Head-Lexicalized Bidirectional Tree LSTMs

Zhiyang Teng and **Yue Zhang**
Singapore University of Technology and Design
zhiyang_teng@mymail.sutd.edu.sg
yue_zhang@sutd.edu.sg

Abstract

Sequential LSTMs have been extended to model tree structures, giving competitive results for a number of tasks. Existing methods model constituent trees by bottom-up combinations of constituent nodes, making direct use of input word information only for leaf nodes. This is different from sequential LSTMs, which contain references to input words for each node. In this paper, we propose a method for automatic head-lexicalization for tree-structure LSTMs, propagating head words from leaf nodes to every constituent node. In addition, enabled by head lexicalization, we build a tree LSTM in the top-down direction, which corresponds to bidirectional sequential LSTMs in structure. Experiments show that both extensions give better representations of tree structures. Our final model gives the best results on the Stanford Sentiment Treebank and highly competitive results on the TREC question type classification task.

1 Introduction

Both sequence structured and tree structured neural models have been applied to NLP problems. Seminal work uses convolutional neural networks (Collobert and Weston, 2008), recurrent neural networks (Elman, 1990; Mikolov et al., 2010) and recursive neural networks (Socher et al., 2011) for sequence and tree modeling. Long short-term memory (LSTM) networks have significantly improved accuracies in a variety of sequence tasks (Sutskever et al., 2014; Bahdanau et al., 2015) compared to vanilla recurrent neural networks. Addressing diminishing gradients effectively, they have been extended to tree structures, achieving promising results for tasks such as syntactic language modeling (Zhang et al., 2016), sentiment analysis (Li et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015; Tai et al., 2015; Teng et al., 2016) and relation extraction (Miwa and Bansal, 2016).

Depending on the node type, typical tree structures in NLP can be categorized to *constituent* trees and *dependency* trees. A salient difference between the two types of tree structures is in the node. While dependency tree nodes are *input words* themselves, constituent tree nodes represent *syntactic constituents*. Only leaf nodes in constituent trees correspond to words. Though LSTM structures have been developed for both types of trees above, we investigate constituent trees in this paper. There are three existing methods for constituent tree LSTMs (Zhu et al., 2015; Tai et al., 2015; Le and Zuidema, 2015), which make essentially the same extension from sequence structured LSTMs. We take the method of Zhu et al. (2015) as our baseline.

Figure 1 shows the sequence structured LSTM of Hochreiter and Schmidhuber (1997) and the tree-structured LSTM of Zhu et al. (2015), illustrating the input (x), cell (c) and hidden (h) nodes at a certain time step t. The most important difference between Figure 1(a) and Figure 1(b) is the branching factor. While a cell in the sequence structure LSTM depends on the single previous hidden node, a cell in the tree-structured LSTM depends on a left hidden node and a right hidden node. Such tree-structured extensions of the sequence structured LSTM assume

Transactions of the Association for Computational Linguistics, vol. 5, pp. 163–177, 2017. Action Editor: Scott Yih.
Submission batch: 5/2016; Revision batch: 12/2016; Published 6/2017.

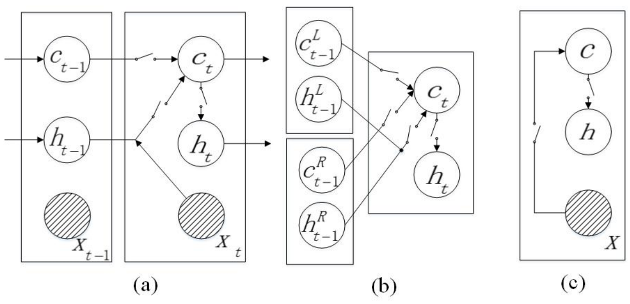


Figure 1: Topology of sequential and tree LSTMs. (a) nodes in sequential LSTM; (b) non-leaf nodes in tree LSTM; (c) leaf nodes in tree LSTM. Shaded nodes represent lexical input vectors. White nodes represent hidden state vectors.

that the constituent tree is binarized, building hidden nodes from the input words in the bottom-up direction. The leaf node structure is shown in Figure 1(c).

A second salient difference between the two types of LSTMs is the modeling of input words. While each cell in the sequence structure LSTM directly depends on its corresponding input word (Figure 1(a)), only leaf cells in the tree structure LSTM directly depend on corresponding input words (Figure 1(c)). This corresponds well to the constituent tree structure, where there is no direct association between non-leaf constituent nodes and input words. However, it leaves the tree structure a degraded version of a perfect binary-branching variation of the sequence-structure LSTM, with one important source of information (i.e. words) missing in forming a cell (Figure 1(b)).

We fill this gap by proposing an extension to the tree LSTM model, injecting lexical information into every node in the tree. Our method takes inspiration from work on head-lexicalization, which shows that each node in a constituent tree structure is governed by a head word. As shown in Figure 2, the head word for the verb phrase "visited Mary" is "visited", and the head word of the adverb phrase "this afternoon" is "afternoon". Research has shown that head word information can significantly improve the performance of syntactic parsing (Collins, 2003; Clark and Curran, 2004). Correspondingly, we use the head lexical information of each constituent word as the input node x for calculating the corresponding cell c in Figure 1(b).

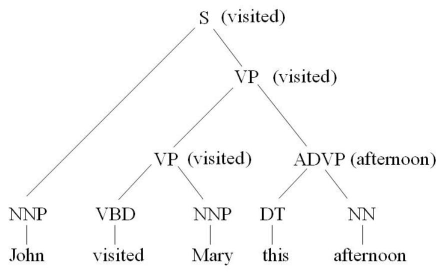


Figure 2: Head-Lexicalized Constituent Tree.

Traditional head-lexicalization relies on specific rules (Collins, 2003; Zhang and Clark, 2009), typically extracting heads from constituent treebanks according to certain grammar formalisms. For better generalization, we use a neural attention mechanism to derive head lexical information automatically, rather than relying on linguistic head rules to find the head lexicon of each constituent, which is language- and formalism-dependent.

Based on such head lexicalization, we further make a bidirectional extension of the tree structured LSTM, propagating information in the top-down direction as well as the bottom-up direction. This is analogous to the bidirectional extension of sequence structured LSTMs, which are commonly used for NLP tasks such as speech recognition (Graves et al., 2013), sentiment analysis (Tai et al., 2015; Li et al., 2015) and machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) tasks.

Results on a standard sentiment classification benchmark and a question type classification benchmark show that our tree LSTM structure gives significantly better accuracies compared with the method of Zhu et al. (2015). We achieve the best reported results for sentiment classification. Interestingly, the head lexical information that is learned automatically from the sentiment treebank consists of both syntactic head information and key sentiment word information. This shows the advantage of automatic head-finding as compared with rule-based head lexicalization. We make our code available under GPL at `https://github.com/zeeeyang/lexicalized_bitreelstm`.

2 Related Work

LSTM Recurrent neural network (RNN) (Elman, 1990; Mikolov et al., 2010) achieves success on modeling linear structures due to its ability to preserve history over arbitrary length sequences. At each step, RNN decides its hidden state based on both the current input and the previous hidden state. In theory, it can carry over unbounded history. Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) is a special type of RNN that leverages multiple gate vectors and a memory cell vector to solve the vanishing and exploding gradient problems of training RNNs. It has been successfully applied to parsing (Vinyals et al., 2015a), sentiment classification (Tai et al., 2015; Li et al., 2015), speech recognition (Graves et al., 2013), machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and image captioning (Vinyals et al., 2015b). There are many variants of sequential LSTMs, such as simple Gated Recurrent Neural Networks (Cho et al., 2014). Greff et al. (2017) compared various architectures of LSTM. In this paper, we take the standard LSTM with peephole connections (Gers and Schmidhuber, 2000) as a baseline.

Structured LSTM There has been a line of research that extends the standard sequential LSTM in order to model more complex structures. Kalchbrenner et al. (2016) proposed Grid LSTMs to process multi-dimensional data. Theis and Bethge (2015) proposed Spatial LSTMs to handle image data. Dyer et al. (2015) designed Stack LSTMs by adding a top pointer to sequential LSTMs to deal with *push* and *pop* sequences of a stack. Tai et al. (2015), Zhu et al. (2015) and Le and Zuidema (2015) extended sequential LSTMs to Tree-Structured LSTMs (**Tree LSTMs**) by adding branching factors. Experiments demonstrated that Tree LSTMs can outperform competitive LSTM baselines on several tasks, such as semantic relatedness prediction and sentiment classification. Li et al. (2015) further investigated the effectiveness of Tree LSTMs on various tasks and discussed when Tree LSTMs are necessary. In addition, Li et al. (2016) employed graph gated units to model graph-based structures.

Tree LSTM The idea of extending linear recurrent structures to tree recurrent structures is reminiscent of extending Recurrent Neural Network to Recursive Neural Network (ReNN) (Socher et al., 2013b; Le and Zuidema, 2014) to support information flow over trees. In addition to Tai et al. (2015), Zhu et al. (2015) and Le and Zuidema (2015), who explicitly named their models as Tree LSTMs, Cho et al. (2014) designed gated recurrent units over tree structures, and Chen et al. (2015) introduced gate mechanisms to recursive neural networks. These can also be regarded as variants of Tree LSTMs.

Both Zhu et al. (2015) and Le and Zuidema (2015) proposed *Binary Tree LSTM* models, which can be applied to situations where there are exactly two children of each internal node in a tree. The difference between Zhu et al. (2015) and Le and Zuidema (2015) is that besides using two forget gates, Le and Zuidema (2015) also make use of two input gates to let a node know its sibling. Tai et al. (2015) introduced *Child-Sum Tree LSTM* and *N-ary Tree LSTM*. Child-Sum Tree LSTMs can support multiple children, while N-ary Tree LSTMs work for trees with a branching factor of at most N. In this perspective, Binary Tree LSTM is a special case of N-ary Tree LSTM with $N = 2$.

When a Child-Sum Tree LSTM is applied to a dependency tree, it is referred to as a *Dependency Tree LSTM*. A *Binary Tree LSTM* is also referred to as a *Constituent Tree LSTM*. Based on Tai et al. (2015), Miwa and Bansal (2016) introduced a Tree LSTM model that can handle different types of children. A dependency tree naturally contains lexical information at every node, while only leaf nodes contain lexical information in a constituent tree. None of these methods (Tai et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015) make direct use of lexical input for internal nodes when using constituent Tree LSTMs.

Bi-LSTM Another common extension to sequential LSTM is to include bidirectional information (Graves et al., 2013), which can model history both left-to-right and right-to-left. The aforementioned Tree LSTM models (Tai et al., 2015; Zhu et al., 2015; Le and Zuidema, 2015) propagate the history of children to their parent in the bottom-up direction only, while ignoring the top-down information flow from parents to children. Zhang et al. (2016) proposed a top-down Tree LSTM to estimate the generation probability of a *dependency* tree. However, no corresponding bottom-up Tree LSTM is incorporated into their model.

Paulus et al. (2014) leveraged bidirectional information over recursive binary trees by propagating global belief down from the tree root to leaf nodes. However, their model is based on recursive neural network rather than LSTM. Miwa and Bansal (2016) adopted a bidirectional Tree LSTM model to jointly extract named entities and relations under a *dependency* tree structure. For *constituent* tree structures, however, their model does not work due to lack of word inputs on non-leaf constituent nodes, and in particular the root node. Our head lexicalization allows us to investigate the top-down constituent Tree LSTM. To our knowledge, we are the first to report a bidirectional constituent Tree LSTM.

3 Baselines

A sequence-structure LSTM estimates a sequence of hidden **state vectors** given a sequence of **input vectors**, through the calculation of a sequence of hidden **cell vectors** using a gate mechanism. For NLP, the input vectors are typically word embeddings (Mikolov et al., 2013), but can also include part-of-speech (POS) embeddings, character embeddings or other types of information. For notational convenience, we refer to the input vectors as **lexical vectors**.

Formally, given an input vector sequence $x_1, x_2, \ldots, x_n$, each state vector h_t is estimated from the Hadamard product of a cell vector c_t and a corresponding **output gate vector** o_t

$$h_t = o_t \otimes \tanh(c_t) \tag{1}$$

Here the cell vector depends on both the previous cell vector c_t, and a combination of the previous state vector h_{t-1}; the current input vector x_t:

$$\begin{aligned} c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\ g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \end{aligned} \tag{2}$$

The combination of c_{t-1} and g_t is controlled by the Hadamard product between a **forget gate** vector f_t and an **input gate** vector i_t, respectively. The gates o_t, f_t and i_t are defined as follows

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \end{aligned} \tag{3}$$

where σ is the sigmoid function. W_{xg}, W_{hg}, b_g, W_{xi}, W_{hi}, W_{ci}, b_i, W_{xf}, W_{hf}, W_{cf}, b_f, W_{xo}, W_{ho}, W_{co} and b_o are model parameters.

The bottom-up Tree LSTM of Zhu et al. (2015) extends the left-to-right sequence LSTM by splitting the *previous* state vector h_{t-1} into a *left child* state vector h^L_{t-1} and a *right child* state vector h^R_{t-1}, and the previous cell vector c_{t-1} into a left child cell vector c^L_{t-1} and a right child cell vector c^R_{t-1}, calculating c_t as

$$c_t = f^L_t \otimes c^L_{t-1} + f^R_t \otimes c^R_{t-1} + i_t \otimes g_t, \tag{4}$$

and the input/output gates i_t/o_t as

$$\begin{aligned} i_t &= \sigma\Big(\sum_{N\in\{L,R\}} (W^N_{hi}h^N_{t-1} + W^N_{ci}c^N_{t-1}) + b_i\Big) \\ o_t &= \sigma\Big(\sum_{N\in\{L,R\}} W^N_{ho}h^N_{t-1} + W_{co}c_t + b_o\Big) \end{aligned} \tag{5}$$

The forget gate f_t is split into f^L_t and f^R_t for regulating c^L_{t-1} and c^R_{t-1}, respectively:

$$\begin{aligned} f^L_t &= \sigma\Big(\sum_{N\in\{L,R\}} (W^N_{hf_l}h^N_{t-1} + W^N_{cf_l}c^N_{t-1}) + b_{f_l}\Big) \\ f^R_t &= \sigma\Big(\sum_{N\in\{L,R\}} (W^N_{hf_r}h^N_{t-1} + W^N_{cf_r}c^N_{t-1}) + b_{f_r}\Big) \end{aligned} \tag{6}$$

g_t depends on both h^L_{t-1} and h^R_{t-1}, but as shown in Figure 1 (b), it does not depend on x_t

$$g_t = \tanh\Big(\sum_{N\in\{L,R\}} W^N_{hg}h^N_{t-1} + b_g\Big) \tag{7}$$

Finally, the hidden state vector h_t is calculated in the same way as in the sequential LSTM model shown in Equation 1. W^L_{hi}, W^R_{hi}, W^L_{ci}, W^R_{ci}, b_i, W^L_{ho}, W^R_{ho}, W_{co}, b_o, $W^L_{hf_l}$, $W^R_{hf_l}$, $W^L_{cf_l}$, $W^R_{cf_l}$, b_{f_l}, $W^L_{hf_r}$, $W^R_{hf_r}$, $W^L_{cf_r}$, $W^R_{cf_r}$, b_{f_r}, W^L_{hg}, W^R_{hg} and b_g are model parameters.

4 Our Model

4.1 Head Lexicalization

We introduce an input lexical vector x_t to the calculation of each cell vector c_t via a bottom-up head propagation mechanism. As shown in the shaded nodes in Figure 3 (b), the head propagation mechanism is parallel to the cell propagation mechanism. In contrast, the method of Zhu et al. (2015) in Figure 3 (a) does not have the input vector x_t for non-leaf constituents.

There are multiple ways to choose a head lexicon for a given binary-branching constituent. One

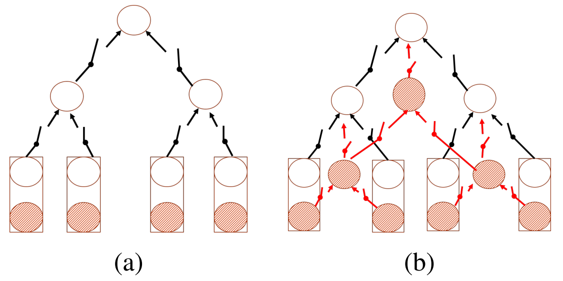


Figure 3: Contrast between Zhu et al. (2015) (a) and this paper (b). Shaded nodes represent lexical input vectors. White nodes represent hidden state vectors.

simple method is to choose the head lexicon of the left child as the head (left-headedness). Correspondingly, an alternative is to use the right child for head lexicon. There is less consistency in the governing head lexicons across variations of the same type of constituents with slightly different typologies. Hence, simple baselines can be less effective compared to linguistically motivated head findings.

Rather than selecting head lexicons using manually-defined head-finding rules, which are language- and formalism-dependent (Collins, 2003), we cast head finding as a part of the neural network model, learning the head lexicon of each constituent by a gated combination of the head lexicons of its two children[1]. Formally,

$$x_t = z_t \otimes x_{t-1}^L + (1 - z_t) \otimes x_{t-1}^R, \quad (8)$$

where x_t represents the head lexicon vector of the current constituent, x_{t-1}^L represents the head lexicon of its left child constituent, and x_{t-1}^R represents the head lexicon of its right child constituent. The gate z_t is calculated based on x_{t-1}^L and x_{t-1}^R,

$$z_t = \sigma(W_{zx}^L x_{t-1}^L + W_{zx}^R x_{t-1}^R + b_z) \quad (9)$$

Here W_{zx}^L, W_{zx}^R and b_z are model parameters.

4.2 Lexicalized Tree LSTM

Given head lexicon vectors for nodes, the Tree LSTM of Zhu et al. (2015) can be extended by leveraging x_t in calculating the corresponding c_t. In particular, x_t is used to estimate the input (i_t), output (o_t) and forget (f_t^R and f_t^L) gates:

$$\begin{aligned}
i_t &= \sigma\Big(\mathbf{W_{xi}x_t} + \\
&\quad \sum_{N\in\{L,R\}} (W_{hi}^N h_{t-1}^N + W_{ci}^N c_{t-1}^N) + b_i\Big) \\
f_t^L &= \sigma\Big(\mathbf{W_{xf}x_t} + \\
&\quad \sum_{N\in\{L,R\}} (W_{hf_l}^N h_{t-1}^N + W_{cf_l}^N c_{t-1}^N) + b_{f_l}\Big) \\
f_t^R &= \sigma\Big(\mathbf{W_{xf}x_t} + \\
&\quad \sum_{N\in\{L,R\}} (W_{hf_r}^N h_{t-1}^N + W_{cf_r}^N c_{t-1}^N) + b_{f_r}\Big) \\
o_t &= \sigma\Big(\mathbf{W_{xo}x_t} + \\
&\quad \sum_{N\in\{L,R\}} W_{ho}^N h_{t-1}^N + W_{co} c_t + b_o\Big)
\end{aligned} \quad (10)$$

In addition, x_t is also used in computing g_t,

$$g_t = \tanh\Big(\mathbf{W_{xg}x_t} + \sum_{N\in\{L,R\}} W_{hg}^N h_{t-1}^N + b_g\Big) \quad (11)$$

With the new definition of i_t, f_t^R, f_t^L and g_t, the computing of c_t remains the same as the baseline Tree LSTM model as shown in Equation 4. Similarly, h_t remains the Hadamard product of c_t and the new o_t as shown in Equation 1.

In this model, W_{xi}, W_{xf}, W_{xg} and W_{xo} are newly-introduced model parameters. The use of x_t in computing the gate and cell values are consistent with those in the baseline sequential LSTM.

4.3 Bidirectional Extensions

Given a sequence of input vectors $[x_1, x_2, \ldots, x_n]$, a bidirectional *sequential* LSTM (Graves et al., 2013) computes two sets of hidden state vectors, $[\tilde{h}_1, \tilde{h}_2, \ldots, \tilde{h}_n]$ and $[\tilde{h}'_n, \tilde{h}'_{n-1}, \ldots, \tilde{h}'_1]$ in the left-to-right and the right-to-left directions, respectively. The final hidden state h_i of the input x_i is the concatenation of the corresponding state vectors in the two LSTMs,

$$h_i = \tilde{h}_i \oplus \tilde{h}'_{n-i+1} \quad (12)$$

The two LSTMs can share the same model parameters or use different parameters. We choose the latter in our baseline experiments.

We make a bidirectional extension to the Lexicalized Tree LSTM in Section 4.2 by following the sequential LSTMs in Section 3, adding an additional

[1] In this paper, we work on binary trees only, which is a common form for CKY and shift-reduce parsing. Typical binarization methods, such as head binarization (Klein and Manning, 2003) , also rely on specific head-finding rules.

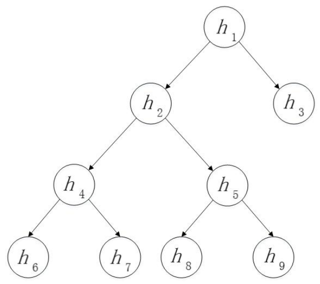


Figure 4: Top-down tree LSTM.

set of hidden state vectors in the top-down direction. Different from the bottom-up direction, each hidden state in the top-down LSTM has exactly one predecessor. In fact, the path from the root of a tree down to any node forms a sequential LSTM.

Note, however, that two different sets of model parameters are used when the current node is the left and the right child of its predecessor. Denoting the two sets of parameters as $\mathbf{U}_L$ and $\mathbf{U}_R$, respectively, the hidden state vector h_7 in Figure 4 is calculated from the hidden state vector h_1 using the parameter set sequence $[\mathbf{U}_L, \mathbf{U}_L, \mathbf{U}_R]$. Similarly, h_8 is calculated from h_1 using $[\mathbf{U}_L, \mathbf{U}_R, \mathbf{U}_L]$. At each step t, the computing of h_t follows the sequential LSTM model:

$$\begin{aligned} h_t &= o_t \otimes \tanh(c_{t-1}) \\ c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\ g_t &= \tanh(W^N_{xg\downarrow} x_{t-1} + W^N_{hg\downarrow} h_{t-1} + b^N_{g\downarrow}) \end{aligned} \tag{13}$$

With the gate values being defined as:

$$\begin{aligned} i_t &= \sigma(W^N_{xi\downarrow} x_t + W^N_{hi\downarrow} h_{t-1} + W^N_{ci\downarrow} c_{t-1} + b^N_{i\downarrow}) \\ f_t &= \sigma(W^N_{xf\downarrow} x_t + W^N_{hf\downarrow} h_{t-1} + W^N_{cf\downarrow} c_{t-1} + b^N_{f\downarrow}) \\ o_t &= \sigma(W^N_{xo\downarrow} x_t + W^N_{ho\downarrow} h_{t-1} + W^N_{co\downarrow} c_t + b^N_{o\downarrow}) \end{aligned} \tag{14}$$

Here $N \in \{L, R\}$ and $\mathbf{U}_N = \{W^N_{xg\downarrow}, W^N_{hg\downarrow}, b^N_{g\downarrow}, W^N_{xi\downarrow}, W^N_{hi\downarrow}, W^N_{ci\downarrow}, b^N_{i\downarrow}, W^N_{xf\downarrow}, W^N_{hf\downarrow}, W^N_{cf\downarrow}, b^N_{f\downarrow}, W^N_{xo\downarrow}, W^N_{ho\downarrow}, W^N_{co\downarrow}, b^N_{o\downarrow}\}$. $\mathbf{U}_L$ and $\mathbf{U}_R$ are model parameters in the top-down Tree LSTM.

One final note is that the top-down Tree LSTM is enabled by the head propagation mechanism, which allows a head lexicon node to be made available for the root constituent node. Without such information, it would be difficult to build top-down LSTM for constituent trees.

5 Usage for Classification

We apply the bidirectional Tree LSTM to classification tasks, where the input is a sentence with its binarized constituent tree, and the output is a discrete label. We denote the bottom-up hidden state vector of the root as $\tilde{h}_{ROOT\uparrow}$, the top-down hidden state vector of the root as $\tilde{h}_{ROOT\downarrow}$ and the top-down hidden state vectors of the input words x_1, x_2, ..., x_n as $\tilde{h}'_1$, $\tilde{h}'_2$, ..., $\tilde{h}'_n$. We take the concatenation of $\tilde{h}_{ROOT\uparrow}$, $\tilde{h}_{ROOT\downarrow}$ and the average of $\tilde{h}'_1, \tilde{h}'_2, \dots, \tilde{h}'_n$ as the final representation h of the sentence:

$$h = \tilde{h}_{ROOT\uparrow} \oplus \tilde{h}_{ROOT\downarrow} \oplus \frac{1}{n}\sum_{i=1}^{n} \tilde{h}'_i \tag{15}$$

A softmax classifier is used to predict the probability p_j of sentiment label j from h by

$$\begin{aligned} h_l &= \mathrm{ReLU}(W_{hl} h + b_{hl}) \\ P &= \mathit{softmax}(W_{lp} h_l + b_{lp}) \\ p_j &= P[j], \end{aligned} \tag{16}$$

where W_{hl}, b_{hl}, W_{lp} and b_{lp} are model parameters, and ReLU is the rectifier function $f(x) = \max(0, x)$. During prediction, the largest probability component of P will be taken as the answer.

6 Training

We train our classifier to maximize the conditional log-likelihood of gold labels of training samples. Formally, given a training set of size $|D|$, the training objective is defined by

$$L(\Theta) = -\sum_{i=1}^{|D|} \log p_{y_i} + \frac{\lambda}{2}||\Theta||^2, \tag{17}$$

where Θ is the set of model parameters, λ is a regularization parameter, y_i is the gold label of the i-th training sample and p_{y_i} is obtained according to Equation 16. For sequential LSTM models, we collect errors over each sequence. For Tree LSTMs, we sum up errors at every node.

The model parameters are optimized using ADAM (Kingma and Ba, 2015) without gradient clipping, with the default hyper-parameters of the AdamTrainer in the *Dynet* toolkits.[2] We also use dropout (Srivastava et al., 2014) at lexical input

[2]https://github.com/clab/dynet

embeddings with a fixed probability p_{drop} to avoid overfitting. p_{drop} is set to 0.5 for all tasks.

Following Tai et al. (2015), Li et al. (2015), Zhu et al. (2015) and Le and Zuidema (2015), we use *Glove-300d* word embeddings[3] to train our model. The pretrained word embeddings are fine-tuned for all tasks. Unknown words are handled in two steps. First, if a word is not contained in the pretrained word embeddings, but its lowercased form exists in the embedding table, we use the lowercase as a replacement. Second, if both the original word and its lowercased form cannot be found, we treat the word as *unk*. The embedding vector of the UNK token is initialized as the average of all embedding vectors.

We use one hidden layer and the same dimensionality settings for both sequential and Tree LSTMs. LSTM hidden states are of size 150. The output hidden size is 128 and 64 for the sentiment classification task and the question type classification task, respectively. Each model is trained for 30 iterations. The same training procedure repeats five times using different random seeds, with parameters being evaluated at the end of every iteration on the development set. The model that gives the best development result is used for final tests.

7 Experiments

The effectiveness of our model is tested mainly on a sentiment classification task and a question type classification task.

7.1 Tasks

Sentiment Classification. For sentiment classification, we use the same data settings as Zhu et al. (2015). Specifically, we use the Stanford Sentiment Treebank (Socher et al., 2013b). Each sentence is annotated with a constituent tree. Every internal node corresponds to a phrase. Each node is manually assigned an integer sentiment label from 0 to 4, that correspond to five sentiment classes: very negative, negative, neutral, positive and very positive, respectively. The root label represents the sentiment label of the whole sentence.

We perform both binary classification and fine-grained classification. Following previous work, we use labels of all phrases for training. Gold-standard tree structures are used for training and testing (Le and Zuidema, 2015; Li et al., 2015; Zhu et al., 2015; Tai et al., 2015). Accuracies are evaluated for both the sentence root labels and phrase labels.

Question Type Classification. For the question type classification task, we use the TREC data (Li and Roth, 2002). Each training sample in this dataset contains a question sentence and its corresponding question type. We work on the six-way coarse classification task, where the six question types are *ENTY*, *HUM*, *LOC*, *DESC*, *NUM* and *ABBR*, corresponding to *ENTITY*, *HUMAN*, *LOCATION*, *DESCRIPTION*, *NUMERIC VALUE* and *ABBREVIATION*, respectively. For example, the type for the sentence “What year did the Titanic sink?” is NUM. The training set consists of 5,452 examples and the test set contains 500 examples. Since there is no development set, we follow Zhou et al. (2015), randomly extracting 500 examples from the training set as a development set. Unlike the sentiment treebank, there is no annotated tree for each sentence. Instead, we obtain an automatically parsed tree for each sentence using ZPar[4] off-the-shelf (Zhang and Clark, 2011). Another difference between the TREC data and the sentiment treebank is that there is only one label, at the root node, rather than a label for each phrase.

7.2 Baselines

We consider two models for our baselines. The first is bidirectional LSTM (**BiLSTM**) (Hochreiter and Schmidhuber, 1997; Graves et al., 2013). Our bidirectional constituency Tree LSTM (**BiConTree**) is compared against BiLSTM to investigate the effectiveness of tree structures. For the sentiment task, following Tai et al. (2015) and Li et al. (2015), we convert the treebank into sequences to allow the bidirectional LSTM model to make use of every phrase span as a training example. The second baseline model is the bottom-up Tree LSTM model of Zhu et al. (2015). We compare this model with our lexicalized bidirectional models to show the effects of adding head lexicalization and top-down information flow.

[3]http://nlp.stanford.edu/data/glove.840B.300d.zip

[4]https://github.com/SUTDNLP/ZPar, version 7.5

Model	5-class		binary	
	Root	Phrase	Root	Phrase
RNTN(Socher et al., 2013b)	45.7	80.7	85.4	87.6
BiLSTM(Li et al., 2015)	49.8	83.3	86.7	-
DepTree(Tai et al., 2015)	48.4	-	85.7	-
ConTree(Le and Zuidema, 2015)	49.9	-	88.0	-
ConTree(Zhu et al., 2015)	50.1	-	-	-
ConTree(Li et al., 2015)	50.4	83.4	86.7	-
ConTree(Tai et al., 2015)	51.0	-	88.0	-
BiLSTM (Our implementation)	49.9	82.7	87.6	91.8
ConTree (Our implementation)	51.2	83.0	88.5	92.5
Top-down ConTree	51.0	82.9	87.8	92.1
ConTree + Lex	52.8	83.2	89.2	92.3
BiConTree	**53.5**	**83.5**	**90.3**	**92.8**

Table 1: Test set accuracies for sentiment classification tasks.

7.3 Main Results

Table 1 shows the main results for the sentiment classification task, where **RNTN** is the recursive neural tensor model of Socher et al. (2013b), **ConTree** and **DepTree** denote constituency Tree LSTMs and dependency Tree LSTMs, respectively. Our reimplementations of sequential bidirectional LSTM and constituent Tree LSTM (Zhu et al., 2015) give comparable results to the original implementations.

After incorporating head lexicalization into our constituent Tree LSTM, the fine-grained sentiment classification accuracy increases from 51.2 to 52.8, and the binary sentiment classification accuracy increases from 88.5 to 89.2, which demonstrates the effectiveness of the head lexicalization mechanism.

Table 1 also shows that a vanilla top-down ConTree LSTM by head-lexicalization (i.e. the top-down half of the final bidirectional model) alone obtains comparable accuracies to the bottom-up ConTree LSTM model. The **BiConTree** model can further improve the classification accuracies by 0.7 points (fine-grained) and 1.3 points (binary) compared to the unidirectional bottom-up lexicalized ConTree LSTM model, respectively.

Table 1 includes 5 class accuracies for all nodes. There is no significant difference between different models, consistent with the observation of Li et al. (2015). To our knowledge, these are the best reported results for this sentiment classification task.

Table 2 shows the question type classification results. Our final model gives better results compared to the BiLSTM model and the bottom-up ConTree model, achieving comparable results to the state-of-the-art SVM classifier with carefully designed features.

Model	Accuracy
Baseline BiLSTM	93.8
Baseline BottomUp ConTree LSTM	93.4
SVM (Silva et al., 2011)	**95.0**
Bidirectional ConTree LSTM	94.8

Table 2: TREC question type classification results.

Model	ConTree	ConTree+Lex	BiConTree
Time (s)	4,664	7,157	11,434

Table 3: Averaged training time over 30 iterations.

7.4 Training Time and Model Size

Introducing head lexicalization and bidirectional extension to the model increases the model complexity. In this section, we analyze training time and model size with the fine-grained sentiment classification task.

We run all the models using an i7-4790 3.60GHz CPU with a single thread. Table 3 shows the average running time for different models over 30 iterations. The baseline **ConTree** model takes about 1.3 hours to finish the training procedure. **ConTree+Lex** takes about 1.5 times longer than **ConTree**. **BiConTree** takes about 3.2 hours, which is about 2.5 times longer than that of **ConTree**.

Table 4 compares the model sizes. We did not count the number of parameters in the lookup table since these parameters are the same for all models. Because the size of LSTM models mainly depends on the dimensionality of the state vector h, we change the size of h to study the effect of model size. When $|h| = 150$, the model size of the baseline model **ConTree** is the smallest, which consists of about 538K parameters. The model size of **ConTree+Lex** is about 1.4 times as large as that of the baseline model. The bidirectional model **BiConTree** is the largest, about 1.7 times as large as that of the **ConTree+Lex** model. However, this parameter set is not very large compared to the modern memory capacity, even for a computer with 16GB RAM. In conclusion, in terms of both time, number of parameters and accuracy, head lexicalization method is

Model	$\vert h\vert$	#Parameter	Accuracy (%)
ConTree	150	538,223	51.2
ConTree+Lex	75	376,673	51.5
ConTree+Lex	150	763,523	52.8
ConTree+Lex	215	1,253,493	52.5
ConTree+Lex	300	2,110,973	51.8
BiConTree	75	564,923	52.6
BiConTree	150	1,297,523	53.5

Table 4: Effect of model size.

a good choice.

Table 4 also helps to clarify whether the gain of the **BiConTree** model over the **ConTree+Lex** model is from the top-down information flow or more parameters. For the same model, increasing the model size can improve the performance to some extent. For example, doubling the size of $|h|$ ($75 \rightarrow 150$) increases the performance from 51.5 to 52.8 for the **ConTree+Lex** model. Similarly, we boost the performance of the **BiConTree** model when doubling the size of $|h|$ from 75 to 150. However, doubling the size of $|h|$ from 150 to 300 empirically decreases the performance of the **ConTree+Lex** model. The size of the **BiConTree** model with $|h| = 75$ is much smaller than that of the **ConTree+Lex** model with $|h| = 150$. However the performance of these two models is quite close, which indicates that top-down information is useful even for a small model. A **ConTree+Lex** model with $|h| = 215$ and a **BiConTree** model with $|h| = 150$ are of similar size. The performance of the **ConTree+Lex** model is again worse than that of the **BiConTree** model (52.5 v.s. 53.5), which shows the effectiveness of top-down information.

7.5 Head Lexicalization Methods

In this experiment, we investigate the effect of our head lexicalization method over heuristic baselines. We consider three baseline methods, namely left branching (*L*), right branching (*R*) and averaging (*A*). For *L*, a parent node accepts lexical information of its left child while ignoring the right child. Correspondingly, for *R*, a parent node accepts lexical information of its right child while ignoring the left child. For *A*, a parent node takes the average of the lexical vectors of its children.

Table 5 shows the accuracies on the test set, where *G* denotes our gated head lexicalization method described in Section 4.1. *R* gives better results compared to *L* due to relatively more right-branching structures in this treebank. A simple average yields similar results compared with right branching. In contrast, *G* outperforms *A* method by considering the relative weights of each branch according to tree-level contexts.

Method	L	R	A	G
Root Accuracy (%)	51.1	51.6	51.8	53.5

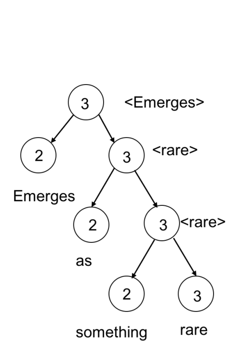
Table 5: Test set accuracies of four head lexicalization methods on fine-grained classification.

We then investigate what lexical heads can be learned by *G*. Interestingly, the lexical heads contain both syntactic and sentiment information. Some heads correspond well to syntactic rules (Collins, 2003), others are driven by subjective words. Compared to Collins' rules, our method found 30.68% and 25.72% overlapping heads on the development and test sets, respectively.

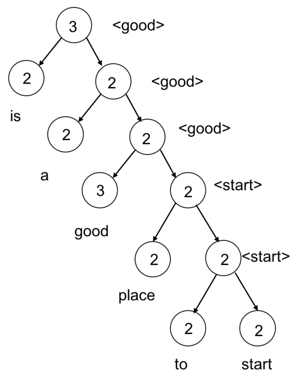
Based on the cosine similarity between the head lexical vector and its children, we visualize the head of a node by choosing the head of the child that gives the largest similarity value. Figure 5 shows some examples, where $<>$ indicates head words, sentiment labels (e.g. 2, 3) are also included. In Figure 5a, "Emerges" is the syntactic head word of the whole phrase, which is consistent with Collins-style head finding. However, "rare" is the head word of the phrase "something rare", which is different from the syntactic head. Similar observations are found in Figure 5b, where "good" is the head word of the whole phrase, rather than the syntactic head "place". The sentiment label of "good" and the sentiment label of the whole phrase are both 3. Figure 5c shows more complex interactions between syntax and sentiment for deciding the head word.

7.6 Error Analysis

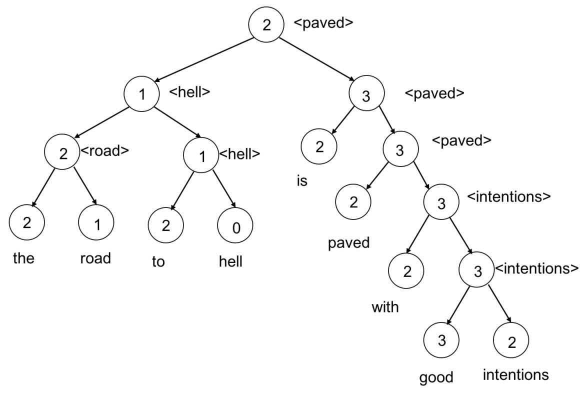
Table 6 shows some example sentences incorrectly predicted by the baseline bottom-up tree model, but correctly labeled by our final model. The head word of sentence #1 by our model is "Gloriously", which is consistent with the sentiment of the whole sentence. This shows how head lexicalization can affect sentiment classification results. Sentences #2 and #3 show the usefulness of top-down informa-



(a) "Emerges as something rare".



(b) "is a good place to start".



(c) "the road to hell is paved with good intentions".

Figure 5: Visualizing head words found automatically by our model.

ID	sentence	baseline	our model
1	Gloriously goofy (and gory) midnight movie stuff .	negative	positive
2	The film is really not so much bad as bland .	positive	negative
3	This is a good movie in spurts , but when it does n't work , it 's at important times .	positive	neutral

Table 6: Example output sentences.

tion for complex semantic structures, where compositionality has subtle effects. Our final model improves the results for the 'very negative' and 'very positive' classes by 10% and 11%, respectively. It also boosts the accuracies for sentences with negation (e.g. "not", "no", and "none") by 4.4%.

Figure 6 shows the accuracy distribution according to the sentence length. We find that our model can improve the classification accuracy for longer sentences (>30 words) by 3.5 absolute points compared to the baseline ConTree LSTM of Zhu et al. (2015), which demonstrates the strength of our model for handling long range information. By considering bidirectional information over tree structures, our model is aware of more contexts for making better predictions.

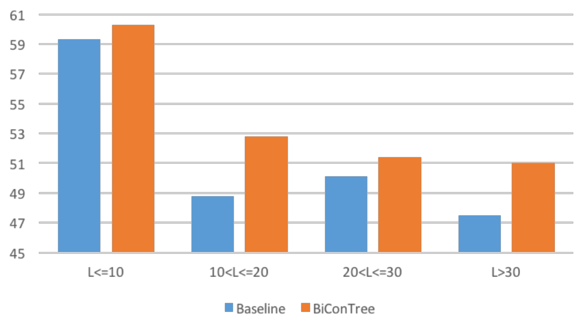


Figure 6: Distribution of 5-class accuracies at the root level according to the sentence length.

8 Applications

Our main results are obtained on semantic-driven sentence classification tasks, where the automatically-learned head words contain mixed syntactic and semantic information. To further investigate the effectiveness of automatically learned head information on a pure syntactic task, we additionally conduct a simple parser reranking experiment. Further, we discuss findings in language modeling by Kuncoro et al. (2017) on the model of recurrent neural network grammars (Dyer et al., 2016). Finally, we show potential future work leveraging our idea for more tasks.

8.1 Syntactic Parsing

We use our tree LSTM models to rerank the 10 best outputs of the Charniak (2000) parser. Given a sentence x, suppose that $Y(x)$ is a set of parse tree candidates generated by a baseline parser for x, the goal of a syntactic reranker is to choose the best parsing hypothesis $\hat{y}$ according to a score function $f(x, y; \Theta)$. Formally,

$$\hat{y} = \arg\max_{y \in Y(x)} \{f(x, y; \Theta)\} \quad (18)$$

For each tree y of sentence x, we follow Socher et al. (2013a) and define the score $f(x, y; \Theta)$ as the sum of scores of each constituent node,

$$f(x, y; \Theta) = \sum_{r \in \text{node}(x,y)} Score(r; \Theta) \tag{19}$$

Without loss of generality, we take a binary node as an example. Given a node A, suppose that its two children are B and C. Let the learned composition state vectors of A, B and C by our proposed Tree-LSTM model be n_A, n_B and n_C, respectively. The head word vector of node A is h_A. $Score(A; \Theta)$ is defined as:

$$\begin{aligned} o_A^{BC} &= ReLU(W_s^L n_B + W_s^R n_C + W_s^H h_A + b^s) \\ \text{Score}_A^{BC} &= \log(softmax(o_A^{BC}))[A], \end{aligned} \tag{20}$$

where W_s^L, W_s^R and b^s are model parameters.

Training. Given a training instance $\langle x_i, Y(x_i) \rangle$ in the training set D, we use a max-margin loss function to train our reranking model. Suppose that the oracle parse tree in $Y(x_i)$ is y_i, the loss function $L(\Theta)$ is

$$L(\Theta) = \frac{1}{|D|} \sum_{i=1}^{|D|} r_i(\Theta) + \frac{\lambda}{2} ||\Theta||^2 \tag{21}$$

Here λ is a regularization parameter and $r_i(\Theta)$ is the margin loss between y_i and the highest score tree $\hat{y}_i$ predicted by the reranking model. $r_i(\Theta)$ is given by

$$\begin{aligned} r_i(\Theta) = \max_{\hat{y}_i \in Y(x_i)} &(0, f(x_i, \hat{y}_i; \Theta) + \\ &\Delta(y_i, \hat{y}_i) - f(x_i, y_i; \Theta)), \end{aligned} \tag{22}$$

where $\Delta(y_i, \hat{y}_i)$ is the structure loss between y_i and $\hat{y}_i$ by counting the number of incorrect nodes in the oracle tree:

$$\Delta(y_i, \hat{y}_i) = \sum_{node \in \hat{y}_i} \kappa \mathbf{1}\{node \notin y_i\}. \tag{23}$$

κ is a scalar. With this loss function, we require the score of the oracle tree to be higher than the other candidates by a score margin. Intuitively, the score of the y_i will increase and the score of $\hat{y}_i$ will decrease during training.

Results. We experiment on the WSJ portion of the Penn Treebank, following the standard split (Collins, 2003). Sections 2-21 are used for training, Section 24 and Section 23 are the development set and test set, respectively. The Charniak parser (Charniak, 2000; Charniak and Johnson, 2005) is adopted for our baseline by following the settings of Choe and Charniak (2016).

To obtain N-best lists on the development set and test set, we first train a baseline parser on the training set. To obtain N-best lists on the training data, we split the training data into 20 folds and trained 20 parsers. Each parser was trained on 19 folds data and used to produce the n-best list of the remaining fold. For the neural reranking model, we use the pretrained word vectors from Collobert et al. (2011). The input dimension is 50. The dimension of state vectors in Tree-LSTM model is 60. These parameters are trained with ADAM (Kingma and Ba, 2015) with a batch size of 20. We set $\kappa = 0.1$ for all experiments. For practical reasons, we use the **ConTree+Lex** model to learn the node representations and define $Y(x_i)$ to be the 10-best parsing trees of x_i.

Table 7 shows the reranking results on WSJ test set. The baseline F1 score is 89.7. Our **ConTree** improves the baseline model to 90.6. Using **ConTree+Lex** model can further improve the performance (90.6 $\rightarrow$ 90.9). This suggests that automatic heads can also be useful for a syntactic task. Among neural rerankers, our model outperforms Socher et al. (2013a), but underperforms current state-of-the-art models, including sequence-to-sequence based LSTM language models (Vinyals et al., 2015a; Choe and Charniak, 2016) and recurrent neural network grammars (Dyer et al., 2016). This is likely due to our simple reranking configurations and settings[5]. Nevertheless, it serves our goal of contrasting the tree LSTM models.

8.2 Language Modeling

Kuncoro et al. (2017) investigate composition functions in recurrent neural network grammars (RNNG) (Dyer et al., 2016), finding that syntactic head information can be automatically learned. Their observa-

[5]Dyer et al. (2016) employs 2-layerd LSTMs with input and hidden dimensions of size 256 and 128. Choe and Charniak (2016) use 3-layered LSTMs with both the input and hidden dimensions of size 1500. In addition, we only use the tree LSTM for scoring candidate parses in order to isolate the effect of tree LSTMs. In contrast, the previous works use the complex feature combinations in order to achieve high accuracies, which is different from our goal.

Model	F_1
Baseline (Charniak (2000))	89.7
ConTree	90.6
ConTree+Lex	90.9
Our 10-best Oracle	94.8

Table 7: Reranking results on WSJ test set.

tion is consistent with ours. Formally, an RNNG is a tuple $\langle N, \Sigma, R, S, \Theta \rangle$, where N is the set of non-terminals, Σ is the set of terminals, R is a set of top-down transition-based rules, S is the start symbol and Θ is the set of model parameters. Given S, the derivation process resembles transition-based parsing, which is performed incrementally from left to right. Unlike surface language models, RNNGs model sentences with explicit grammar. Comparing naive sequence-to-sequence models of syntax (Vinyals et al., 2015a), RNNGs have the advantage of explicitly modeling syntactic composition between constituents, by combining the vector representation of child constituents into a single vector representation of their parent using a neural network. Kuncoro et al. (2017) show that such compositions are the key to the success, and further investigate several alternatives neural network structures. In particular, they compare vanilla LSTMs to attention networks when composing child constituents. Interestingly, the attention values represent syntactic heads among the child constituents to some extent. In addition, the vector constituent representation implicitly reflects constituent types. Their finding is consistent with ours in that a neural network can learn pure syntactic head information from constituent vectors.

8.3 Relation Extraction

Our head-lexicalized tree model can be used for all tasks that require representation learning for sentences, given their constituent syntax. One example of future work is relation extraction. For example, given the sentence "John is from Google Inc.", a relation 'works_in' can be extracted between 'John' and 'Google Inc.'.

Miwa and Bansal (2016) solve this task by using the Child-Sum tree representation of Tai et al. (2015) to represent the input sentence, extracting features for the two entities according to their related nodes in the dependency tree, and then conducting relation classification based on these features. Head-lexicalization and top-down information can potentially be useful for improving relation extraction in the framework of Miwa and Bansal (2016).

9 Conclusion

We proposed lexicalized variants for constituent tree LSTMs. Learning the heads of constituents automatically using a neural model, our lexicalized tree LSTM is applicable to arbitrary binary branching trees in CFG, and is formalism-independent. In addition, lexical information on the root further allows a top-down extension to the model, resulting in a bidirectional constituent Tree LSTM. Experiments on two well-known datasets show that head-lexicalization improves the unidirectional Tree LSTM model. In addition, the bidirectional Tree LSTM gives superior labeling results compared to both unidirectional Tree LSTMs and bidirectional sequential LSTMs.

Acknowledgments

We thank the anonymous reviewers for their detailed and constructive comments. 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. In *Proceedings of the 2016 International Conference on Learning Representations*, San Diego, California, USA, May.

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

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

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015. Sentence modeling with gated recursive neural network. In *Proceedings of the 2015 Conference on Empirical Methods in Natural*

Language Processing, pages 793–798, Lisbon, Portugal, September. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–Decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October. Association for Computational Linguistics.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas, USA, November. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and Log-Linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, pages 103–110, Barcelona, Spain, July. Association for Computational Linguistics.

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

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, New York, NY, USA, July. ACM.

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 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, USA, June. Association for Computational Linguistics.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Felix A. Gers and Jürgen Schmidhuber. 2000. Recurrent nets that time and count. In *Proceedings of the 2000 International Joint Conference on Neural Networks*, pages 189–194, Como, Italy, July. IEEE.

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

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–11.

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

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2016. Grid long short-term memory. In *Proceedings of the 2016 International Conference on Learning Representations*, San Juan, Puerto Rico, May.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 2015 International Conference on Learning Representations*, San Diego, USA, May.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1249–1258, Valencia, Spain, April. Association for Computational Linguistics.

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

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 10–19, Denver, Colorado, USA, June. Association for Computational Linguistics.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational linguistics*, pages 1–7, Taipei, Taiwan, August. Association for Computational Linguistics.

Jiwei Li, Thang Luong, Dan Jurafsky, and Eduard Hovy. 2015. When are tree structures necessary for deep learning of representations? In *Proceedings of the 2015 Conference on Empirical Methods in Natural*

Language Processing, pages 2304–2314, Lisbon, Portugal, September. Association for Computational Linguistics.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated graph sequence neural networks. In *Proceedings of the 2016 International Conference on Learning Representations*, San Juan, Puerto Rico, May.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048, Chiba, Japan, September.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Workshop Proceedings of the 2013 International Conference on Learning Representations*, Scottsdale, Arizona, USA, May.

Makoto Miwa and Mohit Bansal. 2016. End-to-End relation extraction using LSTMs on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1105–1116, Berlin, Germany, August. Association for Computational Linguistics.

Romain Paulus, Richard Socher, and Christopher D. Manning. 2014. Global belief recursive neural networks. In *Advances in Neural Information Processing Systems*, pages 2888–2896, Montreal, Quebec, Canada, December.

Joao Silva, Luísa Coheur, Ana Cristina Mendes, and Andreas Wichert. 2011. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154.

Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 26th International Conference on Machine Learning*, pages 129–136, Bellevue, Washington, USA, June-July.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013a. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465, Sofia, Bulgaria, August. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, Montreal, Quebec, Canada, December.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.

Zhiyang Teng, Vo Duy-Tin, and Yue Zhang. 2016. Context-sensitive lexicon features for neural sentiment analysis. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1629–1638, Austin, Texas, USA, November. Association for Computational Linguistics.

Lucas Theis and Matthias Bethge. 2015. Generative image modeling using spatial LSTMs. In *Advances in Neural Information Processing Systems*, pages 1918–1926, Montreal, Quebec, Canada, December.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015a. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2755–2763, Montreal, Quebec, Canada, December.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015b. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, Boston, MA, USA, June.

Yue Zhang and Stephen Clark. 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, Paris, France, October. 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.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down tree long short-term memory networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 310–320, San Diego, California, USA, June. Association for Computational Linguistics.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1301.3781*.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1604–1612, Lille, France, July.