Molding CNNs for text: Non-linear, non-consecutive convolutions

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>As Published</td>
<td><a href="http://www.emnlp2015.org/proceedings/EMNLP/">http://www.emnlp2015.org/proceedings/EMNLP/</a></td>
</tr>
<tr>
<td>Publisher</td>
<td>Association for Computational Linguistics</td>
</tr>
<tr>
<td>Version</td>
<td>Original manuscript</td>
</tr>
<tr>
<td>Accessed</td>
<td>Sun Mar 31 19:53:37 EDT 2019</td>
</tr>
<tr>
<td>Citable Link</td>
<td><a href="http://hdl.handle.net/1721.1/110753">http://hdl.handle.net/1721.1/110753</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>Creative Commons Attribution-Noncommercial-Share Alike</td>
</tr>
<tr>
<td>Detailed Terms</td>
<td><a href="http://creativecommons.org/licenses/by-nc-sa/4.0/">http://creativecommons.org/licenses/by-nc-sa/4.0/</a></td>
</tr>
</tbody>
</table>
Molding CNNs for text: non-linear, non-consecutive convolutions

Tao Lei, Regina Barzilay, and Tommi Jaakkola
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{taolei, regina, tommi}@csail.mit.edu

Abstract

The success of deep learning often derives from well-chosen operational building blocks. In this work, we revise the temporal convolution operation in CNNs to better adapt it to text processing. Instead of concatenating word representations, we appeal to tensor algebra and use low-rank n-gram tensors to directly exploit interactions between words already at the convolution stage. Moreover, we extend the n-gram convolution to non-consecutive words to recognize patterns with intervening words. Through a combination of low-rank tensors, and pattern weighting, we can efficiently evaluate the resulting convolution operation via dynamic programming. We test the resulting architecture on standard sentiment classification and news categorization tasks. Our model achieves state-of-the-art performance both in terms of accuracy and training speed. For instance, we obtain 51.2% accuracy on the fine-grained sentiment classification task.\(^1\)

1 Introduction

Deep learning methods and convolutional neural networks (CNNs) among them have become de facto top performing techniques across a range of NLP tasks such as sentiment classification, question-answering, and semantic parsing. As methods, they require only limited domain knowledge to reach respectable performance with increasing data and computation, yet permit easy architectural and operational variations so as to fine tune them to specific applications to reach top performance. Indeed, their success is often contingent on specific architectural and operational choices.

\(^1\)Our code and data are available at https://github.com/taolei87/text_convnet

CNNs for text applications make use of temporal convolution operators or filters. Similar to image processing, they are applied at multiple resolutions, interspersed with non-linearities and pooling. The convolution operation itself is a linear mapping over “n-gram vectors” obtained by concatenating consecutive word (or character) representations. We argue that this basic building block can be improved in two important respects. First, the power of n-grams derives precisely from multi-way interactions and these are clearly missed (initially) with linear operations on stacked n-gram vectors. Non-linear interactions within a local context have been shown to improve empirical performance in various tasks (Mitchell and Lapata, 2008; Kartsaklis et al., 2012; Socher et al., 2013). Second, many useful patterns are expressed as non-consecutive phrases, such as semantically close multi-word expressions (e.g., “not that good”, “not nearly as good”). In typical CNNs, such expressions would have to come together and emerge as useful patterns after several layers of processing.

We propose to use a feature mapping operation based on tensor products instead of linear operations on stacked vectors. This enables us to directly tap into non-linear interactions between adjacent word feature vectors (Socher et al., 2013; Lei et al., 2014). To offset the accompanying parametric explosion we maintain a low-rank representation of the tensor parameters. Moreover, we show that this feature mapping can be applied to all possible non-consecutive n-grams in the sequence with an exponentially decaying weight depending on the length of the span. Owing to the low rank representation of the tensor, this operation can be performed efficiently in linear time with respect to the sequence length via dynamic programming. Similar to traditional convolution operations, our non-linear feature mapping can be applied successively at multiple levels.
We evaluate the proposed architecture in the context of sentence sentiment classification and news categorization. On the Stanford Sentiment Treebank dataset, our model obtains state-of-the-art performance among a variety of neural networks in terms of both accuracy and training cost. Our model achieves 51.2% accuracy on fine-grained classification and 88.6% on binary classification, outperforming the best published numbers obtained by a deep recursive model (Tai et al., 2015) and a convolutional model (Kim, 2014). On the Chinese news categorization task, our model achieves 80.0% accuracy, while the closest baseline achieves 79.2%.

2 Related Work

Deep neural networks have recently brought about significant advancements in various natural language processing tasks, such as language modeling (Bengio et al., 2003; Mikolov et al., 2010), sentiment analysis (Socher et al., 2013; Iyyer et al., 2015; Le and Zuidema, 2015), syntactic parsing (Collobert and Weston, 2008; Socher et al., 2011a; Chen and Manning, 2014) and machine translation (Bahdanau et al., 2014; Devlin et al., 2014; Sutskever et al., 2014). Models applied in these tasks exhibit significant architectural differences, ranging from recurrent neural networks (Mikolov et al., 2010; Kalchbrenner and Blunsom, 2013) to recursive models (Pollack, 1990; Küchler and Goller, 1996), and including convolutional neural nets (Collobert and Weston, 2008; Collobert et al., 2011; Yih et al., 2014; Shen et al., 2014; Kalchbrenner et al., 2014; Zhang and LeCun, 2015).

Our model most closely relates to the latter. Since these models have originally been developed for computer vision (LeCun et al., 1998), their application to NLP tasks introduced a number of modifications. For instance, Collobert et al. (2011) use the max-over-time pooling operation to aggregate the features over the input sequence. This variant has been successfully applied to semantic parsing (Yih et al., 2014) and information retrieval (Shen et al., 2014; Gao et al., 2014). Kalchbrenner et al. (2014) instead propose (dynamic) k-max pooling operation for modeling sentences. In addition, Kim (2014) combines CNNs of different filter widths and either static or fine-tuned word vectors. In contrast to the traditional CNN models, our method considers non-consecutive n-grams thereby expanding the representation capacity of the model. Moreover, our model captures non-linear interactions within n-gram snippets through the use of tensors, moving beyond direct linear projection operator used in standard CNNs. As our experiments demonstrate these advancements result in improved performance.

3 Background

Let \( x \in R^{L \times d} \) be the input sequence such as a document or sentence. Here \( L \) is the length of the sequence and each \( x_i \in R^d \) is a vector representing the \( i^{th} \) word. The (consecutive) n-gram vector ending at position \( j \) is obtained by simply concatenating the corresponding word vectors

$$ v_j = [x_{j-n+1}; x_{j-n+2}; \cdots; x_j] $$

Out-of-index words are simply set to all zeros.

The traditional convolution operator is parameterized by filter matrix \( m \in R^{d \times h} \) which can be thought of as \( n \) smaller filter matrices applied to each \( x_i \) in vector \( v_j \). The operator maps each n-gram vector \( v_j \) in the input sequence to \( m^T v_j \in R^h \) so that the input sequence \( x \) is transformed into a sequence of feature representations,

$$ [m^T v_1, \cdots, m^T v_L] \in R^{L \times h} $$

The resulting feature values are often passed through non-linearities such as the hyper-tangent (element-wise) as well as aggregated or reduced by “sum-over” or “max-pooling” operations for later (similar stages) of processing.

The overall architecture can be easily modified by replacing the basic n-gram vectors and the convolution operation with other feature mappings. Indeed, we appeal to tensor algebra to introduce a non-linear feature mapping that operates on non-consecutive n-grams.

4 Model

N-gram tensor Typical n-gram feature mappings where concatenated word vectors are mapped linearly to feature coordinates may be insufficient to directly capture relevant information in the n-gram. As a remedy, we replace concatenation with a tensor product. Consider a 3-gram \((x_1, x_2, x_3)\) and the corresponding tensor product \( x_1 \otimes x_2 \otimes x_3 \). The tensor product is a 3-way array of coordinate interactions such that each \( ijk \)
entry of the tensor is given by the product of the corresponding coordinates of the word vectors

\[(x_1 \otimes x_2 \otimes x_3)_{i,j,k} = x_{1i} \cdot x_{2j} \cdot x_{3k}\]

Here \(\otimes\) denotes the tensor product operator. The tensor product of a 2-gram analogously gives a two-way array or matrix \(x_1 \otimes x_2 \in \mathbb{R}^{d \times d}\). The n-gram tensor can be seen as a direct generalization of the typical concatenated vector\(^2\).

**Tensor-based feature mapping** Since each n-gram in the sequence is now expanded into a high-dimensional tensor using tensor products, the set of filters is analogously maintained as high-order tensors. In other words, our filters are linear mappings over the higher dimensional interaction terms rather than the original word coordinates.

Consider again mapping a 3-gram \((x_1, x_2, x_3)\) into a feature representation. Each filter is a 3-way tensor with dimensions \(d \times d \times d\). The set of \(h\) filters, denoted as \(T\), is a 4-way tensor of dimension \(d \times d \times d \times h\), where each \(d^3\) slice of \(T\) represents a single filter and \(h\) is the number of such filters, i.e., the feature dimension. The resulting \(h\)-dimensional feature representation \(z \in \mathbb{R}^h\) for the 3-gram \((x_1, x_2, x_3)\) is obtained by multiplying the filter \(T\) and the 3-gram tensor as follows. The \(l\)th coordinate of \(z\) is given by

\[
z_l = \sum_{ijk} T_{ijkl} \cdot (x_1 \otimes x_2 \otimes x_3)_{ijk}
\]

The formula is equivalent to summing over all the third-order polynomial interaction terms where tensor \(T\) stores the coefficients.

Directly maintaining the filters as full tensors leads to parametric explosion. Indeed, the size of the tensor \(T\) (i.e. \(h \times d^3\)) would be too large even for typical low-dimensional word vectors where, e.g., \(d = 300\). To this end, we assume a low-rank factorization of the tensor \(T\), represented in the Kruskal form. Specifically, \(T\) is decomposed into a sum of \(h\) rank-1 tensors

\[
T = \sum_{i=1}^{h} P_i \otimes Q_i \otimes R_i \otimes O_i
\]

where \(P, Q, R \in \mathbb{R}^{h \times d}\) and \(O \in \mathbb{R}^{h \times h}\) are four smaller parameter matrices. \(P_i\) (similarly \(Q_i, R_i\) and \(O_i\)) denotes the \(i\)th row of the matrix. Note that, for simplicity, we have assumed that the number of rank-1 components in the decomposition is equal to the feature dimension \(h\). Plugging the low-rank factorization into Eq.(1), the feature-mapping can be rewritten in a vector form as

\[
z = O^T (P x_1 \odot Q x_2 \odot R x_3)
\]

where \(\odot\) is the element-wise product such that, e.g., \((a \odot b)_{k} = a_k \times b_k\) for \(a, b \in \mathbb{R}^m\). Note that while \(P x_1\) (similarly \(Q x_2\) and \(R x_3\)) is a linear mapping from each word \(x_1\) (similarly \(x_2\) and \(x_3\)) into a \(h\)-dimensional feature space, higher order terms arise from the element-wise products.

**Non-consecutive n-gram features** Traditional convolution uses consecutive n-grams in the feature map. Non-consecutive n-grams may nevertheless be helpful since phrases such as “not good”, “not so good” and “not nearly as good” express similar sentiments but involve variable spacings between the key words. Variable spacings are not effectively captured by fixed n-grams.

We apply the feature-mapping in a weighted manner to all n-grams thereby gaining access to patterns such as “not ... good”. Let \(z[i, j, k] \in \mathbb{R}^h\) denote the feature representation corresponding to a 3-gram \((x_i, x_j, x_k)\) of words in positions \(i, j, \) and \(k\) along the sequence. This vector is calculated analogously to Eq.(2),

\[
z[i, j, k] = O^T (P x_i \odot Q x_j \odot R x_k)
\]

We will aggregate these vectors into an \(h\)-dimensional feature representation at each position in the sequence. The idea is similar to neural bag-of-words models where the feature representation for a document or sentence is obtained by averaging (or summing) of all the word vectors. In our case, we define the aggregate representation \(z[k]_i\) in position \(k\) as the weighted sum of all 3-gram feature representations ending at position \(k\), i.e.,

\[
z[k]_i = \sum_{i<j<k} z[i, j, k] \cdot \lambda^{(k-j-1)+(j-i-1)}
\]

\[
= \sum_{i<j<k} z[i, j, k] \cdot \lambda^{k-i-2}
\]

where \(\lambda \in [0, 1]\) is a decay factor that down-weights 3-grams with longer spans (i.e., 3-grams

\(x_{i'} = [x_i, 1]\). The tensor product of \(n\) such vectors includes the concatenated vector as a subset of tensor entries but, in addition, contains all up to \(n^6\)-order interaction terms.
that skip more in-between words). As $\lambda \to 0$ all non-consecutive 3-grams are omitted, $z_3[k] = z[k - 2, k - 1, k]$, and the model acts like a traditional model with only consecutive n-grams. When $\lambda > 0$, however, $z_3[k]$ is a weighted average of many 3-grams with variable spans.

**Aggregating features via dynamic programming** Directly calculating $z_3[:i]$ according to Eq.(3) by enumerating all 3-grams would require $O(L^3)$ feature-mapping operations. We can, however, evaluate the features more efficiently by relying on the associative and distributive properties of the feature operation in Eq.(2).

Let $f_3[k]$ be a dynamic programming table representing the sum of 3-gram feature representations before multiplying with matrix $O$. That is, $z_3[k] = O^\top f_3[k]$ or, equivalently,

$$f_3[k] = \sum_{i<j<k} \lambda^{k-i-2} \cdot (P x_i \odot Q x_j \odot R x_k)$$

We can analogously define $f_1[i]$ and $f_2[j]$ for 1-grams and 2-grams,

$$f_1[i] = P x_i$$
$$f_2[j] = \sum_{i<j} \lambda^{j-i-1} \cdot (P x_i \odot Q x_j)$$

These dynamic programming tables can be calculated recursively according to the following formulas:

$$f_1[i] = P x_i$$
$$s_1[i] = \lambda \cdot s_1[i-1] + f_1[i]$$

$$f_2[j] = s_1[j-1] \odot Q x_j$$
$$s_2[j] = \lambda \cdot s_2[j-1] + f_2[j]$$

$$f_3[k] = s_2[k-1] \odot R x_k$$

$$z[k] = O^\top (f_1[k] + f_2[k] + f_3[k])$$

where $s_1[:]$ and $s_2[:]$ are two auxiliary tables. The resulting $z[:]$ is the sum of 1, 2, and 3-gram features. We found that aggregating the 1, 2 and 3-gram features in this manner works better than using 3-gram features alone. Overall, the n-gram feature aggregation can be performed in $O(Ln)$ matrix multiplication/addition operations, and remains linear in the sequence length.

**The overall architecture** The dynamic programming algorithm described above maps the original input sequence to a sequence of feature representations $\mathbf{z} = z[1 : L] \in \mathbb{R}^{L \times h}$. As in standard convolutional architectures, the resulting sequence can be used in multiple ways. One can directly aggregate it to a classifier or expose it to non-linear element-wise transformations and use it as an input to another sequence-to-sequence feature mapping.

The simplest strategy (adopted in neural bag-of-words models) would be to average the feature representations and pass the resulting averaged vector directly to a softmax output unit

$$\tilde{\mathbf{z}} = \frac{1}{L} \sum_{i=1}^{L} \mathbf{z}[i]$$
$$\tilde{\mathbf{y}} = \text{softmax} \left( \mathbf{W}^\top \tilde{\mathbf{z}} \right)$$

Our architecture, as illustrated in Figure 1, includes two additional refinements. First, we add a non-linear activation function after each feature representation, i.e. $\mathbf{z}' = \text{ReLU}(\mathbf{z} + b)$, where $b$ is a bias vector and ReLU is the rectified linear unit function. Second, we stack multiple tensor-based feature mapping layers. That is, the input sequence $\mathbf{x}$ is first processed into a feature sequence and passed through the non-linear transformation to obtain $\mathbf{z}^{(1)}$. The resulting feature sequence $\mathbf{z}^{(1)}$ is then analogously processed by another layer, parameterized by a different set of feature-mapping matrices $\mathbf{P}, \cdots, \mathbf{O}$, to obtain a higher-level feature sequence $\mathbf{z}^{(2)}$, and so on. The output feature representations of all these layers are averaged within each layer and concatenated as shown in Figure 1. The final prediction is therefore obtained on the basis of features across the levels.

5 Learning the Model

Following standard practices, we train our model by minimizing the cross-entropy error on a given training set. For a single training sequence $\mathbf{x}$ and the corresponding gold label $y \in \{0, 1\}^m$, the error is defined as,

$$\text{loss}(\mathbf{x}, y) = \sum_{l=1}^{m} y_l \log(\hat{y}_l)$$

where $m$ is the number of possible output label.

The set of model parameters (e.g. $\mathbf{P}, \cdots, \mathbf{O}$ in each layer) are updated via stochastic gradient
The movie was fantastic!

Figure 1: Illustration of the model architecture. The input is represented as a matrix where each row is a d-dimensional word vector. Several feature map layers (as described in Section 4) are stacked, mapping the input into different levels of feature representations. The features are averaged within each layer and then concatenated. Finally a softmax layer is applied to obtain the prediction output.

descent using AdaGrad algorithm (Duchi et al., 2011).

**Initialization** We initialize matrices $P, Q, R$ from uniform distribution $\left[-\sqrt{\frac{3}{d}}, \sqrt{\frac{3}{d}}\right]$ and similarly $O \sim U \left[-\sqrt{\frac{3}{h}}, \sqrt{\frac{3}{h}}\right]$. In this way, each row of the matrices is an unit vector in expectation, and each rank-1 filter slice has unit variance as well,

$$E \left[\|P_i \otimes Q_i \otimes R_i \otimes O_i\|^2\right] = 1$$

In addition, the parameter matrix $W$ in the softmax output layer is initialized as zeros, and the bias vectors $b$ for ReLU activation units are initialized to a small positive constant 0.01.

**Regularization** We apply two common techniques to avoid overfitting during training. First, we add L2 regularization to all parameter values with the same regularization weight. In addition, we randomly dropout (Hinton et al., 2012) units on the output feature representations $z^{(t)}$ at each level.

6 Experimental Setup

**Datasets** We evaluate our model on sentence sentiment classification task and news categorization task. For sentiment classification, we use the Stanford Sentiment Treebank benchmark (Socher et al., 2013). The dataset consists of 11855 parsed English sentences annotated at both the root (i.e. sentence) level and the phrase level using 5-class fine-grained labels. We use the standard 8544/1101/2210 split for training, development and testing respectively. Following previous work, we also evaluate our model on the binary classification variant of this benchmark, ignoring all neutral sentences. The binary version has 6920/872/1821 sentences for training, development and testing.

For the news categorization task, we evaluate on Sogou Chinese news corpora. The dataset contains 10 different news categories in total, including Finance, Sports, Technology and Automobile etc. We use 79520 documents for training, 9940 for development and 9940 for testing. To obtain Chinese word boundaries, we use LTP-Cloud, an open-source Chinese NLP platform.

**Baselines** We implement the standard SVM method and the neural bag-of-words model NBoW as baseline methods in both tasks. To assess the proposed tensor-based feature map, we also implement a convolutional neural network model CNN by replacing our filter with traditional linear filter. The rest of the framework (such as feature averaging and concatenation) remains the same.

In addition, we compare our model with a wide range of top-performing models on the sentence sentiment classification task. Most of these models fall into either the category of recursive neural networks (RNNs) or the category of convolutional neural networks (CNNs). The recursive neural

---


[https://github.com/HIT-SCIR/ltp](https://github.com/HIT-SCIR/ltp)
<table>
<thead>
<tr>
<th>Model</th>
<th>Fine-grained</th>
<th>Binary</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev</td>
<td>Test</td>
<td>Dev</td>
</tr>
<tr>
<td>RNN</td>
<td>43.2</td>
<td>82.4</td>
<td>-</td>
</tr>
<tr>
<td>RNTN</td>
<td>45.7</td>
<td>85.4</td>
<td>1657</td>
</tr>
<tr>
<td>DRNN</td>
<td>49.8</td>
<td>86.8</td>
<td>431</td>
</tr>
<tr>
<td>RLSTM</td>
<td>51.0</td>
<td>88.0</td>
<td>140</td>
</tr>
<tr>
<td>DCNN</td>
<td>48.5</td>
<td>86.9</td>
<td>-</td>
</tr>
<tr>
<td>CNN-MC</td>
<td>47.4</td>
<td>88.1</td>
<td>2452</td>
</tr>
<tr>
<td>CNN</td>
<td>48.8</td>
<td>47.2</td>
<td>85.7</td>
</tr>
<tr>
<td>PVEC</td>
<td>48.7</td>
<td>87.8</td>
<td>-</td>
</tr>
<tr>
<td>DAN</td>
<td>48.2</td>
<td>86.8</td>
<td>73</td>
</tr>
<tr>
<td>SVM</td>
<td>40.1</td>
<td>38.3</td>
<td>78.6</td>
</tr>
<tr>
<td>NBoW</td>
<td>45.1</td>
<td>44.5</td>
<td>80.7</td>
</tr>
<tr>
<td>Ours</td>
<td>49.5</td>
<td>50.6</td>
<td>87.0</td>
</tr>
<tr>
<td>+ phrase labels</td>
<td>53.4</td>
<td><strong>51.2</strong></td>
<td>88.9</td>
</tr>
</tbody>
</table>

Table 1: Comparison between our model and other baseline methods on Stanford Sentiment Treebank. The top block lists recursive neural network models, the second block are convolutional network models and the third block contains other baseline methods, including the paragraph-vector model (Le and Mikolov, 2014), the deep averaging network model (Iyyer et al., 2015) and our implementation of neural bag-of-words. The training time of baseline methods is taken from (Iyyer et al., 2015) or directly from the authors. For our implementations, timings were performed on a single core of a 2.6GHz Intel i7 processor.

network baselines include standard RNN (Socher et al., 2011b), RNTN with a small core tensor in the composition function (Socher et al., 2013), the deep recursive model DRNN (Irsoy and Cardie, 2014) and the most recent recursive model using long-short-term-memory units RLSTM (Tai et al., 2015). These recursive models assume the input sentences are represented as parse trees. As a benefit, they can readily utilize annotations at the phrase level. In contrast, convolutional neural networks are trained on sequence-level, taking the original sequence and its label as training input. Such convolutional baselines include the dynamic CNN with k-max pooling DCNN (Kalchbrenner et al., 2014) and the convolutional model with multi-channel CNN-MC by Kim (2014). To leverage the phrase-level annotations in the Stanford Sentiment Treebank, all phrases and the corresponding labels are added as separate instances when training the sequence models. We follow this strategy and report results with and without phrase annotations.

Word vectors The word vectors are pre-trained on much larger unannotated corpora to achieve better generalization given limited amount of training data (Turian et al., 2010). In particular, for the English sentiment classification task, we use the publicly available 300-dimensional GloVe word vectors trained on the Common Crawl with 840B tokens (Pennington et al., 2014). This choice of word vectors follows most recent work, such as DAN (Iyyer et al., 2015) and RLSTM (Tai et al., 2015). For Chinese news categorization, there is no widely-used publicly available word vectors. Therefore, we run word2vec (Mikolov et al., 2013) to train 200-dimensional word vectors on the 1.6 million Chinese news articles. Both word vectors are normalized to unit norm (i.e. $\|w\|^2_2 = 1$) and are fixed in the experiments without fine-tuning.

Hyperparameter setting We perform an extensive search on the hyperparameters of our full model, our implementation of the CNN model (with linear filters), and the SVM baseline. For our model and the CNN model, the initial learning rate of AdaGrad is fixed to 0.01 for sentiment classification and 0.1 for news categorization, and the L2 regularization weight is fixed to $1e-5$ and $1e-6$ respectively based on preliminary runs. The rest of the hyperparameters are randomly chosen as follows: number of feature-mapping layers $\in \{1, 2, 3\}$, n-gram order $n \in \{2, 3\}$, hidden feature dimension $h \in \{50, 100, 200\}$, dropout probability $\in \{0.0, 0.1, 0.3, 0.5\}$, and length de-
We run each configuration 3 times to explore different random initializations. For the SVM baseline, we tune L2 regularization weight \( C \in \{ 0.01, 0.1, 1.0, 10.0 \} \), word cut-off frequency \( \in \{ 1, 2, 3, 5 \} \) (i.e. pruning words appearing less than this times) and n-gram feature order \( n \in \{ 1, 2, 3 \} \).

**Implementation details** The source code is implemented in Python using the Theano library (Bergstra et al., 2010), a flexible linear algebra compiler that can optimize user-specified computations (models) with efficient automatic low-level implementations, including (back-propagated) gradient calculation.

## 7 Results

### 7.1 Overall Performance

Table 1 presents the performance of our model and other baseline methods on Stanford Sentiment Treebank benchmark. Our full model obtains the highest accuracy on both the development and test sets. Specifically, it achieves 52.5% and 88.6% test accuracies on fine-grained and binary tasks respectively. As shown in Table 2, our model performance is relatively stable – it remains high accuracies with around 0.5% standard deviation under different initializations and dropout rates.

Our full model is also several times faster than other top-performing models. For example, the convolutional model with multi-channel (CNN-MC) runs over 2400 seconds per training epoch. In contrast, our full model (with 3 feature layers) runs on average 28 seconds with only root labels and on average 445 seconds with all labels.

Our results also show that the CNN model, where our feature map is replaced with traditional linear map, performs worse than our full model. This observation confirms the importance of the proposed non-linear, tensor-based feature mapping. The CNN model also lags behind the DCNN and CNN-MC baselines, since the latter two propose several advancements over standard CNN.

Table 3 reports the results of SVM, NBoW and our model on the news categorization task. Since the dataset is much larger compared to the sentiment dataset (80K documents vs. 8.5K sentences), the SVM method is a competitive baseline. It achieves 78.5% accuracy compared to 74.4% and 79.2% obtained by the neural bag-of-words model and CNN model. In contrast, our model obtains 80.0% accuracy on both the development and test sets, outperforming the three baselines by a 0.8% absolute margin. The best hyperparameter configuration in this task uses less feature layers and lower n-gram order (specifically, 2 layers and \( n = 2 \)) compared to the sentiment classification task. We hypothesize that the difference is due to the nature of the two tasks: the document classification task requires to handle less compositions or context interactions than sentiment analysis.

### 7.2 Hyperparameter Analysis

We next investigate the impact of hyperparameters in our model performance. We use the models trained on fine-grained sentiment classification task with only root labels.

#### Number of layers

We plot the fine-grained sentiment classification accuracies obtained during hyperparameter grid search. Figure 2 illustrates how the number of feature layers impacts the model performance. As shown in the figure, adding higher-level features clearly improves the classification accuracy across various hyperparameter settings and initializations.

#### Non-consecutive n-gram features

We also analyze the effect of modeling non-consecutive n-
Table 1

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.445</td>
<td>0.4692</td>
<td></td>
</tr>
<tr>
<td>0.446</td>
<td>0.4738</td>
<td></td>
</tr>
<tr>
<td>0.4469</td>
<td>0.4769</td>
<td></td>
</tr>
<tr>
<td>0.4478</td>
<td>0.4629</td>
<td></td>
</tr>
<tr>
<td>0.4487</td>
<td>0.4688</td>
<td></td>
</tr>
<tr>
<td>0.4505</td>
<td>0.4861</td>
<td></td>
</tr>
<tr>
<td>0.4877</td>
<td>0.4959</td>
<td></td>
</tr>
<tr>
<td>0.4886</td>
<td>0.4878</td>
<td></td>
</tr>
<tr>
<td>0.4896</td>
<td>0.4602</td>
<td></td>
</tr>
<tr>
<td>0.4896</td>
<td>0.4937</td>
<td></td>
</tr>
<tr>
<td>0.4914</td>
<td>0.4765</td>
<td></td>
</tr>
<tr>
<td>0.4923</td>
<td>0.4986</td>
<td></td>
</tr>
<tr>
<td>0.495</td>
<td>0.5063</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Dev accuracy (x-axis) and test accuracy (y-axis) of independent runs of our model on fine-grained sentiment classification task. Deeper architectures achieve better accuracies.

Figure 3: Comparison of our model variations in sentiment classification task when considering consecutive n-grams only (decaying factor $\lambda = 0$) and when considering non-consecutive n-grams ($\lambda > 0$). Modeling non-consecutive n-gram features leads to better performance.

Figure 5: Example sentences and their sentiments predicted by our model trained with root labels. The predicted sentiment scores at each word position are plotted. Examples (1)-(5) are synthetic inputs, (6) and (7) are two real inputs from the test set. Our model successfully identifies negation, double negation and phrases with different sentiment in one sentence.

Non-linear activation Finally, we verify the effectiveness of rectified linear unit activation function (ReLU) by comparing it with no activation (or identity activation $f(x) = x$). As shown in Figure 4, our model with ReLU activation generally outperforms its variant without ReLU. The observation is consistent with previous work on convolutional neural networks and other neural network models.
7.3 Example Predictions

Figure 5 gives examples of input sentences and the corresponding predictions of our model in fine-grained sentiment classification. To see how our model captures the sentiment at different local context, we apply the learned softmax activation to the extracted features at each position without taking the average. That is, for each index $i$, we obtain the local sentiment $p = \text{softmax} \left( W^T \left( z^{(1)}[i] + z^{(2)}[i] + z^{(3)}[i] \right) \right)$. We plot the expected sentiment scores $\sum_{s=-2}^{2} s \cdot p(s)$, where a score of 2 means “very positive”, 0 means “neutral” and -2 means “very negative”. As shown in the figure, our model successfully learns negation and double negation. The model also identifies positive and negative segments appearing in the sentence.

8 Conclusion

We proposed a feature mapping operator for convolutional neural networks by modeling n-gram interactions based on tensor product and evaluating all non-consecutive n-gram vectors. The associated parameters are maintained as a low-rank tensor, which leads to efficient feature extraction via dynamic programming. The model achieves top performance on standard sentiment classification and document categorization tasks.

Acknowledgments

We thank Kai Sheng Tai, Mohit Iyyer and Jordan Boyd-Graber for answering questions about their paper. We also thank Yoon Kim, the MIT NLP group and the reviewers for their comments. We acknowledge the support of the U.S. Army Research Office under grant number W911NF-10-1-0533. The work is developed in collaboration with the Arabic Language Technologies (ALT) group at Qatar Computing Research Institute (QCRI). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organizations.

References


Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, Li Deng, and Yelong Shen. 2014. Modeling interestingness with deep neural networks. In...


Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. CoRR.


