66.82	85.01	71.16	95.13	91.02

Table 9: Comparison between the parsers with lookahead features on different phrases types, with the corresponding constituent hierarchy predictor performances.

of unary projections, while *constituent hierarchy* is defined to describe how words can start or end hierarchical constituents (it can be empty if the word cannot start or end constituents). Second, *spines* are extracted from gold trees and used to prune the search space of parsing as hard constraints. In contrast, we use constituent hierarchies as soft features. Third, Carreras et al. (2008) use *spines* to prune chart parsing, while we use *constituent hierarchies* to improve a linear shift-reduce parser.

For lexicalized grammars, *supertags* can benefit parsing significantly since they contain rich syntactic information as *almost parsing* (Bangalore and Joshi, 1999). Recently, there has been a line of work on better supertagging. Zhang et al. (2010a) proposed efficient methods to obtain supertags for HPSG parsing using dependency information. Xu et al. (2015) and Vaswani et al. (2016) leverage recursive neural networks for *supertagging* for CCG parsing. In contrast, our models predict the constituent hierarchy instead of a single supertag for each word in the input sentence.

Our constituent hierarchy predictor is also related to sequence-to-sequence learning (Sutskever et al., 2014), which has been successfully used in neural machine translation (Bahdanau et al., 2015). The neural model encodes the source-side sentence into dense vectors, and then uses them to generate target-side word by word. There has also been work that directly applies sequence-to-sequence models for constituent parsing, which generates constituent trees given raw sentences (Vinyals et al., 2015; Luong et al., 2015). Compared to Vinyals et al. (2015), who predict a full parse tree from input, our predictors tackle a much simpler task, by predicting the constituent hierarchies of each word separately. In addition, the outputs of the predictors are used for soft lookahead features in bottom-up parsing, rather than being taken as output structures directly.

By integrating a neural constituent hierarchy predictor, our parser is related to neural network models for parsing, which has given competitive accuracies for both constituency parsing (Dyer et al., 2016; Cross and Huang, 2016; Watanabe and Sumita, 2015) and dependency parsing (Chen and Manning, 2014; Zhou et al., 2015; Dyer et al., 2015). In particular, our parser is more closely related to neural models that integrate discrete manual features (Socher et al., 2013; Durrett and Klein, 2015). Socher et al. (2013) use neural features to rerank a sparse baseline parser; Durrett and Klein directly integrate sparse features into neural layers in a chart parser. In contrast, we integrate neural information into sparse features in the form of lookahead features.

There has also been work on lookahead features for parsing. Tsuruoka et al. (2011) run a baseline parser for a few future steps, and use the output actions to guide the current action. In contrast to their model, our model leverages full sentential information, yet is significantly faster.

Previous work investigated more efficient parsing without loss of accuracy, which is required by real time applications, such as web parsing. Zhang et al. (2010b) introduced a chart pruner to accelerate a CCG parser. Kummerfeld et al. (2010) proposed a self-training method focusing on increasing the speed of a CCG parser rather than its accuracy.

8 Conclusion

We proposed a novel constituent hierarchy predictor based on recurrent neural networks, aiming to capture global sentential information. The resulting constituent hierarchies are fed to a baseline shift-reduce parser as lookahead features, addressing limitations of shift-reduce parsers in not leveraging right-hand side syntax for local decisions, yet maintaining the same model size and speed. The resulting fully-supervised parser outperforms the state-of-the-art baseline parser by achieving 91.7% F_1 on standard WSJ evaluation and 85.5% F_1 on standard CTB evaluation.

Acknowledgments

We thank the anonymous reviewers for their detailed and constructive comments, and the co-Editor-in-Chief Lillian Lee for her extremely detailed copy editing. This work is supported by T2MOE 201301 of Singapore Ministry of Education. Yue Zhang is the corresponding author.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR*.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *EMNLP*, pages 349–359.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265, June.

Daniel M. Bikel. 2004. On the parameter space of generative lexicalized statistical parsing models. *PhD Thesis, University of Pennsylvania*.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *CoNLL*, pages 9–16, Morristown, NJ, USA. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *ACL*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *ANLP*, pages 132–139.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *COLING*, pages 282–288, Morristown, NJ, USA, August. University of Edinburgh, Association for Computational Linguistics.

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks*, pages 101–106, Universita di Venezia.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL*, Morristown, NJ, USA. Association for Computational Linguistics.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *EMNLP*.

Rebecca Dridan, Valia Kordoni, and Jeremy Nicholson. 2008. Enhancing performance of lexicalised grammars. In *ACL*.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *ACL*, pages 302–312.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL-IJCNLP*, pages 334–343.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *NAACL*, pages 199–209.

Agnieszka Faleńska, Anders Björkelund, Özlem Çetinoğlu, and Wolfgang Seeker. 2015. Stacking or supertagging for dependency parsing – what's the difference? In *Proceedings of the 14th International Conference on Parsing Technologies*.

Joshua Goodman. 1998. Parsing inside-out. *PhD thesis, Harvard University*.

Alex Graves and Jürgen Schmidhuber. 2008. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *IEEE Workshop on Automatic Speech Recognition & Understanding (ASRU)*, pages 273–278. IEEE.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November.

Zhongqiang Huang and Mary P. Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *EMNLP*, pages 832–841.

Zhongqiang Huang, Mary P. Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *EMNLP*, pages 12–22.

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

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *AAAI*.

Jonathan K. Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. 2010. Faster parsing by supertagger adaptation. In *ACL*, pages 345–355. University of Cambridge, Association for Computational Linguistics, July.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *ICLR*.

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

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *HLT-NAACL*, pages 152–159, Morristown, NJ, USA. Association for Computational Linguistics.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *EMNLP*, pages 155–163. University of Manchester, Association for Computational Linguistics, July.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *HLT-NAACL*, pages 404–411.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *EMNLP*.

Brian Roark and Kristy Hollingshead. 2009. Linear complexity context-free parsing pipelines via chart constraints. In *HLT-NAACL*, pages 647–655.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear runtime complexity. In *IWPT*, pages 125–132, Morristown, NJ, USA. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *HLT-NAACL*, pages 129–132, Morristown, NJ, USA. Association for Computational Linguistics.

Cicero D. Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *ICML*, pages 1818–1826.

Yves Schabes. 1992. Stochastic tree-adjoining grammars. In *Proceedings of the workshop on Speech and Natural Language*, pages 140–145. Association for Computational Linguistics.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE transaction*, 45(11):2673–2681.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *ACL*, pages 440–448.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing with compositional vector grammars. In *ACL*, pages 455–465.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. 2013. On the importance of initialization and momentum in deep learning. In *ICML*, pages 1139–1147.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Kazama. 2011. Learning with lookahead: Can history-based models rival globally optimized models? In *CoNLL*, pages 238–246.

Ashish Vaswani, Yonatan Bisk, and Kenji Sagae. 2016. Supertagging with LSTMs. In *NAACL*.

K. Vijay-Shanker and Aravind K. Joshi. 1988. *A study of tree adjoining grammars*. Citeseer.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *NIPS*, pages 2773–2781.

Zhiguo Wang and Nianwen Xue. 2014. Joint POS tagging and transition-based constituent parsing in Chinese with non-local features. In *ACL*, pages 733–742, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhiguo Wang, Haitao Mi, and Nianwen Xue. 2015. Feature optimization for constituent parsing via neural networks. In *ACL-IJCNLP*, pages 1138–1147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *ACL*, pages 1169–1179.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *ACL-IJCNLP*, pages 250–255, Stroudsburg, PA, USA. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-dong Chiou, and Martha Palmer. 2005. The Penn Chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *ICPT*, pages 162–171, Morristown, NJ, USA. 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.

Yaozhong Zhang, Takuya Matsuzaki, and Jun'ichi Tsujii. 2010a. A simple approach for HPSG supertagging using dependency information. In *NAACL-HLT*, pages 645–648. University of Manchester, Association for Computational Linguistics, June.

Yue Zhang, Byung-Gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010b. Chart pruning for fast lexicalised-grammar parsing. In *COLING*, pages 1471–1479.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *ACL*, pages 1213–1222.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL*, pages 434–443.