Effective Use of Word Order for Text Categorization with Convolutional Neural Networks

Rie Johnson
RJ Research Consulting
Tarrytown, NY, USA
riejohnson@gmail.com

Tong Zhang
Baidu Inc., Beijing, China
Rutgers University, Piscataway, NJ, USA
tzhang@stat.rutgers.edu

本文主要是直接基于文本数据进行建模，并没有像传统的利用词向量进行建模。

Abstract

Convolutional neural network (CNN) is a neural network that can make use of the internal structure of data such as the 2D structure of image data. This paper studies CNN on text categorization to exploit the 1D structure (namely, word order) of text data for accurate prediction. Instead of using low-dimensional word vectors as input as is often done, we directly apply CNN to high-dimensional text data, which leads to directly learning embedding of small text regions for use in classification. In addition to a straightforward adaptation of CNN from image to text, a simple but new variation which employs bag-of-word conversion in the convolution layer is proposed. An extension to combine multiple convolution layers is also explored for higher accuracy. The experiments demonstrate the effectiveness of our approach in comparison with state-of-the-art methods.

1 Introduction

Text categorization is the task of automatically assigning pre-defined categories to documents written in natural languages. Several types of text categorization have been studied, each of which deals with different types of documents and categories, such as topic categorization to detect discussed topics (e.g., sports, politics), spam detection (Sahami et al., 1998), and sentiment classification (Pang et al., 2002; Pang and Lee, 2008; Maas et al., 2011) to determine the sentiment typically in product or movie reviews. A standard approach to text categorization is to represent documents by *bag-of-word vectors*, namely, vectors that indicate which words appear in the documents but do not preserve word order, and use classification models such as SVM.

It has been noted that loss of word order caused by bag-of-word vectors (*bow vectors*) is particularly problematic on sentiment classification. A simple remedy is to use word bi-grams in addition to uni-grams (Blitzer et al., 2007; Glorot et al., 2011; Wang and Manning, 2012). However, use of word n-grams with $n > 1$ on text categorization in general is not always effective; e.g., on topic categorization, simply adding phrases or n-grams is not effective (see, e.g., references in (Tan et al., 2002)).

词袋模型失去了词序信息，这对需要情感分析的文本将会很不利

To benefit from word order on text categorization, we take a different approach, which employs *convolutional neural networks (CNN)* (LeCun et al., 1986). CNN is a neural network that can make use of the internal structure of data such as the *2D structure* of image data through convolution layers, where each computation unit responds to a small region of input data (e.g., a small square of a large image). We apply CNN to text categorization to make use of the *1D structure* (word order) of document data so that each unit in the convolution layer responds to a small region of a document (a sequence of words).

CNN has been very successful on image classification; see e.g., the winning solutions of ImageNet Large Scale Visual Recognition Challenge (Krizhevsky et al., 2012; Szegedy et al., 2014; Russakovsky et al., 2014).

On text, since the work on token-level applications (e.g., POS tagging) by Collobert et al. (2011), CNN has been used in systems for entity search, sentence modeling, word embedding learning, product feature mining, and so on (Xu and Sarikaya, 2013; Gao et al., 2014; Shen et al., 2014; Kalchbrenner et

To appear in NAACL HLT 2015.

al., 2014; Xu et al., 2014; Tang et al., 2014; Weston et al., 2014; Kim, 2014). Notably, in many of these CNN studies on text, the first layer of the network converts words in sentences to *word vectors* by table lookup. The word vectors are either trained as part of CNN training, or fixed to those learned by some other method (e.g., word2vec (Mikolov et al., 2013)) from an additional large corpus. The latter is a form of semi-supervised learning, which we study elsewhere. We are interested in the effectiveness of CNN itself *without aid of additional resources*; therefore, word vectors should be trained as part of network training if word vector lookup is to be done.

A question arises, however, whether word vector lookup in a purely supervised setting is really useful for text categorization. The essence of convolution layers is to *convert text regions of a fixed size (e.g., "am so happy" with size 3) to feature vectors*, as described later. In that sense, a word vector learning layer is a special (and unusual) case of convolution layer with region size one. Why is size one appropriate if bi-grams are more discriminating than unigrams? Hence, we take a different approach. We *directly apply CNN to high-dimensional one-hot vectors*; i.e., we *directly* learn *embedding*[1] of text regions without going through word embedding learning. This approach is made possible by solving the computational issue[2] through efficient handling of high-dimensional sparse data on GPU, and it turned out to have the merits of improving accuracy with fast training/prediction and simplifying the system (fewer hyper-parameters to tune). Our CNN code for text is publicly available on the internet[3].

卷积的大小为3

直接学习词向量，而不是事先训练好的词向量

We study the effectiveness of CNN on text categorization and explain why CNN is suitable for the task. Two types of CNN are tested: *seq-CNN* is a straightforward adaptation of CNN from image to text, and *bow-CNN* is a simple but new variation of CNN that employs bag-of-word conversion in the convolution layer. The experiments show that seq-CNN outperforms bow-CNN on sentiment classification, vice versa on topic classification, and the winner generally outperforms the conventional bag-of-n-gram vector-based methods, as well as previous CNN models for text which are more complex. In particular, to our knowledge, this is the first work that has successfully used word order to improve topic classification performance. A simple extension that combines multiple convolution layers (thus combining multiple types of text region embedding) leads to further improvement. Through empirical analysis, we will show that CNN can make effective use of high-order n-grams when conventional methods fail.

在情感分析上，seq-CNN比bow-CNN表现好，在主题分类上，恰恰相反。

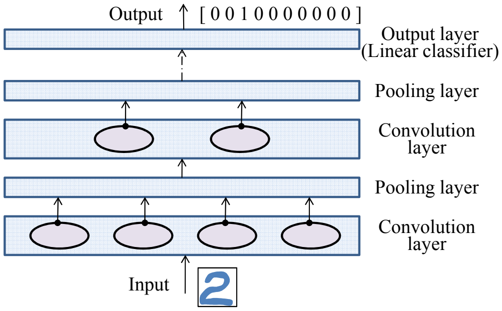


Figure 1: Convolutional neural network.

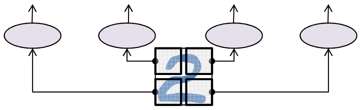


Figure 2: Convolution layer for image. Each computation unit (oval) computes a non-linear function $\boldsymbol{\sigma}(\mathbf{W}\cdot\mathbf{r}_\ell(\mathbf{x})+\mathbf{b})$ of a small region $\mathbf{r}_\ell(\mathbf{x})$ of input image $\mathbf{x}$, where weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ are shared by all the units in the same layer.

2 CNN for document classification

We first review CNN applied to image data and then discuss the application of CNN to document classification tasks to introduce seq-CNN and bow-CNN.

2.1 Preliminary: CNN for image

CNN is a feed-forward neural network with convolution layers interleaved with pooling layers, as illustrated in Figure 1, where the top layer performs classification using the features generated by the layers below. A convolution layer consists of several computation units, each of which takes as input a *region vector* that represents a small region of the input image and applies a non-linear function to it. Typically, the region vector is a concatenation of

[1] We use the term 'embedding' loosely to mean a structure-preserving function, in particular, a function that generates low-dimensional features that preserve the predictive structure.

[2] CNN implemented for image would not handle sparse data efficiently, and without efficient handling of sparse data, convolution over high-dimensional one-hot vectors would be computationally infeasible.

[3] `riejohnson.com/cnn_download.html`

pixels in the region, which would be, for example, 75-dimensional if the region is 5×5 and the number of *channels* is three (red, green, and blue). Conceptually, computation units are placed over the input image so that the entire image is collectively covered, as illustrated in Figure 2. The region stride (distance between the region centers) is often set to a small value such as 1 so that regions overlap with each other, though the stride in Figure 2 is set larger than the region size for illustration.

卷积层的显著特征是权值共享。

A distinguishing feature of convolution layers is *weight sharing*. Given input $\mathbf{x}$, a unit associated with the ℓ-th region computes $\boldsymbol{\sigma}(\mathbf{W} \cdot \mathbf{r}_\ell(\mathbf{x}) + \mathbf{b})$, where $\mathbf{r}_\ell(\mathbf{x})$ is a region vector representing the region of $\mathbf{x}$ at location ℓ, and $\boldsymbol{\sigma}$ is a pre-defined component-wise non-linear activation function, (e.g., applying $\sigma(x) = \max(x, 0)$ to each vector component). The matrix of *weights* $\mathbf{W}$ and the vector of *biases* $\mathbf{b}$ are learned through training, and they are *shared* by the computation units in the same layer. This weight sharing enables learning useful features irrespective of their location, while preserving the location where the useful features appeared.

在同一层，权重W和偏差b是共享的

权重W共享更能够学习到有用的特征，而与他们的位置无关

We regard the output of a convolution layer as an 'image' so that the output of each computation unit is considered to be a 'pixel' of m channels where m is the number of weight vectors (i.e., the number of rows of $\mathbf{W}$) or the number of *neurons*. In other words, *a convolution layer converts image regions to m-dim vectors*, and the locations of the regions are inherited through this conversion.

The output image of the convolution layer is passed to a pooling layer, which essentially shrinks the image by merging neighboring pixels, so that higher layers can deal with more abstract/global information. A pooling layer consists of pooling units, each of which is associated with a small region of the image. Commonly-used merging methods are average-pooling and max-pooling, which respectively compute the channel-wise average/maximum of each region.

2.2 CNN for text

Now we consider application of CNN to text data. Suppose that we are given a document $D = (w_1, w_2, \ldots)$ with vocabulary V. CNN requires vector representation of data that preserves internal locations (word order in this case) as input. A straightforward representation would be to treat each word as a pixel, treat D as if it were an image of $|D| \times 1$ pixels with $|V|$ channels, and to represent each pixel (i.e., each word) as a $|V|$-dimensional one-hot vector[4]. As a running toy example, suppose that vocabulary $V = \{$ "don't", "hate", "I", "it", "love" $\}$ and we associate the words with dimensions of vector in alphabetical order (as shown), and that document D="I love it". Then, we have a document vector:

词典

文本

$$\mathbf{x} = [\,0\;0\;1\;0\;0\,|\,0\;0\;0\;0\;1\,|\,0\;0\;0\;1\;0\,]^\top$$

seq-CNN词向量的构成是将相邻的两个或几个词的one-hot组合成一个词向量，这样相当于利用了词序信息。至于利用几个相邻的词，可以事先确定。本文是用p来表示利用几个词的。

2.2.1 seq-CNN for text

As in the convolution layer for image, we represent each region (which each computation unit responds to) by a concatenation of the pixels, which makes $p|V|$-dimensional region vectors where p is the region size fixed in advance. For example, on the example document vector $\mathbf{x}$ above, with $p = 2$ and stride 1, we would have two regions "I love" and "love it" represented by the following vectors:

$$\mathbf{r}_0(\mathbf{x}) = \left[\begin{array}{c} 0 \\ 0 \\ \mathbf{1} \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{1} \end{array}\right] \begin{array}{l} \text{don't} \\ \text{hate} \\ \mathbf{I} \\ \text{it} \\ \text{love} \\ \\ \text{don't} \\ \text{hate} \\ \text{I} \\ \text{it} \\ \mathbf{love} \end{array} \qquad \mathbf{r}_1(\mathbf{x}) = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ \mathbf{1} \\ \hline 0 \\ 0 \\ 0 \\ \mathbf{1} \\ 0 \end{array}\right] \begin{array}{l} \text{don't} \\ \text{hate} \\ \text{I} \\ \text{it} \\ \mathbf{love} \\ \\ \text{don't} \\ \text{hate} \\ \text{I} \\ \mathbf{it} \\ \text{love} \end{array}$$

The rest is the same as image; *the text region vectors are converted to feature vectors*, i.e., the convolution layer learns to *embed text regions* into low-dimensional vector space. We call a neural net with a convolution layer with this region representation *seq-CNN* ('seq' for keeping sequences of words) to distinguish it from *bow-CNN*, described next.

seq-CNN没有像图像那样，有多通道，而是只有只有一个通道。

2.2.2 bow-CNN for text

A potential problem of seq-CNN however, is that unlike image data with 3 RGB channels, the number of 'channels' $|V|$ (size of vocabulary) may be very large (e.g., 100K), which could make each region vector $\mathbf{r}_\ell(\mathbf{x})$ very high-dimensional if the region size

[4] Alternatively, one could use *bag-of-letter-n-gram vectors* as in (Shen et al., 2014; Gao et al., 2014) to cope with out-of-vocabulary words and typos.

bow-CNN改善了seq-CNN模型，seq-CNN模型是直接将相邻几个词的one-hot连接起来，这样会造成矩阵的维度过高，矩阵过于稀疏等问题。
bow-CNN模型直接将相邻几个词的在词汇表中的位置写入一个向量中。这样矩阵的维度还是V维，不至于过高，而且矩阵也能相对不那么稀疏。

p is large. Since the dimensionality of region vectors determines the dimensionality of weight vectors, having high-dimensional region vectors means more parameters to learn. If $p|V|$ is too large, the model becomes too complex (w.r.t. the amount of training data available) and/or training becomes unaffordably expensive even with efficient handling of sparse data; therefore, one has to lower the dimensionality by lowering the vocabulary size $|V|$ and/or the region size p, which may or may not be desirable, depending on the nature of the task.

An alternative we provide is to perform bag-of-word conversion to make region vectors $|V|$-dimensional instead of $p|V|$-dimensional; e.g., the example region vectors above would be converted to:

$$\mathbf{r}_0(\mathbf{x}) = \begin{array}{c|l} \left[\begin{array}{c} 0 \\ 0 \\ \mathbf{1} \\ 0 \\ \mathbf{1} \end{array}\right] & \begin{array}{l} \text{don't} \\ \text{hate} \\ \textbf{I} \\ \text{it} \\ \textbf{love} \end{array} \end{array} \qquad \mathbf{r}_1(\mathbf{x}) = \begin{array}{c|l} \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \mathbf{1} \\ \mathbf{1} \end{array}\right] & \begin{array}{l} \text{don't} \\ \text{hate} \\ \text{I} \\ \textbf{it} \\ \textbf{love} \end{array} \end{array}$$

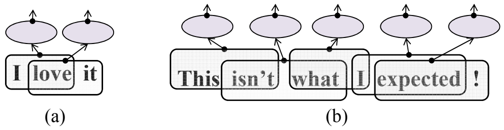
With this representation, we have fewer parameters to learn. Essentially, the expressiveness of bow-convolution (which loses word order only within small regions) is somewhere between seq-convolution and bow vectors.

2.2.3 Pooling for text

对于不同的长短文本，经过同一stride后，其卷积层输出长度是可变的，

Whereas the size of images is fixed in image applications, documents are naturally variable-sized, and therefore, with a fixed stride, the output of a convolution layer is also variable-sized as shown in Figure 3. Given the variable-sized output of the convolution layer, standard pooling for image (which uses a fixed pooling region size and a fixed stride) would produce variable-sized output, which can be passed to another convolution layer. To produce fixed-sized output, which is required by the fully-connected top layer[5], we fix the number of pooling units and dynamically determine the pooling region size on each data point so that the entire data is covered without overlapping.

In the previous CNN work on text, pooling is typically max-pooling over the entire data (i.e., one pooling unit associated with the whole text). The *dynamic k-max pooling* of (Kalchbrenner et al., 2014) for sentence modeling extends it to take the k largest values where k is a function of the sentence length, but it is again over the entire data, and the operation is limited to max-pooling. Our pooling differs in that it is a natural extension of standard pooling for image, in which not only max-pooling but other types can be applied. With multiple pooling units associated with different regions, the top layer can receive locational information (e.g., if there are two pooling units, the features from the first half and last half of a document are distinguished). This turned out to be useful (along with average-pooling) on topic classification, as shown later.



(a) (b)

Figure 3: Convolution layer for variable-sized text.

2.3 CNN vs. bag-of-n-grams

Traditional methods represent each document *entirely* with one bag-of-n-gram vector and then apply a classifier model such as SVM. However, since high-order n-grams are susceptible to data sparsity, use of a large n such as 20 is not only infeasible but also ineffective. Also note that a bag-of-n-gram represents each n-gram by a one-hot vector and ignores the fact that some n-grams share constituent words. By contrast, CNN internally learns *embedding of text regions* (given the consituent words as input) *useful for the intended task*. Consequently, a large n such as 20 can be used especially with the bow-convolution layer, which turned out to be useful on topic classification. A neuron trained to assign a large value to, e.g., “I love” (and a small value to “I hate”) is likely to assign a large value to “we love” (and a small value to “we hate”) as well, *even though “we love” was never seen during training*. We will confirm these points empirically later.

2.4 Extension: parallel CNN

We have described CNN with the simplest network architecture that has one pair of convolution and pooling layers. While this can be extended in several ways (e.g., with deeper layers), in our experiments, we explored *parallel CNN*, which has two or

平行CNN，就是对给定的one-hot，采用不同的卷积方式和pool方式

[5] In this work, the top layer is fully-connected (i.e., each neuron responds to the entire data) as in CNN for image. Alternatively, the top layer could be convolutional so that it can receive variable-sized input, but such CNN would be more complex.

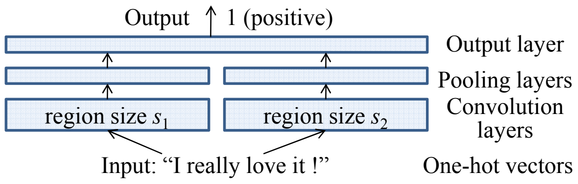


Figure 4: CNN with two convolution layers in parallel.

more convolution layers in parallel[6], as illustrated in Figure 4. The idea is to learn multiple types of embedding of small text regions so that they can complement each other to improve model accuracy. In this architecture, multiple convolution-pooling pairs with different region sizes (and possibly different region vector representations) are given one-hot vectors as input and produce feature vectors for each region; the top layer takes the concatenation of the produced feature vectors as input.

3 Experiments

We experimented with CNN on two tasks, topic classification and sentiment classification. Detailed information for reproducing the results is available on the internet along with our code.

3.1 CNN

We fixed the activation function to rectifier $\sigma(x) = \max(x, 0)$ and minimized square loss with L_2 regularization by *stochastic gradient descent* (SGD). We only used the 30K words that appeared most frequently in the training set; thus, for example, in seq-CNN with region size 3, a region vector is 90K dimensional. Out-of-vocabulary words were represented by a zero vector. On bow-CNN, to speed up computation, we used *variable region stride* so that a larger stride was taken where repetition[7] of the same region vectors can be avoided by doing so. Padding[8] size was fixed to $p - 1$ where p is the region size.

没在词汇表上的单词向量用零表示

We used two techniques commonly used with CNN on image, which typically led to small performance improvements. One is *dropout* (Hinton et al., 2012) optionally applied to the input to the top layer. The other is *response normalization* as in (Krizhevsky et al., 2012), which in our case scales the output of the pooling layer $\mathbf{z}$ at each location by multiplying $(1 + |\mathbf{z}|^2)^{-1/2}$.

3.2 Baseline methods

For comparison, we tested SVM with the linear kernel and fully-connected neural networks (see e.g., Bishop (1995)) with bag-of-n-gram vectors as input. To experiment with fully-connected neural nets, as in CNN, we minimized square loss with L_2 regularization and optional dropout by SGD, and activation was fixed to rectifier. To generate bag-of-n-gram vectors, on topic classification, we first set each component to $\log(x + 1)$ where x is the word frequency in the document and then scaled them to unit vectors, which we found always improved performance over raw frequency. On sentiment classification, as is often done, we generated binary vectors and scaled them to unit vectors. We tested three types of bag-of-n-gram: bow1 with $n \in \{1\}$, bow2 with $n \in \{1, 2\}$, and bow3 with $n \in \{1, 2, 3\}$; that is, bow1 is the traditional bow vectors, and with bow3, each component of the vectors corresponds to either uni-gram, bi-gram, or tri-gram of words.

在文本分类中，首先对词频取对数，然后再对其单位化

在情感分类中，先二值化向量，再单位化。

We used SVMlight[9] for the SVM experiments.

NB-LM We also tested NB-LM, which first appeared (but without performance report[10]) as NB-SVM in WM12 (Wang and Manning, 2012) and later with a small modification produced performance that exceeds state-of-the-art supervised methods on IMDB (which we experimented with) in MMRB14 (Mesnil et al., 2014). We experimented with the MMRB14 version, which generates binary bag-of-n-gram vectors, multiplies the component for each n-gram f_i with $\log(P(f_i|Y = 1)/P(f_i|Y = -1))$ (*NB-weight*) where the probabilities are estimated using the training data, and does logistic regression training. We used MMRB14's software[11] with a modification so that

[6] Similar architectures have been used for image. Kim (2014) used it for text, but it was on top of a word vector conversion layer.

[7] For example, if we slide a window of size 3 over "* * foo * *" where "*" is out of vocabulary, a bag of "foo" will be repeated three times with stride fixed to 1.

[8] As is commonly done, to the beginning and the end of each document, special words that are treated as unknown words (and converted to zero vectors instead of one-hot vectors) were added as 'padding'. The purpose is to equally treat the words at the edge and words in the middle.

[9] `http://svmlight.joachims.org/`

[10] WM12 instead reported the performance of an ensemble of NB and SVM as it performed better.

[11] `https://github.com/mesnilgr/nbsvm`

the regularization parameter can be tuned on development data.

3.3 Model selection

For all the methods, the hyper-parameters such as net configurations and regularization parameters were chosen based on the performance on the development data (held-out portion of the training data), and using the chosen hyper-parameters, the models were re-trained using all the training data.

3.4 Data, tasks, and data preprocessing

IMDB: movie reviews The IMDB dataset (Maas et al., 2011) is a benchmark dataset for sentiment classification. The task is to determine if the movie reviews are positive or negative. Both the training and test sets consist of 25K reviews. For preprocessing, we tokenized the text so that emoticons such as ":-)" are treated as tokens and converted all the characters to lower case.

Elec: electronics product reviews Elec consists of electronic product reviews. It is part of a large Amazon review dataset (McAuley and Leskovec, 2013). We chose electronics as it seemed to be very different from movies. Following the generation of IMDB (Maas et al., 2011), we chose the training set and the test set so that one half of each set consists of positive reviews and the other half is negative, regarding rating 1 and 2 as negative and 4 and 5 as positive, and that the reviewed products are disjoint between the training set and test set. Note that to extract text from the original data, we *only* used the *text section*, and we did *not* use the *summary section*. This way, we obtained a test set of 25K reviews (same as IMDB) and training sets of various sizes. The training and test sets are available on the internet[12]. Data preprocessing was the same as IMDB.

RCV1: topic categorization RCV1 is a corpus of Reuters news articles as described in LYRL04 (Lewis et al., 2004). RCV1 has 103 topic categories in a hierarchy, and one document may be associated with more than one topic. Performance on this task (multi-label categorization) is known to be sensitive to thresholding strategies, which are algorithms additional to the models we would like to test. Therefore, we also experimented with single-label categorization to assign one of 55 second-level topics to each document to directly evaluate models. For this task, we used the documents from a one-month period as the test set and generated various sizes of training sets from the documents with *earlier* dates. Data sizes are shown in Table 1. As in LYRL04, we used the concatenation of the headline and text elements. Data preprocessing was the same as IMDB except that we used the stopword list provided by LYRL04 and regarded numbers as stopwords.

	label	#train	#test	#class
Table 2	single	15,564	49,838	55
Fig. 6	single	varies	49,838	55
Table 4	multi	23,149	781,265	103

Table 1: RCV1 data summary.

3.5 Performance results

Table 2 shows the error rates of CNN in comparison with the baseline methods. The first thing to note is that on all the datasets, the best-performing CNN outperforms the baseline methods, which demonstrates the effectiveness of our approach.

To look into the details, let us first focus on CNN with one convolution layer (seq- and bow-CNN in the table). On sentiment classification (IMDB and Elec), the configuration chosen by model selection was: region size 3, stride 1, 1000 weight vectors, and max-pooling with one pooling unit, for both types of CNN; seq-CNN outperforms bow-CNN, as well as all the baseline methods except for one. Note that with a small region size and max-pooling, if a review contains a short phrase that conveys strong sentiment (e.g., "A great movie!"), the review could receive a high score irrespective of the rest of the review. It is sensible that this type of configuration is effective on sentiment classification.

By contrast, on topic categorization (RCV1), the configuration chosen for bow-CNN by model selection was: region size 20, variable-stride$\geq$2, average-pooling with 10 pooling units, and 1000 weight vectors, which is very different from sentiment classification. This is presumably because on topic classification, a larger context would be more predictive than short fragments (→ larger region size), the entire document matters (→ the effectiveness of average-pooling), and the location of predictive text also matters (→ multiple pooling units). The last

[12] `riejohnson.com/cnn_data.html`

point may be because news documents tend to have crucial sentences (as well as the headline) at the beginning. On this task, while both seq and bow-CNN outperform the baseline methods, bow-CNN outperforms seq-CNN, which indicates that in this setting the merit of having fewer parameters is larger than the benefit of keeping word order in each region.

Now we turn to parallel CNN. On IMDB, seq2-CNN, which has two seq-convolution layers (region size 2 and 3; 1000 neurons each; followed by one unit of max-pooling each), outperforms seq-CNN. With more neurons (3000 neurons each; Table 3) it further exceeds the best-performing baseline, which is also the best previous supervised result. We presume the effectiveness of seq2-CNN indicates that the length of predictive text regions is variable.

The best performance 7.67 on IMDB was obtained by 'seq2-bow$n$-CNN', equipped with three layers in parallel: two seq-convolution layers (1000 neurons each) as in seq2-CNN above and one layer (20 neurons) that *regards the entire document as one region* and represents the region (document) by a bag-of-n-gram vector (bow3) as input to the computation unit; in particular, we generated bow3 vectors by multiplying the NB-weights with binary vectors, motivated by the good performance of NB-LM. This third layer is a bow-convolution layer[13] with one region of variable size that takes one-hot vectors with n-gram vocabulary as input to learn document embedding. The seq2-bow$n$-CNN for Elec in the table is the same except that the regions sizes of seq-convolution layers are 3 and 4. On both datasets, performance is improved over seq2-CNN. The results suggest that what can be learned through these three layers are distinct enough to complement each other. The effectiveness of the third layer indicates that not only short word sequences but also global context in a large window may be useful on this task; thus, inclusion of a bow-convolution layer with n-gram vocabulary with a large fixed region size might be even more effective, providing more focused context, but we did not pursue it in this work.

Baseline methods Comparing the baseline methods with each other, on sentiment classification, reducing the vocabulary to the most frequent n-grams notably hurt performance (also observed on NB-LM and NN) even though some reduction is a common practice. Error rates were clearly improved by addition of bi- and tri-grams. By contrast, on topic categorization, bi-grams only slightly improved accuracy, and reduction of vocabulary did not hurt performance. NB-LM is very strong on IMDB and poor on RCV1; its effectiveness appears to be data-dependent, as also observed by WM12.

methods	IMDB	Elec	RCV1
SVM bow3 (30K)	10.14	9.16	10.68
SVM bow1 (all)	11.36	11.71	10.76
SVM bow2 (all)	9.74	9.05	10.59
SVM bow3 (all)	9.42	8.71	10.69
NN bow3 (all)	9.17	8.48	10.67
NB-LM bow3 (all)	8.13	8.11	13.97
bow-CNN	8.66	8.39	**9.33**
seq-CNN	8.39	7.64	9.96
seq2-CNN	8.04	7.48	–
seq2-bow$n$-CNN	**7.67**	**7.14**	–

Table 2: Error rate (%) comparison with bag-of-n-gram-based methods. Sentiment classification on IMDB and Elec (25K training documents) and 55-way topic categorization on RCV1 (16K training documents). '(30K)' indicates that the 30K most frequent n-grams were used, and '(all)' indicates that all the n-grams (up to 5M) were used. CNN used the 30K most frequent words.

SVM bow2 [WM12]	10.84	–
WRRBM+bow [DAL12]	10.77	–
NB+SVM bow2 [WM12]	8.78	ensemble
NB-LM bow3 [MMRB14]	8.13	–
Paragraph vectors [LM14]	7.46	unlabeled data
seq2-CNN (3K×2) [Ours]	7.94	–
seq2-bow$n$-CNN [Ours]	**7.67**	–

Table 3: Error rate (%) comparison with previous best methods on IMDB.

Comparison with state-of-the-art results As shown in Table 3, the previous best supervised result on IMDB is 8.13 by NB-LM with bow3 (MMRB14), and our best error rate 7.67 is better by nearly 0.5%. (Le and Mikolov, 2014) reports 7.46 with the semi-supervised method that learns low-dimensional vector representations of documents from unlabeled data. Their result is not directly comparable with our supervised results due to use of additional resource. Nevertheless, our best result rivals their result.

We tested bow-CNN on the multi-label topic categorization task on RCV1 to compare with

[13] It can also be regarded as a fully-connected layer that takes bow3 vectors as input.

models	micro-F	macro-F
LYRL04's best SVM	81.6	60.7
bow-CNN	**84.0**	**64.8**

Table 4: RCV1 micro-averaged and macro-averaged F-measure results on multi-label task with LYRL04 split.

LYRL04. We used the same thresholding strategy as LYRL04. As shown in Table 4, bow-CNN outperforms LYRL04's best results even though our data preprocessing is much simpler (no stemming and no tf-idf weighting).

Previous CNN We focus on the sentence classification studies due to its relation to text categorization. Kim (2014) studied fine-tuning of pre-trained word vectors to produce input to parallel CNN. He reported that performance was poor when word vectors were trained as part of CNN training (i.e., no additional method/corpus). On our tasks, we were also unable to outperform the baselines with this type of model. Also, with our approach, a system is simpler with one fewer layer – no need to tune the dimensionality of word vectors or meta-parameters for word vector learning.

Kalchbrenner et al. (2014) proposed complex modifications of CNN for sentence modeling. Notably, given word vectors $\in \mathbb{R}^d$, their convolution with m feature maps produces for each region a matrix $\in \mathbb{R}^{d \times m}$ (instead of a vector $\in \mathbb{R}^m$ as in standard CNN). Using the provided code, we found that their model is too resource-demanding for our tasks. On IMDB and Elec[14] the best error rates we obtained by training with various configurations that fit in memory for 24 hours each on GPU (cf. Fig 5) were 10.13 and 9.37, respectively, which is no better than SVM bow2. Since excellent performances were reported on short sentence classification, we presume that their model is optimized for short sentences, but not for text categorization in general.

Performance dependency CNN training is known to be expensive, compared with, e.g., linear models – linear SVM with bow3 on IMDB only takes 9 minutes using SVMlight (single-core) on a high-end Intel CPU. Nevertheless, with our code on GPU, CNN training only takes minutes (to a few hours) on these datasets shown in Figure 5.

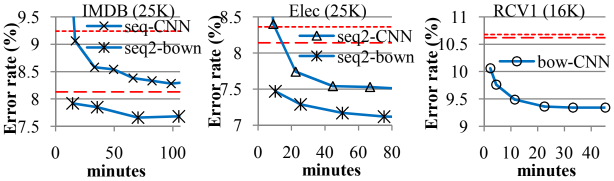


Figure 5: Training time (minutes) on Tesla K20. The horizontal lines are the best-performing baselines.

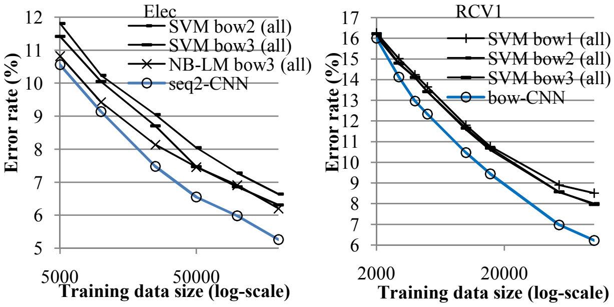


Figure 6: Error rate in relation to training data size. For readability, only representative methods are shown.

Finally, the results with training sets of various sizes on Elec and RCV1 are shown in Figure 6.

3.6 Why is CNN effective?

In this section we explain the effectiveness of CNN through looking into what it learns from training.

First, for comparison, we show the n-grams that SVM with bow3 found to be the most predictive; i.e., the following n-grams were assigned the 10 largest weights by SVM with binary features on Elec for the negative and positive class, respectively:

- poor, useless, returned, not worth, return, worse, disappointed, terrible, worst, horrible
- great, excellent, perfect, love, easy, amazing, awesome, no problems, perfectly, beat

Note that, even though SVM was also given bi- and tri-grams, the top 10 features chosen by SVM with binary features are mostly uni-grams; furthermore, the top 100 features (50 for each class) include 28 bi-grams but only four tri-grams. This means that, with the given size of training data, SVM still heavily counts on uni-grams, which could be ambiguous, and cannot fully take advantage of higher-order n-grams. By contrast, NB-weights tend to promote n-grams with a larger n; the 100 features that were assigned the largest NB-weights are 7 uni-, 33 bi-, and 60 tri-grams. However, as seen above, NB-weights do not always lead to the best performance.

[14] We could not train adequate models on RCV1 on either Tesla K20 or M2070 due to memory shortage.

N1	completely useless ., return policy .
N2	it won’t even, but doesn’t work
N3	product is defective, very disappointing !
N4	is totally unacceptable, is so bad
N5	was very poor, it has failed
P1	works perfectly !, love this product
P2	very pleased !, super easy to, i am pleased
P3	’m so happy, it works perfect, is awesome !
P4	highly recommend it, highly recommended !
P5	am extremely satisfied, is super fast

Table 5: Examples of predictive text regions in the training set.

In Table 5, we show some of text regions learned by seq-CNN to be predictive on Elec. This net has one convolution layer with region size 3 and 1000 neurons; thus, embedding by the convolution layer produces a 1000-dim vector for each region, which (after pooling) serves as features in the top layer where weights are assigned to the 1000 vector components. In the table, Ni/Pi indicates the component that received the i-th highest weight in the top layer for the negative/positive class, respectively. The table shows the text regions (in the training set) whose embedded vectors have a large value in the corresponding component, i.e., predictive text regions.

Note that the embedded vectors for the text regions listed in the same row are close to each other as they have a large value in the same component. That is, Table 5 also shows that the *proximity of the embedded vectors* tends to reflect the *proximity in terms of the relations to the target classes* (positive/negative sentiment). This is the effect of embedding, which helps classification by the top layer.

were unacceptably bad, is abysmally bad, were universally poor, was hugely disappointed, was enormously disappointed, is monumentally frustrating, are endlessly frustrating
best concept ever, best ideas ever, best hub ever, am wholly satisfied, am entirely satisfied, am incredicbly satisfied, ’m overall impressed, am awfully pleased, am exceptionally pleased, ’m entirely happy, are acoustically good, is blindingly fast,

Table 6: Examples of text regions that contribute to prediction. They are from the *test set*, and they did *not* appear in the training set, either entirely or partially as bi-grams.

With the bag-of-n-gram representation, only the n-grams that appear in the training data can participate in prediction. By contrast, one strength of CNN is that n-grams (or text regions of size n) *can contribute to accurate prediction even if they did not appear in the training data*, as long as (some of) their constituent words did, because input of embedding is the constituent words of the region. To see this point, in Table 6 we show the text regions from the *test set*, which *did not appear in the training data*, either entirely or partially as bi-grams, and yet whose embedded features have large values in the heavily-weighted (predictive) component thus contributing to the prediction. There are many more of these, and we only show a small part of them that fit certain patterns. One noticeable pattern is (be-verb, adverb, sentiment adjective) such as “am entirely satisfied” and “’m overall impressed”. These adjectives alone could be ambiguous as they may be negated. To know that the writer is indeed “satisfied”, we need to see the sequence “am satisfied”, but the insertion of adverb such as “entirely” is very common. “best X ever’ is another pattern that a discriminating pair of words are not adjacent to each other. These patterns require tri-grams for disambiguation, and seq-CNN successfully makes use of them even though the exact tri-grams were not seen during training, as a result of learning, e.g., “am X satisfied” with non-negative X (e.g., “am very satisfied”, “am so satisfied”) to be predictive of the positive class through training. That is, CNN can effectively use word order when bag-of-n-gram-based approaches fail.

4 Conclusion

This paper showed that CNN provides an alternative mechanism for effective use of word order for text categorization through direct embedding of small text regions, different from the traditional bag-of-n-gram approach or word-vector CNN. With the parallel CNN framework, several types of embedding can be learned and combined so that they can complement each other for higher accuracy. State-of-the-art performances on sentiment classification and topic classification were achieved using this approach.

Acknowledgements

We thank the anonymous reviewers for useful suggestions. The second author was supported by NSF IIS-1250985 and NSF IIS-1407939.

References

Christopher Bishop. 1995. *Neural networks for pattern recognition*. Oxford University Press.

John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies, bollywood, boom-boxes, and blenders: Domain adaptation for sentiment classification. In *Proceedings of ACL*.

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

Jianfeng Gao, Patric Pantel, Michael Gamon, Xiaodong He, and Li dent. 2014. Modeling interestingness with deep neural networks. In *Proceedings of EMNLP*.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of ICML*.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modeling sentences. In *Proceedings of ACL*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP*, pages 1746–1751.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of NIPS*.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of ICML*.

Yann LeCun, León Bottou, Yoshua Bengio, and Patrick Haffner. 1986. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Marchine Learning Research*, 5:361–397.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of ACL*.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *RecSys*.

Grégoire Mesnil, Tomas Mikolov, Marc'Aurelio Ranzato, and Yoshua Bengio. 2014. Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *arXiv:1412.5335v5 (4 Feb 2015 version)*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NIPS*.

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

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2014. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575*.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. 1998. A bayesian approach to filtering junk e-mail. In *Proceedings of AAAI'98 Workshop on Learning for Text Categorization*.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mensnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of CIKM*.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going deeper with convolutions. *arXiv:1409.4842*.

Chade-Meng Tan, Yuan-Fang Wang, and Chan-Do Lee. 2002. The use of bigrams to enhance text categorization. *Information Processing and Management*, 38:529–546.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of ACL*, pages 1555–1565.

Sida Wang and Christopher D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of ACL (short paper)*.

Jason Weston, Sumit Chopra, and Keith Adams. 2014. #tagspace: Semantic embeddings from hashtags. In *Proceedings of EMNLP*, pages 1822–1827.

Puyang Xu and Ruhi Sarikaya. 2013. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *ASRU*.

Liheng Xu, Kang Liu, Siwei Lai, and Jun Zhao. 2014. Product feature mining: Semantic clues versus syntactic constituents. In *Proceedings of ACL*.