Adaptive Convolutional ReLUs

Hongyang Gao,[1] **Lei Cai,**[2] **Shuiwang Ji**[1]
[1]Texas A&M University, College Station, TX, USA
[2]Washington State University, Pullman, WA, USA
{hongyang.gao, sji}@tamu.edu, lei.cai@wsu.edu

Abstract

Rectified linear units (ReLUs) are currently the most popular activation function used in neural networks. Although ReLUs can solve the gradient vanishing problem and accelerate training convergence, it suffers from the dying ReLU problem in which some neurons are never activated if the weights are not updated properly. In this work, we propose a novel activation function, known as the adaptive convolutional ReLU (ConvReLU), that can better mimic brain neuron activation behaviors and overcome the dying ReLU problem. With our novel parameter sharing scheme, ConvReLUs can be applied to convolution layers that allow each input neuron to be activated by different trainable thresholds without involving a large number of extra parameters. We employ the zero initialization scheme in ConvReLU to encourage trainable thresholds to be close to zero. Finally, we develop a partial replacement strategy that only replaces the ReLUs in the early layers of the network. This resolves the dying ReLU problem and retains sparse representations for linear classifiers. Experimental results demonstrate that our proposed ConvReLU has consistently better performance compared to ReLU, LeakyReLU, and PReLU. In addition, the partial replacement strategy is shown to be effective not only for our ConvReLU but also for LeakyReLU and PReLU.

Introduction

Convolutional neural networks (CNNs) (LeCun et al. 1998b) have shown great capability in various fields such as computer vision (Ren et al. 2015; Laina et al. 2016) and natural language processing (Johnson and Zhang 2017). In CNNs, activation functions play important roles for introducing nonlinearity to networks. Among various activation functions such as $\tanh(\cdot)$ and sigmoid$(\cdot)$, ReLU is the most popular one. ReLU computes the identity for positive arguments while outputs zero for negative ones. It was initially proposed for Boltzmann machines (Nair and Hinton 2010) and has been successfully applied to neural networks for its non-saturating property, which alleviates the gradient vanishing problem and accelerates convergence speed.

Though effective and efficient, ReLUs suffer from the dying ReLU problem, which makes some neurons in the network never activated again (Chen et al. 2017). Many attempts have tried to resolve or alleviate this problem such as LeakyReLUs (Maas, Hannun, and Ng 2013) and PReLUs (He et al. 2015). These works tackle the problem by using a small constant or trainable slope for negative arguments. However, the behaviors of such activation functions are different from how biological neurons are activated, as biological neurons actually employ different thresholds within a certain range (Islam and Islam 2016).

In this work, we propose a simple but effective method known as adaptive ReLUs, which activates the arguments based on trainable thresholds. Based on the adaptive ReLUs, we develop a novel parameter sharing scheme that is especially applicable to convolution layers. This leads to adaptive convolutional ReLUs (ConvReLUs). For the parameters involved in ConvReLUs, we employ the zero initialization with L2-regularization as it encourages the trainable thresholds to be close to zero. Finally, we propose the partial replacement strategy for ReLUs replacement in the network to relieve dying ReLU problem and retain sparse representations for linear classifiers in neural networks.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Related Work

Activation functions have been a popular research field due to its importance in deep neural networks. Initially, $\tanh(\cdot)$ and sigmoid$(\cdot)$ were applied in neural networks with $\tanh(\cdot)$ being preferred for its zero-centering property (LeCun et al. 1998a). To solve the gradient vanishing problem suffered by saturating activation functions like $\tanh(\cdot)$ and sigmoid$(\cdot)$, the non-saturating function ReLU (Nair and Hinton 2010) was proposed and successfully applied in various state-of-the-art deep neural networks (He et al. 2016; Huang et al. 2017; Gao et al. 2019).

ReLUs compute the function

$$f(x) = \max(0, x), \tag{1}$$

which solves the gradient vanishing problem and accelerates convergence in training (Krizhevsky, Sutskever, and Hinton 2012). However, it suffers from a problem known as the "dying ReLU" (Chen et al. 2017). In this scenario, a large weight update may cause the ReLU neuron to be never activated again. Thus gradients that flow through those neurons will

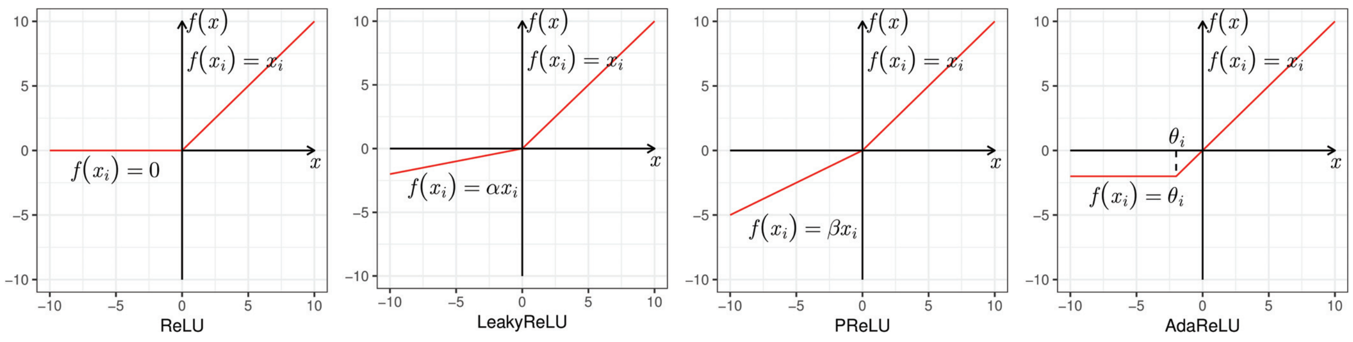

Figure 1: Illustrations of ReLU, LeakyReLU, PReLU, and the proposed adaptive ReLU (AdaReLU). ReLU computes the function $f(x_i) = \max(0, x_i)$. LeakyReLU computes $f(x_i) = \max(\alpha x_i, x_i)$, where α is a small constant. PReLU computes the same function as LeakyReLU with a trainable β. Our AdaReLU computes the function $f(x_i) = \max(\theta_i, x_i)$, where θ_i is a trainable parameter corresponding to x_i. AdaReLU can be naturally used in convolutional layers, leading to ConvReLU.

be zero. Leaky ReLU (LeakyReLU) (Maas, Hannun, and Ng 2013) attempts to address the dying ReLU problem by employing a small slope instead of zero when $x < 0$. It computes the function

$$f(x) = \max(\alpha x, x), \tag{2}$$

where α is a small constant like 0.01. Parametric Rectified Linear Unit (PReLU) (He et al. 2015) generalizes this function by making α to be a trainable parameter. ReLU, LeakyReLU, and PReLU functions are illustrated in Figure 1.

Other types of activation functions (Xu et al. 2015) have been proposed with different functional forms. Exponential Linear Units (ELUs) (Clevert, Unterthiner, and Hochreiter 2015) computes the function

$$f(x_i) = \begin{cases} x_i, & \text{if } x_i > 0 \\ \alpha(\exp(x) - 1), & \text{if } x_i \leq 0. \end{cases} \tag{3}$$

Based on ELUs, Scaled Exponential Linear Units (SELUs) induced self-normalizing properties (Klambauer et al. 2017). However, both ELUs and SELUs only fitted fully-connected layers, and the use in convolution layers is not clear.

Adaptive Convolutional Rectified Linear Units

In this section, we firstly present the adaptive ReLUs (AdaReLU). Then we propose the parameter sharing mechanism in neural networks especially for convolution layers, leading to convolutional ReLUs (ConvReLUs). In addition, we discuss the parameter initialization and activation function replacement in neural networks for ConvReLUs.

Background and Motivations

It is well known that neural networks mimic computational activities in biological brains. Each neuron, the basic computational unit in brain, receives signals from its dendrites and outputs signal when its strength is above a certain threshold known as threshold potentials (Chen et al. 2006). Figure 2 (a) illustrates a simplified version of this process, which is also called feed-forward network with ReLUs. In brain, the thresholds above which neurons fire are not the same but are largely in the range between -50 and -55 mV (Seifter, Sloane, and Ratner 2005). This is different from our popular activation functions ReLUs in which the same zero threshold is applied to every neuron as illustrated in Figure 2 (b). With the support from biological theories (Seifter, Sloane, and Ratner 2005), we propose that each input neuron should have different activation thresholds, which can better mimic brain functions.

Adaptive ReLU

To enable different thresholds for different input neurons, we propose the Adaptive ReLUs (AdaReLUs) activation function defined as

$$f(x_i) = \begin{cases} x_i, & \text{if } x_i > \theta_i \\ \theta_i, & \text{if } x_i \leq \theta_i, \end{cases} \tag{4}$$

where x_i is the argument of the activation function $f(\cdot)$, and θ_i is the corresponding threshold for input x_i. Note that θ_i is learned automatically from data. Figure 1 illustrates the adaptive ReLU. The subscript i in θ_i indicates that the nonlinear activation function can vary for different inputs in terms of thresholds. This gives an extra degree of flexibility for its applications in neural networks. Each input neuron can have different thresholds, which are learned from data. In addition, the input neurons of the same channel or even the same layer can share the same threshold. Adaptive ReLU reduces to ReLU when $\theta_i = 0$ for all i.

AdaReLUs can be trained by back-propagation algorithms (LeCun et al. 1989) along with other layers simultaneously. The update equation for trainable parameters $\{\theta_i\}$ can be derived from the chain rule. For one layer, the gradient of θ_i can be expressed as:

$$\frac{\partial \mathcal{E}}{\partial \theta_i} = \sum_{x_i} \frac{\partial \mathcal{E}}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial \theta_i}, \tag{5}$$

where $\mathcal{E}$ is the objective function and $\frac{\partial \mathcal{E}}{\partial f(x_i)}$ represents the gradient back-propagated from the deeper layer. The summation $\sum_{x_i}$ runs over all positions on the feature map that employ θ_i as the activation threshold. The gradient of the activation threshold θ_i can be written as:

$$\frac{\partial f(x_i)}{\partial \theta_i} = \begin{cases} 0, & \text{if } x_i > \theta_i \\ 1, & \text{if } x_i \leq \theta_i. \end{cases} \tag{6}$$

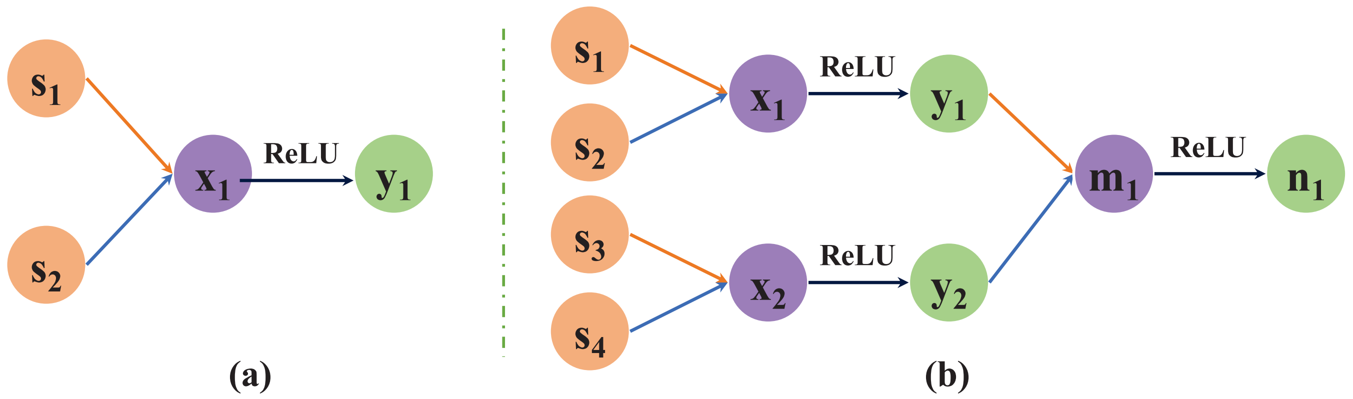

Figure 2: Illustrations of one-layer (a) and multi-layer (b) feed-forward networks. In the one-layer network (a), the output neuron x_i is activated by ReLU after the element-wise multiplication and summation. But in the view of multi-layer network (b), all input neurons x_1 and x_2 for the second layer are activated by the same activation function or threshold.

Adaptive Convolutional ReLUs

By employing different thresholds for different input neurons, AdaReLUs may potentially incur a large number of extra parameters as compared to ReLUs. For example, we can simply use different thresholds for each input neuron, thereby requiring a large number of extra parameters and increasing the risk of over-fitting. On the other hand, we can use parameter sharing schemes to reduce the number of extra parameters. For example, we can require the input neurons of the same layer to share one trainable threshold like PReLUs. Then the number of extra parameters involved in this sharing scheme will be negligible.

In this work, we propose a new parameter sharing scheme that works especially well for convolutional layers. In our scheme, the input units sharing the same outgoing weight in convolution are also required to share the same threshold. We term this version of adaptive ReLUs as convolutional ReLUs (ConvReLUs). In convolution layers, each input unit is connected to multiple outgoing weights depending on the sizes of convolution kernels. For instance, in an 1D convolution layer with a kernel of size $k \times 1$, an input unit (not on the boundary) is used k times with k different outgoing weights w_i acting on it. In ConvReLUs, we have k corresponding different thresholds $\{\theta_1 \dots \theta_k\}$ for each input unit. When w_i is acting on the input unit, a unit x is activated by computing

$$f(x) = \max(\theta_i, x), \tag{7}$$

where w_i and θ_i share the same index.

Note that the trainable thresholds θ in ConvReLUs are shared across output channels but not input channels. Suppose we have n_{in} input and n_{out} output channels in convolution layer, the number of trainable thresholds in ConvReLUs is $n_{in} \times k$ while the number of weights is $n_{in} \times k \times n_{out}$. In this way, we set different thresholds for input neurons without inducing excessive extra parameters. Figure 3 illustrates how ConvReLUs share parameters in 1D convolution layers. The same parameter sharing scheme can be extended to 2D and 3D convolution layers.

The unit $x^l_{i,j}$ in output feature map d is calculated as:

$$x^l_{i,j,d} = \sum_{c=0}^{n_{in}-1}\sum_{a=0}^{k-1}\sum_{b=0}^{k-1} w_{a,b,c,d}\max(x^{l-1}_{i+a,j+b,c}, \theta_{a,b,c}) \tag{8}$$

where kernel size $k \times k$ is used in this 2D convolution and n_{in} is the number of input channels. x^{l-1} and x^l are the input and output of layer l. θ and w are trainable thresholds and weights in our ConvReLU layer, respectively. In our ConvReLU, θ is shared across output feature maps. Suppose we have an input $[2, 3, 4]$, weights $[1, -1]$, and thresholds $[0, 3.5]$. When applying 1D ConvReLU without padding, the output is $[-1.5, -1]$.

Parameter Initialization

We observe that parameter initialization has an impact on model performance. In this section, we discuss a few methods for initialing parameters involved in our proposed ConvReLUs.

Random initialization: In deep neural networks, the weights are commonly initialized randomly from Gaussian or uniform distributions with zero means (Krizhevsky, Sutskever, and Hinton 2012). Compared to zero initialization, random initialization serves as a symmetry-breaking tool that makes every neuron perform different computations.

Glot initialization: Glorot and Bengio (Glorot and Bengio 2010) used a scaled uniform distribution for parameter initialization, which is also known as the Xavier initialization. In this scheme, the weights are drawn from a distribution with zero mean but a specific variance. The variance recommended in that work is $var(w) = 2/(n_{in} + n_{out})$, where n_{in} and n_{out} are the numbers of input and output channels, respectively. However, this recommendation is based on the assumption that the activation functions are linear, which is not the case with ReLUs and its variants (He et al. 2015).

Zero initialization: Compared to the previous two initialization methods, zero initialization was rarely used in neural networks, since neurons initialized to zero perform the same computation. However, the parameters in our proposed ConvReLUs correspond to activation thresholds. We expect the

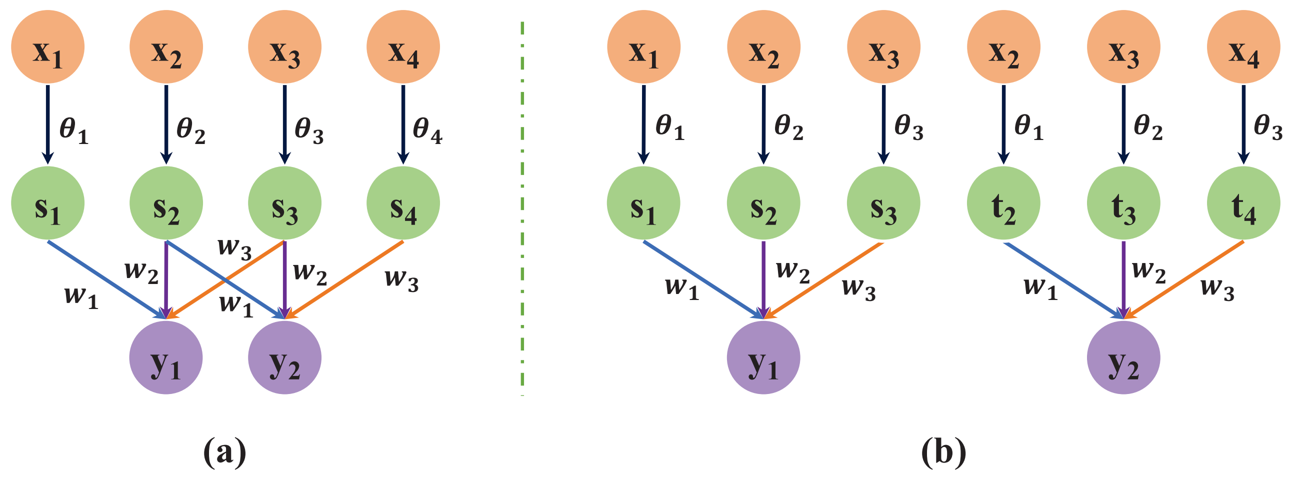

Figure 3: Illustrations of parameter sharing of ConvReLUs in 1D convolution layers. The left figure (a) shows the sharing scheme in which each input unit has a different trainable threshold. Apparently, this scheme incurs a large number of extra parameters. When $\theta_i = 0$, ConvReLUs reduce to ReLUs. The right figure (b) illustrates our proposed parameter sharing scheme in which the input units sharing the same outgoing weights also share the same threshold. The network in (b) is a decomposed version of the network in (a). Nodes with the same symbol are replicated versions of the same node for illustration purposes. In this example, we have three thresholds θ_i $(i = 1, 2, 3)$ corresponding to three weights w_i $(i = 1, 2, 3)$ used in this convolution layer. When computing y_1, the weight w_2 acts on the input unit x_2. We apply threshold θ_2 on x_2. For computing y_2, the threshold θ_1 is applied on the unit x_2 since the weight w_1 is used.

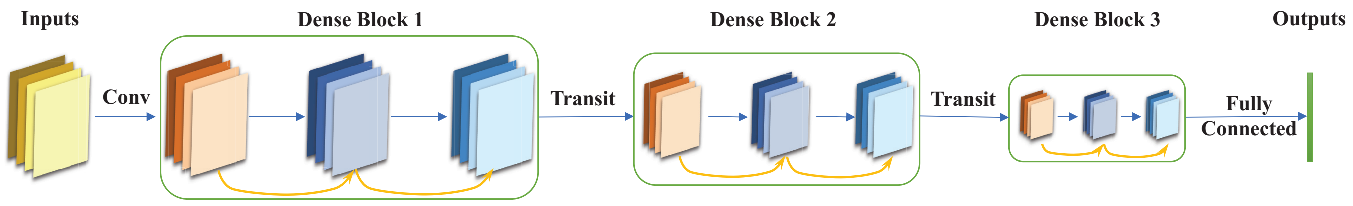

Figure 4: The DenseNet-40 used for image classification tasks. There are three dense blocks, each of which has 8 layers with a growth rate of 12.

thresholds to be close to, but may not exactly equal to, zero. From this point, initializing these trainable thresholds to zero is a valid strategy. We observe from our experiments that zero initialization with L2-regularization performs the best.

Activation Functions in Deep Multi-Layer Networks

In previous studies on improving ReLUs (Maas, Hannun, and Ng 2013; He et al. 2015; Klambauer et al. 2017), ReLUs in all layers are completely replaced by their proposed new activation functions. Despite the possibility of resulting in dying neurons in the network, ReLUs have the advantage of inducing higher sparsity for the computed features as compared to other activation functions. This may consequently reduce the risk of over-fitting (Glorot, Bordes, and Bengio 2011). Although these activation functions can overcome the dying ReLU problem, the sparsity of the outputs is significantly reduced, especially for the final linear classifier layer. This may increase the risk of over-fitting.

To alleviate the dying ReLU problem and also preserve sparsity patterns in the computed features, we propose to use a partial replacement strategy, in which only the first several ReLU layers are replaced by our proposed ConvReLUs in the network. The use of ConvReLUs in early layers ensures the neurons are activated by some non-zero values. The remaining ReLU layers in the top part of networks can provide sparse feature representations for final linear classifiers, thus avoiding the over-fitting problem. In this work, we observe that our proposed partial replacement strategy yields better performance not only for our proposed ConvReLUs, but also for LeakyReLUs and PReLUs. We provide detailed experiments and discussions in experimental studies.

Experimental Studies

In this section, we evaluate our proposed ConvReLUs activation function on both image classification and text classification tasks. We conduct experiments to compare ConvReLUs with popular rectified activation functions, including ReLUs, LeakyReLUs, and PReLUs. In addition, performance studies are used to compare three parameter initialization strategies and the replacement strategies.

Table 1: Comparison between ConvReLU and other activation functions in terms of top-1 accuracy on image classification datasets, including Cifar10, Cifar100, and Tiny ImageNet.

Function	Cifar10	Cifar100	ImageNet
ReLU	94.29%	75.87%	56.31%
LeakyReLU	94.45%	76.06%	56.38%
LeakyReLU (A)	94.32%	75.29%	55.97%
PReLU	94.58%	75.78%	56.09%
PReLU (A)	94.03%	74.87%	54.38%
ConvReLU	**94.72%**	**76.41%**	**56.47%**

Table 2: Comparison between ConvReLU function and other activation functions in terms of top-1 accuracy on text classification datasets, including MR, AG's News, and Yelp Full datasets.

Function	MR	AG	Yelp
ReLU	78.51%	88.64%	62.69%
LeakyReLU	77.48%	88.98%	62.93%
LeakyReLU (A)	78.33%	88.55%	62.69%
PReLU	77.19%	88.77%	63.03%
PReLU (A)	79.83%	88.52%	62.94%
ConvReLU	**80.39%**	**89.13%**	**63.19%**

Datasets

We evaluate our methods on six datasets, including three datasets on image classification tasks and three datasets on text classification tasks.

Image classification datasets: For image classification tasks, we use three image datasets including Cifar10, Cifar100 (Krizhevsky, Hinton, and others 2009), and Tiny ImageNet (Yao and Miller 2015). Cifar10 and Cifar100 contain natural images with 32×32 pixels. Cifar10 consists of images from 10 classes, while the images in Cifar100 are drawn from 100 classes. Both datasets contain 50,000 training and 10,000 testing images. Tiny ImageNet dataset is a tiny version of ImageNet dataset (Deng et al. 2009). It has 200 classes, each of which contains 500 training, 50 validation, and 50 testing images.

Text classification datasets: For text classification tasks, we choose three datasets; those are, MR, AG's News, and Yelp Full. MR is a Movie Review dataset (Pang and Lee 2005), which includes positive and negative reviews for sentiment classification. Each sample in MR is a sentence with positive or negative sentiment label. AG's News is a topic classification dataset with four topics: World, Sports, Business, and Sci/Tech (Zhang, Zhao, and LeCun 2015). Yelp Full is formed based on the Yelp Dataset Challenge 2015 (Zhang, Zhao, and LeCun 2015).

Experimental Setup

For image and text classification tasks, we use different settings in terms of the model architecture.

Image classification settings: For image tasks, we mainly use the DenseNet architecture (Huang et al. 2017), which achieves state-of-the-art performances in various image classification tasks, including the ILSVRC 2012 challenge (Deng et al. 2009). On all three image datasets, we use DenseNet-40 as illustrated in Figure 4 with minor adjustments to accommodate different datasets. The network includes three dense blocks with a depth of 8 and a growth rate of 12. During training, the standard data augmentation scheme widely used in (Huang et al. 2017; Simonyan and Zisserman 2015; He et al. 2016) is applied on these image datasets for fair comparisons.

Text classification task settings: On text classification tasks, we employ the state-of-the-art VGG-like architecture in (Zhang, Zhao, and LeCun 2015) without using any unsupervised learning method. In this VGG-like network, there are 6 convolution layers, 3 pooling layers, and 3 fully-connected layers. Note that more recent models like ResNet and DenseNet have not achieved better performance on these tasks.

The following setups are shared for both experimental settings. In training, the SGD optimizer (LeCun, Bengio, and Hinton 2015) is used with a learning rate that starts from 0.1 and decays by 0.1 at the 150^{th} and 250^{th} epoch. The batch size is 128. These hyper-parameters are tuned on the Cifar10 and AG's News datasets, then applied on other datasets.

Comparison of ConvReLU with Other Activation Functions

Based on DenseNet-40 and VGG networks, we compare our proposed ConvReLUs with other activation functions on both image and text classification tasks. The results are summarized in Tables 1 and 2 for image and text datasets, respectively. In the experiments using ConvReLUs, we replace the ReLUs in the first dense block by ConvReLUs for image tasks, and replace the ReLUs in the first two convolution layers by ConvReLUs for text tasks. For both LeakyReLUs and PReLUs, we evaluate two replacement strategies; namely, one using the same replacement strategy as ConvReLU, and the other one with all layers using LeakyReLUs or PReLUs. For the LeakyReLU and PReLU experiments, we add (A) to indicate experiments using LeakyReLU or PReLU in all layers.

We can observe from both Tables 1 and 2 that our proposed ConvReLUs achieve consistently better performance than ReLUs, LeakyReLUs, and PReLUs. For baseline values listed for text classification tasks, they are the state-of-the-art without using unsupervised learning methods. Note that some studies (Zhang, Zhao, and LeCun 2015; Johnson and Zhang 2017) reported better results on these datasets by employing unsupervised learning methods. Since our method is orthogonal to these methods, we make use of the results without unsupervised learning for simplicity. These results demonstrate the effectiveness of our methods in both computer vision and text analysis fields. While for LeakyReLUs and PReLUs, they only perform better on some not all of datasets than ReLUs. Given that we do not change the model architectures and only add several thousands of training parameters, the performance improvements over other activation functions are significant.

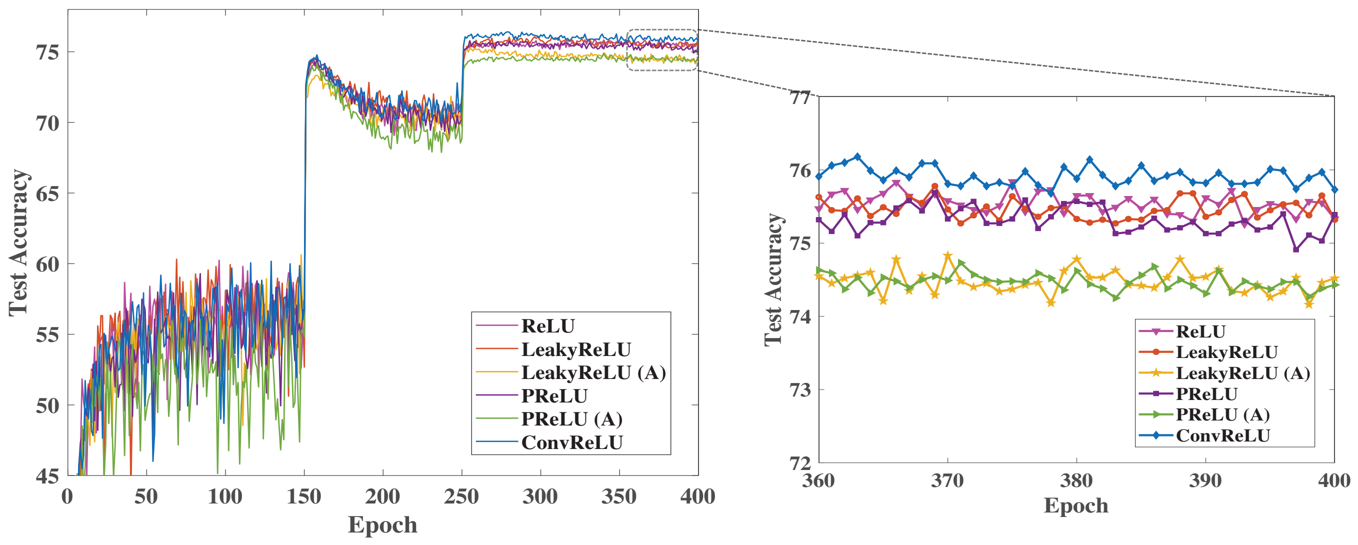

Figure 5: Comparison of top-1 accuracy curves on the testing dataset of Cifar100 for ReLU, LeakyReLU, PReLU, and ConvReLU. The symbol (A) in the legend indicates experiments with all layers using the Notably, both LeakyReLUs and PReLUs with partial corresponding activation function.

Table 3: Comparison of ConvReLU with ReLU on popular networks in terms of top-1 accuracy on image classification datasets Cifar10 and Cifar100.

	Cifar10		Cifar100	
Network	**ReLU**	**ConvReLU**	**ReLU**	**ConvReLU**
VGG-16	93.71%	**94.01%**	73.31%	**74.04%**
ResNet-18	94.37%	**94.48%**	75.17%	**75.97%**
DenseNet-40	94.29%	**94.72%**	75.87%	**76.41%**

Notably, both LeakyReLUs and PReLUs with partial replacement strategy achieve better performance than those using the full replacement strategy on most datasets. This shows that the proposed partial replacement strategy can not only overcome the dying ReLU problem but also retain the sparse representation in the network. This is shown to be effective for LeakyReLUs and PReLUs. In experimental studies, we will show that this partial replacement strategy also benefits our ConvReLUs.

Figure 5 shows the test accuracy curves of different activation functions on the Cifar100 dataset. We can easily observe from the figure that the performance of our proposed ConvReLU is consistently better than other activation functions by about a margin of 0.5%. Both LeakyReLU and PReLU with partial replacement strategy outperform their full replacement versions by a margin of about 1%. This again demonstrates the effectiveness of the partial replacement strategy.

Performance Study on Popular Networks

Our previous experiments on image classification tasks are mainly based on the DenseNet-40. It can be argued that our proposed activation function is only effective on this model architecture. In this section, we wish to investigate the performance of ConvReLUs on other popular networks such as VGG (Simonyan and Zisserman 2015) and ResNet (He et al. 2016). We compare the performance of ConvReLUs with ReLUs on the Cifar10 and Cifar100 datasets, and the results are summarized in Table 3. We can observe from these results that our proposed ConvReLUs are consistently better than ReLUs on both datasets using three popular deep networks. These deep networks employ different architecture designs and have different advantages. These results demonstrate the superiority of our ConvReLUs over ReLUs on multiple networks and datasets.

Table 4: Comparison of three initialization methods in terms of top-1 Accuracy on image classification datasets Cifar10 and Cifar100.

	Zero Init	**Random Init**	**Glot Init**
Cifar10	**94.72%**	94.55%	94.45%
Cifar100	**76.41%**	75.81%	75.44%

Performance Study of Different Initialization Methods

Based on the DenseNet-40, we perform comparisons of the three initialization methods discussed in Section on the Cifar10 and Cifar100 datasets. For all the three initialization methods, we employ L2-regularization to encourage the learned thresholds to be small. The results are summarized

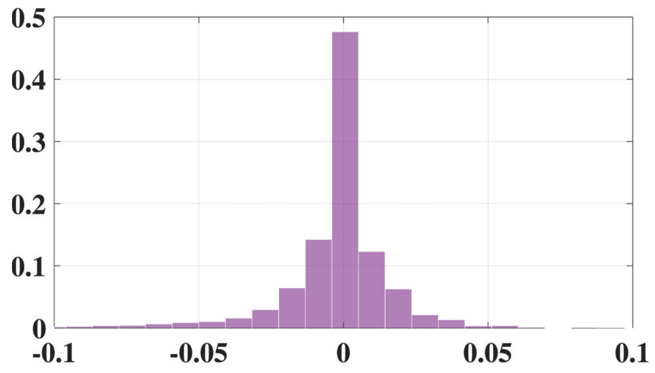

Figure 6: The distribution of threshold values learned in ConvReLUs based on DenseNet-40.

Table 5: Comparison of the three initialization strategies in terms of top-1 accuracy on image classification dataset Cifar10.

	R1 (First)	**R2 (Last)**	**R3 (All)**
Cifar10	**94.72%**	94.34%	94.55%

in Table 4. We can observe from the results that the zero initialization method outperforms random and glot initialization methods on both datasets. Figure 6 shows the distribution of the learned threshold parameters in ConvReLUs. The histogram in the figure demonstrates that the learned thresholds are very close to but not equal to zero. This is consistent with our expectation on using zero initialization. This simple parameter initialization method is very suitable for our proposed ConvReLUs.

Performance Study of Replacement Strategies

We investigate the performance of ConvReLU with different replacement strategies. We develop three replacement strategies: R1, R2, and R3. For strategy R1, we replace ReLUs with ConvReLUs in convolution layers of the first block in DenseNet-40. We use ConvReLUs instead of ReLUs in the last block for strategy R2. In the strategy R3, we replace all ReLUs by our proposed ConvReLUs.

We test these replacement strategies on DenseNet-40 using the Cifar10 dataset. The results are summarized in Table 5. We can see from the results that the first replacement strategy has the best performance among the three strategies. The observations reflected in Figure 5 demonstrate that the first replacement strategy also benefits LeakyReLUs and PReLUs. From the results, replacement of ReLUs in the beginning part of the network can overcome the dying ReLU problem and retain sparse representations for the linear classifiers simultaneously.

Parameter Number Study

Since our proposed activation function ConvReLUs involve more parameters than ReLUs, LeakyReLUs, and PReLUs, we study the number of additional parameters for ConvReLUs based on DenseNet-40. The results are given in Table 6. We can observe from the results that ConvReLUs only needs 0.28% or less additional parameters compared to ReLUs and other activation functions. We believe this marginal increase in parameter number is negligible and will not cause the over-fitting problem. This is a result of our parameter sharing scheme used in ConvReLUs, which allows each input neuron acts on different thresholds and avoids a large number of extra parameters.

Table 6: Comparison of ConvReLU with other activation functions in terms of parameter numbers based on DenseNet-40 on the dataset Cifar10.

Function	**#Params**	**Ratio**
ReLU	1,035,268	0.00%
LeakyReLU	1,035,268	0.00%
LeakyReLU (A)	1,035,268	0.00%
PReLU	1,035,280	0.00%
PReLU (A)	1,035,387	0.01%
ConvReLU	1,038,184	0.28%

Conclusion

In this work, we propose the adaptive ReLUs and its variant ConvReLUs to solve the dying ReLU problem suffered by ReLUs. The dying ReLU problem is mostly caused by its zero activation for negative arguments. By making the thresholds to be trainable instead of zero, adaptive ReLUs allow each input neuron to be activated by different trainable thresholds. Other than common parameter sharing methods such as layer sharing used in PReLUs, we propose a novel parameter sharing scheme, in which the trainable threshold parameters are shared based on the weights in convolution layers, thereby leading to our ConvReLUs. When computing with a specific weight in the convolution layer, the input neuron is activated by its corresponding threshold. In this way, the input neuron is acted by different thresholds without involving excessive extra parameters in neural networks.

The experimental results on image and text classification tasks demonstrate consistent performance improvements of ConvReLUs compared to ReLUs, LeakyReLUs, and PReLUs. For the extra parameters involved in ConvReLUs, we propose to use the zero initialization method with L2-regularization such that the trainable thresholds are close to but not equal to zero. Both the quantitative results and the histogram of threshold values confirm our intuitions and expectations on the zero initialization method. Finally, we propose the partial replacement strategy that helps to solve the dying ReLU problem and retain sparse representations for linear classifiers in neural networks. Our results indicate that the partial replacement strategy can help not only our ConvReLUs but also LeakyReLUs and PReLUs, which demonstrates its broad applicability.

Acknowledgements

This work was supported in part by National Science Foundation grants IIS-1908166 and DBI-1661289.

References

Chen, N.; Chen, X.; Yu, J.; and Wang, J. 2006. Afterhyperpolarization improves spike programming through lowering threshold potentials and refractory periods mediated by voltage-gated sodium channels. *Biochemical and Biophysical Research Communications* 346(3):938–945.

Chen, J.; Sathe, S.; Aggarwal, C.; and Turaga, D. 2017. Outlier detection with autoencoder ensembles. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, 90–98. SIAM.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Gao, H.; Yuan, H.; Wang, Z.; and Ji, S. 2019. Pixel transposed convolutional networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 1(1):1–1.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 315–323.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Islam, M. M., and Islam, N. 2016. Measuring threshold potentials of neuron cells using hodgkin-huxley model by applying different types of input signals. *Dhaka University Journal of Science* 64(1):15–20.

Johnson, R., and Zhang, T. 2017. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 562–570.

Klambauer, G.; Unterthiner, T.; Mayr, A.; and Hochreiter, S. 2017. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 972–981.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Laina, I.; Rupprecht, C.; Belagiannis, V.; Tombari, F.; and Navab, N. 2016. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth International Conference on 3D Vision*, 239–248. IEEE.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436.

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1(4):541–551.

LeCun, Y.; Bottou, L.; Orr, G.; and Muller, K. 1998a. Efficient backprop. In Orr, G., and K., M., eds., *Neural Networks: Tricks of the trade*. Springer.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998b. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Pang, B., and Lee, L. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, 115–124. Association for Computational Linguistics.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Seifter, J.; Sloane, D.; and Ratner, A. 2005. *Concepts in medical physiology*. Lippincott Williams & Wilkins.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. *Proceedings of the International Conference on Learning Representations*.

Xu, B.; Wang, N.; Chen, T.; and Li, M. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Yao, L., and Miller, J. 2015. Tiny imagenet classification with convolutional neural networks. *CS 231N*.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.