A C-LSTM Neural Network for Text Classification

Chunting Zhou[1], **Chonglin Sun**[2], **Zhiyuan Liu**[3], **Francis C.M. Lau**[1]
Department of Computer Science, The University of Hong Kong[1]
School of Innovation Experiment, Dalian University of Technology[2]
Department of Computer Science and Technology, Tsinghua University, Beijing[3]

Abstract

Neural network models have been demonstrated to be capable of achieving remarkable performance in sentence and document modeling. Convolutional neural network (CNN) and recurrent neural network (RNN) are two mainstream architectures for such modeling tasks, which adopt totally different ways of understanding natural languages. In this work, we combine the strengths of both architectures and propose a novel and unified model called C-LSTM for sentence representation and text classification. C-LSTM utilizes CNN to extract a sequence of higher-level phrase representations, and are fed into a long short-term memory recurrent neural network (LSTM) to obtain the sentence representation. C-LSTM is able to capture both local features of phrases as well as global and temporal sentence semantics. We evaluate the proposed architecture on sentiment classification and question classification tasks. The experimental results show that the C-LSTM outperforms both CNN and LSTM and can achieve excellent performance on these tasks.

1 Introduction

As one of the core steps in NLP, sentence modeling aims at representing sentences as meaningful features for tasks such as sentiment classification. Traditional sentence modeling uses the bag-of-words model which often suffers from the curse of dimensionality; others use composition based methods instead, e.g., an algebraic operation over semantic word vectors to produce the semantic sentence vector. However, such methods may not perform well due to the loss of word order information. More recent models for distributed sentence representation fall into two categories according to the form of input sentence: sequence-based models and tree-structured models. Sequence-based models construct sentence representations from word sequences by taking in account the relationship between successive words (Johnson and Zhang, 2015). Tree-structured models treat each word token as a node in a syntactic parse tree and learn sentence representations from leaves to the root in a recursive manner (Socher et al., 2013b).

Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have emerged as two widely used architectures and are often combined with sequence-based or tree-structured models (Tai et al., 2015; Lei et al., 2015; Tang et al., 2015; Kim, 2014; Kalchbrenner et al., 2014; Mou et al., 2015).

Owing to the capability of capturing local correlations of spatial or temporal structures, CNNs have achieved top performance in computer vision, speech recognition and NLP. For sentence modeling, CNNs perform excellently in extracting n-gram features at different positions of a sentence through convolutional filters, and can learn short and long-range relations through pooling operations. CNNs have been successfully combined with both sequence-based model (Denil et al., 2014; Kalchbrenner et al., 2014) and tree-structured model (Mou et al., 2015) in sentence modeling.

The other popular neural network architecture – RNN – is able to handle sequences of any length and capture long-term dependencies. To avoid the

problem of gradient exploding or vanishing in the standard RNN, Long Short-term Memory RNN (LSTM) (Hochreiter and Schmidhuber, 1997) and other variants (Cho et al., 2014) were designed for better remembering and memory accesses. Along with the sequence-based (Tang et al., 2015) or the tree-structured (Tai et al., 2015) models, RNNs have achieved remarkable results in sentence or document modeling.

To conclude, CNN is able to learn local response from temporal or spatial data but lacks the ability of learning sequential correlations; on the other hand, RNN is specilized for sequential modelling but unable to extract features in a parallel way. It has been shown that higher-level modeling of x_t can help to disentangle underlying factors of variation within the input, which should then make it easier to learn temporal structure between successive time steps (Pascanu et al., 2014). For example, Sainath et al. (Sainath et al., 2015) have obtained respectable improvements in WER by learning a deep LSTM from multi-scale inputs. We explore training the LSTM model directly from sequences of higher-level representaions while preserving the sequence order of these representaions. In this paper, we introduce a new architecture short for C-LSTM by combining CNN and LSTM to model sentences. To benefit from the advantages of both CNN and RNN, we design a simple end-to-end, unified architecture by feeding the output of a one-layer CNN into LSTM. The CNN is constructed on top of the pre-trained word vectors from massive unlabeled text data to learn higher-level represention of n-grams. Then to learn sequential correlations from higher-level suqence representations, the feature maps of CNN are organized as sequential window features to serve as the input of LSTM. In this way, instead of constructing LSTM directly from the input sentence, we first transform each sentence into successive window (n-gram) features to help disentangle factors of variations within sentences. We choose sequence-based input other than relying on the syntactic parse trees before feeding in the neural network, thus our model doesn't rely on any external language knowledge and complicated pre-processing.

In our experiments, we evaluate the semantic sentence representations learned from C-LSTM with two tasks: sentiment classification and 6-way question classification. Our evaluations show that the C-LSTM model can achieve excellent results with several benchmarks as compared with a wide range of baseline models. We also show that the combination of CNN and LSTM outperforms individual multi-layer CNN models and RNN models, which indicates that LSTM can learn long-term dependencies from sequences of higher-level representations better than the other models.

2 Related Work

Deep learning based neural network models have achieved great success in many NLP tasks, including learning distributed word, sentence and document representation (Mikolov et al., 2013b; Le and Mikolov, 2014), parsing (Socher et al., 2013a), statistical machine translation (Devlin et al., 2014), sentiment classification (Kim, 2014), etc. Learning distributed sentence representation through neural network models requires little external domain knowledge and can reach satisfactory results in related tasks like sentiment classification, text categorization.

In many recent sentence representation learning works, neural network models are constructed upon either the input word sequences or the transformed syntactic parse tree. Among them, convolutional neural network (CNN) and recurrent neural network (RNN) are two popular ones.

The capability of capturing local correlations along with extracting higher-level correlations through pooling empowers CNN to model sentences naturally from consecutive context windows. In (Collobert et al., 2011), Collobert et al. applied convolutional filters to successive windows for a given sequence to extract global features by max-pooling. As a slight variant, Kim et al. (2014) proposed a CNN architecture with multiple filters (with a varying window size) and two 'channels' of word vectors. To capture word relations of varying sizes, Kalchbrenner et al. (2014) proposed a dynamic k-max pooling mechanism. In a more recent work (Lei et al., 2015), Tao et al. apply tensor-based operations between words to replace linear operations on concatenated word vectors in the standard convolutional layer and explore

the non-linear interactions between nonconsective n-grams. Mou et al. (2015) also explores convolutional models on tree-structured sentences.

As a sequence model, RNN is able to deal with variable-length input sequences and discover long-term dependencies. Various variants of RNN have been proposed to better store and access memories (Hochreiter and Schmidhuber, 1997; Cho et al., 2014). With the ability of explicitly modeling time-series data, RNNs are being increasingly applied to sentence modeling. For example, Tai et al. (2015) adjusted the standard LSTM to tree-structured topologies and obtained superior results over a sequential LSTM on related tasks.

In this paper, we stack CNN and LSTM in a unified architecture for semantic sentence modeling. The combination of CNN and LSTM can be seen in some computer vision tasks like image caption (Xu et al., 2015) and speech recognition (Sainath et al., 2015). Most of these models use multi-layer CNNs and train CNNs and RNNs separately or throw the output of a fully connected layer of CNN into RNN as inputs. Our approach is different: we apply CNN to text data and feed consecutive window features directly to LSTM, and so our architecture enables LSTM to learn long-range dependencies from higher-order sequential features. In (Li et al., 2015), the authors suggest that sequence-based models are sufficient to capture the compositional semantics for many NLP tasks, thus in this work the CNN is directly built upon word sequences other than the syntactic parse tree. Our experiments on sentiment classification and 6-way question classification tasks clearly demonstrate the superiority of our model over single CNN or LSTM model and other related sequence-based models.

3 C-LSTM Model

The architecture of the C-LSTM model is shown in Figure 1, which consists of two main components: convolutional neural network (CNN) and long short-term memory network (LSTM). The following two subsections describe how we apply CNN to extract higher-level sequences of word features and LSTM to capture long-term dependencies over window feature sequences respectively.

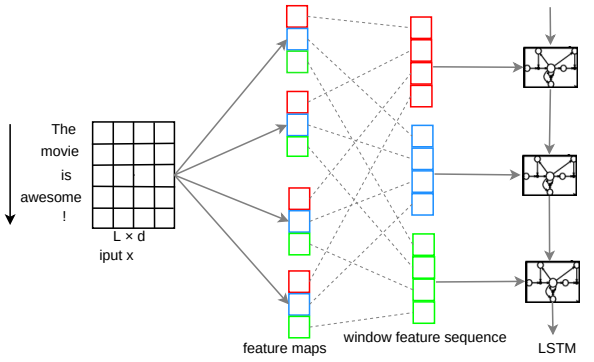

Figure 1: The architecture of C-LSTM for sentence modeling. Blocks of the same color in the feature map layer and window feature sequence layer corresponds to features for the same window. The dashed lines connect the feature of a window with the source feature map. The final output of the entire model is the last hidden unit of LSTM.

3.1 N-gram Feature Extraction through Convolution

The one-dimensional convolution involves a filter vector sliding over a sequence and detecting features at different positions. Let $\mathbf{x}_i \in \mathbb{R}^d$ be the d-dimensional word vectors for the i-th word in a sentence. Let $\mathbf{x} \in \mathbb{R}^{L \times d}$ denote the input sentence where L is the length of the sentence. Let k be the length of the filter, and the vector $\mathbf{m} \in \mathbb{R}^{k \times d}$ is a filter for the convolution operation. For each position j in the sentence, we have a window vector $\mathbf{w}_j$ with k consecutive word vectors, denoted as:

$$\mathbf{w}_j = [\mathbf{x}_j, \mathbf{x}_{j+1}, \cdots, \mathbf{x}_{j+k-1}] \tag{1}$$

Here, the commas represent row vector concatenation. A filter $\mathbf{m}$ convolves with the window vectors ($\mathbf{k}$-grams) at each position in a valid way to generate a feature map $\mathbf{c} \in \mathbb{R}^{L-k+1}$; each element c_j of the feature map for window vector $\mathbf{w}_j$ is produced as follows:

$$c_j = f(\mathbf{w}_j \circ \mathbf{m} + b), \tag{2}$$

where $\circ$ is element-wise multiplication, $b \in \mathbb{R}$ is a bias term and f is a nonlinear transformation function that can be sigmoid, hyperbolic tangent, etc. In our case, we choose ReLU (Nair and Hinton, 2010) as the nonlinear function. The C-LSTM model uses multiple filters to generate multiple feature maps. For n filters with the same length, the generated n

feature maps can be rearranged as feature representations for each window w_j,

$$\mathbf{W} = [\mathbf{c}_1; \mathbf{c}_2; \cdots; \mathbf{c}_n] \tag{3}$$

Here, semicolons represent column vector concatenation and $\mathbf{c}_i$ is the feature map generated with the i-th filter. Each row $\mathbf{W}_j$ of $\mathbf{W} \in \mathbb{R}^{(L-k+1)\times n}$ is the new feature representation generated from n filters for the window vector at position j. The new successive higher-order window representations then are fed into LSTM which is described below.

A max-over-pooling or dynamic k-max pooling is often applied to feature maps after the convolution to select the most or the k-most important features. However, LSTM is specified for sequence input, and pooling will break such sequence organization due to the discontinuous selected features. Since we stack an LSTM neural neural network on top of the CNN, we will not apply pooling after the convolution operation.

3.2 Long Short-Term Memory Networks

Recurrent neural networks (RNNs) are able to propagate historical information via a chain-like neural network architecture. While processing sequential data, it looks at the current input x_t as well as the previous output of hidden state h_{t-1} at each time step. However, standard RNNs becomes unable to learn long-term dependencies as the gap between two time steps becomes large. To address this issue, LSTM was first introduced in (Hochreiter and Schmidhuber, 1997) and reemerged as a successful architecture since Ilya et al. (2014) obtained remarkable performance in statistical machine translation. Although many variants of LSTM were proposed, we adopt the standard architecture (Hochreiter and Schmidhuber, 1997) in this work.

The LSTM architecture has a range of repeated modules for each time step as in a standard RNN. At each time step, the output of the module is controlled by a set of gates in $\mathbb{R}^d$ as a function of the old hidden state h_{t-1} and the input at the current time step x_t: the forget gate f_t, the input gate i_t, and the output gate o_t. These gates collectively decide how to update the current memory cell c_t and the current hidden state h_t. We use d to denote the memory dimension in the LSTM and all vectors in this architecture share the same dimension. The LSTM transition functions are defined as follows:

$$\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
q_t &= \tanh(W_q \cdot [h_{t-1}, x_t] + b_q) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot q_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{4}$$

Here, σ is the logistic sigmoid function that has an output in $[0, 1]$, $\tanh$ denotes the hyperbolic tangent function that has an output in $[-1, 1]$, and $\odot$ denotes the elementwise multiplication. To understand the mechanism behind the architecture, we can view f_t as the function to control to what extent the information from the old memory cell is going to be thrown away, i_t to control how much new information is going to be stored in the current memory cell, and o_t to control what to output based on the memory cell c_t. LSTM is explicitly designed for time-series data for learning long-term dependencies, and therefore we choose LSTM upon the convolution layer to learn such dependencies in the sequence of higher-level features.

4 Learning C-LSTM for Text Classification

For text classification, we regard the output of the hidden state at the last time step of LSTM as the document representation and we add a softmax layer on top. We train the entire model by minimizing the cross-entropy error. Given a training sample $\mathbf{x}^{(i)}$ and its true label $y^{(i)} \in \{1, 2, \cdots, k\}$ where k is the number of possible labels and the estimated probabilities $\widetilde{y}_j^{(i)} \in [0, 1]$ for each label $j \in \{1, 2, \cdots, k\}$, the error is defined as:

$$L(\mathbf{x}^{(i)}, y^{(i)}) = \sum_{j=1}^{k} 1\{y^{(i)} = j\} \log(\widetilde{y}_j^{(i)}), \tag{5}$$

where $1\{\text{condition}\}$ is an indicator such that $1\{\text{condition is true}\} = 1$ otherwise $1\{\text{condition is false}\} = 0$. We employ stochastic gradient descent (SGD) to learn the model parameters and adopt the optimizer RMSprop (Tieleman and Hinton, 2012).

4.1 Padding and Word Vector Initialization

First, we use $maxlen$ to denote the maximum length of the sentence in the training set. As the convolution layer in our model requires fixed-length input, we pad each sentence that has a length less than $maxlen$ with special symbols at the end that indicate the unknown words. For a sentence in the test dataset, we pad sentences that are shorter than $maxlen$ in the same way, but for sentences that have a length longer than $maxlen$, we simply cut extra words at the end of these sentences to reach $maxlen$.

We initialize word vectors with the publicly available `word2vec` vectors[1] that are pre-trained using about 100B words from the Google News Dataset. The dimensionality of the word vectors is 300. We also initialize the word vector for the unknown words from the uniform distribution [-0.25, 0.25]. We then fine-tune the word vectors along with other model parameters during training.

4.2 Regularization

For regularization, we employ two commonly used techniques: dropout (Hinton et al., 2012) and L2 weight regularization. We apply dropout to prevent co-adaptation. In our model, we either apply dropout to word vectors before feeding the sequence of words into the convolutional layer or to the output of LSTM before the softmax layer. The L2 regularization is applied to the weight of the softmax layer.

5 Experiments

We evaluate the C-LSTM model on two tasks: (1) sentiment classification, and (2) question type classification. In this section, we introduce the datasets and the experimental settings.

5.1 Datasets

Sentiment Classification: Our task in this regard is to predict the sentiment polarity of movie reviews. We use the Stanford Sentiment Treebank (SST) benchmark (Socher et al., 2013b). This dataset consists of 11855 movie reviews and are split into train (8544), dev (1101), and test (2210). Sentences in this corpus are parsed and all phrases along with the sentences are fully annotated with 5 labels: very positive, positive, neural, negative, very negative. We consider two classification tasks on this dataset: fine-grained classification with 5 labels and binary classification by removing neural labels. For the binary classification, the dataset has a split of train (6920) / dev (872) / test (1821). Since the data is provided in the format of sub-sentences, we train the model on both phrases and sentences but only test on the sentences as in several previous works (Socher et al., 2013b; Kalchbrenner et al., 2014).

Question type classification: Question classification is an important step in a question answering system that classifies a question into a specific type, e.g. *"what is the highest waterfall in the United States?"* is a question that belongs to "location". For this task, we use the benchmark TREC (Li and Roth, 2002). TREC divides all questions into 6 categories, including `location`, `human`, `entity`, `abbreviation`, `description` and `numeric`. The training dataset contains 5452 labelled questions while the testing dataset contains 500 questions.

5.2 Experimental Settings

We implement our model based on Theano (Bastien et al., 2012) – a python library, which supports efficient symbolic differentiation and transparent use of a GPU. To benefit from the efficiency of parallel computation of the tensors, we train the model on a GPU. For text preprocessing, we only convert all characters in the dataset to lower case.

For SST, we conduct hyperparameter (number of filters, filter length in CNN; memory dimension in LSTM; dropout rate and which layer to apply, etc.) tuning on the validation data in the standard split. For TREC, we hold out 1000 samples from the training dataset for hyperparameter search and train the model using the remaining data.

In our final settings, we only use one convolutional layer and one LSTM layer for both tasks. For the filter size, we investigated filter lengths of 2, 3 and 4 in two cases: a) single convolutional layer with the same filter length, and b) multiple convolutional layers with different lengths of filters in parallel. Here we denote the number of filters of length i by n_i for ease of clarification. For the first case, each n-gram window is transformed into n_i convoluted

[1] `http://code.google.com/p/word2vec/`

Model	Fine-grained (%)	Binary (%)	Reported in
SVM	40.7	79.4	(Socher et al., 2013b)
NBoW	42.4	80.5	(Kalchbrenner et al., 2014)
Paragraph Vector	48.7	87.8	(Le and Mikolov, 2014)
RAE	43.2	82.4	(Socher, Pennington, et al., 2011)
MV-RNN	44.4	82.9	(Socher et al., 2012)
RNTN	45.7	85.4	(Socher et al., 2013b)
DRNN	49.8	86.6	(Irsoy and Cardie, 2014)
CNN-non-static	48.0	87.2	(Kim, 2014)
CNN-multichannel	47.4	88.1	(Kim, 2014)
DCNN	48.5	86.8	(Kalchbrenner et al., 2014)
Molding-CNN	51.2	88.6	(Lei et al., 2015)
Dependency Tree-LSTM	48.4	85.7	(Tai et al., 2015)
Constituency Tree-LSTM	51.0	88.0	(Tai et al., 2015)
LSTM	46.6	86.6	our implementation
Bi-LSTM	47.8	87.9	our implementation
C-LSTM	49.2	87.8	our implementation

Table 1: Comparisons with baseline models on Stanford Sentiment Treebank. **Fine-grained** is a 5-class classification task. **Binary** is a 2-classification task. The second block contains the recursive models. The third block are methods related to convolutional neural networks. The fourth block contains methods using LSTM (the first two methods in this block also use syntactic parsing trees). The first block contains other baseline methods. The last block is our model.

features after convolution and the sequence of window representations is fed into LSTM. For the latter case, since the number of windows generated from each convolution layer varies when the filter length varies (see $L-k+1$ below equation (3)), we cut the window sequence at the end based on the maximum filter length that gives the shortest number of windows. Each window is represented as the concatenation of outputs from different convolutional layers. We also exploit different combinations of different filter lengths. We further present experimental analysis of the exploration on filter size later. According to the experiments, we choose a single convolutional layer with filter length 3.

For SST, the number of filters of length 3 is set to be 150 and the memory dimension of LSTM is set to be 150, too. The word vector layer and the LSTM layer are dropped out with a probability of 0.5. For TREC, the number of filters is set to be 300 and the memory dimension is set to be 300. The word vector layer and the LSTM layer are dropped out with a probability of 0.5. We also add L2 regularization with a factor of 0.001 to the weights in the softmax layer for both tasks.

6 Results and Model Analysis

In this section, we show our evaluation results on sentiment classification and question type classification tasks. Moreover, we give some model analysis on the filter size configuration.

6.1 Sentiment Classification

The results are shown in Table 1. We compare our model with a large set of well-performed models on the Stanford Sentiment Treebank.

Generally, the baseline models consist of recursive models, convolutional neural network models, LSTM related models and others. The recursive models employ a syntactic parse tree as the sentence structure and the sentence representation is computed recursively in a bottom-up manner along the parse tree. Under this category, we choose recursive autoencoder (**RAE**), matrix-vector (**MV-RNN**), tensor based composition (**RNTN**) and multi-layer stacked (**DRNN**) recursive neural network as baselines. Among CNNs, we compare with Kim's (2014) CNN model with fine-tuned word vectors (**CNN-non-static**) and multi-channels (**CNN-multichannel**), **DCNN** with dynamic k-max pool-

Model	Acc	Reported in
SVM	95.0	Silva et al .(2011)
Paragraph Vector	91.8	Zhao et al .(2015)
Ada-CNN	92.4	Zhao et al .(2015)
CNN-non-static	93.6	Kim (2014)
CNN-multichannel	92.2	Kim (2014)
DCNN	93.0	Kalchbrenner et al. (2014)
LSTM	93.2	our implementation
Bi-LSTM	93.0	our implementation
C-LSTM	94.6	our implementation

Table 2: The 6-way question type classification accuracy on TREC.

ing, Tao's CNN (**Molding-CNN**) with low-rank tensor based non-linear and non-consecutive convolutions. Among LSTM related models, we first compare with two tree-structured LSTM models (**Dependence Tree-LSTM** and **Constituency Tree-LSTM**) that adjust LSTM to tree-structured network topologies. Then we implement one-layer LSTM and Bi-LSTM by ourselves. Since we could not tune the result of Bi-LSTM to be as good as what has been reported in (Tai et al., 2015) even if following their untied weight configuration, we report our own results. For other baseline methods, we compare against **SVM** with unigram and bigram features, **NBoW** with average word vector features and **paragraph vector** that infers the new paragraph vector for unseen documents.

To the best of our knowledge, we achieve the fourth best published result for the 5-class classification task on this dataset. For the binary classification task, we achieve comparable results with respect to the state-of-the-art ones. From Table 1, we have the following observations: (1) Although we did not beat the state-of-the-art ones, as an end-to-end model, the result is still promising and comparable with thoes models that heavily rely on linguistic annotations and knowledge, especially syntactic parse trees. This indicates C-LSTM will be more feasible for various scenarios. (2) Comparing our results against single CNN and LSTM models shows that LSTM does learn long-term dependencies across sequences of higher-level representations better. We could explore in the future how to learn more compact higher-level representations by replacing standard convolution with other non-linear feature mapping functions or appealing to tree-structured topologies before the convolutional layer.

6.2 Question Type Classification

The prediction accuracy on TREC question classification is reported in Table 2. We compare our model with a variety of models. The **SVM** classifier uses unigrams, bigrams, wh-word, head word, POS tags, parser, hypernyms, WordNet synsets as engineered features and 60 hand-coded rules. **Ada-CNN** is a self-adaptiive hierarchical sentence model with gating networks. Other baseline models have been introduced in the last task. From Table 2, we have the following observations: (1) Our result consistently outperforms all published neural baseline models, which means that C-LSTM captures intentions of TREC questions well. (2) Our result is close to that of the state-of-the-art SVM that depends on highly engineered features. Such engineered features not only demands human laboring but also leads to the error propagation in the existing NLP tools, thus couldn't generalize well in other datasets and tasks. With the ability of automatically learning semantic sentence representations, C-LSTM doesn't require any human-designed features and has a better scalibility.

6.3 Model Analysis

Here we investigate the impact of different filter configurations in the convolutional layer on the model performance.

In the convolutional layer of our model, filters are used to capture local n-gram features. Intuitively, multiple convolutional layers in parallel with differ-

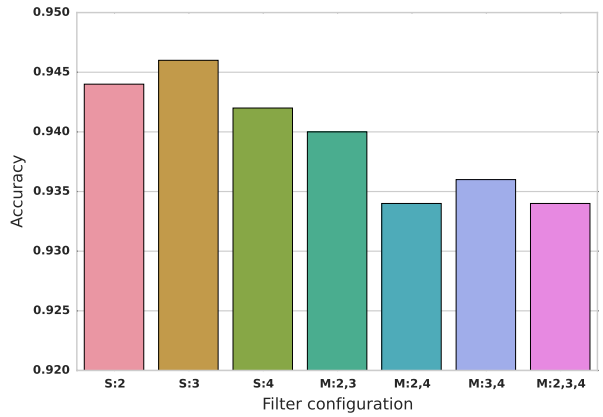

Figure 2: Prediction accuracies on TREC questions with different filter size strategies. For the horizontal axis, S means single convolutional layer with the same filter length, and M means multiple convolutional layers in parallel with different filter lengths.

ent filter sizes should perform better than single convolutional layers with the same length filters in that different filter sizes could exploit features of different n-grams. However, we found in our experiments that single convolutional layer with filter length 3 always outperforms the other cases.

We show in Figure 2 the prediction accuracies on the 6-way question classification task using different filter configurations. Note that we also observe the similar phenomenon in the sentiment classification task. For each filter configuration, we report in Figure 2 the best result under extensive grid-search on hyperparameters. It it shown that single convolutional layer with filter length 3 performs best among all filter configurations. For the case of multiple convolutional layers in parallel, it is shown that filter configurations with filter length 3 performs better that those without tri-gram filters, which further confirms that tri-gram features do play a significant role in capturing local features in our tasks. We conjecture that LSTM could learn better semantic sentence representations from sequences of tri-gram features.

7 Conclusion and Future Work

We have described a novel, unified model called C-LSTM that combines convolutional neural network with long short-term memory network (LSTM). C-LSTM is able to learn phrase-level features through a convolutional layer; sequences of such higher-level representations are then fed into the LSTM to learn long-term dependencies. We evaluated the learned semantic sentence representations on sentiment classification and question type classification tasks with very satisfactory results.

We could explore in the future ways to replace the standard convolution with tensor-based operations or tree-structured convolutions. We believe LSTM will benefit from more structured higher-level representations.

References

[Bastien et al.2012] 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.

[Cho et al.2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

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

[Denil et al.2014] Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. 2014. Modelling, visualising and summarising documents with a single convolutional neural network. *arXiv preprint arXiv:1406.3830*.

[Devlin et al.2014] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1370–1380.

[Hinton et al.2012] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *The Computing Research Repository (CoRR)*.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Irsoy and Cardie2014] Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Advances in Neural Information Processing Systems*, pages 2096–2104.

[Johnson and Zhang2015] Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with convolutional neural networks. *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, pages 103–112.

[Kalchbrenner et al.2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *Association for Computational Linguistics (ACL)*.

[Kim2014] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of Empirical Methods on Natural Language Processing*.

[Le and Mikolov2014] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196.

[Lei et al.2015] Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding cnns for text: non-linear, non-consecutive convolutions. In *Proceedings of Empirical Methods on Natural Language Processing*.

[Li and Roth2002] Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

[Li et al.2015] Jiwei Li, Dan Jurafsky, and Eudard Hovy. 2015. When are tree structures necessary for deep learning of representations? In *Proceedings of Empirical Methods on Natural Language Processing*.

[Mikolov et al.2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mou et al.2015] Lili Mou, Hao Peng, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2015. Discriminative neural sentence modeling by tree-based convolution. *Unpublished manuscript: http://arxiv. org/abs/1504. 01106v5. Version*, 5.

[Nair and Hinton2010] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

[Pascanu et al.2014] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to construct deep recurrent neural networks. In *Proceedings of the conference on International Conference on Learning Representations (ICLR)*.

[Sainath et al.2015] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Hasim Sak. 2015. Convolutional, long short-term memory, fully connected deep neural networks. *IEEE International Conference on Acoustics, Speech and Signal Processing*.

[Silva et al.2011] Joao Silva, Luísa Coheur, Ana Cristina Mendes, and Andreas Wichert. 2011. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154.

[Socher et al.2012] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of Empirical Methods on Natural Language Processing*, pages 1201–1211.

[Socher et al.2013a] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013a. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer.

[Socher et al.2013b] 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 *Proceedings of Empirical Methods on Natural Language Processing*, volume 1631, page 1642. Citeseer.

[Sutskever et al.2014] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Tai et al.2015] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *Association for Computational Linguistics (ACL)*.

[Tang et al.2015] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of Empirical Methods on Natural Language Processing*.

[Tieleman and Hinton2012] T. Tieleman and G Hinton. 2012. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning.

[Xu et al.2015] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of 2015th International Conference on Machine Learning*.

[Zhao et al.2015] Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence

model. In *Proceedings of International Joint Conferences on Artificial Intelligence*.