Encoding Syntactic Knowledge in Neural Networks for Sentiment Classification

MINLIE HUANG*, QIAO QIAN*, and XIAOYAN ZHU, State Key Laboratory of Intelligent Technology and Systems, National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 10084, China

Phrase/Sentence representation is one of the most important problems in natural language processing. Many neural network models such as Convolutional Neural Network (CNN), Recursive Neural Network (RNN), and Long Short-Term Memory (LSTM) have been proposed to learn representations of phrase/sentence, however, rich syntactic knowledge has not been fully explored when composing a longer text from its shorter constituent words. In most traditional models, only word embeddings are utilized to compose phrase/sentence representations, while the syntactic information of words is yet to be explored. In this article, we discover that encoding syntactic knowledge (part-of-speech tag) in neural networks can enhance sentence/phrase representation. Specifically, we propose to learn tag-specific composition functions and tag embeddings in recursive neural networks, and propose to utilize POS tags to control the gates of tree-structured LSTM networks. We evaluate these models on two benchmark datasets for sentiment classification, and demonstrate that improvements can be obtained with such syntactic knowledge encoded.

CCS Concepts: • **Computing methodologies → Neural networks**; **Lexical semantics;**

Additional Key Words and Phrases: Neural networks, recursive neural network, long short-term memory, deep learning, representation learning, sentiment classification, sentiment analysis

ACM Reference Format:
Minlie Huang, Qiao Qian, and Xiaoyan Zhu. 2017. Encoding syntactic knowledge in neural ntworks for sentiment classification. ACM Trans. Inf. Syst. 35, 3, Article 26 (June 2017), 27 pages.
DOI: http://dx.doi.org/10.1145/3052770

1. INTRODUCTION

Phrase/sentence representation, which represents a phrase or sentence with a real-valued, continuous vector, is a fundamental problem in natural language processing. Representation learning for text has been shown to advance many Natural Language Processing (NLP) tasks such as paraphrase detection [Socher et al. 2011a], sentiment analysis [Socher et al. 2011b], question answering [Dong et al. 2015], text generation [Sutskever et al. 2014], and many more. Recently, many neural network models have been proposed for learning phrase/sentence representations, and these methods generically fall into three categories: bag-of-words models, sequence models, and tree-structured models.

*The two authors contribute equally.

This work was partly supported by the National Basic Research Program (973 Program) under grant No. 2013CB329403, the National Science Foundation of China under grant No. 61272227/61332007, and the Beijing Higher Education Young Elite Teacher Project.

Authors' addresses: M. Huang, Q. Qian, and X. Zhu, State Key Laboratory of Intelligent Technology and Systems, National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 10084, China; emails: aihuang@tsinghua.edu.cn, qianqiaodecember29@126.com, zxy-dcs@tsinghua.edu.cn.

ACM 1046-8188/2017/06-ART26 $15.00
DOI: http://dx.doi.org/10.1145/3052770

In bag-of-words models, phrase and sentence representations are usually averaged over constituent phrase representations, and word order is constantly ignored [Landauer and Dumais 1997; Foltz et al. 1998]. The shorter text (e.g., word) is represented by a continuous vector that can be obtained via the CBOW [Mikolov et al. 2013b] or Skip-gram models [Mikolov et al. 2013a], or Glove [Pennington et al. 2014], or other approaches. Then, the representation of a phrase or sentence is an average (sometimes weighted average) of all words' vectors.

In contrast, sequence models build sentence representations as an order-sensitive function of the sequence of tokens [Elman 1990; Mikolov 2012]. Many neural network models fall into this line. Convolutional Neural Network (CNN) defines convolution operations from the beginning of a sentence to the end [Kalchbrenner et al. 2014]. The convolution layer firstly concatenates consecutive words' vectors and then operates convolution filters to the concatenated vector; the pooling layer takes the maximum or average of convoluted input vectors as the final or intermediate representation of the input text. Recurrent Neural Network (ReNN) models the units of a sequence by introducing a recurrent connection, where the current hidden state is determined by the previous hidden state and the current word input. For very long sequences, ReNN defines a very deep structure, and hence suffers from the gradient vanishing or explosion problems. The vanishing gradients were successfully addressed by Hochreiter and Schmidhuber [1997], where the Long Short-Term Memory (LSTM) architecture was proposed. Since LSTM is resistant to the vanishing gradient problem, the model has been commonly used to represent sentences, paragraphs, and documents [Tang et al. 2015; Li et al. 2015].

Tree-structured models compose phrase or sentence representation from its constituent phrases according to a tree structure. A recursive autoencoder network (Recursive Neural Network (RNN)) [Socher et al. 2011a, 2011b; Dong et al. 2014; Qian et al. 2015] defines the phrase composition process according to a parsing tree (or rarely binary trees generated by greedy algorithms), and the tree structure is grown up in a bottom-up manner where the root node corresponds to the sentence representation. Such recursive structures can be stacked to form a multilayer RNN, which is termed deep RNN [Irsoy and Cardie 2014]. LSTM can also be accommodated with tree structures, that is, tree-LSTM [Zhu et al. 2015; Tai et al. 2015]. With slight modifications on the memory state, input, output, and forget gates, tree-LSTM can work better than sequential LSTMs.

In the aforementioned models, few have fully explored the syntactic knowledge for phrase or sentence representation. Though the parsing tree structure is used to lead the composition process, the part-of-speech tag, and the relationship between the child and parent nodes have not been investigated. In this article, we aim at learning *sentiment-favorable representations* to improve the performance of sentiment classification. We define sentiment-favorable representation as what is learned by a proper way of expressing sentiment, and is usually optimized with a sentiment-specific loss. Most existing methods for learning sentiment-favorable representation are to combine a sentiment-specific loss with the original loss, and then back-propagate the supervision to low-level layers [Tang et al. 2014; Liu et al. 2015]. However, in deep structures, the supervison from the output layer may vanish gradually when it is propagated to lower layers. In Socher et al. [2011b, 2012, 2013b], the sentiment-specific cross-entropy loss is attached to every node of a parsing tree such that much more supervision can be used for learning representations, but unfortunately, this requires heavy annotation on all subphrases of a parsing tree. In spite of the success of these models, they have not fully utilized rich syntactic knowledge (for instance, part-of-speech tags) of all nodes in a parsing tree. We believe that better representations may be obtained if the recursive structure can encode more syntactic knowledge at all layers.

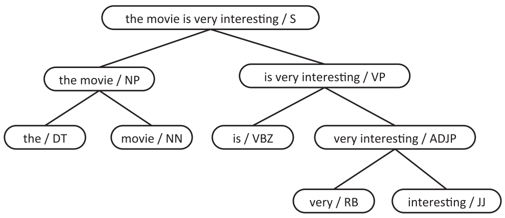

Fig. 1. The parse tree for the sentence *"The movie is very interesting."* The part-of-speech tags ({S, NP, VP, DT, NN, VBZ, ADJP, RB, JJ}) are attached to the words/phrases with a slash symbol.

The idea of using syntactic knowledge for phrase composition is motivated by the running example shown in Figure 1. First, the composition means for the noun phrase *"the movie / NP"* should be different from that for the adjective phrase *"very interesting / ADJP"* since the two phrases are syntactically quite different. More specifically to sentiment analysis, a noun phrase is much less likely to express sentiment than an adjective phrase. Accordingly, our idea is to learn a tag-specific composition functions or weights for phrase composition. For instance, we design a function for composing noun phrase (*NP*) and another one for adjective phrase (*ADJP*), or we learn a weight for *NP* phrases and another weight for *ADJP* phrases and then leverage these weights for phrase composition.

Second, when composing the adjective phrase *"very interesting / ADJP"* from the left node *"very / RB"* and the right node *"interesting / JJ,"* the right node is obviously more important than the left one in expressing sentiment. The right node *"interesting / JJ"* apparently contributes more to sentiment expression. To address this issue, we propose to learn an embedding vector for each part-of-speech tag, and then let the tag vector participate in the composition process. For instance, we learn tag embedding vectors for *DT, NN, RB, JJ, ADJP, NP*, etc., and the tag vectors are then used in composing the parent's vector. In this manner, information about part-of-speech tags can be encoded in the composition process.

To summarize, our contributions are as follows:

—We propose the idea of encoding syntactic knowledge in neural networks. One way is to learn tag-specific composition functions or weights for different part-of-speech tags; the other way is to learn tag embedding vectors and then let the tag vectors participate in the composition process.
—We propose Tag-Guided Recursive Neural Network (TG-RNN), which employs tag-specific composition functions for different POS tags. We propose to learn tag embedding vectors and combine them in RNN and RNTN (Recursive Neural Tensor Network). The corresponding models are Tag Embedded RNN (TE-RNN) and Tag Embedded RNTN (TE-RNTN), which both leverage the POS tags of child nodes when generating the vector of parent nodes.
—We study the effectiveness of employing part-of-speech tags in tree-structured LSTM. One way is to learn tag-specific weights to control the gates of LSTM and the resulting model is Tag Weighted LSTM (TW-LSTM). We can see that although the POS tag has been largely abstracted, this model is superior or comparable to many popular models such as CNNs and RNNs. The other way is to learn the embedding vector of POS tags and let tag embeddings participate in the control of LSTM gates. The resulting model, Tag Embedded LSTM (TE-LSTM), is superior to all other models.
—Although these models are yet very simple, the proposed models are efficient and effective. The models are very compact, and the scale of the parameters is well controlled. The evaluation on two benchmark datasets for sentiment classification show that all the proposed models are superior to their counterparts.

The rest of the article is organized as follows. We survey related work in Section 2. We present how to encode syntactic knowledge in RNN in Section 3, and in LSTM in Section 4. Experiments and detailed analysis are presented in Section 5. Our work is summarized in Section 7.

2. RELATED WORK

The traditional way for phrase/sentence representation is bag-of-words representation, which adopts a sparse vector over a large vocabulary (usually thousands of lexical entries). This type of representation has been widely used in many NLP tasks, including sentiment analysis [Pang and Lee 2008], however, it also suffers from the sparsity issue particularly for short text, and is not able to encode corpus-level context information. Thanks to the renaissance of neural network models, representing word/phrase/sentence with a low-dimensional dense vector has been substantially advanced in recent years.

Representing words with real-valued, dense vectors is first approached by the Neural Language Model [Bengio et al. 2003]. Similar to the same idea, CBOW [Mikolov et al. 2013a] and Skip-gram [Mikolov et al. 2013b] introduce a simpler network structure but make computation more efficient to enable billions of samples feasible for training. The two models either predict the contextual words given a word, or predict a current word given its neighboring words. Although the models yield very simple neural network structures, the generated word embeddings have been shown to be very effective in various tasks, well capable of maintaining language regularities, and soon become prevalent in serving as input vectors to neural network models. Unlike these prediction-based methods, Glove is count based [Pennington et al. 2014], which is a global bilinear regression model that is purely based on corpus-level cooccurrence statistics.

Once we are able to represent words effectively, the next issue is the semantic composition problem, which aims to obtain representations for a longer text from its short segments. In many previous works, a phrase vector is usually obtained by average [Landauer and Dumais 1997; Foltz et al. 1998], addition, element-wise multiplication [Mitchell and Lapata 2008], or tensor product [Smolensky 1990] of word vectors. In addition to using vector representations, matrices can also be used to represent phrases and the composition process can be done through matrix multiplication [Rudolph and Giesbrecht 2010; Yessenalina and Cardie 2011].

While there are so many various methods for semantic composition, in this survey, we are particulary focusing on those methods that are based on typical neural network models. Sequence models, including CNN, ReNN, and LSTM, are widely studied for representing phrases and sentences. CNN usually constructs sentence presentations by defining convolution operations over adjacent words from the beginning to the end of a sentence [Kim 2014; Kalchbrenner et al. 2014; Lei et al. 2015]. The convolution layer concatenates vectors of consecutive words, and operates convolution filters on the concatenated vector. A pooling layer simply takes the maximum or average of vectors from the preceding layer as the final representation of given texts. ReNN models a sequence of words with a recurrent connection, where the representation up to the current position is dependent on the previous word's representation and the current word vector, and the final representation is obtained at the end of the sequence. However, for long sentences, ReNN becomes difficult to train due to the vanishing gradient problems. An improved version of ReNN is Gated Recurrent Units (GRUs) [Chung et al. 2014], which introduce controlled gates into the network. By introducing additional memory layers and input/output/forget gates, LSTM is more competitive to model long-range dependency than ReNN and GRU, becoming prevalent for representing sentences, paragraphs, and documents [Tang et al. 2015; Li et al. 2015]. Many variants of LSTM have been proposed, such as Bidirectional LSTM, which models sentences in normal and reversed order separately, and deep LSTM, which stacks multiple LSTM layers together [Graves et al. 2013].

Tree-structured models usually compose a longer text from its short segments recursively according to a tree structure, including RNN and its variants such as RNTN and matrix vector RNN (MV-RNN) [Socher et al. 2011a, 2011b, 2012, 2013b], and tree-structured LSTM [Tai et al. 2015; Zhu et al. 2015]. RNN defines a recursive structure over a parsing tree, and then composes phrase representation recursively with a composition function in a bottom-up way. Adaptive Recursive Neural Network [Dong et al. 2014] uses a weighted sum of multiple composition functions during the composition process, while Qian et al. [2015] encodes syntactic knowledge in the composition function of RNN. Several recursive structures can be stacked together to form a deep structure [Irsoy and Cardie 2014]. LSTM can also be used to model a parsing tree of a sentence [Zhu et al. 2015; Tai et al. 2015], by making slight modifications of memory states and controlled gates over tree structures.

Some studies exist for learning sentiment-favorable representations. A typical way is to use the end-to-end learning strategy where a sentiment-specific loss is combined with the original loss and the combined loss is then back-propagated into low-level layers. For instance, Tang et al. [2014] proposes to learn sentiment-specific word embedding on top of the C&W model [Collobert et al. 2011; Collobert and Weston 2008]. Severyn and Moschitti [2015b] shows that pretraining using weakly supervised data (distant supervision) to initialize the parameters of a CNN network can benefit the learning of sentiment-favorable representation and can result in state-of-the-art accuracy for sentiment analysis. Chen et al. [2017] demonstrates that sentence type is a factor of sentence-level sentiment classification and classifying sentences of each type separately leads to the performance improvement. Liu et al. [2015] proposes to learn sentiment-favorable representation of sentences/documents on top of stacked denoising autoencoders in the setting of domain adaptation. In these works, sentiment-specific representation learning is fully driven by sentiment-specific loss functions, however, rich grammatical or syntactical details in the language have been largely neglected.

The employment of syntactic information in neural networks is still in its infancy. In Socher et al. [2013a], the part-of-speech tag of child nodes is considered for combining the processes of both phrase composition and sentence parsing, with a purpose of improving parsing performance. In Hermann and Blunsom [2013], the authors designed composition functions according to the combinatory rules and categories in CCG grammar. However, only marginal improvement against Naive Bayes was reported. Our work presented in Qian et al. [2015] proposed a more elaborated way to encode more syntactic knowledge in all layers of a recursive neural network. In this article, we have substantially extended the work in Qian et al. [2015], and study how syntactic knowledge (part-of-speech tag) can help to learn sentiment-favorable representation with recursive neural networks and tree-structured LSTM. Two means for encoding syntactic knowledge have been studied: one way is to learn tag-specific weights or functions, and the other way is to learn tag embeddings during the composition process. These proposed models have been shown to be competitive against their counterparts.

In order to learn sentiment-favorable presentation for longer texts, we believe that sentiment lexicons [Hu and Liu 2004; Wilson et al. 2005] are quite important since such resources provide the prior polarity of a word. Semantic composition of a long text will definitely benefit from such prior knowledge. There are a few works on automatic construction of such lexicons [Severyn and Moschitti 2015a; Chen and Skiena 2014; Vo and Zhang 2016], however, to the best of our knowledge, there is still not much work reported on how such lexicon resources can be seamlessly integrated with neural networks. Teng et al. [2016] approaches sentiment classification as a weighted sum of

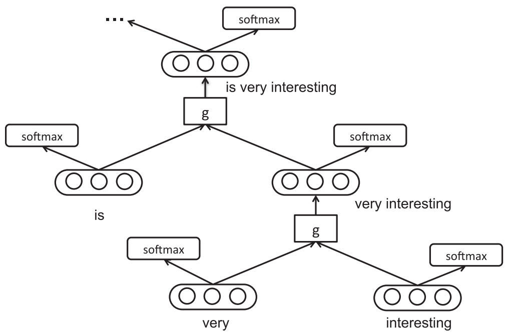

Fig. 2. The example process of vector composition in recursive neural networks. The vector of node *"very interesting"* is composed from the vectors of node *"very"* and node *"interesting."* Similarly, the node *"is very interesting"* is composed from the phrase node *"very interesting"* and the word node *"is."*

the prior polarity of each term in a sentence and the weights are learned by neural network.

3. ENCODING SYNTACTIC KNOWLEDGE IN RECURSIVE NEURAL NETWORKS

3.1. Preliminaries

In recursive neural models, the vector of a longer text (e.g., sentence) is recursively composed from those of its shorter segments (e.g., words or phrases). To compose a sentence vector through word/phrase vectors, a tree structure is usually required, which can be obtained by a parser. The composition process is exemplified in Figure 2. The leaf nodes represent words and interior nodes represent phrases. Vectors of interior nodes are computed recursively by composing child nodes' vectors. Particularly, the root vector is regarded as the sentence representation.

More formally, the representation vector $h_i \in R^d$ for node i is calculated via

$$h_i = f\big(g\big(h_i^l, h_i^r\big)\big), \tag{1}$$

where h_i^l and h_i^r are the left and right child vectors, g is a composition function, and f is a nonlinearity function, usually *tanh*. Different recursive neural models mainly differ in applying different composition functions. The most traditional recursive neural model takes the following function:

$$g\big(h_i^l, h_i^r\big) = W\begin{bmatrix} h_i^l \\ h_i^r \end{bmatrix} + b, \tag{2}$$

where $W \in R^{d\times 2d}$ is a composition matrix and b is a bias vector. And the composition function for RNTN is as follows:

$$g\big(h_i^l, h_i^r\big) = \begin{bmatrix} h_i^l \\ h_i^r \end{bmatrix} T^{[1:d]} \begin{bmatrix} h_i^l \\ h_i^r \end{bmatrix} + W\begin{bmatrix} h_i^l \\ h_i^r \end{bmatrix} + b, \tag{3}$$

where W and b are defined as previously, and $T^{[1:d]} \in R^{2d\times 2d\times d}$ is the tensor that defines multiple bilinear forms.

The vectors are used as feature inputs to a softmax classifier. The posterior probability over class labels on a node vector h_i is given by

$$y_i = \text{softmax}(W_s h_i + b_s). \tag{4}$$

The parameters in these models include a word table L, a composition matrix W in RNN, and W and $T^{[1:d]}$ in RNTN, and a classification matrix W_s for the softmax classifier.

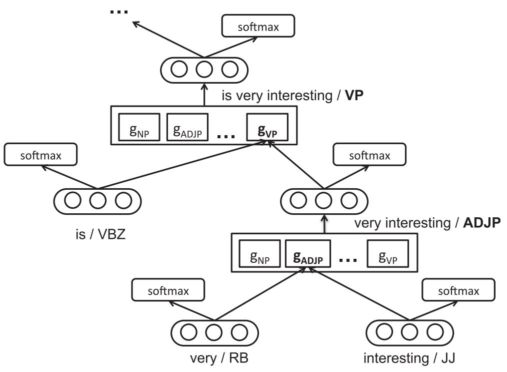

Fig. 3. The example composition process of TG-RNN. The vector of a phrase *"very interesting"* is composed with highlighted $\boldsymbol{g_{ADJP}}$ and *"is very interesting"* with $\boldsymbol{g_{VP}}$. The composition function is chosen according to the part-of-speech tag.

The idea of encoding syntactic knowledge in RNN is inspired by the fact that words/phrases of different part-of-speech tags play different roles in semantic composition. As discussed in the Introduction, a noun phrase (e.g., *a movie*/*NP*) may be composed in a different way from that for a verb phrase (e.g., *love movie*/*VP*). Furthermore, when composing the phrase *a movie*/*NP*, the two child words, *a*/*DT* and *movie*/*NN*, may play different roles in the composition process. However, the traditional RNN models neglect such syntactic information, though the model indeed employs the parsing structure of a sentence.

We have two approaches to improve the composition process by leveraging tags on parent nodes and child nodes. One approach is to use different composition matrices for parent nodes with different tags such that the composition process could be guided by phrase type, for example, the matrix for "*NP*" is different from that for "*VP*." The other approach is to introduce "tag embedding" for words and phrases, for example, to learn tag vectors for *"NP, VP, ADJP,"* etc., and then integrate the tag vectors with the word/phrase vectors during the composition process.

3.2. Tag Guided Recursive Neural Network

We propose Tag Guided Recursive Neural Network (TG-RNN) to model the influence of the tag of a parent phrase during the composition process. The model chooses a composition function according to the part-of-speech tag of a phrase. For example, since *"the movie"* has a tag of *NP*, *"very interesting"* has a tag of ; the two phrases have different composition matrices.

More formally, we design composition functions g with a factor of the part-of-speech tag of a parent node. The composition function then becomes

$$g(t_i, h_i^l, h_i^r) = g_{t_i}(h_i^l, h_i^r) = W_{t_i}\begin{bmatrix} h_i^l \\ h_i^r \end{bmatrix} + b_{t_i}, \tag{5}$$

where t_i is the tag for node i (a word or phrase), and W_{t_i} and b_{t_i} are the parameters of function g_{t_i}, similar to those in Equation (2). In other words, phrase nodes with different tags have their own composition functions such as g_{NP}, g_{VP}, and so on. There are totally k composition functions in this model where k is the number of phrase tags. When composing a parent vector from its child vectors, a function is chosen from the function pool according to the tag of the parent node. The process is depicted in Figure 3. We term this model Tag guided RNN; TG-RNN for short.

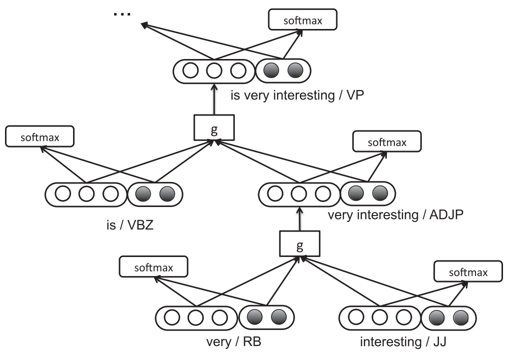

Fig. 4. The composition process of TE-RNN. There is a tag embedding table, storing vectors for all POS tags. The phrase vector *"very / RB interesting / JJ"* is composed from the vectors for *"very / RB"* and *"nteresting / JJ,"* and the tag vectors for *RB* and *JJ*.

But some tags have few occurrences in the corpus. It is hard and meaningless to train composition functions for those infrequent tags. So we simply choose top k frequent tags and train k composition functions. A common composition function is shared across phrases with all infrequent tags. The value of k depends on the size of the training set and the occurrences of each tag. Specially, when $k = 0$, the model is the same as the traditional RNN.

3.3. Tag Embedded Recursive Neural Network/Tensor Network

To model the effect of the part-of-speech tags of child nodes on semantic composition, we propose tag embedded recursive neural network (TE-RNN) and tag embedded recursive neural tensor network (TE-RNTN). As mentioned previously, the tag of a phrase indeed influences how the phrase is composed from its segments. However, some phrases with the same tag should be composed in different ways. For example, *"is / VB interesting / JJ"* and *"like / VB swimming / NN"* both have the same *VP* tag. But it is not reasonable to compose the two phrases using the previous model because the part-of-speech tags of their children are quite different. If we use different composition functions for children with different tags like TG-RNN, the number of tag pairs will amount to as many as $k \times k$, which makes the models infeasible due to too many parameters.

In order to capture the compositional effects of the tags of child nodes, an embedding $e_t \in R^{d_e}$ is created for each tag t, where d_e is the dimension of the tag vector. The tag vector and phrase vector are concatenated during composition as illustrated in Figure 4.

Formally, the phrase vector is composed by the function

$$g(h_i^l, e_{t_i^l}, h_i^r, e_{t_i^r}) = W \begin{bmatrix} h_i^l \\ e_{t_i^l} \\ h_i^r \\ e_{t_i^r} \end{bmatrix} + b, \tag{6}$$

where t_i^l and t_i^r are tags of the left and the right nodes, respectively, $e_{t_i^l}$ and $e_{t_i^r}$ are tag vectors, and $W \in R^{d \times (2d_e + 2d)}$ is the composition matrix. We term this model Tag embedded RNN; TE-RNN for short.

Similarly, this idea can be applied to RNTN [Socher et al. 2013b]. In RNTN, the tag vector and the phrase vector can be interweaved together through a tensor. More specifically, the phrase vectors and tag vectors are multiplied by the tensor, as follows:

$$g(h_i^l, e_{t_i^l}, h_i^r, e_{t_i^r}) = \begin{bmatrix} h_i^l \\ e_{t_i^l} \\ h_i^r \\ e_{t_i^r} \end{bmatrix} T^{[1:d]} \begin{bmatrix} h_i^l \\ e_{t_i^l} \\ h_i^r \\ e_{t_i^r} \end{bmatrix} + W \begin{bmatrix} h_i^l \\ e_{t_i^l} \\ h_i^r \\ e_{t_i^r} \end{bmatrix} + b, \tag{7}$$

where the variables are similar to those defined in Equations (3) and (6). We term this model Tag embedded RNTN; TE-RNTN for short.

The phrase vectors and tag vectors are used as input to a softmax classifier, giving the posterior probability over labels via the following equation:

$$y_i = \text{softmax}\left(W_s \begin{bmatrix} h_i \\ e_{t_i} \end{bmatrix} + b_s\right). \tag{8}$$

3.4. Model Training

Let $\hat{y}_j$ be the gold-standard sentiment distribution for phrase j, and y_j be the predicted distribution, which is given by either Equation (4) or (8). The learning goal is to minimize the cross-entropy error between y_j and $\hat{y}_j$ for all nodes. The loss function is defined as follows:

$$\mathcal{L}(\theta) = -\sum_j \sum_{m=1}^{M} \hat{y_j^m} \log y_j^m + \lambda ||\theta||_2^2, \tag{9}$$

where M is the total number of classes (e.g., positive and negative), λ is the weight for the L_2-regularization term, and θ is the parameter set.

The parameters for our models are as follows:

TG-RNN: There are k composition matrices for top k frequent tags, which are defined as $W_t \in R^{k \times d \times 2d}$. The original composition matrix W is occupied for all infrequent tags. As a result, the parameter set of TG-RNN is $\theta = (L, W, W_t, W_s)$ where L is the word embedding table, which will be fine-tuned during training.

TE-RNN: The parameters include the tag embedding table E, which contains all the embeddings for part-of-speech tags for words and phrases; the matrix $W \in R^{d \times (2d+2d_e)}$ in Equation (6), and the weight matrix of the softmax classifier $W_s \in R^{N \times (d_e+d)}$ in Equation (8). Thus, the parameter set of TE-RNN is $\theta = (L, E, W, W_s)$.

TE-RNTN: This model has one more tensor $T \in R^{(2d+2d_e) \times (2d+2d_e) \times d}$ than TE-RNN, as presented in Equation (7). The parameter set of TE-RNTN is $\theta = (L, E, W, T, W_s)$.

4. ENCODING SYNTACTIC KNOWLEDGE IN LSTM

4.1. Preliminaries

LSTM has been very popular due to its capability to model long-range dependency in sequential data. Zhu et al. [2015] and Tai et al. [2015] proposed that LSTM can also be applied to tree-structured data, for instance, parsing trees of natural language sentences. Such LSTM models are termed Tree-LSTM in this article.

The example process of a Tree-LSTM can be seen from Figure 5. Although Tree-LSTM can be applied to N-ary trees, for simplicity, we will present the bottom-up composition process with a binary tree. As depicted, there are hidden representations for each child phrase; say, h_j^l for the left phrase "*is*" and h_j^r for the right phrase "*very*

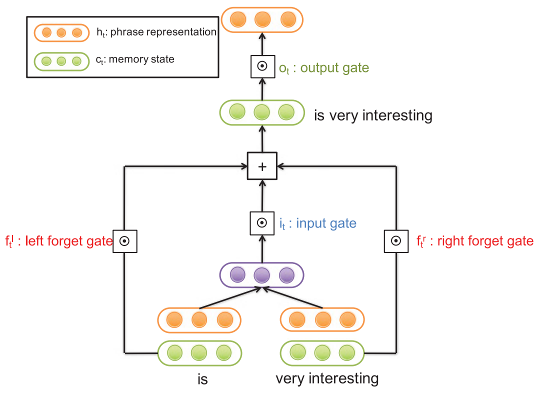

Fig. 5. The phrase composition process of Tree-LSTM. There are input, forget, and output gates in LSTM. Each node is associated with a phrase representation vector and memory state vector. $\odot$ denotes element-wise multiplication. The variable symbols are also presented in the figure.

interesting.” A temporary representation can be obtained immediately, as follows:

$$u_j = f\left(W\begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix} + b\right). \tag{10}$$

In addition to the hidden state h_j for each phrase j, there is a memory state vector c_j in Tree-LSTM. The memory state vector of phrase j can be obtained via

$$c_j = i_j \odot u_j + f_j^l \odot c_j^l + f_j^r \odot c_j^r, \tag{11}$$

where $c_j^l/c_j^r \in R^d$ are memory state vectors for the right/left child phrase, which can be obtained recursively; the operator $\odot$ denotes element-wise multiplication; and i_j, f_j^l, $f_j^r \in R^d$ are the input gate, the left forget gate, and the right forget gate, respectively. d is the dimension of word embeddings.

The phrase representation h_j is then computed as follows:

$$h_j = o_j \odot f(c_j), \tag{12}$$

where $o_j \in R^d$ is the output gate.

The input, forget, and output gates can be computed via the following equations:

$$i_j = S_i\begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}, \tag{13}$$

$$f_j^l = S_f^l\begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}, \tag{14}$$

$$f_j^r = S_f^r\begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}, \tag{15}$$

$$o_j = S_o\begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}, \tag{16}$$

where $S_i, S_f^l, S_f^r, S_o \in R^{d\times 2d}$ are weight matrices to be learned in Tree-LSTM.

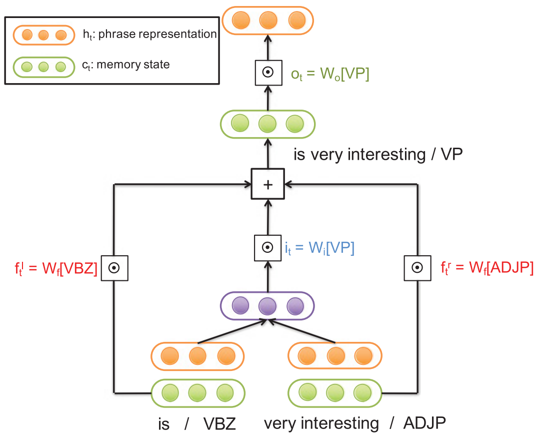

Fig. 6. The composition process in Tag Weighted LSTM. The information input to the gates is simply looked up from three global weight matrices according to the corresponding part-of-speech tags. The variable symbols are also presented in the figure.

For leaf nodes that have no child phrase, the representation of such nodes (h_j) and the memory state vector (c_j) can be computed via

$$c_j = W_u v_j, \tag{17}$$

$$h_j = \sigma\left(S_o^{leaf} v_j\right) \odot f(c_j), \tag{18}$$

where W_u, $S_o^{leaf} \in R^{d\times d}$ are parameters to learn, and v_j is the word embedding vector.

Our goal is to leverage syntactic knowledge to learn sentiment-favorable representation for classification. Let us revisit the motivating example: *"the/DT movie/NN is/VBZ very/RB interesting/JJ."* Obviously, in this example, the adjective and adjunctive words play a critical role in expressing sentiment. Therefore, we believe that such part-of-speech tags may be useful in learning sentiment-favorable representations. This inspires us to ulitilize part-of-speech tags to control information flowing through the input, forget, and output gates of Tree-LSTM.

More specifically, we design three ways for this purpose: First, we learn a global weight matrix for each specific tag, where the weight matrix controls the gates in LSTM; second, we learn an embedding vector for each tag, and then use tag embeddings to control the gates in LSTM; third, since part-of-speech tags have been overabstracted, we combine tag embeddings and phrase representations to control the gates.

4.2. Learning Tag-specific Gate Weights: Tag Weighted LSTM

A simple idea is to use the part-of-speech tag of each word/phrase to directly control the gates in LSTM. This idea is illustrated in Figure 6.

We design three global weight matrices for the input, forget, and output gates, respectively. Instead of feeding the hidden presentation of child nodes to the gates, we directly look up the weight matrices according to the tag of a child node. This simple strategy can be clearly seen in the following equations:

$$i_j = \sigma(W_i[t_j]), \tag{19}$$

$$f_j^l = \sigma(W_f[t_j^l]), \tag{20}$$

$$f_j^r = \sigma(W_f[t_j^r]), \tag{21}$$

$$o_j = \sigma(W_o[t_j]), \tag{22}$$

where $W_i, W_f, W_o \in R^{d \times n_t}$ are the weight matrices for the input gate, forget gate, and output gate, respectively. $W[t]$ is to fetch a column vector corresponding to tag t where each column in these matrices (W_i, W_f, W_o) corresponds to the weights of a POS tag. n_t is the number of POS tags. t_j is the POS tag for node j, and t_j^l and t_j^r are the POS tags for the left child and right child of node j, respectively.

Subsequently, the memory state c_j and the hidden representation of a phrase h_j could be calculated with those gates, as shown in Figure 6. Formally, we have

$$\begin{aligned} c_j = \sigma(W_i[t_j]) \odot u_j + \sigma\left(W_f\left[t_j^l\right]\right) \odot c_j^l \\ + \sigma\left(W_f\left[t_j^r\right]\right) \odot c_j^r, \end{aligned} \tag{23}$$

$$h_j = \sigma(W_o[t_j]) \odot tanh(c_j). \tag{24}$$

We can clearly see the difference between the traditional Tree-LSTM and this new LSTM: in traditional Tree-LSTMs, the input, forget, and output gates are calculated with hidden representations of child nodes, while the new model simply uses the POS tag to control the gates.

4.2.1. Modeling Child-Parent Association. When composing a parent phrase from its child phrases, the tags of both child and parent phrases are influential, particularly in learning sentiment-favorable representations. For instance, "interesting movie/NN" is a noun phrase composed from an adjective word "interesting/JJ" and a noun word "movie/NN." "not good/ADJP" is an adjective phrase composed from a negation word "not/RB" and an adjective word "good/JJ." The former example shows the parent phrase can be determined by the adjective subphrase, while the latter example depends on both subphrases. In other words, the same tag (e.g., JJ) may play a different role in different types of parent phrases (NN vs. ADJP). Therefore, it is necessary to model the syntactic association between the child phrase and parent phrase, for instance, JJ-NN, JJ-ADJP, RB-ADJP, etc.

We propose to model the association between the child phrases and the parent phrase as follows: we design a weight matrix (W_{f_c}) for child phrases and another weight matrix (W_{f_p}) for parent phrases; and then use the two matrices to control the forget gates of the child phrases. Formally, we have

$$f_j^l = \sigma\left(W_{f_c}\left[t_j^l\right] + W_{f_p}[t_j]\right), \tag{25}$$

$$f_j^r = \sigma\left(W_{f_c}\left[t_j^r\right] + W_{f_p}[t_j]\right), \tag{26}$$

where $W_{f_c}, W_{f_p} \in R^{d \times n_t}$ are the weight matrices to control the forget gates of a child phrase and a parent phrase, respectively, and t_j^l, t_j^r, t_j are the POS tags of the left child, right child, and parent phrase, respectively.

4.3. Learning Gate Weights with Tag Embeddings: Tag Embedded LSTM

In the previous models, we simply learn global weights for each tag. Similar to Tag Embedded RNN, a better way is to learn an embedding vector for each tag, and let the tag embeddings participate in the control of the LSTM's gates. Following this idea, a tag vector $E[t] \in R^{d_e}$ will be learned for each tag t, where d_e is the dimension of the tag vector, usually far less than that of word embeddings (d). The tag embeddings are then taken as input to the gates of LSTM, as follows:

$$i_j = \sigma(U_i E[t_j]), \tag{27}$$

$$f_j^l = \sigma(U_f E[t_j^l]), \tag{28}$$

$$f_j^r = \sigma(U_f E[t_j^r]), \tag{29}$$

$$o_j = \sigma(U_o E[t_j]), \tag{30}$$

where $E \in R^{d_e \times n_t}$ is the tag embedding table (each column corresponds to a tag vector) to be learned, d_e is the dimension of tag embedding, n_t is the number of tags, and $U_i, U_f, U_o \in R^{d \times d_e}$ are the weight matrices similar to conventional LSTMs.

4.3.1. Learning Gate Weights with Parent Phrase. Similar to Section 4.2.1, we propose to take into account the association between child phrases and parent phrase. In this setting, the embedding vector of a parent phrase ($E[t_j]$) is concatenated by the vector of a child phrase ($E[t_j^l]$ or $E[t_j^r]$), and the concatenated vector is then taken as input to the left or right gates. This can be formally stated as follows:

$$f_j^l = \sigma\left(U_f \begin{bmatrix} E[t_j] \\ E[t_j^l] \end{bmatrix}\right), \tag{31}$$

$$f_j^r = \sigma\left(U_f \begin{bmatrix} E[t_j] \\ E[t_j^r] \end{bmatrix}\right), \tag{32}$$

where $U_f \in R^{d \times 2d_e}$ is the weight matrix for the right and left gate, respectively.

4.3.2. The Difference between LW-LSTM and TE-LSTM. TW-LSTM adopts global gate weights $W \in R^{d \times n_t}$ where d is the word embedding dimension (usually 300), and n_t is the number of tags (70 in this article). Hence, there is a d-dimensional vector to represent each tag, which is overparameterized since the total number of tags is 70. For those infrequent tags, the parameters are not well optimized due to the lack of data. For instance, there are 28 tags that appear less than 300 times on the SST datasets (see the experiments). Thus, for those infrequent tags, 300-dimensional vectors are not easy to be fully trained.

In TE-LSTM, we design weight matrices $U_f, U_i, U_o \in R^{d \times d_e}$, which are shared across different tags, and a global tag embedding matrix $E \in R^{d_e \times n_t}$, which is a lookup table for all tags. The number of parameters of this part in TE-LSTM ($\approx$300 * 20 + 70 * 20 = 7,400) is less than that of TW-LSTM ($\approx$300 * 70 = 21,000), and thus TE-LSTM is more compact. Experiments show that LE-LSTM outperforms TW-LSTM even though the former has less parameters. The reasons may be attributed to the following: (1) the U's in TE-LSTM are shared across different tags so that they can be sufficiently trained for even less frequent tags; and (2) a lower dimensional vector (e.g., 20) will be sufficient to represent the tag since the total number of tags is 70 and lower dimensional embedding can be well trained even for less frequent tags (note that there are more than 18 tags that appear less than 100 times on the Standford Sentiment Treebank (SST) dataset).

To summarize, the weight matrices in TE-LSTM are shared across all tags and the lower dimensional tag embeddings are more reasonable, facilitating the learning of these parameters for even long-tail, infrequent tags.

4.4. Combining Tag Embeddings and Phrase Representations

In the previous models, we only consider the part-of-speech tag of word/phrase to control the gates. However, the part-of-speech tag has been largely abstracted. For instance, *"is"* and *"like"* are both verbs, *"red"* and *"wonderful"* are both adjectives, but these words differ in sentiment expression. Therefore, it may not be sufficient to learn sentiment-favorable representations with only POS tags.

It is natural to combine the hidden representation of child phrases in our proposed models. Formally, we have

$$i_j = \sigma\left(\alpha \cdot U_i E[t_j] + (1-\alpha) \cdot S_i \begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}\right), \tag{33}$$

$$f_j^l = \sigma\left(\alpha \cdot U_f \begin{bmatrix} E[t_j] \\ E[t_j^l] \end{bmatrix} + (1-\alpha) \cdot S_f^l \begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}\right), \tag{34}$$

$$f_j^r = \sigma\left(\alpha \cdot U_f \begin{bmatrix} E[t_j] \\ E[t_j^r] \end{bmatrix} + (1-\alpha) \cdot S_f^r \begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}\right), \tag{35}$$

$$o_j = \sigma\left(\alpha \cdot U_o E[t_j] + (1-\alpha) \cdot S_o \begin{bmatrix} h_j^l \\ h_j^r \end{bmatrix}\right), \tag{36}$$

where α is a hyperparameter to control the trade-off between the input from POS tags and that from words/phrases. In this setting, we can clearly see that the input/forget/output gates are controlled by both syntactic and verbal information.

4.5. Model Training

The learning goal is the same as that in Equation (9). For brevity, we will not repeat the equation here but only list the parameter set for each model, as follows.

TG-LSTM: The parameter set $\theta = (L, W, W_i, W_f, W_o, W_u, W_s)$ where L is the word embedding table to be fine-tuned, $W \in R^{2d \times d}$ is the composition matrix for composing the left and right child nodes (as defined in Equation (10)), $W_i, W_f, W_o \in R^{d \times n_t}$ are the matrices for computing input, forget, and output gates, respectively, W_u is the matrix of transforming word vectors to memory representations for leaf nodes (as defined in Equation (17)), and $W_s \in R^{d \times M}$ is the weight matrix of the softmax layer.

TG-LSTM with child-parent association (+p): $\theta = (L, W, W_i, W_{f_c}, W_{f_p}, W_o, W_u, W_s)$ where we have additional parameters W_{f_c}, W_{f_p} for the child and parent forget gates, respectively.

TE-LSTM: $\theta = (L, W, E, U_i, U_f, U_o, W_u, W_s)$ where $E \in R^{d_e \times n_t}$ is the tag embedding table, $U_i, U_f, U_o \in R^{d \times d_e}$ are the parameters for computing the input, forget, and output gates, respectively, and all the other parameters are similar to the aforementioned.

TE-LSTM + with parent phrase (+p): The only difference to those in TE-LSTM is that $U_f \in R^{d \times 2d_e}$, as can be seen from Equations (31) and (32).

Models with combination of tag and word representations (+c): There are additional parameters $\theta' = (S_i, S_f^l, S_f^r, S_o, S_o^{leaf})$ where the first four matrices are for the input, left forget, right forget, and output gates, respectively, and S_o^{leaf} is for the leaf node as defined in Equation (18).

5. EXPERIMENTS

5.1. Datasets

Two datasets are used for evaluating the proposed models. The first one is the SST dataset [Socher et al. 2013b] and the second one is the Movie Review (MR) dataset [Pang and Lee 2005]. SST contains 11,855 sentences, and has been split into the training/validation/test parts, containing 8,544/1,101/2,210 sentences, respectively. All phrases (obtained by a parser) in each sentence have been annotated with five sentiment labels, ranging from very negative, negative, neutral, to positive and very positive. There are two tasks on this dataset: fine-grained classification on the five class labels, and binary classification on positive/negative, which ignores all neutral nodes.

The MR dataset is different from SST in that MR only includes sentence-level annotation with positive/negative labels. The dataset contains 10,662 sentences; half for positive and another half for negative. To be consistent with other baselines, our experiments are conducted with 10-fold cross validation; nine folds for training and the remainder for test. Additionally, 10% of the training data is held out for validation. To obtain the tree structure of a sentence, we resort to Stanford Parser [Klein and Manning 2003]. Since we have no annotation on internal phrases of a parser tree, the models learn with supervision from the root node's annotation, which corresponds to the sentence-level label.

In all the following experiments, accuracy is adopted for measuring the performance of sentence-level sentiment classification.[1]

5.2. Experiment Settings

5.2.1. Settings for RNN Models. The word vectors were pretrained on an unlabeled corpus (about 100,000 movie reviews collected by ourselves, and will be released soon) by word2vec [Mikolov et al. 2013b] as initial values and the other vectors are initialized by sampling from a uniform distribution $\mathcal{U}(-\epsilon, \epsilon)$ where ϵ is 0.01 in our experiments. The dimension of word vectors is 25 for RNN models and 20 for RNTN models.[2] *Tanh* is chosen as the nonlinearity function. And after computing the output of node i with $h_i = f(g(h_i^l, h_i^r))$, we set $h_i = \frac{h_i}{||h_i||}$ so that the resulting vector has a limited norm. A back-propagation algorithm [Rumelhart et al. 1986] is used to compute gradients, implemented with Theano [Bastien et al. 2012]. We trained all our models using Stochastic Gradient Descent (SGD) with a batch size of 30 examples, momentum of 0.9, L_2-regularization weight of 0.0001, and a constant learning rate of 0.005.

5.2.2. Settings for LSTM Models. We adopt Glove vectors [Pennington et al. 2014] as the initial setting of word embeddings.[3] The tag-specific weight matrices (W_i, W_f, W_o) are initialized with the distribution of $Uniform(0, 0.2)$. The tag embedding matrix (E) is initialized with $Uniform(0, 0.5)$. Other parameters are initialized with $Uniform(0, 1/sqrt(d))$, where d is the dimension of hidden representation, and we set d=300. The dimension of tag embedding (d_e) is set to 20; however, it is insensitive to performance in our experiments. We adopt adaGrad to train the models, and the learning rate is 0.05. It is worth noting that we adopt stochastic gradient descent to update the word embeddings (V), with a learning rate of 0.1 but without momentum.

The λ parameter for the regularization term is set to 0.0001. During training, we adopt the dropout operation before the softmax layer, with a rate of 0.5. Minibatch is taken to train the models, each batch containing 25 samples. After training with 12 epochs, if the model performs best on the validation dataset, we will choose that point as our final model.

[1]Although some works reported phrase-level classification on SST, most of the works only include sentence-level classification, like ours in this article.

[2]The number is chosen to make our models fairly comparable with the baselines including RNN [Socher et al. 2011b], RNTN [Socher et al. 2013b], and AdaMC-RNN/RNTN [Dong et al. 2014] (see Section 5.3).

[3]We choose word embeddings that differ from those for RNN models, to make our models fairly comparable with the counterpart baselines because Tree-LSTM reports the best performance when Glove is used [Tai et al. 2015].

5.3. Performance Comparison

We include several genres of classical baselines in the evaluation. The baselines are listed as follows:

—**SVM.** A SVM model with bag-of-words representation [Pang and Lee 2008] where each dimension in the feature vector corresponds to a lexical feature.
—**MNB/bi-MNB.** Multinomial Naive Bayes and its bigram variant, adopted from Wang and Manning [2012].
—**RNN.** The traditional RNN model proposed by Socher et al. [2011b].
—**MV-RNN.** Matrix Vector Recursive Neural Network [Socher et al. 2012] represents each word/phrase with a vector and a matrix. As reported, this model suffers from too many parameters.
—**RNTN.** Recursive Neural Tensor Network [Socher et al. 2013b] employs a tensor for composition function, which is claimed to model the meaning of longer phrases and capture negation rules.
—**AdaMC-RNN/RNTN.** Adaptive Multi-Compositionality for RNN and RNTN [Dong et al. 2014] trains with a group of composition functions and adaptively learns the weight for each function.
—**CNN/DCNN.** A conventional Convolutional Neural Network [Kim 2014], and Dynamic Convolutional Neural Network [Kalchbrenner et al. 2014], which adopts a dynamic pooling function in the pooling layer. Note that the authors reported that they adopted 300-dimensional word embeddings obtained by word2vect.
—**DRNN.** Deep Recursive Neural Network [Irsoy and Cardie 2014] stacks multiple recursive layers.
—**LSTM/Bidirectional LSTM.** The traditional sequence-based LSTM and Bidirectional LSTM, which models a sentence in normal order and reverse order separately.
—**Tree-LSTM.** Tree-Structured Long Short-Term Memory [Tai et al. 2015]. The best performance is obtained when Glove vectors and constituency trees are applied, with word embeddings fine-tuned. Hence, to make comparison fair, we also adopt Glove vectors, with word embeddings fine-tuned.

First of all, we evaluate the proposed models on Stanford Sentiment Treebank. To make the results more convincing, we repeat the same experiment four times for each model, and present the mean and standard deviation of the results from four runs. The results are shown in Table I, where the results of baselines are reprinted[4] from the original papers. As a matter of fact, on this dataset, fine-grained classification accuracy is more convincing to evaluate the models than binary classification accuracy. From the results, we have the following observations:

—**TG-RNN** outperforms RNN, RNTN, MV-RNN, AdMC-RNN/RNTN. Compared with RNN, the fine-grained accuracy and binary accuracy of TG-RNN is improved by 3.8% and 3.9%, respectively. When compared with AdaMC-RNN, the accuracy of our method is enhanced by 1.2% on the fine-grained prediction. The results show that the syntactic knowledge indeed facilitates semantic composition in this task.
—**TE-RNN:** The fine-grained accuracy of TE-RNN is boosted by 4.8% compared with RNN. TE-RNN is comparable to CNN and DCNN. Furthermore, TE-RNN is also better than TG-RNN. This implies that learning the tag embeddings for child nodes

[4]Note that this dataset has a fixed partition of training/validation/test data; the baselines' results are directly comparable even without implementation of these models. Reprinting the results on this dataset is commonly seen in the literature. Nevertheless, we reimplemented RNN/RNTN in Table I, and RNN has a performance of 44.8%/83.6% for fine-grained and binary classification, respectively, while RNTN has a performance of 45.8%/84.9%, both close to what were reported in the original paper.

Table I. Accuray on the Sentiment Treebank Dataset. *p* Means Considering Child-Parent Association, and *c* Means Combining the Representations of Phrases and Tags. The Hyperparameter α is Set to an Optimal Value of 0.5 (See Section 5.5). The Values in Parentheses are the Standard Deviation*

Method	Fine-grained	Pos./Neg.
SVM [Pang and Lee 2008]	40.7	79.4
MNB [Wang and Manning 2012]	41.0	81.8
bi-MNB [Wang and Manning 2012]	41.9	83.1
RNN [Socher et al. 2011b]	43.2	82.4
RNTN [Socher et al. 2013b]	45.7	85.4
MV-RNN [Socher et al. 2012]	44.4	82.9
AdaMC-RNN [Dong et al. 2014]	45.8	87.1
AdaMC-RNTN [Dong et al. 2014]	46.7	88.5
DRNN [Irsoy and Cardie 2014]	49.8	86.6
TG-RNN (ours)	46.1(0.3)	86.2(0.3)
TE-RNN (ours)	47.8(0.3)	86.5(0.4)
TE-RNTN (ours)	48.8(0.4)	87.2(0.1)
CNN [Kim 2014]	48.0	88.1
DCNN [Kalchbrenner et al. 2014]	48.5	86.8
LSTM [Tai et al. 2015]	46.4(1.1)	84.9(0.6)
Bi-directional LSTM [Tai et al. 2015]	49.1(1.0)	87.5(0.5)
Tree-LSTM [Tai et al. 2015]	51.0(0.5)	88.0(0.3)
TW-LSTM (ours)	49.9(0.4)	87.4(0.4)
TW-LSTM+p (ours)	50.6(0.4)	87.7(0.1)
TE-LSTM (ours)	50.3(0.2)	87.8(0.5)
TE-LSTM+p (ours)	51.3(0.4)	88.2(0.5)
TW-LSTM+c (ours)	52.0(0.4)	89.2(0.3)
TW-LSTM+c,p (ours)	52.1(0.4)	89.5(0.3)
TE-LSTM+c (ours)	52.3(0.4)	89.4(0.4)
TE-LSTM+c,p (ours)	52.6(0.6)	89.6(0.4)

*The dimension of word/phrase embeddings (d) is 30 for RNN and RNTN, 25 for AdaMC-RNN, 15 for AdaMC-RNTN, and 300 for DRNN, all adopted from the references. For the CNN models, $d = 300$, obtained by word2vect, and for the LSTM models, $d = 300$, obtained by Glove.

is more effective than simply choosing composition functions according to the POS tag of a phrase.

—**TE-RNTN:** The fine-grained accuracy of TE-RNTN is improved by 3.2% compared with RNTN. TE-RNTN also outperforms the AdaMC-RNTN by 2.2% on the fine-grained classification task, and is better than CNN and DCNN. TE-RNTN is worse than DRNN, but the complexity of DRNN is much higher than TE-RNTN, which will be discussed in the next section.

—**TW-LSTM:** Tag Weighted LSTM outperforms RNN and CNN, and is comparable to DRNN. Although the POS tag has been largely abstracted (many different words may have the same tag); this shows that the POS tag can be effective in learning phrase/sentence representations.

—**TE-LSTM:** When combining representations of phrases and tags, the proposals outperform Tree-LSTM,[5] as can be seen from the $+c$ rows. Even without phrase representations (see those without $+c$ rows), these models are comparable to Tree-LSTM.

—Although the POS tag is effective, learning with or without phrase representations makes a difference; see TE-LSTM+c (52.3%) vs. TE-LSTM (50.3%), TW-LSTM+c

[5]Note that this model only uses phrase representations.

Table II. Classification Accuracy on the MR Dataset. Results of Some Baselines are Obtained by Our Own Implementation*

Method	Accuracy
RNN (implemented by ourselves)	76.2
RNTN (implemented by ourselves)	75.9
CNN [Kim 2014]	81.5
TG-RNN (ours)	76.4
TE-RNN (ours)	77.9
TE-RNTN (ours)	76.6
LSTM (implemented by ourselves)	77.4
Bidirectional LSTM (implemented by ourselves)	79.7
Tree-LSTM (implemented by ourselves)	80.7
TW-LSTM (ours)	80.2
TW-LSTM+p (ours)	80.6
TE-LSTM (ours)	80.7
TE-LSTM+p (ours)	80.1
TW-LSTM+c (ours)	82.0
TW-LSTM+c,p (ours)	81.9
TE-LSTM+c (ours)	81.6
TE-LSTM+c,p (ours)	82.2

*The dimension of word/phrase embeddings is 25 for RNN and RNTN, and 300 for CNN (word2vect vectors) and LSTM models (Glove vectors).

(52.0%) vs. TW-LSTM (49.9%), and TE-LSTM+c,p (52.6%) vs. TE-LSTM+p (51.3%). Results from rows with $+p$ show that considering child-parent association can improve results.

—All the LSTM models (except sequential LSTM and Bidirectional LSTM) outperform RNN, CNN, and other models.

As aforementioned, SST has phrase-level annotation for each node in a parsing tree, which is too expensive to afford. To demonstrate whether our models can work well with only sentence-level annotation, we conduct further experiments on the MR dataset. Therefore, we further evaluate our models on MR with 10-fold cross validation. Since most baselines shown in Table I did not perform experiments on this dataset, we implement some baselines by ourselves, but not all of them, due to the fact that most neural network models are too hard to reproduce the original results because of too many parameters to be tuned and many details not being reported.

From the results shown in Table II, we can see the following observations:

—The margin obtained by TG-RNN and TE-RNN/RNTN over their counterparts is not as significant as that on SST. We conjecture that this may be due to the fact that the supervision at the root node may not be easily back-propagated into the low-level nodes when the parsing trees are complex and deep.

—TW-LSTM performs almost as well as Tree-LSTM, indicating that the POS tag is fairly useful in sentence representation although the POS tag has been largely abstracted.

—When combining tag and word representations, our LSTM models are better than Tree-LSTM. This shows that our proposal is effective in the setting of only sentence-level annotation.

If we consider the results in Tables I and II together, we have the following discussions:

—When trained with phrase-level annotation, TE-LSTM is better than TW-LSTM as shown in Table I, while without phrase-level annotation, the two models perform

Table III. The Model Size of RNN/RNTN Models

Method	Model size	# of parameters	Accuracy on SST
RNN [Socher et al. 2011b]	$O(2 \times d^2)$	$\approx 1.8K$	43.2
RNTN [Socher et al. 2013b]	$O(4 \times d^3)$	$\approx 108K$	45.7
AdaMC-RNN [Dong et al. 2014]	$O(2 \times d^2 \times c)$	$\approx 18.7K$	45.8
AdaMC-RNTN [Dong et al. 2014]	$O(4 \times d^3 \times c)$	$\approx 202K$	46.7
DRNN [Irsoy and Cardie 2014]	$O(d \times h \times l + 2 \times h^2 \times l)$	$\approx 451K$	49.8
TG-RNN (ours)	$O(2 \times n_t \times d^2)$	$\approx 8.8K$	46.1
TE-RNN (ours)	$O(2 \times (d + d_e) \times d)$	$\approx 1.7K$	47.8
TE-RNTN (ours)	$O(4 \times (d + d_e)^2 \times d)$	$\approx 54K$	48.8

d: the dimension of word/phrase vectors. The optimal value is 30 for RNN and RNTN, 25 for AdaMC-RNN, 15 for AdaMC-RNTN, and 300 for DRNN, all adopted from the references.
$\mathbf{d_e}$: the dimension of tag embedding, whose optimal value is 8 for TE-RNN and 6 for TE-RNTN.
c: the number of composition functions (15 is the optimal setting) for AdaMC-RNN/RNTN.
l and **h**: the number of layers and the width for each layer, respectively, in DRNN, and the optimal values are $l = 4$ and $h = 174$, as reported in the references.
$\mathbf{n_t}$: the number of frequent tags. In TG-RNN, $n_t = 6$, corresponding to the number of composition functions (k).

very closely as shown in Table II. As discussed in Section 4.3.2, TE-LSTM is a better way for addressing infrequent tags since it adopts shared weight matrices for all tags and lower dimensional tag embedding vectors. We believe that this setting is more appropriate for those extremely infrequent tags.

—Modeling child-parent association is effective when phrase-level annotation is available. As shown in Table I, all models with the $+p$ option outperform their counterparts, however, the option seems to be not that effective when only sentence-level annotation is available (see Table II). This is in accordance with our design principle since more supervision is needed to model the relationship between child phrase and parent phrase.

To summarize this subsection, we can draw the following concluding statements:

—When combining the representations of tags and words, either RNN or LSTM models can be improved, demonstrating that the POS tag can benefit sentence representation for classification tasks.
—Although the POS tag has been largely abstracted (since many different words have the same POS tag), the models with only tag representation (for instance, TW-LSTMs) can produce comparable results to those using word representation.
—The preceding claims hold not only under the setting of phrase-level annotation, but also sentence-level annotation.

5.4. Complexity Analysis

To gain a deeper understanding of the models presented in Tables I and II, here we discuss the parameter scale of the neural network models since the prediction power of neural network models is highly correlated with the number of parameters.

The results are presented in Tables III and IV. Note that the parameters do not include the word embedding tables since they are almost the same for the same type of models. Therefore, we ignore this part but focus on the remainder parameters that are related to model complexity, termed *model size* (for simplicity, we also drop the size caused by all the bias vectors, a small part of the total size.). Also note that the optimal hyperparameters are different across models, and the optimal values are present at the bottom of each table. The optimal hyperparameters of baselines are adopted from the original references.

Table IV. The Model Size of CNN and LSTM Models

Method	Model size	# of parameters	Accuracy on SST
CNN [Kim 2014]	$O(\sum n_i \times f_i \times d)$	$\approx 360K$	48.0
DCNN [Kalchbrenner et al. 2014]	$O(\sum n_i \times f_i \times d)$	$\approx 360K$	48.5
LSTM [Tai et al. 2015]	$O(8 \times d^2)$	$\approx 720K$	46.4
Bidirectional LSTM [Tai et al. 2015]	$O(8 \times d^2)$	$\approx 720K$	49.1
Tree-LSTM [Tai et al. 2015]	$O(10 \times d^2)$	$\approx 900K$	51.0
TW-LSTM (ours)	$O(2 \times d^2 + 3 \times n_t \times d)$	$\approx 225K$	49.9
TW-LSTM+c (ours)	$O(10 \times d^2 + 3 \times n_t \times d)$	$\approx 945K$	52.0
TE-LSTM (ours)	$O(2 \times d^2 + n_t \times d_e + 3 \times d_e \times d)$	$\approx 199K$	50.3
TE-LSTM+c (ours)	$O(10 \times d^2 + n_t \times d_e + 3 \times d_e \times d)$	$\approx 919K$	52.3

d: the dimension of word/phrase vectors. For all CNN and LSTM models, $d = 300$.
$\mathbf{n_i}$ and $\mathbf{f_i}$: the width of convolution filters and the number of feature maps in CNN and DCNN, respectively. $n_i \in \{3, 4, 5\}$ and $f_i = 100$.
$\mathbf{d_e}$: the dimension of tag embedding, and $d_e = 20$ for TW-LSTMs and TE-LSTMs.
$\mathbf{n_t}$: the number of frequent tags. In TW-LSTM and TE-LSTM, $n_t = 50$.

Table V. The Distribution of Tags in the SST Dataset. The Top Six Frequency Tags Cover more than 95% Phrases. There are 28 Tags that AppearLess Than 300 Times, 18 Tags Less Than 100 Times, and 10 Tags Less Than 20 Times

Phrase tag	Frequency	Phrase tag	Frequency	Phrase tag	Frequency	Phrase tag	Frequency
NP	72,865	DT	15,403	NNP	6,371	VB	4,181
S	33,290	PP	15,121	CC	6,004	TO	2,961
VP	28,443	ADJP	9,335	RB	5,996	VBG	2,222
NN	20,650	.	8,217	SBAR	5,317	VBP	2,181
IN	15,697	VBZ	7,737	NNS	5,098	VBN	2,057
JJ	15,687	,	7,021	ADVP	4,604	PRP$	2,023

Results in Table III show that, our models, TG-RNN/TE-RNN, have much less parameters than RNTN and AdMC-RNN/RNTN, but have better performance. Although TE-RNTN is worse than DRNN, the parameters of DRNN are as large as nine times ours. This indicates that DRNN is much more complex, which requires much more data and time to train. As a matter of fact, our TE-RNTN only takes 20 epochs to converge.

The results in Table IV show that TW-LSTM+c and TE-LSTM+c have a comparable number of parameters with Tree-LSTM (about $900K$), but our performance is better. TW-LSTM and TE-LSTM have less parameters than CNNs but our performance is also better. TW-LSTM and TE-LSTM have less parameters than LSTM and Bidirectional LSTM (about $200K$ vs. $720K$) but have better performance. Further, although TW-LSTM and TE-LSTM are slightly worse than Tree-LSTM, Tree-LSTM has four to five times the parameters of our models.

5.5. Parameter Tuning

There are some important parameters in our proposed models. For TG-RNN, the number of composition functions (k) is critical, while for TE-RNN/RNTN/TE-LSTM, the dimension of tag embedding vectors (d_e) is quite important. For TW-LSTM and TE-LSTM, the weight of combining phrase vectors and tag vectors (α, as shown in Section 4.4) is a key hyperparameter. All the following experiments are performed on the Stanford Sentiment Treebank dataset.

First, we will study how the number of composition functions will influence the performance of **TG-RNN**. Let us start from the corpus statistic. As shown in Table V, the corpus contains 314,810 phrases but the distribution of phrase tags is extremely imbalanced. For example, the phrase tag *"NP"* appears 72,865 times, while *"NAC"* appears only 10 times. Hence, it is impossible to learn a composition function for those infrequent phrase tags. A feasible way is to choose the top k frequent tags, each of

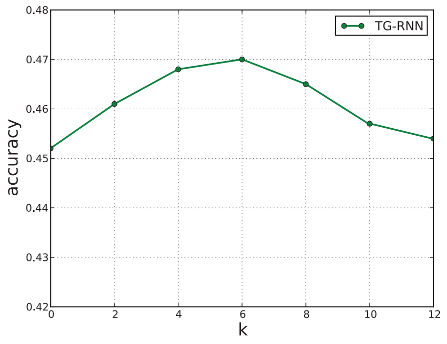

Fig. 7. The accuracy for TG-RNN with different k (the number of composition functions). The optimal value is 6.

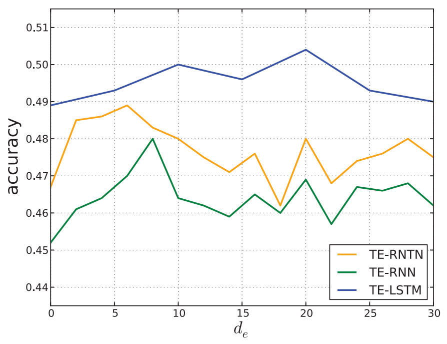

Fig. 8. The accuracy for TE-RNN/RNTN and TE-LSTM with different dimensions of d_e. The optimal value is 8 for TE-RNN, 6 for TE-RNTN, and 20 for TE-LSTM. Note that the step size for TE-LSTM is 5 and that for TE-RNN/RNTN is 2.

which corresponds to a unique composition function, while all the other phrase tags share a same function. We experiment with different k for TG-RNN. The accuracy is shown in Figure 7. Our model obtains the best performance when k is 6, where the top six frequency tags cover more than 95% phrases.

Second, we investigate how the dimension of tag vectors will affect the performance of **TE-RNN/RNTN** and **TE-LSTM**. In the corpus, we have 70 types of tags for leaf nodes (words) and interior nodes (phrases). Infrequent tags whose frequency is less than 1,000 are ignored. There are 30 tags left and we learn an embedding for each of these frequent tags. We vary the dimension of the embedding d_e from 0 to 30.[6] Figure 8 shows the accuracy for TE-RNN/RNTN and TE-LSTM with different dimensions of d_e. Our model obtains the best performance when d_e is 8 for TE-RNN, 6 for TE-RNTN, and 20 for TE-LSTM. The results show that too small dimensions may not be sufficient to encode the syntactic information of tags and too large dimensions damage the performance.

Third, we justify how the hyperparameter α will affect the performance of **TE-LSTM**. As demonstrated in Table I, for TE-LSTM, superior performance can be obtained when combining the representations of phrases and tags. The α parameter controls the balance of syntactic information and phrase information. We experiment with the setting of TE-LSTM+c,p, varying α from 0 to 1.0 with a step of 0.25. Results in Table VI show that the optimal value is close to 0.5. Note that when $\alpha = 0.0$ the model reduces to Tree-LSTM, and when $\alpha = 1.0$ the model reduces to TE-LSTM+p.

[6]Since the total number of tags is 70, this range (i.e., (0,30]) will be sufficient to find the optimal value.

Table VI. Classification Accuracy on the Stanford Sentiment Tree bank Dataset for Different α

α	Fine-grained	Pos./Neg.
0.0	51.0(0.5)	88.0(0.3)
0.25	51.2(1.1)	89.3(0.3)
0.50	52.6(0.6)	89.6(0.4)
0.75	52.0(0.8)	89.5(0.3)
1.0	51.3(0.4)	88.2(0.5)

Table VII. The Top Five Nearest Neighboring Tags for a Query Tag

Query Tag	Model	Most Similar Tags
JJ (Adjective)	TE-RNN	ADJP VBZ DT NP RB
	TE-LSTM	NNP ADJP VBZ RB VP
VBZ (Verb, third person singular present)	TE-RNN	NP ADJP JJ PP DT
	TE-LSTM	JJ ADJP RB PP IN
DT (Determiner)	TE-RNN	PP RB NP VB JJ
	TE-LSTM	PP ADJP NP CC VB
NN (Noun phrase)	TE-RNN	VP RB NP VBZ JJ
	TE-LSTM	RB VP IN NP VB
.	TE-RNN	, : DT PP RB
	TE-LSTM	, DT JJ IN :

ADJP: adjective phrase; JJ: adjective; RB: adverb.
VB: verb, base form; VBZ: verb, third person singular present; VP: verb phrase.
NN: noun, singular/mass; NP: noun phrase; NNP: proper noun, singular.
DT: determiner; PP: prepositional phrase; IN: preposition/subordinating conjunction; CC: coordinating conjunction.

5.6. Analysis on the POS Tags

We have shown the success of encoding the part-of-speech tag in neural networks. In this section, we will further reveal how the tags will play a role in these neural network models.

In order to justify whether tag vectors obtained from tag embedded models are meaningful, we inspect the similarity between tags. The experiment is conducted with TE-RNN and TE-LSTM, respectively. For each tag vector, we find the top five nearest neighbors based on Euclidean distance, as exemplified in Table VII. Adjectives and adverbs are of significant importance in sentiment analysis. Although "*JJ*" is a tag for words and "*ADJP*" for phrases, they are close in vector space, since they play a similar role in sentences. *VP (VBZ, VP, etc.)* are close to *NP (NN, NNP, etc.)*, probably due to the fact that *VP* are usually adjacent to *NP* in composition and they cooccur very frequently. What is more interesting is that the nearest neighbor of "*Dot*" is "*Colon*," because both of them are punctuation marks and play a similar role in composition.

If we compare the most similar sets between TE-RNN and TE-LSTM, the overlap in the pairs shows that similar tag topologies can be obtained although the rank is different (note that rank is sensitive to the distance score; a small score difference may cause a big change in rank). For instance, for all query tags, the two models have three tags in common. Note that it is hard to generate perfect topologies for these models since our optimization goal is the classification performance instead of the tag representation, while tag embeddings are just a side product of the goal. Generally speaking, these results show that tags of similar syntactic role or sentiment informativeness are close in vector space, demonstrating that the semantics of embedded tag vectors is effective.

Table VIII. The Importance of Tags for Semantic Composition in TW-LSTM and TE-LSTM

Tag	TW-LSTM	TE-LSTM
ADJP	0.742	0.881
VP	0.750	0.819
JJ	0.674	0.776
NP	0.580	0.698
VBZ	0.463	0.593
NN	0.402	0.570
CC	0.368	0.445
IN	0.307	0.310
DT	0.246	0.270

To further reveal how POS tags influence the process of semantic composition, we attempt to measure the importance of tags learned by TW-LSTM and TE-LSTM. Since the forget gate is the most important parameter [Greff et al. 2015], we take the average of all dimensions of the output of the forget gates as the value to quantify the importance of a tag. Specifically, for TW-LSTM, the value is $\frac{1}{d}\sum_{i=1}^{d}\sigma(W_f[t])[i]$ for tag t. For TE-LSTM, the value is $\frac{1}{d}\sum_{i=1}^{d}\sigma(U_f * E[t])[i]$. The larger the value, the more important the tag.

The results are shown in Table VIII. As can be seen, the most important tags are *ADJP, VP, and JJ* in both models where such words usually carry strong sentiment information, while *DT and IN* are much less influential for learning sentiment-favorable representation. This is in accordance with our intuition that words with such tags as *ADJP, VP, and JJ* are more possible to carry sentiment expressions.

6. DISCUSSIONS

We have shown that the part-of-speech tag can benefit the learning of sentiment favorable representations and thus successfully enhance the performance of sentiment classification. Generally speaking, the prediction power of neural networks is highly correlated with the number of parameters, where this claim is also justified in this article. However, this is not to say that more parameters must have better performance. Our work reveals *two key factors: how much useful knowledge has been put into the networks and how the models are properly designed*.

Recall the results in Table III: our TE-RNN model has much less parameters than RNTN (1.7K vs. 108K) but better performance (47.8% vs. 45.7%). This is similar for TE-RNTN and RNTN (parameter: 54K vs. 108K; performance: 48.8% vs. 45.7%). This shows that syntactic knowledge such as the part-of-speech tag is very effective for semantic composition. If we compare TE-RNN with TG-RNN (parameter: 1.7K vs. 8.8K; performance: 47.5% vs. 46.1%), where TE-RNN learns tag embeddings while TG-RNN chooses tag-specific composition functions, we can see that a proper way to encode such knowledge is also important.

The results in Table IV also demonstrate that the factors are indeed influential. Unlike Tree-LSTM that employs word embeddings to control gates, TE-LSTM only employs part-of-speech tags and produces comparable results with much less parameters. When combined with word embeddings, TE-LSTM+c has better performance than Tree-LSTM, with comparable parameters. Although TE-LSTM and TW-LSTM have close parameters and performance, we have discussed that TE-LSTM is a better way to encode syntactic knowledge.

In order to make neural networks perform competitively, we have shown that encoding useful knowledge can be of much benefit, and that whether the parameters are properly designed is also crucial to performance improvement.

7. CONCLUSIONS

In this article, we have presented that syntactic knowledge can be encoded in neural networks for dealing with semantic composition and experiments show that it is effective to learn phrase/sentence representation with such knowledge. More specifically, we have successfully encoded part-of-speech tags in RNN, RNTN, and LTSM networks. In the proposed (TG-RNN, TE-RNN/RNTN, TW-LSTM, and TE-LSTM), the part-of-speech tag can be used to either choose composition functions in neural networks, or learn tag embeddings and then employ those tag embeddings to control components of the neural networks. The evaluation on two benchmark datasets, SST and MR, has shown that our proposals outperform the corresponding reference baselines. The results show that (1) comparable results can be obtained when only considering the part-of-speech tag, although POS tags have been largely abstracted in comparison to words, and (2) that superior results can be obtained when combining the representations of word/phrase and part-of-speech tag.

In most conventional models, the composition process is usually optimized with a task-specific loss and the supervision from the output layer is back-propagated into the lower layers. This can be trained end-to-end, without a necessity to attend the details of the blackbox, while we show here that better performance can be obtained if we can properly model the intermediate details (such as part-of-speech tags and relationships between words or phrases) of the composition process of representation learning. The goal of this article is to demonstrate that such intermediate details should be well explored for designing an appropriate neural network. We have successfully shown that such efforts are worth making.

ACKNOWLEDGMENTS

We would like to thank Mr. Bo Tian, Mr. Hao Zhou, and Ms. Liangqi Liu for their support to this work.

REFERENCES

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: New features and speed improvements. Deep Learning and Unsupervised Feature Learning. NIPS 2012 Workshop.

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

Tao Chen, Ruifeng Xu, Yulan He, and Xuan Wang. 2017. Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN. *Expert Systems with Applications* 72 (2017), 221–230.

Yanqing Chen and Steven Skiena. 2014. Building sentiment lexicons for all major languages. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 383–389. DOI:http://dx.doi.org/10.3115/v1/P14-2063

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555* (2014).

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, 160–167.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2014. Adaptive multi-compositionality for recursive neural models with applications to sentiment analysis. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*. AAAI.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural 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)*. Association for Computational Linguistics, 260–269.

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

Peter W. Foltz, Walter Kintsch, and Thomas K. Landauer. 1998. The measurement of textual coherence with latent semantic analysis. *Discourse Processes* 25, 2–3 (1998), 285–307.

Alan Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *Proceedings of the 2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 273–278.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *arXiv:1503.04069* (2015).

Karl Moritz Hermann and Phil Blunsom. 2013. The role of syntax in vector space models of compositional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computer Linguistics, 894–904.

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

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 168–177.

Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. 2096–2104.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14)*. Association for Computer Linguistics, 655–665.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*. Association for Computational Linguistics, 1746–1751.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics—Volume 1*. Association for Computational Linguistics, 423–430.

Thomas K. Landauer and Susan T. Dumais. 1997. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review* 104, 2 (1997), 211.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding CNNs for text: Non-linear, non-consecutive convolutions. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association of Computational Linguistics, 1565–1575.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. 2015. A hierarchical neural autoencoder for paragraphs and documents. *arXiv:1506.01057* (2015).

Biao Liu, Minlie Huang, Jiashen Sun, and Xuan Zhu. 2015. Incorporating domain and sentiment supervision in representation learning for domain adaptation. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 1277–1283.

Tomas Mikolov. 2012. Statistical language models based on neural networks. Presentation at Google, Mountain View, April 2, 2012.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR* (2013).

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS'13)*. 3111–3119.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of ACL*. 236–244.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 115–124.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval* 2, 1–2 (2008), 1–135.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP'14)* 12 (2014), 1532–1543.

Qiao Qian, Bo Tian, Minlie Huang, Yang Liu, Xuan Zhu, and Xiaoyan Zhu. 2015. Learning tag embeddings and tag-specific composition functions in recursive neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Vol. 1. 1365–1374.

Sebastian Rudolph and Eugenie Giesbrecht. 2010. Compositional matrix-space models of language. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'11)*. Association for Computer Linguistics, 907–916.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323 (1986), 533–536.

Aliaksei Severyn and Alessandro Moschitti. 2015a. On the automatic learning of sentiment lexicons. In *Proceedings of the NAACL HLT 2015 Conference of the North American Chapter of the Association for Computational Linguistics*. 1397–1402.

Aliaksei Severyn and Alessandro Moschitti. 2015b. Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM SIGIR, 959–962.

Paul Smolensky. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence* 46, 1 (1990), 159–216.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013a. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computer Linguistics, 455–465.

Richard Socher, Eric H. Huang, Jeffrey Pennin, Christopher D. Manning, and Andrew Y. Ng. 2011a. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11)*. 801–809.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'12)*. Association for Computational Linguistics, 1201–1211.

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 Conference on Empirical Methods in Natural Language Processing (EMNLP'11)*. Association for Computational Linguistics, 151–161.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*. Association for Computational Linguistics, 1631–1642.

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

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv:1503.00075* (2015).

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 1422–1432.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1555–1565.

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

Tin Duy Vo and Yue Zhang. 2016. Don't count, predict! an automatic approach to learning sentiment lexicons for short text. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 219–224. DOI:http://dx.doi.org/10.18653/v1/P16-2036

Sida I. Wang and Christopher D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers—Volume 2 (ACL'12)*. Association for Computational Linguistics, 90–94.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT'05)*. 347–354.

Ainur Yessenalina and Claire Cardie. 2011. Compositional matrix-space models for sentiment analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'15)*. Association for Computer Linguistics, 172–182.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over tree structures. *arXiv:1503.04881*.

Received June 2016; revised October 2016; accepted December 2016