Neural Computing and Applications
https://doi.org/10.1007/s00521-024-10935-3

ORIGINAL ARTICLE

Constructing a smoothed Leaky ReLU using a linear combination of the smoothed ReLU and identity function

Meng Zhu[1] · Weidong Min[1,2,3] · Jiahao Li[1] · Mengxue Liu[1] · Ziyang Deng[1] · Yao Zhang[1]

Received: 2 March 2024 / Accepted: 13 December 2024

Abstract
Convolutional neural networks (CNNs) have made tremendous progress in solving many challenging problems. Good activation functions can improve the performance of CNNs. The existing activation functions exhibit inconsistent performance gains across different training settings, models, datasets and tasks. To solve this problem, we propose a general smoothed approximation for the maximum function $\max(x_i, \alpha x_i)$ using the linear combination of the smoothed rectified linear unit and the identity function. And we use exponential moving average to training the negative slope in this smoothed approximation. To validate the effectiveness of our approach, we also present a smoothed approximation case named leaky power function linear unit (LPFLU) to compare with the current state-of-the-art activation functions. Experimental results demonstrate that our LPFLU outperforms the existing state-of-the-art activation functions in improved robustness across different training settings, models, datasets and tasks.

Keywords Convolutional neural networks · Activation functions · Smoothed approximation · Maximum function · Linear combination

1 Introduction

In recent years, deep convolutional neural networks (CNNs) have made significant strides and had a profound impact on real-world applications, e.g., object detection [1], semantic segmentation [2], vehicle re-identification [3] and human pose estimation [4]. Weight initialization [5], loss functions [6], overfitting [7], optimization [8, 9] and etc. are crucial components of networks.

The activation functions play a very crucial role in CNNs by learning the abstract features through nonlinear transformations [10]. The widely used activation function is the Rectified Linear Unit (ReLU) [11–13], defined as $f(x) = \max(x, 0)$. The use of ReLUs was a breakthrough in enabling the fully supervised training of state-of-the-art deep CNNs. Deep CNNs using ReLUs are easily optimized due to the smoothed flow of gradients when the input to ReLU is positive. The simplicity and effectiveness of ReLU have made it the default choice for activation function in the deep learning community.

However, ReLU suffers from the vanishing gradient problem in the negative part and is not differentiable at the origin. The vanishing gradient problem refers to the situation where the gradient of the objective function with respect to the parameters becomes close to zero. When training a network using optimization techniques such as stochastic gradient descent (SGD) [14, 15], this means that the parameters hardly get updated in the presence of vanishing gradients. Consequently, training becomes extremely challenging or even impossible in the presence of vanishing gradients. To overcome the shortcomings of ReLU, various variants of ReLU have been proposed [16–20]. The Gaussian Error Linear Unit (GELU) gains popularity in the deep learning community due to its efficacy in natural language processing tasks. GELU has been used in BERT [21], GPT-3 [22], Instruct GPT [23]

✉ Weidong Min
minweidong@ncu.edu.cn

[1] School of Mathematics and Computer Science, Nanchang University, Nanchang 330031, China

[2] Institute of Metaverse, Nanchang University, Nanchang 330031, China

[3] Jiangxi Provincial Key Laboratory of Virtual Reality, Nanchang 330031, China

architectures and etc.. Swish shares similar properties like the GELU, and it shows some good improvement of ReLU.

Moreover, the performance gains of the existing activation functions are not robust across different training settings, models, datasets and tasks. In theory, the trainable activation functions can enhance robustness. However, in practice, the trainable activation functions often fail to achieve consistent performance and may even perform worse than the fixed activation functions due to the influence of different training settings. The parameters of the trainable activation functions are often scalar parameters, which are not on par with tensor parameters and may require different learning rates. Additionally, the trainable activation functions also increase the computational complexity of backpropagation compared against the fixed activation functions. Therefore, is it possible to find an activation function design that does not require parameters to be trainable while still being robust to different training settings, models, datasets and tasks?

To solve this problem, we propose a universal smoothed approximation for the maximum function $\max(x_i, \alpha x_i)$ via the linear combination of the smoothed ReLU and the identity function. Additionally, we utilize exponential moving average (EMA) to train the negative slope within this smoothed approximation. To assess the efficacy of our methodology, we also present a specific smoothed approximation instance called Leaky Power Function Linear Unit (LPFLU) to compare with the existing state-of-the-art activation functions.

The main contributions of this paper can be summarized as follows.

1. We propose a general smoothed approximation formula for the maximum function $\max(x_i, \alpha x_i)$ using the linear combination of the smoothed ReLU and the identity function. This general smoothed approximation is simple but efficient.
2. We use EMA to train the negative slope in this smoothed approximation.

The rest of this paper is organized as follows. Related work is analyzed in Sect. 2. Details of our proposed approach are given in Sect. 3. Performance of LPFLU and some state-of-the-art activation functions in different settings is analyzed in Sect. 4. Experimental results and analysis are reported in Sect. 5. The discussion is made in Sect. 6. The conclusion is drawn in Sect. 7.

2 Related work

A linear function can be seen as a simple activation function where the output is a linear transformation of the input, which is essentially a constant. Linear activation functions do not introduce nonlinearity to the neural network. However, neural networks require the introduction of nonlinearity. Otherwise, despite having multiple layers, the output of the neural network would be a linear function of the input. Additionally, in practice, data is often not linearly separable. Therefore, nonlinear layers help to project the data in a nonlinear manner into a feature space, enabling them to be used with different objective functions.

To introduce nonlinearity into neural networks, the Sigmoid and Tanh activation functions are early used, defined as Eqs. (1) and (2), respectively,

$$\sigma(x_i) = \frac{1}{1+\exp(-x_i)}, \tag{1}$$

$$\mathrm{Tanh}(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)}. \tag{2}$$

The motivation for using the Sigmoid and Tanh activation functions is from biologically inspired neurons. However, both the Sigmoid and Tanh functions suffer from saturation at high and low input values, leading to the vanishing gradient problem.

To alleviate the vanishing gradient problem, Jarrett et al. proposed the ReLU [13], defined as Eq. (3),

$$\mathrm{ReLU}(x_i) = \max(x_i, 0) = \begin{cases} x_i, & \text{if } x_i > 0, \\ 0, & \text{if } x_i \leq 0. \end{cases} \tag{3}$$

ReLU is an identity function in the positive part and zero in the negative part. As a result, ReLU has a gradient of 1 in the positive part and a gradient of 0 in the negative part. Due to its simplicity and improved performance, ReLU has become the state-of-the-art activation function in the neural networks. ReLU addresses the vanishing gradient problem that existed in the Sigmoid and Tanh activation functions for the positive part. However, ReLU still suffers from the zero gradient problem in the negative part. Additionally, ReLU is not differentiable at the origin.

To overcome these limitations, various variants of ReLU have been studied, e.g., Leaky ReLU (LReLU) [16], Parametric ReLU (PReLU) [24], Randomized LReLU (RReLU) [25], Parametric Tan Hyperbolic Linear Unit [26], Exponential Linear Unit (ELU) [17], Scaled ELU [27], Parametric ELU [28], Multiple Parametric ELU [29], Fast ELU [30] and Elastic ELU [31]. These variants aim to improve upon the drawbacks of ReLU while maintaining its advantages. These activation functions have demonstrated their superiority in convergence time. However, the performance improvements of these activation functions are often inconsistent across different models and datasets.

Through the combination of ReLU, dropout [32], and zoneout [33], Hendrycks et al. presented the GELU [18], defined as Eq. (4),

$$\mathrm{GELU}(x_i) = x_i\Phi(x_i) = x_i\mathrm{P}(X \leq x_i). \tag{4}$$

Here $\Phi(x_i)$ is the cumulative distribution function of the standard normal distribution. As x_i decreases, the probability of "dropping" the input increases, resulting in a stochastic but input-dependent transformation applied to x_i. This mechanism, known as self-gating, shields the input while retaining uncertainty and dependence on input values. GELU has several prominent advantages, including nonsaturation in the positive part, sparsity in the negative part, smoothness, and nonmonotonicity. These characteristics make GELU perform exceptionally well in local response normalization. Additionally, GELU can be seen as a smoothed approximation of ReLU. Inspired by GELU, a series of nonmonotonic activation functions have been proposed, e.g., Swish [19], SiLU [34], Hardswish [35], Rectified Exponential Unit (REU) [36], Mish [37], Logish [38], Power Function Linear Unit (PFLU) [39], Fast PFLU (FPFLU) [39], Smish [40] and Gish [41]. However, under the condition of limited computing resources, these activation functions have a higher computational cost compared to ReLU.

Recently, the Padé Activation Unit (PAU) [20] was proposed, which approximates the LReLU function using a rational polynomial of a given order, defined as Eq. (5),

$$\mathrm{PAU}(x_i) = \frac{P(x_i)}{Q(x_i)}. \tag{5}$$

Here $P(x_i)$ and $Q(x_i)$ are two polynomials of order m and n, respectively. While PAU improves network performance in image classification problems, surpassing ReLU, its variants and Swish, it suffers from the drawback of having a number of trainable parameters. This significantly increases network complexity and computational costs. To address this drawback, several alternatives have been proposed, including Smooth Activation Unit (SAU) [42], Smooth Maximum Unit (SMU) [43], and SMU-1 [43].

In summary, the existing activation functions have made certain progress in addressing issues such as vanishing gradients, dead neurons, and computational costs, but each activation function has its own limitations. Therefore, there is a need to design a novel activation function that is both simple and effective, and can adapt to different training settings, models, datasets, and tasks. This is the motivation behind the proposal of LPFLU in this paper.

3 Proposed approach

In this section, we propose a novel general approximation formula for LReLU [16] using the linear combination of the smoothed ReLU and the identity function. To enhance the adaptability of this approximation formula, we utilize EMA technique to train the negative slope.

3.1 Smoothed approximation of the Leaky ReLU

LReLU is a widely used activation function that introduces a fixed negative slope α when the input is negative, to avoid the zero gradient problem of the ReLU function in the negative input region. LReLU is defined as Eq. (6),

$$\begin{aligned}\mathrm{LReLU}(x_i) &= \max(x_i, \alpha x_i)\\ &= \begin{cases} x_i, & \text{if } x_i > 0,\\ \alpha x_i, & \text{if } x_i \leq 0.\end{cases}\end{aligned} \tag{6}$$

Here x_i denotes the i-th element of the input tensor $\boldsymbol{X}$, and α denotes the negative slope. However, LReLU has discontinuity at the zero point, which may adversely affect the training efficiency and stability of neural networks.

To address this issue, we use interval mapping technique and smoothed approximation of ReLU to derive the general smoothed approximations of LReLU. First, we rewrite Eq. (6) using interval mapping techniques, thereby deriving Eq. (7),

$$\begin{aligned} f(x_i) &= x_i(\theta(x_i)(b-a)+a)\\ &= (b-a)x_i\theta(x_i) + ax_i\\ &= (b-a)\mathrm{ReLU}(x_i) + ax_i,\\ &\text{s.t. } b > a \geq 0. \end{aligned} \tag{7}$$

Here $\theta(x_i)$ denotes the step function, b denotes the positive slope, and a denotes the negative slope. The interval mapping technique is specifically denotes $\theta(x_i)(b-a)+a$ in Eq. (7), to map the interval (0, 1) to the interval (a, b). When $a = \alpha$ and $b = 1$, Eq. (7) is equivalent to Eq. (6). This states that LReLU can be expressed as the sum of the weighted (with weight $1-\alpha$) ReLU function and the weighted (with weight α) input variable.

Further, when the ReLU in Eq. (7) is replaced with its smoothed version, then Eq. (7) naturally represents the smoothed version of LReLU. Then, we seek the smoothed version of ReLU. There are multiple known approximations to $\mathrm{ReLU}(x_i)$, e.g., $\mathrm{GELU}(x_i)$, $\mathrm{Swish}(x_i)$ and $\mathrm{PFLU}(x_i)$. In this paper, we focus on one specific approximation to $\mathrm{ReLU}(x_i)$, namely $\mathrm{PFLU}(x_i)$. By means of replacing $\mathrm{ReLU}(x_i)$ with $\mathrm{PFLU}(x_i)$ and setting $b = 1$ in Eq. (7), we obtain a smoothed approximation case for LReLU, which is expressed as Eq. (8),

$$f(x_i) = (1-a)\mathrm{PFLU}(x_i) + ax_i. \tag{8}$$

The function in Eq. (8) is named to as LPFLU.

The first-order derivative of Eq. (8) with respect to the input variable x_i is given by Eq. (9),

$$\frac{\partial}{\partial x_i}\mathrm{LPFLU}(x_i) = (1-a)\frac{\partial}{\partial x_i}\mathrm{PFLU}(x_i) + a. \tag{9}$$

Figure 1 displays comparison between LReLU and LPFLU when the negative slope is 0.1. As shown in Fig. 1, our LPFLU provides a smoothed approximation of LReLU from below. Figure 2 plots LPFLUs and their first-order derivatives for various negative slope values.

3.2 Training the negative slope using exponential moving average

In LPFLU, the training of the negative slope a is mainly done in three different forms. The first form is a fixed constant, the second form is a trainable parameter, and the third form involves using a random number during training and a fixed value during testing. In Sect. 1, we have analyzed that trainable scalar parameters cannot be treated on par with trainable tensor parameters, i.e., trainable scalar parameters require more fine-grained learning rate settings. Therefore, we utilize EMA to train the negative slope parameter a.

In the train phase, a_t (the subscript t denotes training iterations) is a random number sampled from a uniform distribution $\mathcal{U}(l,u)$, as formulated in Eq. (10),

$$a_t \sim \mathcal{U}(l,u), \text{s.t. } l < u \text{ and } l,u \in [0,1). \tag{10}$$

In the test phase, we take EMA of all the a_t in training to get a deterministic result, as formulated in Eq. (11),

$$\overline{a}_t = m\overline{a}_{t-1} + (1-m)a_t, \text{s.t. } m \in (0,1). \tag{11}$$

Here $\overline{a}_0 = (l+u)/2$.

In the Residual/Dense/Shuffle connection blocks, the EMA algorithm is activated (as shown in Fig. 3), while in the feedforward connection blocks, a fixed negative slope value is used (as shown in Fig. 4), in order to avoid introducing too much randomness into the network model.

4 Analysis of typical CNN components

In this section, our aim is to showcase the superior performance of the LPFLU against the existing state-of-the-art activation functions. We will demonstrate the image classification performance of various components within the CNN, including batch size, learning rate, network depth, initializer, optimizer, data augmentation and noise, using the LPFLU and existing state-of-the-art activation functions, which include ReLU, Mish, PFLU, SMU and Gish. For all experiments, we considered SMU as trainable activation functions. We initialized μ at 1.0 for SMU. All trainable parameters were updated using the backpropagation [44] algorithm. We set the hyperparameter α at 0.2 for SMU. And we set $l = 0$, $u = 0.02$ for LPFLU. The value of m for LPFLU was set to be equal to the value of momentum in the batch normalization (BN) layer [45]. For this purpose, we used EfficientNet [46] as the underlying architecture for batch size, learning rate, network depth and initializer. We also used RegNet [47] as the underlying architecture for optimizer, data augmentation and noise. All experiments were trained and evaluated on the CIFAR-100 [48] dataset. The EfficientNet and RegNet models only retained the last three spatial downsamplings.

The CIFAR-100 dataset consists of colored natural images with dimensions of 32×32 pixels and includes 100 categories, each containing 600 images. During the training process on CIFAR-100, we employed standard data augmentation techniques such as random color space transformation and random horizontal flipping.

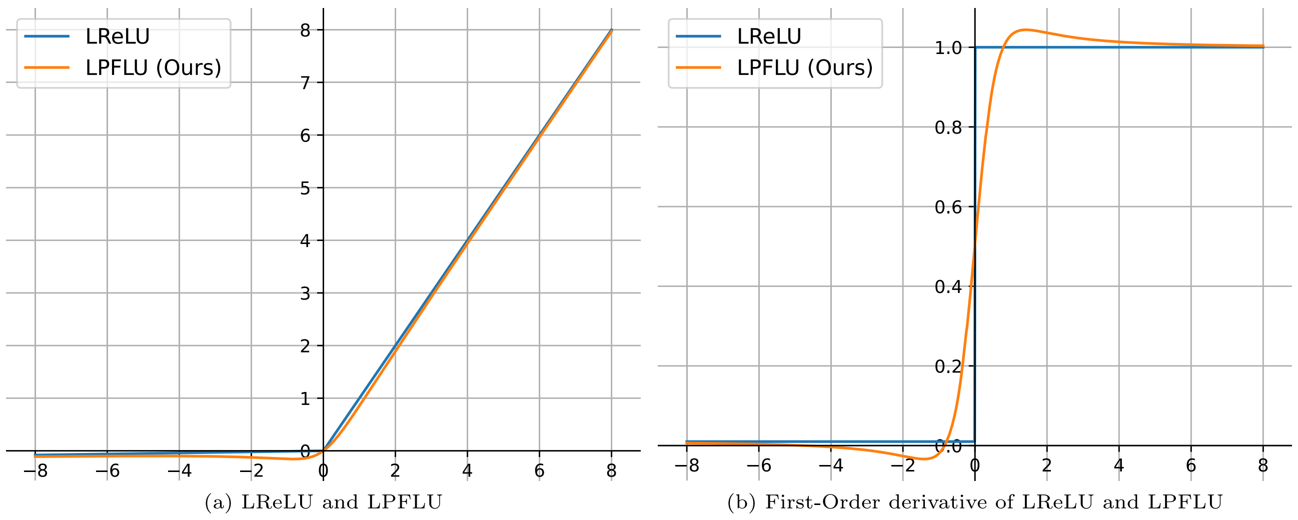
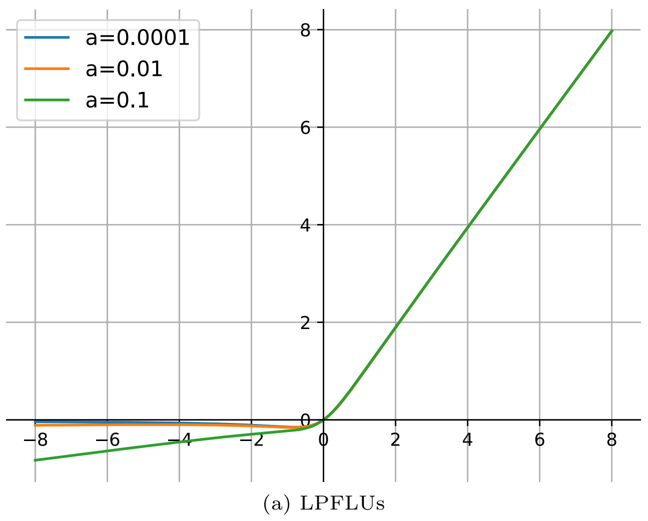


Fig. 1 Comparison between LReLU and LPFLU when the negative slope is 0.01. Best viewed in color



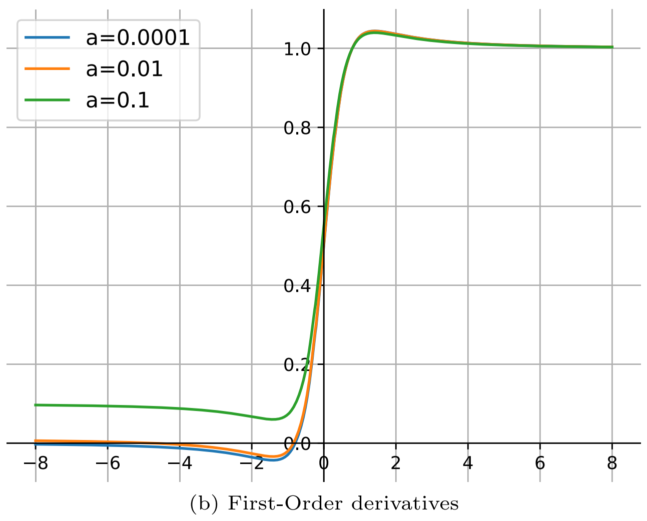


Fig. 2 LPFLUs and their first-order derivatives for various negative slope values. Best viewed in color

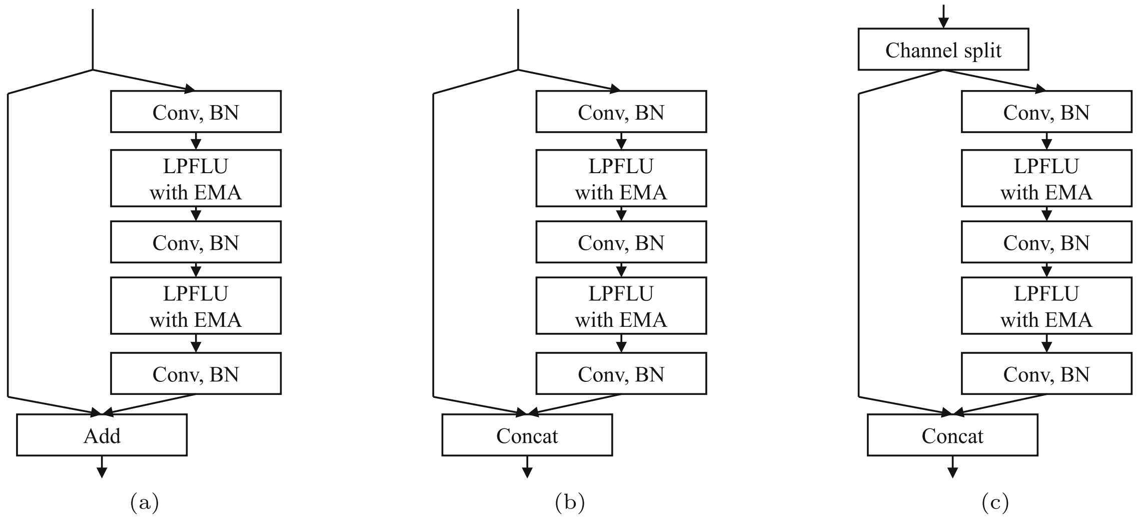


Fig. 3 Activating the EMA algorithm of LPFLU in the Residual/Dense/Shuffle connection blocks

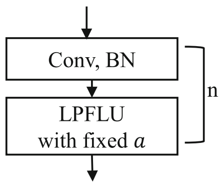


Fig. 4 Fixing negative slope value of LPFLU in the feedforward connection blocks

The evaluation metrics included the top-1 error rate and top-5 error rate. Top-1 error rate refers to the situation where the category with the highest probability predicted by the model does not match the actual category. In classification tasks, the model assigns a probability to each category, with the category having the highest probability being considered the model's predicted category. If this category with the highest probability does not align with the true category, it is counted as a top-1 error rate. This metric is one of the important indicators for measuring the accuracy of model predictions, especially in application scenarios that require high accuracy. Top-5 error rate refers to the situation where the true category is not among the top five highest probability categories predicted by the model. This metric is usually used to measure the performance of the model under more lenient prediction conditions, that is, as long as the true category is among the top five predicted by the model, the model is considered to have made a correct prediction.

All experimental tasks were conducted on a single computer with the following specifications: CPU Intel(R) Core(TM) i7-11700K, GPU Nvidia RTX A5000, RAM 32GB, Python 3.11.5, PyTorch [49] 2.1.2, Cudatoolkit 12.1 and Cudnn 8.9.

4.1 Analysis of various batch size

Batch size refers to the number of samples selected for a training set, and it plays a crucial role in model optimization. Typically, a larger batch size leads to a more accurate descent direction and reduced training fluctuations. However, there comes a point where further increasing the batch size has little impact on the descent direction. It is common practice to set the batch size as a power of 2, which facilitates binary calculations and enhances graphics processing unit computing and storage efficiency.

In this study, we conducted experiments using different batch sizes, specifically 16, 32, 64, and 128. The underlying architecture was set to EfficientNet-B0. Parameters of networks were optimized using the Adam [9]. All models were trained from scratch for a total of 80 epochs. The initial learning rate was set to $2e-3$. For the first 390 steps, the learning rate scaling factor linearly increased from 0 to 1. From step 390 to 21450, the learning rate scaling factor linearly decayed from 1 to 0.01. For step values greater than or equal to 21450, the learning rate scaling factor remained constant at 0.01. For each convolution layer and fully connected layer, the weights were initialized using Xavier uniform distribution [5], and if biases were used, they were initialized with zeros. In each BN layer [45], if the scaling parameter (γ) was used, it was initialized with ones, and if the shift parameter ($\boldsymbol{\beta}$) was used, it was initialized with zeros. The corresponding experimental results are shown in Fig. 5.

As shown in Fig. 5a, Mish, PFLU, and LPFLU consistently achieved lower top-1 error rates than ReLU, SMU, and Gish across different batch sizes. As shown in Fig. 5b, ReLU achieved the lowest top-5 error rate across different batch sizes, and both ReLU and LPFLU consistently achieved lower top-5 error rates than Mish, PFLU, SMU, and Gish.

4.2 Analysis of various initial learning rate

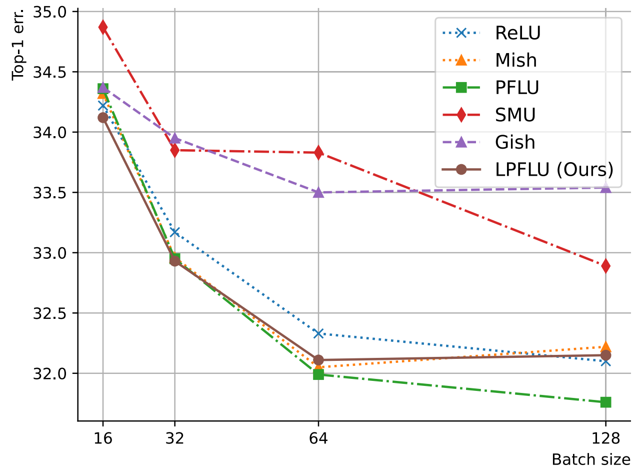
The learning rate represents the information accumulation speed in the neural network training. Here we selected initial learning rates of 0.002, 0.02 and 0.2, respectively. The underlying architecture was set to EfficientNet-B0. Parameters of networks were optimized using Stochastic Gradient Descent (SGD) [8, 14, 15] with momentum of 0.9. The batch size was set to 128. For each convolution layer and fully connected layer, the weights were initialized using Xavier uniform distribution. The corresponding experimental results are shown in Fig. 6.

As shown in Fig. 6, when the learning rate was set to 2, Gish and LPFLU achieved lower error rates compared to ReLU, Mish, PFLU, and SMU. When the learning rate was set to 0.002, 0.02, and 0.2, these activation functions exhibited similar performance.

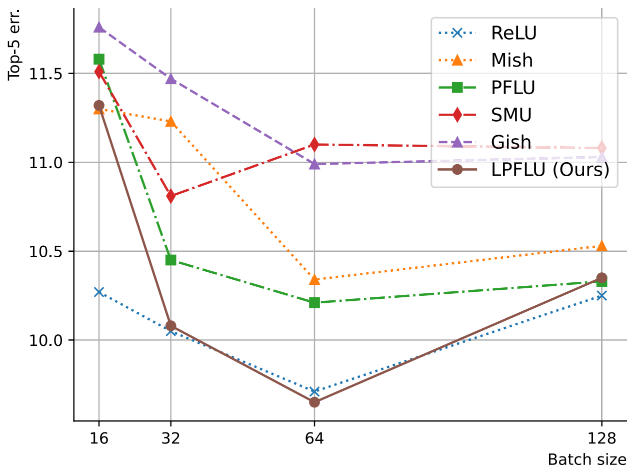
4.3 Analysis of various network depth

The network depth plays a crucial role in various aspects such as the expressive power of the network, feature learning, and model performance. Here we selected EfficientNet-B0, EfficientNet-B2 and EfficientNet-B4, respectively. Parameters of networks were optimized using Adam. The batch size was set to 128. The initial learning rate was set to $2e-3$. For each convolution layer and fully connected layer, the weights were initialized using Xavier uniform distribution. The corresponding experimental results are shown in Fig. 7.

As shown Fig. 7a, across different depths of the EfficientNet network, Mish and LPFLU achieved lower top-1

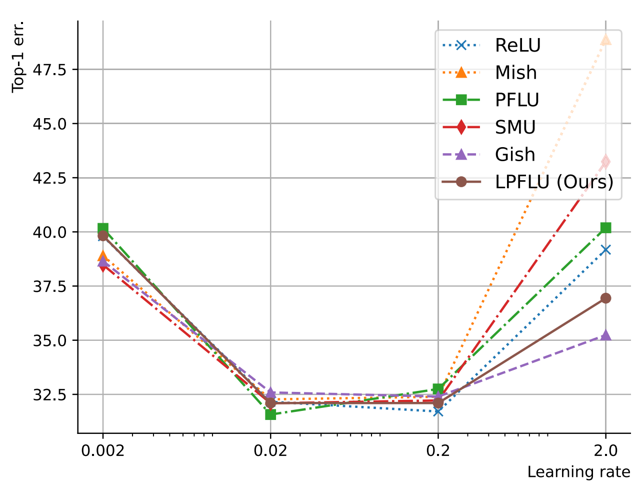


(a) Top-1 error rate *vs.* batch size

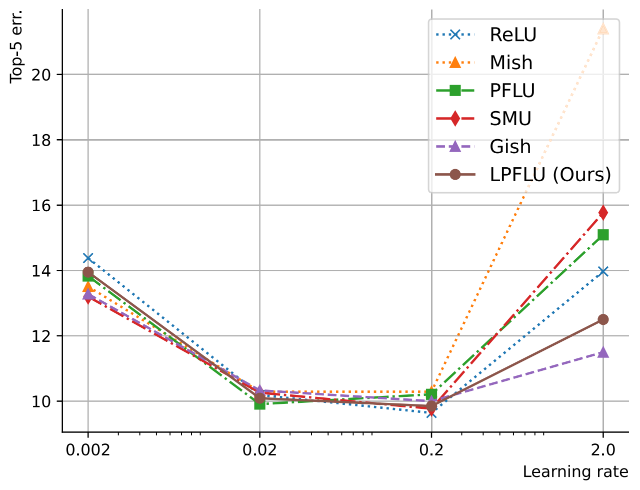


(b) Top-5 error rate *vs.* batch size

Fig. 5 Test error rate vs. batch size for ReLU, Mish, PFLU, SMU, Gish and LPFLU. Best viewed in color

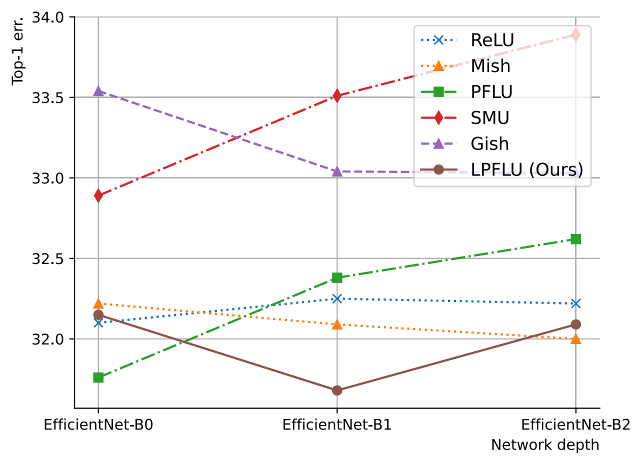


(a) Top-1 error rate *vs.* learning rate

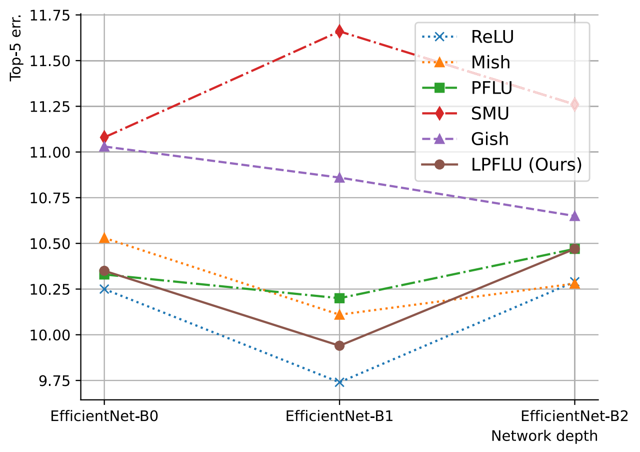


(b) Top-5 error rate *vs.* learning rate

Fig. 6 Test error rate vs. learning rate for ReLU, Mish, PFLU, SMU, Gish and LPFLU. Best viewed in color



(a) Top-1 error rate *vs.* network depth



(b) Top-5 error rate *vs.* network depth

Fig. 7 Test error rate vs. network depth for ReLU, Mish, PFLU, SMU, Gish and LPFLU. Best viewed in color

error rates compared to ReLU, PFLU, SMU, and Gish. As shown in Fig. 7b, ReLU achieved the lowest top-5 error rate across different depths of the EfficientNet network.

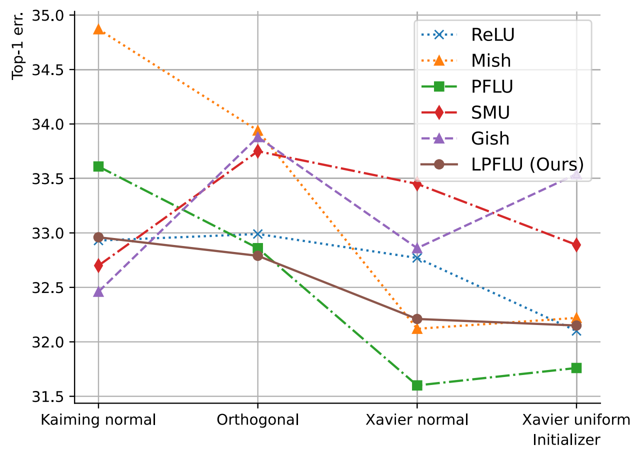
4.4 Analysis of various initializer

Initialization plays a crucial role in neural networks as it determines the initial state of the network during training. Proper initialization methods can help the network learn and converge more effectively, while improper initialization can lead to training difficulties or performance degradation. Here we selected Kaiming normal [24], Orthogonal [50], Xavier normal [5] and Xavier uniform [5], respectively. The underlying architecture was set to EfficientNet-B0. Parameters of networks were optimized using Adam. The batch size was set to 128. The initial learning rate was set to $2e-3$. The corresponding experimental results are shown in Fig. 8.

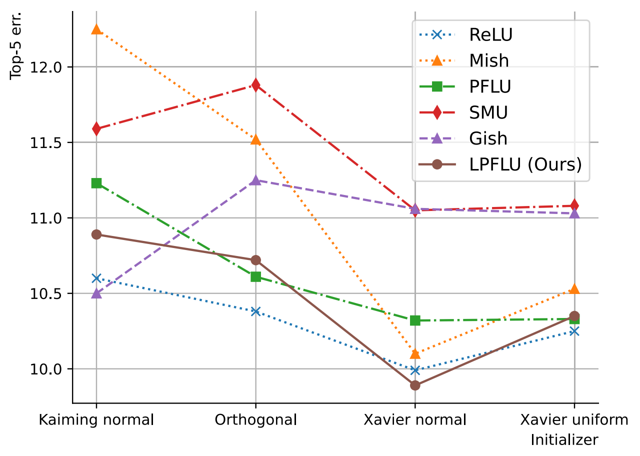
As shown in Fig. 8a, across different initializations, PFLU and LPFLU achieved lower top-1 error rates compared to ReLU, Mish, SMU, and Gish. As shown in Fig. 8b, ReLU achieved more robust top-5 error rate across different initializations.

4.5 Analysis of various optimizer

The objective of deep learning encompasses the iterative refinement of network parameters to enable them to execute a spectrum of nonlinear transformations on input data, thereby aligning with the desired output. This endeavor is fundamentally an exploration for the most effective solution. Consequently, the identification of an appropriate optimizer and the subsequent parameter adjustments are



(a) Top-1 error rate *vs.* initializer



(b) Top-5 error rate *vs.* initializer

Fig. 8 Test error rate vs. initializer for ReLU, Mish, PFLU, SMU, Gish and LPFLU. Best viewed in color

pivotal to advancing deep learning studies. Here we selected Adam and SGD with momentum of 0.9. The underlying architecture was set to RegNetY-800MF. The batch size was set to 128. If the optimizer was Adam, the initial learning rate was set to $2e-3$. If the optimizer was SGD, the initial learning rate was set to 0.2. For each convolution layer and fully connected layer, the weights were initialized using Xavier uniform distribution. The corresponding experimental results are listed in Table 1.

As shown in Table 1, LPFLU achieved the lowest top-1 error rate of 31.26% and the lowest top-5 error rate of 10.21% when parameters of networks were optimized using Adam. LPFLU performed a little worse when parameters of networks were optimized using SGD.

4.6 Analysis of whether using data augmentation or not

Data augmentation enhances the diversity of training data, which improves the model's generalization ability and thus enhances the model's performance. Here we selected using data augmentation and not using data augmentation. The underlying architecture was set to RegNetY-800MF. Parameters of networks were optimized using Adam. The batch size was set to 128. The initial learning rate was set to $2e-3$. The corresponding experimental results are listed in Table 2.

As shown in Table 2, LPFLU achieved the lowest top-1 error rate of 31.26% and the lowest top-5 error rate of 10.21% when data augmentation was used. LPFLU achieved the second-lowest top-1 error rate of 36.29% when data augmentation was not used.

4.7 Analysis of whether using noise in the data or not

Adding random noise to the training data simulates the variations found in real-world data, thereby increasing the model's robustness and helping it learn more generalized features. Here we selected using Gaussian noise ($Z \sim \mathcal{N}(0, 0.1)$) and not using Gaussian noise. The underlying architecture was set to RegNetY-800MF. Parameters of networks were optimized using Adam. The batch size was set to 128. The initial learning rate was set to $2e-3$. The corresponding experimental results are listed in Table 3.

Table 1 Test error rate vs. optimizer for ReLU, Mish, PFLU, SMU, Gish and LPFLU

Activation	Adam [9]		SGD [8]	
	Top-1 err. (%) ↓	Top-5 err. (%) ↓	Top-1 err. (%) ↓	Top-5 err. (%) ↓
ReLU [13]	32.69	10.61	35.71	12.17
Mish [37]	31.65	10.26	33.36	10.87
PFLU [39]	31.98	10.23	33.30	10.69
SMU [43]	31.71	10.33	**32.02**	**9.42**
Gish [41]	32.76	10.99	33.13	10.36
LPFLU (Ours)	**31.26**	**10.21**	33.40	11.05

The bold denotes the best result

Table 2 Test error rate vs. data augmentation for ReLU, Mish, PFLU, SMU, Gish and LPFLU

Activation	Using data augmentation		Not using data augmentation	
	Top-1 err. (%) ↓	Top-5 err. (%) ↓	Top-1 err. (%) ↓	Top-5 err. (%) ↓
ReLU [13]	32.69	10.61	38.48	14.13
Mish [37]	31.65	10.26	36.46	**12.50**
PFLU [39]	31.98	10.23	37.45	13.08
SMU [43]	31.71	10.33	36.49	12.55
Gish [41]	32.76	10.99	**36.11**	12.52
LPFLU (Ours)	**31.26**	**10.21**	36.29	12.94

The bold denotes the best result

Table 3 Test error rate vs. noise for ReLU, Mish, PFLU, SMU, Gish and LPFLU

Activation	Not using noise		Using noise	
	Top-1 err. (%) ↓	Top-5 err. (%) ↓	Top-1 err. (%) ↓	Top-5 err. (%) ↓
ReLU [13]	32.69	10.61	35.02	**11.38**
Mish [37]	31.65	10.26	36.07	12.55
PFLU [39]	31.98	10.23	35.28	11.80
SMU [43]	31.71	10.33	35.65	12.60
Gish [41]	32.76	10.99	37.63	13.90
LPFLU (Ours)	**31.26**	**10.21**	**34.73**	11.63

The bold denotes the best result

As shown in Table 3, LPFLU consistently achieved the lowest top-1 error rate regardless of whether noise was used. When noise was not used, LPFLU achieved the lowest top-5 error rate of 10.21%. When noise was used, LPFLU achieved the second-lowest top-5 error rate of 11.63%.

4.8 Comprehensive analysis of the above all typical CNN components

Sections 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7 analyzed the impact of batch size, initial learning rate, network depth, initializers, optimizers, data augmentation, and noise on the performance of activation functions. Although these analyses provided the independent effects of each component, in practice, these components often act together on the model. Therefore, this section will attempt to integrate the existing results for a comprehensive analysis, focusing on the performance of the LPFLU activation function under integrated conditions.

By means of comparing the performance of various activation functions under different component settings (as shown in Figs. 5, 6, 7 and 8 and Tables 1, 2 and 3), two common trends can be observed.

Superiority of LPFLU in most components Whether under different batch sizes, initial learning rates, network depths, initializer settings, or in terms of optimizer choices, data augmentation, and noise handling, LPFLU generally exhibited lower top-1 and top-5 error rates. This indicates that LPFLU has strong adaptability and stability across various component combinations.

Interactions between components Although the impact of each component on model performance may vary when analyzed individually, their interactions under comprehensive conditions can be more complex, e.g., a larger batch size usually helps the model to converge, but an excessively high learning rate may lead to unstable training; data augmentation and noise handling can improve the model's generalization ability, but the specific effects also depend on the settings of other components.

Based on the two common trends, the following analyses can be drawn.

Stability and adaptability LPFLU is more likely to exhibit stable performance across different combinations of components.

Great performance In most comprehensive conditions, LPFLU is more likely to have lower top-1 and top-5 error rates compared to other activation functions in image classification tasks.

5 Experimental results

In Sect. 4, apart from the component analysis of LPFLU, we will conduct image classification and object detection experiments using LPFLU and other activation functions in different CNN architectures. The CIFAR [48] and PASCAL VOC [51] datasets will be used to evaluate the performance of LPFLU.

5.1 Implementation details

For evaluating and comparing various activation functions on CIFAR [48] classification, we utilized two commonly used CNN architectures as backbone models: EfficientNet-B0 [46] and RegNetY-800MF [47]. All backbone network models only retained the last three spatial downsamplings. The network parameters were optimized using the Adam optimizer [9] with a mini-batch size of 128. All models were trained from scratch for 80 epochs. The initial learning rate was set to $2e-3$. If the step value was less than 390, the learning rate scaling factor linearly increased from 0 to 1. For step values between 390 and 21450, the learning rate scaling factor linearly decayed from 1 to 0.01. If the step value was greater than or equal to 21450, the learning rate scaling factor remained constant at 0.01.

We also evaluated and compared various activation functions on PASCAL VOC [51] using the SSDLite architecture [35]. The network was trained from scratch for 80 epochs using the Adam optimizer [9] with a mini-batch size of 100. The initial learning rate was set to $4e-3$. If the step value was less than 171, the learning rate scaling factor linearly increased from 0 to 1. For step values between 171 and 9405, the learning rate scaling factor linearly decayed from 1 to 0.01. If the step value was greater than or equal to 9405, the learning rate scaling factor remained constant at 0.01.

We further evaluated and compared various activation functions on CityScapes [52] using U-net [53]. The network was trained from scratch for 80 epochs using the Adam optimizer [9] with a mini-batch size of 16. The initial learning rate was set to $2e-3$. If the step value was less than 185, the learning rate scaling factor linearly increased from 0 to 1. For step values between 185 and 10175, the learning rate scaling factor linearly decayed from 1 to 0.01. If the step value was greater than or equal to 10175, the learning rate scaling factor remained constant at 0.01.

In each convolution layer and fully connected layer, the weights were initialized using Xavier uniform distribution [5], and if biases were used, they were initialized with zeros. In each BN layer [45], the scaling parameter (γ) was initialized with ones, and if the shift parameter (β) was used, it was initialized with zeros.

All experimental tasks were conducted on a single computer with the following specifications: CPU Intel(R) Core(TM) i7-11700K, GPU Nvidia RTX A5000, RAM 32GB, Python 3.11.5, PyTorch [49] 2.1.2, Cudatoolkit 12.1 and Cudnn 8.9. All time benchmark tasks were conducted on another single computer with the following specifications: CPU Intel(R) Core(TM) i9-10900K, GPU Nvidia Quadro RTX 4000, RAM 64GB, Python 3.12.4, PyTorch 2.3.1, Cudatoolkit 12.1 and Cudnn 8.9.

5.2 Image classification performance on CIFAR-100 using different network architectures

In this subsection, we will show the test error rate results of LFPLU with ReLU, Mish, PFLU, SMU and Gish on CIFAR-100. Table 4 shows the test error rate of various popular network architectures including EfficientNet-B0 and RegNetY-800MF.

As shown in Table 4, it could be seen that in EfficientNet-B0, LPFLU achieved the second-lowest top-1 error rate and the third lowest top-5 error rate. In RegNetY-800MF, LPFLU achieved the best results in both top-1 and top-5 errs.. In RegNetY-800MF, LPFLU reduced the top-1 error rate by 1.11% and the top-5 error rate by 0.84% compared to ReLU. In RegNetY-800MF, LPFLU reduced the top-1 error rate by 1.07% and the top-5 error rate by 0.49% compared to Mish.

Figure 9 depicts the training curve of epoch vs. top-1 error rate. As shown in this figure, it could be seen that all models were well trained, and no overfitting phenomenon has occurred.

5.3 Object detection performance on PASCAL VOC

Subsequently, we proceeded to conduct experiments on the PASCAL VOC 2007/2012 dataset [51]. This dataset consists of 20 classes. The PASCAL VOC 2012trainval dataset comprises 11530 images with 27450 ROI annotated objects. We trained all object detectors using the VOC 2012trainval dataset and evaluated the performance on the VOC 2007test dataset for comparison. During the training process on PASCAL VOC, we applied standard data augmentation techniques, including random color space transformation and random horizontal flipping. The input image resolution for the detectors was set to 320×320 pixels. We utilized the metrics provided by the COCO API as the evaluation criteria. The results are presented in Table 5, where LPFLU achieved the highest performance in terms of mAP@50, mAP, and mAP@75, respectively.

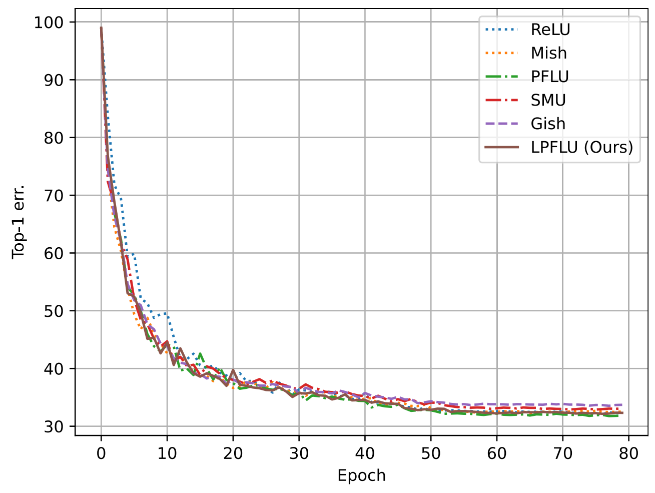
5.4 Semantic segmentation performance on CityScapes

We further presented additional findings on the well-known CityScapes [52] dataset. This dataset encompasses a wide range of urban street scenes from over 50 distinct cities captured at different times of the year, and includes ground

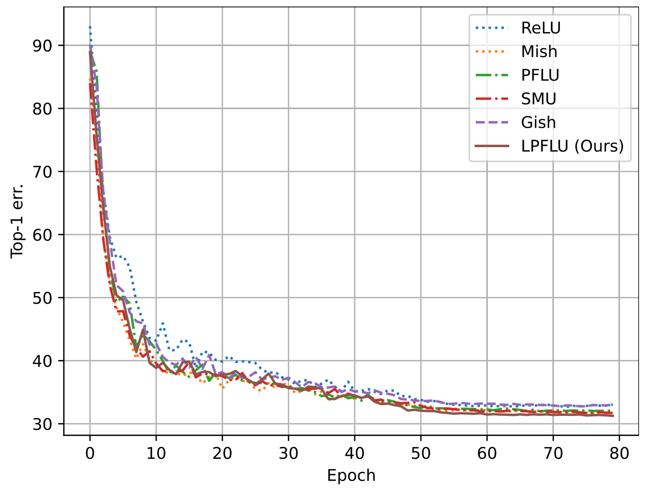
Table 4 Comparison results between different activation functions on CIFAR-100

Activation	EfficientNet-B0 [46]		RegNetY-800MF [47]	
	Top-1 err. (%) ↓	Top-5 err. (%) ↓	Top-1 err. (%) ↓	Top-5 err. (%) ↓
ReLU [13]	32.10	**10.25**	32.69	10.61
Mish [37]	32.22	10.53	31.65	10.26
PFLU [39]	**31.76**	10.33	31.98	10.23
SMU [43]	32.89	11.08	31.71	10.33
Gish [41]	33.54	11.03	32.76	10.99
LPFLU (Ours)	32.15	10.35	**31.26**	**10.21**

The bold denotes the best result



(a) If network architecture being EfficientNet-B0



(b) If network architecture being RegNetY-800MF

Fig. 9 Epoch vs. top-1 error rate. Best viewed in color

Table 5 Comparison results between different activation functions on PASCAL VOC 2007test. The bold denotes the best result

Activation	mAP@50 (%) ↑	mAP (%) ↑	mAP@75 (%) ↑
ReLU [13]	36.4	19.4	18.6
Mish [37]	36.4	19.9	19.4
PFLU [39]	37.1	20.0	19.3
SMU [43]	37.1	20.0	19.4
Gish [41]	36.7	19.8	19.0
LPFLU (Ours)	**37.8**	**20.5**	**19.8**

The bold denotes the best result

Table 6 Comparison results between different activation functions on CityScapes

Activation	mIOU (%) ↑	Pixel acc. (%) ↑
ReLU [13]	48.45	91.45
Mish [37]	49.88	91.64
PFLU [39]	49.76	91.63
SMU [43]	48.20	91.37
Gish [41]	48.81	91.58
LPFLU (Ours)	**50.23**	**91.77**

The bold denotes the best result

truth data for semantic segmentation and instance-level segmentation. The segmentation labels cover more than 30 classes, we trained our segmentation models using only 20 of them. Our evaluation metrics were the mean Intersection-Over-Union (mIOU) and the pixel accuracy, and our results are shown in Table 6, where LPFLU achieved the highest performance in terms of mIOU and pixel acc., respectively.

5.5 Computational complexity

Table 7 lists the computational complexity of various activation functions. The inference speed was measured as follows: The input size was $\boldsymbol{X} \in \mathbb{R}^{32\times1024\times56\times56}$ (32 for batch size, 1024 for the number of channels, 56 for height, and 56 for width), and the measurements were repeated 300 times.

Table 7 Inference speed of various activation functions

Method	CPU (FPS) ↑	GPU (FPS) ↑
ReLU [13]	**22.27**	**435.09**
Mish [37]	0.90	19.13
PFLU [39]	3.65	55.86
SMU [43]	0.78	15.93
Gish [41]	3.41	69.01
LPFLU (Ours)	2.20	39.23

The bold denotes the best result

As seen in Table 7, the inference speed of LPFLU is faster than Mish and SMU, but slower than ReLU, PFLU, and Gish.

6 Discussion

General analysis of experimental results The comprehensive experimental results presented in Sect. 4 demonstrate the superior performance of the proposed LPFLU compared to the existing state-of-the-art activation functions, including ReLU, Mish, PFLU, SMU, and Gish. These findings are robust across various CNN components, training configurations, models, and datasets.

Stability and adaptability of LPFLU The analysis in Sects. 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7 highlights the strong stability and adaptability of LPFLU. Across different batch sizes, initial learning rates, network depths, initializers, optimizers, data augmentation, strategies, and the presence or absence of noise, LPFLU consistently exhibited stable performance. This indicates that models utilizing LPFLU are more likely to maintain good performance when there are variations in component configurations, enhancing their robustness in practical applications.

Superior performance in image classification As shown in Table 4, LPFLU achieved the best results in both top-1 and top-5 error rates on the CIFAR-100 dataset using the RegNetY-800MF architecture. Specifically, LPFLU reduced the top-1 error rate by 1.11% and the top-5 error rate by 0.84% compared to ReLU, and by 1.07% and 0.49% compared to Mish, respectively. These improvements demonstrate the superior performance of LPFLU in image classification tasks.

Effectiveness in object detection and semantic segmentation Beyond image classification, LPFLU also demonstrated effectiveness in object detection and semantic segmentation tasks. As shown in Table 5, LPFLU achieved the highest performance in terms of mAP@50, mAP, and mAP@75 on the PASCAL VOC dataset. Similarly, in Table 6, LPFLU achieved the highest performance in terms of mIOU and pixel accuracy on the CityScapes dataset. These results further confirm the broad applicability and effectiveness of LPFLU across different computer vision tasks.

Computational complexity While LPFLU exhibits superior performance in terms of accuracy and robustness, its computational complexity should also be considered. As shown in Table 7, the inference speed of LPFLU is faster than Mish and SMU but slower than ReLU, PFLU, and Gish. This trade-off between performance and complexity needs to be taken into account when selecting an activation function for a particular application.

Future directions To enhance the inference speed of LPFLU, the following areas in future work will be explored. (1) Optimizing algorithm implementation: By means of conducting performance analysis of the code, identifying and eliminating bottlenecks, ensure efficient execution of the algorithm. (2) Low-precision computation: Consider adopting low-precision computation techniques, such as using half-precision floating-point numbers (FP16) or even lower precision data types for calculations. Low-precision computation can significantly speed up inference without markedly reducing model performance. (3) Hardware acceleration: Explore the use of modern hardware acceleration technologies, such as GPUs and TPUs, to accelerate LPFLU. By means of optimizing the parallelism of computational tasks and leveraging specific hardware acceleration instruction sets, further enhance inference speed.

Future directions Despite LPFLU's strong robustness under noisy conditions, its performance can be further improve in future work using the following methods. (1) Enhance noise adaptability: Explore the introduction of more types of noise during training (e.g., salt-and-pepper noise, impulse noise) to train LPFLU's adaptability to more complex noise patterns. Using this method, further improvements in LPFLU's performance under noisy conditions can be expected. (2) Optimize parameter training strategies: Continue to research and optimize training strategies for the negative slope parameter, such as exploring the use of more advanced optimization algorithms or adaptive learning rate adjustment strategies. This helps LPFLU better learn parameter settings suitable for noisy data during training. (3) Combine with other regularization techniques: Consider using LPFLU in conjunction with other regularization techniques (e.g., Dropout, L2 regularization) to enhance the model's generalization ability under noisy conditions. These regularization techniques can further improve LPFLU's performance on noisy data by means of reducing the risk of model overfitting. (4) In-depth analysis of noise impact: Conduct a more in-depth analysis of how noise affects the model training process to understand the internal mechanisms behind LPFLU's good performance under noisy conditions. This will help to propose more targeted improvement strategies to further enhance LPFLU's performance.

7 Conclusion

In this paper, we present a universal smoothed approximation formula for the maximum function $\max(x_i, \alpha x_i)$ using the linear combination of the smoothed ReLU and the identity function. This general smoothed approximation is both straightforward and effective. Additionally, we employ the EMA to train the negative slope within this smoothed approximation. The experimental results provide evidence that our LPFLU surpasses the current state-of-the-art activation functions in enhanced robustness across various training configurations, models, datasets, and tasks.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 62076117, Recipient W. Min) and the Jiangxi Provincial Key Laboratory of Virtual Reality (Grant No. 2024SSY03151, Recipient W. Min).

Author contributions M. Zhu contributed to the manuscript through conceptualization, methodology development, data curation, software implementation, investigation, formal analysis, and writing of the original draft. W. Min played a role in the manuscript through conceptualization, funding acquisition, resource management, supervision, and writing of the review and editing. J. Li contributed to the manuscript through investigation. M. Liu contributed to the manuscript through software development. Z. Deng contributed to the manuscript through visualization. Y. Zhang contributed to the manuscript through validation.

Data availability The datasets during the current study are open source and available in http://www.cs.toronto.edu/~kriz/cifar.html, http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html and https://www.cityscapes-dataset.com repositories, respectively.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Menezes AG, de Moura G, Alves C et al (2023) Continual object detection: a review of definitions, strategies, and challenges. Neural Netw 161:476–493. https://doi.org/10.1016/j.neunet.2023.01.041
2. Shi M, Shen J, Yi Q et al (2023) Lmffnet: a well-balanced lightweight network for fast and accurate semantic segmentation. IEEE Trans Neural Netw Learn Syst 34(6):3205–3219. https://doi.org/10.1109/TNNLS.2022.3176493
3. Wang Q, Zhong Y, Min W et al (2023) Dual similarity pre-training and domain difference encouragement learning for vehicle re-identification in the wild. Pattern Recogn 139:1–11. https://doi.org/10.1016/j.patcog.2023.109513
4. Gai D, Feng R, Min W et al (2023) Spatiotemporal learning transformer for video-based human pose estimation. IEEE Trans Circuits Syst Video Technol. https://doi.org/10.1109/TCSVT.2023.3269666
5. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, vol 9, pp 249–256
6. Wang Q, Ma Y, Zhao K, Tian Y (2022) A comprehensive survey of loss functions in machine learning. Ann Data Sci 9(2):187–212. https://doi.org/10.1007/s40745-020-00253-5
7. Cawley GC, Talbot NL (2007) Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters. J Mach Learn Res 8(4):841–861
8. Sutskever I, Martens J, Dahl G et al (2013) On the importance of initialization and momentum in deep learning. In: Proceedings of the International Conference on Machine Learning, pp 1139–1147
9. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of the International Conference on Machine Learning
10. Duch W, Jankowski N (1999) Survey of neural transfer functions. Neural Comput Surv 2(1):163–212
11. Hahnloser RHR, Sarpeshkar R, Mahowald MA et al (2000) Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 405(6789):947–951
12. Jarrett K, Kavukcuoglu K, Ranzato M, LeCun Y (2009) What is the best multi-stage architecture for object recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 2146–2153. https://doi.org/10.1109/ICCV.2009.5459469
13. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the International Conference on Machine Learning, pp 807–814
14. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 23(3):400–407
15. Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. Ann Math Stat 23(3):462–466
16. Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the International Conference on Machine Learning
17. Clevert DA, Unterthiner T, Hochreiter S (2016) Fast and accurate deep network learning by exponential linear units (elus). In: Proceedings of the International Conference on Learning Representations
18. Hendrycks D, Gimpel K (2016) Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. arXiv:1606.08415
19. Ramachandran P, Zoph B, Le QV (2018) Searching for activation functions. In: Proceedings of the International Conference on Learning Representations
20. Molina A, Schramowski P, Kersting K (2020) Padé activation units: End-to-end learning of flexible activation functions in deep networks. In: Proceedings of the International Conference on Learning Representations
21. Devlin J, Chang MW, Lee K, Toutanova K (2019) Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp 4171–4186
22. Brown T, Mann B, Ryder N et al (2020) Language models are few-shot learners. In: Proceedings of the International Conference on Neural Information Processing Systems, vol 33, pp 1877–1901
23. Ouyang L, Wu J, Jiang X et al (2022) Training language models to follow instructions with human feedback. In: Proceedings of the International Conference on Neural Information Processing Systems, vol 35, pp 27730–27744

24. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 1026–1034
25. Xu B, Wang N, Chen T, Li M (2015) Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv:1505.00853
26. Duggal R, Gupta A (2017) P-telu: Parametric tan hyperbolic linear unit activation for deep neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 974–978
27. Klambauer G, Unterthiner T, Mayr A, Hochreiter S (2017) Self-normalizing neural networks. In: Proceedings of the International Conference on Neural Information Processing Systems, vol 30
28. Trottier L, Giguere P, Chaib-draa B (2017) Parametric exponential linear unit for deep convolutional neural networks. In: Proceedings of the IEEE International Conference on Machine Learning and Applications, pp 207–214. https://doi.org/10.1109/ICMLA.2017.00038
29. Li Y, Fan C, Li Y et al (2018) Improving deep neural network with multiple parametric exponential linear units. Neurocomputing 301:11–24. https://doi.org/10.1016/j.neucom.2018.01.084
30. Qiumei Z, Dan T, Fenghua W (2019) Improved convolutional neural network based on fast exponentially linear unit activation function. IEEE Access 7:151359–151367. https://doi.org/10.1109/ACCESS.2019.2948112
31. Kim D, Kim J, Kim J (2020) Elastic exponential linear units for convolutional neural networks. Neurocomputing 406:253–266. https://doi.org/10.1016/j.neucom.2020.03.051
32. Hinton GE, Srivastava N, Krizhevsky A et al (2012) Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. arXiv:1207.0580
33. Krueger D, Maharaj T, Kramar J et al (2017) Zoneout: Regularizing rnns by randomly preserving hidden activations. In: Proceedings of the International Conference on Neural Information Processing Systems
34. Elfwing S, Uchibe E, Doya K (2018) Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural Netw 107:3–11. https://doi.org/10.1016/j.neunet.2017.12.012
35. Howard A, Sandler M, Chu G et al (2019) Searching for mobilenetv3. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 1314–1324
36. Ying Y, Su J, Shan P et al (2019) Rectified exponential units for convolutional neural networks. IEEE Access 7:101633–101640. https://doi.org/10.1109/ACCESS.2019.2928442
37. Misra D (2019) Mish: A Self Regularized Non-Monotonic Neural Activation Function. arXiv:1908.08681
38. Zhu H, Zeng H, Liu J, Zhang X (2021) Logish: a new nonlinear nonmonotonic activation function for convolutional neural network. Neurocomputing 458:490–499. https://doi.org/10.1016/j.neucom.2021.06.067
39. Zhu M, Min W, Wang Q et al (2021) PFLU and FPFLU: Two novel non-monotonic activation functions in convolutional neural networks. Neurocomputing 429:110–117. https://doi.org/10.1016/j.neucom.2020.11.068
40. Wang X, Ren H, Wang A (2022) Smish: a novel activation function for deep learning methods. Electronics 11(4):550. https://doi.org/10.3390/electronics11040540
41. Kaytan M, Aydilek IB, Yeroglu C (2023) Gish: a novel activation function for image classification. Neural Comput Appl. https://doi.org/10.1007/s00521-023-09035-5
42. Biswas K, Kumar S, Banerjee S, Kumar Pandey A (2022) Sau: Smooth activation function using convolution with approximate identities. In: Proceedings of the European Conference on Computer Vision, pp 313–329. https://doi.org/10.1007/978-3-031-19803-8_19
43. Biswas K, Kumar S, Banerjee S, Kumar Pandey A (2022) Smooth maximum unit: Smooth activation function for deep networks using smoothing maximum technique. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 784–793
44. LeCun Y, Boser B, Denker JS et al (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551. https://doi.org/10.1162/neco.1989.1.4.541
45. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine Learning, vol 37, pp 448–456
46. Tan M, Le Q (2019) Efficientnet: Rethinking model scaling for convolutional neural networks. In: Proceedings of the International Conference on Machine Learning, vol 97, pp 6105–6114
47. Radosavovic I, Kosaraju RP, Girshick R et al (2020) Designing network design spaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 10428–10436
48. Krizhevsky A, Hinton GE (2009) Learning Multiple Layers of Features from Tiny Images. https://citeseerx.ist.psu.edu
49. Paszke A, Gross S, Massa F et al (2019) Pytorch: An imperative style, high-performance deep learning library. In: Proceedings of the International Conference on Neural Information Processing Systems, pp 8026–8037
50. Saxe AM, McClelland JL, Ganguli S (2013) Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks. arXiv:1312.6120
51. Everingham M, Winn J (2011) The pascal visual object classes challenge 2012 (voc2012) development kit. Pattern Analysis, Statistical Modelling and Computational Learning 8
52. Cordts M, Omran M, Ramos S et al (2016) The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 3213–3223
53. Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: Proceedings of the Medical Image Computing and Computer-Assisted Intervention, vol 9351, pp 234–241. https://doi.org/10.1007/978-3-319-24574-4_28