Discriminative Neural Sentence Modeling by Tree-Based Convolution

Lili Mou[*], Hao Peng[*], Ge Li[†], Yan Xu, Lu Zhang, Zhi Jin
{doublepower.mou, penghao.pku}@gmail.com
{lige, xuyan14, zhanglu, zhijin}@sei.pku.edu.cn
Software Institute, Peking University, 100871, P. R. China

Abstract

This paper proposes a tree-based convolutional neural network (TBCNN) for discriminative sentence modeling. Our model leverages either constituency trees or dependency trees of sentences. The tree-based convolution process extracts sentences structural features, which are then aggregated by max pooling. Such architecture allows short propagation paths between the output layer and underlying feature detectors, enabling effective structural feature learning and extraction. We evaluate our models on two tasks: sentiment analysis and question classification. In both experiments, TBCNN outperforms previous state-of-the-art results, including existing neural networks and dedicated feature/rule engineering. We also make efforts to visualize the tree-based convolution process, shedding light on how our models work.

1 Introduction

Discriminative sentence modeling aims to capture sentence meanings, and classify sentences according to certain criteria (e.g., sentiment). It is related to various tasks of interest, and has attracted much attention in the NLP community (Allan et al., 2003; Su and Markert, 2008; Zhao et al., 2015). Feature engineering—for example, n-gram features (Cui et al., 2006), dependency subtree features (Nakagawa et al., 2010), or more dedicated ones (Silva et al., 2011)—can play an important role in modeling sentences. Kernel machines, e.g., SVM, are exploited in Moschitti (2006) and Reichartz et al. (2010) by specifying a certain measure of similarity between sentences, without explicit feature representation.

Recent advances of neural networks bring new techniques in understanding natural languages, and have exhibited considerable potential. Bengio et al. (2003) and Mikolov et al. (2013) propose unsupervised approaches to learn word embeddings, mapping discrete words to real-valued vectors in a meaning space. Le and Mikolov (2014) extend such approaches to learn sentences' and paragraphs' representations. Compared with human engineering, neural networks serve as a way of automatic feature learning (Bengio et al., 2013).

Two widely used neural sentence models are convolutional neural networks (CNNs) and recursive neural networks (RNNs). CNNs can extract words' neighboring features effectively with short propagation paths, but they do not capture inherent sentence structures (e.g., parse trees). RNNs encode, to some extent, structural information by recursive semantic composition along a parse tree. However, they may have difficulties in learning deep dependencies because of long propagation paths (Erhan et al., 2009). (CNNs/RNNs and a variant, recurrent networks, will be reviewed in Section 2.)

A curious question is whether we can combine the advantages of CNNs and RNNs, i.e., whether we can exploit sentence structures (like RNNs) effectively with short propagation paths (like CNNs).

In this paper, we propose a novel neural architecture for discriminative sentence modeling, called the *Tree-Based Convolutional Neural Network* (TBCNN).[1] Our models can leverage different sentence parse trees, e.g., constituency trees and dependency trees. The model variants are denoted as c-TBCNN and d-TBCNN, respectively. The idea of tree-based convolution is to apply a set of subtree feature detectors, sliding over the entire

[*]These authors contribute equally to this paper.
[†]To whom correspondence should be addressed.

[1]The model of tree-based convolution was firstly proposed to process program source code in our (unpublished) previous work (Mou et al., 2014).

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2315–2325,
Lisbon, Portugal, 17-21 September 2015.

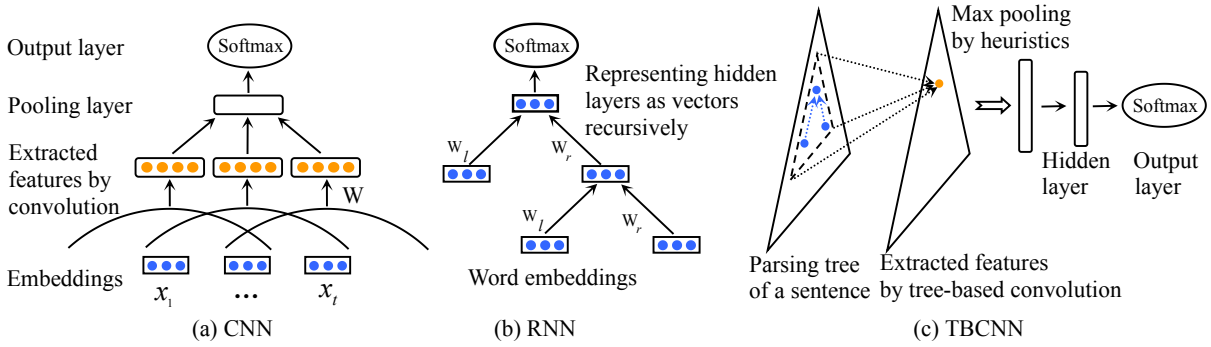

Figure 1: A comparison of information flow in the convolutional neural network (CNN), the recursive neural network (RNN), and the tree-based convolutional neural network (TBCNN).

parse tree of a sentence; then pooling aggregates these extracted feature vectors by taking the maximum value in each dimension. One merit of such architecture is that all features, along the tree, have short propagation paths to the output layer, and hence structural information can be learned effectively.

TBCNNs are evaluated on two tasks, sentiment analysis and question classification; our models have outperformed previous state-of-the-art results in both experiments. To understand how TBCNNs work, we also visualize the network by plotting the convolution process. We make our code and results available on our project website.[2]

2 Background and Related Work

In this section, we present the background and related work regarding two prevailing neural architectures for discriminative sentence modeling.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs), early used for image processing (LeCun, 1995), turn out to be effective with natural languages as well. Figure 1a depicts a classic convolution process on a sentence (Collobert and Weston, 2008). A set of fixed-width-window feature detectors slide over the sentence, and output the extracted features. Let t be the window size, and $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t \in \mathbb{R}^{n_e}$ be n_e-dimensional word embeddings. The output of convolution, evaluated at the current position, is

$$\boldsymbol{y} = f\left(W \cdot [\boldsymbol{x}_1; \cdots; \boldsymbol{x}_t] + \boldsymbol{b}\right)$$

where $\boldsymbol{y} \in \mathbb{R}^{n_c}$ (n_c is the number of feature detectors). $W \in \mathbb{R}^{n_c \times (t \cdot n_e)}$ and $\boldsymbol{b} \in \mathbb{R}^{n_c}$ are parameters; f is the activation function. Semicolons represent column vector concatenation. After convolution, the extracted features are pooled to a fixed-size vector for classification.

Convolution can extract neighboring information effectively. However, the features are "local"—words that are not in a same convolution window do not interact with each other, even though they may be semantically related. Blunsom et al. (2014) build deep convolutional networks so that local features can mix at high-level layers. Similar CNNs include Kim (2014) and Hu et al. (2014). All these models are "flat," by which we mean no structural information is used explicitly.

2.2 Recursive Neural Networks

Recursive neural networks (RNNs), proposed in Socher et al. (2011b), utilize sentence parse trees. In the original version, RNN is built upon a binarized constituency tree. Leaf nodes correspond to words in a sentence, represented by n_e-dimensional embeddings. Non-leaf nodes are sentence constituents, coded by child nodes recursively. Let node p be the parent of c_1 and c_2, vector representations denoted as $\boldsymbol{p}$, $\boldsymbol{c}_1$, and $\boldsymbol{c}_2$. The parent's representation is composited by

$$\boldsymbol{p} = f(W \cdot [\boldsymbol{c}_1; \boldsymbol{c}_2] + \boldsymbol{b}) \quad (1)$$

where W and $\boldsymbol{b}$ are parameters. This process is done recursively along the tree; the root vector is then used for supervised classification (Figure 1b).

Dependency parse and the combinatory categorical grammar can also be exploited as RNNs' skeletons (Hermann and Blunsom, 2013; Iyyer et al., 2014). Irsoy and Cardie (2014) build deep RNNs to enhance information interaction. Im-

[2] https://sites.google.com/site/tbcnnsentence/

provements for semantic compositionality include matrix-vector interaction (Socher et al., 2012), tensor interaction (Socher et al., 2013). They are more suitable for capturing logical information in sentences, such as negation and exclamation.

One potential problem of RNNs is that the long propagation paths—through which leaf nodes are connected to the output layer—may lead to information loss. Thus, RNNs bury illuminating information under a complicated neural architecture. Further, during back-propagation over a long path, gradients tend to vanish (or blow up), which makes training difficult (Erhan et al., 2009). Long short term memory (LSTM), first proposed for modeling time-series data (Hochreiter and Schmidhuber, 1997), is integrated to RNNs to alleviate this problem (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015).

Recurrent networks. A variant class of RNNs is the recurrent neural network (Bengio et al., 1994; Shang et al., 2015), whose architecture is a rightmost tree. In such models, meaningful tree structures are also lost, similar to CNNs.

3 Tree-based Convolution

This section introduces the proposed tree-based convolutional neural networks (TBCNNs). Figure 1c depicts the convolution process on a tree.

First, a sentence is converted to a parse tree, either a constituency or dependency tree. The corresponding model variants are denoted as c-TBCNN and d-TBCNN. Each node in the tree is represented as a distributed, real-valued vector.

Then, we design a set of fixed-depth subtree feature detectors, called the *tree-based convolution window*. The window slides over the entire tree to extract structural information of the sentence, illustrated by a dashed triangle in Figure 1c. Formally, let us assume we have t nodes in the convolution window, $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t$, each represented as an n_e-dimensional vector. Let n_c be the number of feature detectors. The output of the tree-based convolution window, evaluated at the current subtree, is given by the following generic equation.

$$\boldsymbol{y} = f\left(\sum_{i=1}^{t} W_i \cdot \boldsymbol{x}_i + \boldsymbol{b}\right) \tag{2}$$

where $W_i \in \mathbb{R}^{n_c \times n_e}$ is the weight parameter associated with node x_i; $\boldsymbol{b} \in \mathbb{R}^{n_c}$ is the bias term.

Extracted features are thereafter packed into one or more fixed-size vectors by max pooling, that is, the maximum value in each dimension is taken. Finally, we add a fully connected hidden layer, and a $\mathrm{softmax}$ output layer.

From the designed architecture (Figure 1c), we see that our TBCNN models allow short propagation paths between the output layer and any position in the tree. Therefore structural feature learning becomes effective.

Several main technical points in tree-based convolution include: (1) How can we represent hidden nodes as vectors in constituency trees? (2) How can we determine weights, W_i, for dependency trees, where nodes may have different numbers of children? (3) How can we pool varying sized and shaped features to fixed-size vectors?

In the rest of this section, we explain model variants in detail. Particularly, Subsections 3.1 and 3.2 address the first and second problems; Subsection 3.3 deals with the third problem by introducing several pooling heuristics. Subsection 3.4 presents our training objective.

3.1 c-TBCNN

Figure 2a illustrates an example of the constituency tree, where leaf nodes are words in the sentence, and non-leaf nodes represent a grammatical constituent, e.g., a noun phrase. Sentences are parsed by the Stanford parser;[3] further, constituency trees are binarized for simplicity.

One problem of constituency trees is that non-leaf nodes do not have such vector representations as word embeddings. Our strategy is to pretrain the constituency tree with an RNN by Equation 1 (Socher et al., 2011b). After pretraining, vector representations of nodes are fixed.

We now consider the tree-based convolution process in c-TBCNN with a two-layer-subtree convolution window, which operates on a parent node p and its direct children c_l and c_r, their vector representations denoted as $\boldsymbol{p}$, $\boldsymbol{c}_l$, and $\boldsymbol{c}_r$. The convolution equation, specific for c-TBCNN, is

$$\boldsymbol{y} = f\left(W_p^{(c)} \cdot \boldsymbol{p} + W_l^{(c)} \cdot \boldsymbol{c}_l + W_r^{(c)} \cdot \boldsymbol{c}_r + \boldsymbol{b}^{(c)}\right)$$

where $W_p^{(c)}$, $W_l^{(c)}$, and $W_r^{(c)}$ are weights associated with the parent and its child nodes. Superscript (c) indicates that the weights are for c-TBCNN. For leaf nodes, which do not have children, we set $\boldsymbol{c}_l$ and $\boldsymbol{c}_r$ to be $\mathbf{0}$.

[3] http://nlp.stanford.edu/software/lex-parser.shtml

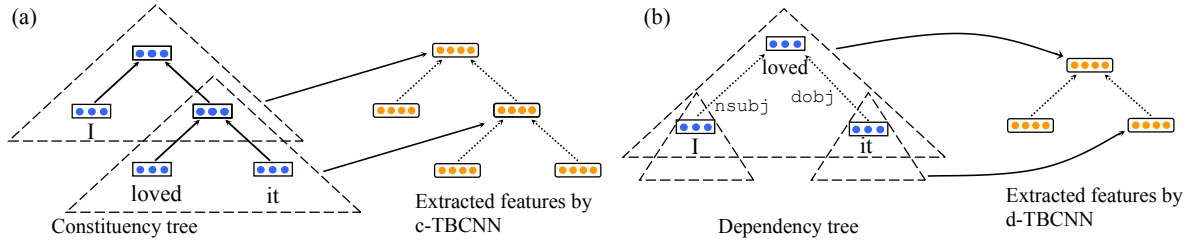

Figure 2: Tree-based convolution in (a) c-TBCNN, and (b) d-TBCNN. The parse trees correspond to the sentence "I loved it." The dashed triangles illustrate a shared-weight convolution window sliding over the tree. For clarity, only two positions are drawn in c-TBCNN. Notice that dotted arrows are not part of neural connections; they merely indicate the topologies of tree structures. Specially, an edge $a \xrightarrow{r} b$ in the dependency tree refers to a being governed by b with dependency type r.

Tree-based convolution windows can be extended to arbitrary depths straightforwardly. The complexity is exponential to the depth of the window, but linear to the number of nodes. Hence, tree-based convolution, compared with "flat" CNNs, does not add to computational cost, provided the same amount of information to process at a time. In our experiments, we use convolution windows of depth 2.

3.2 d-TBCNN

Dependency trees are another representation of sentence structures. The nature of dependency representation leads to d-TBCNN's major difference from traditional convolution: there exist nodes with different numbers of child nodes. This causes trouble if we associate weight parameters according to positions in the window, which is standard for traditional convolution, e.g., Collobert and Weston (2008) or c-TBCNN.

To overcome the problem, we extend the notion of convolution by assigning weights according to dependency types (e.g, `nsubj`) rather than positions. We believe this strategy makes much sense because dependency types (de Marneffe et al., 2006) reflect the relationship between a governing word and its child words. To be concrete, the generic convolution formula (Equation 2) for d-TBCNN becomes

$$\boldsymbol{y} = f\left(W_p^{(d)} \cdot \boldsymbol{p} + \sum_{i=1}^{n} W_{r[c_i]}^{(d)} \cdot \boldsymbol{c}_i + \boldsymbol{b}^{(d)}\right)$$

where $W_p^{(d)}$ is the weight parameter for the parent p (governing word); $W_{r[c_i]}^{(d)}$ is the weight for child c_i, who has grammatical relationship $r[c_i]$ to its parent, p. Superscript (d) indicates the parameters are for d-TBCNN. Note that we keep 15 most frequently occurred dependency types; others appearing rarely in the corpus are mapped to one shared weight matrix.

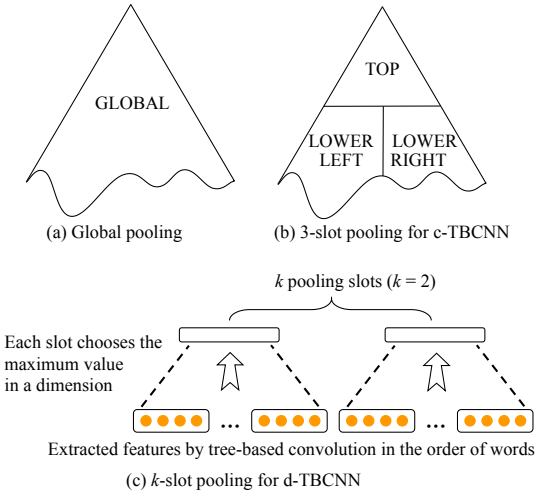

Figure 3: Pooling heuristics. (a) Global pooling. (b) 3-slot pooling for c-TBCNN. (c) k-slot pooling for d-TBCNN.

Both c-TBCNN and d-TBCNN have their own advantages: d-TBCNN exploits structural features more efficiently because of the compact expressiveness of dependency trees; c-TBCNN may be more effective in integrating global features due to the underneath pretrained RNN.

3.3 Pooling Heuristics

As different sentences may have different lengths and tree structures, the extracted features by tree-based convolution also have topologies varying in size and shape. Dynamic pooling (Socher et al., 2011a) is a common technique for dealing with

Task	Data samples	Label
Sentiment Analysis	Offers that rare combination of entertainment and education.	++
	An idealistic love story that brings out the latent 15-year-old romantic in everyone.	+
	Its mysteries are transparently obvious, and it's too slowly paced to be a thriller.	−
Question Classification	What is the temperature at the center of the earth?	`number`
	What state did the Battle of Bighorn take place in?	`location`

Table 1: Data samples in sentiment analysis and question classification. In the first task, "++" refers to strongly positive; "+" and "−" refer to positive and negative, respectively.

this problem. We propose several heuristics for pooling along a tree structure. Our generic design criteria for pooling include: (1) Nodes that are pooled to one slot should be "neighboring" from some viewpoint. (2) Each slot should have similar numbers of nodes, in expectation, that are pooled to it. Thus, (approximately) equal amount of information is aggregated along different parts of the tree. Following the above intuition, we propose pooling heuristics as follows.

- Global pooling. All features are pooled to one vector, shown in Figure 3a. We take the maximum value in each dimension. This simple heuristic is applicable to any structure, including c-TBCNN and d-TBCNN.
- 3-slot pooling for c-TBCNN. To preserve more information over different parts of constituency trees, we propose 3-slot pooling (Figure 3b). If a tree has maximum depth d, we pool nodes of less than $\alpha \cdot d$ layers to a TOP slot (α is set to 0.6); lower nodes are pooled to slots LOWER_LEFT or LOWER_RIGHT according to their relative position with respect to the root node.
  For a constituency tree, it is not completely obvious how to pool features to more than 3 slots and comply with the aforementioned criteria at the same time. Therefore, we regard 3-slot pooling for c-TBCNN is a "hard mechanism" temporarily. Further improvement can be addressed in future work.
- k-slot pooling for d-TBCNN. Different from constituency trees, nodes in dependency trees are one-one corresponding to words in a sentence. Thus, a total order on features (after convolution) can be defined according to their corresponding word orders. For k-slot pooling, we can adopt an "equal allocation" strategy, shown in Figure 3c. Let i be the position of a word in a sentence ($i = 1, 2, \cdots, n$). Its extracted feature vector is pooled to the j-th slot, if

$$(j-1)\frac{n}{k} \leq i \leq j\frac{n}{k}$$

We assess the efficacy of pooling quantitatively in Section 4.3.1. As we shall see by the experimental results, complicated pooling methods do preserve more information along tree structures to some extent, but the effect is not large. TBCNNs are not very sensitive to pooling methods.

3.4 Training Objective

After pooling, information is packed into one or more fixed-size vectors (slots). We add a hidden layer, and then a $\mathrm{softmax}$ layer to predict the probability of each target label in a classification task. The error function of a sample is the standard cross entropy loss, i.e., $J = -\sum_{i=1}^{c} t_i \log y_i$, where $\boldsymbol{t}$ is the ground truth (one-hot represented), $\boldsymbol{y}$ the output by $\mathrm{softmax}$, and c the number of classes. To regularize our model, we apply both ℓ_2 penalty and dropout (Srivastava et al., 2014). Training details are further presented in Section 4.1 and 4.2.

4 Experimental Results

In this section, we evaluate our models with two tasks, sentiment analysis and question classification. We also conduct quantitative and qualitative model analysis in Subsection 4.3.

4.1 Sentiment Analysis

4.1.1 The Task and Dataset

Sentiment analysis is a widely studied task for discriminative sentence modeling. The Stanford sentiment treebank[4] consists of more than 10,000 movie reviews. Two settings are considered for sentiment prediction: (1) fine-grained classification with 5 labels (`strongly positive`, `positive`, `neutral`, `negative`, and `strongly negative`), and (2) coarse-gained polarity classification with 2 labels (`positive` versus `negative`). Some examples are shown in

[4] http://nlp.stanford.edu/sentiment/

Table 1. We use the standard split for training, validating, and testing, containing 8544/1101/2210 sentences for 5-class prediction. Binary classification does not contain the `neutral` class.

In the dataset, phrases (sub-sentences) are also tagged with sentiment labels. RNNs deal with them naturally during the recursive process. We regard sub-sentences as individual samples during training, like Blunsom et al. (2014) and Le and Mikolov (2014). The training set therefore has more than 150,000 entries in total. For validating and testing, only whole sentences (root labels) are considered in our experiments.

Both c-TBCNN and d-TBCNN use the Stanford parser for data preprocessing.

4.1.2 Training Details

This subsection describes training details for d-TBCNN, where hyperparameters are chosen by validation. c-TBCNN is mostly tuned synchronously (e.g., optimization algorithm, activation function) with some changes in hyperparameters. c-TBCNN's settings can be found on our website.

In our d-TBCNN model, the number of units is 300 for convolution and 200 for the last hidden layer. Word embeddings are 300 dimensional, pretrained ourselves using `word2vec` (Mikolov et al., 2013) on the English Wikipedia corpus. 2-slot pooling is applied for d-TBCNN. (c-TBCNN uses 3-slot pooling.)

To train our model, we compute gradient by back-propagation and apply stochastic gradient descent with mini-batch 200. We use ReLU (Nair and Hinton, 2010) as the activation function .

For regularization, we add ℓ_2 penalty for weights with a coefficient of 10^{-5}. Dropout (Srivastava et al., 2014) is further applied to both weights and embeddings. All hidden layers are dropped out by 50%, and embeddings 40%.

4.1.3 Performance

Table 2 compares our models to state-of-the-art results in the task of sentiment analysis. For 5-class prediction, d-TBCNN yields 51.4% accuracy, outperforming the previous state-of-the-art result, achieved by the RNN based on long-short term memory (Tai et al., 2015). c-TBCNN is slightly worse. It achieves 50.4% accuracy, ranking third in the state-of-the-art list (including our d-TBCNN model).

Regarding 2-class prediction, we adopted a simple strategy in Irsoy and Cardie (2014),[5] where the 5-class network is "transferred" directly for binary classification, with estimated target probabilities (by 5-way $\mathrm{softmax}$) reinterpreted for 2 classes. (The `neutral` class is discarded as in other studies.) This strategy enables us to take a glance at the stability of our TBCNN models, but places itself in a difficult position. Nonetheless, our d-TBCNN model achieves 87.9% accuracy, ranking forth in the list.

In a more controlled comparison—with shallow architectures and the basic interaction (linearly transformed and non-linearly squashed)—TBCNNs, of both variants, consistently outperform RNNs (Socher et al., 2011b) to a large extent (50.4–51.4% versus 43.2%); they also consistently outperform "flat" CNNs by more than 10%. Such results show that structures are important when modeling sentences; tree-based convolution can capture these structural information more effectively than RNNs.

We also observe d-TBCNN achieves higher performance than c-TBCNN. This suggests that compact tree expressiveness is more important than integrating global information in this task.

4.2 Question Classification

We further evaluate TBCNN models on a question classification task.[6] The dataset contains 5452 annotated sentences plus 500 test samples in TREC 10. We also use the standard split, like Silva et al. (2011). Target labels contain 6 classes, namely `abbreviation`, `entity`, `description`, `human`, `location`, and `numeric`. Some examples are also shown in Table 1.

We chose this task to evaluate our models because the number of training samples is rather small, so that we can know TBCNNs' performance when applied to datasets of different sizes. To alleviate the problem of data sparseness, we set the dimensions of convolutional layer and the last hidden layer to 30 and 25, respectively. We do not back-propagate gradient to embeddings in this

[5] Richard Socher, who first applies neural networks to this task, thinks direct transfer is fine for binary classification. We followed this strategy for simplicity as it is non-trivial to deal with the neutral sub-sentences in the training set if we train a separate model. Our website reviews some related work and provides more discussions.

[6] http://cogcomp.cs.illinois.edu/Data/QA/QC/

Group	Method	5-class accuracy	2-class accuracy	Reported in
Baseline	SVM	40.7	79.4	Socher et al. (2013)
	Naïve Bayes	41.0	81.8	Socher et al. (2013)
CNNs	1-layer convolution	37.4	77.1	Blunsom et al. (2014)
	Deep CNN	48.5	86.8	Blunsom et al. (2014)
	Non-static	48.0	87.2	Kim (2014)
	Multichannel	47.4	**88.1**	Kim (2014)
RNNs	Basic	43.2	82.4	Socher et al. (2013)
	Matrix-vector	44.4	82.9	Socher et al. (2013)
	Tensor	45.7	85.4	Socher et al. (2013)
	Tree LSTM (variant 1)	48.0	–	Zhu et al. (2015)
	Tree LSTM (variant 2)	51.0	88.0	Tai et al. (2015)
	Tree LSTM (variant 3)	49.9	88.0	Le and Zuidema (2015)
	Deep RNN	49.8	86.6†	Irsoy and Cardie (2014)
Recurrent	LSTM	45.8	86.7	Tai et al. (2015)
	bi-LSTM	49.1	86.8	Tai et al. (2015)
Vector	Word vector avg.	32.7	80.1	Socher et al. (2013)
	Paragraph vector	48.7	87.8	Le and Mikolov (2014)
TBCNNs	c-TBCNN	50.4	86.8†	Our implementation
	d-TBCNN	**51.4**	87.9†	Our implementation

Table 2: Accuracy of sentiment prediction (in percentage). For 2-class prediction, "†" remarks indicate that the network is transferred directly from that of 5-class.

Method	Acc. (%)	Reported in
SVM 10k features + 60 rules	95.0	Silva et al. (2011)
CNN-non-static	93.6	Kim (2014)
CNN-mutlichannel	92.2	Kim (2014)
RNN	90.2	Zhao et al. (2015)
Deep-CNN	93.0	Blunsom et al. (2014)
Ada-CNN	92.4	Zhao et al. (2015)
c-TBCNN	94.8	Our implementation
d-TBCNN	**96.0**	Our implementation

Table 3: Accuracy of 6-way question classification.

task. Dropout rate for embeddings is 30%; hidden layers are dropped out by 5%.

Table 3 compares our models to various other methods. The first entry presents the previous state-of-the-art result, achieved by traditional feature/rule engineering (Silva et al., 2011). Their method utilizes more than 10k features and 60 hand-coded rules. On the contrary, our TBCNN models do not use a single human-engineered feature or rule. Despite this, c-TBCNN achieves similar accuracy compared with feature engineering; d-TBCNN pushes the state-of-the-art result to 96%. To the best of our knowledge, this is the first time that neural networks beat dedicated human engineering in this question classification task.

The result also shows that both c-TBCNN and d-TBCNN reduce the error rate to a large extent, compared with other neural architectures in this task.

4.3 Model Analysis

In this part, we analyze our models quantitatively and qualitatively in several aspects, shedding some light on the mechanism of TBCNNs.

4.3.1 The Effect of Pooling

The extracted features by tree-based convolution have topologies varying in size and shape. We propose in Section 3.3 several heuristics for pooling. This subsection aims to provide a fair comparison among these pooling methods.

One reasonable protocol for comparison is to tune all hyperparameters for each setting and compare the highest accuracy. This methodology, however, is too time-consuming, and depends largely on the quality of hyperparameter tuning. An alternative is to predefine a set of sensible hyperparameters and report the accuracy under the same setting. In this experiment, we chose the latter protocol, where hidden layers are all 300-dimensional; no ℓ_2 penalty is added. Each configuration was run five times with different random initializations. We summarize the mean and standard deviation in Table 4.

As the results imply, complicated pooling is better than global pooling to some degree for both model variants. But the effect is not strong; our models are not that sensitive to pooling methods, which mainly serve as a necessity for dealing with varying-structure data. In our experiments, we apply 3-slot pooling for c-TBCNN and 2-slot pooling for d-TBCNN.

Model	Pooling method	5-class accuracy (%)
c-TBCNN	Global	48.48 ± 0.54
	3-slot	48.69 ± 0.40
d-TBCNN	Global	49.39 ± 0.24
	2-slot	49.94 ± 0.63

Table 4: Accuracies of different pooling methods, averaged over 5 random initializations. We chose sensible hyperparameters manually in advance to make a fair comparison. This leads to performance degradation (1–2%) vis-a-vis Table 2.

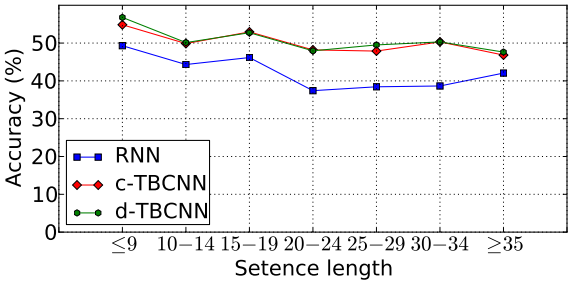

Figure 4: Accuracies versus sentence lengths.

Comparing with other studies in the literature, we also notice that pooling is very effective and efficient in information gathering. Irsoy and Cardie (2014) report 200 epochs for training a deep RNN, which achieves 49.8% accuracy in the 5-class sentiment classification. Our TBCNNs are typically trained within 25 epochs.

4.3.2 The Effect of Sentence Lengths

We analyze how sentence lengths affect our models. Sentences are split into 7 groups by length, with granularity 5. A few too long or too short sentences are grouped together for smoothing; the numbers of sentences in each group vary from 126 to 457. Figure 4 presents accuracies versus lengths in TBCNNs. For comparison, we also reimplemented RNN, achieving 42.7% overall accuracy, slightly worse than 43.2% reported in Socher et al. (2011b). Thus, we think our reimplementation is fair and that the comparison is sensible.

We observe that c-TBCNN and d-TBCNN yield very similar behaviors. They consistently outperform the RNN in all scenarios. We also notice the gap, between TBCNNs and RNN, increases when sentences contain more than 20 words. This result confirms our theoretical analysis in Section 2—for long sentences, the propagation paths in RNNs are deep, causing RNNs' difficulty in information processing. By contrast, our models explore structural information more effectively with tree-based convolution. As information from any part of the tree can propagate to the output layer with short paths, TBCNNs are more capable for sentence modeling, especially for long sentences.

4.3.3 Visualization

Visualization is important to understanding the mechanism of neural networks. For TBCNNs, we would like to see how the extracted features (after convolution) are further processed by the max pooling layer, and ultimately related to the supervised task.

To show this, we trace back where the max pooling layer's features come from. For each dimension, the pooling layer chooses the maximum value from the nodes that are pooled to it. Thus, we can count the fraction in which a node's features are gathered by pooling. Intuitively, if a node's features are more related to the task, the fraction tends to be larger, and vice versa.

Figure 5 illustrates an example processed by d-TBCNN in the task of sentiment analysis.[7] Here, we applied global pooling because information tracing is more sensible with one pooling slot. As shown in the figure, tree-based convolution can effectively extract information relevant to the task of interest. The 2-layer windows corresponding to "*visual will impress viewers*," "*the stunning dreamlike visual*," say, are discriminative to the sentence's sentiment. Hence, large fractions (0.24 and 0.19) of their features, after convolution, are gathered by pooling. On the other hand, words like *the*, *will*, *even* are known as stop words (Fox, 1989). They are mostly noninformative for sentiment; hence, no (or minimal) features are gathered. Such results are consistent with human intuition.

We further observe that tree-based convolution does integrate information of different words in the window. For example, the word *stunning* appears in two windows: (a) the window "*stunning*" itself, and (b) the window of "*the stunning dreamlike visual*," with root node *visual*, *stunning* acting as a child. We see that Window b is more relevant to the ultimate sentiment than Window a, with fractions 0.19 versus 0.07, even though the root *visual* itself is neutral in sentiment. In fact,

[7] We only have space to present one example in the paper. This example was not chosen deliberately. Similar traits can be found through out the entire gallery, available on our website. Also, we only present d-TBCNN, noticing that dependency trees are intrinsically more suitable for visualization since we know the "meaning" of every node.

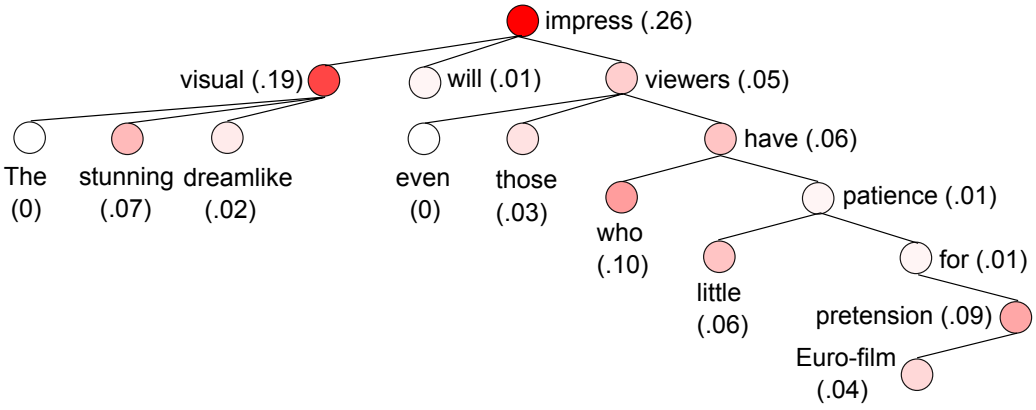

Figure 5: Visualizing how features (after convolution) are related to the sentiment of a sentence. The sample corresponds a sentence in the dataset, "The stunning dreamlike visual will impress even those viewers who have little patience for Euro-film pretension." The numbers in brackets denote the fraction of a node's features that are gathered by the max pooling layer (also indicated by colors).

Window a has a larger fraction than the sum of its children's (the windows of "*the*," "*stunning*," and "*dreamlike*").

5 Conclusion

In this paper, we proposed a novel neural discriminative sentence model based on sentence parsing structures. Our model can be built upon either constituency trees (denoted as c-TBCNN) or dependency trees (d-TBCNN).

Both variants have achieved high performance in sentiment analysis and question classification. d-TBCNN is slightly better than c-TBCNN in our experiments, and has outperformed previous state-of-the-art results in both tasks. The results show that tree-based convolution can capture sentences' structural information effectively, which is useful for sentence modeling.

Acknowledgments

This research is supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201 and the National Natural Science Foundation of China under Grant Nos. 61232015 and 91318301.

References

James Allan, Courtney Wade, and Alvaro Bolivar. 2003. Retrieval and novelty detection at the sentence level. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, pages 314–321. ACM.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Yoshua Bengio, Aaron Courville, and Pierre Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Phil Blunsom, Edward Grefenstette, and Nal Kalchbrenner. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

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*, pages 160–167.

Hang Cui, Vibhu Mittal, and Mayur Datar. 2006. Comparative experiments on sentiment classification for online product reviews. In *Proceedings 21st AAAI Conference on Artificial Intelligence*, volume 6, pages 1265–1270.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of Language Resource and Evaluation Conference*, volume 6, pages 449–454.

Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. 2009. The difficulty of training deep architectures and the

effect of unsupervised pre-training. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, pages 153–160.

Christopher Fox. 1989. A stop list for general text. In *ACM SIGIR Forum*, volume 24, pages 19–21.

Karl M. 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*, pages 894–904.

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

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, pages 2042–2050.

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

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. 2014. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 633–644.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. *arXiv preprint arXiv:1503.02510*.

Yann LeCun. 1995. Comparison of learning algorithms for handwritten digit recognition. In *Proceedings of International Conference on Artificial Neural Networks*, volume 60, pages 53–60.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of European Conference of Machine Learning*, pages 318–329. Springer.

Lili Mou, Ge Li, Zhi Jin, Lu Zhang, and Tao Wang. 2014. TBCNN: A tree-based convolutional neural network for programming language processing. *arXiv preprint arXiv:1409.5718*.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814.

Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. 2010. Dependency tree-based sentiment classification using CRFs with hidden variables. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 786–794.

Frank Reichartz, Hannes Korte, and Gerhard Paass. 2010. Semantic relation extraction with kernels over typed dependency trees. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 773–782.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1577–1586.

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

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 *Advances in Neural Information Processing Systems*, pages 801–809.

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*, pages 151–161.

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*, pages 1201–1211.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Fangzhong Su and Katja Markert. 2008. From words to senses: a case study of subjectivity recognition. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 825–832. Association for Computational Linguistics.

Kaisheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1556–1566.

Han Zhao, Zhengdong Lu, and Pascal Poupart. 2015. Self-adaptive hierarchical sentence model. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 4069–4076.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over tree structures. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1604–1612.