Multi-Bias Non-linear Activation in Deep Neural Networks

Hongyang Li   YANGLI@EE.CUHK.EDU.HK
Wanli Ouyang   WLOUYANG@EE.CUHK.EDU.HK
Xiaogang Wang   XGWANG@EE.CUHK.EDU.HK
The Chinese University of Hong Kong

Abstract

As a widely used non-linear activation, Rectified Linear Unit (ReLU) separates noise and signal in a feature map by learning a threshold or bias. However, we argue that the classification of noise and signal not only depends on the magnitude of responses, but also the context of how the feature responses would be used to detect more abstract patterns in higher layers. In order to output multiple response maps with magnitude in different ranges for a particular visual pattern, existing networks employing ReLU and its variants have to learn a large number of redundant filters. In this paper, we propose a multi-bias non-linear activation (MBA) layer to explore the information hidden in the magnitudes of responses. It is placed after the convolution layer to decouple the responses to a convolution kernel into multiple maps by multi-thresholding magnitudes, thus generating more patterns in the feature space at a low computational cost. It provides great flexibility of selecting responses to different visual patterns in different magnitude ranges to form rich representations in higher layers. Such a simple and yet effective scheme achieves the state-of-the-art performance on several benchmarks.

1. Introduction

Deep neural networks (Hinton & Salakhutdinov, 2006; Krizhevsky et al., 2012) have made great progress on different domains and applications in recent years. The community has witnessed the machine trained with deep networks and massive data being the first computer program defeating a European professional in the game of Go (Silver et al., 2016); the convolutional neural network surpassing human-level performance in image classification (He
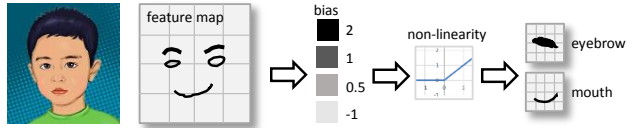
Figure 1. Example illustrating how biases can select different patterns. The eyes and mouth have different magnitudes in their responses to the curved edges. Whether responses of various strength should be considered as informative signal or noise depends on more abstract patterns to be detected in higher layers. By adding different biases to the map with MBA, they are strengthened, preserved, or eliminated according to each bias.

et al., 2015); the deep neural network framework to build an acoustic model in speech recognition (Hinton et al., 2012).

The importance of activation function has been recognized in the design of deep models. It not only fits complex data distributions but also achieves important invariance to various noise, data corruption and transforms affecting recognition (Bruna & Mallat, 2013). It is placed after every convolution layer with sigmoid or hyperbolic tangent as nonlinear activation. If the input responses are too large positively or negatively, they are compressed to a saturation value through the nonlinear mapping and thus invariance is achieved. The rectified linear unit (ReLU) is found to be particularly effective (Nair & Hinton, 2010; Krizhevsky et al., 2012) in deep neural networks and widely used. It separates noisy signals and informative signals in a feature map by learning a threshold (bias). Certain amount of information is discarded after the non-linear activation. However, ReLU also has limitation because of the observations.

Given the same convolution kernel in the convolution layer, we observe that different magnitudes of responses may indicate different patterns. An illustrative example is shown in Figure 1. The eyes should have higher magnitudes in the responses to curved edges compared with those on the mouth because edge contrast on eyes is generally higher than that on mouths. Therefore, it is desirable to separate the responses according to its magnitude.

More importantly, the separation between informative sig-

nal and noise not only depends on the magnitudes of responses, but also the context of how the feature responses would be used to detect more abstract patterns in higher layers. In the hierarchical representations of deep neural networks, a pattern detected at the current layer serves as a sub-pattern to be combined into a more abstract pattern in its subsequent layers. For example, curved edges in Figure 1 are detected in the current layer and one of filters in its subsequent layer detects eyes. It requires high response of curved edges and treat the response with moderate magnitude as noise. However, for another filter in its subsequent layer to detect mouth, moderate responses are enough.

Unfortunately, if feature responses with ReLU module are removed by thresholding, they cannot be recovered. A single thresholding cannot serve for multiple purposes. In order to output multiple response maps with magnitudes in different ranges for a particular visual pattern, networks employing ReLU have to learn multiple redundant filters. A set of redundant filters learn similar convolution kernels but distinct bias terms. It unnecessarily increases the computation cost and model complexity, and is easier to overfit training data.

Many delicate activation functions have been proposed to increase the flexibility of nonlinear function. Parametric ReLU (He et al., 2015) generalizes Leaky ReLU (Mass et al., 2010) by learning the slope of the negative input, which yields an impressive learning behavior on large-scale image classification benchmark. Other variants in the ReLU family include Randomized Leaky ReLU (Xu et al., 2015) where the slope of the negative input is randomly sampled, and Exponential Linear Unit (Clevert et al., 2015) which has an exponential shape in the negative part and ensures a noise-robust deactivation state. Although these variants can reweight feature responses whose magnitudes are in different ranges, they cannot separate them into different feature maps.

As summarized in Figure 2, given a feature map as input, non-linear activation ReLU and its variants only output a single feature map. However, the MBA module outputs multiple feature maps without having to learn as many kernels as does ReLU. In some sense, our idea is opposite to the maxout network (Goodfellow et al., 2013). Maxout is also a non-linear activation. However, given K feature maps generated by K convolution kernels, it combines them to a single feature map:

$$h(\mathbf{x}) = \arg \max_{i \in \{1, \cdots, K\}} \mathbf{w}_i \mathbf{x} + \mathbf{b}_i,$$

where $\mathbf{x} \in \mathbb{R}^d$ is an image, $\mathbf{w}_i \in \mathbb{R}^{d \times p \times K}$ is a convolution kernel, and $b_i \in \mathbb{R}^{p \times K}$ is a bias. The motivations of MBA and maxout are different and can be jointly used in the network design to balance the number of feature maps.
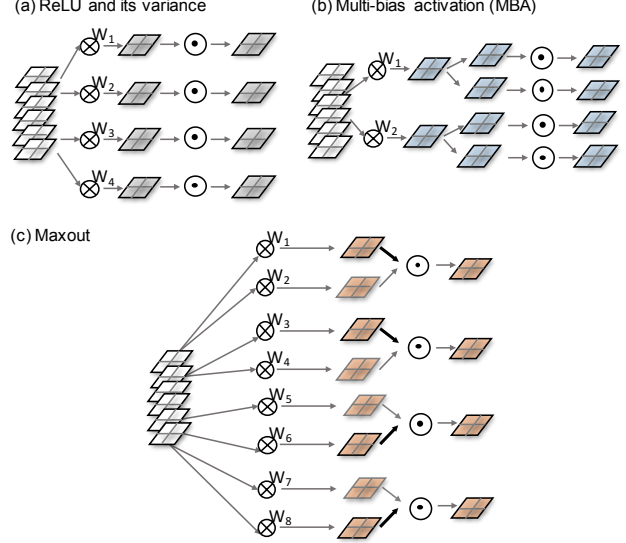

Figure 2. A comparison of MBA with ReLU and its variants, and maxout. (a) Given a feature map as input, ReLU and its variants output a single feature map. In order to decouple the responses to a visual pattern to multiple magnitude ranges, the network has to learn multiple filters which have similar convolution kernels $\{\mathbf{W}_k\}$ and distinct biases. (b) MBA takes one feature map as input and output multiple band maps by introducing multiple biases. It does not need to learn redundant convolution kernels. The decoupling of signal strength increase the expressive power of the net. (c) Maxout combines multiple input feature maps to a single output map. $\otimes$ denotes convolution operation and $\odot$ denotes non-linear activation.

To this end, we propose a multi-bias non-linear activation (MBA) layer for deep neural networks. It decouples a feature map obtained from a convolution kernel to multiple maps, called *band maps*, according to the magnitudes of responses. This is implemented by introducing different biases, which share the *same* convolution kernel, imposed on the feature maps and then followed by the standard ReLU. Each decoupled band map corresponds to a range in the response magnitudes to a convolution kernel, and the range is learned. The responses in different magnitude ranges in the current layer are selected and combined in a flexible way by each filter in the subsequent layer. We provide analysis on the effect of the MBA module when taking its subsequent layer into account. Moreover, it is shown that the piece-wise linear activation function (Agostinelli et al., 2015) is a special case of the proposed module, where MBA provides more flexibility in decoupling the magnitudes of signals and combining band maps from different convolution kernels. The experimental results on the CIFAR and SVHN datasets show that such a simple and yet effective algorithm can achieve state-of-the-art performance. Finally, the implementation code is available at https://github.com/hli2020/caffe/tree/bias.
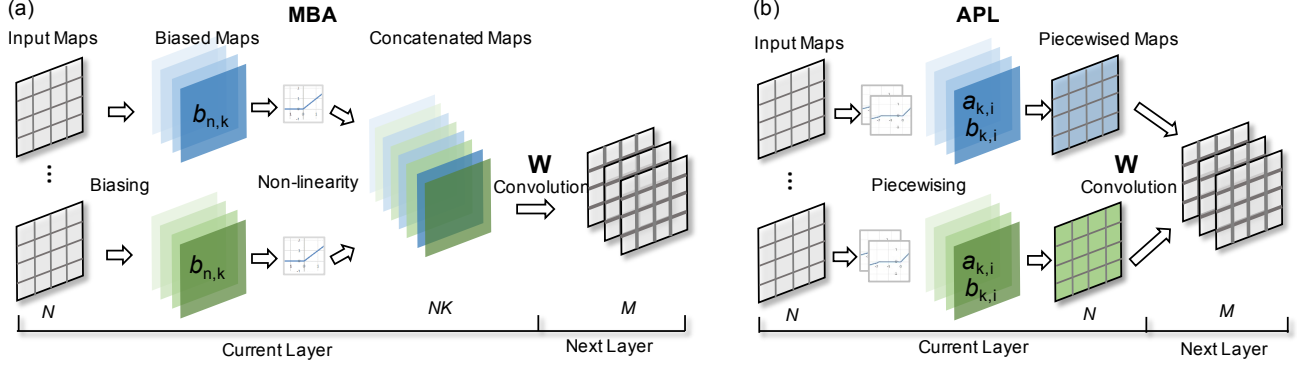
Figure 3. (a) The proposed multi-bias activation (MBA) model. Given the input feature maps, the MBA module adds biases $b_{n,k}$ on these maps to generate NK band maps, then these 'biasing' maps are fed into the subsequent convolutional layer in a flexible way. (b) The piecewise linear function (APL) module where a set of K learnable parameters $\{a_k, b_k\}$ sums up the maps within each channel before feeding N output maps into the next convolution layer and thus providing no cross-channel information. The additional parameters brought by these two modules are NK and $2KWH$, respectively.

2. Multi-bias Non-linear Activation

The goal of this work is to decouple a feature map into multiple band maps by introducing a MBA layer, thus enforcing different thresholds on the same signal where in some cases responses in a certain range carry useful patterns and in another case they are merely noise. After passing through ReLU, these band maps are selected and combined by the filters in the subsequent convolution to represent more abstract visual patterns with large diversity.

2.1. Model formulation

Fig.3 (a) depicts the pipeline of the MBA module. After convolution in CNN, we obtain a set of feature maps. We represent the n-th feature map by vector $\mathbf{x}_n \in \mathbb{R}^{WH}$, where $n = 1 \ldots N$, W and H denote the spatial width and height of the feature map, respectively. If the input of the MBA layer is the response obtained by a fully-connected layer, we can simply treat $W = H = 1$. The MBA layer separates $\mathbf{x}_n$ into K feature maps $\hat{\mathbf{x}}_{n,k}$ as follows:

$$\hat{\mathbf{x}}_{n,k} = \sigma(\mathbf{x}_n + b_{n,k}) \text{ for } k = 1, \ldots, K, \qquad (1)$$

where $\sigma(\cdot)$ is the element-wise nonlinear function. Note that the only parameter introduced by the MBA module is a scalar $b_{n,k}$. Denote $x_{n,i}$ and $\hat{x}_{n,k,i}$ as the i-th element in the map $\mathbf{x}_n$ and $\hat{\mathbf{x}}_{n,k}$ respectively, where $i = 1, \ldots, WH$, we have the element-wise output form of the MBA module defined as:

$$\hat{x}_{n,i} = \sigma(x_{n,i} + b_{n,k}). \qquad (2)$$

In this paper, we mainly consider using ReLU as the nonlinear function because it is found to be successful in many applications. In ReLU, the response $x_{n,i}$ is thresholded by

the bias $b_{n,k}$ as follows:

$$\begin{aligned} \text{if } x_{n,i} \leq -b_{n,k}, \quad &\text{then } \hat{x}_{n,i} = 0, \\ \text{if } x_{n,i} > -b_{n,k}, \quad &\text{then } \hat{x}_{n,i} = x_{n,i} + b_{n,k}. \end{aligned} \qquad (3)$$

2.2. MBA with its subsequent convolutional layer

The output of the MBA module is a set of maps $\{\hat{\mathbf{x}}_{n,k} | n = 1, \ldots, N, k = 1, \ldots, K\}$. These maps are then linearly combined by its subsequent convolutional layer into $\mathbf{h}_m$ as follows:

$$\mathbf{h}_m = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{W}_{m,n,k} \hat{\mathbf{x}}_{n,k} \qquad (4)$$

where $m = 1, \ldots, M$ and $\mathbf{W}_{m,n,k} \hat{\mathbf{x}}_{n,k}$ is the representation of convolution by matrix multiplication. Denote the j-th element in $\mathbf{h}_m$ by $h_{m,j}$, we have

$$h_{m,j} = \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{i=1}^{WH} w_{m,n,k,i,j} \hat{x}_{n,k,i}, \qquad (5)$$

where $w_{m,n,k,i,j}$ denotes the (j, i)-th element in the matrix $\mathbf{W}_{m,n,k}$. Taking the representation of $\hat{x}$ by x in (2) into consideration, we have the factorized version of (5):

$$\begin{aligned} h_{m,j} &= \sum_{i=1}^{WH} \sum_{n=1}^{N} \sum_{k=1}^{K} w_{m,n,k,j,i} \sigma(x_{n,i} + b_{n,k}), \\ &= \sum_{i=1}^{WH} \sum_{n=1}^{N} w'_{m,n,j,i} \sum_{k=1}^{K} a_{m,n,k,j,i} \sigma(x_{n,i} + b_{n,k}), \\ &= \sum_{i=1}^{WH} \sum_{n=1}^{N} w'_{m,n,j,i} u_{m,n,j,i}, \end{aligned}$$
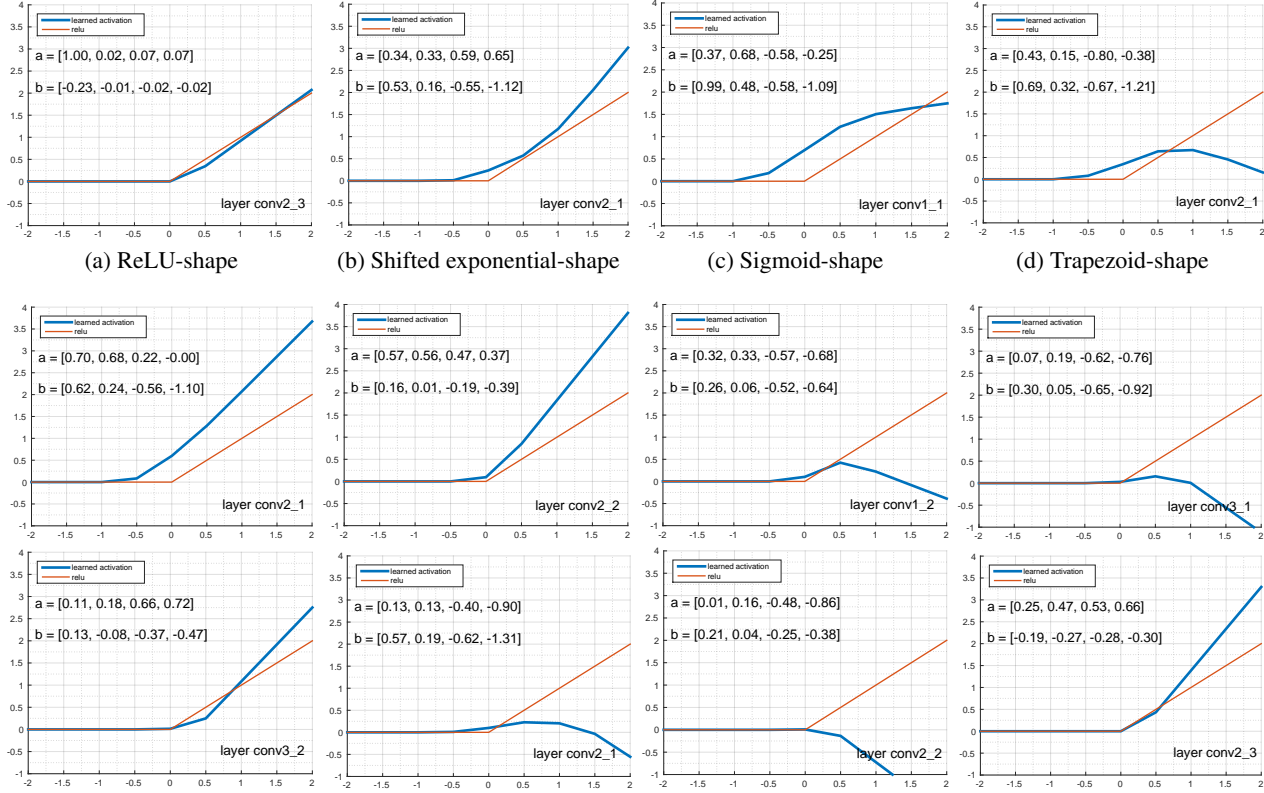
$$(6)$$

Figure 4. The learned MBA parameters a, b in a certain layer ($K = 4$). We use the MBA model #9 in Table 2 as an illustration. Each figure plots the mapping function $B_{m,n,j,i}$. The horizontal axis indicates $x_{n,i}$ in the input feature map and vertical axis indicates $u_{m,n,j,i}$ in the output map in (6). The index of feature channels m and n, and locations j and i are dropped to be concise. The fist row shows four typical shape during the activation learning whilst the rest gives more visual examples.

where w' and u take the forms of:

$$w_{m,n,k,j,i} \triangleq w'_{m,n,j,i} a_{m,n,k,j,i}, \tag{7}$$

$$u_{m,n,j,i} \triangleq \sum_{k=1}^{K} a_{m,n,k,j,i} \sigma(x_{n,i} + b_{n,k}), \tag{8}$$

here w' is an intermediate variable to generate the coefficient $a_{m,n,k,j,i}$. The formulation in (8) shows that the element $x_{n,i}$ in the feature map is separated by multiple biases to obtain multiple ReLU functions $\sigma(x_{n,i} + b_{n,k})$, and then these ReLU functions are linearly combined by the weights $a_{m,n,k,j,i}$ to obtain $u_{m,n,j,i}$, which serves as the decoupled pattern in the MBA layer for the m-th channel in the next convolutional layer at location j. The j-th element in $\mathbf{h}$, *i.e.*, $h_{m,j}$ is a weighted sum of $u_{m,n,j,i}$ as shown in (6). Therefore, the key is to study the mapping $x_{n,i}$ at location i in an input feature map to $u_{m,n,j,i}$ at location j in an output feature map in (8). Such a mapping is *across* feature channels and locations. (8) can be defined as a mapping function $u_{m,n,j,i} = B_{m,n,j,i}(x_{n,i})$. There is a large set of such mapping functions $\{B_{m,n,j,i}\}$ which are characterized by parameters $\{a_{m,n,k,j,i}\}$ and $\{b_{n,k}\}$. In the following discussion, we skip the subscripts $_{m,n,j,i}$ to

be concise.

We show the learned parameters of a_k and b_k for the input x and the decoupled pattern u in Figure 4. Specifically, Fig.4 (a) approximates the ReLU unit where $a \approx [1, 0, 0, 0]$ is the base along the first axis in a 4-dimension space; Fig.4 (b) displays the property of leaky ReLU (Xu et al., 2015) to allow gradient back-propagation around small negative input; moreover, it has a steeper slope in the positive region than ReLU, which makes the training process faster. Fig.4 (c) stimulates the case where the activation function serves as a sigmoid non-linearity, *i.e.*, only allowing small values around zero to backpropagate the gradients. The non-linear activation function in Fig.4 (d) forms a trapezoid-shaped and serves as the histogram bin 'collector' - only selecting the input x within a small range and shuttering all the other activations outside the range. Note that the mapping is concave due to the negative values of a_{ik} when $x \to +\infty$, in which case neither the standard ReLU nor APL unit could describe. In addition, the second and third rows of Figure 4 show more examples of the mappings decomposed from parameters in the convolution layer, from which we can see a wide diversity of patterns captured by the multi-bias
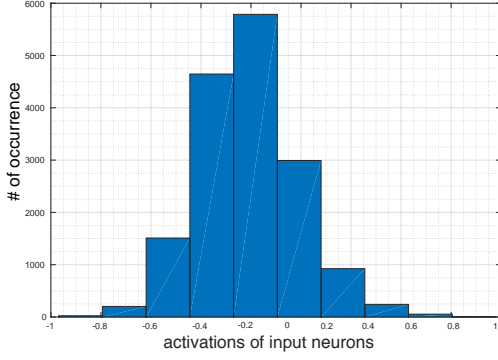
Figure 5. Histogram of the neuron activations before the MBA module, of which most activations are sparse and centered around zeros.

mechanism to decouple the signal strength and the cross-channel sharing scheme to combine visual representations in a much more flexible way.

Figure 5 shows the histogram of response magnitudes in the input feature map before feeding into the MBA module. We adopt the architecture of model #6 in Table 2 on CIFAR-10 to obtain the distribution result, where we randomly select 1,000 samples to get the average activations over all the MBA output. The histogram distributes around zero, which indicates the learned pattern u or mapping B affects the neurons only in a small range near zero.

2.3. Relationship with piecewise-linear functions

To see further the advantage of our algorithm, we compare our MBA layer with the recently proposed method called adaptive piecewise linear function unit (APL) (Agostinelli et al., 2015). It formulates the activation function as a sum of hinge-shaped functions. Fig.3 (b) describes the pipeline of the APL module. Given the feature map $\mathbf{x}_n$, it generates the output $\hat{x}_{n,k,i}$ from k piecewise linear functions from each element $x_{n,i}$ in $\mathbf{x}_n$ as follows:

$$\hat{x}_{n,i} = \max(0, x_{n,i}) + \sum_{k=1}^{K} a_{k,i} \max(0, -x_{n,i} + b_{k,i}),$$
(9)

where $a_{k,i}, b_{k,i}$ are learnable parameters to control the shape of the non-linearity function. Then the subsequent convolutional layer computes the weighted sum of 'piecewise-linearized' maps to have the output $h_{m,j}$ of channel m at location j as follows:

$$h_{m,j} = \sum_{i=1}^{WH} \sum_{n=1}^{N} w_{m,n,i,j} \left(\sum_{k} a_{k,i} \max(0, -x_{n,i} + b_{k,i}) \right),$$

$$= \sum_{i=1}^{WH} \sum_{n=1}^{N} w_{m,n,i,j} u_{n,i},$$
(10)

Table 1. Investigation of the MBA module in different architectures on the MNIST and CIFAR-10 dataset. We choose $K = 4$ for both MBA and APL. Bracket [·] denotes a stack of three convolution layers. 'VaCon' means the vanilla neural network. '@' represents a different number of output maps before the MBA module, see context for details.

Model	# Params	MNIST	CIFAR-10
Shallow network: 32-64-128-1024-1024			
VaCon	93k	2.04%	34.27%
VaCon@4x	594k	0.95%	22.75%
APL	120k	1.08%	28.72%
APL@4x	620k	1.15%	23.80%
APL@same	358k	1.17%	31.53%
MBA	369k	0.83%	22.39%
Deep network: [32]-[64]-[128]-1024-1024			
VaCon	480k	1.03%	19.25%
VaCon@4x	7.6M	0.42%	14.38%
APL	550k	1.37%	22.54%
APL@4x	7.7M	0.54%	13.77%
APL@same	2.1M	0.82%	17.29%
MBA	2M	0.31%	12.32%

where w is the parameters of the subsequent convolution kernel and we define

$$u_{n,i} \triangleq \sum_{k=1}^{K} a_{k,i} \max(0, -x_{n,i} + b_{k,i}),$$
(11)

in a similar derivation through (4) to (6). It is obviously seen that APL represented in (10) is a special case of MBA by enforcing $u_{m_1,n,j,i} = u_{m_2,n,j,i}, \forall m_1, m_2$ in (8). Therefore, for different target channel with index m_1 and m_2, the piecewise-linear function provides the *same* u_* while MBA provides *different* u_*. Take again the case in Figure 1 for instance, the output channel $m_1 = 1$ for eyes requires u with high magnitude while $m_2 = 2$ for mouth requires u with low magnitude. This kind of requirement cannot be met by APL but can be met by our MBA. This is because our MBA can separate single x into different u_* for different m according to the magnitude of x while APL cannot.

A closer look at the difference between our algorithm and the APL unit is through the two diagrams in Figure 3. When an input feature map is decoupled into multiple band maps after the biasing process, APL only recombines band maps from the same input feature map to generate one output. However, MBA concatenates band maps across different input maps and allows to select and combine them in a flexible way, thus generating a much richer set of maps (patterns) than does the APL.

Table 1 shows the investigation breakdown on MNIST and CIFAR-10 when comparing MBA with APL and vanilla

neural networks[1]. Here we adopt two types of network: the shallow one has three convolutional layers with the number of output 32, 64, 128, respectively and two fully connected layers which both have an output neurons of 1024; the deep one has nine convolutional layers divided into three stacks with each stack having the same number of output 32, 64, 128 and two fully connected layers. All convolutional layers have the same kernel size of 3 and we use max pooling of size 2, stride 2, at the end of each convolution or stack layer. Also we keep training parameters the same across models within each architecture to exclude external factors.

The number of parameters in each setting of Table 1 does not count those in the fully connected layers and we can compare the computational cost of MBA, APL and vanilla net quantitatively. Take the deep architecture case for example, the vanilla network has about $480k$ parameters with the designated structure; by applying the MBA module on it, the additional parameters are (a) the increasing channels of kernels in each subsequent layer, *i.e.*, $Nq^2M(K-1)$, where $q = 3$ is the kernel size; and (b) a small fraction of the bias term, NK. Therefore we have a computational cost of $480k + 1.5M + 3k = 2M$. However, if we force the vanilla model to have the same output maps to be fed into the subsequent layers (models denoted as '@4x'), there has to be $N(K-1)$ more maps coming from the convolutional kernel in the current layer. As mentioned in Section 1, such a scheme would increase the parameter overhead of kernels to a great extent. That is approximately K times the size of the MBA module (in this case, 7.6M vs 2M).

Several remarks can be drawn from Table 1. First, both MBA and APL modules imposed on the vanilla net can reduce test errors. Second, as the number of feature maps increases, vanilla networks can further boost the performance. However, it is less inferior compared with the MBA module (0.42% vs 0.31% on MNIST) where the latter has a much smaller set of parameters. Third, the piecewise linear function does not perform well compared with the proposed method, even though it has the same network width (APL@4x) or similar parameters (APL@same, by changing the output number of feature maps) as in the MBA model. This is probably due to the limited expressive power, or inferior ability of feature representation in (11). Therefore, these observations further proves the importance of applying the MBA module to separate the responses of various signals and feed the across-channel information into the next layer in a simple way, instead of buying more convolutional kernels.

[1] We do not use the tricks presented in APL (Agostinelli et al., 2015), for example, changing the dropout ratio or adding them before certain pooling layer.

3. Experimental results

We evaluate the proposed MBA module and compare with other state-of-the-arts on several benchmarks. The CIFAR-10 dataset (Krizhevsky et al., 2012) consists of 32×32 color images on 10 classes with 50,000 training images and 10,000 testing images. The CIFAR-100 dataset has the same size and format as CIFAR-10, but contains 100 classes, with only one tenth as many labeled examples per class. The SVHN (Netzer et al., 2011) dataset resembles MNIST and consists of color images of house numbers captured by Google street view. We use the seoncd format of the dataset where each image is of size 32×32 and the task is to classify the digit in the center. Additional digits may appear beside it and must be ignored. All images are preprocessed by subtracting each pixel value by the mean computed from the corresponding training set. We follow a similar split-up of the validation set from the training set as (Goodfellow et al., 2013), where one tenth of samples per class from the training set on CIFAR, and 400 plus 200 samples per class from the training and the extra set on SVHN, are selected to build a validation set.

3.1. Implementation Details

Our baseline network has three stacks of convolutional layers with each stack containing three convolutional layers, resulting in a total number of nine layers. Each stack has [96-96-96], [128-128-128] and [256-256-512] filters, respectively. The kernel size is 3 and padded by 1 pixel on each side with stride 1 for all convolutional layers. At the end of each convolutional stack is a max-pooling operation with kernel and stride size of 2. The two fully connected layers have 2048 neurons each. We also apply dropout with ratio 0.5 after each fully connected layers. The final layer is a softmax classification layer.

The optimal training hyperparameters are determined on each validation set. We set the momentum as 0.9 and the weight decay to be 0.005. The base learning rate is set to be $0.1, 0.1, 0.05$, respectively. We drop the learning rate by 10% around every 40 epochs in a continuous exponential way and stop to decrease the learning rate until it reaches a minimum value (0.0001). For the CIFAR-100 and SVHN datasets, we use a slightly longer cycle of 50 epochs to drop the rate by 10%. The whole training process takes around 100, 150, and 150 epochs on three benchmarks. We use the hyperparameter $K = 4$ for the MBA module and the mini-batch size of 100 for stochastic gradient descent. All the convolutional layers are initialized with Gaussian distribution with mean of zero and standard variation of 0.05 or 0.1. We do not carefully cross-validate the initialized biases in the MBA module to find the optimal settings but simply choose a set of constants to differentiate the initial biases.

Table 2. Ablation study of applying the bias module with different width and depth into the network on CIFAR-10. Empty entry means that MBA is not included while the rest employs a MBA module and specifies the number of output feature maps for the corresponding convolution layer.

Model	Conv-1	Conv-2	Conv-3	Test Error.
Baseline	96-96-96	128-128-128	256-256-512	9.4%
1	-	-	64-64-128	8.5%
2	-	-	128-128-256	7.3%
3	-	-	256-256-512	7.2%
4	-	32-32-32	64-64-128	10.4%
5	-	64-64-64	128-128-256	8.2%
6	-	128-128-128	256-256-512	**6.7%**
7	24-24-24	32-32-32	64-64-128	11.7%
8	48-48-48	64-64-64	128-128-256	8.8%
9	96-96-96	128-128-128	256-256-512	6.8%

Table 3. Effects of the hyperparameter K of MBA. The architecture is the one used in model #6 from Table 2. (conv*) means that we set a particular value of K in that convolution stack only.

Method	Val. Error
Baseline	9.4%
$K = 2$	8.9%
$K = 4$ (conv2), $K = 2$ (conv3)	8.1%
$K = 4$	6.7%
$K = 4$ (fixed MBA)	10.8%
$K = 8$	6.6%

3.2. Ablation Study

First, we explicitly explore the ability of the MBA module in CNN with different numbers of channels in each layer. From Table 2 we conclude that adding more MBA layers into the network generally reduces the classification error. Also, the width of the network, *i.e.*, the number of filters in each stack, plays an important role to reduce the classification error. Considering models #4-#6, we can see that larger number of filters results in more expressive power the network and thus smaller classification error. It can be observed that the use of more MBA layers in model #6 performs better than the use of fewer MBA layers in model #3. However, the MBA module imposed on all stacks does not perform better than the one imposed on stack 2 and 3 only. This result shows that it is not necessary to use the MBA layer for lower convolutional layers that are close to raw data. Moreover, the improvement of our method does not come from introducing more parameters. For example, model #5 has much fewer parameters than the baseline and it still outperforms the baseline.

Second, we investigate the effect of the hyperparameter K in the MBA module (Table 3), which is conducted on the

Table 4. Classification test errors on CIFAR dataset with and without data augmentation. The best results are in bold.

Method	CIFAR-10	CIFAR-100
Without Data Augmentation		
ReLU (Srivastava et al., 2014)	12.61%	37.20%
Channel-out (Wang & JaJa, 2013)	13.20%	36.59%
Maxout (Goodfellow et al., 2013)	11.68%	38.57%
NIN (Lin et al., 2014)	10.41%	35.68%
DSN (Lee et al., 2014)	9.78%	34.57%
APL (Agostinelli et al., 2015)	9.59%	34.40%
Ours	**6.73 %**	**26.14%**
With Data Augmentation		
Maxout (Goodfellow et al., 2013)	9.38%	-
DropConnect (Wan et al., 2013)	9.32%	-
SelAtten (Stollenga et al., 2014)	9.22%	33.78%
NIN (Lin et al., 2014)	8.81%	-
DSN (Lee et al., 2014)	8.22%	-
APL (Agostinelli et al., 2015)	7.51%	30.83%
BayesNet (Snoek et al., 2015)	6.37%	27.40%
ELU (Clevert et al., 2015)	6.55%	24.28%
Ours	**5.38%**	**24.1%**

CIFAR-10 validation set. In the MBA layer, a channel is decoupled to K channels. We can observe that the inclusion of MBA layer on the network reduces the classification error when $K = 2, 4$ and 8. To further validate the necessity of the learnable biases, we fix the bias parameters after initialization. In this case, the validation error increases from 6.7% for learned bias to 10.8% for fixed bias. Moreover, we find that setting a large $K = 8$ does not reduce the classification error further compared with $K = 4$ because it is not necessary to decouple a single channel into too many channels.

3.3. Comparison to State-of-the-Arts

We show the comparison results of the proposed MBA with other state-of-the-arts, including ReLU (Srivastava et al., 2014), Channel-out (Wang & JaJa, 2013), Maxout (Goodfellow et al., 2013), Network in Network (NIN) (Lin et al., 2014), Deep Supervision (DSN) (Lee et al., 2014), APL (Agostinelli et al., 2015), DropConnect (Wan et al., 2013), Selective Attention Model (Stollenga et al., 2014), Scalable Bayes Network (Snoek et al., 2015) and Exponential Linear Unit (Clevert et al., 2015) on the CIFAR and SVHN datasets. We will use the network architecture of candidate model #6 as the final MBA model thereafter.

CIFAR. Without data augmentation, Table 4 indicates that we achieve a relative 29.8% and 24% gain over previous state-of-the-arts on CIFAR-10 and CIFAR-100, respectively. As for the data augmentation version of our

Table 5. Classification test errors on SVHN dataset without data augmentation. The best results are in bold.

Method	Test Error
StoPool (Zeiler & Fergus, 2013)	2.80%
ReLU (Srivastava et al., 2014)	2.55%
Maxout (Goodfellow et al., 2013)	2.47%
NIN (Lin et al., 2014)	2.35%
DropConnect (Wan et al., 2013)	1.94%
DSN (Lee et al., 2014)	1.92%
GenPool (Lee et al., 2015)	**1.69%**
Ours	1.80%

model, during trainin we first resize each image to a random size sampled from $[32, 40]$ and then crop a 32×32 region randomly out of the resized image. Horizontal flip is also adopted. For testing, we employ a multi-crop voting scheme, where crops from five corners (center, top right and left, bottom right nad left) are extracted and the final score is determined by their average. Note that we do not aggressively resort to trying all kinds of data augmentation techniques (Snoek et al., 2015), such as color channel shift, scalings, etc.; or extend the network's depth to extremely deep, for example, ELU (Clevert et al., 2015) used a model of 18 convolutional layers. Our algorithm performs better than previous ones by an absolute reduction of 1.17% and 0.18% with data augmentation on these two datasets.

SVHN. We further conduct the comparison experiment on the house number dataset and we achieve a test error rate of 1.80% without data augmentation (Table 5).

4. Conclusion and Discussion

In this work, we propose a multi-bias non-linearity activation (MBA) module in deep neural networks. A key observation is that magnitudes of the responses from convolutional kernels have a wide diversity of pattern representations in the network, and it is not proper to discard weaker signals with single thresholding. The MBA unit placed after the feature maps helps to decouple response magnitudes to multiple maps and generates more patterns in the feature space at a low computational cost. We demonstrate that our algorithm is effective by conducting various independent component analysis as well as comparing the MBA method with other state-of-the-art network designs. Experiments show that such a design has superior performance than previous state-of-the-arts.

While the MBA layer could enrich the expressive power of the network, we believe more exploration of the discriminative features can be investigated to leverage the information hidden in the magnitude of response. Such an intuition is triggered by the fact that the non-linearity actually preserves or maintains the depth property of a network. One simple way is to divide the input feature maps, feed them into multiple non-linearities, and gather together again as the input of the subsequent convolutional layer.

References

Agostinelli, Forest, Hoffman, Matthew, Sadowski, Peter, and Baldi, Pierre. Learning activation functions to improve deep neural networks. In *ICLR workshop*, 2015.

Bruna, Joan and Mallat, Stephane. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1872–1886, August 2013.

Clevert, Djork-Arne, Unterthiner, Thomas, and Hochreiter, Sepp. Fast and accurate deep network learning by exponential linear units. *arXiv preprint: 1511.07289*, 2015.

Goodfellow, Ian J., Warde-farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. In *ICML*. 2013.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint:1502.01852*, 2015.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, 2006.

Hinton, Geoffrey E., Deng, Li, Yu, Dong, Dahl, George E., Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N., and Kingsbury, Brian. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012.

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

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. *arXiv preprint: 1409.5185*, 2014.

Lee, Chen-Yu, Gallagher, Patrick W., and Tu, Zhuowen. Generalizing pooling functions in convolutional neural networks: Mixed, Gated, and Tree. *arXiv preprint:1509.08985*, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. In *ICLR*, 2014.

Mass, Andrew L., Hannum, Awin Y., and Ng, Andew Y. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2010.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop*, 2011.

Silver, David, Huang, Aja, Maddison, Chris J., Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, and Hassabis, Demis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, Mr, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015.

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

Stollenga, Marijn F, Masci, Jonathan, Gomez, Faustino, and Schmidhuber, Juergen. Deep networks with internal selective attention through feedback connections. In *NIPS*, 2014.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Cun, Yann Le, and Fergus, Rob. Regularization of neural networks using dropconnect. In *ICML*, 2013.

Wang, Qi and JaJa, Joseph. From maxout to channel-out: Encoding information on sparse pathways. *arXiv preprint: 1312.1909*, 2013.

Xu, Bing, Wang, Naiyan, Chen, Tianqi, and Li, Mu. Empirical evaluation of rectified activations in convolution network. *arXiv preprint: 1505.00853*, 2015.

Zeiler, Matthew D. and Fergus, Rob. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint:1301.3557*, 2013.