Transformable Convolutional Neural Network for Text Classification

Anonymous Submission

Abstract

Convolutional neural networks (CNNs) have shown their promising performance for natural language processing tasks, which extract N-gram features with fixed geometric structure. However, N-gram based CNNs are inherently limited to proactively adapt to the transformations of features. In this paper, we introduce two modules, giving CNNs the adaptability for transformation, namely, transformable convolution and transformable pooling. Both modules combine the thoughts of two mainstream end-to-end transformable methods by adding dynamic and static deviations to sampling locations to offset the transformation of features, which is learned from current task without extra supervision. Our modules can be easily employed by other CNN models to generate new transformable convolutional networks. We conduct extensive experiments with our modules, and the results demonstrate the effectiveness of our transformable methods on text classification tasks.

Introduction

CNNs have been proven effective in many nature language processing tasks such as representation learning (Liu et al. 2015), information retrieval (Shen et al. 2014), text classification (Kalchbrenner, Grefenstette, and Blunsom 2014), etc. Convolutional layer in the CNNs extracts features with geometrically fixed filters, which can be regarded as an implementation of N-gram language model.

However, regular CNNs inevitably face the challenge of adapting the feature transformation. This drawback is caused because CNNs have many geometrically fixed modules, such as convolutional layer limiting the filter shape into N-gram, pooling layer selecting the superior features by the solid chunks. These fixed structures render two main limitations for language representation:

- All the features and the chunks are continuous and shape-fixed due to the solid structures, which is sub-optimal. It would be difficult to cover some complex situations such as the discontinuous or oversize ones. In another word, it cannot fit the shape of features.
- Traditional CNNs are unable to actively adjust themselves to adapt to the transformation of features. This is undesirable because the features in different locations has varying shapes.

Typically, there are two ways to cope with above challenges. The first one is to increase the volume of datasets to cover more varying features. But it is a labor-consuming work to build a large-scale dataset. It will also increases the complexity of models, which requires more parameters and training time. The second method is to extract relatively constant features, which already has many successful implementations such as SIFT (Lowe 1999), SURF (Bay, Tuytelaars, and Gool 2006) and ORB (Rublee et al. 2011). However, designing algorithm to extract constant features not only has a strong requirement for sophisticated skills, and may also overlook many complex characteristics.

Recently, in image field, some works (Jia, Huang, and Darrell 2012) (Jaderberg et al. 2015) (Jeon and Kim 2017) (Dai et al. 2017) focus on inherently augmenting the adaptability of CNN models for feature transformation (deformation), which allow the models to reshape some layers to offset the feature transformation. They can be divided into two mainstreams: learning *static* shape parameters through back-propagation to capture only global transformation patterns (Jeon and Kim 2017); learning *dynamical* shape parameters according to the current positions to obtain only local transformation patterns (Dai et al. 2017). Each of these two manners only considers a part of the patterns, and they are not suitable for NLP tasks.

This paper gives consideration to both scenarios and proposes two end-to-end modules for text classification. Both of them can enhance the capability of CNNs for modeling the transformation patterns, namely, *Transformable Convolution* and *Transformable Pooling*. Transformable convolution adds 1-Dimension (1D) deviations to the sampling locations, which can offset the transformation of features. The deviations are partially learned from the feature maps through an additional convolutional layer and partially from back-propagation, without any extra supervision or human experience.

Transformable pooling shares the spirit with transformable convolution. It adds 1D deviations to the chunks in pooling layer to increase the adaptability of selecting essential features in input feature maps. And the deviations are also learned automatically in a similar way with transformable convolution.

Our modules can be easily applied in the state-of-the-art models by displacing their counterparts. The newly formed architectures are lightweight, adding only a few parameters and training time, which are called *Transformable ConvNets* (TF-ConvNets).

We conduct extensive experiments on 17 datasets for text classification. Result shows that, comparing with their plain versions, TF-ConvNets demonstrate great performance gains for these tasks.

In short, the contributions of this paper can be summarized as follows:

- This paper introduces two new modules to replace their counterparts, which can greatly improve the adaptability of CNNs for feature transformation.
- Our modules consider both dynamic and static transformation patterns, which combine the advantages of two mainstream end-to-end methods.
- We successfully implement the transformation thought on NLP task and demonstrate its efficacy and efficiency with extensive experiments.

Transformable Convolution and Pooling

In this section, we introduce the mechanisms of two proposed transformable modules, which have learnable shapes to adapt the varieties of features. Conventionally, the shapes of filters and chunks in convolution and pooling are fixed from the start, while the transformable modules add the learned position deviations onto filters or chunks, making their shapes adaptable. Position deviations are comprised of the dynamic and static parts. The former is current feature related, whose deviations are instantly decided by the current features to capture local transformation knowledge; values for the latter are static and are independent with the current feature, which is designed to model a global transformation pattern.

Transformable Convolution

CNNs for NLP usually utilize more than one filters or channels. For simplicity, we take one filter and channel here for example. In conventional convolution, the shape of filter can be described by the center position p_0 and the sampling locations p_i, which are the relative distances from p_0. We use $\mathbb{A}$ to represent the collection of p_i.

For example, when the length of a filter in conventional convolution is 5,

$$\mathbb{A} = \{-2, -1, 0, 1, 2\}. \tag{1}$$

Output feature map is generated by a convolving operation between weight $\mathbf{W}$ and input $\mathbf{X}$, which can be formulated as Eq.(2):

$$\mathbf{Y} = \mathbf{W} * \mathbf{X}. \tag{2}$$

And every item in feature map $\mathbf{Y}$ can be calculated by:

$$\mathbf{Y}(p_0) = \sum_{p_i \in \mathbb{A}} \mathbf{W}(p_i) \cdot \mathbf{X}(p_0 + p_i), \tag{3}$$

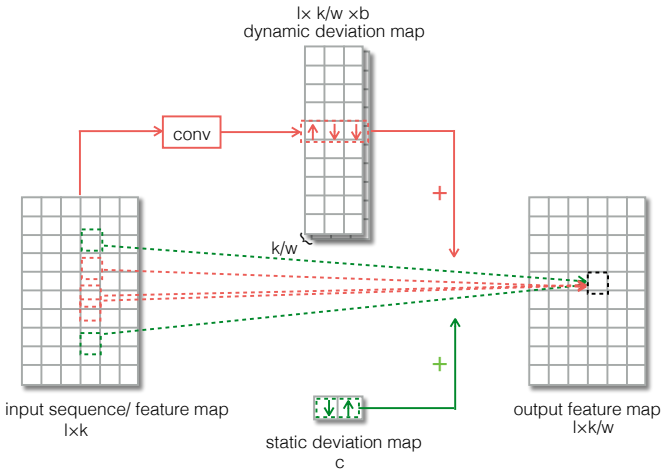


Figure 1: Illustration of Transformable Convolution. l, k, b, c and w represent the length of feature map, width of feature map, size of $\mathbb{B}$, size of $\mathbb{C}$ and the width of filters respectively. The width k of the output feature map is divided by w because convolution for text is 1D.

where p_0 is a location in output feature map and p_i enumerates the points in deviation collection $\mathbb{A}$.

In transformable convolution, $\mathbb{A}$ is divided into two parts, one is current feature related, and the other is globally depended. Let $\mathbb{B}$ and $\mathbb{C}$ denote them respectively. Then they are added with different deviations Δp_i^B and Δp_i^C. And Eq.(3) becomes

$$\begin{aligned}\mathbf{Y}(p_0) = &\sum_{p_i \in \mathbb{B}} \mathbf{W}(p_i) \cdot \mathbf{X}(p_0 + p_i + \Delta p_i^B) \\ &+ \sum_{p_i \in \mathbb{C}} \mathbf{W}(p_i) \cdot \mathbf{X}(p_0 + p_i + \Delta p_i^C).\end{aligned} \tag{4}$$

Now, sampling locations of filters are redistributed, no longer a regular rectangular shape. Considering that Δp_i^B and Δp_i^C are often fractional, value of function $X(\cdot)$ is calculated by bilinear interpolation:

$$\mathbf{X}(p) = \sum_q K(p, q) \cdot \mathbf{X}(q). \tag{5}$$

Here p denotes $p_0 + p_i + \Delta p_i^B$ or $p_0 + p_i + \Delta p_i^C$, q enumerates locations in the whole input feature map, and K is the linear interpolation kernel. In linear space, K can be computed by:

$$K(p, q) = \max(0, 1 - |p - q|), \tag{6}$$

where $\max$ functions as a "mask", limiting the dilation of the interpolation area into 1.

As shown in Figure 1, there are two deviation maps in transformable convolution. The top dynamic one is learned from current input sequence/feature map through a plain convolution layer, whose values are added to regular sampling locations in $\mathbb{B}$. So the filter deforms dynamically according to the current features to fit local transformation

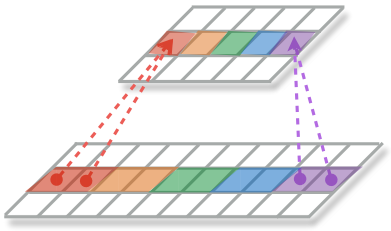


Figure 2: Illustration of a plain version of Chunk-Max Pooling. The input feature map is cut into chunks. And output feature map is constructed by the max values of these chunks.

pattern. The bottom static one is initialized randomly and trained by back-propagation, reaching a convergent value. It can reshape the sampling locations in $\mathbb{C}$ to model the global transformation patterns.

Transformable Pooling

There are many versions of pooling operation, which are used to select the most important features and reduce the resolution. Here we introduce a plain version (Fig. 2) of Chunk-Max Pooling, which has been proven useful to keep the position information of features.

Given a feature map $\mathbf{X}$ of size $l \times k$, Chunk-Max Pooling first divides it into $m \times n (0 < m < l, 0 < n \leq k)$[1] chunks. Then, it outputs a $m \times n$ feature map $\mathbf{Y}$ by concatenating the maximum value in each chunk. Let p_0 denotes top-left corner of (i, j)-th chunk, the process can be formulated as:

$$\mathbf{Y}(p_0) = \max_{p_i \in \mathbb{D}}(\{\mathbf{X}(p_0 + p_i)\}), \tag{7}$$

where p enumerates the sampling locations $\mathbb{D}$ in $chunk(i, j)$, and $i \left\lfloor \frac{l}{m} \right\rfloor < p_i < (i+1) \left\lceil \frac{l}{m} \right\rceil, j \left\lfloor \frac{k}{n} \right\rfloor < p_j < (j+1) \left\lceil \frac{k}{n} \right\rceil$.

Similar to transformable convolution, we separate the sampling locations to $\mathbb{E}, \mathbb{F}$ and add deviations $\Delta p_i^{\mathbb{E}}, \Delta p_i^{\mathbb{F}} \in \Delta\mathbf{p}$ respectively to enhance the adaptability to feature selection. Eq.(7) becomes

$$\mathbf{Y}(p_0) = \max_{p_i^E \in E, p_i^F \in F} (\mathbf{X}(p_0 + p_i^{\mathbb{E}} + \Delta p_i^{\mathbb{E}}) \oplus \mathbf{X}(p_0 + p_i^{\mathbb{F}} + \Delta p_i^{\mathbb{F}})), \tag{8}$$

where $\oplus$ denotes the concatenation operation.

In the same way as Eq.(5) and (6), the value of $\mathbf{X}$ is calculated by 1D bilinear interpolation (Eq.(6)), since the $\Delta\mathbf{p}^{\mathbb{E}}$ and $\Delta\mathbf{p}^{\mathbb{F}}$ are generally fractional.

As illustrated in Fig. 3, the top dynamic deviation map is learned from the input feature map. Chunk-Max Pooling first generates a feature map, on which the fully connected layer emits the normalized deviation $\Delta\hat{\mathbf{p}}$. And static deviation map is relatively stable, whose normalized value $\Delta\hat{\mathbf{p}}^{\mathbb{F}}$

[1]Typically, n equals k in text classification tasks.

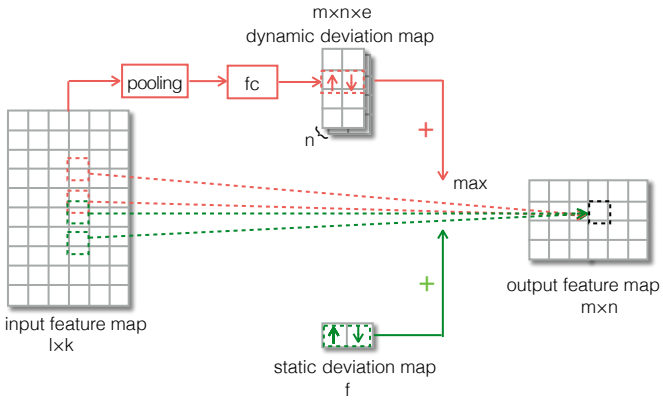


Figure 3: Illustration of transformable pooling. e and f denote the sizes of $\mathbb{E}$ and $\mathbb{F}$ respectively.

is initialized randomly and optimized by back-propagation for the whole dataset.

Then, these normalized deviations are used to calculate the deviations by $\Delta\mathbf{p} = \mu \cdot \Delta\hat{\mathbf{p}} \circ (l, k)$, where l and k are the length and width of input feature map and μ is the coefficient to adjust the scale of deviations. μ is set to 0.1 empirically in this paper. This normalization process is designed keep the deviation scale independent to the size of chunk.

Back-Propagation

The parameters of two modules are easy to learn by back-propagation. Especially for the static part, derivatives of weights $\mathbf{W}$ and deviations Δp_n have to be calculated to update the values. As shown in Eq. (4), weights are separated into two parts, so their derivatives can be differentiated and computed. For the static part corresponding to $\mathbb{C}$,

$$\begin{aligned} \frac{\partial Y(p_0)}{\partial W(p_i)} &= X(p_0 + p_i + \Delta p_i^{\mathbb{C}}) \\ &= \sum_q K(p_0 + p_i + \Delta p_i^{\mathbb{C}}, q) \cdot X(q). \end{aligned} \tag{9}$$

Similarly, for the dynamic part corresponding to $\mathbb{B}$, we just need to change the $\mathbb{C}$ to $\mathbb{B}$.

Since dynamic deviations are learned forward from convolutional layer, so we just need to calculate the gradient of static deviations in Eq. (4). For $\Delta p_i^{\mathbb{C}}$, we have

$$\begin{aligned} \frac{\partial Y(p_0)}{\partial \Delta p_i^{\mathbb{C}}} &= \sum_{p_i \in \mathbb{C}} \mathbf{W}(p_i) \cdot \frac{\partial \mathbf{X}(p_0 + p_n + \Delta p_i^{\mathbb{C}})}{\partial \Delta p_i^{\mathbb{C}}} \\ &= \sum_{p_i \in \mathbb{C}} \mathbf{W}(p_i) \cdot \sum_q \frac{\partial K(q, p_0 + p_n + \Delta p_i^{\mathbb{C}}) \cdot q}{\partial \Delta p_i^{\mathbb{C}}}. \end{aligned} \tag{10}$$

In the same way, for transformable pooling module, the

derivatives of static deviations in Eq. (8) are computed by

$$\frac{\partial Y(p_0)}{\partial \Delta p_i^{\mathbb{F}}} = \frac{\partial \max_{\mathbf{p}^E \in E, p_i^F \in F}(\mathbf{X}(p_0 + p_i^{\mathbb{F}} + \Delta p_i^{\mathbb{F}}), \mathbf{X}(p_0 + p_i^{\mathbb{F}} + \Delta p_i^{\mathbb{F}})}{\partial p_i^{\mathbb{F}}} = \frac{\partial K(q, p_{max}) \cdot q}{\partial p_i^{\mathbb{F}}}, \tag{11}$$

where p_{max} denotes the max value of chunk.

Normalized Gradient

The gradients of deviations in the Eq. (10) and (11) determine the moving speed of the sampling locations. However, they are dependent on other changeable parameters like the weights or inputs. So their gradient values may change drastically, which makes the magnitude of deviations unpredictable. If the deviations are too small, the modules cannot live up to their potential as they are expected. On the contrary, the sampling locations are too dispersed to obtain useful information. Hence, controlling the gradient of deviations is necessary.

In the paper, we use normalized gradient to properly limit the movement of sampling locations. As shown in Eq. (12), normalized gradient is formalized as

$$\overline{\frac{\partial L}{\partial \Delta p}} = \frac{\partial L}{\partial \Delta p} \Big/ \left| \frac{\partial L}{\partial \Delta p} \right|, \tag{12}$$

where L is the loss function and Δp denotes all kinds of deviations. This means we only use the direction of gradient and limited the stride up to 1 in one iteration. Empirically, learning rate is set to 0.001, which indicates that the stride is up to 0.001 in one iteration.

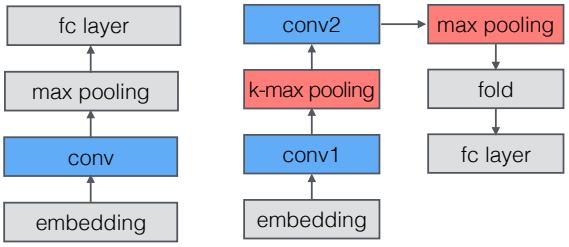
Transformable ConvNets

Our modules are elaborately designed to be compatible for existing models. They can be readily used by replacing their counterparts, since they have the same interfaces with the plain versions and are easy to train with back-propagation algorithm. The thoughts of our work can also be readily transferred to some exotic works. In this section, we implement our modules on two distinct state-of-the-art CNN models for text classification.

MCCNN As Fig. 4(a) shows, Multichannel CNN (MCCNN) is a four layer network proposed by (Kim 2014) to represent text sequences, which only used one convolution layer and pooling layer for feature extraction. Its architecture has been proven to be simple yet practical.

For word embedding, MCCNN exploits two lookup tables to vectorize the input sequence. One is initialized by the pre-trained word vectors that are learned by unsupervised language models to overcome the absence of large-scale supervised training set. The other one has the same size with the former, but is randomly initialized.

For feature extraction, MCCNN first uses a convolutional layer with multiple filters (100×3 filter of 3 different sizes).



(a) MCCNN (b) DCNN

Figure 4: Comparison of two plain CNNs for text classification. (a) We replace only the convolutional layers for TF-ConvNet. (b) We replace both convolutional and pooling layers.

Each filter $\mathbf{w}$ is applied on both embedding results $\mathbf{x}_c$, which are added up to emit the feature:

$$F = \sum_c f(\mathbf{w} * \mathbf{x}_c + b) \tag{13}$$

where f is the rectified linear unit and b is the bias. Then, the 300 output feature maps are put into max-over-time pooling layer to form a 300-dimension feature map.

We improve MCCNN by displacing the convolutional layer to transformable convolution and keeping the rest constant. Since two lookup tables are involved in MCCNN, our dynamic deviation maps are generated from both embedding results by Eq. (13). In another side, the static deviation maps are initialized with random values.

DCNN Dynamic convolutional neural network (DCNN) (Kalchbrenner, Grefenstette, and Blunsom 2014) is a deeper and slimmer text classification model with 7 layers, which is named after its dynamic k-max pooling. As Fig. 4(b) shows, it alternates wide convolutional layers with pooling layers to capture the high-level semantic features. Two convolutional layers use multiple filters of size 2 and 3 respectively. And the maximum widths of chunks in two pooling layers are set to 5 and 3 respectively. It also inserts a folding layer before the second pooling layer to create interaction across dimensions.

To transfer DCNN into a TF-ConvNet, we replace two convolutional and pooling layers with the corresponding transformable modules and keep others same.

Experiments

Datasets

We extensively evaluate our deformable ConvNets on 17 datasets, which are collected in different domains with different targets. Their statistics are listed in Table 2.

The first 14 datasets are all product reviews, which are comprised of Amazon product reviews in 14 domains, including books, DVDs, cameras, etc. These corpora are classified according to the sentiment of positiveness or negativeness. They are derived from the raw data published by (Blitzer, Dredze, and Pereira 2007).

Dataset	Error Rate (%)				Runtime (ms/sample)			
	DCNN	TF-DCNN	MCCNN	TF-MCCNN	DCNN	TF-DCNN	MCCNN	TF-MCCNN
Books	20.3	17.2 (-3.1)	20.1	15.7 (-4.4)	0.192	0.209	0.232	0.288
Electronics	19.6	18.4 (-1.2)	23.0	18.9 (-4.1)	0.151	0.107	0.180	0.205
DVDs	18.4	15.8 (-2.6)	18.8	16.7 (-2.1)	0.178	0.184	0.176	0.177
Kitchen	22.2	19.8 (-2.4)	20.3	18.8 (-1.5)	0.108	0.113	0.109	0.111
Apparel	16.8	15.2 (-1.6)	14.1	14.0 (-0.1)	0.093	0.096	0.112	0.128
Camera	15.1	14.0 (-1.1)	12.9	13.5 (+0.6)	0.185	0.204	0.204	0.235
Health	17.8	14.3 (-3.5)	17.7	14.1 (-3.6)	0.271	0.352	0.290	0.377
Music	23.7	15.7 (-7.0)	23.0	19.6 (-4.4)	0.166	0.196	0.183	0.212
Toys	17.3	16.3 (-1.0)	16.9	12.8 (-4.1)	0.104	0.108	0.105	0.105
Video	18.1	16.8 (-1.3)	16.6	16.2 (-0.4)	0.160	0.172	0.174	0.207
Baby	16.0	12.2 (-3.8)	16.1	15.3 (-0.8)	0.115	0.120	0.128	0.133
Magazines	10.3	10.8 (+0.5)	10.5	11.0 (+0.5)	0.124	0.135	0.135	0.153
Software	15.7	16.5 (+0.8)	15.0	13.3 (-1.7)	0.134	0.142	0.154	0.185
Sports	18.5	16.8 (-1.7)	18.9	16.4 (-2.5)	0.109	0.114	0.123	0.130
RN	16.4	15.7 (-0.7)	16.2	15.8 (-0.4)	0.081	0.095	0.105	0.109
SUBJ	7.0	6.8 (-0.2)	6.8	6.5 (-0.3)	0.113	0.122	0.130	0.140
TREC	7.0	6.4 (-0.4)	7.8	6.2 (-1.6)	0.106	0.110	0.122	0.134
Avg.	**16.5**	**14.6 (-1.9)**	**16.2**	**14.4 (-1.8)**	**0.141**	**0.152**	**0.157**	**0.178**

Table 1: Error rate and running time of the models on 17 benchmarks of text classification. Number in the bracket notes for the improvement relative to plain model.

Dataset	Train	Dev.	Test	Vocab.	Avg.L
Books	1398	200	400	22K	159
Electronics	1398	200	400	11K	111
DVDs	1400	200	400	22K	189
Kitchen	1400	200	400	10K	93
Apparel	1400	200	400	8K	64
Camera	1397	200	400	10K	130
Health	1400	200	400	10K	90
Music	1400	200	400	18K	141
Toys	1400	200	400	10K	98
Video	1400	200	400	20K	157
Baby	1300	200	400	9K	173
Magazines	1370	200	400	12K	123
Software	1315	200	400	11K	140
Sports	1400	200	400	11K	102
RN	7860	1122	2246	29K	147
SUBJ	8000	1000	1000	21K	23
TREC	4907	545	500	10K	10

Table 2: Statistics of datasets. Columns 2-4 show the number of samples in training, development and test sets. Vocab. and Avg.L denote the vocabulary size and average length of the dataset.

The rest 3 datasets are RN, SUBJ and TREC. RN is a dataset about news topic classification, which is collected from Reuters Newswire and published by (Velasco et al. 1994); SUBJ[2] is a subjectivity dataset, whose task is to classify a sentence level text as being subjective or objective (Pang et al. 2004); TREC[3] dataset has the task of classifying a question into 6 types (the questions are about location, person, numeric information, etc.) (Li and Roth 2002)

Hyperparameters

For TF-MCCNN, as original, we use word vectors from 'Word2Vec'[4] that is trained by bag-of-words model on 100B Google News corpus (Mikolov et al. 2013). For structure, TF-MCCNN uses three different lengths of $4, 5, 6$ for every 100 filters. Correspondingly, the sizes of sampling locations for $\mathbb{B}$ and $\mathbb{C}$ are set to $2, 3, 4$ and $2, 2, 2$ respectively. For training, all involved parameters are randomly initialized from a truncated normal distribution with zero mean and standard deviation. And the learning rates are $10^{-3}, 10^{-4}$ for the first $\frac{2}{3}$ and last $\frac{1}{3}$ iterations. Mini-batch is generated by randomly selected 50 samples every time from the corpus. Training is implement through SGD with the Adadelta update rule (Zeiler 2012).

For TF-DCNN, both word vectors and other parameter are normally initialized in [-0.1, 0.1]. The filter widths in transformable convolutional layer are $4, 5$ with $|\mathbb{B}| = 2, 3$ and $|\mathbb{A}| = 2, 2$; The shapes of chunks in two transformable pooling layers are $7 \times 1, 5 \times 1$ with $|\mathbb{B}| = 5, 3; |\mathbb{A}| = 2, 2$. For training, the network is trained with mini-batches of size 50 by back-propagation and the gradient-based optimization is performed using the Adagrad update rule (Duchi, Hazan, and Singer 2011). Learning rates are also set to $10^{-3}, 10^{-4}$ for the first $\frac{2}{3}$ and last $\frac{1}{3}$ iterations.

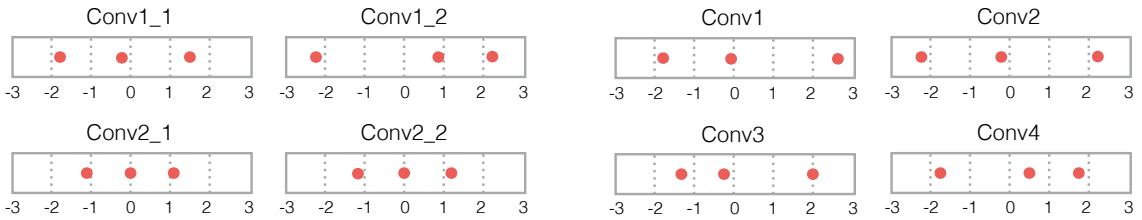
Performance

Table 1 illustrates the error rate and runtime of four models on 17 datasets for text classification. The column of "Error Rate" shows the performance of two baseline models

[2]http://www.cs.cornell.edu/people/pabo/movie-review-data/

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

[4]https://code.google.com/p/word2vec/



(a) Learned static sampling locations in TF-DCNN (b) Learned static sampling locations in TF-MCCNN

Figure 5: Visualization of the static part of filters. Red points denote the learned sampling locations of the filters. (a) Each row illustrates one transformable layer with two filters in TF-DCNN. Note that low layer dilates more than high layer. (b) Four randomly selected filters in TF-MCCNN.

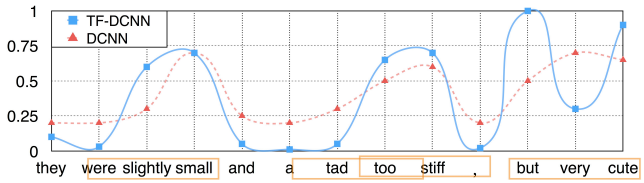


Figure 6: Adaptive phenomenon in transformable modules. X-axis represents input sequence, and Y-axis represents the normalized values of activation units. Orange boxes represent the positions of chunks.

(vanilla DCNN and MCCNN) and two TF-ConvNets that employ transformable convolution and pooling.

From the table, we can see that our baseline networks DCNN and MCCNN obtain the average error rates of 16.5% and 16.2% respectively. When we change the plain layers to transformable modules, there are obvious improvements on most datasets over the previous works. More concretely, with transformable modules equipped, TF-DCNN obtains an average improvement of 1.9%, and TF-MCCNN achieves an average improvement of 1.8%. The great performance on extensive datasets with different targets demonstrates the efficacy of our both modules.

Complexity and Running time

Considering the great improvement obtained, the increase on complexity and runtime of modules are light.

The column of "Runtime" in Fig. 1 reports the runtime[5] of transformable ConvNets against their counterparts. From the table, we know that transformable modules only add a small overhead over computation. These statistics indicate that the advantages of transformable ConvNets are from their elaborate architectures, rather than increasing the volume of parameters.

Visualization

For static deviation map, its values are learned via back-propagation and finally reach constringency. Figure 5 illustrates the final shapes learned by solid part of filters. From extensive experiments, we can find some regular patterns. In low layer, the filers tend to dilate. But in higher layer, the filters are similar to conventional convolution. We think that filters in low layers expand to capture longer features, while the higher layers focus on building connection between close features.

To intuitively understand how the transformable ConvNet works compared with the models of plain version. We design an experiment to show what patterns the models capture and what key features the models focus on. We first randomly extract a sentence from the test set of Baby dataset and feed it into TF-DCNN. Then, for the first transformable convolution layer, we visualize its activation units with normalized values. Figure 6 visually shows the effect of transformable convolution for a positive sentence " they were slightly small and a tad too stiff . but very cute" with a smooth line. We can have a clear observation that transformable module focuses more on key words and strides over the words with little information. This is because transformable convolution has the ability to move their sampling locations adaptively. So it sensitively captures the crucial structure of "but...cute" and makes a right prediction. However the red line for plain model is more flat, which brings more noise into the final score. It only focuses on the word "every" and other negative words like "small" and "stiff", which misleads it to a negative prediction.

For the transformable pooling, the phenomenon of adaptation is similar. As Figure 6 shows, the chunks (orange boxes) no longer stick to the regular grids. They actively adjust the positions according to the current features and move to the nearby areas where meaningful word groups exist.

Ablation Study

In this section, we conduct more investigations to study the independent effect of each module and the influence of transformable convolution number.

Effect of Each Module To prove the effectiveness of both modules, we do some ablation experiments on TF-DCNN since it uses both modules. As Table 3 shows, when transformable convolution is applied on the whole network, an average performance improvement of 1.6% is obtained on 17 datasets. With transformable pooling only, it performs a boost of average 0.7%. This demonstrates that both modules

[5]Runtime is counted on a workstation with Intel E5-2630 v3 CPU and Nvidia K20 GPU.

Usage of Modules	Amazon	RN	SUBJ	TREC	**Avg.**
DCNN	17.8	16.4	7.0	7.0	**16.5**
+TF-conv	16.0	15.9	6.8	6.4	**14.9**
+TF-pooling	17.1	16.2	6.8	6.8	**15.8**
+Both	15.7	15.7	6.8	6.4	**14.6**
DCNN(0,0)	17.8	16.4	7.0	7.0	**16.5**
DCNN(2,0)	16.2	16.0	6.9	6.5	**15.1**
DCNN(0,2)	16.9	16.2	6.8	6.8	**15.7**
DCNN(2,2)	16.0	16.0	6.8	6.4	**14.9**
MCCNN(0)	17.4	16.2	6.8	7.8	**16.2**
MCCNN(10)	15.7	15.9	6.6	6.4	**14.6**
MCCNN(20)	15.5	15.8	6.5	6.2	**14.4**

Table 3: Ablation study of transformable ConvNets. Amazon column reports the average error rates of 14 datasets from amazon product reviews. The tuples following DCNN represent numbers of used transformable convolution in the first and second convolutional layer. The numbers following MCCNN represent the numbers of transformable convolution used in MCCNN.

make a contribution to the final promising scores.

Number of Transformable Convolution On both ConvNets, we implements a series of incremental experiments to study the influence of transformable convolution number on performance. The results reported in Table 3 indicates that 1) Transformable convolution shows more improvement in lower layers. Because it is mainly designed for feature extraction, while it also works on feature fusion in high level. This concurs with the phenomenon that filter shapes change more in lower layers (Figure 5(a)). 2) Performance steadily improves when more transformable convolution layers are involved until a point that improvement saturates. Specifically, the improvement saturates when 20 transformable layers used in MCCNN.

Related Work

CNN inherently suffers from the limitation to model the transformation of objects because of its fixed geometric structure. Most of the works focus on increasing the dataset to cover more variance of features or composing layers, rather than the inner problem. Some works have been done to augment the actively transformable (deformable) ability of CNN by revising the convolutional or pooling layers, which can be classified into two categories: **semi-end-to-end** models and **full-end-to-end** models.

For models in the first category, they generally reshape the convolution or pooling layers with the patterns manually made by experts. For example, spacial transform Network (STN) (Jaderberg et al. 2015) tries to learn the transformable parameters, by which STN warps the the inputs, such as affine transformation. However, the parameters and warping process are global, which are hard to handle some complex or local transformation that simple parameters cannot describe.

Atrous Convolution (Holschneider et al. 1988) improves the convolution by scattering filter' sampling locations with the same strides, which increases the receptive field of the filters with the same parameter complexity. It has been applied in a lot of image tasks and obtains promising improvements. It can be regard as a special case of transformable convolution, which has a more free distribution of sampling locations.

Receptive Field Learning (Jia, Huang, and Darrell 2012) is designed to learn the shapes of the pooling region, which is based on the work of (Lazebnik, Schmid, and Ponce 2006). It learns a subset of pooling regions via the idea of over-complete from a large number of candidates. However, the candidates are handcrafted and not learned end-to-end.

For the second category, models learn the sampling location of modules autonomously and the final shapes are not restricted by manually made patterns. That means this kind of models is more general and fully end-to-end, which can adapt to more intricate transformations. To be specific, Active convolution (Jeon and Kim 2017) learns the free shapes of filters in convolution with a few extra parameters, which can adjust itself according the global datasets via back-propagation. The shapes of filters are static lacking variance for temporary input and sampling location, which mean the transformations it learn are too global and general.

Another end-to-end model Deformable CNN (Dai et al. 2017) attempts to model the transformation through learning the offsets for convolution filters and pooling regions. It also provides freedom to shapes for both modules, while the offsets drastically vary with current input and sampling location. This makes it highly fit the current field losing outline and consistent features for the integral task.

In this paper, we overcome the drawbacks of above two types of works by fusing the thoughts of them. It is noteworthy that our work successfully apples the transformation idea to NLP field and demonstrates its effectiveness. Our two modules separate the sampling locations of filters and chunks of pooling into stochastic and solid part respectively. They separately model the global and local transformation in an easily trainable method.

Conclusion and Future work

In this paper, we present two transformable modules for CNN and transfer the transformation thought to NLP field. Our modules give consideration to both global and local transformation patterns by separating the sampling locations or pooling region into two parts, which are learned in dynamic and static manner respectively. We equip two modules on two previous works and perform extensive experiments of text classification on them, which demonstrates our work is end-to-end, easily trainable and compatible for conventional models.

Dynamic and static transformable modules are two different methods that can enhance each other in a complementary way. In this study, we introduce a separative way to fuse them up. There may be other ways to combine them more organically, such as simply summing up or interacting through an intermediate matrix. In future work, we would like to try more possibilities out and investigate the mutual influence between two methods.

References

Bay, H.; Tuytelaars, T.; and Gool, L. J. V. 2006. SURF: speeded up robust features. In *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part I*, 404–417.

Blitzer, J.; Dredze, M.; and Pereira, F. 2007. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*.

Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; and Wei, Y. 2017. Deformable convolutional networks. *CoRR* abs/1703.06211.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. JMLR.org.

Holschneider, M.; Kronland-Martinet, R.; Morlet, J.; and Tchamitchian, P. 1988. A real-time algorithm for signal analysis with the help of the wavelet transform. 286–297.

Jaderberg, M.; Simonyan, K.; Zisserman, A.; and Kavukcuoglu, K. 2015. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2017–2025.

Jeon, Y., and Kim, J. 2017. Active convolution: Learning the shape of convolution for image classification. *CoRR* abs/1703.09076.

Jia, Y.; Huang, C.; and Darrell, T. 2012. Beyond spatial pyramids: Receptive field learning for pooled image features. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, 3370–3377.

Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, 655–665.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1746–1751.

Lazebnik, S.; Schmid, C.; and Ponce, J. 2006. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*, 2169–2178.

Li, X., and Roth, D. 2002. Learning question classifiers. *Coling* 12(24):556–562.

Liu, X.; Gao, J.; He, X.; Deng, L.; Duh, K.; and Wang, Y. Y. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 912–921.

Lowe, D. G. 1999. Object recognition from local scale-invariant features. In *ICCV*, 1150–1157.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems* 26:3111–3119.

Pang; Bo; Lee; and Lillian. 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. *Proceedings of Acl* 271–278.

Rublee, E.; Rabaud, V.; Konolige, K.; and Bradski, G. R. 2011. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, 2564–2571.

Shen, Y.; He, X.; Gao, J.; Deng, L.; and Mesnil, G. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, 101–110.

Velasco, E.; Thuler, L. C.; Martins, C. A.; Dias, L. M.; and Gonalves, V. M. 1994. Automated learning of decision rules for text categorization. *Acm Transactions on Information Systems* 12(3):233–251.

Zeiler, M. D. 2012. Adadelta: An adaptive learning rate method. *Computer Science*.