Combining Convolution and Recursive Neural Networks for Sentiment Analysis

Vinh D. Van
Faculty of Information Technology
VNUHCM - University of Sciences
Nguyen Van Cu
Ho Chi Minh, Vietnam
vdvinh@apcs.vn

Thien Thai
Faculty of Information Technology
VNUHCM - University of Sciences
Nguyen Van Cu
Ho Chi Minh, Vietnam
tthien@apcs.vn

Minh-Quoc Nghiem
Faculty of Information Technology
VNUHCM - University of Sciences
Nguyen Van Cu
Ho Chi Minh, Vietnam
nqminh@fit.hcmus.edu.vn

ABSTRACT
This paper addresses the problem of sentence-level sentiment analysis. In recent years, Convolution and Recursive Neural Networks have been proven to be effective network architecture for sentence-level sentiment analysis. Nevertheless, each of them has its own potential drawbacks. For alleviating their weaknesses, we combined Convolution and Recursive Neural Networks into a new network architecture. In addition, we employed transfer learning from a large document-level labeled sentiment dataset to improve the word embedding in our models. The resulting models outperform all recent Convolution and Recursive Neural Networks. Beyond that, our models achieve comparable performance with state-of-the-art systems on Stanford Sentiment Treebank.

CCS CONCEPTS
•Information systems → Sentiment analysis; •Computing methodologies → Transfer learning;

KEYWORDS
Sentence-level Sentiment Analysis, Convolution Neural Network, Recursive Neural Network, Transfer Learning

ACM Reference format:
Vinh D. Van, Thien Thai, and Minh-Quoc Nghiem. 2017. Combining Convolution and Recursive Neural Networks for Sentiment Analysis. In Proceedings of Eighth International Symposium on Information and Communication Technology; Nha Trang City; Viet Nam, December 7–8, 2017 (SoICT ’17), 7 pages.
DOI: 10.1145/3155133.3155158

1 INTRODUCTION
In recent years, thanks to the dramatic growth of social media, customers’ opinions are expressed in the highest speed and volume ever recorded in history. It is inefficient to read, analyze, or even collect such a large amount of data manually. Sentiment analysis offers a way to collect and process public opinion automatically.

Sentence-level sentiment analysis is used to determine whether an opinion about a specific product, event, or organization is positive or negative. Formally, given document d, the main objective of sentiment analysis is to extract the following quintuple [t] :

\[(e_t, a_{ij}, s_{ijk}, h_k, t_l)\]

Where:
- \(e_t\): entity \(t\) (entity extraction and categorization)
- \(a_{ij}\): aspect \(j\) of entity \(i\) (entity extraction and categorization)
- \(h_k\): holder \(k\) (opinion holder extraction and categorization)
- \(t_l\): time \(l\) (time extraction and standardization)
- \(s_{ijk}\): opinion of holder \(k\) about aspect \(j\) of entity \(i\) at time \(l\) (aspect sentiment classification)

Sentence-level sentiment analysis is to determine whether a sentence expresses positive or negative sentiment. This level of analysis assumes that every sentence contains one opinion toward an entity (e.g., a single movie) [t].

In this paper, we explore two ideas: “Combining Convolution and Recursive Neural Networks” (the main idea) and “Transfer Learning From Large Review Dataset” (the supporting idea).

Combining Convolution and Recursive Neural Networks. Convolution (CNN) and Recursive Neural Networks (RNN) have been proven to be effective network architectures for sentence-level sentiment analysis. Nevertheless, each of them has drawbacks (Section 2). For alleviating their weaknesses, we combined CNN and RecNN into a new network architecture (Section 3) which is able to outperform both CNN and RecNN (Section 5.3) on Stanford Sentiment Treebank (Section 5.1). This approach is closely related to the paper of Wang et al. [t] which investigates the of combination CNN and Recurrent Neural Networks (RNN).

Transfer Learning From Large Review Dataset. One obstacle of solving sentence-level sentiment analysis is the lack of labeled data which potentially causes many drawbacks, one of which is overfitting word embedding (Section 4.1). Since most opinions are expressed in form the of a documents (i.e., multiple sentences), sentence-level labeled dataset required more work to produce. Until now, the largest dataset for sentence-level sentiment analysis is Stanford Sentiment Treebank which only contains 11,855 sentences. The number is insignificant compared to Amazon Review dataset which has 83.68 million reviews. We utilized Amazon Review dataset
2 RELATED WORK AND MOTIVATION

In recent years, sentiment analysis has enjoyed dramatic improvements by applying different variations of CNN, RNN and RecNN.

Although originally invented for Computer Vision, CNNs have been proven to be effective models for document classification. Nevertheless, for composing fixed-length representation vector of documents, several types of max pooling layers were employed [? ? ? ?]. Although max pooling layer largely simplified the network (which is good for preventing over-fit), this solution have a clear disadvantage. By down-sampling a feature map, although the resulted vector still contains information about the existence of a feature, it is likely that the information about the position or order of the feature is lost. This can be harmful because the order of words and phrases is important for understanding the sentiments of a sentence.

RNN and RecNN have been especially designed for dealing with variable-length sequential input. They have been successfully applied to a variety of NLP tasks, include: speech recognition [? ? ], sentiment analysis [? ? ], text summarization [? ? ], machine translation [? ? ], language modeling [? ? ].

Wang et al. [? ] successfully combined CNN with RNN for Sentiment Analysis by utilizing CNN for capturing phrase-level features and RNN for composing these features. Their network architecture was similar to that of Yoon Kim et al. [? ], the main different was the usage of RNN for replacing the max-over-time pooling layer. Nevertheless, it was able to significantly outperform the state-of-the-art system at the time [? ].

In this paper, we adopting the idea of Wang et al. [? ] by combining CNN with RecNN. The main different is that the RNN module is replaced by a RecNN module. RecNN have several advantages over RNN:

• In case the input sequence belongs to a recursively defined language, given only a small subset of the data with limited length sentences, tree structures model have better ability to generalize comparing to sequential ones. However, when the limited length of sentences in the training data is increased, the advantage of tree over sequential models decreases fast [? ].

• Tree can break down complicated sentences into simpler phrases which are easier for generalization [? ] [? ].

• Some features which are far apart when a sentence is presented as sequence become closer when it is presented as a tree [? ].

For our RecNN module, Tree-LSTMs [? ] was employed. The core idea behind the design of Tree-LSTMs are to generalize the

LSTM for tree-structured inputs. Tree-LSTMs were able to achieve state-of-the-art performance on two tasks: predicting the semantic relatedness of two sentences (SemEval 2014, Task 1 [? ]) and sentiment classification (Stanford Sentiment Treebank [? ]). Nevertheless, Tree-LSTMs also have some drawbacks, including:

• Sentences can be wrongly parsed, especially when comments are expressed in informal language. The performance of the system depends on the parser being used.

• At their leaf-module, Tree-LSTMs have only a simple logistic regression layer on top of the vector presentation of a single word at that position. The simple leaf-module of Tree-LSTMs might be its weakness when dealing with the problem of words ambiguity. This weakness becomes even more severe when the sentence is wrongly parsed.

We hypothesized that the convolution layer helps Constituency Tree-LSTM to mitigate the problem of lacking local context and words ambiguity at leaf nodes.

3 COMBINING CONVOLUTION AND RECURSIVE NEURAL NETWORKS

Our model architecture is shown in Fig. 1. The model has three modules: word embedding layer, convolution layer, and Constituency Tree-LSTM.

Figure 1: CNN-Tree-LSTM

3.1 Word Embedding Layer

Suppose that \( Z = \{e_0, e_1, \ldots, e_m\} \) is a set input channels with each channel uses a different word embedding. The first word in a sentence is indexed as word-0th. If the sentence is padded with dummy words, left padded dummy words are indexed by negative integers.

For any word embedding \( e \), the vector presentation of the word\(-i\)th is denoted as \( w_i(e) \in \mathbb{R}^{d_e} \). The vector presentation of word\(-i\)th through the set of input channels \( Z \) is expressed as follow:

\[
x_i = w_i(e_0) \odot w_i(e_1) \odot \ldots \odot w_i(e_m)
\]

In Eq.(1), \( \odot \) is concatenation operator which results in the vector \( x_i \in \mathbb{R}^d \) with \( d = \sum_{e \in Z} d_e \). Any sequence of words starting from

---

*Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download) publicly available at https://nlp.stanford.edu/projects/glove/
word-th word to word-th is present as the following matrix:

\[ X_{i:j} = x_i \oplus x_{i+1} \oplus \ldots \oplus x_j \]  

(2)

In Eq(2), \( \oplus \) is concatenation operator which results in the matrix \( X_{i:j} \in \mathbb{R}^{d \times (j-i+1)} \).

3.2 Convolution Layer

Given that \( F \) is the set of all filters of the convolution layer, for any filter \( v \in F \) which has window size \( l \) and set of parameters \( \theta(v) = \{ W(v), b(v) \mid W(v) \in \mathbb{R}^{d \times l}, b(v) \in \mathbb{R} \} \), filter \( v \) is applied on any sequence of word-th to word-(i+1)-th through the following equation:

\[ c_j^{(v)} = f(W(v) \times x_{i:i+l-1} + b(v)) \]  

(3)

In Eq(3), operator \( \oplus \) is the Hadamard product. \( b \in \mathbb{R} \) is bias term. \( f \) is an activation function. For indexing, \( j = i+x \) with \( x \in \mathbb{N} \) and \( 0 \leq x < l \). If half-padding policy is employed then \( j = i + \left\lfloor \frac{l}{2} \right\rfloor \).

By slicing the filter \( v \) through the sentence (i.e. applying the filter \( v \) on different sequences of length \( l \) along the sentence) we get vector \( c^{(v)} = [c_1^{(v)}, c_2^{(v)}, \ldots] \) which is a feature map of the sentence \( s \). The length of the feature map \( c^{(v)} \) depends on the length of the input sentence, the way which filter \( v \) was slid through the sentence and its window size \( l \).

In our model, all filters in \( F \) are restricted to have odd window sizes and being applied on the sentence according to half padding, unit strides policy. These conditions guarantee that the lengths of all feature maps produced from a sentence are equal to the number of words in that sentence. Suppose the size of the set of filters \( F \) is \( m \), all the feature maps of a sentence of length \( n \) produced by the set of filters is concatenated into one matrix \( P \in \mathbb{R}^{m \times n} \).

The \( i \)-th column vector of \( P \) are treated as the vector representation of the \( i \)-th word in the sentence.

3.3 Constituency Tree-LSTM

Let \( d \) be the size of the input vectors, \( r \) be the size of the memory cell and \( z \) be the number of sentiment classes.

**Leaf module.** Given any input vector \( x \in \mathbb{R}^d \), the calculation steps inside the leaf module are expressed as follow:

\[ o = \sigma (W^{(o)} x + d^{(o)}) \]  

(4)

\[ c = W^{(c)} x + d^{(c)} \]  

(5)

\[ h = o \odot \tanh(c) \]  

(6)

In this module, \( W^{(o)}, W^{(c)} \in \mathbb{R}^{r \times d} \) and \( d^{(o)}, d^{(c)} \in \mathbb{R}^r \).

**Composer module.** Given the input vectors \( h_l \) and \( c_l \) from the left child node, \( h_r \) and \( c_r \) from the right child node, the calculation steps inside the composer module are expressed as follow:

\[ i = \sigma (U^{(i)} h_l + U^{(i)} h_r + b^{(i)}) \]  

(7)

\[ f_l = \sigma (U^{(f)} h_l + U^{(f)} h_r + b^{(f)}) \]  

(8)

\[ f_r = \sigma (U^{(f)} h_l + U^{(f)} h_r + b^{(f)}) \]  

(9)

\[ o = \sigma (U^{(o)} h_l + U^{(o)} h_r + b^{(o)}) \]  

(10)

\[ u = \tanh (U^{(u)} h_l + U^{(u)} h_r + b^{(u)}) \]  

(11)

\[ c = i \odot u + f_l \odot c_l + f_r \odot c_r \]  

(12)

\[ h = o \odot \tanh(c) \]  

(13)

For any \( j \in \{ i, l, r, o, u \} \) and \( x \in \{ l, r \} \), \( U^{(j)} \in \mathbb{R}^{r \times r} \) and \( b^{(j)} \in \mathbb{R}^r \).

**Output module.** Denoting sequence of words spanned by a subtree rooted at node \( j \) as \( \{ x_j \} \), the prediction at node \( j \) is computed by the output module as follow:

\[ p_{\theta}(y \mid \{ x_j \}) = \text{softmax}(W^{(s)} h_j + b^{(s)}) \]  

(14)

\[ \hat{y}_j = \text{argmax}_y p_{\theta}(y \mid \{ x_j \}) \]  

(15)

With \( W^{(s)} \in \mathbb{R}^{r \times r} \) and \( b^{(s)} \in \mathbb{R}^r \).

Composing sentence. Given any sentence, its parse tree and set of vectors representation of each word in the sentence, Constituency Tree-LSTM is applied on the sentence as follow:

- At leaf node, leaf module takes input from previous layer (convolution layer) output of corresponding word.
- At non-leaf node, composer module is applied recursively.
- After that, at every node \( j \), output module takes input from leaf module or composer module to predict sentiment of sub-tree root \( j \).

Apart from using Constituency Tree-LSTM to combine the set of vectors representation of each word in a sentence produced by the convolution layer, we also used LSTM to replace Constituency Tree-LSTM in several experiments.

3.4 LSTM

Denoting input sequence as \( I = \{ i_0, \ldots, i_n \} \), \( \forall t, i_t \in \mathbb{R}^n \), LSTM unit is expressed as the following recursive formula:

\[ w_t = \sigma (W^{(w)} i_t + U^{(w)} h_{t-1} + b^{(w)}) \]  

(16)

\[ f_t = \sigma (W^{(f)} i_t + U^{(f)} h_{t-1} + b^{(f)}) \]  

(17)

\[ o_t = \sigma (W^{(o)} i_t + U^{(o)} h_{t-1} + b^{(o)}) \]  

(18)

\[ u_t = \tanh (W^{(u)} i_t + U^{(u)} h_{t-1} + b^{(u)}) \]  

(19)

\[ c_t = r_t \odot u_t + f_t \odot c_{t-1} \]  

(20)

\[ h_t = o_t \odot \tanh(c_t) \]  

(21)

The operation \( \odot \) denotes the element-wise vector product. Traditionally, \( w_t, f_t \) and \( o_t \) are called input/write gate, forget/deallocate gate and output/read gate respectively and \( c_t \) is called memory cell. Intuitively, we can interpret how the network works as follow:

- \( h_{t-1} \) can be viewed as a short-term memory of the network.
4 TRANSFER LEARNING FROM LARGE REVIEW DATASET

4.1 Motivation
There are many words (e.g. “B-rated”, “Batman”, “Nolan”, “cartoon-like”) which rarely appear in regular documents but more often in movie reviews. These words might not appear or not have good vector representations in the pre-trained Glove Common Crawl. Additionally, the ways people use words in movie reviews might be different from their use in general documents. For example, the vector presentations of “sympathy” and “disappointed”, or “boom” and “insult” are very close to each other in Glove Common Crawl but if we have to predict the sentiment of a movie comment which has one of these words. Ideally, they should be distinctive because “disappointed” and “insult” are likely to express negative sentiment, while “boom” and “sympathy” are likely to express positive sentiment. One solution is to update word embedding during the training process so that we can have better word embedding for specific tasks and domains. Nevertheless, this method can harm generalization by updating only words which appear in the training set, and thus over-fitting occurs.

Moreover, we observed that the available amount of document-level labeled sentiment data (e.g., Amazon Reviews dataset) has 83.68 million reviews) is gigantic compared to the amount of sentence or phrase-level sentiment data (e.g., Stanford Sentiment Treebank). This might help our models achieving better generalization when training on small sentence-level sentiment dataset (e.g. Stanford Sentiment Treebank). We have five steps to preprocess Amazon dataset for training a new word embedding using Glove method:

1. We only used some partitions of Amazon Reviews dataset which includes: “Amazon Movies and TV” (7,850,072 reviews) and “Books” (22,507,155 reviews).
2. All the reviews were grouped by product-ID (“asin” keyword in the JSON schema of the dataset).
3. In each product-ID group, the reviews were sorted increasingly by their ratings ("overall" keyword in the JSON schema of the dataset).
4. All the reviews were dumped into a plain text file.
5. The text file produced from the previous step was tokenized using Stanford Tokenizer.

There is no definition of end-of-document in Glove model, which means words which appear in the beginning part of a document will be included in the context of words in the last part of the previous document which leads to noise in training data.

We set \( x_{max} = 100 \), vector size to 300, windows size to 20 and the minimum number of word occurrences to be included in the vocabulary to 5. The training process took the plain text file from preprocessing steps as input. In total, the corpus contains 4.7 billion tokens. After the training process, the resulting word embedding has vocabulary size of 1,734,244. We named this new word embedding Glove Amazon.

5 EXPERIMENTS

5.1 Datasets
We evaluated our model on Standford Sentiment Treebank dataset. Standford Sentiment Treebank contains total 11,835 sentences. We used provided train/dev/test sets which contain 8544, 1101 and 2210 sentences, respectively. In this dataset, every sentence was parsed using Stanford (constituency) parser into multiple phrases. There are a total of 215,154 labeled phrases in the whole dataset. Thus, every sentence in the corpus has a fully labeled parse tree. For training a Recurrent Neural Network, any phrase spanned by a labeled node is treated as a training sample.

Fine-grained setting. We partitioned sentiment labels into 5 classes: “Positive”, “Somewhat Positive”, “Neutral”, “Somewhat Negative” and “Negative”.

Binary setting. We removed all “Neutral” sentences. For the remaining 6920/872/1821 sentences in train/dev/test sets, we merged “Somewhat Positive” into “Positive” and “Somewhat Negative” into “Negative”.

5.2 Setups

5.2.1 Experiment Descriptions. We did experiment with Glove Amazon and different variations of our model.

CNN-Tree-LSTM. Our basic model with only one input channel. We initialized word representations with the standard Glove vectors.

CNN-Tree-LSTM (Glove Amazon) CNN-Tree-LSTM with word vectors initialized from Glove Amazon.

\(^1\)http://jmcauley.ucsd.edu/data/amazon/

\(^2\)Publicly available on Github https://github.com/stanfordnlp/GloVe
2-channel CNN-Tree-LSTM: CNN-Tree-LSTM with two input channels from two different word embedding matrices, which are initialized from Glove Common Craw and Glove Amazon, respectively.

CNN-LSTM: Similar to CNN-Tree-LSTM. However, we replaced Constituency Tree-LSTM module by an LSTM unit. CNN-LSTM has only one input channel initialized from the standard Glove vectors.

CNN-LSTM (Glove Amazon): CNN-LSTM with word vectors initialized from Glove Amazon.

2-channel CNN-LSTM: A CNN-LSTM model with two input channels at CNN layers. The CNN layers are similar to CNN layers in 2-channel CNN-Tree-LSTM model.

Constituency Tree-LSTM (Glove Amazon): Glove Amazon is used to replace the standard Glove vector for initializing word embedding layer of Constituency Tree-LSTM. Apart from that, the whole training process and hyper-parameters of Constituency Tree-LSTM are kept unchanged. The case of Tree-LSTM using both Glove Amazon, standard Glove were not experimented on because we have found no reliable way to extend the original Tree-LSTM for multichannel input.

Since the cost functions of neural networks are nonconvex and algorithms used to train neural networks are only able to find local optimum, different runs of one model can converge in various local optimums depend on the initialized parameters of the models. Base on the evaluating method used by Tai et al. [1], we evaluated the above models based on mean, standard deviation of 5 runs. In addition, the maximum accuracy among 5 runs is also reported.

We index all our experimented models along with their number of parameters in Table 1.

Table 1: Size of memory cell $r$ and number of trainable parameters $|\theta|$ of our models.

| Model                      | $r$  | $|\theta|$ |
|----------------------------|------|-----------|
| CNN-LSTM                   | 168  | 489,347   |
| CNN-Tree-LSTM              | 150  | 482,153   |
| 2-channel CNN-LSTM         | 168  | 729,347   |
| 2-channel CNN-Tree-LSTM    | 150  | 722,153   |

5.2.2 Hyper-parameters and Training. We trained models on training set and tuned hyper-parameters on development set of Stanford. Our models were trained using AdaGrad [?] with learning rate of $\{0.1, 0.05, 0.01\}$, L2 regularization strength of $\{1e^{-3}, 1e^{-4}, 1e^{-5}\}$ and batch size of 25. Word vectors are updated with learning rate $\alpha$ of $\{0.1, 0.05, 0.01\}$. In convolution layers, treated number of filter and filter size as hyper-parameters. We found that 100 filters of size 3 and 100 filters of size 5 yield better results compared to single filters size or the number of filters larger than 200. We regularized the convolution layers with input dropout rate of 0.5, and output dropout rate of 0.2. At output layer, we regularize with dropout rate of 0.5. Training with Adagrad’s learning rate of 0.01 and word vectors’ learning rate 0.1 give the best result.

Our models were trained for 60 epochs.

5.3 Results

Experiment results are summaries in Table 2. Table 2 contains two parts. Block A to E contain all baselines model. Block F contains all models we proposed and evaluated.

Block A contains convolution neural networks. CNN-nonstatic and CNN-multichannel [?] are single layer CNN. DCNN [?] and MVCNN [?] are multilayer CNNs, with MVCNN is a large model which has 2 layers and 5 input channels.

Block B contains recurrent neural network models and their variations. All the models in this Block B process sentences sequentially. DMN [?] is a sophisticated model used GRU with attention mechanism and episodic memory. Byte mLSTM [?] is the state-of-the-art system on binary setting.

Block C contains models which belong to the family of Recursive Neural Networks (tree-structured models). RNTN [?] is the first recursive neural network to successfully apply on sentence-level sentiment analysis (Stanford Sentiment Treebank). DRNN [?] is a multilayered extension of RNTN. TE-RNTN is an extension of RNTN which utilizes the local syntactic information at each node of a sentence’s parse tree. Constituency Tree-LSTM Ensemble [?] is an ensemble of 30 Constituency Tree-LSTMs. This model is the state-of-the-art system on the fine-grained setting of Stanford Sentiment Treebank.

Block D contains transfer learning methods, which utilized a large amount of data other than Stanford Sentiment Treebank. GICF [?] learns to classify sentiments of sentences (in Stanford Sentiment Treebank) using only document-level sentiment labels training dataset. Paragraph-Vec [?] learns to encode any sequence of words into a vector with the purpose of maximizing the likelihood of words which appear in that sequence given the encoding vector.

Block E contains models which combine Convolution Neural Networks and Recurrent Neural Networks.

5.4 Discussion

5.4.1 Combination of Convolution and Recursive Neural Networks. The fact that CNN-Tree-LSTM outperforms Constituency Tree-LSTM [?] and CNN-multichannel [?] supports our hypothesis on the benefits of combining convolution layers with Tree-LSTM. Additionally, the combination of CNN and LSTM or TreeLSTM outperforms most Convolution Network Networks in Block A. Furthermore, the results support our hypothesis that max pooling layer can be harmful to CNN by ignoring the position of features.

5.4.2 Glove Amazon versus standard Glove. On binary setting (Fig. 2), Glove Amazon helps single input-channel models to achieve higher accuracy compared to the standard Glove. However, on fine-grained setting (Fig. 3), Glove Amazon is worse than the standard Glove in most cases except for CNN-Tree-LSTM. In short, Glove Amazon is good for models on binary setting however harmful in the fine-grained setting. The reason is that in review dataset that we trained Glove Amazon on, words which are used to express the same sentiment are more likely to co-occur. Therefore,
Table 2: Experiment results of models evaluated on Stanford Sentiment Treebank. The accuracies of models in blocks A to E are taken from their original papers. We highlight the best results among our models and underline the state-of-the-art results. (*) Mean and standard deviation of 5 runs. (**) Mean of 100 runs, std was not reported.

| Block | Model | Binary | Fine-grained |
|-------|-------|--------|--------------|
| A     | CNN-non-static [?] | 87.2   | 48.0         |
|       | CNN-multichannel [?] | 88.1   | 47.4         |
|       | DCNN [?] | 86.8   | 48.5         |
|       | MVCNN [?] | 89.4   | 49.6         |
|       | LSTM [?] | 84.9 (0.6)* | 46.4 (1.1)* |
|       | BiLSTM [?] | 87.5 (0.5)* | 49.1 (1.0)* |
|       | 2-layer LSTM [?] | 86.3 (0.6)* | 46.0 (1.3)* |
|       | 2-layer Bidirectional LSTM [?] | 87.2 (1.0)* | 48.5 (1.0)* |
|       | DMN [?] | 88.6   | 52.1         |
|       | Byte mLSTM [?] | 91.80** | 52.90**       |
| B     | RNTN [?] | 85.4   | 45.7         |
|       | DRNN [?] | 86.6   | 49.8         |
|       | TE-RNTN [?] | 87.7   | 48.9         |
|       | Dependency Tree-LSTM [?] | 85.7 (0.4)* | 48.4 (0.4)* |
|       | Constituency Tree-LSTM [?] | 88.0 (0.3)* | 51.0 (0.5)* |
|       | Constituency Tree-LSTM Ensemble [?] | 90.2   | 53.6         |
| C     | GICF [?] | 85.7   | -            |
|       | Paragraph-Vec [?] | 87.8   | 48.7         |
|       | LSTM (PARAGRAM-SL999) [?] | 89.2   | -            |
| D     | CNN-GRU-word2vec [?] | 89.95  | 50.68        |
|       | CNN-LSTM-word2vec [?] | 89.56  | 51.50        |

F

| Model | Mean(std) | Max | Mean(std) | Max |
|-------|-----------|-----|-----------|-----|
| Constituency Tree-LSTM [?] (Glove Amazon) | 88.85 (0.44) | 89.35 | 50.53 (0.98) | 51.31 |
| CNN-LSTM | 89.10 (0.39) | 89.40 | 51.92 (0.63) | 52.66 |
| CNN-LSTM (Glove Amazon) | 89.25 (0.73) | 90.39 | 50.84 (0.79) | 51.85 |
| 2-channel CNN-LSTM | 89.44 (0.51) | 90.01 | 51.70 (0.57) | 52.53 |
| CNN-Tree-LSTM | 88.82 (0.13) | 88.92 | 51.35 (1.45) | 52.94 |
| CNN-Tree-LSTM (Glove Amazon) | 88.96 (0.24) | 89.18 | 51.51 (0.99) | 52.80 |
| 2-channel CNN-Tree-LSTM | 89.70 (0.36) | 90.12 | 52.46 (0.55) | 53.03 |

Figure 2: Mean accuracy of 5 runs on binary setting of Tree-LSTM, CNN-LSTM and CNN-Tree-LSTM using different word embeddings

Figure 3: Mean accuracy of 5 runs on fine-grained setting of Tree-LSTM, CNN-LSTM and CNN-Tree-LSTM using different word embeddings
the Glove Amazon word vectors are more dependent on the sentiment expressed in the words compared to the word vectors in the standard Glove which capture more general meaning of words.

A combination of Glove Amazon and the standard Glove improves the accuracy of both 2-channel CNN-LSTM and 2-channel CNN-Tree-LSTM on fine-grained setting as well as 2-channel CNN-Tree-LSTM on the binary setting. Yet, the improvements of CNN-Tree-LSTM are more significant compared to CNN-LSTM.

5.4.3 Tree-structured Versus Sequential Models. With single input channel, CNN-LSTM outperforms CNN-Tree-LSTM in most cases except the case of using Glove Amazon on the fine-grained setting. With two input channels using both Glove Amazon and the standard Glove, CNN-Tree-LSTM gains a large improvement and outperforms 2-channel CNN-LSTM on both binary and fine-grained setting. On average, 2-channel CNN-Tree-LSTM achieves the highest accuracy among all our models in both settings.

5.4.4 Comparing to the State-of-the-art Models. Constituency Tree-LSTM Ensemble is the state-of-the-art model on the fine-grained setting. Although the performance of 2-channel CNN-Tree-LSTM is lower than Constituency Tree-LSTM Ensemble, Constituency Tree-LSTM Ensemble is not an novel model and is an ensemble of 30 Constituency Tree-LSTMs. Since, the main purpose of Looks et al. were to use Constituency Tree-LSTM as an example to demonstrate the concise and batch-wise parallelism of its TensorFlow Fold implementation.

2-channel CNN-Tree-LSTM is comparable with Byte mLSTM on the fine-grained setting but underperforms Byte mLSTM in the binary setting. Similar to our approach, Radford et al. did transfer learning from large review dataset (which is also Amazon Reviews dataset) to improve the performance of their models on Stanford Sentiment Treebank. But different from our work, they trained a byte-level multiplicative LSTM language model on Amazon Reviews dataset. After that, the vector representations of all sentences in Stanford Sentiment Treebank were computed using the trained byte-level multiplicative LSTM. These vector representations along with their corresponding labels were used to train a logistic regression classifier.

On Stanford Sentiment Treebank, their method achieved state-of-the-art performance on the binary setting and comparable performance with Constituency Tree-LSTM Ensemble on fine-grained setting. Despite that, the performances of their model on other NLP tasks were not impressive, these tasks include: semantic relatedness (SICK), subjectivity/objectivity detection (SUBJ), opinion polarity (MPQA) and paraphrase detection (Microsoft Paraphrase Corpus). It is likely that these tasks are out-of-domain for their model which only trained on review dataset.

While conducting this research, we have trained CNN-LSTM as language model on Amazon Reviews dataset. Nevertheless, we have not gained any success with this method.

6 CONCLUSION

In this paper, we introduced a combination of Recursive Neural Network and Convolution Neural Network for sentence-level sentiment analysis. We experimented with both tree-structured and sequential Recursive Neural Networks. Using Stanford Sentiment Treebank, we demonstrated that Recurrent or Recursive Neural Networks can be used for combining phrase-level features produced by Convolution Neural Networks. Our experiments show that these combinations outperform most pure Convolution, Recurrent and Recursive Neural Networks. These results provide further support for the hypothesis that the usage of Recurrent or Recursive Neural Networks is better than k-max-pooling layer in the respect of preserving features’ position information and capturing long-range dependencies between features.

Additionally, in an attempt to improve vector presentations of words, we trained Glove vectors on the gigantic Amazon Reviews dataset (Glove Amazon). We have demonstrated Glove Amazon is good for these models on the binary setting but can be harmful to them in the fine-grained setting. Our experiments also show that a combination of both Glove Amazon and the standard Glove is more beneficial for CNN-Tree-LSTM compared to CNN-LSTM as 2-channel CNN-Tree-LSTM outperforms 2-channel CNN-LSTM on both binary and fine-grained setting.

ACKNOWLEDGMENTS

We gratefully acknowledges the support of VNG Corporation and research funding from Advanced Program in Computer Science, University of Science, Vietnam National University - Ho Chi Minh City.