Parametric Exponential Linear Unit for Deep Convolutional Neural Networks

Ludovic Trottier
Department of Computer Science
and Software Engineering
Laval University
Québec, Canada
Email: ludovic.trottier.1@ulaval.ca

Philippe Giguère
Department of Computer Science
and Software Engineering
Laval University
Québec, Canada
Email: philippe.giguere@ift.ulaval.ca

Brahim Chaib-draa
Department of Computer Science
and Software Engineering
Laval University
Québec, Canada
Email: brahim.chaib-draa@ift.ulaval.ca

Abstract—**Object recognition is an important task for improving the ability of visual systems to perform complex scene understanding. Recently, the Exponential Linear Unit (ELU) has been proposed as a key component for managing bias shift in Convolutional Neural Networks (CNNs), but defines a parameter that must be set by hand. In this paper, we propose learning a parameterization of ELU in order to learn the proper activation shape at each layer in the CNNs. Our results on the MNIST, CIFAR-10/100 and ImageNet datasets using the NiN, Overfeat, All-CNN and ResNet networks indicate that our proposed Parametric ELU (PELU) has better performances than the non-parametric ELU. We have observed as much as a 7.28% relative error improvement on ImageNet with the NiN network, with only 0.0003% parameter increase. Our visual examination of the non-linear behaviors adopted by Vgg using PELU shows that the network took advantage of the added flexibility by learning different activations at different layers.**

I. Introduction

Recognizing objects using light from the visible spectrum is a essential ability for performing complex scene understanding with a visual system. Vision-based applications, such as face verification, robotic grasping or autonomous driving, require the fundamental skill of object recognition for carrying out their tasks. They must first identify the different elements in their surrounding environment in order to create a high-level representation of the scene. Since scene understanding is performed by analyzing the spatial relations and the taxonomy of the representation, the overall performance of the visual system depends on the capability of recognizing objects. Integrating novel object recognition advances for building fully-automated vision systems is one of the first steps towards general visual perception.

Over the past few years, Convolutional Neural Networks (CNNs) have become the leading approach in computer vision [1], [2], [3], [4], [5], [6]. Through a series of non-linear transformations, CNNs can process high-dimensional input observations into simple low-dimensional concepts. The key principle in CNNs is that features at each layer are composed of features from the layer below, which creates a hierarchical organization of increasingly abstract concepts. Since levels of organization are often seen in complex biological structures, CNNs are particularly well-adapted for capturing high-level abstractions in real-world observations.

The activation function plays a crucial role for learning representative features. The recently proposed Exponential Linear Unit (ELU) has the interesting property of reducing *bias shift* [7]. Defined as the change of a neuron's mean value due to weight update, bias shift can lead to oscillations and impede learning when not taken into account [7]. Clevert *et al.* [7] have shown that either centering the neuron values with a Batch Normalization layer [8] or using activation functions with negative values helps to manage this problem. Defined as identity for positive arguments and $a(\exp(h)-1)$ for negative ones (where $a=1$ in [7]), ELU's negative values for negative inputs make the activation function a well-suited candidate for reducing bias shift.

Choosing a proper ELU parameterization can however be relatively cumbersome considering that certain parameterizations are more suitable in some networks than others. The objective of this paper is to alleviate this limitation by learning a parameterization of the activation function, which we refer to as the Parametric ELU (PELU). We contribute in the following ways:

1) We define parameters controlling different aspects of the function and show how to learn them during back-propagation. Our parameterization preserves differentiability by acting on both the positive and negative parts of the function. It has the same computational complexity as ELU and adds only $2L$ additional parameters, where L is the number of layers.
2) We perform an experimental evaluation on the MNIST, CIFAR-10/100 and ImageNet tasks using the ResNet [9], Network in Network [10], All-CNN [11], Vgg [12] and Overfeat [13] networks. Our results indicates that PELU has better performances than ELU.
3) We evaluate the effect of using Batch Normalization (BN) before our PELU activation, and show that BN increases the error rate of ResNet.
4) We experiment with different PELU parameterizations, and show that the proposed one obtains the best performance among the possible parameterizations.
5) We finally show different PELU non-linear behaviors adopted during training by the VGG network. These

results highlight the effects of our parameterization in order to better understand the advantage of the activation.

The rest of the paper is organized as follows. We present related works in Section II and described our proposed approach in Section III. We detail our experimentations in Section IV and discuss the results in Section V. We conclude the paper in Section VI.

II. Related Work

Our proposed PELU activation function is related to other parametric approaches in the literature. Parametric ReLU (PReLU) [14] learns a parameterization of the Leaky ReLU (LReLU) [15] activation, defined as $\max\{h, 0\} + a \min\{h, 0\}$ where $a > 0$. PReLU learns a *leak* parameter a in order to find a proper positive slope for negative inputs. This prevents negative neurons from dying, i.e. neurons that are always equal to zero, which is caused by a null derivative that blocks the back-propagated error signal. Based on the empirical evidence that learning the leak parameter a rather than setting it to a predefined value (as done in LReLU) improves performance [14], our goal is further improving the performance of ELU by learning a proper parameterization of the function.

The Adaptive Piecewise Linear (APL) unit aims learning a weighted sum of S parametrized Hinge functions [16]. One drawback of APL is that the number of points at which the function is non-differentiable increase linearly with S. Differentiable activation functions usually give better parameter updates during back-propagation than activation functions with non-differentiable points [2]. Moreover, although APL has the flexibility to be either a convex or non-convex function, the rightmost linear function is forced to have unit slope and zero bias. This may be an inappropriate constraint which could affect the CNN ability to learn representative features.

Another activation function is Maxout, which outputs the maximum over K affine functions for each input neuron [17]. The main drawback of Maxout is that it multiplies by K the amount of weights to be learned in each layer. In the context of CNNs where the max operator is applied over the feature maps of each K convolutional layers, the increased computational burden can be too demanding for deep network. Unlike Maxout, our PELU adds only $2L$ parameters, where L is the number of layers, which makes our activation as computationally demanding as the original ELU function.

The S-Shaped ReLU (SReLU) imitates the Webner-Fechner law and the Stevens law by learning a combination of three linear functions [18]. Although this parametric function can be either convex or non-convex, SReLU has two points at which it is non-differentiable. Unlike SReLU, our PELU is fully differentiable, since our parameterization acts on both the positive and negative sides of the function. This in turn improves the back-propagation weight and bias updates.

III. Parametric Exponential Linear Unit

The standard Exponential Linear Unit (ELU) is defined as identity for positive arguments and $a(\exp(h) - 1)$ for negative

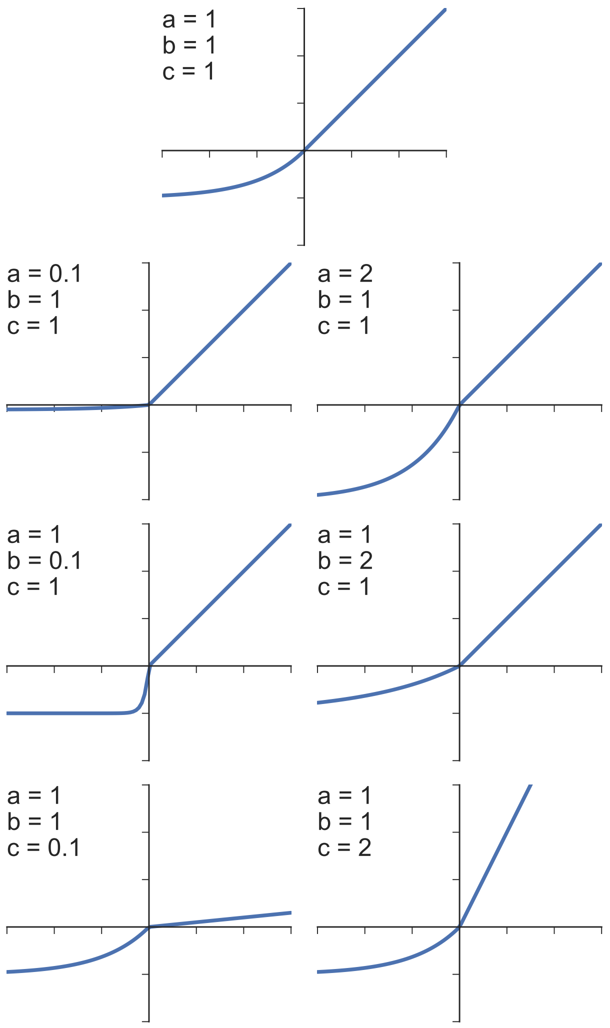


Fig. 1. Effects of parameters a, b and c on the Exponential Linear Unit (ELU) activation function. The original ELU is shown at the top, where $a = b = c = 1$. We show the effect of a on the second row, the effect of b on the third row, and the effect of c on the fourth row. The saturation point decreases when a increases, the function saturates faster when b decreases, and the slope of the linear part increases when c increases.

arguments ($h < 0$) [7]. Although the parameter a can be any positive value, Clevert *et al.* [7] proposed using $a = 1$ to have a fully differentiable function. For other values $a \neq 1$, the function is non-differentiable at $h = 0$. Directly learning parameter a would break differentiability at $h = 0$, which could impede back-propagation [2].

For this reason, we first start by adding two additional parameters to ELU:

$$f(h) = \begin{cases} ch & \text{if } h \geq 0 \\ a(\exp(\frac{h}{b}) - 1) & \text{if } h < 0 \end{cases}, \quad a, b, c > 0\,, \qquad (1)$$

We have ch for positive arguments ($h \geq 0$) and $a(\exp(\frac{h}{b}) - 1)$ for negative arguments ($h < 0$). The original ELU can be recovered when $a = b = c = 1$. As shown in Figure 1, each parameter controls different aspects of the activation. Parameter c changes the slope of the linear function in the positive quadrant (the larger c, the steeper the slope), parameter b affects the scale of the exponential decay (the larger b, the smaller the decay), while a acts on the saturation point in the negative quadrant (the larger a, the lower the saturation point). Constraining the parameters in the positive quadrant forces the activation to be a monotonic function, such that reducing the weight magnitude during training always lowers the neuron contribution.

Using this parameterization, the network can control its non-linear behavior throughout the course of the training phase. It may increase the slope with c, the decay with b or lower the saturation point with a. However, a standard gradient update on parameters a, b, c would make the function non-differentiable at $h = 0$ and impair back-propagation. Instead of relying on a projection operator to restore differentiability after each update, we constrain our parameterization to always have differentiability at $h = 0$. By equaling the derivatives on both sides of zero, solving for c gives $c = \frac{a}{b}$ as solution. The proposed Parametric ELU (PELU) is then as follows:

$$f(h) = \begin{cases} \frac{a}{b}h & \text{if } h \geq 0 \\ a(\exp(\frac{h}{b}) - 1) & \text{if } h < 0 \end{cases}, \quad a, b > 0 \tag{2}$$

With this parameterization, in addition to changing the saturation point and exponential decay respectively, both a and b adjust the slope of the linear function in the positive part to ensure differentiability at $h = 0$.

PELU is trained simultaneously with all the network parameters during back-propagation. Using the chain rule of derivation, the gradients of f with respect to a, b is given by:

$$\frac{\partial f(h)}{\partial a} = \begin{cases} \frac{h}{b} & \text{if } h \geq 0 \\ \exp(h/b) - 1 & \text{if } h < 0 \end{cases} \tag{3}$$

$$\frac{\partial f(h)}{\partial b} = \begin{cases} -\frac{ah}{b^2} & \text{if } h \geq 0 \\ -\frac{a}{b^2}\exp(h/b) & \text{if } h < 0 \end{cases}. \tag{4}$$

For preserving parameter positivity after the updates, we constrain them to always be greater than 0.1.

IV. Experimentations

In this section, we present our experiments in supervised learning on the CIFAR-10/100 and ImageNet tasks.

A. MNIST Auto-Encoder

As first experiment, we performed unsupervised learning, which is the task of learning feature representations from unlabeled observations. Unsupervised learning can be useful in cases like deep learning data fusion [19]. For evaluating our proposed PELU activation, we trained a deep auto-encoder on unlabeled MNIST images [20]. We refer to this network as DAA-net. The encoder has four fully connected layers of sizes 1000, 500, 250, 30, and the decoder is symmetrical to the encoder (the weights are not tied). We used Dropout with probability 0.2 after each activation [21]. For ReLU, we put a Batch Normalization (BN) layer before the activation. We trained DAA-Net with RMSProp [22] at a learning rate of 0.001, smoothing constant of 0.9 and a batch size of 128.

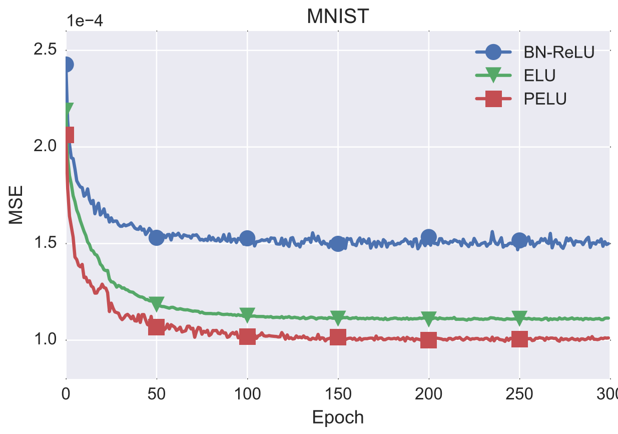


Fig. 2. Auto-encoder results on the MNIST task. We compare PELU to ELU, and include BN-ReLU as additional reference. Compared to ELU, PELU obtained a lower test mean squared error.

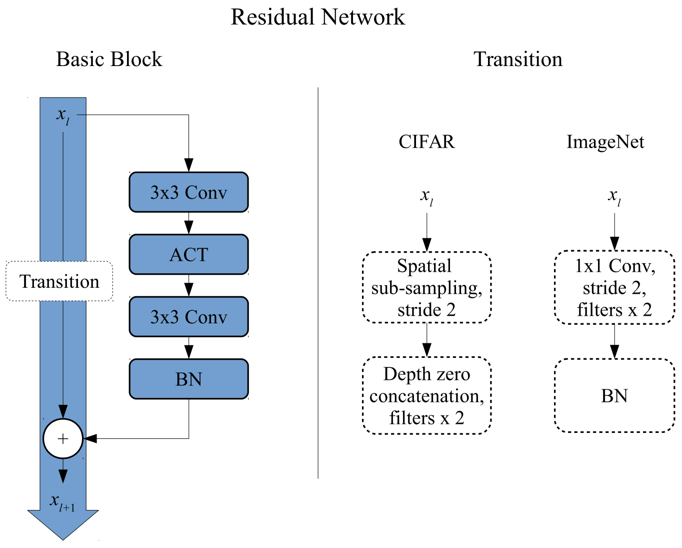
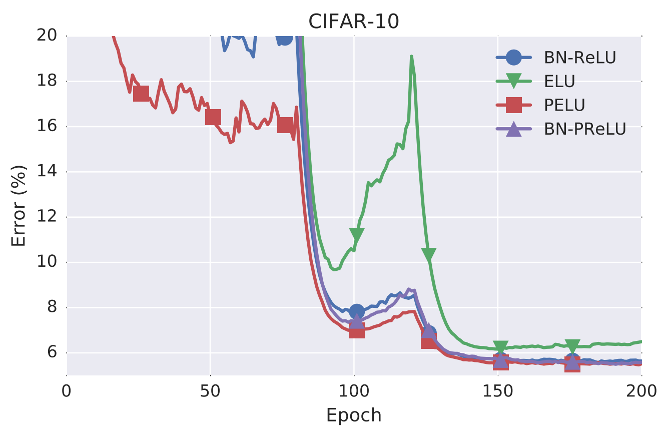


Fig. 3. Residual network building block structure. On the left, the main basic block structure, and on the right, the transition block structure for reducing the input spatial dimensions and increasing the number of filters. For our CIFAR experiments, we opted for sub-sampling followed by zero concatenation as transition block, while for our ImageNet experiments, we opted for strided convolution followed by batch normalization as transition block.

Figure 2 presents the progression of test mean squared error averaged over five tries of DAA-Net on MNIST dataset. These results show that PELU outperformed ELU and ReLU for both convergence speed and reconstruction error. PELU converged approximatively at epoch 75 with a MSE of $1.04\text{e}{-4}$, while ELU converged at epoch 100 with a MSE of $1.12\text{e}{-4}$ and ReLU at epoch 100 with a MSE of $1.49\text{e}{-4}$.

B. CIFAR-10/100 Object Recognition

We performed object classification on the CIFAR-10 and CIFAR-100 datasets (60,000 32x32 colored images, 10 and 100 classes respectively) [1]. We trained a 110-layer residual network (ResNet) following Facebook's Torch implementation



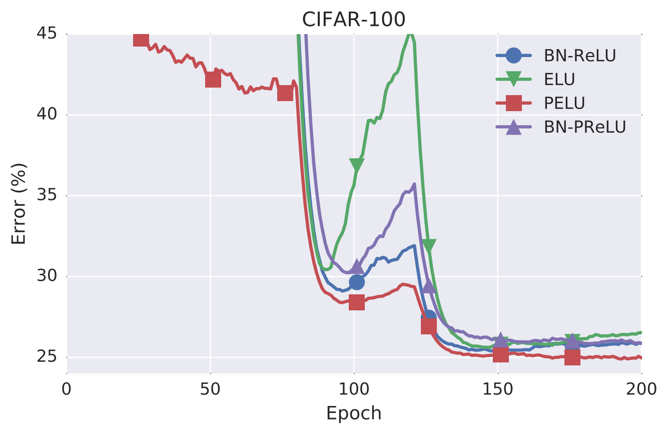


Fig. 4. ResNet 110 layers test error (in %) medians over five tries on both CIFAR-10 and CIFAR-100 datasets. We compare PELU to ELU, and also include BN-ReLU and BN-PReLU as additional references. PELU has a better convergence and lower recognition errors than ELU.

TABLE I
RESNET 110 LAYERS TEST ERROR (IN %) ON BOTH CIFAR-10 AND CIFAR-100 DATASETS. WE REPORT THE MEAN ERROR OVER THE LAST FIVE EPOCHS AND THE MINIMUM ERROR OVER ALL EPOCHS (INSIDE PARENTHESIS) OF THE MEDIAN ERROR OVER FIVE TRIES. OUR ELU PARAMETERIZATION IMPROVES PERFORMANCE, BUT USING BN BEFORE THE ACTIVATION WORSEN THE PERFORMANCE.

ACT	CIFAR-10	CIFAR-100
BN-ReLU	5.67 (5.41)	25.92 (24.99)
ELU	6.55 (5.99)	26.59 (25.08)
PELU	**5.51 (5.36)**	**25.02 (24.55)**
BN-PReLU	5.61 (**5.36**)	25.83 (25.50)
BN-PELU	6.24 (5.85)	26.04 (25.38)
BN-ELU	11.20 (10.39)	35.51 (34.75)

`fb.resnet.torch`[1]. In order not to favor PELU to the detriment of ELU and BN+ReLU, we performed minimal changes by only replacing the activation function.

The building block structure for the network is shown in Figure 3. We show the basic block structure on the left of Figure 3 and the transition block structure on the right of Figure 3. The ACT module can be PELU, ELU, ReLU or PReLU, with or without BN. The network contains mainly basic blocks, and a few transition blocks for reducing the spatial dimensions of the input image and increasing the number of filters. The ResNet for our CIFAR experiments has a transition block structure with spatial sub-sampling and zero concatenation, while the ResNet for our ImageNet experiments (see Section IV-D) has a transition block structure with a strided convolution followed by Batch Normalization.

[1]https://github.com/facebook/fb.resnet.torch

To train the network, we used stochastic gradient descent with a weight decay of $1\mathrm{e}{-3}$, momentum of 0.9 and mini batch-size of 256. The learning rate starts at 0.1 and is divided by 10 after epoch 81, and by 10 again after epoch 122. We performed standard center crop + horizontal flip for data augmentation: four pixels were added on each side of the image, and a random 32 x 32 crop was extracted, which was randomly flipped horizontally. Only color-normalized 32 x 32 images were used during the test phase.

Figure 4 presents ResNet test error (in %) medians over five tries on both CIFAR datasets. ResNet obtained a minimum median error rate on CIFAR-10 of 5.41% with BN+ReLU, 5.99% with ELU, 5.36% with PELU and 5.26% with BN-PReLU, while ResNet obtained a minimum median error rate on CIFAR-100 of 24.99% with BN+ReLU, 25.08% with ELU, 24.55% with PELU and 25.50% with BN+PReLU. In comparison to ELU, PELU obtained a relative improvement of 10.52% and 2.11% on CIFAR-10 and CIFAR-100 respectively. It is interesting to note that PELU only adds 112 additional parameters, a negligible increase of 0.006% over the total number of parameters.

We observed that PELU has a better convergence behavior than ELU. As shown in Figure 4, ELU has a large test error rate increase at the end of the second stage of the training phase on both CIFAR-10 and CIFAR-100 datasets. Although PELU has also a test error rate increase at the end of the second stage, it does not increase as high as ELU. We further observe a small test error rate increase at the end of the training phase for ELU, while PELU converges in a steady way without a test error rate increase. These results show that training a ResNet with our parameterization can improve the performance and the convergence behavior over a ResNet with ELU activation.

Compared to ReLU, PReLU obtained a smaller minimum median error rate on CIFAR-10 and a smaller average median error rate on CIFAR-100. As shown in Table I, PReLU obtained a minimum median error rate of 5.36 compared to 5.41 on CIFAR-10, and an average median error rate of 25.83 compared to 25.92 on CIFAR-100. Although PReLU obtained the same minimum median error rate than PELU on CIFAR-10, it is significantly higher on CIFAR-100. Note that our main contribution is showing performance improvement over ELU, and that we only add PReLU as an additional reference. Nonetheless, we observe that our PELU parameterization of ELU obtains higher relative improvements than the PReLU parameterization of ReLU.

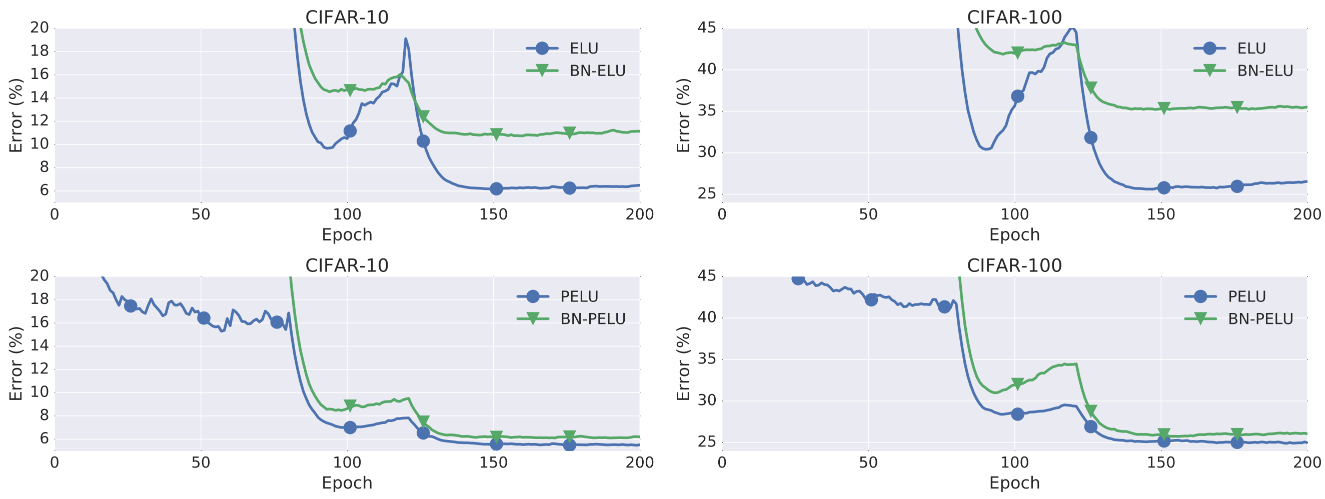


Fig. 5. Effect of using BN before ELU (first row) and PELU (second row) activations in a ResNet with 110 layers on both CIFAR-10 and CIFAR-100 datasets. We show the convergence behavior of the median test error over five tries. In both cases, BN worsen performance of ELU and PELU. Note that we still use BN after the second conv layer, as seen in Figure 3.

C. *Understanding the effect of Batch Normalization*

In this section, we show that using BN before our PELU activation has a detrimental effect on its performance. Figure 5 presents the influence of using BN before ELU and PELU in a ResNet with 110 layers on both CIFAR-10 and CIFAR-100 datasets. We trained the networks using the same framework as in Section IV-B, but added BN before each activate. Note that in all cases, we use BN after the second convolutional layer in the basic block (see Figure 3).

The results show a large error rate increase on both CIFAR-10 and CIFAR-100 dataset for each ELU and PELU activation. The minimum median test error for ELU increases from 5.99% and 25.08% to 10.39% and 34.75% on CIFAR-10 and CIFAR-100 respectively, while for PELU it increases from 5.36% and 24.55% to 5.85% and 25.38%. We also observe that the relative error rate increase for our PELU is smaller than for ELU. Indeed, ELU has a relative minimum test error rate increase of 73% and 39% on CIFAR-10 and CIFAR-100 respectively, while PELU has 9% and 3%. Although this shows that our PELU parameterization reduces the detrimental effect of using BN before the activation, PELU should not be preceded by BN.

D. *ImageNet Object Recognition*

We tested the proposed PELU on the ImageNet 2012 task (ILSVRC2012) using four different network architectures: ResNet18 [9], Network in Network (NiN) [10], All-CNN [11] and Overfeat [13]. The ResNet18 building block structure is shown in Figure 3. In order not to favor PELU to the detriment of ELU and BN+ReLU, we performed minimal changes by only replacing the activation function. We used either PELU, ELU or BN+ReLU for the activation module. Each network was trained following Chintala's Torch implementation `imagenet-multiGPU.torch` [2] with the training regimes

TABLE II
IMAGENET TRAINING REGIMES FOR MODIFYING THE LEARNING RATE AND THE WEIGHT DECAY. WE TRAINED NIN AND RESNET18 USING THE FIRST ONE, AND TRAINED ALL-CNN AND OVERFEAT USING THE SECOND ONE. THE FIRST REGIME STARTS AT A HIGHER LEARNING RATE, BUT HAS A LARGER DECAY THAN THE SECOND REGIME.

	Regime #1 (ResNet18, NiN)			
Epoch	1	10	20	25
Learning Rate	1e-1	1e-2	1e-3	1e-4
Weight Decay	5e-4	5e-4	0	0

	Regime #2 (Overfeat, AllCNN)				
Epoch	1	19	30	44	53
Learning Rate	1e-2	5e-3	1e-3	5e-4	1e-4
Weight Decay	5e-4	5e-4	0	0	0

shown in Table II. Regime #1 starts at a higher learning rate than regime #2, and has a larger learning rate decay.

Figure 6 presents the TOP-1 error rate (in %) of all four networks on ImageNet 2012 validation dataset. In all cases, the networks using PELU outperformed the networks using ELU. NiN obtained the best result of 36.06% with PELU, which corresponds to a relative improvement of 7.29% compared to ELU (40.40%). Since only 24 additional parameters were added to the network, this performance improvement indicates that PELU's parameterization acts in a different way than the weights and biases. Adding 24 additional weights throughout the network would not have been sufficient to increase the representative ability enough to get the observed performance improvement. Since such a low number of weights cannot significantly increase the expressive power of the network, these results indicate that the networks benefit from PELU.

As shown in Figure 6, the training regimes have an interesting effect on the convergence of the networks. The performance of PELU is closer to the performance of ELU for regime #2, but is significantly better than ELU for regime #1.

[2]https://github.com/soumith/imagenet-multiGPU.torch

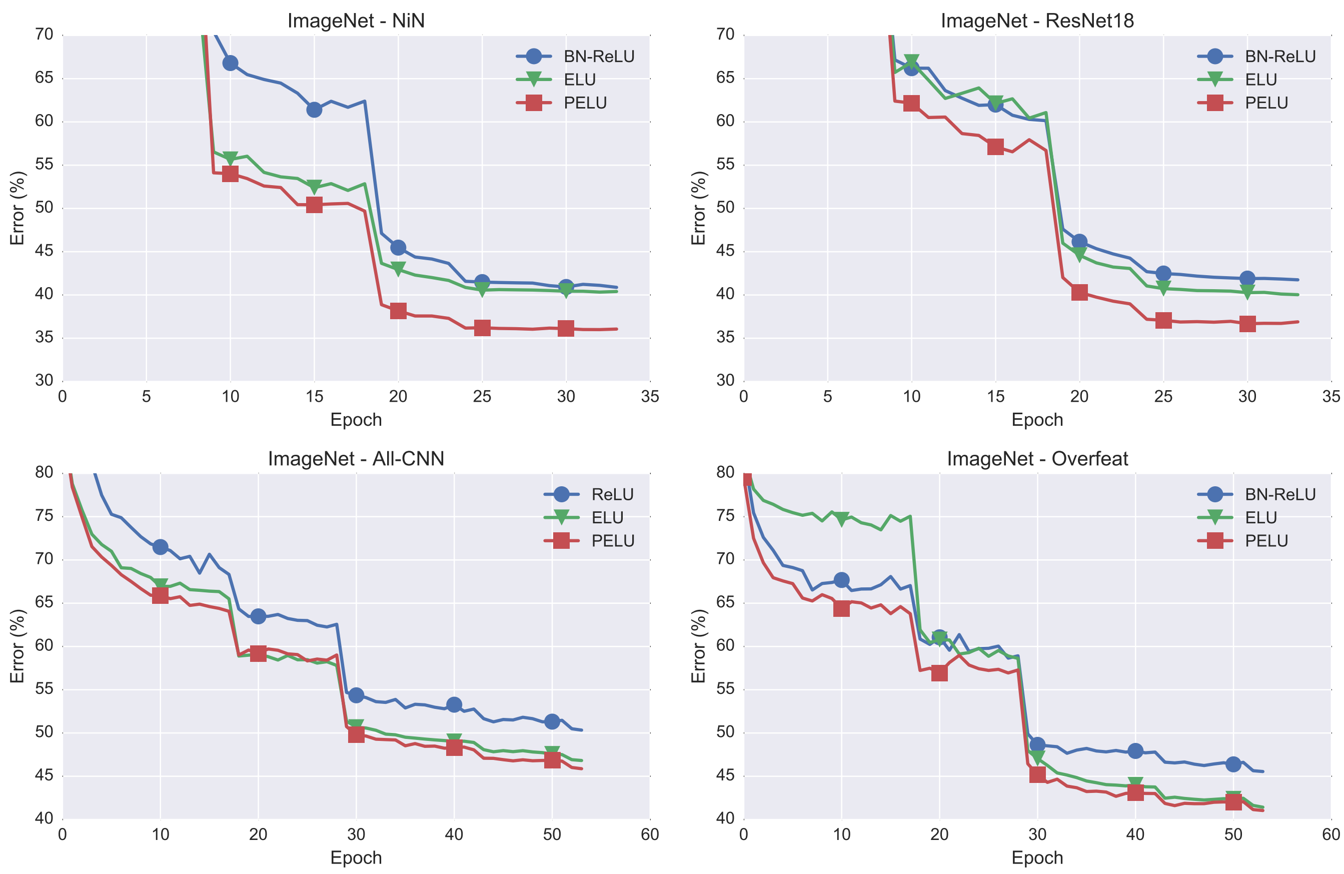


Fig. 6. TOP-1 error rate progression (in %) of ResNet18, NiN, Overfeat and All-CNN on ImageNet 2012 validation set. NiN and ResNet18 (top row) used training regime #1, while All-CNN and Overfeat (bottom row) used training regime #2 (see Table II). PELU has the lowest error rates for all networks. Regime #1 shows a greater performance gap between ELU and PELU than regime #2.

TABLE III
RESNET 110 LAYERS TEST ERROR (IN %) ON BOTH CIFAR-10 AND CIFAR-100 DATASETS. WE REPORT THE MEAN ERROR OVER THE LAST FIVE EPOCHS AND THE MINIMUM ERROR OVER ALL EPOCHS (INSIDE PARENTHESIS) OF THE MEDIAN ERROR OVER FIVE TRIES. WE COMPARE EACH FOUR PELU CONFIGURATIONS. OUR PROPOSED PELU CONFIGURATION $(a, 1/b)$ OBTAINED THE BEST PERFORMANCE.

Configuration	CIFAR-10	CIFAR-100
$(a, 1/b)$	**5.51 (5.36)**	**25.02 (24.55)**
$(1/a, 1/b)$	5.73 (5.60)	25.68 (25.17)
$(1/a, b)$	6.51 (6.00)	26.33 (25.48)
(a, b)	6.74 (6.12)	26.20 (25.24)

We also observe that the error rates of All-CNN and Overfeat with PELU increase by a small amount starting at epoch 44, but stay steady for ELU and ReLU. These results suggest that training regimes with larger learning rates and decays help PELU to obtain a better performance improvement.

E. Experimenting with Parameter Configuration

The proposed PELU activation function (2) has two parameters a and b, where a is used with a multiplication and b with a division. A priori, any of the four configurations (a, b), $(a, 1/b)$, $(1/a, b)$ or $(1/a, 1/b)$ could be used as parameterization. Note that these configurations are not reciprocal due to weight decay, which favors low weight magnitude. For instance, favoring low magnitude for parameter b with the (a, b) configuration favors a high PELU slope a/b. On the contrary, favoring low magnitude for parameter b with the $(a, 1/b)$ favors a low PELU slope ab. In order to better understand the difference between each configuration, we performed an experimental evaluation on the CIFAR-10 and CIFAR-100 datasets using the 110-layers ResNet as defined in Section IV-B.

As shown in Table III, our proposed parameterization $(a, 1/b)$ obtained the best accuracy. Parameterization $(a, 1/b)$ obtained minimum test error medians of 5.36% and 24.55% on CIFAR-10 and CIFAR-100 respectively, while $(1/a, 1/b)$ obtained 5.60% and 25.17%, $(1/a, b)$ obtained 6.00% and 25.48%, and (a, b) obtained 6.12% and 25.24%. These results also show that the two parameterizations with $1/b$ obtained a significantly lower error rate than the two parameterizations with b. From the convergence behavior in Figure 7, we see that the parameterizations with b have a larger error increase during the second stage of the training phase than parameterizations with $1/b$, and converge to lower error rates. These results concur with our observations in Section III of the effect of the parameters. Since weight decay pushes the weight magnitude

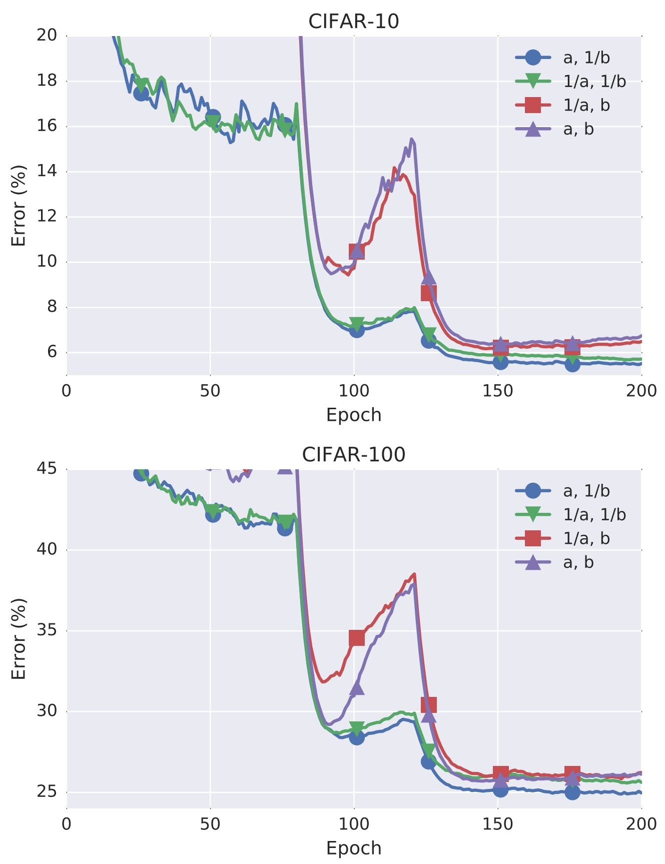


Fig. 7. Experimenting with the PELU parameter configuration in a ResNet with 110 layers on both CIFAR-10 and CIFAR-100 datasets. We show the convergence behavior of the median test error over five tries. Our proposed parameterization $(a, 1/b)$ obtained the best performance.

towards zero, the function saturates slower as b decreases with parameterizations using b. This encourages the function to have an almost linear shape on all its input values, which removes the non-linear characteristic of the activation. On the contrary, the function saturates faster a b decreases with parameterizations using $1/b$. This helps the activation to keep its non-linear nature, which explains the observed performance gaps between b and $1/b$.

F. Parameter Progression

We perform a visual evaluation of the non-linear behaviors adopted by a Vgg network during training on the CIFAR-10 dataset [12]. Figure 8 shows the progression of the slope ($\frac{a}{b}$) and the negative of the saturation point (parameter a) for PELU at each layer of Vgg.

We can see different behaviors. At layers 2, 4, 7 and 10, the slope quickly increased to a large value, then decreased and converged at a value near 1. As for parameter a, it quickly converged to a value near 0. A slope near 1 and a negative saturation near 0 indicates that the network learned activations having the same shape as ReLU. This is an interesting result because ReLU has the important effect of promoting activation sparsity [23], [15]. Although we do not have a clear understanding to why the network increases the slope then decreases

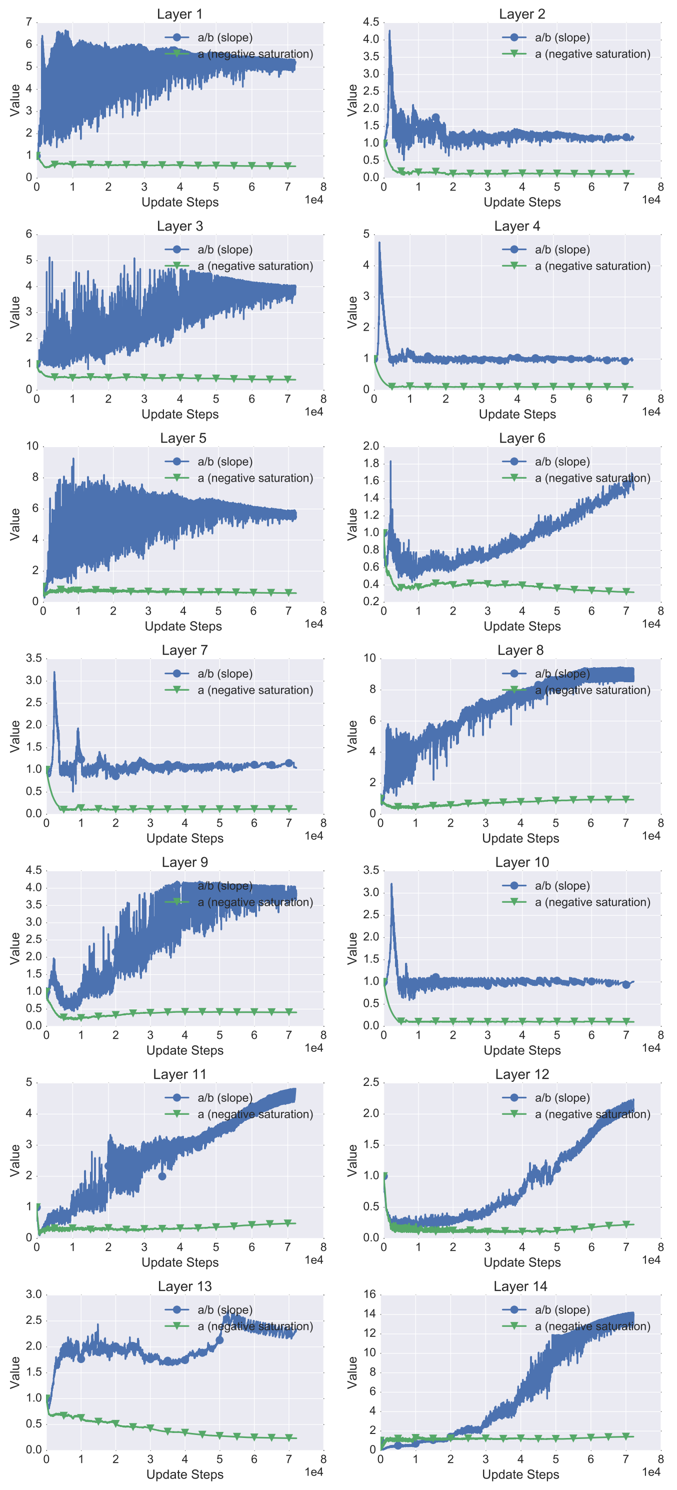


Fig. 8. PELU parameter progression at each layer of Vgg trained on CIFAR-10. We present the variation of the slope ($\frac{a}{b}$) and the negative saturation (parameter a). The network adopted different non-linear behaviors throughout the training phase.

it before converging to ReLU, we believe that increasing the slope helps early during training to disentangle redundant neurons. Since peak activations scatter more the inputs than flat ones, spreading values may allow the network to declutter neurons activating similarly to the same input patterns.

Another interesting observation is that, apart from the *ReLU* layers (layers 2, 4, 7 and 10), the negative saturations of all layers converged at values other than 0. For instance, parameter a converges to a value near 0.5 at layer 1, while it converges to a value near 2 at layer 14. A negative saturation other than zero indicates that the learned PELU activations outputs negative values for negative arguments. The Vgg network had the possibility to learn all activation functions with a zero negative saturation (i.e. shaped like ReLU), but opted for a majority of activations with a non-zero negative saturation. Having activation functions with negative values has been previously analyzed in the context of the standard ELU activation, and it has been proposed that it helps to manage bias shift [7]. These results constitute an additional experimental evidence that this characteristic is important for the network.

V. Discussion

During all our experiments with ELU and PELU, we did not use Batch Normalization (BN) before the activations. This is due to the detrimental effect of preceding PELU and ELU with BN, as we have observed in Section IV-C with our ResNet experiments on CIFAR-10 and CIFAR-100. Although this detrimental effect has also been previously observed with ELU by Clevert and his coworkers [7], it is unclear why BN before ELU and PELU increases error rate, but reduces error rate before ReLU. One important difference is that ReLU is positively scale invariant and ELU is not. Indeed, for ReLU we have $\max\{0, kx\} = k \max\{0, x\}$, where $k \geq 0$, while for ELU, which can be expressed as $\max\{0, x\} + \min\{0, \exp\{x\} - 1\}$, we have $\min\{0, \exp\{kx\} - 1\} \neq k \min\{0, \exp\{x\} - 1\}$. The fact that ReLU is positively scale invariant and ELU is not may be part of the reason why BN before ReLU helps but harms before ELU. Given that BN performs mean and standard deviation scaling, followed by an affine transformation (scaled by γ and shifted by β), using a positively scale invariant activation function may be essential for BN to properly reduce internal covariate shift [8] or manage bias shift [7]. We could validate this hypothesis by experimenting with a new positively scale invariant activation function and observing whether BN helps or not. We leave this idea as future work.

VI. Conclusion

Object recognition is an essential ability for improving visual perception in automated vision systems performing complex scene understanding. In a recent work, the Exponential Linear Unit (ELU) has been proposed as a key element in Convolutional Neural Networks (CNNs) for reducing bias shift, but has the inconvenience of defining a parameter that must be set by hand. In this paper, we proposed the Parametric ELU (PELU) that alleviates this limitation by learning a parameterization of the ELU activation function. Our results on the CIFAR-10/100 and ImageNet datasets using the ResNet, NiN, All-CNN and Overfeat networks show that CNNs with PELU have better performance than CNNs with ELU. Our experiments with Vgg have shown that the network uses the added flexibility provided by PELU by learning different activation shapes at different locations in the network. Parameterizing other activation functions, such as Softplus, Sigmoid or Tanh, could be worth investigating.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation for providing the Tesla K20, K80 and Titan X for our experiments.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *CVPR*, 2015, pp. 3156–3164.

[4] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *NIPS*, 2015, pp. 2017–2025.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[6] J. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?" *PAMI*, vol. 38, no. 4, pp. 814–830, 2016.

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

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[9] A. Shah, E. Kadam, H. Shah, and S. Shinde, "Deep residual networks with exponential linear unit," *arXiv preprint arXiv:1604.04112*, 2016.

[10] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[11] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *ICLR (workshop track)*, 2015.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015, pp. 1026–1034.

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

[16] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning activation functions to improve deep neural networks," *arXiv preprint arXiv:1412.6830*, 2014.

[17] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *ICML*, 2013, pp. 1319–1327.

[18] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," *arXiv preprint arXiv:1512.07030*, 2015.

[19] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *NIPS*, 2012, pp. 2222–2230.

[20] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.

[22] G. H. Tijmen Tieleman, "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude." *COURSERA: Neural Networks for Machine Learning*, vol. 4, 2012.

[23] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814.