LAYER-WISE BALANCED ACTIVATION MECHANISM

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a novel activation mechanism called LayerAct mechanism to develop layer-wise balanced activation functions that converge faster and perform better than existing activation functions. During the backpropagation in neural networks, the scale of activation function determines how much the parameters will be trained using each sample. This fact indicates that training of a neural network can be biased against samples when the distribution of activation is imbalanced among samples. With a simple experiment on the unnormalized network with rectified linear units (ReLUs) for activation, we show that there is a relationship between the sum of the activation scale and the training loss, which indicates that an imbalanced activation scale among samples can result in a bias in learning. The layer normalization (LayerNorm) can be used to avoid such problems of bias in learning by balancing the layer-wise distribution of inputs for activation functions. However, LayerNorm loses the mean and variance statistics of activated instances among samples during re-scaling and re-centering. Our proposed LayerAct mechanism balances the layer-wise distribution of activation outputs for all samples without re-scaling and re-centering; this way, LayerAct functions avoid not only the problem of bias in learning, but also the dilution problem of key statistics. LayerAct functions allow negative activation outputs when the activated signals have to be negative; thus, the machine can avoid bias shifts during learning, enabling rich representations at the end. Moreover, the proposed LayerAct mechanism can be used with the batch normalization (BatchNorm). Experiments show that LayerAct functions outperform the unbalanced element-level activation functions on two benchmark image classification datasets, CIFAR10 and CIFAR100. Given the essential role of activation in traditional multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and modern deep learning frameworks, our original work on the layer-wise activation fundamentally addressing the core mechanism of learning through multiple layers will contribute in developing high-performance machine learning frameworks.

1 INTRODUCTION

The success of deep neural networks (DNNs) partly relies on the activation function that identifies nonlinear relationships in data, and the activation controls training of a network by determining each parameter's scale of gradient in forward and back-propagation. Most of the existing activation functions have an element-level activation mechanism as they operate on a single element of the input vector in a layer individually (Nair & Hinton, 2010; Maas et al., 2013; Ramachandran et al., 2017). Such an element-level activation mechanism leads to imbalanced distributions of activation when the distribution of inputs to the layer is imbalanced among training samples, which results in an imbalanced distribution of gradient scale in back-propagation. In case of using rectified linear units (ReLUs, Nair & Hinton (2010)) with gradient scale of 0 or 1, the training samples with many activated elements naturally contribute more parameters to be optimized than other samples with a small number of activated elements. As such, the element-level activation mechanism of back-propagation induces a bias in the training of a network against samples; more parameters in the network are fitted to some training samples more than others. LayerNorm (Ba et al., 2016) that balances the distribution of activation and scale of gradient can address such a bias problem by re-scaling and re-centering, but the representation of activation becomes similar across all samples as the mean and variance of the layer become not different among samples (Lubana et al., 2021).

In this paper, we propose a novel *layer-level activation* mechanism to develop layer-wise balanced activation (LayerAct) functions, which balance the distribution of activation among samples without re-scaling or re-centering the inputs directly. Unlike the general mechanism of the activation functions that the nonlinearity criterion (i.e., scale function) does not change as it depends only on the single element of a layer, the criterion of LayerAct functions relies on the distribution of all inputs of a layer. This way, LayerAct functions do not lose the diversity of statistics among samples as they do not directly re-scale or re-center the inputs. With such a mechanism of layer-level activation, LayerAct enables a layer-wise balanced distribution of activation, whether the distribution of inputs is imbalanced or not, with different statistics of activation outputs among samples. This property makes all training samples have a fair amount of influence on training parameters in back-propagation, thereby ensuring rich representations.

LayerAct functions also comply with key requirements of activation functions and their beneficial properties. Recent studies, such as exponential linear units (ELUs, Clevert et al. (2015)) or flexible rectified linear units (FReLUs, Qiu et al. (2018)), emphasize the important properties of an activation function: (i) the function should enable negative values in activation and (ii) the activation mean should be "zero-like" (i.e., push the mean of activation to zero as possible). To achieve these properties, existing activation functions involve additional learnable parameters (Clevert et al., 2015; Qiu et al., 2018), which increase the complexity of the network. However, LayerAct functions allow negative values and the zero-like mean without any additional parameters as the distribution of the given inputs to the functions determines. As such, LayerAct functions with layer-wise activation fundamentally address the core mechanism of learning through multiple layers.

This is an original study to tackle the imbalanced activation across the elements of a network layer, to devise the concept of layer-wise balanced activation, and to develop a mechanism and activation functions to enable it. To present the necessity and contribution of our proposed LayerAct mechanism, the remainder of this paper is organized as follows. Section 2 reviews preliminaries and related studies on activation and normalization, as well as formulates our motivation. Section 3 explains the proposed LayerAct mechanism and functions. In Section 4, we show a simple experiment with the MNIST dataset that there is a relationship between the sum of activation scale and the loss during training. The implication of this experimental result, "the training of a network can be biased among samples if the scale of activation function in a layer is not balanced", empirically shows the necessity of layer-level balanced activation for efficient and effective training. Accordingly, we compare LayerAct functions with other activation functions using two image datasets, CIFAR10 and CIFAR100. As expected, LayerAct functions showed faster convergence and better performance than existing element-level activation functions. We believe the proposed LayerAct mechanism will contribute in developing high-performance machine learning frameworks, given the indispensable use and significance of activation in traditional multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and modern deep learning research and practice.

2 Background and Motivation

2.1 Activation scale in networks

Consider the l^{th} layer in an MLP with linear projection and an activation function. The computation of the layer with input vector x^l, weight matrix W^l and non-linear function f as the activation function is defined as follows:

$$y^l = {W^l}^T x^l, \quad a^l = f\left(y^l\right) \tag{1}$$

where y^l and a^l are the output vector of linear projection and activation of the l^{th} layer, respectively. The output vector y^l of linear projection is the input of the activation function. For some activation functions with the element-level activation mechanism, the output of the activation function can be expressed as $a^l = y^l s\left(y^l\right)$, which is the multiplication of y^l and non-linear activation scale $s(y^l)$. For example, the activation scale of ReLUs and sigmoid linear units (SiLUs; Elfwing et al. (2018)) for the i^{th} element are presented as follows:

$$s^{ReLU}\left(y_i^l\right) = \begin{cases} 1, & \text{if } y_i^l \geq 0 \\ 0, & \text{if } y_i^l < 0 \end{cases}, \quad s^{SiLU}\left(y_i^l\right) = sig\left(y_i^l\right) \tag{2}$$

where y_i^l is the i^{th} element of y^l; sig is the sigmoid function, and s^{ReLUs} and s^{SiLUs} are non-linear scale functions of ReLUs and SiLUs, respectively. In such an activation function, the scale

function is the one that carries out the non-linearity output of activation in forward-propagation. In back-propagation, the activation scale function controls the distribution of gradients for training parameters. Therefore, the activation scale $s\left(y_i^l\right)$ represents the effect of activation function on parameters related to the i^{th} element of the l^{th} layer.

2.2 MOTIVATION OF THIS WORK

In the previous section, we discussed the activation scale of element-level activation. Here, when the distribution of activation scale $s\left(y^l\right)$ is imbalanced between samples, it causes imbalanced activation outputs and gradients in forward and back-propagation, respectively. More specifically, the activation scale of l^{th} layer resizes the gradients from the $(l+1)^{th}$ layer and passes it to the $(l-1)^{th}$ layer when a gradient-descent-based optimization algorithm is used. In back-propagation, the gradient passed from the l^{th} layer to the $(l-1)^{th}$ layer under the chain rule is:

$$\frac{\partial L}{\partial a^{l-1}} = \frac{\partial y^l}{\partial x^l}\frac{\partial a^l}{\partial y^l}\frac{\partial L}{\partial x^{l+1}}, \quad \frac{\partial a^l}{\partial y^l} = s\left(y^l\right) + \frac{\partial s\left(y^l\right)}{\partial y^l} \tag{3}$$

where L is the loss of network, while $a^{l-1} = x^l$ and $a^l = x^{l+1}$ as an activation output is the input for the next layer. Thus, we can expect that if a sample has a very small sum of activation scale at a certain training epoch, then the parameters placed behind the l^{th} layer will likely to have small gradients than other samples during the epoch, which causes a bias of training between samples. We refer to this phenomenon as *imbalanced activation*.

2.3 NORMALIZATION IN NETWORKS

Normalization has become an essential technique for DNNs with the great success of BatchNorm (Ioffe & Szegedy, 2015; Lubana et al., 2021). BatchNorm makes the network stable during training in forward and back propagation by re-scaling and re-centering the inputs (Bjorck et al., 2018). In addition, BatchNorm avoids channel-wise collapse, which indicates the problem of linear activation, by balancing channel-wise elements (Daneshmand et al., 2020). However, the use of BatchNorm requires a sufficiently large batch size, and it cannot solve the bias problem among samples (Labatie et al., 2021).

Meanwhile, LayerNorm normalizes elements in the layer-dimension, instead of the batch-dimension to address the batch-dependency of BatchNorm. Specifically, LayerNorm normalizes the elements of a layer with the layer-wise mean and standard deviation defined as follows:

$$n_i^l = \frac{g_i^l}{\sigma^l}\left(y_i^l - \mu^l\right) + b_i^l, \qquad \mu^l = \sum_{i=1}^{H} y_i^l, \qquad \sigma^l = \sqrt{\sum_{i=1}^{H}\left(y_i^l - \mu^l\right)^2} \tag{4}$$

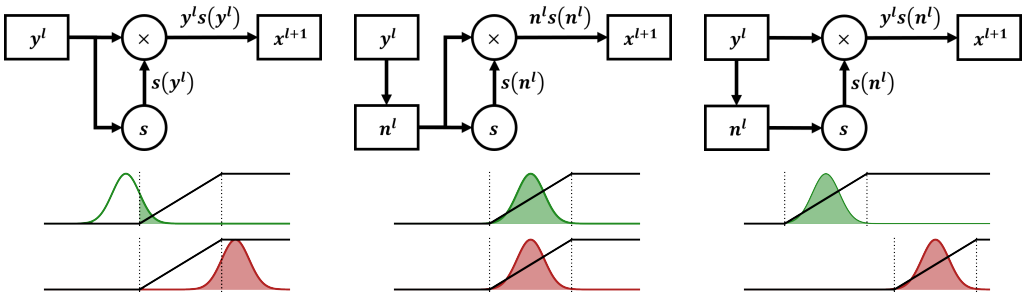
where H is the number of elements in y^l; n_i^l is the i^{th} normalized element, and g_i^l and b_i^l are the learnable parameters of gain and bias of normalization. The distribution of n^l follows $N(b^l, {g^l}^2)$ for all samples, which leads to the balanced distribution of activation scale $s^l(n^l)$. Such a layer-level balancing mechanism can solve the bias problem among samples in training. However, LayerNorm loses all the mean and variance statistics of linear projection; thus, the final outputs of the l^{th} layer across samples become similar (Lubana et al., 2021). To avoid this dilution problem, a layer-level balancing mechanism should be performed without directly re-scaling or re-centering the input of activation.

3 THE PROPOSED LAYERACT MECHANISM AND FUNCTIONS

In this section, we introduce a novel layer-level activation mechanism that addresses the aforementioned limitations of existing methods for activation and normalization, which has the following beneficial properties: i) able to preserve the distribution of inputs y^l and balance the distribution of activation outputs, as well as ii) to maintain the different statistics between samples. Fig.1 and Table.1 show the difference between the element-level activation mechanisms with and without LayerNorm, and our proposed mechanism for layer-wise balanced activation.

Table 1: This table compares the layer-level activation mechanism that we propose in this paper with the conventional element-level activation mechanisms with and without LayerNorm. ✓and X denote if the corresponding method has the specific property or not. BatchNorm is not compared as its objective is not a layer-level balancing; rather, BatchNorm is a complementary with the LayerAct mechanism proposed in this work.

	Maintaining statistics	Layer-wise balanced output of activation	Negative output of activation
Element-level activation without LayerNorm	✓	X	Depends on the activation function
Element-level activation with LayerNorm	X	Depends on the activation function	Depends on the activation function
Proposed mechanism of layer-level activation	✓	✓	Depends on the distribution of inputs



(a) Unnormalized network (b) Network with LayerNorm (c) Network with LayerAct

Figure 1: **Upper**: the mechanisms of the (a) element-level activation without normalization, (b) element-level activation with LayerNorm, and (c) proposed layer-level activation (LayerAct). **Lower**: Non-zero activated distribution of two samples (green and red) with the element-level activation cases versus the layer-level activation case. While the criterion of activation does not change between samples in the element-level activation, the criterion of proposed LayerAct mechanism fits to the layer-wise distribution of the samples.

3.1 LayerAct mechanism

The proposed LayerAct mechanism is defined as the multiplication of the input y_l and the activation scale $s(n^l)$ which has the layer-wise normalized input n_i^l. The forward and backward passes of LayerAct functions are defined as follows:

$$a_i^{l\,LayerAct} = y_i^l s\left(n_i^l\right), \quad n_i^l = \frac{\left(y_i^l - \mu^l\right)}{\sigma^l} \tag{5}$$

$$\frac{\partial a_i^{l\,LayerAct}}{\partial y_i^l} = s\left(n_i^l\right) + \sum_{j=1}^{H} y_j^l \frac{\partial s\left(n_j^l\right)}{\partial n_j^l} \frac{\partial n_j^l}{\partial y_i^l} \tag{6}$$

where μ^l and σ^l are the layer-wise mean and standard deviation of Eq.4, respectively. No matter how different the distributions of linear projection $\left(y^l\right)$ for the samples in the dataset, the scale functions with normalized input vectors n^l fall into a single distribution for all samples. The shape of non-linearity of all samples becomes same at layer-level, while the activation output of a LayerAct function $a^{l\,LayerAct}$ still maintains the different statistics between samples as y_i^l is neither re-scaled nor re-centered (See Fig. 1c). This layer-level activation mechanism to take care of samples equally can reduce the variance of training loss among samples, and therefore making the training process

converge faster than element-level activation mechanisms and achieving higher performance at the end of training. Moreover, LayerAct allows negative activation outputs according to the distribution of inputs, which is known to be useful for training DNNs (Clevert et al., 2015; Qiu et al., 2018).

The normalization techniques, including LayerNorm, use affine transformation function as Eq.4 (Ioffe & Szegedy, 2015; Ba et al., 2016; Wu & He, 2018). However, Xu et al. (2019) argued that the affine transformation parameters (gain and bias) of LayerNorm may cause over-fitting, and do not improve the performance of a network. We use the layer-wise normalized term n^l as the input of activation scale function, and the parameters might act the same as in the case of using LayerNorm. Thus, we will discuss whether the gain and bias would work in LayerAct functions well or not in Section 4 with experimental results.

3.2 Designing LayerAct functions

Proper design of the activation scale functions is very important in designing a neural-network-based learning framework, and the non-linearity of LayerAct functions relies on the form of scale function. Thus, we propose *must have properties* of LayerAct functions as follows.

Continuous. The activation functions should be continuous at every point for stable inference. In contrast to some piece-wise activation functions under element-level activation mechanisms that can have discrete scale functions at 0 but still themselves are continuous, LayerAct functions are discrete if the scale functions are discrete.

Bounded. The scale of activation $s\left(n_i^l\right)$ should be bounded between 0 and 1. If the scale is lower than 0, there is a possibility that the same activation output ${a_i^l}^{LayerAct}$ will derive from more than one activation input which means that the expressiveness is decreased as passing the activation function. If the scale is larger than 1, it is obvious that the gradient (Eq.6) may explode during the back-propagation.

Gradual. Considering Eq.6, the first derivative (i.e. gradient) of activation scale function affects the gradient in back-propagation directly. If the first derivative of the scale function is too large, the gradient would explode with the same reason of the **Bounded** property we discussed above, so the scale function has to be gradual.

With the consideration above, Sigmoid and HardSigmoid functions are suitable for the scale function of LayerAct. Both functions are continuous, bounded exactly between 0 and 1, and the first derivatives are also continuous and stable with the maximum $1/4$ and $1/6$, respectively. We propose the following two LayerAct functions, LA-SiLUs and LA-HardSiLUs, which are layer-level transformed versions of SiLUs and HardSiLUs:

$$ {a_i^l}^{LA\text{-}SiLUs} = y_i^l sig\left(n_i^l\right) \tag{7} $$

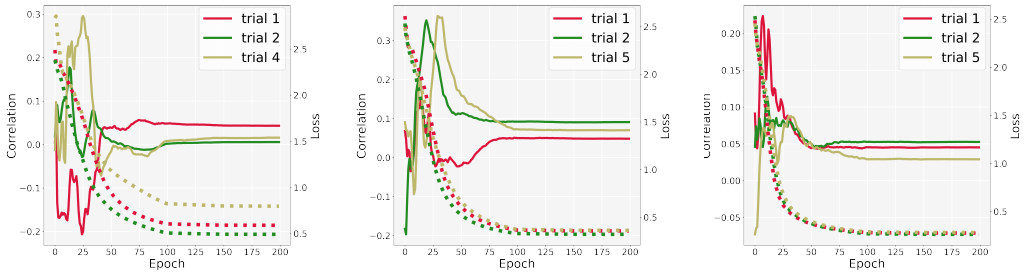
$$ {a_i^l}^{LA\text{-}HardSiLUs} = \begin{cases} y_i^l, & \text{if } n_i^l \geq 3 \\ y_i^l\left(n_i^l/6 + 1/2\right), & \text{if } -3 \leq n_i^l < 3 \\ 0, & \text{if } n_i^l < -3 \end{cases} \tag{8} $$

4 Experiments and results

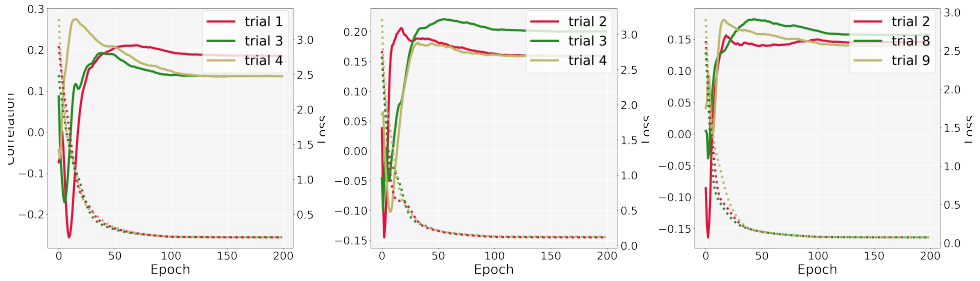
In this section, we show the experimental results on three image datasets, MNIST (gray image, 10 classes, 60k trained and 10k tested), CIFAR10 (color images, 10 classes, 50k training and 10k testing) and CIFAR100 (color images, 100 classes, 50k training and 10k tested). For training, we used 10% of the training dataset as a validation set. Fully-connected MLPs and Convolutional Neural Network (CNN) based models are used for MNIST and the other datasets, respectively. We initialized the weight of the networks following He et al. (2015). For the detailed settings of these experiments, see Appendix A.2.

4.1 Experimental analysis on activation scale

To identify if the imbalance between samples does affect the training, we performed a simple experiment with the MNIST dataset and 4-layer MLPs with different numbers of element units per layer



(a) 20X4 MLP (b) 40X4 MLP (c) 100X4 MLP



(d) 20X4 MLP with BN (e) 40X4 MLP with BN (f) 100X4 MLP with BN

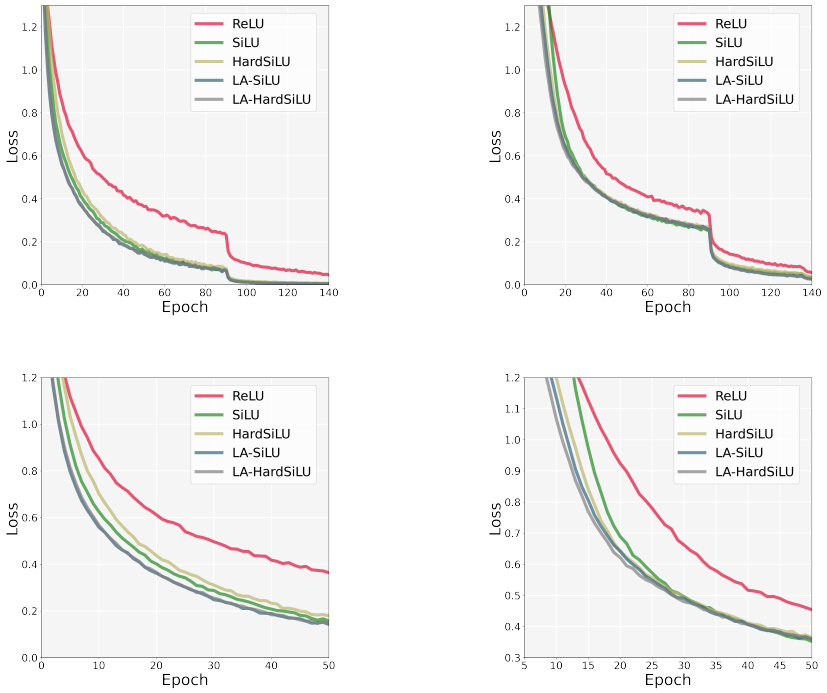
Figure 2: The correlation between the sum of activation scale and the training loss of classification task on the MNIST dataset. AX4 denotes 4 layers with A element units per layer. The figures show three best cases according to the final training loss among 5-runs.

Table 2: Performance comparison of LayerAct functions and other element-level activation functions on CNN for CIFAR10. The test accuracy(%) is shown as best/mean among the 5 runs. Best results are in bold.

Activation	CNN	CNN with BN	CNN with LN
ReLU	87.08 / 86.98	89.72 / 89.21	83.9 / 83.28
SiLU	87.85 / 87.65	90.69 / 90.23	88.07 / 87.71
HardSiLU	87.48 / 87.35	90.41 / 90.02	88.08 / 87.54
LA-SiLU	88.04 / 87.818	90.65 / 90.26	-
LA-HardSiLU	88.26 / 87.92	**90.95** / 90.08	-
Affine LA-SiLU	**88.62 / 88.35**	90.71 / **90.39**	-
Affine LA-HardSiLU	88.16 / 87.89	89.62 / 89.08	-

(20, 40, and 100) for classification. As a correlation analysis can measure the linear relationship between two variables, we analyzed the correlation between the sum of activation scale and training loss while training the network. If there is any positive or negative correlation during training, it means that the imbalanced activation scale has a certain relationship with the training loss.

Fig.2 shows the trend of correlation between the sum of activation scale and the training loss of same networks with different random seeds for weight initialization. There are positive or negative relationships between the sum of activation scale and the loss. In some networks, such a relationship also maintained after training is over, meaning that the network could not overcome the bias in training among samples even at the end of training. The LayerAct mechanism proposed in this paper can ensure a balanced sum of activation scale between samples to avoid the bias in training at every epoch. This way, the new activation mechanism can achieve a low variance of loss among samples, faster convergence, and eventually better performance than element-level activation mechanisms.



(a) unnormalized CNN

(b) CNN with BatchNorm

Figure 3: Plot of training loss of CNN. The best model of the activation function is selected. These figures show that LayerAct functions converge faster than element-level activation functions at the early stage of training.

4.2 Experimental results on CIFAR10 and CIFAR100

Here, we show the experimental results with the CIFAR10 and CIFAR100 dataset. We used the CNN, ResNet20, and ResNet44 (He et al., 2016) models as the benchmarking bases. For the detailed structure of CNN, see Appendix A.1.

4.2.1 Results on CNN with LayerAct and Affine Transformation

We used LA-SiLU and LA-HardSiLU, as LayerAct functions for the CIFAR10 dataset. Following the findings from Xu et al. (2019) we investigate whether the affine transformation parameters (gain and bias) of LayerNorm should be used for LayerAct or not (i.e., to show the capability of the LayerAct mechanism without affine transformation). In Table 2, we denote the LayerAct functions with affine transformation as Affine LayerAct.

The experimental results with CNN are shown in Tables 2. LayerAct functions outperformed the element-level activation functions in both the best case and the mean of 5-runs. Fig.3 also shows that LayerAct functions converge faster than the competitors. Revisiting Eq.2 and Eq.5, the difference between element-level activation functions and LayerAct functions is whether the activation scale is based on the unnormalized or layer-wise normalized input; in case of using LayerAct, the sum of activation scale is balanced among the samples. This experimental result supports the findings we discussed in Section 4.1.

Affine LA-SiLU is superior among the activation functions in unnormalized networks. However, there was no big difference between LayerAct and Affine LayerAct functions on their performance, while the functions without affine transformation showed a better result when used with BatchNorm. Thus, we think that the use of affine transformation is worth when the network does not use BatchNorm, but it would be better to use LayerAct functions without affine transformation for the network

Table 3: Performance comparison of LayerAct functions and other element-level activation functions on ResNet20 for CIFAR10 and CIFAR100. The test accuracy(%) is shown as best/mean among the 5 runs. Best results are in bold.

Activation	CIFAR10	CIFAR100	
		top1	top5
ReLU	91.68 / 91.41	66.25 / 65.98	89.81 / 89.62
SiLU	91.92 / 91.52	**66.68** / 66.09	90.23 / 89.59
LA-SiLU	**91.97 / 91.75**	66.65 / **66.42**	**90.32 / 89.97**
LA-HardSiLU	91.74 / 91.47	66.64 / 66.30	90.04 / 89.84

Table 4: Performance comparison of LayerAct functions and other element-level activation functions on ResNet40 for CIFAR10 and CIFAR100. The test accuracy(%) is shown as best/mean among the 5 runs. Best results are in bold.

Activation	CIFAR10	CIFAR100	
		top1	top5
ReLU	92.59 / 92.38	68.68 / 68.44	90.47 / 90.29
SiLU	92.31 / 91.94	**68.99** / 68.37	90.55 / 90.04
LA-SiLU	**93.00 / 92.67**	68.89 / **68.68**	**90.80 / 90.42**
LA-HardSiLU	91.93 / 91.68	67.67 / 66.80	89.60 / 89.34

Table 5: The standard deviation of loss distribution between samples. The standard deviation is shown as best/mean among the 5 runs. LayerAct functions show lower standard deviation values than element-level activation functions.

Activation	CIFAR10		CIFAR100	
	ResNet20	ResNet44	ResNet20	ResNet44
ReLU	1.51 / 1.53	1.54 / 1.64	2.57 / 2.62	3.18 / 3.24
SiLU	1.57 / 1.61	1.59 / 1.67	2.78 / 2.82	3.32 / 3.34
LA-SiLU	**1.33 / 1.39**	**1.45 / 1.48**	2.43 / 2.45	**2.79 / 2.82**
LA-HardSiLU	1.38 / 1.42	1.61 / 1.64	**2.36 / 2.41**	2.84 / 2.89

with BatchNorm as Affine LayerAct functions have more parameters (i.e., the gain and bias) with a small or no advance in performance.

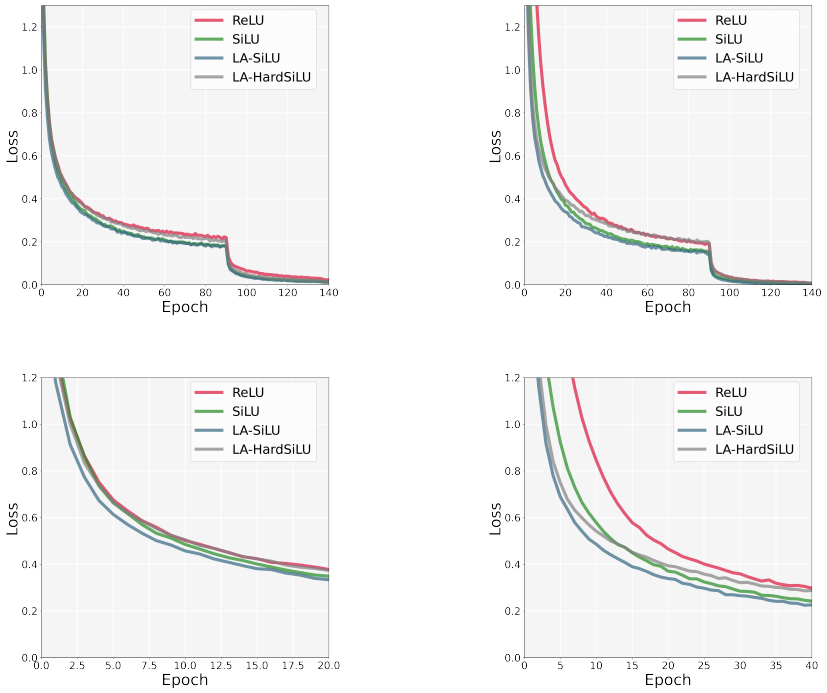
4.2.2 Results on ResNets with LayerAct

For ResNet, we used ReLU and SiLU as the competitors of LayerAct functions. Table 3 and Table 4 show that LA-SiLU outperforms element-level activation functions in most cases of ResNet20 and ResNet44. We had argued that a layer-level activation mechanism can reduce the variance of loss between samples in Section 3.1. In Table 5 shows the standard deviation of the test loss between samples. LayerAct functions have lower standard deviation than others, which support that the layer-level activation mechanism can reduce the bias in training between samples.

5 Concluding remark

In this paper, we originally show the following contributions and implications for improving the core mechanism of learning through multiple layers. First, it is clear that element-level activation functions can cause an imbalanced activation among samples, and the parameter optimization is eventually affected by this imbalance issue. We found a case of the biased activation's effect on training with a simple experiment that shows whether there is a negative or positive relationship between the sum of activation scale and the training loss.

Second, as a solution to address this issue, we developed a novel layer-level activation mechanism, and proposed two LayerAct functions. The LayerAct functions can balance the distribution of activation scale among samples while maintaining the mean and variance statistics of inputs for rich representation. Interestingly, the sensory system of biological neural networks has a layer-wise acti-



(a) ResNet20 (b) ResNet44

Figure 4: Plot of training loss of ResNet and CIFAR10. The best model of the activation function is selected. These figures show that LayerAct functions converge faster than element-level activation functions at the early stage of training.

vation mechanism (Shen et al., 2021) just as LayerAct functions. For example, fruit fly's neurons to detect odors has a consistent pattern; while the distribution of firing rate is exponential for all odors (i.e., the distribution of neurons' outputs has the same pattern between samples), only the mean of the distribution is related to the odor concentration (Stevens, 2016).

Lastly, the proposed LayerAct functions outperformed the element-level activation functions in most cases of image classification tasks. We also showed that LayerAct functions can reduce the variance of loss between samples. Although we introduced two LayerAct functions in this paper, there are possibilities to develop other LayerAct functions. We expect this work will contribute in developing high-performance machine learning frameworks, given the indispensable use and essential role of activation functions in traditional and modern machine learning frameworks, such as MLPs, CNNs, ResNets, and any other models operating multiple layers with many elements.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in Neural Information Processing Systems*, 31, 2018.

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

Hadi Daneshmand, Jonas Kohler, Francis Bach, Thomas Hofmann, and Aurelien Lucchi. Batch normalization provably avoids ranks collapse for randomly initialised deep networks. *Advances in Neural Information Processing Systems*, 33:18387–18398, 2020.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456. PMLR, 2015.

Antoine Labatie, Dominic Masters, Zach Eaton-Rosen, and Carlo Luschi. Proxy-normalizing activations to match batch normalization while removing batch dependence. *Advances in Neural Information Processing Systems*, 34:16990–17006, 2021.

Ekdeep S Lubana, Robert Dick, and Hidenori Tanaka. Beyond batchnorm: towards a unified understanding of normalization in deep learning. *Advances in Neural Information Processing Systems*, 34:4778–4791, 2021.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, pp. 3. Citeseer, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.

Suo Qiu, Xiangmin Xu, and Bolun Cai. Frelu: flexible rectified linear units for improving convolutional neural networks. In *International Conference on Pattern Recognition*, pp. 1223–1228. IEEE, 2018.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Yang Shen, Julia Wang, and Saket Navlakha. A correspondence between normalization strategies in artificial and biological neural networks. *Neural Computation*, 33(12):3179–3203, 2021.

Charles F Stevens. A statistical property of fly odor responses is conserved across odors. *Proceedings of the National Academy of Sciences*, 113(24):6737–6742, 2016.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision*, pp. 3–19, 2018.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

A Appendix

Our source code can be accessed from our anonymous GitHub repository for reproduction[1]. Additional experiments and results will be archived in this repository continuously.

A.1 Structure of CNN

The structure of CNN for Section 4.2.1 is shown in Table 6. Normalization and activation have been used after the convolutional layer.

[1] https://github.com/LayerAct/LayerAct

Table 6: The structure of CNN for the CIFAR10 dataset.

Output size	Network Structure
32X32, 16	Conv [1X1, 16], stride 1, padding 0
32X32 16	Conv [3X3, 16], stride 1, padding 1
32X32 32	Conv [3X3, 32], stride 1, padding 1
16X16 32	Maxpool 2X2, stride 2
16X16 32	Conv [1X1, 32], stride 1, padding 0
16X16 32	Conv [3X3, 32], stride 1, padding 1
16X16 64	Conv [3X3, 64], stride 1, padding 1
8X8 64	Maxpool 2X2, stride 2
8X8 64	Conv [1X1, 64], stride 1, padding 0
8X8 64	Conv [3X3, 64], stride 1, padding 1
8X8 128	Conv [3X3, 128], stride 1, padding 1
4X4 128	Maxpool 2X2, stride 2
4X4 128	Conv [1X1, 128], stride 1, padding 0
4X4 128	Conv [3X3, 128], stride 1, padding 1
4X4 256	Conv [3X3, 256], stride 1, padding 1
2X2 256	Maxpool 2X2, stride 2
2X2 256	Conv [1X1, 256], stride 1, padding 0
2X2 256	Conv [3X3, 256], stride 1, padding 1
2X2 512	Conv [3X3, 512], stride 1, padding 1
1X1 512	Maxpool 2X2, stride 2

Table 7: The devices used for experiments

	Server 1	Server 2	Server 3
CPU	Intel Xeon Gold 5220R@2.20GHz	Intel Xeon Gold 6242@2.80GHz	Intel Xeon Gold 5220@2.20GHz
GPU	NVIDIA TITAN RTX	NVIDIA A100	Tesla V100

A.2 EXPERMENTAL SETTING

A.2.1 EXPERIMENTAL ENVIRONMENT

We used three devices as shown in Table 7. The experiments on MNIST, CNN and CIFAR10 with ResNet44 were performed on device 1. Only the experiment on CIFAR10 with ResNet20 progressed using device 3, and the others were conducted using device 2. The version of Python and the packages were the same in all devices as follows: Python 3.9.12, numpy 1.19.5, pytorch 1.11.0 and torchvision 0.12.0.

A.2.2 EXPERIMENTAL SETUP

The experimental setup is shown in Table 8

Table 8: Setup for experiments

	MNIST	CIFAR10	CIFAR100
Loss function	CrossEntropyLoss(reduction='mean')		
Max iteration	6400	64000	
Batch size	128		
Optimizer	Momentum opimization (0.9)		
Weight decay	0.0001		
Learning rate	0.1 with normalization, otherwise 0.01		
Learining rate Scheduler	0.1 at 3200 and 4800	0.1 at 32000 and 48000	