Sentiment Classification Via Recurrent Convolutional Neural Networks

CHANGSHUN DU and LEI HUANG

ABSTRACT

The state-of-the-art methods used for sentiment classification are primarily based on statistical machine learning, and their performance strongly depends on the quality of the extracted features. The extracted features are often derived from the output of pre-existing natural language processing (NLP) systems, which leads to the propagation of errors in the existing tools and hinders the performance of these systems. In contrast to traditional methods, this paper introduces a recurrent convolutional neural network for text classification that works independently of and without human-designed features. The model applies a recurrent structure to capture as much contextual information as far as possible when learning word representations, which may introduce considerably less noise compared to traditional window-based neural networks. In addition, we also employ a max-pooling layer that automatically judges which words play key roles in sentiment classification to capture the key components in texts. We also conduct experiments on movie review datasets. These experimental results show that the proposed method outperforms current state-of-the-art methods.

KEYWORDS

Sentiment classification, Recurrent neural networks, Convolutional neural networks

INTRODUCTION

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level and determining whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Detecting sentiments in plain texts is a challenging task which has recently spawned great interest [1].

At present, there are some neural networks based methods that have been used in the sentiment classification task. Socher et al. [2, 3, 4] proposed the Recursive Neural Network (RecursiveNN). It has been shown to be effective in constructing sentence representation. However, the RecursiveNN is first captured by the tree structure in order to capture the semantics of the sentence. To a large extent, the performance of
text tree structure determines its performance. In addition, the construction of such a text tree exhibits a time complexity factor of at least $O(n^2)$, where the text’s length is $n$.

It takes a long time to apply this model to a long sentence or a document. In addition, relationships between the two sentences are difficult to represent using a tree structure. So, recursion is not suitable for modeling long sentences and documents. Another model, which only exhibits a time complexity factor of $O(n)$, is the Recurrent Neural Network (RecurrentNN). This model uses a word to analyze a text word, and it stores all the previous text semantics in the hidden layer of a fixed size. RecurrentNN has the advantage of being more able to capture contextual information. This may be useful for capturing the semantics of long texts. However, the RecurrentNN is a biased model, where later words are carry more weight than earlier words. As we know, critical components may appear in any location in the document, not just at its end. So, when it is used to capture the semantics of an entire document, its effectiveness will be diminished and it may overlook important information in practice. In addition, some work also uses the Convolutional Neural Network (CNN) for sentiment classification. It has been introduced into the Natural Language Processing mission to solve the problem of deviation because it is unbiased in determining the distinct phrases in a text using the maximum pool layer. As a result, compared with recursive or recurrent neural networks, CNN may be more beneficial to the process of capturing text semantics. CNN's time complexity is $O(n)$, however, previous studies on CNNs tend to use simple convolutional kernels such as a fixed window [5, 6]. When using such a kernel, it is difficult to determine the size of the window. Small windows can cause important information to be lost while large windows result in huge parameter spaces (which may be difficult to train). The literature [7] uses a CNN and a LSTM, which is a deformation of the RNN model, to model sentences respectively. Although it can improve experimental results, it still can’t overcome the defects of CNN and RNN.

![Figure 1. The structure of the cyclic convolution neural network.](image)

To address the limitations of the above RecursiveNN, RecurrentNN and CNN models, we use a Recurrent Convolutional Neural Network (RCNN) [8] and apply it to emotional classification. First, we employ a bi-directional repetition structure, allowing for the traditional neural network based on windows to be incorporated with significantly less noise, while capturing as much context information as possible through word learning. In addition, when learning text, the model can maintain a larger range of word order. Second, we use a max-pooling layer to automatically
determine which features play a key role in the emotional classification and capture
the key components of the text. We combine the cyclic structure with the maximum
pool layer, and give it the combined advantages of both the recurrent neural model and
the convolution model. In addition, the time complexity of our model is and it has a
linear relationship with the length of the text.

METHODOLOGY

In this paper, our task is to identify the polarity of documents, namely as
positive and negative. We use two sets of criteria to evaluate sentiment analysis
which include a 2-class and 5-class classification task. The 2-class includes positive
and negative, and the 5-class contains positive, somewhat positive, neutral,
somewhat negative and negative. The latter is a more fine-grained and thorough
measurement. We will report accuracies and show analyses of the two metrics.

To capture the semantics of a document, we present a deep neural model. Figure
1 is our model’s network structure. The input of the network is the document \( D \),
which is the word \( w_1, w_2, \ldots, w_n \) sequence. The output of the network contains class
elements. We use \( p(k \mid D, \theta) \) to denote the probability of the document being class
\( k \), where \( \theta \) represents the parameters in the network.

This figure is a partial example of the sentence “A sunset stroll along the South
Bank affords an array of stunning vantage points”, and the subscript denotes the
position of the corresponding word in the original sentence.

Pre-trained Word Vectors

Word embedding is a distributed representation of words. The distributed
representation is applied to the input of the neural network. Traditional
representations, such as thermal representations, lead to dimensionality. Recent
studies have shown that neural networks can converge to a better local minimum
with an appropriate unsupervised pre-training program.

In this work, we use the Skip-gram model to pre-embed the word. This model is
the most advanced of all the NLP tasks. The Skip-gram model trains the embedding
of words \( w_1, w_2, \ldots, w_T \) by maximizing the average log probability

\[
\frac{1}{T} \sum_{t=1}^{T} \sum_{j \neq i, j \neq 0} \log p(w_{i,j} \mid w_i) \tag{1}
\]

\[
p(w_i \mid w_j) = \frac{\exp(e'(w_i)^T e(w_j))}{\sum_{k=1}^{|V|} \exp(e'(w_i)^T e(w_k))} \tag{2}
\]

where \( |V| \) is the vocabulary of the unlabeled text. \( e'(w_i) \) is another embedded
\( w_i \). We also use embedded \( e \) when we use some accelerated methods (e.g.,
hierarchical softmax), and \( e' \) is not evaluated in practice. These word embeddings
can be used as initial word vectors for the next steps.
Word Representation Learning

We combine a word and its context to express a word. We get more precise semantics through context. In our model, we use a regular structure to capture the background which is a bidirectional recurrent neural network.

We define \( c_l(w_i) \) as the left context of the word \( w_i \), \( c_r(w_i) \) as the right context of the word \( w_i \), and \( c_f(w_i) \) are the \( |C| \) of the dense vector of the real value of the elements. The left side context \( c_l(w_i) \) of word \( w_i \) is calculated by formula (3), where \( e(w_{i-1}) \) is the word embedding of word \( w_{i-1} \), which is a dense vector with \( |e| \) real value elements. \( c_f(w_{i-1}) \) is the left side of the context of the previous word \( w_{i-1} \). The left side of the first word in any document uses the same shared parameter \( c_l(w_1) \). \( W^{(l)} \) is the transformation of the hidden layer (context) to the next layer of the hidden layer of the matrix. \( W^{(sl)} \) is a matrix that is used to blend the semantics of the current word into the left context of the next word. \( f \) is a nonlinear activation function. \( c_r(w_i) \) is calculated in a similar manner as shown in the formula (4). \( c_r(w_n) \) is for the right context of the last word in the document.

\[
c_l(w_i) = f(w^{(l)}c_l(w_{i-1}) + w^{(sl)}e(w_{i-1})) \tag{3}
\]

\[
c_r(w_i) = f(w^{(r)}c_r(w_{i+1}) + w^{(sr)}e(w_{i+1})) \tag{4}
\]

We can see from formulas (3) and (4) that the semantics of all the left and right contexts can be captured by the context vector. For example, in Figure 1, \( c_l(w_7) \) encodes the semantics of the left-side context “stroll along the South” along with all texts in previous sentences, and \( c_r(w_7) \) encodes the semantics of the right-side context “affords an ...”. Afterwards, we define the representation of word \( w_i \) in formula (5), which is the concatenation of the left-side context vector \( c_l(w_i) \), the word embedding \( e(w_i) \), and the right-side context vector \( c_r(w_i) \). In this way, with the aid of this contextual information, our model is more powerful, more accurate, and more meaningful (i.e., it uses only part of the textual information) than the traditional neural model which uses only a fixed window of the neural model.

\[
x_i = [c_l(w_i); e(w_i); c_r(w_i)] \tag{5}
\]

Regular structures can take \( c_l \) in all forward-scanned text, and get \( c_r \) in all backward-scanned text. Time complexity is \( O(n) \). We use \( x_i \) to represent \( w_i \) and we apply a linear transformation and hyperbolic tangent activation function to \( x_i \) and then input the result to the next layer.

\[
y_i^{(2)} = \tanh(W^{(2)}x_i + b^{(2)}) \tag{6}
\]

\( y_i^{(2)} \) is a latent semantic vector, in which each semantic factor will be analyzed to find out the most useful one to represent the text.
Text Representation Learning

In our model, convolutional neural networks represent text. In convolutional neural networks, the recurrent structure we mentioned earlier is convoluted.

When all the words are evaluated, we apply a maximum pool level to it.

\[ y^{(3)} = \max_{i=1}^{n} y^{(2)}_i \]  

(7)

Here's the max function to smart elements. The \( k \)th element of \( y^{(3)} \) is the maximum in the \( k \)th element of \( y^{(2)} \).

The pooling layer converts texts of various lengths into a fixed length vector. At the convergence layer, we can capture information throughout the document. There are other types of pool layers, such as the average pool layer [5]. We do not use average pooling here, because only a few words and their combinations are useful in capturing document meaning. The maximum layer attempts to find the most important underlying semantic elements in the document. The convergence layer uses the output of the regular structure as input. The convergence layer’s time complexity is \( O(n) \). The overall model is a cascade of regular structures and a maximum pond layer, so our model’s time complexity is still \( O(n) \).

Our model’s last part is an output layer. As in the traditional neural network, it is defined as:

\[ y^{(4)} = W^{(4)} y^{(3)} + b^{(4)} \]  

(8)

At last, the softmax function is applied to \( y^{(4)} \). It can output data into probabilities.

\[ p_i = \frac{\exp(y^{(4)}_i)}{\sum_{k=1}^{n} \exp(y^{(4)}_k)} \]  

(9)

Training

We define the parameters of all we should train for \( \theta \).

\[ \theta = \{ E, b^{(2)}, b^{(4)}, c_l(w_1), c_r(w_n), W^{(2)}, W^{(4)}, W^{(l)}, W^{(r)}, W^{(sl)}, W^{(sr)} \} \]  

(10)

In detail, the parameters are the word embeddings \( E \in \mathbb{R}^{|e| \times |V|} \), the bias vectors \( b^{(2)} \in \mathbb{R}^H, b^{(4)} \in \mathbb{R}^O \), the initial contexts \( c_l(w_1), c_r(w_n) \in \mathbb{R}^k \) and the transformation matrices \( W^{(2)} \in \mathbb{R}^{H \times (|e|+2|c|)}, W^{(4)} \in \mathbb{R}^{O \times H}, W^{(l)} \in \mathbb{R}^{k \times |k|}, W^{(r)} \in \mathbb{R}^{k \times |k|}, W^{(sl)} \in \mathbb{R}^{|e| \times |c|}, W^{(sr)} \in \mathbb{R}^{|e| \times |c|} \).

Where \( |V| \) is the number of words in vocabulary, \( H \) is the hidden layer size and \( O \) is the number of document types.

The network training goal is to maximize the logarithm of \( \theta \) like:

\[ \theta \rightarrow \sum_{D \in \Delta} \log p(\text{class}_D | D, \theta) \]  

(11)
Where $\Delta$ is the training document set, $\text{class}_D$ is the correct class of document $D$. We use a stochastic gradient descent to optimize training objectives. In each step, we randomly select an example $(D, \text{class}_D)$, and make a gradient step.

$$
\theta \leftarrow \theta + \alpha \frac{\partial \log p(\text{class}_D | D, \theta)}{\partial \theta}
$$

(12)

$\alpha$ is the learning rate.

We use one trick to train phrases, which is widely used when training neural networks with a stochastic gradient. We call initialization parameters from the uniform distribution of the neural network. Maximum or minimum size is equal to the square root of the "fan-in" [9]. The number is the network node of the previous layer in our model. The vector of the layer is divided by the "fan-in".

**EXPERIMENTS**

In this section, we will do some experiments to evaluate our model and show their analysis.

**Dataset**

We consider the corpus (Stanford Sentiment Treebank) of movie review excerpts from the rotten-tomatoes.com website originally collected and published by Pang and Lee (2005). The original dataset includes 10,662 sentences, half of which were considered positive and the other half negative. Each label is extracted from a longer movie review and reflects the writer's overall intention for this review. The normalized, lowercase text is first used to recover, from the original website, the text with capitalization. Remaining HTML tags and sentences that are not in English are deleted. The Stanford Parser is used to parse all 10,662 sentences. [4] uses Amazon Mechanical Turk to label the resulting 215,154 phrases. It transfers the original 25 different values into 2 and 5 classes for classification. Our paper uses this dataset in our experiments. The dataset can be downloaded from http://nlp.stanford.edu/sentiment/.

As each phrase and sentence is labelled in the dataset, we will report the phrase and sentence prediction accuracies in experiments.

**Experiments Settings**

Using Pre-trained Word Vectors Initializing word vectors with those obtained from an unsupervised neural language model is a popular method to improve performance in the absence of a large supervised training set [2] [3] [5]. We use the publicly available word2vec vectors that were trained on 100 billion words from Google News. The vectors have a dimensionality of 300 and were trained using the continuous bag-of-words architecture [10] [11]. Words not present in the set of pre-trained words are initialized randomly. In practice, the initial word vectors always bring better experimental results than those randomly initialized. This advantage has been shown in other scientific research literature.
Hyper-parameters and Training. In our model, there are mainly three kinds of hyper-parameters: the number of hidden layer nodes $n$, the learning rate $\alpha$ for SGD and the coefficient $\lambda$ for regularization items. We choose these hyper-parameters by obtaining the highest level of accuracy on a valid set. We select the number of hidden layer nodes among {50,100,150}, the regularization coefficients among {0.001,0.0001,0.00001} and the learning rate among {0.1,0.01,0.001}. On the movie review dataset for a 2-class classification, we select these hyper-parameters as $n = 100$, $\alpha = 0.01$ and $\lambda = 0.0001$. As for a 5-class classification, we select them as $n = 100$, $\alpha = 0.01$ and $\lambda = 0.001$.

Baseline Methods. We report the performance of the proposed models on sentiment classification tasks, and compare them with that of other competitor baseline systems. We show the related neural network baseline methods as follows: Recursive Neural Network, Matrix-Vector RNN, Recursive Neural Tensor Network, Convolutional Neural Network.

Results and Analysis

We include two types of analyses. The first type includes several large quantitative evaluations on the test set. The second type focuses on two linguistic phenomena that are important in sentiments. For all models, we use the dev set and cross-validate over regularization of the weights, word vector size, as well as learning rate and minibatch size for AdaGrad. We compare this with commonly used methods that use bag of words features with Naive Bayes and SVMs, as well as Naive Bayes with bag of bigram features. We abbreviate these with NB, SVM and biNB. We also compare with a model that averages neural word vectors and ignores word order (VecAvg).

<table>
<thead>
<tr>
<th>Model</th>
<th>Fine-grained</th>
<th>Positive/Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Root</td>
</tr>
<tr>
<td>NB</td>
<td>67.2</td>
<td>41.0</td>
</tr>
<tr>
<td>SVM</td>
<td>64.3</td>
<td>40.7</td>
</tr>
<tr>
<td>BiNB</td>
<td>71.0</td>
<td>41.9</td>
</tr>
<tr>
<td>VecAvg</td>
<td>73.3</td>
<td>32.7</td>
</tr>
<tr>
<td>RNN</td>
<td>79.0</td>
<td>43.2</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>78.7</td>
<td>44.4</td>
</tr>
<tr>
<td>RNTN</td>
<td>80.7</td>
<td>45.7</td>
</tr>
<tr>
<td>CNN</td>
<td>79.3</td>
<td>45.5</td>
</tr>
<tr>
<td>RCNN (this paper)</td>
<td><strong>80.9</strong></td>
<td><strong>45.8</strong></td>
</tr>
<tr>
<td>Model</td>
<td>Accuracy</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Negated Positive</td>
<td>Negated Negative</td>
</tr>
<tr>
<td>BiNB</td>
<td>19.0</td>
<td>27.3</td>
</tr>
<tr>
<td>RNN</td>
<td>33.3</td>
<td>45.5</td>
</tr>
<tr>
<td>MV-RNN</td>
<td>52.4</td>
<td>54.6</td>
</tr>
<tr>
<td>RNTN</td>
<td>71.4</td>
<td>81.8</td>
</tr>
<tr>
<td>BCNN(this paper)</td>
<td>71.7</td>
<td>82.2</td>
</tr>
</tbody>
</table>

Table II shows the accuracy of negation detection. Negated positive is measured as correct sentiment inversions. Negated negative is measured as increases in positive activations. This evaluation is conducted on sentences level of 2-class (negative/positive).

Conclusions and Future Work

We introduced Recurrent Convolutional Neural Networks and did experiments on sentiment classification. Our combination of a new model and data results in a system for single sentence sentiment detection exhibits state-of-the-art performance in positive/negative sentence classification. Aside from this standard setting, the dataset also poses important new challenges and allows for new evaluation metrics. For instance, the RCNN obtains 80.9% accuracy on fine-grained sentiment prediction across all phrases and captures negation of different sentiments and scope more accurately than previous models. These experimental results show the rationality and correctness of our approach.
In the future, the work we may explore includes 1) extending the RCNN to other natural language processing tasks including relation extraction, sentence classification and sentence matching; 2) exploring the algorithm to reduce the number of parameters because our model requires numerous parameters for large datasets.

REFERENCES