Top-down Tree Long Short-Term Memory Networks

Xingxing Zhang, Liang Lu and **Mirella Lapata**
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
{x.zhang,liang.lu}@ed.ac.uk,mlap@inf.ed.ac.uk

Abstract

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network with a more complex computational unit, have been successfully applied to a variety of sequence modeling tasks. In this paper we develop Tree Long Short-Term Memory (TREELSTM), a neural network model based on LSTM, which is designed to predict a tree rather than a linear sequence. TREELSTM defines the probability of a sentence by estimating the generation probability of its dependency tree. At each time step, a node is generated based on the representation of the generated subtree. We further enhance the modeling power of TREELSTM by explicitly representing the correlations between left and right dependents. Application of our model to the MSR sentence completion challenge achieves results beyond the current state of the art. We also report results on dependency parsing reranking achieving competitive performance.

1 Introduction

Neural language models have been gaining increasing attention as a competitive alternative to n-grams. The main idea is to represent each word using a real-valued feature vector capturing the contexts in which it occurs. The conditional probability of the next word is then modeled as a smooth function of the feature vectors of the preceding words and the next word. In essence, similar representations are learned for words found in similar contexts resulting in similar predictions for the next word. Previous approaches have mainly employed feed-forward (Bengio et al., 2003; Mnih and Hinton, 2007) and recurrent neural networks (Mikolov et al., 2010; Mikolov, 2012) in order to map the feature vectors of the context words to the distribution for the next word. Recently, RNNs with Long Short-Term Memory (LSTM) units (Hochreiter and Schmidhuber, 1997; Hochreiter, 1998) have emerged as a popular architecture due to their strong ability to capture long-term dependencies. LSTMs have been successfully applied to a variety of tasks ranging from machine translation (Sutskever et al., 2014), to speech recognition (Graves et al., 2013), and image description generation (Vinyals et al., 2015).

Despite superior performance in many applications, neural language models essentially predict sequences of words. Many NLP tasks, however, exploit syntactic information operating over tree structures (e.g., dependency or constituent trees). In this paper we develop a novel neural network model which combines the advantages of the LSTM architecture and syntactic structure. Our model estimates the probability of a sentence by estimating the generation probability of its dependency tree. Instead of explicitly encoding tree structure as a set of features, we use four LSTM networks to model four types of dependency edges which altogether specify how the tree is built. At each time step, one LSTM is activated which predicts the next word conditioned on the sub-tree generated so far. To learn the representations of the conditioned sub-tree, we force the four LSTMs to share their hidden layers. Our model is also capable of generating trees just by sampling from a trained model and can be seamlessly integrated with text generation applications.

Our approach is related to but ultimately different from recursive neural networks (Pollack, 1990) a class of models which operate on structured inputs. Given a (binary) parse tree, they recursively generate parent representations in a bottom-up fashion, by combining tokens to produce representations for phrases, and eventually the whole sentence. The learned representations can be then used in classification tasks such as sentiment analysis (Socher et al., 2011b) and paraphrase detection (Socher et al., 2011a). Tai et al. (2015) learn distributed representations over syntactic trees by generalizing the LSTM architecture to tree-structured network topologies. The key feature of our model is not so much that it can learn semantic representations of phrases or sentences, but its ability to predict tree structure and estimate its probability.

Syntactic language models have a long history in NLP dating back to Chelba and Jelinek (2000) (see also Roark (2001) and Charniak (2001)). These models differ in how grammar structures in a parsing tree are used when predicting the next word. Other work develops dependency-based language models for specific applications such as machine translation (Shen et al., 2008; Zhang, 2009; Sennrich, 2015), speech recognition (Chelba et al., 1997) or sentence completion (Gubbins and Vlachos, 2013). All instances of these models apply Markov assumptions on the dependency tree, and adopt standard n-gram smoothing methods for reliable parameter estimation. Emami et al. (2003) and Sennrich (2015) estimate the parameters of a structured language model using feed-forward neural networks (Bengio et al., 2003). Mirowski and Vlachos (2015) re-implement the model of Gubbins and Vlachos (2013) with RNNs. They view sentences as sequences of words over a tree. While they ignore the tree structures themselves, we model them explicitly.

Our model shares with other structured-based language models the ability to take dependency information into account. It differs in the following respects: (a) it does not artificially restrict the depth of the dependencies it considers and can thus be viewed as an infinite order dependency language model; (b) it not only estimates the probability of a string but is also capable of generating dependency trees; (c) finally, contrary to previous dependency-based language models which encode syntactic information as features, our model takes tree structure into account more directly via representing different types of dependency edges explicitly using LSTMs. Therefore, there is no need to manually determine which dependency tree features should be used or how large the feature embeddings should be.

We evaluate our model on the MSR sentence completion challenge, a benchmark language modeling dataset. Our results outperform the best published results on this dataset. Since our model is a general tree estimator, we also use it to rerank the top K dependency trees from the (second order) MSTPasrser and obtain performance on par with recently proposed dependency parsers.

2 Tree Long Short-Term Memory Networks

We seek to estimate the probability of a sentence by estimating the generation probability of its dependency tree. Syntactic information in our model is represented in the form of dependency paths. In the following, we first describe our definition of dependency path and based on it explain how the probability of a sentence is estimated.

2.1 Dependency Path

Generally speaking, a dependency path is the path between ROOT and w consisting of the nodes on the path and the edges connecting them. To represent dependency paths, we introduce four types of edges which essentially define the "shape" of a dependency tree. Let w_0 denote a node in a tree and $w_1, w_2, \ldots, w_n$ its left dependents. As shown in Figure 1, LEFT edge is the edge between w_0 and its first left dependent denoted as (w_0, w_1). Let w_k (with $1 < k \leq n$) denote a non-first left dependent of w_0. The edge from w_{k-1} to w_k is a NX-LEFT edge (NX stands for NEXT), where w_{k-1} is the right adjacent sibling of w_k. Note that the NX-LEFT edge (w_{k-1}, w_k) replaces edge (w_0, w_k) (illustrated with a dashed line in Figure 1) in the original dependency tree. The modification allows information to flow from w_0 to w_k through $w_1, \ldots, w_{k-1}$ rather than directly from w_0 to w_k. RIGHT and NX-RIGHT edges are defined analogously for right dependents.

Given these four types of edges, dependency paths (denoted as $\mathcal{D}(w)$) can be defined as follows

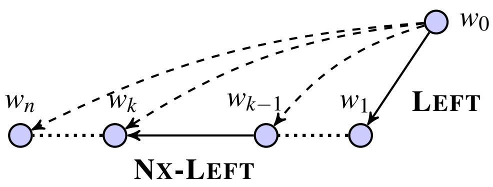

Figure 1: LEFT and NX-LEFT edges. Dotted line between w_1 and w_{k-1} (also between w_k and w_n) indicate that there may be ≥ 0 nodes inbetween.

bearing in mind that the first right dependent of ROOT is its only dependent and that w^p denotes the parent of w. We use $(\dots)$ to denote a sequence, where $()$ is an empty sequence and $\|$ is an operator for concatenating two sequences.

(1) if w is ROOT, then $\mathcal{D}(w) = ()$
(2) if w is a **left dependent** of w^p
  (a) if w is the first left dependent, then $\mathcal{D}(w) = \mathcal{D}(w^p) \| (\langle w^p, \text{LEFT} \rangle)$
  (b) if w is not the first left dependent and w^s is its right adjacent sibling, then $\mathcal{D}(w) = \mathcal{D}(w^s) \| (\langle w^s, \text{NX-LEFT} \rangle)$
(3) if w is a **right dependent** of w^p
  (a) if w is the first right dependent, then $\mathcal{D}(w) = \mathcal{D}(w^p) \| (\langle w^p, \text{RIGHT} \rangle)$
  (b) if w is not the first right dependent and w^s is its left adjacent sibling, then $\mathcal{D}(w) = \mathcal{D}(w^s) \| (\langle w^s, \text{NX-RIGHT} \rangle)$

A dependency tree can be represented by the set of its dependency paths which in turn can be used to reconstruct the original tree.[1]

Dependency paths for the first two levels of the tree in Figure 2 are as follows (ignoring for the moment the subscripts which we explain in the next section). $\mathcal{D}(\text{sold}) = (\langle \text{ROOT}, \text{RIGHT} \rangle)$ (see definitions (1) and (3a)), $\mathcal{D}(\text{year}) = \mathcal{D}(\text{sold}) \| (\langle \text{sold}, \text{LEFT} \rangle)$ (see (2a)), $\mathcal{D}(\text{manufacturer}) = \mathcal{D}(\text{year}) \| (\langle \text{year}, \text{NX-LEFT} \rangle)$ (see (2b)), $\mathcal{D}(\text{cars}) = \mathcal{D}(\text{sold}) \| (\langle \text{sold}, \text{RIGHT} \rangle)$ (see (3a)), $\mathcal{D}(\text{in}) = \mathcal{D}(\text{cars}) \| (\langle \text{cars}, \text{NX-RIGHT} \rangle)$ (according to (3b)).

2.2 Tree Probability

The core problem in syntax-based language modeling is to estimate the probability of sentence S given its corresponding tree T, $P(S|T)$. We view the probability computation of a dependency tree as a generation process. Specifically, we assume dependency trees are constructed top-down, in a breadth-first manner. Generation starts at the ROOT node. For each node at each level, first its left dependents are generated from closest to farthest and then the right dependents (again from closest to farthest). The same process is applied to the next node at the same level or a node at the next level. Figure 2 shows the breadth-first traversal of a dependency tree.

[1]Throughout this paper we assume all dependency trees are projective.

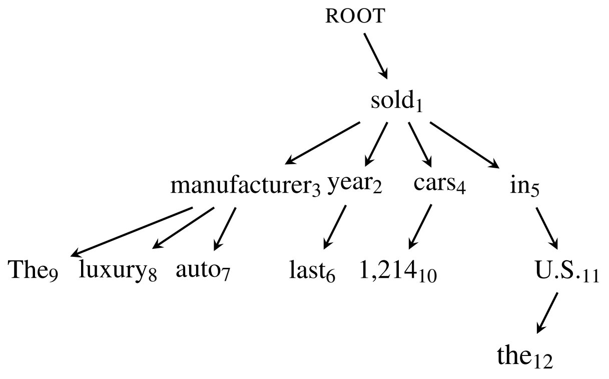

Figure 2: Dependency tree of the sentence *The luxury auto manufacturer last year sold 1,214 cars in the U.S.* Subscripts indicate breadth-first traversal. ROOT has only one dependent (i.e., *sold*) which we view as its first right dependent.

Under the assumption that each word w in a dependency tree is *only* conditioned on its *dependency path*, the probability of a sentence S given its dependency tree T is:

$$P(S|T) = \prod_{w \in \text{BFS}(T) \setminus \text{ROOT}} P(w|\mathcal{D}(w)) \qquad (1)$$

where $\mathcal{D}(w)$ is the dependency path of w. Note that each word w is visited according to its breadth-first search order (BFS(T)) and the probability of ROOT is ignored since every tree has one. The role of ROOT in a dependency tree is the same as the begin of sentence token (BOS) in a sentence. When computing $P(S|T)$ (or $P(S)$), the probability of ROOT (or BOS) is ignored (we assume it always exists), but is used to predict other words. We explain in the next section how TREELSTM estimates $P(w|\mathcal{D}(w))$.

2.3 Tree LSTMs

A dependency path $\mathcal{D}(w)$ is subtree which we denote as a sequence of $\langle$*word*, *edge-type*$\rangle$ tuples. Our

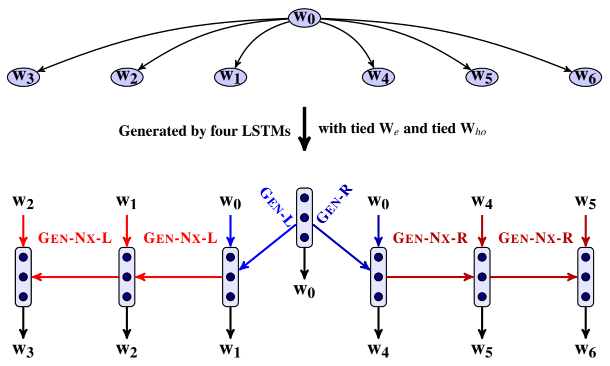

Figure 3: Generation process of left (w_1, w_2, w_3) and right (w_4, w_5, w_6) dependents of tree node w_o (top) using four LSTMs (GEN-L, GEN-R, GEN-NX-L and GEN-NX-R). The model can handle an arbitrary number of dependents due to GEN-NX-L and GEN-NX-R.

innovation is to learn the representation of $\mathcal{D}(w)$ using four LSTMs. The four LSTMs (GEN-L, GEN-R, GEN-NX-L and GEN-NX-R) are used to represent the four types of edges (LEFT, RIGHT, NX-LEFT and NX-RIGHT) introduced earlier. GEN, NX, L and R are shorthands for GENERATE, NEXT, LEFT and RIGHT. At each time step, an LSTM is chosen according to an edge-type; then the LSTM takes a word as input and predicts/generates its dependent or sibling. This process can be also viewed as adding an edge and a node to a tree. Specifically, LSTMs GEN-L and GEN-R are used to generate the first left and right dependent of a node (w_1 and w_4 in Figure 3). So, these two LSTMs are responsible for going deeper in a tree. While GEN-NX-L and GEN-NX-R generate the remaining left/right dependents and therefore go wider in a tree. As shown in Figure 3, w_2 and w_3 are generated by GEN-NX-L, whereas w_5 and w_6 are generated by GEN-NX-R. Note that the model can handle any number of left or right dependents by applying GEN-NX-L or GEN-NX-R multiple times.

We assume time steps correspond to the steps taken by the breadth-first traversal of the dependency tree and the sentence has length n. At time step t ($1 \leq t \leq n$), let $\langle w_{t'}, z_t \rangle$ denote the last tuple in $\mathcal{D}(w_t)$. Subscripts t and t' denote the breadth-first search order of w_t and $w_{t'}$, respectively. $z_t \in \{\text{LEFT}, \text{RIGHT}, \text{NX-LEFT}, \text{NX-RIGHT}\}$ is the edge type (see the definitions in Section 2.1). Let $\mathbf{W}_e \in \mathbb{R}^{s \times |V|}$ denote the word embedding matrix and $\mathbf{W}_{ho} \in \mathbb{R}^{|V| \times d}$ the output matrix of our model, where $|V|$ is the vocabulary size, s the word embedding size and d the hidden unit size. We use tied $\mathbf{W}_e$ and tied $\mathbf{W}_{ho}$ for the four LSTMs to reduce the number of parameters in our model. The four LSTMs also share their hidden states. Let $\mathbf{H} \in \mathbb{R}^{d \times (n+1)}$ denote the *shared* hidden states of all time steps and $e(w_t)$ the one-hot vector of w_t. Then, $\mathbf{H}[:, t]$ represents $\mathcal{D}(w_t)$ at time step t, and the computation[2] is:

$$\mathbf{x}_t = \mathbf{W}_e \cdot e(w_{t'}) \tag{2a}$$

$$\mathbf{h}_t = \text{LSTM}^{z_t}(\mathbf{x}_t, \mathbf{H}[:, t']) \tag{2b}$$

$$\mathbf{H}[:, t] = \mathbf{h}_t \tag{2c}$$

$$\mathbf{y}_t = \mathbf{W}_{ho} \cdot \mathbf{h}_t \tag{2d}$$

where the initial hidden state $\mathbf{H}[:, 0]$ is initialized to a vector of small values such as 0.01. According to Equation (2b), the model selects an LSTM based on edge type z_t. We describe the details of LSTM^{z_t} in the next paragraph. The probability of w_t given its dependency path $\mathcal{D}(w_t)$ is estimated by a *softmax* function:

$$P(w_t | \mathcal{D}(w_t)) = \frac{\exp(\mathbf{y}_{t, w_t})}{\sum_{k'=1}^{|V|} \exp(\mathbf{y}_{t, k'})} \tag{3}$$

We must point out that although we use four jointly trained LSTMs to encode the hidden states, the training and inference complexity of our model is no different from a regular LSTM, since at each time step only one LSTM is working.

We implement LSTM^z in Equation (2b) using a deep LSTM (to simplify notation, from now on we write z instead of z_t). The inputs at time step t are $\mathbf{x}_t$ and $\mathbf{h}_{t'}$ (the hidden state of an earlier time step t') and the output is $\mathbf{h}_t$ (the hidden state of current time step). Let L denote the layer number of LSTM^z and $\hat{\mathbf{h}}_t^l$ the internal hidden state of the l-th layer of the LSTM^z at time step t, where $\mathbf{x}_t$ is $\hat{\mathbf{h}}_t^0$ and $\mathbf{h}_{t'}$ is $\hat{\mathbf{h}}_{t'}^L$. The LSTM architecture introduces multiplicative gates and memory cells $\hat{\mathbf{c}}_t^l$ (at l-th layer) in order to address the *vanishing gradient* problem which makes it difficult for the standard RNN model to learn long-distance correlations in a sequence. Here, $\hat{\mathbf{c}}_t^l$ is a linear combination of the current input signal $\mathbf{u}_t$ and an earlier memory cell $\hat{\mathbf{c}}_{t'}^l$. How much input information $\mathbf{u}_t$ will flow into $\hat{\mathbf{c}}_t^l$ is controlled

[2] We ignore all bias terms for notational simplicity.

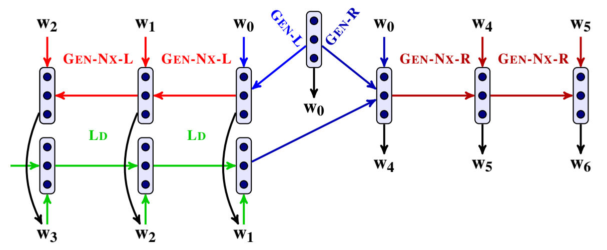

Figure 4: Generation of left and right dependents of node w_0 according to LDTREELSTM.

by input gate $\mathbf{i}_t$ and how much of the earlier memory cell $\hat{\mathbf{c}}^l_{t'}$ will be forgotten is controlled by forget gate $\mathbf{f}_t$. This process is computed as follows:

$$\mathbf{u}_t = \tanh(\mathbf{W}^{z,l}_{ux} \cdot \hat{\mathbf{h}}^{l-1}_t + \mathbf{W}^{z,l}_{uh} \cdot \hat{\mathbf{h}}^l_{t'}) \tag{4a}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^{z,l}_{ix} \cdot \hat{\mathbf{h}}^{l-1}_t + \mathbf{W}^{z,l}_{ih} \cdot \hat{\mathbf{h}}^l_{t'}) \tag{4b}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{z,l}_{fx} \cdot \hat{\mathbf{h}}^{l-1}_t + \mathbf{W}^{z,l}_{fh} \cdot \hat{\mathbf{h}}^l_{t'}) \tag{4c}$$

$$\hat{\mathbf{c}}^l_t = \mathbf{f}_t \odot \hat{\mathbf{c}}^l_{t'} + \mathbf{i}_t \odot \mathbf{u}_t \tag{4d}$$

where $\mathbf{W}^{z,l}_{ux} \in \mathbb{R}^{d\times d}$ ($\mathbf{W}^{z,l}_{ux} \in \mathbb{R}^{d\times s}$ when $l = 1$) and $\mathbf{W}^{z,l}_{uh} \in \mathbb{R}^{d\times d}$ are weight matrices for $\mathbf{u}_t$, $\mathbf{W}^{z,l}_{ix}$ and $\mathbf{W}^{z,l}_{ih}$ are weight matrices for $\mathbf{i}_t$ and $\mathbf{W}^{z,l}_{fx}$, and $\mathbf{W}^{z,l}_{fh}$ are weight matrices for $\mathbf{f}_t$. σ is a sigmoid function and $\odot$ the element-wise product.

Output gate $\mathbf{o}_t$ controls how much information of the cell $\hat{\mathbf{c}}^l_t$ can be seen by other modules:

$$\mathbf{o}_t = \sigma(\mathbf{W}^{z,l}_{ox} \cdot \hat{\mathbf{h}}^{l-1}_t + \mathbf{W}^{z,l}_{oh} \cdot \hat{\mathbf{h}}^l_{t'}) \tag{5a}$$

$$\hat{\mathbf{h}}^l_t = \mathbf{o}_t \odot \tanh(\hat{\mathbf{c}}^l_t) \tag{5b}$$

Application of the above process to all layers L, will yield $\hat{\mathbf{h}}^L_t$, which is $\mathbf{h}_t$. Note that in implementation, all $\hat{\mathbf{c}}^l_t$ and $\hat{\mathbf{h}}^l_t$ ($1 \le l \le L$) at time step t are stored, although we only care about $\hat{\mathbf{h}}^L_t$ ($\mathbf{h}_t$).

2.4 Left Dependent Tree LSTMs

TREELSTM computes $P(w|\mathcal{D}(w))$ based on the dependency path $\mathcal{D}(w)$, which ignores the interaction between left and right dependents on the same level. In many cases, TREELSTM will use a verb to predict its object directly without knowing its subject. For example, in Figure 2, TREELSTM uses ⟨ROOT, RIGHT⟩ and ⟨ *sold*, RIGHT ⟩ to predict *cars*. This information is unfortunately not specific to *cars* (many things can be sold, e.g., *chocolates*, *candy*). Considering *manufacturer*, the left dependent of *sold* would help predict *cars* more accurately.

In order to jointly take left and right dependents into account, we employ yet another LSTM, which goes from the furthest left dependent to the closest left dependent (LD is a shorthand for left dependent). As shown in Figure 4, LD LSTM learns the representation of all left dependents of a node w_0; this representation is then used to predict the first right dependent of the same node. Non-first right dependents can also leverage the representation of left dependents, since this information is injected into the hidden state of the first right dependent and can percolate all the way. Note that in order to retain the generation capability of our model (Section 3.4), we only allow right dependents to leverage left dependents (they are generated before right dependents).

The computation of the LDTREELSTM is almost the same as in TREELSTM except when $z_t = \text{GEN-R}$. In this case, let $\mathbf{v}_t$ be the corresponding left dependent sequence with length K ($\mathbf{v}_t = (w_3, w_2, w_1)$ in Figure 4). Then, the hidden state ($\mathbf{q}_k$) of $\mathbf{v}_t$ at each time step k is:

$$\mathbf{m}_k = \mathbf{W}_e \cdot e(\mathbf{v}_{t,k}) \tag{6a}$$

$$\mathbf{q}_k = \text{LSTM}^{\text{LD}}(\mathbf{m}_k, \mathbf{q}_{k-1}) \tag{6b}$$

where $\mathbf{q}_K$ is the representation for all left dependents. Then, the computation of the current hidden state becomes (see Equation (2) for the original computation):

$$\mathbf{r}_t = \begin{bmatrix} \mathbf{W}_e \cdot e(w_{t'}) \\ \mathbf{q}_K \end{bmatrix} \tag{7a}$$

$$\mathbf{h}_t = \text{LSTM}^{\text{GEN-R}}(\mathbf{r}_t, \mathbf{H}[:, t']) \tag{7b}$$

where $\mathbf{q}_K$ serves as additional input for $\text{LSTM}^{\text{GEN-R}}$. All other computational details are the same as in TreeLSTM (see Section 2.3).

2.5 Model Training

On small scale datasets we employ Negative Log-likelihood (NLL) as our training objective for both TREELSTM and LDTREELSTM:

$$\mathcal{L}^{\text{NLL}}(\theta) = -\frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} \log P(S|T) \tag{8}$$

where S is a sentence in the training set $\mathcal{S}$, T is the dependency tree of S and $P(S|T)$ is defined as in Equation (1).

On large scale datasets (e.g., with vocabulary size of 65K), computing the output layer activations and the *softmax* function with NLL would become prohibitively expensive. Instead, we employ Noise Contrastive Estimation (NCE; Gutmann and Hyvärinen (2012), Mnih and Teh (2012)) which treats the normalization term $\hat{Z}$ in $\hat{P}(w|\mathcal{D}(w_t)) = \frac{\exp(\mathbf{W}_{ho}[w,:]\cdot\mathbf{h}_t)}{\hat{Z}}$ as constant. The intuition behind NCE is to discriminate between samples from a data distribution $\hat{P}(w|\mathcal{D}(w_t))$ and a known noise distribution $P_n(w)$ via binary logistic regression. Assuming that noise words are k times more frequent than real words in the training set (Mnih and Teh, 2012), then the probability of a word w being from our model $P_d(w, \mathcal{D}(w_t))$ is $\frac{\hat{P}(w|\mathcal{D}(w_t))}{\hat{P}(w|\mathcal{D}(w_t))+kP_n(w)}$. We apply NCE to large vocabulary models with the following training objective:

$$\mathcal{L}^{\text{NCE}}(\theta) = -\frac{1}{|\mathcal{S}|}\sum_{T\in\mathcal{S}}\sum_{t=1}^{|T|}\Bigg(\log P_d(w_t, \mathcal{D}(w_t)) + \sum_{j=1}^{k}\log[1 - P_d(\tilde{w}_{t,j}, \mathcal{D}(w_t))]\Bigg)$$

where $\tilde{w}_{t,j}$ is a word sampled from the noise distribution $P_n(w)$. We use smoothed unigram frequencies (exponentiating by 0.75) as the noise distribution $P_n(w)$ (Mikolov et al., 2013b). We initialize $\ln\hat{Z} = 9$ as suggested in Chen et al. (2015), but instead of keeping it fixed we also learn $\hat{Z}$ during training (Vaswani et al., 2013). We set $k = 20$.

3 Experiments

We assess the performance of our model on two tasks: the Microsoft Research (MSR) sentence completion challenge (Zweig and Burges, 2012), and dependency parsing reranking. We also demonstrate the tree generation capability of our models. In the following, we first present details on model training and then present our results. We implemented our models using the Torch library (Collobert et al., 2011) and our code is available at `https://github.com/XingxingZhang/td-treelstm`.

3.1 Training Details

We trained our model with back propagation through time (Rumelhart et al., 1988) on an Nvidia GPU Card with a mini-batch size of 64. The objective (NLL or NCE) was minimized by stochastic gradient descent. Model parameters were uniformly initialized in $[-0.1, 0.1]$. We used the NCE objective on the MSR sentence completion task (due to the large size of this dataset) and the NLL objective on dependency parsing reranking. We used an initial learning rate of 1.0 for all experiments and when there was no significant improvement in log-likelihood on the validation set, the learning rate was divided by 2 per epoch until convergence (Mikolov et al., 2010). To alleviate the exploding gradients problem, we rescaled the gradient g when the gradient norm $||g|| > 5$ and set $g = \frac{5g}{||g||}$ (Pascanu et al., 2013; Sutskever et al., 2014). Dropout (Srivastava et al., 2014) was applied to the 2-layer TREELSTM and LDTREELSTM models. The word embedding size was set to $s = d/2$ where d is the hidden unit size.

3.2 Microsoft Sentence Completion Challenge

The task in the MSR Sentence Completion Challenge (Zweig and Burges, 2012) is to select the correct missing word for 1,040 SAT-style test sentences when presented with five candidate completions. The training set contains 522 novels from the Project Gutenberg which we preprocessed as follows. After removing headers and footers from the files, we tokenized and parsed the dataset into dependency trees with the Stanford Core NLP toolkit (Manning et al., 2014). The resulting training set contained 49M words. We converted all words to lower case and replaced those occurring five times or less with UNK. The resulting vocabulary size was 65,346 words. We randomly sampled 4,000 sentences from the training set as our validation set.

The literature describes two main approaches to the sentence completion task based on word vectors and language models. In vector-based approaches, all words in the sentence and the five candidate words are represented by a vector; the candidate which has the highest average similarity with the sentence words is selected as the answer. For language model-based methods, the LM computes the probability of a test sentence with each of the five candidate words, and picks the candidate completion which gives the highest probability. Our model belongs to this class of models.

Model	*d*	$\|\theta\|$	**Accuracy**
Word Vector based Models			
LSA	—	—	49.0
Skip-gram	640	102M	48.0
IVLBL	600	96.0M	55.5
Language Models			
KN5	—	—	40.0
UDepNgram	—	—	48.3
LDepNgram	—	—	50.0
RNN	300	48.1M	45.0
RNNME	300	1120M	49.3
depRNN+3gram	100	1014M	53.5
ldepRNN+4gram	200	1029M	50.7
LBL	300	48.0M	54.7
LSTM	300	29.9M	55.00
LSTM	400	40.2M	57.02
LSTM	450	45.3M	55.96
Bidirectional LSTM	200	33.2M	48.46
Bidirectional LSTM	300	50.1M	49.90
Bidirectional LSTM	400	67.3M	48.65
Model Combinations			
RNNMEs	—	—	55.4
Skip-gram + RNNMEs	—	—	58.9
Our Models			
TREELSTM	300	31.6M	55.29
LDTREELSTM	300	32.5M	57.79
TREELSTM	400	43.1M	56.73
LDTREELSTM	400	44.7M	**60.67**

Table 1: Model accuracy on the MSR sentence completion task. The results of KN5, RNNME and RNNMEs are reported in Mikolov (2012), LSA and RNN in Zweig et al. (2012), UDepNgram and LDepNgram in Gubbins and Vlachos (2013), depRNN+3gram and depRNN+4gram in Mirowski and Vlachos (2015), LBL in Mnih and Teh (2012), Skip-gram and Skip-gram+RNNMEs in Mikolov et al. (2013a), and IVLBL in Mnih and Kavukcuoglu (2013); *d* is the hidden size and $|\theta|$ the number of parameters in a model.

Table 1 presents a summary of our results together with previoulsy published results. The best performing word vector model is IVLBL (Mnih and Kavukcuoglu, 2013) with an accuracy of 55.5, while the best performing single language model is LBL (Mnih and Teh, 2012) with an accuracy of 54.7. Both approaches are based on the log-bilinear language model (Mnih and Hinton, 2007). A combination of several recurrent neural networks and the skip-gram model holds the state of the art with an accuracy of 58.9 (Mikolov et al., 2013b). To fairly compare with existing models, we restrict the layer

Parser	Development		Test	
	UAS	LAS	UAS	LAS
MSTParser-2nd	92.20	88.78	91.63	88.44
TREELSTM	92.51	89.07	91.79	88.53
TREELSTM*	92.64	89.09	91.97	88.69
LDTREELSTM	**92.66**	**89.14**	**91.99**	**88.69**
NN parser*	92.00	89.70	91.80	89.60
S-LSTM*	**93.20**	**90.90**	**93.10**	**90.90**

Table 2: Performance of TREELSTM and LDTREELSTM on reranking the top dependency trees produced by the 2nd order MSTParser (McDonald and Pereira, 2006). Results for the NN and S-LSTM parsers are reported in Chen and Manning (2014) and Dyer et al. (2015), respectively. * indicates that the model is initialized with pre-trained word vectors.

size of our models to 1. We observe that LDTREELSTM consistently outperforms TREELSTM, which indicates the importance of modeling the interaction between left and right dependents. In fact, LDTREELSTM ($d = 400$) achieves a new state-of-the-art on this task, despite being a single model. We also implement LSTM and bidirectional LSTM language models.[3] An LSTM with $d = 400$ outperforms its smaller counterpart ($d = 300$), however performance decreases with $d = 450$. The bidirectional LSTM is worse than the LSTM (see Mnih and Teh (2012) for a similar observation). The best performing LSTM is worse than a LDTREELSTM ($d = 300$). The input and output embeddings ($\mathbf{W}_e$ and $\mathbf{W}_{ho}$) dominate the number of parameters in all neural models except for RNNME, depRNN+3gram and ldepRNN+4gram, which include a ME model that contains 1 billion sparse n-gram features (Mikolov, 2012; Mirowski and Vlachos, 2015). The number of parameters in TREELSTM and LDTREELSTM is not much larger compared to LSTM due to the tied $\mathbf{W}_e$ and $\mathbf{W}_{ho}$ matrices.

3.3 Dependency Parsing

In this section we demonstrate that our model can be also used for parse reranking. This is not possible for sequence-based language models since they cannot estimate the probability of a tree. We use our models to rerank the top *K* dependency trees produced by the second order MSTParser (McDon-

[3]LSTMs and BiLSTMs were also trained with NCE ($s = d/2$; hyperparameters were tuned on the development set).

ald and Pereira, 2006).[4] We follow closely the experimental setup of Chen and Manning (2014) and Dyer et al. (2015). Specifically, we trained TREELSTM and LDTREELSTM on Penn Treebank sections 2–21. We used section 22 for development and section 23 for testing. We adopted the Stanford basic dependency representations (De Marneffe et al., 2006); part-of-speech tags were predicted with the Stanford Tagger (Toutanova et al., 2003). We trained TREELSTM and LDTREELSTM as language models (singletons were replaced with UNK) and did not use any POS tags, dependency labels or composition features, whereas these features are used in Chen and Manning (2014) and Dyer et al. (2015). We tuned d, the number of layers, and K on the development set.

Table 2 reports unlabeled attachment scores (UAS) and labeled attachment scores (LAS) for the MSTParser, TREELSTM (d = 300, 1 layer, K = 2), and LDTREELSTM (d = 200, 2 layers, K = 4). We also include the performance of two neural network-based dependency parsers; Chen and Manning (2014) use a neural network classifier to predict the correct transition (NN parser); Dyer et al. (2015) also implement a transition-based dependency parser using LSTMs to represent the contents of the stack and buffer in a continuous space. As can be seen, both TREELSTM and LDTREELSTM outperform the baseline MSTParser, with LDTREELSTM performing best. We also initialized the word embedding matrix $\mathbf{W}_e$ with pre-trained GLOVE vectors (Pennington et al., 2014). We obtained a slight improvement over TREELSTM (TREELSTM* in Table 2; d = 200, 2 layer, K = 4) but no improvement over LDTREELSTM. Finally, notice that LDTREELSTM is slightly better than the NN parser in terms of UAS but worse than the S-LSTM parser. In the future, we would like to extend our model so that it takes labeled dependency information into account.

3.4 Tree Generation

This section demonstrates how to use a trained LDTREELSTM to generate tree samples. The generation starts at the ROOT node. At each time step t, for each node w_t, we add a new edge and node to the tree. Unfortunately during generation, we do not know which type of edge to add. We therefore use four binary classifiers (ADD-LEFT, ADD-RIGHT, ADD-NX-LEFT and ADD-NX-RIGHT) to predict whether we should add a LEFT, RIGHT, NX-LEFT or NX-RIGHT edge.[5] Then when a classifier predicts true, we use the corresponding LSTM to generate a new node by sampling from the predicted word distribution in Equation (3). The four classifiers take the previous hidden state $\mathbf{H}[:,t']$ and the output embedding of the current node $\mathbf{W}_{ho} \cdot e(w_t)$ as features.[6] Specifically, we use a trained LDTREELSTM to go through the training corpus and generate hidden states and embeddings as input features; the corresponding class labels (true and false) are "read off" the training dependency trees. We use two-layer rectifier networks (Glorot et al., 2011) as the four classifiers with a hidden size of 300. We use the same LDTREELSTM model as in Section 3.3 to generate dependency trees. The classifiers were trained using AdaGrad (Duchi et al., 2011) with a learning rate of 0.01. The accuracies of ADD-LEFT, ADD-RIGHT, ADD-NX-LEFT and ADD-NX-RIGHT are 94.3%, 92.6%, 93.4% and 96.0%, respectively. Fig-

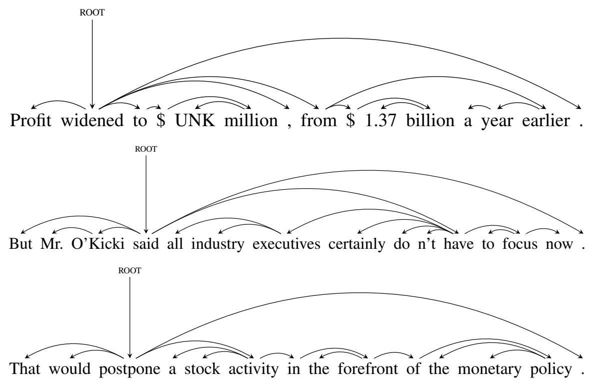
ROOT
Profit widened to $ UNK million , from $ 1.37 billion a year earlier .
ROOT
But Mr. O'Kicki said all industry executives certainly do n't have to focus now .
ROOT
That would postpone a stock activity in the forefront of the monetary policy .

Figure 5: Generated dependency trees with LDTREELSTM trained on the PTB.

[4]http://www.seas.upenn.edu/ strctlrn/MSTParser

[5]It is possible to get rid of the four classifiers by adding START/STOP symbols when generating left and right dependents as in (Eisner, 1996). We refrained from doing this for computational reasons. For a sentence with N words, this approach will lead to $2N$ additional START/STOP symbols (with one START and one STOP symbol for each word). Consequently, the computational cost and memory consumption during training will be three times as much rendering our model less scalable.

[6]The input embeddings have lower dimensions and therefore result in slightly worse classifiers.

ure 5 shows examples of generated trees.

4 Conclusions

In this paper we developed TREELSTM (and LDTREELSTM), a neural network model architecture, which is designed to predict tree structures rather than linear sequences. Experimental results on the MSR sentence completion task show that LDTREELSTM is superior to sequential LSTMs. Dependency parsing reranking experiments highlight our model's potential for dependency parsing. Finally, the ability of our model to generate dependency trees holds promise for text generation applications such as sentence compression and simplification (Filippova et al., 2015). Although our experiments have focused exclusively on dependency trees, there is nothing inherent in our formulation that disallows its application to other types of tree structure such as constituent trees or even taxonomies.

Acknowledgments

We would like to thank Adam Lopez, Frank Keller, Iain Murray, Li Dong, Brian Roark, and the NAACL reviewers for their valuable feedback. Xingxing Zhang gratefully acknowledges the financial support of the China Scholarship Council (CSC). Liang Lu is funded by the UK EPSRC Programme Grant EP/I031022/1, Natural Speech Technology (NST).

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131. Association for Computational Linguistics.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4):283–332.

Ciprian Chelba, David Engle, Frederick Jelinek, Victor Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, et al. 1997. Structure and performance of a dependency language model. In *EUROSPEECH*. Citeseer.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October. Association for Computational Linguistics.

X Chen, X Liu, MJF Gales, and PC Woodland. 2015. Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *In 40th IEEE International Conference on Accoustics, Speech and Signal Processing*, pages 5401–5405, Brisbane, Australia.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2011. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

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

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. Association for Computational Linguistics.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Ahmad Emami, Peng Xu, and Frederick Jelinek. 2003. Using a connectionist model in a syntactical based language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 372–375, Hong Kong, China.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *EMNLP*, pages 360–368.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

Joseph Gubbins and Andreas Vlachos. 2013. Dependency language models for sentence completion. In *EMNLP*, pages 1405–1410, Seattle, Washington, USA, October. Association for Computational Linguistics.

Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361.

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

Sepp Hochreiter. 1998. Vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 6(2):107–116.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Tomas Mikolov. 2012. *Statistical Language Models based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

Piotr Mirowski and Andreas Vlachos. 2015. Dependency recurrent neural language models for sentence completion. In *ACL*, pages 511–517, Beijing, China, July. Association for Computational Linguistics.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, pages 641–648.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems 26*, pages 2265–2273.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758, Edinburgh, Scotland.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1310–1318, Atlanta, Georgia, USA.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *EMNLP*, 12:1532–1543.

Jordan B. Pollack. 1990. Recursive distributed representations. *Artificial Intelligence*, 1–2(46):77–105.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3.

Rico Sennrich. 2015. Modelling and optimizing on syntactic n-grams for statistical machine translation. *Transactions of the Association for Computational Linguistics*, 3:169–182.

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL-08: HLT*, pages 577–585, Columbus, Ohio, USA.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Christopher D. Manning, and Andrew Ng. 2011a. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, pages 801–809.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011b. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161, Edinburgh, Scotland, UK.

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 VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

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 (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *The IEEE Conference on Computer Vision and Pattern Recognition*, Boston, Massachusetts, USA.

Ying Zhang. 2009. *Structured language models for statistical machine translation*. Ph.D. thesis, Johns Hopkins University.

Geoffrey Zweig and Chris J.C. Burges. 2012. A challenge set for advancing language modeling. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 29–36, Montréal, Canada.

Geoffrey Zweig, John C Platt, Christopher Meek, Christopher JC Burges, Ainur Yessenalina, and Qiang Liu. 2012. Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 601–610. Association for Computational Linguistics.