Dynamic Compositionality in Recursive Neural Networks with Structure-Aware Tag Representations

Taeuk Kim,[1] Jihun Choi,[1] Daniel Edmiston,[2] Sanghwan Bae,[1] Sang-goo Lee[1]
[1]Department of Computer Science and Engineering, Seoul National University, Seoul, Korea
[2]Department of Linguistics, University of Chicago, Chicago, IL, USA
taeuk@europa.snu.ac.kr, jhchoi@europa.snu.ac.kr, danedmiston@uchicago.edu,
sanghwan@europa.snu.ac.kr, sglee@europa.snu.ac.kr

Abstract

Most existing recursive neural network (RvNN) architectures utilize only the structure of parse trees, ignoring syntactic tags which are provided as by-products of parsing. We present a novel RvNN architecture that can provide dynamic compositionality by considering comprehensive syntactic information derived from both the structure and linguistic tags. Specifically, we introduce a structure-aware tag representation constructed by a separate tag-level tree-LSTM. With this, we can control the composition function of the existing word-level tree-LSTM by augmenting the representation as a supplementary input to the gate functions of the tree-LSTM. In extensive experiments, we show that models built upon the proposed architecture obtain superior or competitive performance on several sentence-level tasks such as sentiment analysis and natural language inference when compared against previous tree-structured models and other sophisticated neural models.

1 Introduction

One of the most fundamental topics in natural language processing is how best to derive high-level representations from constituent parts, as natural language meanings are a function of their constituent parts. How best to construct a sentence representation from distributed word embeddings is an example domain of this larger issue. Even though sequential neural models such as recurrent neural networks (RNN) (Elman 1990) and their variants including Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Unit (GRU) (Cho et al. 2014) have become the de-facto standard for condensing sentence-level information from a sequence of words into a fixed vector, there have been many lines of research towards better sentence representation using other neural architectures, e.g. convolutional neural networks (CNN) (Kim 2014) or self-attention based models (Shen et al. 2018).

From a linguistic point of view, the underlying tree structure—as expressed by its constituency and dependency trees—of a sentence is an integral part of its meaning. Inspired by this fact, some recursive neural network (RvNN[1]) models are designed to reflect the syntactic tree structure, achieving impressive results on several sentence-level tasks such as sentiment analysis (Socher et al. 2012; 2013), machine translation (Yang et al. 2017), natural language inference (Bowman et al. 2016), and discourse relation classification (Wang et al. 2017).

However, some recent works have (Yogatama et al. 2017; Choi, Yoo, and Lee 2018) proposed latent tree models, which learn to construct task-specific tree structures without explicit supervision, bringing into question the value of linguistically-motivated recursive neural models. Witnessing the surprising performance of the latent tree models on some sentence-level tasks, there arises a natural question: *Are linguistic tree structures the optimal way of composing sentence representations for NLP tasks?*

In this paper, we demonstrate that linguistic priors are in fact useful for devising effective neural models for sentence representations, showing that our novel architecture based on constituency trees and their tag[2] information obtains superior performance on several sentence-level tasks, including sentiment analysis and natural language inference.

A chief novelty of our approach is that we introduce a small separate tag-level tree-LSTM to control the composition function of the existing word-level tree-LSTM, which is in charge of extracting helpful syntactic signals for meaningful semantic composition of constituents by considering both the structures and linguistic tags of constituency trees simultaneously. In addition, we demonstrate that applying a typical LSTM to preprocess the leaf nodes of a tree-LSTM greatly improves the performance of the tree models. Moreover, we propose a clustered tag set to replace the existing tags on the assumption that the original syntactic tags are too fined-grained to be useful in neural models.

In short, our contributions in this work are as follows:

- We propose a new linguistically-motivated neural model which generates high-quality sentence representations by considering all the information extracted from constituency parse trees.
- In addition, we demonstrate the superiority of the pro-

[1]To avoid confusion, we call recursive neural networks (or tree-structured NNs) RvNNs to distinguish them from recurrent neural networks RNNs, following the convention of some previous works.

[2]In this work, we refer to both part-of-speech (POS) tags (e.g. DT-determiner, JJ-adjective) for words and phrase-level tags (e.g. NP-noun phrase, VP-verb phrase) simply as 'tags'.

posed models achieving new state-of-the-art performance within the similar model class on 4 out of 5 sentence classification benchmarks, as well as showing competitive results compared to other types of neural models.

- We empirically show that another key point to the success of tree-structured models is to contextualize input word embeddings so that the corresponding input for each word in a sentence can better reflect the meaning of the whole sentence.

2 Related Work

Recursive neural networks (RvNN) are a kind of neural architecture which model sentences by exploiting syntactic structure. While earlier RvNN models proposed utilizing diverse composition functions, including feed-forward neural networks (Socher et al. 2011), matrix-vector multiplication (Socher et al. 2012), and tensor computation (Socher et al. 2013), tree-LSTMs (Tai, Socher, and Manning 2015) remain the standard for several sentence-level tasks.

Even though classic RvNNs have demonstrated superior performance on a variety of tasks, their inflexibility, i.e. their inability to handle *dynamic compositionality* for different syntactic configurations, is a considerable weakness. For instance, it would be desirable if our model could distinguish e.g. adjective-noun composition from that of verb-noun or preposition-noun composition, as models failing to make such a distinction ignore real-world syntactic considerations such as '-arity' of function words (i.e. types), and the adjunct/argument distinction.

To enable dynamic compositionality in recursive neural networks, many previous works (Hashimoto et al. 2013; Dong et al. 2014; Qian et al. 2015; Wang et al. 2017; Liu, Qiu, and Huang 2017b; Huang, Qian, and Zhu 2017; Teng and Zhang 2017) have proposed various methods.

One main direction of research leverages tag information, which is produced as a by-product of parsing. In detail, Qian et al. (2015) suggested TG-RNN, a model employing different composition functions according to POS tags, and TE-RNN/TE-RNTN, models which leverage tag embeddings as additional inputs for the existing tree-structured models. Despite the novelty of utilizing tag information, the explosion of the number of parameters (in case of the TG-RNN) and the limited performance of the original models (in case of the TE-RNN/TE-RNTN) have prevented these models from being widely adopted. Meanwhile, Wang et al. (2017) and Huang, Qian, and Zhu (2017) proposed models based on a tree-LSTM which also uses the tag vectors to control the gate functions of the tree-LSTM. In spite of their impressive results, there is a limitation that the trained tag embeddings are too simple to reflect the rich information which tags provide in different syntactic structures. To alleviate this problem, we introduce structure-aware tag representations in the next section.

Another way of building dynamic compositionality into RvNNs is to take advantage of a meta-network (or hypernetwork). Inspired by recent works on dynamic parameter prediction, DC-TreeLSTMs (Liu, Qiu, and Huang 2017b) dynamically create the parameters for compositional functions in a tree-LSTM. Specifically, the model has two separate tree-LSTM networks whose architectures are similar, but the smaller of the two is utilized to calculate the weights of the bigger one. A possible problem for this model is that it may be easy to be trained such that the role of each tree-LSTM is ambiguous, as they share the same input, i.e. word information. Therefore, we design two disentangled tree-LSTMs in our model so that one focuses on extracting useful features from only syntactic information while the other composes semantic units with the aid of the features. Furthermore, our model reduces the complexity of computation by utilizing typical tree-LSTM frameworks instead of computing the weights for each example.

Finally, some recent works (Yogatama et al. 2017; Choi, Yoo, and Lee 2018) have proposed latent tree-structured models that learn how to formulate tree structures from only sequences of tokens, without the aid of syntactic trees or linguistic information. The latent tree models have the advantage of being able to find the optimized task-specific order of composition rather than a sequential or syntactic one. In experiments, we compare our model with not only syntactic tree-based models but also latent tree models, demonstrating that modeling with explicit linguistic knowledge can be an attractive option.

3 Model

In this section, we introduce a novel RvNN architecture, called **SATA Tree-LSTM**[3] (**S**tructure-**A**ware **T**ag **A**ugmented **Tree-LSTM**). This model is similar to typical Tree-LSTMs, but provides dynamic compositionality by augmenting a separate tag-level tree-LSTM which produces structure-aware tag representations for each node in a tree. In other words, our model has two independent tree-structured modules based on the same constituency tree, one of which (word-level tree-LSTM) is responsible for constructing sentence representations given a sequence of words as usual, while the other (tag-level tree-LSTM) provides supplementary syntactic information to the former.

In section 3.1, we first review tree-LSTM architectures. Then in section 3.2, we introduce a tag-level tree-LSTM and structure-aware tag representations. In section 3.3, we discuss an additional technique to boost the performance of tree-structured models, and in section 3.4, we describe the entire architecture of our model in detail.

3.1 Tree-LSTM

The LSTM (Hochreiter and Schmidhuber 1997) architecture was first introduced as an extension of the RNN architecture to mitigate the vanishing and exploding gradient problems. In addition, several works have discovered that applying the LSTM cell into tree structures can be an effective means of modeling sentence representations.

To be formal, the composition function of the cell in a

[3]The implementation of our model and supplemental materials are available at https://github.com/galsang/SATA-Tree-LSTM.

tree-LSTM can be formulated as follows:

$$\begin{bmatrix} \mathbf{i} \\ \mathbf{f}_l \\ \mathbf{f}_r \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W} \begin{bmatrix} \mathbf{h}_l \\ \mathbf{h}_r \end{bmatrix} + \mathbf{b} \right) \tag{1}$$

$$\mathbf{c} = \mathbf{f}_l \odot \mathbf{c}_l + \mathbf{f}_r \odot \mathbf{c}_r + \mathbf{i} \odot \mathbf{g} \tag{2}$$

$$\mathbf{h} = \mathbf{o} \odot \tanh(\mathbf{c}) \tag{3}$$

where $\mathbf{h}, \mathbf{c} \in \mathbb{R}^d$ indicate the hidden state and cell state of the LSTM cell, and $\mathbf{h}_l, \mathbf{h}_r, \mathbf{c}_l, \mathbf{c}_r \in \mathbb{R}^d$ the hidden states and cell states of a left and right child. $\mathbf{g} \in \mathbb{R}^d$ is the newly composed input for the cell and $\mathbf{i}, \mathbf{f}_l, \mathbf{f}_r, \mathbf{o} \in \mathbb{R}^d$ represent an input gate, two forget gates (left, right), and an output gate respectively. $\mathbf{W} \in \mathbb{R}^{5d \times 2d}$ and $\mathbf{b} \in \mathbb{R}^{5d}$ are trainable parameters. σ corresponds to the sigmoid function, $\tanh$ to the hyperbolic tangent, and $\odot$ to element-wise multiplication.

Note the equations assume that there are only two children for each node, i.e. binary or binarized trees, following the standard in the literature. While RvNN models can be constructed on any tree structure, in this work we only consider constituency trees as inputs.

In spite of the obvious upside that recursive models have in being so flexible, they are known for being difficult to fully utilize with batch computations as compared to other neural architectures because of the diversity of structure found across sentences. To alleviate this problem, Bowman et al. (2016) proposed the SPINN model, which brings a shift-reduce algorithm to the tree-LSTM. As SPINN simplifies the process of constructing a tree into only two operations, i.e. shift and reduce, it can support more effective parallel computations while enjoying the advantages of tree structures. For efficiency, our model also starts from our own SPINN re-implementation, whose function is exactly the same as that of the tree-LSTM.

3.2 Structure-aware Tag Representation

In most previous works using linguistic tag information (Qian et al. 2015; Wang et al. 2017; Huang, Qian, and Zhu 2017), tags are usually represented as simple low-dimensional dense vectors, similar to word embeddings. This approach seems reasonable in the case of POS tags that are attached to the corresponding words, but phrase-level constituent tags (e.g. NP, VP, ADJP) vary greatly in size and shape, making them less amenable to uniform treatment. For instance, even the same phrase tags within different syntactic contexts can vary greatly in size and internal structure, as the case of NP tags in Figure 1 shows. Here, the NP consisting of DT[the]-NN[stories] has a different internal structure than the NP consisting of NP[the film 's]-NNS[shortcomings].

One way of deriving *structure-aware* tag representations from the original tag embeddings is to introduce a separate tag-level tree-LSTM which accepts the typical tag embeddings at each node of a tree and outputs the computed structure-aware tag representations for the nodes. Note that the module concentrates on extracting useful syntactic features by considering only the tags and structures of the trees, excluding word information.

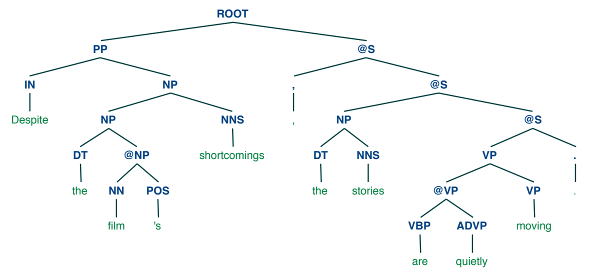

Figure 1: A constituency tree example from Stanford Sentiment Treebank.

Formally, we denote a tag embedding for the tag attached to each node in a tree as $\mathbf{e} \in \mathbb{R}^{d_\mathrm{T}}$. Then, the function of each cell in the tag tree-LSTM is defined in the following way. Leaf nodes are defined by the following:

$$\begin{bmatrix} \hat{\mathbf{c}} \\ \hat{\mathbf{h}} \end{bmatrix} = \tanh(\mathbf{U}_\mathrm{T} \mathbf{e} + \mathbf{a}_\mathrm{T}) \tag{4}$$

while non-leaf nodes are defined by the following:

$$\begin{bmatrix} \hat{\mathbf{i}} \\ \hat{\mathbf{f}}_l \\ \hat{\mathbf{f}}_r \\ \hat{\mathbf{o}} \\ \hat{\mathbf{g}} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W}_\mathrm{T} \begin{bmatrix} \hat{\mathbf{h}}_l \\ \hat{\mathbf{h}}_r \\ \mathbf{e} \end{bmatrix} + \mathbf{b}_\mathrm{T} \right) \tag{5}$$

$$\hat{\mathbf{c}} = \hat{\mathbf{f}}_l \odot \hat{\mathbf{c}}_l + \hat{\mathbf{f}}_r \odot \hat{\mathbf{c}}_r + \hat{\mathbf{i}} \odot \hat{\mathbf{g}} \tag{6}$$

$$\hat{\mathbf{h}} = \hat{\mathbf{o}} \odot \tanh(\hat{\mathbf{c}}) \tag{7}$$

where $\hat{\mathbf{h}}, \hat{\mathbf{c}} \in \mathbb{R}^{d_\mathrm{T}}$ represent the hidden state and cell state of each node in the tag tree-LSTM. We regard the hidden state ($\hat{\mathbf{h}}$) as a structure-aware tag representation for the node. $\mathbf{U}_\mathrm{T} \in \mathbb{R}^{2d_\mathrm{T} \times d_\mathrm{T}}, \mathbf{a}_\mathrm{T} \in \mathbb{R}^{2d_\mathrm{T}}, \mathbf{W}_\mathrm{T} \in \mathbb{R}^{5d_\mathrm{T} \times 3d_\mathrm{T}}$, and $\mathbf{b}_\mathrm{T} \in \mathbb{R}^{5d_\mathrm{T}}$ are trainable parameters. The rest of the notation follows equations 1, 2, and 3. In case of leaf nodes, the states are computed by a simple non-linear transformation. Meanwhile, the composition function in a non-leaf node absorbs the tag embedding ($\mathbf{e}$) as an additional input as well as the hidden states of the two children nodes. The benefit of revising tag representations according to the internal structure is that the derived embedding is a function of the corresponding makeup of the node, rather than a monolithic, categorical tag.

With regard to the tags themselves, we conjecture that the taxonomy of the tags currently in use in many NLP systems is too complex to be utilized effectively in deep neural models, considering the specificity of many tag sets and the limited amount of data with which to train. Thus, we cluster POS (word-level) tags into 12 groups following the universal POS tagset (Petrov, Das, and McDonald 2012) and phrase-level tags into 11 groups according to criteria analogous to the case of words, resulting in 23 tag categories in total. In this work, we use the revised coarse-grained tags instead of the original ones. For more details, we refer readers to the supplemental materials.

3.3 Leaf-LSTM

An inherent shortcoming of RvNNs relative to sequential models is that each intermediate representation in a tree is unaware of its external context until all the information is gathered together at the root node. In other words, each composition process is prone to be locally optimized rather than globally optimized.

To mitigate this problem, we propose using a leaf-LSTM following the convention of some previous works (Eriguchi, Hashimoto, and Tsuruoka 2016; Yang et al. 2017; Choi, Yoo, and Lee 2018), which is a typical LSTM that accepts a sequence of words in order. Instead of leveraging word embeddings directly, we can use each hidden state and cell state of the leaf-LSTM as input tokens for leaf nodes in a tree-LSTM, anticipating the proper contextualization of the input sequence.

Formally, we denote a sequence of words in an input sentence as $w_{1:n}$ (n: the length of the sentence), and the corresponding word embeddings as $\mathbf{x}_{1:n}$. Then, the operation of the leaf-LSTM at time t can be formulated as,

$$\begin{bmatrix} \tilde{\mathbf{i}} \\ \tilde{\mathbf{f}} \\ \tilde{\mathbf{o}} \\ \tilde{\mathbf{g}} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W}_{\mathrm{L}} \begin{bmatrix} \tilde{\mathbf{h}}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\mathrm{L}} \right) \tag{8}$$

$$\tilde{\mathbf{c}}_t = \tilde{\mathbf{f}} \odot \tilde{\mathbf{c}}_{t-1} + \tilde{\mathbf{i}} \odot \tilde{\mathbf{g}} \tag{9}$$

$$\tilde{\mathbf{h}}_t = \tilde{\mathbf{o}} \odot \tanh\left(\tilde{\mathbf{c}}_t\right) \tag{10}$$

where $\mathbf{x}_t \in \mathbb{R}^{d_w}$ indicates an input word vector and $\tilde{\mathbf{h}}_t, \tilde{\mathbf{c}}_t \in \mathbb{R}^{d_h}$ represent the hidden and cell state of the LSTM at time t ($\tilde{\mathbf{h}}_{t-1}$ corresponds to the hidden state at time t-1). $\mathbf{W}_{\mathrm{L}}$ and $\mathbf{b}_{\mathrm{L}}$ are learnable parameters. The remaining notation follows that of the tree-LSTM above.

In experiments, we demonstrate that introducing a leaf-LSTM fares better at processing the input words of a tree-LSTM compared to using a feed-forward neural network. We also explore the possibility of its bidirectional setting in ablation study.

3.4 SATA Tree-LSTM

In this section, we define **SATA Tree-LSTM** (**S**tructure-**A**ware **T**ag **A**ugmented **Tree-LSTM**, see Figure 2) which joins a tag-level tree-LSTM (section 3.2), a leaf-LSTM (section 3.3), and the original word tree-LSTM together.

As above we denote a sequence of words in an input sentence as $w_{1:n}$ and the corresponding word embeddings as $\mathbf{x}_{1:n}$. In addition, a tag embedding for the tag attached to each node in a tree is denoted by $\mathbf{e} \in \mathbb{R}^{d_{\mathrm{T}}}$. Then, we derive the final sentence representation for the input sentence with our model in two steps.

First, we compute structure-aware tag representations ($\hat{\mathbf{h}}$) for each node of a tree using the tag tree-LSTM (the right side of Figure 2) as follows:

$$\begin{bmatrix} \hat{\mathbf{c}} \\ \hat{\mathbf{h}} \end{bmatrix} = \begin{cases} \text{Tag-Tree-LSTM}(\mathbf{e}) & \text{if a leaf node} \\ \text{Tag-Tree-LSTM}(\hat{\mathbf{h}}_l, \hat{\mathbf{h}}_r, \mathbf{e}) & \text{otherwise} \end{cases} \tag{11}$$

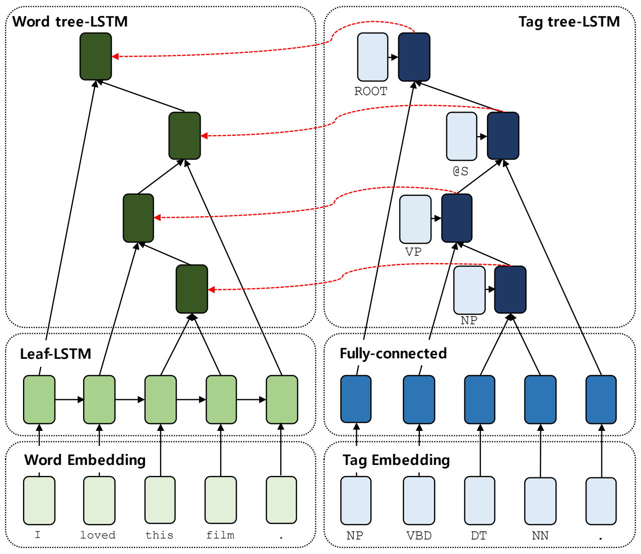

Figure 2: A diagram of SATA Tree-LSTM. The model has two separate tree-LSTM modules, the right of which (tag tree-LSTM) extracts a structure-aware tag representation to control the composition function of the remaining tree-LSTM (word tree-LSTM). Fully-connected: one-layered non-linear transformation.

where Tag-Tree-LSTM indicates the module we described in section 3.2.

Second, we combine semantic units recursively on the word tree-LSTM in a bottom-up fashion. For leaf nodes, we leverage the Leaf-LSTM (the bottom-left of Figure 2, explained in section 3.3) to compute $\tilde{\mathbf{c}}_t$ and $\tilde{\mathbf{h}}_t$ in sequential order, with the corresponding input $\mathbf{x}_t$.

$$\begin{bmatrix} \tilde{\mathbf{c}}_t \\ \tilde{\mathbf{h}}_t \end{bmatrix} = \text{Leaf-LSTM}(\tilde{\mathbf{h}}_{t-1}, \mathbf{x}_t) \tag{12}$$

Then, the $\tilde{\mathbf{c}}_t$ and $\tilde{\mathbf{h}}_t$ can be utilized as input tokens to the word tree-LSTM, with the left (right) child of the target node corresponding to the tth word in the input sentence.

$$\begin{bmatrix} \check{\mathbf{c}}_{\{l,r\}} \\ \check{\mathbf{h}}_{\{l,r\}} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{c}}_t \\ \tilde{\mathbf{h}}_t \end{bmatrix} \tag{13}$$

In the non-leaf node case, we calculate phrase representations for each node in the word tree-LSTM (the upper-left of Figure 2) recursively as follows:

$$\check{\mathbf{g}} = \tanh\left(\mathbf{U}_{\mathrm{w}} \begin{bmatrix} \check{\mathbf{h}}_l \\ \check{\mathbf{h}}_r \end{bmatrix} + \mathbf{a}_{\mathrm{w}} \right) \tag{14}$$

$$\begin{bmatrix} \check{\mathbf{i}} \\ \check{\mathbf{f}}_l \\ \check{\mathbf{f}}_r \\ \check{\mathbf{o}} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \end{bmatrix} \left(\mathbf{W}_{\mathrm{w}} \begin{bmatrix} \check{\mathbf{h}}_l \\ \check{\mathbf{h}}_r \\ \hat{\mathbf{h}} \end{bmatrix} + \mathbf{b}_{\mathrm{w}} \right) \tag{15}$$

$$\check{\mathbf{c}} = \check{\mathbf{f}}_l \odot \check{\mathbf{c}}_l + \check{\mathbf{f}}_r \odot \check{\mathbf{c}}_r + \check{\mathbf{i}} \odot \check{\mathbf{g}} \tag{16}$$

$$\check{\mathbf{h}} = \check{\mathbf{o}} \odot \tanh\left(\check{\mathbf{c}}\right) \tag{17}$$

where $\check{\mathbf{h}}$, $\check{\mathbf{c}} \in \mathbb{R}^{d_h}$ represent the hidden and cell state of each node in the word tree-LSTM. $\mathbf{U}_{\mathrm{w}} \in \mathbb{R}^{d_h \times 2d_h}$, $\mathbf{W}_{\mathrm{w}} \in \mathbb{R}^{4d_h \times (2d_h + d_{\mathrm{T}})}$, $\mathbf{a}_{\mathrm{w}} \in \mathbb{R}^{d_h}$, $\mathbf{b}_{\mathrm{w}} \in \mathbb{R}^{4d_h}$ are learned parameters. The remaining notation follows those of the previous sections. Note that the structure-aware tag representations ($\hat{\mathbf{h}}$) are only utilized to control the gate functions of the word tree-LSTM in the form of additional inputs, and are not involved in the semantic composition ($\check{\mathbf{g}}$) directly.

Finally, the hidden state of the root node ($\check{\mathbf{h}}_{\mathrm{root}}$) in the word-level tree-LSTM becomes the final sentence representation of the input sentence.

4 Experiment and Discussion

4.1 Quantitative Analysis

Sentence classification tasks One of the most basic approaches to evaluate a sentence encoder is to measure the classification performance with the sentence representations made by the encoder. Thus, we conduct experiments on the following five datasets. (Summary statistics for the datasets are reported in the supplemental materials.)

- **MR**: A group of movie reviews with binary (positive / negative) classes. (Pang and Lee 2005)
- **SST-2**: Stanford Sentiment Treebank (Socher et al. 2013). Similar to MR, but each review is provided in the form of a binary parse tree whose nodes are annotated with numeric sentiment values. For SST-2, we only consider binary (positive / negative) classes.
- **SST-5**: Identical to SST-2, but the reviews are grouped into fine-grained (very negative, negative, neutral, positive, very positive) classes.
- **SUBJ**: Sentences grouped as being either subjective or objective (binary classes). (Pang and Lee 2004)
- **TREC**: A dataset which groups questions into six different question types (classes). (Li and Roth 2002)

As a preprocessing step, we construct parse trees for the sentences in the datasets using the Stanford PCFG parser (Klein and Manning 2003). Because syntactic tags are by-products of constituency parsing, we do not need further preprocessing.

To classify the sentence given our sentence representation ($\check{\mathbf{h}}_{\mathrm{root}}$), we use one fully-connected layer with a ReLU activation, followed by a softmax classifier. The final predicted probability distribution of the class y given the sentence $w_{1:n}$ is defined as follows,

$$\mathbf{s} = \mathrm{ReLU}(\mathbf{W}_{\mathrm{s}}\check{\mathbf{h}}_{\mathrm{root}} + \mathbf{b}_{\mathrm{s}}) \tag{18}$$

$$p(y|w_{1:n}) = \mathrm{softmax}(\mathbf{W}_{\mathrm{c}}\mathbf{s} + \mathbf{b}_{\mathrm{c}}) \tag{19}$$

where $\mathbf{s} \in \mathbb{R}^{d_{\mathrm{s}}}$ is the computed task-specific sentence representation for the classifier, and $\mathbf{W}_{\mathrm{s}} \in \mathbb{R}^{d_{\mathrm{s}} \times d_h}$, $\mathbf{W}_{\mathrm{c}} \in \mathbb{R}^{d_{\mathrm{c}} \times d_s}$, $\mathbf{b}_{\mathrm{s}} \in \mathbb{R}^{d_s}$, $\mathbf{b}_{\mathrm{c}} \in \mathbb{R}^{d_c}$ are trainable parameters. As an objective function, we use the cross entropy of the predicted and true class distributions.

The results of the experiments on the five datasets are shown in table 1. In this table, we report the test accuracy of our model and various other models on each dataset in terms of percentage. To consider the effects of random initialization, we report the best numbers obtained from each several runs with hyper-parameters fixed.

Compared with the previous syntactic tree-based models as well as other neural models, our SATA Tree-LSTM shows superior or competitive performance on all tasks. Specifically, our model achieves new state-of-the-art results within the tree-structured model class on 4 out of 5 sentence classification tasks—SST-2, SST-5, MR, and TREC. The model shows its strength, in particular, when the datasets provide phrase-level supervision to facilitate tree structure learning (i.e. SST-2, SST-5). Moreover, the numbers we report for SST-5 and TREC are competitive to the existing state-of-the-art results including ones from structurally pre-trained models such as ELMo (Peters et al. 2018), proving our model's superiority. Note that the SATA Tree-LSTM also outperforms the recent latent tree-based model, indicating that modeling a neural model with explicit linguistic knowledge can be an attractive option.

On the other hand, a remaining concern is that our SATA Tree-LSTM is not robust to random seeds when the size of a dataset is relatively small, as tag embeddings are randomly initialized rather than relying on pre-trained ones in contrast with the case of words. From this observation, we could find out there needs a direction of research towards pre-trained tag embeddings.

Natural language inference To estimate the performance of our model beyond the tasks requiring only one sentence at a time, we conduct an experiment on the Stanford Natural Language Inference (Bowman et al. 2015) dataset, each example of which consists of two sentences, the premise and the hypothesis. Our objective given the data is to predict the correct relationship between the two sentences among three options— contradiction, neutral, or entailment.

We use the siamese architecture to encode both the premise ($p_{1:m}$) and hypothesis ($h_{1:n}$) following the standard of sentence-encoding models in the literature. (Specifically, $p_{1:m}$ is encoded as $\check{\mathbf{h}}^p_{\mathrm{root}} \in \mathbb{R}^{d_h}$ and $h_{1:n}$ is encoded as $\check{\mathbf{h}}^h_{\mathrm{root}} \in \mathbb{R}^{d_h}$ with the same encoder.) Then, we leverage some heuristics (Mou et al. 2016), followed by one fully-connected layer with a ReLU activation and a softmax classifier. Specifically,

$$\mathbf{z} = \left[\check{\mathbf{h}}^p_{\mathrm{root}}; \check{\mathbf{h}}^h_{\mathrm{root}}; |\check{\mathbf{h}}^p_{\mathrm{root}} - \check{\mathbf{h}}^h_{\mathrm{root}}|; \check{\mathbf{h}}^p_{\mathrm{root}} \odot \check{\mathbf{h}}^h_{\mathrm{root}}\right] \tag{20}$$

$$\mathbf{s} = \mathrm{ReLU}(\mathbf{W}_{\mathrm{s}}\mathbf{z} + \mathbf{b}_{\mathrm{s}}) \tag{21}$$

$$p(y|p_{1:m}, h_{1:n}) = \mathrm{softmax}(\mathbf{W}_{\mathrm{c}}\mathbf{s} + \mathbf{b}_{\mathrm{c}}) \tag{22}$$

where $\mathbf{z} \in \mathbb{R}^{4d_h}$, $\mathbf{s} \in \mathbb{R}^{d_s}$ are intermediate features for the classifier and $\mathbf{W}_{\mathrm{s}} \in \mathbb{R}^{d_{\mathrm{s}} \times 4d_h}$, $\mathbf{W}_{\mathrm{c}} \in \mathbb{R}^{d_{\mathrm{c}} \times d_s}$, $\mathbf{b}_{\mathrm{s}} \in \mathbb{R}^{d_s}$, $\mathbf{b}_{\mathrm{c}} \in \mathbb{R}^{d_c}$ are again trainable parameters.

Our experimental results on the SNLI dataset are shown in table 2. In this table, we report the test accuracy and number of trainable parameters for each model. Our SATA-LSTM again demonstrates its decent performance compared against the neural models built on both syntactic trees and latent trees, as well as the non-tree models. (Latent Syntax Tree-LSTM: Yogatama et al. (2017), Tree-based CNN: Mou et al. (2016), Gumbel Tree-LSTM: Choi, Yoo, and Lee

Models	SST-2	SST-5	MR	SUBJ	TREC
Tree-structured models					
RNTN (Socher et al. 2013)	85.4	45.7	-	-	-
AdaMC-RNTN (Dong et al. 2014)	88.5	46.7	-	-	-
TE-RNTN (Qian et al. 2015)	87.7	49.8	-	-	-
TBCNN (Mou et al. 2015)	87.9	51.4	-	-	96.0
Tree-LSTM (Tai, Socher, and Manning 2015)	88.0	51.0	-	-	-
AdaHT-LSTM-CM (Liu, Qiu, and Huang 2017a)	87.8	50.2	81.9	94.1	-
DC-TreeLSTM (Liu, Qiu, and Huang 2017b)	87.8	-	81.7	93.7	93.8
TE-LSTM (Huang, Qian, and Zhu 2017)	89.6	52.6	82.2	-	-
BiConTree (Teng and Zhang 2017)	90.3	53.5	-	-	94.8
Gumbel Tree-LSTM* (Choi, Yoo, and Lee 2018)	90.7	53.7	-	-	-
TreeNet (Cheng et al. 2018)	-	-	83.6	$\underline{95.9}$	96.1
SATA Tree-LSTM (Ours)	**91.3**	**$\underline{54.4}$**	**83.8**	**95.4**	**$\underline{96.2}$**
Other neural models					
CNN (Kim 2014)	88.1	48.0	81.5	93.4	93.6
AdaSent (Zhao, Lu, and Poupart 2015)	-	-	83.1	95.5	92.4
LSTM-CNN (Zhou et al. 2016)	89.5	52.4	82.3	94.0	96.1
byte-mLSTM† (Radford, Jozefowicz, and Sutskever 2017)	$\underline{91.8}$	52.9	$\underline{86.9}$	94.6	-
BCN + Char + CoVe† (McCann et al. 2017)	90.3	53.7	-	-	95.8
BCN + Char + ELMo† (Peters et al. 2018)	-	$\underline{54.7 \pm 0.5}$	-	-	-

Table 1: The comparison of various models on different sentence classification tasks. We report the test accuracy of each model in percentage. Our SATA Tree-LSTM shows superior or competitive performance on all tasks, compared to previous tree-structured models as well as other sophisticated models. $\star$: Latent tree-structured models. $\dagger$: Models which are pre-trained with large external corpora.

(2018), NSE: Munkhdalai and Yu (2017), Reinforced Self-Attention Network: Shen et al. (2018), Residual stacked encoders: Nie and Bansal (2017), BiLSTM with generalized pooling: Chen, Ling, and Zhu (2018).) Note that the number of learned parameters in our model is also comparable to other sophisticated models, showing the efficiency of our model.

Even though our model has proven its mettle, the effect of tag information seems relatively weak in the case of SNLI, which contains a large amount of data compared to the others. One possible explanation is that neural models may learn some syntactic rules from large amounts of text when the text size is large enough, reducing the necessity of external linguistic knowledge. We leave the exploration of the effectiveness of tags relative to data size for future work.

Experimental details Here we go over the settings common across our models during experimentation. For more task-specific details, refer to the supplemental materials.

For our input embeddings, we used 300 dimensional 840B GloVe (Pennington, Socher, and Manning 2014) as pre-trained word embeddings, and tag representations were randomly sampled from the uniform distribution [-0.005, 0.005]. Tag vectors are revised during training while the fine-tuning of the word embedding depends on the task. Our models were trained using the Adam (Kingma and Ba 2014) or Adadelta (Zeiler 2012) optimizer, depending on task. For regularization, weight decay is added to the loss function except for SNLI following Loshchilov and Hutter (2017) and Dropout (Srivastava et al. 2014) is also applied for the word embeddings and task-specific classifiers. Moreover, batch normalization (Ioffe and Szegedy 2015) is adopted for the classifiers. As a default, all the weights in the model are initialized following He et al. (2015) and the biases are set to 0. The total norm of the gradients of the parameters is clipped

Models	Acc.	# Params
Tree-structured models		
100D Latent Syntax Tree-LSTM*	80.5	500K
300D Tree-based CNN	82.1	3.5M
300D SPINN-PI	83.2	3.7M
300D Gumbel Tree-LSTM*	85.6	2.9M
300D SATA Tree-LSTM (Ours)	**85.9**	**3.3M**
Other neural models		
300D NSE	84.6	3.0M
300D Reinforced Self-Attention Network	86.3	3.1M
600D Residual stacked encoders	86.0	29M
600D BiLSTM with generalized pooling	$\underline{86.6}$	65M

Table 2: The accuracy of diverse models on Stanford Natural Language Inference. For fair comparison, we only consider sentence-encoding based models. Our model achieves a comparable result with a moderate number of parameters. $\star$: Latent tree models.

not to be over 5 during training.

Our best models for each dataset were chosen by validation accuracy in cases where a validation set was provided as a part of the dataset. Otherwise, we perform a grid search on probable hyper-parameter settings, or run 10-fold cross-validation in cases where even a test set does not exist.

4.2 Ablation Study

In this section, we design an ablation study on the core modules of our model to explore their effectiveness. The dataset used in this experiment is SST-2. To conduct the experiment, we only replace the target module with other candidates while maintaining the other settings. To be specific, we focus on two modules, the leaf-LSTM and structure-aware tag embeddings (tag-level tree-LSTM). In the first case, the leaf-LSTM is replaced with a fully-connected layer with a tanh activation or Bi-LSTM. In the second case, we replace

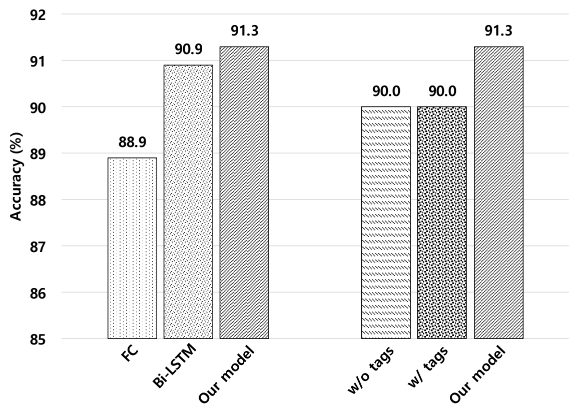

Figure 3: An ablation study on the core modules of our model. The test accuracy of each model on SST-2 is reported. The results demonstrate that the modules play an important role for achieving the superior performance of our model. FC: A fully connected-layer with a tanh function. w/o tags: Tag embeddings are not used. w/ tags: The naive tag embeddings are directly inserted into each node of a tree.

the structure-aware tag embeddings with naive tag embeddings or do not employ them at all.

The experimental results are depicted in Figure 3. As the chart shows, our model outperforms all the other options we have considered. In detail, the left part of the chart shows that the leaf-LSTM is the most effective option compared to its competitors. Note that the sequential leaf-LSTM is somewhat superior or competitive than the bidirectional leaf-LSTM when both have a comparable number of parameters. We conjecture this may because a backward LSTM does not add additional useful knowledge when the structure of a sentence is already known. In conclusion, we use the uni-directional LSTM as a leaf module because of its simplicity and remarkable performance.

Meanwhile, the right part of the figure demonstrates that our newly introduced structure-aware embeddings have a real impact on improving the model performance. Interestingly, employing the naive tag embeddings made no difference in terms of the test accuracy, even though the absolute validation accuracy increased (not reported in the figure). This result supports our assumption that tag information should be considered in the structure.

4.3 Qualitative Analysis

In previous sections, we have numerically demonstrated that our model is effective in encouraging useful composition of semantic units. Here, we directly investigate the computed representations for each node of a tree, showing that the remarkable performance of our model is mainly due to the gradual and recursive composition of the intermediate representations on the syntactic structure.

To observe the phrase-level embeddings at a glance, we draw a scatter plot in which a point represents the corresponding intermediate representation. We utilize PCA (Principal Component Analysis) to project the representations into a two-dimensional vector space. As a target parse tree, we reuse the one seen in Figure 1. The result is shown in Figure 4.

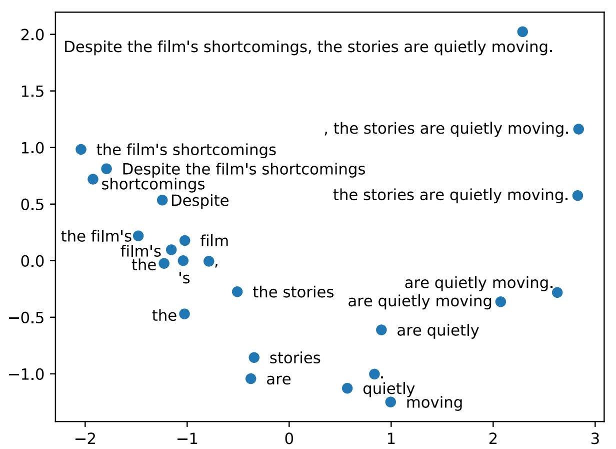

Figure 4: A scatter plot whose points represent the intermediate representations for each node of the tree in Figure 1. From this figure, we can see the tendency of constructing the representations recursively from the low to the high level.

From this figure, we confirm that the intermediate representations have a hierarchy in the semantic space, which is very similar to that of the parse tree. In other words, as many tree-structured models pursue, we can see the tendency of constructing the representations from the low-level (the bottom of the figure) to the high-level (the top-left and top-right of the figure), integrating the meaning of the constituents recursively. An interesting thing to note is that the final sentence representation is near that of the phrase *', the stories are quietly moving.'* rather than that of *'Despite the film's shortcomings'*, catching the main meaning of the sentence.

5 Conclusion

We have proposed a novel RvNN architecture to fully utilize linguistic priors. A newly introduced tag-level tree-LSTM demonstrates that it can effectively control the composition function of the corresponding word-level tree-LSTM. In addition, the proper contextualization of the input word vectors results in significant performance improvements on several sentence-level tasks. For future work, we plan to explore a new way of exploiting dependency trees effectively, similar to the case of constituency trees.

Acknowledgments

We thank anonymous reviewers for their constructive and fruitful comments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF2016M3C4A7952587).

References

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*.

Bowman, S. R.; Gauthier, J.; Rastogi, A.; Gupta, R.; Manning, C. D.; and Potts, C. 2016. A fast unified model for parsing and sentence understanding. In *ACL*, 1466–1477.

Chen, Q.; Ling, Z.-H.; and Zhu, X. 2018. Enhancing sentence embedding with generalized pooling. In *COLING*, 1815–1826.

Cheng, Z.; Yuan, C.; Li, J.; and Yang, H. 2018. Treenet: Learning sentence representations with unconstrained tree structure. In *IJCAI*.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 1724–1734.

Choi, J.; Yoo, K. M.; and Lee, S.-g. 2018. Learning to compose task-specific tree structures. In *AAAI*.

Dong, L.; Wei, F.; Zhou, M.; and Xu, K. 2014. Adaptive multi-compositionality for recursive neural models with applications to sentiment analysis. In *AAAI*, 1537–1543.

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

Eriguchi, A.; Hashimoto, K.; and Tsuruoka, Y. 2016. Tree-to-sequence attentional neural machine translation. In *ACL*, 823–833.

Hashimoto, K.; Miwa, M.; Tsuruoka, Y.; and Chikayama, T. 2013. Simple customization of recursive neural networks for semantic relation classification. In *EMNLP*, 1372–1376.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 1026–1034.

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

Huang, M.; Qian, Q.; and Zhu, X. 2017. Encoding syntactic knowledge in neural networks for sentiment classification. *ACM Transactions on Information Systems (TOIS)* 35(3):26.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*, 448–456.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *EMNLP*, 1746–1751.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klein, D., and Manning, C. D. 2003. Accurate unlexicalized parsing. In *ACL*, 423–430.

Li, X., and Roth, D. 2002. Learning question classifiers. In *COLING*, 1–7.

Liu, P.; Qiu, X.; and Huang, X. 2017a. Adaptive semantic compositionality for sentence modelling. In *IJCAI*, 4061–4067.

Liu, P.; Qiu, X.; and Huang, X. 2017b. Dynamic compositional neural networks over tree structure. In *IJCAI*, 4054–4060.

Loshchilov, I., and Hutter, F. 2017. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*.

McCann, B.; Bradbury, J.; Xiong, C.; and Socher, R. 2017. Learned in translation: Contextualized word vectors. In *NIPS*, 6294–6305.

Mou, L.; Peng, H.; Li, G.; Xu, Y.; Zhang, L.; and Jin, Z. 2015. Discriminative neural sentence modeling by tree-based convolution. In *EMNLP*, 2315–2325.

Mou, L.; Men, R.; Li, G.; Xu, Y.; Zhang, L.; Yan, R.; and Jin, Z. 2016. Natural language inference by tree-based convolution and heuristic matching. In *ACL*, 130.

Munkhdalai, T., and Yu, H. 2017. Neural semantic encoders. In *ACL*, 397–407.

Nie, Y., and Bansal, M. 2017. Shortcut-stacked sentence encoders for multi-domain inference. In *RepEval*, 41–45.

Pang, B., and Lee, L. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL*, 271.

Pang, B., and Lee, L. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, 115–124.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *EMNLP*, 1532–1543.

Peters, M.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *NAACL*, 2227–2237.

Petrov, S.; Das, D.; and McDonald, R. 2012. A universal part-of-speech tagset. In *LREC*.

Qian, Q.; Tian, B.; Huang, M.; Liu, Y.; Zhu, X.; and Zhu, X. 2015. Learning tag embeddings and tag-specific composition functions in recursive neural network. In *ACL*, volume 1, 1365–1374.

Radford, A.; Jozefowicz, R.; and Sutskever, I. 2017. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Shen, T.; Tianyi, Z.; Guodong, L.; Jing, J.; Sen, W.; and Chengqi, Z. 2018. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. *arXiv preprint arXiv:1801.10296*.

Socher, R.; Lin, C. C.-Y.; Ng, A. Y.; and Manning, C. D. 2011. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 129–136.

Socher, R.; Huval, B.; Manning, C. D.; and Ng, A. Y. 2012. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP*, 1201–1211.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 1631–1642.

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

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, 1556–1566.

Teng, Z., and Zhang, Y. 2017. Head-lexicalized bidirectional tree lstms. *TACL* 5:163–177.

Wang, Y.; Li, S.; Yang, J.; Sun, X.; and Wang, H. 2017. Tag-enhanced tree-structured neural networks for implicit discourse relation classification. In *IJCNLP*, 496–505.

Yang, B.; Wong, D. F.; Xiao, T.; Chao, L. S.; and Zhu, J. 2017. Towards bidirectional hierarchical representations for attention-based neural machine translation. In *EMNLP*, 1432–1441.

Yogatama, D.; Blunsom, P.; Dyer, C.; Grefenstette, E.; and Ling, W. 2017. Learning to compose words into sentences with reinforcement learning. In *ICLR*.

Zeiler, M. D. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhao, H.; Lu, Z.; and Poupart, P. 2015. Self-adaptive hierarchical sentence model. In *IJCAI*, 4069–4076.

Zhou, P.; Qi, Z.; Zheng, S.; Xu, J.; Bao, H.; and Xu, B. 2016. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. In *COLING*, 3485–3495.