Learning to Compose Task-Specific Tree Structures

Jihun Choi and Kang Min Yoo and Sang-goo Lee
Seoul National University
Seoul, Republic of Korea
{jhchoi,kangminyoo,sglee}@europa.snu.ac.kr

Abstract

For years, recursive neural networks (RvNNs) have been shown to be suitable for representing text into fixed-length vectors and achieved good performance on several natural language processing tasks. However, the main drawback of RvNNs is that they require structured input, which makes data preparation and model implementation hard. In this paper, we propose Gumbel Tree-LSTM, a novel tree-structured long short-term memory architecture that efficiently learns how to compose task-specific tree structures only from plain text data. Our model uses Straight-Through Gumbel-Softmax estimator to decide the parent node among candidates dynamically and to calculate gradients of the discrete decision. We evaluate the proposed model on natural language inference and sentiment analysis, and show that our model outperforms or is at least comparable to previous models. We also find that our model converges significantly faster than other models.

1 Introduction

Techniques for mapping natural language into vector space have received a lot of attention, due to their capability of representing ambiguous semantics of natural language using dense vectors. Among them, methods of learning representation of words, e.g. word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014), are relatively well-studied empirically and theoretically (Baroni et al., 2014; Levy and Goldberg, 2014), and some of them became typical choices to consider when initializing word representations for better performance at downstream tasks.

However, unlike word embedding methods, research on sentence representation is still in active progress. Methods for learning sentence representation can be roughly categorized into three major architectures, where each of them is designed with different intuition and tailored for different tasks. Convolutional neural networks (CNNs) (Kim, 2014; Kalchbrenner et al., 2014) utilize local distribution of words to encode sentences, similar to n-gram models. Recurrent neural networks (RNNs) (Dai and Le, 2015; Kiros et al., 2015; Hill et al., 2016) encode sentences by reading words in sequential order. Recursive neural networks (RvNNs) (Socher et al., 2013; Irsoy and Cardie, 2014; Bowman et al., 2016), on which this paper focuses, rely on structured input (e.g. parse tree) to encode sentences, based on the intuition that there is significant semantics in the hierarchical structure of words. It is also notable that RvNNs are generalization of RNNs, as linear chain structures on which RNNs operate are equivalent to left- or right-skewed trees.

Although there is significant benefit in processing a sentence in a tree-structured recursive manner, data annotated with parse trees could be expensive to prepare and hard to be computed in batches (Bowman et al., 2016). Furthermore, the optimal hierarchical composition of words might differ depending on the properties of a task.

In this paper, we propose Gumbel Tree-LSTM, which is a novel RvNN architecture that does not require structured data and learns to compose task-specific tree structures without explicit guidance. Our Gumbel Tree-LSTM model is based on tree-structured long short-term memory (Tree-LSTM) architecture (Tai et al., 2015; Zhu et al., 2015),

1In some RvNN papers, the term ‘recursive neural network’ is often abbreviated to ‘RNN’, however to avoid confusion with recurrent neural network we decided to use the acronym ‘RvNN’.
which is one of the most renowned variants of RvNN.

To learn how to compose task-specific tree structures without depending on structured input, our model introduces composition query vector that measures validity of a composition. Using validity scores computed by the composition query vector, our model recursively selects compositions until only a single representation remains. We use Straight-Through (ST) Gumbel-Softmax estimator (Jang et al., 2017; Maddison et al., 2017) to sample compositions in the training phase. ST Gumbel-Softmax estimator relaxes the discrete sampling operation to be continuous in the backward pass, thus our model can be trained via the standard backpropagation. Also, since the computation is performed layer-wise, our model is easy to implement and naturally supports batched computation.

From experiments on natural language inference and sentiment analysis tasks, we find that our proposed model outperforms or is at least comparable to previous sentence encoder models and converges significantly faster than them.

The contributions of our work are as follows:

- We designed a novel sentence encoder architecture that learns to compose task-specific trees from plain text data, using the recently proposed Gumbel-Softmax estimator.

- We showed from experiments that the proposed architecture outperforms or is competitive to state-of-the-art models. We also observed that our model converges faster than others.

- Specifically, we saw that our model significantly outperforms previous RvNN works trained on parse trees in all conducted experiments, from which we hypothesize that constituency-based or dependency-based parse tree may not be the best structure for every task and the optimal structure could differ from task to task.

In the next section, we briefly introduce previous RvNN-based works which have similar objectives to that of our work. Then we describe the proposed model in detail and present findings from experiments. Lastly we summarize the overall content and discuss future work.

2 Related Work

There have been several RvNN-based works that share similar objectives to ours. Some models carry unsupervised learning on structures by making composition operations soft. To the best of our knowledge, gated recursive convolutional neural network (grConv) (Cho et al., 2014) is the first model of its kind. The grConv architecture uses gating mechanism to control the information flow from children to parent. Following (Cho et al., 2014), where grConv is used as an encoder for machine translation, grConv and its variants are also applied in sentence classification tasks (Chen et al., 2015; Zhao et al., 2015). Neural tree indexer (NTI) (Munkhdalai and Yu, 2017b) also utilizes soft tree structures, however it uses Tree-LSTM to learn structures, instead of grConv.

Although models that operate with soft structures are naturally capable of being trained via backpropagation, the structures predicted by them are ambiguous (i.e. a node can be built up in more than one way), and thus it is hard to interpret them. Unsupervised Tree-LSTM (Maillard et al., 2017) resolves this ambiguity while maintaining the soft property by introducing the concept of CYK parsing algorithm (Kasami, 1965; Younger, 1967; Cocke, 1970). Though their model naturally removes the ambiguity by representing a node as a weighted sum of all candidate compositions, it is memory intensive since the number of candidates linearly increases by depth. Hence it is difficult to increase the size of the model or handle long sentences, and the model might not be applicable to practical problems.

On the other hand, there exist some previous works that present a different solution to the unsupervised structure learning problem: instead of relying on the soft trees, they maintain the discreteness of tree composition processes (Socher et al., 2011; Yogatama et al., 2017).

The architecture proposed by Socher et al. (2011) greedily selects two adjacent nodes whose reconstruction error is the smallest and merges them into the parent. Their model has a similar motivation to ours in that it selects composition one by one in a bottom-up manner. However, rather than directly optimizing the tree composition process on the classification loss like ours, they train the model on the weighted sum of the reconstruction error and the classification loss. In addition, our model uses weighted sampling based
Meanwhile, there is work that introduces reinforcement learning to achieve the desired effect of discretization. REINFORCE algorithm (Williams, 1992) allows a model to use any reward function regardless of its continuity, since it computes gradients using estimated future rewards. Yogatama et al. (2017) show the REINFORCE algorithm can be used in learning task-specific tree structures. However, in the reinforcement learning setting, the computation is hard to be batched, and thus slow convergence is one of its drawbacks, according to the authors.

In the research area outside the RvNN, there also exist several works designed to learn latent hierarchical structures from data. To name but a few, hierarchical multiscale RNN (HM-RNN) (Chung et al., 2017) uses a specially designed multi-scale RNN that can learn underlying hierarchical structure of data unsupervisedly, and structured attention mechanism (Kim et al., 2017) extends attention mechanism to incorporate structure biases into a network.

## 3 Model Description

Our proposed architecture is built based on the tree-structured long short-term memory network architecture. We introduce several additional components into the Tree-LSTM architecture to allow the model to dynamically compose tree structure in a bottom-up manner and to effectively encode a sentence into a vector. In this section, we describe the components of our model in detail.

### 3.1 Tree-LSTM

Tree-structured long short-term memory network (Tree-LSTM) (Tai et al., 2015; Zhu et al., 2015) is an elegant variant of RvNN, where it controls information flow from children to parent using similar mechanism to long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). Tree-LSTM introduces cell state in computing parent representation, which assists each cell to capture distant vertical dependencies.

The following are formulae that our model uses to compute parent representation from its children:

$$
\begin{bmatrix}
i \\ f_i \\ f_r \\ o \\ g
\end{bmatrix}
=\begin{bmatrix}
i \\ f_i \\ f_r \\ o \\ g
\end{bmatrix}
\begin{bmatrix}
\sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh
\end{bmatrix}
\begin{bmatrix}
h_l \\ h_r
\end{bmatrix}
+ b_{comp}
$$

(1)

$$
c_p = f_i \odot c_l + f_r \odot c_r + i \odot g
$$

(2)

$$
h_p = o \odot \tanh(c_p),
$$

(3)

where $W_{comp} \in \mathbb{R}^{5D_h \times 2D_h}$, $b_{comp} \in \mathbb{R}^{2D_h}$, and $\odot$ is the element-wise product. Note that our formulation is akin to that of SPINN (Bowman et al., 2016), but our version does not include the tracking LSTM. Instead, our model can apply an LSTM to leaf nodes, which we will soon describe.

### 3.2 Gumbel-Softmax

Gumbel-Softmax (Jang et al., 2017) (or Concrete distribution (Maddison et al., 2017)) is a method for utilizing discrete random variables in a network. Since it approximates one-hot vectors by making them continuous, models that use Gumbel-Softmax can be trained using the standard backpropagation. Gumbel-Softmax is known to have an advantage over score-function-based gradient estimators such as REINFORCE (Williams, 1992) which suffer from high variance and slow convergence (Jang et al., 2017).

Given unnormalized probabilities $\pi_1, \ldots, \pi_k$, the continuous and differentiable approximation of arg max function, namely Gumbel-Softmax, is defined by

$$
y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)}
$$

(4)

$$
g_i = -\log(-\log(u_i))
$$

(5)

$$
u_i \sim \text{Uniform}(0, 1),
$$

(6)

where $\tau$ is a temperature parameter. As $\tau$ diminishes to zero, a sample from the Gumbel-Softmax distribution becomes cold and resembles the one-hot vector.

Straight-Through (ST) Gumbel-Softmax estimator (Jang et al., 2017), whose name reminds of Straight-Through estimator (STE) (Bengio et al., 2013), is a discrete version of the continuous Gumbel-Softmax estimator. Similar to the STE, it
maintains sparsity by taking different paths in the forward and backward propagation. Obviously ST estimators are biased, however they perform well in practice, according to several previous works (Chung et al., 2017; Gu et al., 2017) and our own result.

In the forward pass, it discretizes a continuous probability vector $y = (y_1, \cdots, y_k)$ sampled from Gumbel-Softmax distribution into the one-hot vector $y^{ST} = (y^{ST}_1, \cdots, y^{ST}_k)$, where

$$y^{ST}_i = \begin{cases} 1 & i = \arg\max_j y_j \\ 0 & \text{otherwise} \end{cases}.$$  (7)

And in the backward pass it simply uses the continuous $y$, thus the error signal is still able to backpropagate. See Figure 1 for the visualization of the forward and backward pass.

ST Gumbel-Softmax estimator is useful when a model needs to utilize discrete values directly, for example in the case that a model alters its computation path based on samples drawn from a categorical distribution.

### 3.3 Gumbel Tree-LSTM

In our Gumbel Tree-LSTM model, an input sentence composed of $N$ words is represented as a sequence of word vectors $(x_1, \cdots, x_N)$, where $x_i \in \mathbb{R}^{D_x}$. Our basic model applies an affine transformation to each $x_i$ to obtain the initial hidden and cell state:

$$r^t_i = \begin{bmatrix} h^t_i \\ c^t_i \end{bmatrix} = W_{leaf} x_i + b_{leaf},$$  (8)

which we call leaf transformation. In Eq. 8, $W_{leaf} \in \mathbb{R}^{2D_h \times D_x}$ and $b_{leaf} \in \mathbb{R}^{2D_h}$. Note that we denote the representation of $i$-th node at $t$-th layer as $r^t_i = [h^t_i; c^t_i]$.

Assume that $t$-th layer consists of $M_t$ node representations: $(r^t_1, \cdots, r^t_{M_t})$. If two adjacent nodes, say $r^t_j$ and $r^t_{j+1}$, are selected to be merged, then Eqs. 1–3 are applied by assuming $[h_i; c_i] = r^t_j$ and $[h_p; c_p] = r^t_{j+1}$ to obtain the parent representation $[h_p; c_p] = r^{t+1}_j$. Node representations which are not selected are copied to the corresponding positions at layer $t+1$. In other words, the $(t+1)$-th layer is composed of $M_{t+1} = M_t - 1$ representations $(r^{t+1}_1, \cdots, r^{t+1}_{M_{t+1}})$, where

$$r^{t+1}_j = \begin{cases} r^t_j & j < i \\ \text{Tree-LSTM} \left( r^t_j, r^t_{j+1} \right) & j = i \\ r^t_{j+1} & j > i \end{cases}.$$  (9)

This procedure is repeated until the model reaches $N$-th layer and only a single node is left.

**Parent selection** Since information about the tree structure of an input is not given to the model, a special mechanism is needed for the model to learn to compose task-specific tree structures in an end-to-end manner. We now describe the mechanism for building up the tree structure from an unstructured sentence.

First, our model introduces the trainable composition query vector $q \in \mathbb{R}^{D_h}$. The composition query vector measures how valid a representation is. Specifically, the validity score of a representation $r = [h; c]$ is defined by $q \cdot h$.

At layer $t$, the model computes candidates for the parent representations using Eqs. 1–3: $(r^{t+1}_1, \cdots, r^{t+1}_{M_{t+1}})$. Then, it calculates the validity score of each candidate and normalize it so that $\sum_{i=1}^{M_{t+1}} v_i = 1$:

$$v_i = \frac{\exp(q \cdot \hat{h}^{t+1}_i)}{\sum_{j=1}^{M_{t+1}} \exp(q \cdot \hat{h}^{t+1}_j)}.$$  (10)
In the training phase, the model samples a parent from candidates weighted on $v_1$, using the ST Gumbel-Softmax estimator described above. Since the continuous Gumbel-Softmax function is used in the backward pass, the error backpropagation signal safely passes through the sampling operation, hence the model is able to construct the task-specific tree structures that minimize the loss by backpropagation.

In the validation (or testing) phase, the model simply selects the parent which maximizes the validity score.

An example of the parent selection is depicted in Figure 2.

LSTM-based leaf transformation The basic leaf transformation using an affine transformation (Eq. 8) does not consider information about the entire sentence of an input and thus the parent selection is performed based only on local information.

SPINN (Bowman et al., 2016) addresses this issue by using the tracking LSTM which sequentially reads input words. The tracking LSTM makes the SPINN model hybrid, where the model takes advantage of both tree-structured composition and sequential reading. However, the tracking LSTM is not applicable to our model, since our model does not use shift-reduce parsing or maintain a stack.

In the tracking LSTM’s stead, our model applies an LSTM on input representations to give information about previous words to each leaf node:

$$r_i = \begin{bmatrix} h_i \\ c_i \end{bmatrix} = \text{LSTM}(x_i, h_{i-1}, c_{i-1}), \quad (11)$$

where $h_0 = c_0 = \vec{0}$.

From the experimental results, we validate that the LSTM applied to leaf nodes has a substantial gain over the basic leaf transformer.

4 Experiments

We evaluate performance of the proposed Gumbel Tree-LSTM model on two tasks: natural language inference and sentiment analysis. The implementation is made publicly available.\(^2\)

4.1 Natural Language Inference

Natural language inference (NLI) is a task of predicting the relationship between two sentences (hypothesis and premise). In the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), which we use for NLI experiments, a relationship is either contradiction, entailment, or neutral. For a model to correctly predict the relationship between two sentences, it should en-
| Model                                  | Accuracy (%) | # Params | Time (hours) |
|----------------------------------------|--------------|----------|--------------|
| 100D Gumbel Tree-LSTM (Ours)           | 82.6         | 262k     | 0.6          |
| 100D Gumbel Tree-LSTM, without Leaf LSTM (Ours) | 81.8         | 202k     | 0.7          |
| 100D Latent Syntax Tree-LSTM (Yogatama et al., 2017) | 80.5       | 500k     | 72–96†      |
| 100D CYK Tree-LSTM (Maillard et al., 2017) | 81.6         | 231k     | 240          |
| 300D Gumbel Tree-LSTM (Ours)           | 85.6         | 2.9M     | 1.6          |
| 300D Gumbel Tree-LSTM, without Leaf LSTM (Ours) | 84.4         | 2.3M     | 3.1          |
| 300D LSTM (Bowman et al., 2016)        | 80.6         | 3.0M     | 4†           |
| 300D SPINN (Bowman et al., 2016)       | 83.2         | 3.7M     | 67††         |
| 300D NSE (Munkhdalai and Yu, 2017a)    | 84.6         | 3.0M     | 26†           |
| 600D Gumbel Tree-LSTM (Ours)           | 86.0         | 10.3M    | 3.4†          |
| 600D (300+300) Gated-Attention BiLSTM (Chen et al., 2017) | 85.5         | 11.6M    | 8.5††        |
| 512–1024–2048D Shortcut-Stacked BiLSTM (Nie and Bansal, 2017) | 86.1         | 140.2M   | 3.8††        |

Table 1: Results of SNLI experiments. The above two sections group models of similar numbers of parameters. The bottom section contains results of state-of-the-art models. Word embedding parameters are not included in the number of parameters. *: values reported in the original papers. †: values estimated from per-epoch training time on the same machine our models trained on. ††: cuDNN library is used in RNN computation.

code semantics of sentences accurately, thus the task has been used as one of standard tasks for evaluating the quality of sentence representations.

The SNLI dataset is composed of about 550,000 sentences, each of which is binary-parsed. However, since our model operate on plain text, we do not use the parse tree information in both training and testing. The classifier architecture used in our SNLI experiments follows (Mou et al., 2016; Chen et al., 2017). Given the premise sentence vector ($h^{pre}$) and the hypothesis sentence vector ($h^{hyp}$) which are encoded by the proposed Gumbel Tree-LSTM model, the probability of relationship $r \in \{\text{entailment, contradiction, neutral}\}$ is computed by the following equations:

$$p(r|h^{pre}, h^{hyp}) = \text{softmax}(W_{clf}^{r}a + b_{clf}^{r})$$ (12)

$$a = \Phi(f)$$ (13)

$$f = \begin{bmatrix} h^{pre} \\ h^{hyp} \\ h^{pre} - h^{hyp} \\ h^{pre} \odot h^{hyp} \end{bmatrix},$$ (14)

where $W_{clf}^{r} \in \mathbb{R}^{1 \times D_{c}}, b_{clf}^{r} \in \mathbb{R}^{1}$, and $\Phi$ is a multi-layer perceptron (MLP) with the rectified linear unit (ReLU) activation function.

For 100D experiments (where $D_{x} = D_{h} = 100$), we use a single-hidden layer MLP with 200 hidden units (i.e. $D_{c} = 200$). The word vectors are initialized with GloVe (Pennington et al., 2014) 100D pretrained vectors\(^3\) and fine-tuned during training.

For 300D experiments (where $D_{x} = D_{h} = 300$), we set the number of hidden units of a single-hidden layer MLP to 1024 ($D_{c} = 1024$) and added batch normalization layers (Ioffe and Szegedy, 2015) followed by dropout (Srivastava et al., 2014) with probability 0.1 to the input and the output of the MLP. We also apply dropout on the word vectors with probability 0.1. Similar to 100D experiments, we initialize the word embedding matrix with GloVe 300D pretrained vectors\(^4\), however we do not update the word representations during training.

Since our model converges relatively fast, it is also possible to train a model of larger size in a reasonable time. In the 600D experiment, we set $D_{x} = 300$, $D_{h} = 600$, and an MLP with three hidden layers ($D_{c} = 1024$) is used. The dropout probability is set to 0.2 and word embeddings are not updated during training.

The size of mini-batches is set to 128 in all experiments, and hyperparameters are tuned using the validation split. The temperature parameter $\tau$ of Gumbel-Softmax is set to 1.0, and we did not find that temperature annealing improves performance. For training models, Adam optimizer (Kingma and Ba, 2015) is used.

The results of SNLI experiments are summarized in Table 1. First, we can see that LSTM-
based leaf transformation has a clear advantage over the affine-transformation-based one. It improves the performance substantially and also leads to faster convergence.

Secondly, comparing ours with other models, we find that our 100D and 300D model outperform all other models of similar numbers of parameters. Our 600D model achieves the accuracy of 86.0%, which is comparable to that of the state-of-the-art model (Nie and Bansal, 2017), while using far less parameters.

It is also worth noting that our models converge much faster than other models. All of our models converged within a few hours on a machine with NVIDIA Titan Xp GPU. We also plot validation accuracies of various models during first 5 training epochs in Figure 3, and validate that our models converge significantly faster than others, not only in terms of total training time but also in the number of iterations.\footnote{In the figure, our models and 300D NSE are trained with batch size 128. 100D CYK and 300D SPINN are trained with batch size 16 and 32 respectively, as in the original papers. We observed that our models still converge faster than others when a smaller batch size (16 or 32) is used.}

4.2 Sentiment Analysis

To evaluate the performance of our model in single-sentence classification, we conducted experiments on Stanford Sentiment Treebank (SST) (Socher et al., 2013) dataset. In the SST dataset, each sentence is represented as a binary parse tree, and each subtree of a parse tree is annotated with the corresponding sentiment score. Following the experimental setting of previous works, we use all subtrees and their labels for training, and only the root labels are used for evaluation.

The classifier has a similar architecture to SNLI experiments. Specifically, for a sentence embedding $\mathbf{h}$, the probability for the sentence to be predicted as label $s \in \{0, 1\}$ (in the binary setting, SST-2) or $s \in \{1, 2, 3, 4, 5\}$ (in the fine-grained setting, SST-5) is computed as follows:

$$p(s|h) = \text{softmax}(W^s_{clf}a + b^s_{clf})$$

where $W^s_{clf} \in \mathbb{R}^{1 \times D_c}$, $b^s_{clf} \in \mathbb{R}^1$, and $\Phi$ is a single-hidden layer MLP with the ReLU activation function. Note that subtrees labeled as neutral are ignored in the binary setting in both training and evaluation.

$$a = \Phi(h),$$

We trained our SST-2 model with hyperparameters $D_x = 300, D_h = 300, D_p = 300$. The word vectors are initialized with GloVe 300D pretrained vectors and fine-tuned during training. We apply dropout ($p = 0.5$) on the output of the word embedding layer and the input and the output of the MLP layer. The size of mini-batches is set to 32 and Adadelta (Zeiler, 2012) optimizer is used for optimization.

For our SST-5 model, hyperparameters are set to $D_x = 300, D_h = 300, D_p = 1024$. Similar to the SST-2 model, we optimize the model using Adadelta optimizer with batch size 64 and apply dropout with $p = 0.5$.

Table 2 summarizes the results of SST experiments. Our SST-2 model outperforms all other models substantially except byte-mLSTM (Radford et al., 2017), where a byte-level language model trained on the large product review dataset (McAuley et al., 2015) is used to obtain sentence representations.

We also see that the performance of our SST-5 model is on par with that of the current state-of-the-art model (McCann et al., 2017), which is pretrained on large parallel datasets and uses character n-gram embeddings alongside word embeddings, even though our model does not utilize external resources other than GloVe vectors and only uses word-level representations. The authors of (McCann et al., 2017) stated that utilizing pretraining and character n-gram embeddings improves validation accuracy by 2.8% (SST-2) or 1.7% (SST-5).

In addition, from the fact that our models substantially outperform all other RvNN-based models, we conjecture that task-specific tree structures built by our model help encode sentences into vectors more efficiently than constituency-based or dependency-based parse trees do.

4.3 Qualitative Analysis

We conduct a set of experiments to observe various properties of our trained models. First, to see how well the model encodes sentences with similar meaning or syntax into close vectors, we find nearest neighbors of a query sentence. Second, to validate that the trained composition functions are non-trivial and task-specific, we visualize the trees composed by SNLI and SST model given an identical sentence.
Table 2: Results of SST experiments. The above section contains results of RvNN-based models. Underlined score indicates the best among RvNN-based models.

| Model                                                                 | SST-2 (%) | SST-5 (%) |
|----------------------------------------------------------------------|-----------|-----------|
| Gumbel Tree-LSTM (Ours)                                              | 90.7      | 53.7      |
| RNTN (Socher et al., 2013)                                           | 85.4      | 45.7      |
| Dependency Tree-LSTM (Tai et al., 2015)                              | 85.7      | 48.4      |
| Constituency Tree-LSTM (Tai et al., 2015)                            | 88.0      | 51.0      |
| NTI-SLSTM-LSTM (Munkhdalai and Yu, 2017b)                            | 89.3      | 53.1      |
| Latent Syntax Tree-LSTM (Yogatama et al., 2017)                      | 86.5      | –         |
| Dynamic Memory Network (Kumar et al., 2016)                          | 88.6      | 52.1      |
| NSE (Munkhdalai and Yu, 2017a)                                       | 89.7      | 52.8      |
| CT-LSTM (single) (Looks et al., 2017)                                | 89.4      | 52.3      |
| CT-LSTM (ensemble) (Looks et al., 2017)                              | 90.2      | 53.6      |
| byte-nLSTM (Radford et al., 2017)                                    | 91.8      | 52.9      |
| BCN+Char+CoVe (McCann et al., 2017)                                  | 90.3      | 53.7      |

Table 3: Nearest neighbor sentences of query sentences. Each query sentence is unseen in the dataset.

| #   | sunshine is on a man ’s face . | a girl is staring at a dog . | the woman is wearing boots . |
|-----|--------------------------------|------------------------------|-----------------------------|
| 1   | a man is walking on sunshine .  | the woman is looking at a dog . | the girl is wearing shoes    |
| 2   | a guy is in a hot , sunny place | a girl takes a photo of a dog . | a person is wearing boots    |
| 3   | a man is working in the sun .  | a girl is petting her dog .   | the woman is wearing jeans   |
| 4   | it is sunny .                  | a man is taking a picture of a dog , while a woman watches . | a woman wearing sunglasses . |
| 5   | a man enjoys the sun coming through the window . | a woman is playing with her dog . | the woman is wearing a vest . |

Nearest neighbors We encode sentences in the test split of SNLI dataset using the trained 300D model and find nearest neighbors given a query sentence. Table 3 presents five nearest neighbors for each selected query sentence. In finding nearest neighbors, cosine distance is used as metric. The result shows that our model effectively maps similar sentences into vectors close to each other; the neighboring sentences are similar to a query sentence not only in terms of word overlap, but also in semantics. For example in the second column, the nearest sentence is ‘the woman is looking at a dog’, whose meaning is almost same as the query sentence. We can also see that other neighbors partially share semantics with the query sentence.

Tree examples Figure 4 and Figure 5 show that two models (300D SNLI and SST-2) generate different tree structures given an identical sentence. In Figure 4, the SNLI model groups the phrase ‘i love this’ first, while the SST model groups ‘this very much’ first. Figure 5 presents how the two models differently process a sentence containing relative pronoun ‘which’. It is intriguing that the models compose visually plausible tree structures, where the sentence is divided into two phrases by relative pronoun, even though they are trained without explicit parse trees. We hypothesize that these examples demonstrate that each model generates a distinct tree structure based on semantic properties of the task and learns non-trivial tree composition scheme.

5 Conclusion

In this paper, we propose Gumbel Tree-LSTM, a novel Tree-LSTM-based architecture that learns to compose task-specific tree structures. Our model introduces the composition query vector to compute validity of the candidate parents and selects the appropriate parent according to validity scores. In training time, the model samples the parent from candidates using ST Gumbel-Softmax estimator (Jang et al., 2017), hence it is able to be trained by standard backpropagation while maintaining its property of discretely determining the computation path in forward propagation.

From experiments, we validate that our model outperforms all other RvNN models and is competitive to state-of-the-art models, and also observed that our model converges faster than other complex models. The result poses an important question: what is the optimal input structure for RvNN? We empirically showed that the optimal structure might differ per task, and investigating task-specific latent tree structures could be an interesting future research direction.
For future work, we plan to apply the core idea beyond sentence encoding. The performance could be further improved by applying intra-sentence or inter-sentence attention mechanisms. We also plan to design an architecture that generates sentences using recursive structures.

References

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don’t count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In ACL. pages 238–247.

Joshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In EMNLP. pages 632–642.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In ACL. pages 1466–1477.

Q. Chen, X. Zhu, Z.-H. Ling, S. Wei, H. Jiang, and D. Inkpen. 2017. Recurrent neural network-based sentence encoder with gated attention for natural language inference. arXiv preprint arXiv:1708.01353.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Shiyu Wu, and Xuanjing Huang. 2015. Sentence modeling with gated recursive neural network. In EMNLP. pages 793–798.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In SSST-8. pages 103–111.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In ICLR.

John Cocke. 1970. Programming languages and their compilers: Preliminary notes. Courant Institute Mathematical Science.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In NIPS. pages 3079–3087.

Jiatao Gu, Daniel Jiwoong Im, and Victor O. K. Li. 2017. Neural machine translation with Gumbel-Greedy decoding. arXiv preprint arXiv:1706:07518.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In NAACL-HLT. pages 1367–1377.

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

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML. pages 448–456.

Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In NIPS. pages 2096–2104.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with Gumbel-Softmax. In ICLR.
