LAU: A NOVEL TWO-PARAMETER LEARNABLE LOGMOID ACTIVATION UNIT

Anonymous authors
Paper under double-blind review

ABSTRACT

The activation function in deep neural networks has a major impact on the performance of the training stage. In this work, we proposed a novel learnable Logmoid Activation Unit (LAU), $f(x) = x\ln(1 + \alpha\text{sigmoid}(\beta x))$, with two free parameters α and β that can be optimized via back-propagation algorithm. We design quasi-interpolation type neural network operators with Logmoid-1 in a given feed-forward neural network for approximating any continuous function in closed spaces. This provides a theoretical basis for the excellent empirical performance of LAUs in experimental simulations. For instance, compared with ReLUs the proposed LAUs improves Top-1 classification accuracy on ImageNet-200 by 7% respectively in ShuffleNet-V2, on CIFAR-10 by 6% respectively in EfficientNet-B0, and on CIFAR-100 by 5% respectively in MobileNet-V2. Our simulations show that end-to-end learning deep neural networks with learnable Logmoids can increase the predictive performance.

1 INTRODUCTION

In recent years, deep learning has achieved unprecedented success in numerous classification problems (LeCun et al., 2015; Sriperumbudur et al., 2010). It is due to the representation and learning of the unknown structures in machine learning tasks by using deep neural networks (DNNs), which are often modeled by some unknown function mappings or unknown probability distributions. An important component of DNNs is a nonlinear function called activation function. It plays a core role both in training the network model and predicting classification performance. If an inappropriate activation function is selected, it will stop the training procedure, due to the small residuals after multi-layers convolutions in CNNs (Krizhevsky et al., 2017). Therefore, There is a growing interest in building new activation functions to improve learning capabilities (Nwankpa et al., 2018) and the expressive power of DNNs for approximating functions (Liang & Srikant, 2016; Shen et al., 2019).

Currently, activation function is mainly divided into linear activation function, nonlinear monotonic activation function and non-linear non-monotonic activation function. In the early developing stage, the linear activation function was widely used. It mainly consists of a linear threshold function and a simple linear function. The typical linear activation functions are Step Function (Klein et al., 2009), Sign Function (Huang & Babri, 1998) and Identity function. Step function mimics the biological neurons property. However, it failed to be applied to neural network because of its discontinuous derivative. Sign function is favorable because of its antisymmetry, but it lacks biological motivation. The most typical and simplest linear function is the Identity function, where the output of the node is equal to the input, but it has no classification ability. Therefore, the linear activation function are not widely used for practical application with CNN.

The famous nonlinear monotonic activation function include Sigmoid, Tanh and ReLU family. In the early nonlinear activation function literature, Sigmoid (Hassell et al., 1977) and Tanh (Kalman & Kwasny, 1992) activation functions are widely used. However, their small derivatives may cause the gradient disappear (He & Xu, 2010; Klambauer et al., 2017). In 2011, the Softplus function provided greater results, which further boost the development of ReLU function. Compared with Sigmoid and Tanh function, ReLU shows educed saturation, sparsity, efficiency, and ease of use and thus can alleviate the vanishing gradient problem. However, its negative value directly returns to zero, the neural network may lose some valid information. This has intrigued improvements for exploring new activation functions such as Leaky ReLU (Maas et al., 2013), ELU (Clevert et al.,

2015), RReLU (Xu et al., 2015). Leaky Rectified Linear Unit (Leaky ReLU) (Maas et al., 2013) with $f(x) = \max\{x, 0.01x\}$ introduces an non-zero gradient in the negative part. Parametric Rectified Linear Unit(PReLU) (He et al., 2015) with $f(x) = \max\{x, \alpha x\}$ gives a similar solution by introducing a parameter in the slope of the negative part. Exponential Linear Unit (ELU) (Clevert et al., 2015) with $f(x) = \max\{0, x\} + \min\{0, e^x - 1\}$ has a certain soft saturation in negative part. Both of them have reduced the presence of silent neurons and made negative gradient learning possible.

Unfortunately, how to design the optimal activation function is still an open question. Recently, nonlinear non-monotonic activation function has achieved good performance in neural network. Swish (Ramachandran et al., 2017), which is bounded below and not bounded above,is proved to be a more robust activation function beyond ReLU. Later, researchers construct new activation functions by combining different units. For example, Mish (Misra, 2019) and TanhExp (Liu & Di, 2020) which are also bounded below and not bounded above were proposed. They are smooth at almost all points on the curve, which allows better information to penetrate the neural network to obtain better accuracy and generalization performance.

Biswas et al. (2020) and Zhou et al. (2020) show that activation functions with parameters may have better performance beyond parameter-free functions. In 2021, Zhu et al. (2021) also proposed Logish activation function and its variant, which exhibits better performance and Top-1 accuracy. However, different parameters will result in performance changes.This will not follow a universal solution for general tasks. Recently, trainable activation function has received increasing attention. Molina et al. (2019) proposed Pad$\acute{e}$ Activation Unit (PAU), in which some learnable hyper-parameter(s) are updated during training by back-propagation algorithm (LeCun et al., 1989) to reduce the impact on the first selection of a fixed activation function.

With the above discussion, we can conclude that the new activation function needs to satisfy the following characteristics: non-linearity, differentiable almost everywhere, nearly linear mapping, not too complicated calculation, etc. In addition, we hope that the newly proposed activation function can have fewer trainable parameters to find the activation function needed at each network layer automatically. Therefore, this paper proposed a new family of activation function called Logmoid family to solve the above challenges. The expression is

$$f(x; \alpha, \beta) = x \ln(1 + \alpha \text{Sigmoid}(\beta x)) \tag{1}$$

where the logarithmic operation is used to reduce the range of the Sigmoid(x) function value and make sure smooth curve, and the coefficients α and β can be constants or trainable parameters. The main contributions of this paper can be summarized as follows:

1. A new family of activation function called Logmoid family is proposed.
2. Two quasi-interpolation type neural network operators with sub-family Logmoid-1 activation function are constructed to approximate univariate continuous functions.
3. The trainable Logmoid Activation Unit has better performance than other activation functions in image classification experiments in CIFAR, MNIST, Imagenet datasets.

The rest of this paper is organized as follows: Section 2 gives a brief description of some relative activation functions. In Section 3, we propose a family of activation function called Logmoid family and illustrate some properties of its variant. Section 4 analyzes the performances of the subfamily Lgomoid-1. In Section 5, we introduce a learnable activation unit called LAUs, and we do image classification task with LAUs and some different activation functions in different networks to verify the effectiveness of LAUs while Section 6 concludes the results.

2 RELATED WORKS

In this section, we will introduce the relative activation functions, Swish, TanhSoft and PAU.

2.1 SWISH

Swish (Ramachandran et al., 2017) was found by utilizing reinforcement learning based on exhaustive search, which is defined by

$$f(x) = x\text{Sigmoid}(x) \tag{2}$$

and the derivative formula of Swish can be presented as

$$\frac{df(x)}{dx} = \frac{e^x(1+e^x+x)}{(1+e^x)^2}. \tag{3}$$

The curve and derivative curve of Swish are illustrated in Fig.1. Ramachandran et al. (2017) used the same value for gating to simplify the gating mechanism, it is known as self-gating. The advantage of self-gating is that it only requires one simple scalar input, which makes it easy to replace ReLU. In addition, Swish activation function has some outstanding characteristics such as unsaturation, smooth and non–monotonicity. In Swish, the derivative of a continuous order is infinite, which is an advantage over ReLU. The continuous order of ReLU is 0, indicating that it is not continuously differentiable, which can potentially present problems in gradient-based optimization.

2.2 TANHSOFT

TanhSoft family is indexed by four hyper-parameters, which is created by combining a hyper-parametric versions of the tanh and the Softplus activation functions. TanhSoft family can be expressed as

$$f(x;\alpha,\beta,\gamma,\delta) = \text{tanh}(\alpha x + \beta e^{\gamma x})\text{ln}(\delta + e^x). \tag{4}$$

Any function in this family can be used as a activation function for hyper-parameter values. Bidwas performed an organized search of activation functions within the TanhSoft family by varying the values of hyper-parameters with different neural networks on CIFAR10 dataset and considered two sub-families called TanhSoft-1 and TanhSoft-2, which are given as

$$\begin{aligned} &TanhSoft-1: \\ &f(x;\alpha,0,\gamma,1) &= \text{tanh}(\alpha x)\text{Softplus}(x) \\ &&= \text{tanh}(\alpha x)\text{ln}(1+e^x). \end{aligned} \tag{5}$$

$$TanhSoft-2: f(x;0,\beta,\gamma,0) = x\text{tanh}(\beta e^{\gamma x}). \tag{6}$$

Moreover, Biswas et al. (2020) reported that TanhSoft-1 (α=0.87) and TanhSoft-2 (β=0.6, $\gamma = 1$) showed higher Top-1 performance than other activation functions.

2.3 PAU

Padé Activation Units (PAU) is a learnable activation function based on the Padé approximation, i.e., rational function. Instead of a fixed activation function $f(x)$, the padée approximation is the rational function $F(x)$ over polynomials $P(x)$, $Q(x)$ of order m, n of the form

$$\begin{aligned} F(x) = \frac{P(x)}{Q(x)} &= \frac{\sum_{j=0}^{m} a_j x^j}{1+\sum_{k=1}^{n} b_k x^k} \\ &= \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m}{1 + b_1 x + b_2 x^2 + \cdots + b_n x^n} \end{aligned} \tag{7}$$

which agrees with $f(x)$ the best. The Padé approximant gives an approximation of a function $f(x)$, instead of fixing the coefficient a, b to approximate a particular activation function, PAU allows them to be free parameters that can be optimized end-to-end with the rest of the neural network, which allows the optimization process to find the activation function needed at each layer automatically and allows deep neural networks to provide high predictive performance.

With the above discussion, we summarize the following conclusion:

1. Swish and TanhSoft family approximate to $z(x) = \alpha x$ in the positive direction of x-axis and they are smooth in the negative direction of x-axis, which can make the network more stable and the back propagation operation easier.

2. Swish, TanhSoft-1, TanhSoft-2 functions are unsaturated, and the derivatives of them are not always zero in the negative direction of x-axis, which can effectively alleviate the gradient vanishing problem of ReLU. The major structure of them are similar to ReLU, which is the product of the

original input x and the composite function, it brings sparsity to the neural network and reduces calculating amount of the network.

3. Padé approximant can approximate a fixed activation function by a rational function of given orders m and n. Padé activation Units coefficients are allowed to be optimized with the rest of the neural network to find the activation function needed at each layer automatically.

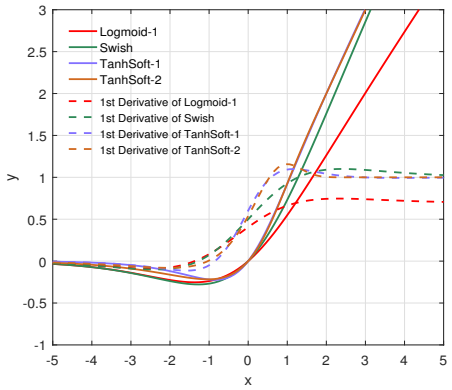
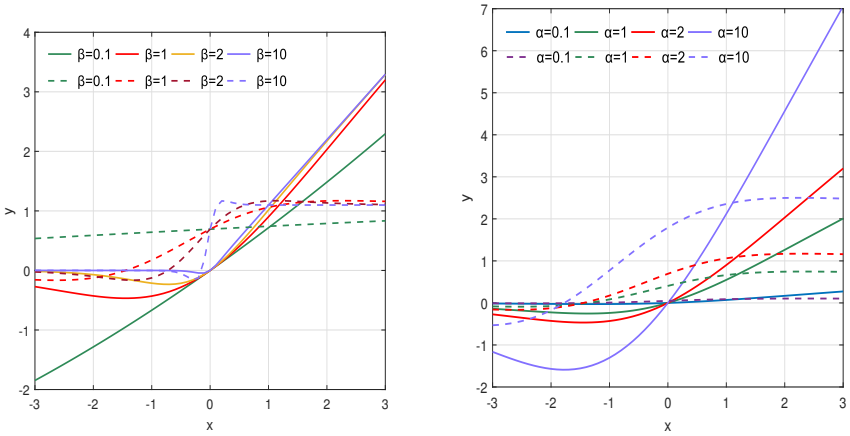


Fig. 1: Numeric evaluations of Logmoid-1, Swish, TanhSoft-1 ($\alpha = 0.87$) and TanhSoft-2 ($\beta = 0.6$, $\gamma = 1$).

3 LOGMOID ACTIVATION FUNCTION FAMILY

In this section, we will propose a family of functions indexed by two hyper-parameters, we refer to this family as the Logmoid family.



(a) (b)

Fig. 2: (a) Numerical evaluations of Logmoid (solid line) and its derivative (dashed) in terms of β with $\alpha = 1$. (b) Numerical evolutions of Logmoid (solid line) and its first derivative (dashed line) with respect to α and $\beta = 1$.

Standard ANN training process involves tuning the weights in the linear part of the network, however, there is a merit in the ability to custom design activation functions, to better fit the problem at hand. Here, rather than looking at individual activation functions, we propose a family of functions indexed by two hyper-parameters. We refer to this family as the Logmoid family as it is created by combining a hyper-parametric versions of the logarithmic Sigmoid activation functions. Explicitly, we express it as

$$f(x; \alpha, \beta) = x \ln(1 + \alpha\sigma(\beta x)) \tag{8}$$

Table 1: The Top-1 accuracy of Logmoid with parameters

Parameter($\alpha = 1$)	Top-1	Parameter($\beta = 1$)	Top-1
20	77.73%	20	70.34%
15	76.42%	15	64.95%
10	76.93%	10	68.48%
5	•**78.2**%	5	73.99%
1	77.5%	1	•**77.5**%
0.1	51.6%	0.1	-
0.01	36.2%	0.01	-
-0.01	41.28%	-0.01	-
-0.1	51.82%	-0.1	-
-1	77.11%	-1	-
-5	77.98%	-5	-
-10	77.06%	-10	-

where σ refers to the Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Table 1 shows the Top-1 accuracy of Logmoid with different parameters. When $\alpha = 1$, the result for β from 1 to 20 or from -10 to -1 are similar, the accuracy decreases faster when $-1 < \beta < 1$, and if $\beta = 5$, it gets the 78.2% Top-1 accuracy. When $\beta = 1$, the accuracy decreases faster with the gradual increase of α, Logmoid with $\alpha = 1$ has a better performance, and $\alpha < 1$ could not converge.

Fig.2(a) shows Logmoid and its derivative in terms of β, where the first derivative of Logmoid is given by

$$\frac{df(x)}{dx} = \ln(1 + \alpha\sigma(\beta x)) + \frac{\alpha\beta x\sigma(-\beta x)}{1 + \alpha + e^{-\beta x}} \tag{9}$$

For $\alpha = 1$, Logmoid is a scaled linear function $f(x) = kx$ if $\beta \to 0$. Similar to ReLU function, the sigmoid component in Logmoid becomes a 0-1 function as $\beta \to +\infty$. Logmoid is then regarded as a smooth nonlinear function which interpolates between a linear function and ReLU. Here, β shows how fast the first derivative changes asymptotically. When $\beta = 1$, the derivative has magnitude less than 1 for inputs that are less than around 1, which implies that the gradient preserves property of ReLU (i.e., having a derivative of 1 when $x > 0$). This can avoid the so-called dying problem.

From Fig.2(b), Logmoid is far away from the linear function $f(x) = x$ in the positive part for large α and $\beta = 1$. Here, we speculate its maximum slopes in the negative part. The derivative increases in the positive part as α increasing. With lots of simulation experiments in α, the gradient will explode or disappear when it is too large or too small.

The interplay between the hyper-parameters α and β plays a major role for the Logmoid family and controls the slope of the curve in both positive and negative axes. Though for experimental and practical purpose, we restrict to small ranges of the hyper-parameters ($1 \leqslant \alpha \leqslant 5$, $1 \leqslant \beta \leqslant 5$), which also has the similar properties like Swish. In this paper, we consider the sub-family $f(x; 1, 1)$ and called it as Logmoid-1. In what follows, we discuss the properties of the Logmoid-1.

4 SUBFAMILY: LOGMOID-1

Similar to other smooth functions, Logmoid-1 is unbounded above and bounded below. Different from ReLU, Logmoid-1 is smooth and non-monotonic activation function. Logmoid-1 also inherits the self-gated property defined in Swish. Here, the self-gated refers to the input itself and a function with the input as its argument, i.e. $f(x) = xg(x)$. This implies the network will not change the initial distribution of the input on the positive part, meanwhile it produces a buffer at the negative part near zero. The curve and derivative curve of Logmoid-1 are illustrated in Fig.1.

The output landscapes of a simple five-layer fully connected network with different activation functions are shown in Fig.3. The landscapes are generated by passing in the coordinates to a five-layer randomly initialized neural network, which outputs the corresponding scalar magnitude. ReLU has

a lot of sharp transitions as compared to the smooth profiles of others. Meanwhile, Logmoid-1 shows a continuous and fluent transition shape. Smoother output landscapes suggest smooth loss landscapes (Li et al., 2018). This property of Logmoid-1 can help in easier optimization and better generalization.

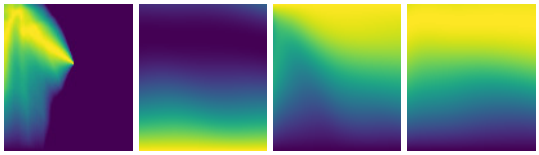


Fig. 3: Comparison between the output landscapes of ReLU, Swish, TanhExp and Logmoid-1 on a 5-layer network.

4.1 LOGMOID-1 APPROXIMATION

In this section, we estimate the approximate error by using the method of neural network operators with special Logmoid-1.

4.1.1 THE BELL-SHAPED FUNCTION CONSTRUCTED WITH LOGMOID-1

From Eq.(8) a simple Logmoid-1 is defined as

$$g(x) = x\ln(1+\sigma(x)), x \in \mathbb{R} \tag{10}$$

that is, $\alpha = \beta = 1$. This implies a new function as

$$\varphi(x) = g(x+\frac{1}{2}) - g(x-\frac{1}{2}), x \in \mathbb{R} \tag{11}$$

where $\varphi(x)$ is bounded function. Now, consider the following function

$$\begin{aligned} \psi(x) &= \varphi(x+\frac{1}{2}) - \varphi(x-\frac{1}{2}) \\ &= g(x+1) - 2g(x) + g(x-1) \end{aligned} \tag{12}$$

Here, $\psi(x)$ is not an even function. According to the bell-shaped function defined in Ref.(Cardaliaguet & Euvrard, 1992), the function $\psi(x)$ in Eq.(12) should be non-negative and increasing on $(-\infty, \mu)$ and decreasing on $[\mu, +\infty)$, where μ belongs to $\mathbb{R}$. So, we get a new bell-shaped function as follows

$$\Psi(x) = \begin{cases} \psi(x) + 2|Y_1|, & \text{if } T_0 \leq x < T_1; \\ \lambda(\psi(x) + 2|Y_2|), & \text{if } T_2 < x < T_0; \\ -\psi(x), & \text{if } x \geq T_1; \\ -\lambda\psi(x), & \text{if } x \leq T_2. \end{cases} \tag{13}$$

where $\lambda = 0.934$, $T_0 = -0.2536$, $T_1 = 3.5025$ and $T_2 = -3.7326$ are the extreme points, and $Y_1 = -0.0181$ and $Y_2 = -0.0307$ are the values of the $\psi(x)$ at points T_1 and T_2. Here, the parameter λ is used to ensure the approximate continuity of $\Psi(x)$ at T_0.

A simple computation gives

$$\sum_{k=-\infty}^{\infty} \Psi(x-k) = 1.1918 := \xi \tag{14}$$

From Eq.(14), a new bell-shaped function $\Phi(x)$ is defined as

$$\Phi(x) = \frac{1}{\xi}\Psi(x) \tag{15}$$

It follows that $\sum_{k=-\infty}^{\infty} \Phi(x-k) = 1$. Now, we define $\Gamma(x)$ as

$$\Gamma(x) = \begin{cases} e^{-\frac{1}{2}x-\frac{1}{2}}, & \text{if } x > 0; \\ e^{\frac{1}{2}x-\frac{1}{2}}, & \text{if } x \leq 0. \end{cases}$$

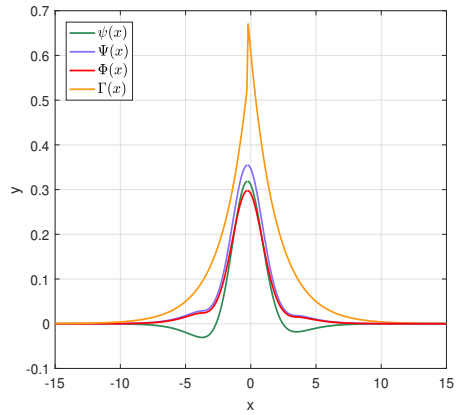


Fig. 4: Numeric evaluations of $\psi(x)$, $\Psi(x)$, $\Phi(x)$ and $\Gamma(x)$.

Here, $\Gamma(x)$ is an even function. Numeric evaluations of functions $\psi(x), \Psi(x), \Phi(x)$ and $\Gamma(x)$ are shown in Fig.4.

Proposition 1 *The bell-shaped functions $\Phi(x)$ and $\Gamma(x)$ satisfy the following properties.*

(1) $\Phi(x) > 0$ *for any* $x \in \mathbb{R}$*;*

(2) $\sum_{k=-\infty}^{\infty} \Phi(x-k) = 1$ *for any* $x \in \mathbb{R}$*;*

(3) $\sum_{k=-\infty}^{\infty} \Phi(nx-k) = 1$ *for* $x \in \mathbb{R}, \forall n \in \mathbb{N}$*;*

(4) $\Phi(x)$ *is decreasing on* $(T_0, +\infty)$*, and increasing on* $(-\infty, T_0]$*;*

(5) $\Phi(x) \leq \Gamma(x)$ *for any* $x \in \mathbb{R}$*.*

The proof of Proposition 1 is easily completed by straight forward evaluations.

4.1.2 Approximation errors

In this subsection, we firstly present some preliminary results. And then, we introduce a class of quasi-interpolation operators with the function $\Phi(x)$ defined by Eq.(15).

Feed-forward neural networks (FNNs) with a hidden layer and $n+1$ neurons can be represented by

$$N_n(x) = \sum_{j=0}^{n} c_j \varsigma(\langle a_j \cdot x \rangle + b_j) + d_j, x \in \mathbb{R}^s, s \in \mathbb{N} \tag{16}$$

where ς is the activation function of the network, $j \in [0, n]$ and $b_j \in \mathbb{R}$ are thresholds of the hidden layer, $a_j \in \mathbb{R}^s$ are the connection weights between the input layer and hidden layer, $c_j \in \mathbb{R}$ are the connection weights between the hidden layer and output layer, $d_j \in \mathbb{R}$ are the thresholds of the output layer, and $\langle \cdot \rangle$ denotes the inner product. FNNs are known to be universal approximation operators (Hornik et al., 1989; Cybenko, 1989). It is known that by increasing the number of hidden neurons, any continuous function on a compact set can be theoretically approximated up to any desired precision (Cybenko, 1989; Hornik et al., 1990; Hornik, 1991; Funahashi, 1989; Leshno et al., 1993). Our goal here is to present a tight estimation of approximation error with Logmoid-1 in Eq.(10).

Let $\mathcal{C}[-1, 1]$ be the space of continuous functions defined on $[-1, 1]$. For any function $f \in \mathcal{C}[a, b]$, the modulus of its continuity is defined by

$$w(f, \delta) := \max_{\substack{a \leq x, y \leq b \\ |x-y| \leq \delta}} |f(x) - f(y)|. \tag{17}$$

This modulus is used for measuring its approximation error and smoothness. The function f is called (L,α)-Lipschitz continuous with $\alpha \in (0,1]$ (e.g., $f \in Lip(L,\alpha)$) if and only if there is a constant $L > 0$ such that

$$w(f,\delta) \leq L\delta^{\alpha} \tag{18}$$

We now construct FNN operators and prove approximation results. In fact, for any function $f \in \mathcal{C}[-1,1]$, a FNN operator is given by

$$G_{Logmoid-1}(f,x) := \sum_{k=-n}^{n} f(\frac{k}{n})\Phi(nx-k) \tag{19}$$

where $\Phi(x)$ is defined by Eq.(15). The main result is as follows.

Theorem 1 *Let $f \in \mathcal{C}[-1,1]$, $n \in \mathbb{N}$, $0 < \alpha \leq 1$. For the neural networks approximation operators $G_{Logmoid-1}(f,x)$ defined by Eq.(19), we have*

$$|f(x) - G_{Logmoid-1}(f,x)| \leq w(f,\frac{1}{n^{\alpha}}) + (8e^{-\frac{1}{2}n^{1-\alpha}} + \frac{4}{e})\|f\|_{\infty}$$

where $\|\cdot\|_{\infty}$ denotes the uniform norm.

Theorem 1 means that the approximation error of $G_{Logmoid-1}(f,x)$ and $f(x)$ can be controlled by n and α. This means that for any $f \in \mathcal{C}[-1,1]$, the constructed network $G_{Logmoid-1}(f,x)$ can approximate $f(x)$ to arbitrary precision, which depends on the number of hidden neurons n and smoothness exponent of the target function α. The proof of Theorem 1 is shown in Appendix A.

In general, let $\mathcal{C}_B(\mathbb{R})$ be the set of continuous and bounded function on $\mathbb{R}$. For any $f \in \mathcal{C}_B(\mathbb{R})$, we construct FNN operator as

$$\overline{G}_{Logmoid-1}(f,x) := \sum_{n=-\infty}^{\infty} f(\frac{k}{n})\Phi(nx-k). \tag{20}$$

Similar to Theorem 1 we prove the following theorem for any continuous and bounded function on $\mathbb{R}$.

Theorem 2 *For any function $f \in \mathcal{C}_B(\mathbb{R})$, $n \in \mathbb{N}$, $0 < \alpha \leq 1$, we have*

$$\left|f(x) - \overline{G}_{Logmoid-1}(f,x)\right| \leq w(f,\frac{1}{n^{\alpha}}) + 8e^{-\frac{1}{2}n^{1-\alpha}}\|f\|_{\infty}$$

where $\overline{G}_{Logmoid-1}(f,x)$ is defined in Eq.(20).

The proof of Theorem 2 is shown in Appendix B.

To verify the correctness of the present theorems, we present two examples. One is for the function $y = x^2$ on [-1,1], as shown in Fig.5(a). The average absolute approximate error is 0.13 (0.08 or 0.02) for $n = 5$ ($n = 10$ or 50). The other is to approximate the function $y = \sin(\pi x)$ on [-5,5] by using $\overline{G}_{Logmoid-1}(f,x)$, as shown in Fig.5(b). The average absolute approximate error is 0.11, 0.02 and 0.01 for $n = 10$, 30, and 100, respectively. Both approximation errors decrease with the increasing of neuron number n.

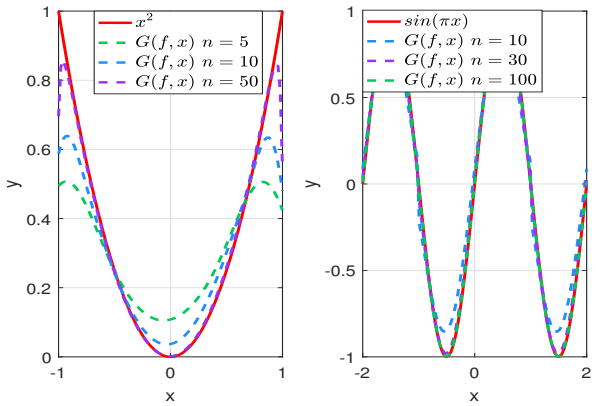
Similarly, we now construct a new bell-shaped function using Swish activation function combined with Eq.(13, 14, 15), the expression is

$$\Phi_{Swish}(x) = \begin{cases} \xi(\phi(x) + 2|Y_1|), & \text{if } -T_1 \leq x \leq T_1; \\ -\xi\phi(x), & \text{otherwise.} \end{cases} \tag{21}$$

where $\lambda = 1.758$, T_1 is the extreme point, $Y_1 = -0.034$ is the value of the $\phi(x)$ at point T_1. $\phi(x)$ can be defined as follows

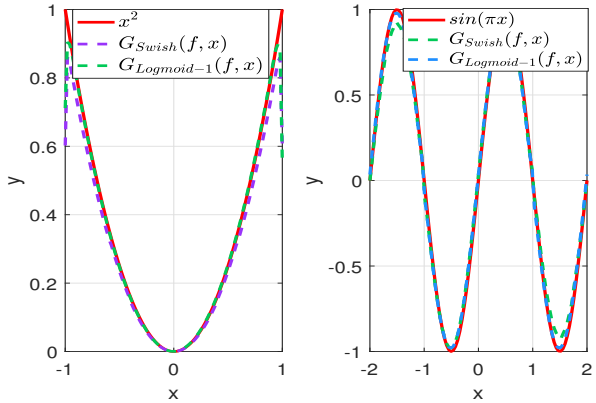
$$\phi(x) = \text{Swish}(x+1) - 2 * \text{Swish}(x) + \text{Swish}(x-1) \tag{22}$$

where Swish is in Eq.(2), it is obviously that $\phi(x)$ is an even function.



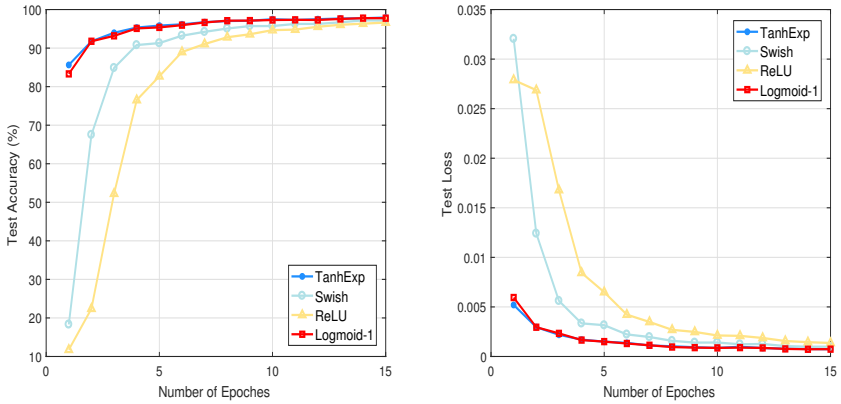
(a) (b)

Fig. 5: (a) Uniform approximation of function $y = x^2$ on [-1,1] with the network operator $G_{Logmoid-1}(f, x)$. (b) Uniform approximation of function $y = \sin(\pi x)$ on [-5,5] with the network operator $\overline{G}_{Logmoid-1}(f, x)$.



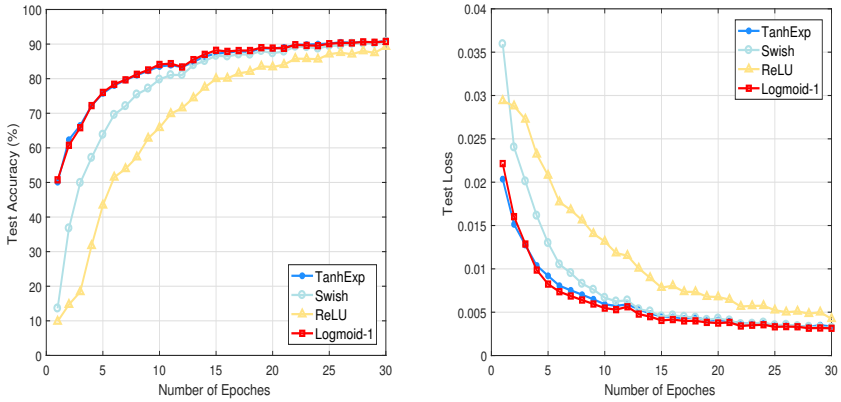
(a) (b)

Fig. 6: Uniform approximation using the network operators $\overline{G}_{Logmoid-1}(f, x)$ and $\overline{G}_{Swish}(f, x)$. (a) $y = x^2$ on [-1,1]. (b) $y = \sin(\pi x)$ on [-5,5] for $n = 30$.



(a) (b)

Fig. 7: Test of Logmoid-1, TanhExp, Swish, and ReLU with a 15-layer basic network on MNIST. (a) Test accuracy. (b) Test loss.



(a) (b)

Fig. 8: Test of Logmoid-1, TanhExp, Swish, and ReLU with a 15-layer basic network on KMNIST. (a) Test accuracy. (b) Test loss.

Now, we can construct the FNN operator $G_{Swish}(f,x)$ based on $\Phi_{Swish}(x)$, to further verify the performance of Logmoid-1, we compared the approximate errors by $G_{Logmoid-1}(f,x)$ and $G_{Swish}(f,x)$. One is for function $y = x^2$ on [-1,1], as shown in Fig.6(a), the average absolute approximate error is 0.01, 0.03 for $G_{Logmoid-1}(f,x)$, $G_{Swish}(f,x)$ when $n = 100$. The other is to approximate the function $y = \sin(\pi x)$ on [-5,5] by using $\overline{G}_{Logmoid-1}(f,x)$, $\overline{G}_{Swish}(f,x)$, as shown in Fig.6(b). The average approximate error is 0.03, 0.05 for $\overline{G}_{Logmoid-1}(f,x)$, $\overline{G}_{Swish}(f,x)$ when $n = 30$.

4.2 Experiments with Logmoid-1

In this subsection, several ablation experiments will be performed to evaluate the effectiveness of Logmoid-1 on MNIST and KMNIST, which both have 10 different classes. All the experiments are simulated with a basic network (Liu & Di, 2020), which is designed for TanhExp. The network architecture has 15 layers. Each one of the last 12 layers contains a batch normalization part (Ioffe & Szegedy, 2015), a dropout rate (Srivastava et al., 2014) of 0.25, an activation function, and a dense layer with 500 neurons.

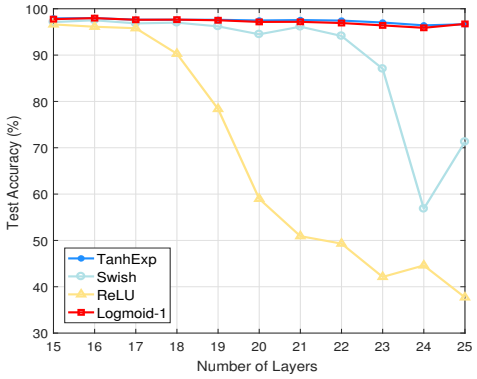


Fig. 9: Test accuracy of Logmoid-1, TanhExp, Swish, and ReLU by increasing the layer of the basic network on MNIST.

Comparison of learning speed. The learning speed is compared by the baseline activation functions and Logmoid-1 in the same settings. We trained the basic 15 layers network with 15 epoches on MNIST and 30 epoches on KMNIST. As shown in Fig.7, with the increasing of epoch, the test accuracy steadily increase while the loss decreases on all activation functions. Logmoid-1 is going beyond Swish and ReLU in both the convergence speed and the final accuracy, and has a similar performance with TanhExp. The test accuracy of Logmoid-1 after the first epoch is 0.6352. This implies Logmoid-1 is able to update the parameters rapidly and force the network to fit the dataset in an effective way.

Comparison of the ability of preventing over-fitting. We evaluated the ability of preventing over-fitting when increasing the number of layers in the basic network. Experiments were performed with Logmoid-1, ReLU, Swish, and TanhExp on MNIST and varied the number of blocks from 15 to 25. Similar experiments were carried with the same hyper-parameters for KMNIST. From Fig.9, there is a sharp decrease in accuracy for both ReLU and Swish. This means the network suffers from over-fitting when the network goes deeper for ReLU and Swish. Instead, Logmoid-1 and TanhExp can maintain a significantly high accuracy in large models where the optimization becomes difficult. So, Logmoid-1 and TanhExp can prevent the network from over-fitting while other smooth activation functions do not maintain the stability as the increasing of the number of layers.

5 LOGMOID ACTIVATION UNITS(LAU)

The above section study the performance of Logmoid-1 and other activation functions under different dataset conditions. Now, let's now formally introduce LAU, the formula of LAU with parameters is

$$f(x;\alpha,\beta) = x\ln(1+\alpha\sigma(\beta x)) \tag{23}$$

where α and β are trainable hyper-parameters. Here, instead of fixing the coefficients α, β to show a particular activation function, we allow them to be free parameters that can be optimized end-to-end with the rest of the neural network to find the activation function needed at each layer automatically.

Having LAUs at hand, one can define Logmoid networks as follows: Logmoid networks are feed-forward networks with LAUs that may include convolutional and residual architectures with pooling layers. To effectively use Logmoid networks, one can simply replace the standard activation functions in a neural network by LAUs, and then continued to optimize all parameters. However, even if every LAU contains a low number of parameters, in the extreme case, learning one LAU per neuron may considerably increase the complexity of the networks and in turn the learning time. To ameliorate and be inspired by the idea of PAUs, we suppose to learn one LAU per layer. Therefore, we only add $2L$ parameters, where L is the number of activation layers in the network. The last step missing before we can start the the optimization process is to initialize the coefficients of the LAUs. Surely, one can do random initialization of the coefficients and allow the optimizer to train the network end-to-end. However, we obtained better results after initializing all LAUs with Logmoid-1.

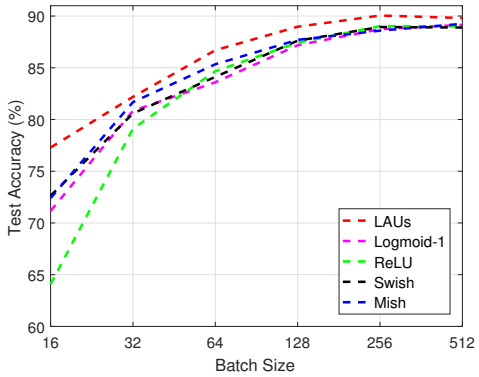


Fig. 10: The test accuracy vs Batch size for LAUs, Logmoid-1, ReLU, Swish, Mish.

Table 2: The Top-1 accuracy (in %) of different learning rates (lr) for LAUs, Logmoid-1, ReLU, Swish, Mish.

lr	ReLU	Swish	Mish	Logmoid-1	LAUs
10^{-1}	87.42	87.62	87.7	87.18	•**88.97**
10^{-2}	87.91	88.15	88.38	88.36	•**88.58**
10^{-3}	77.92	80.38	•**80.99**	80.17	80.44
10^{-4}	50.33	51.48	53.66	52.53	•**55.36**

5.1 Experiments with LAUs

In this subsection, we simulate LAUs and other baseline activation functions including ReLU, Swish, Mish, TanhExp and Logmoid-1 using standard deep neural networks. We benchmark the results in the following datasets: MNIST (LeCun et al., 2010), Kuzushiji-MNIST (KMNIST) (Clanuwat et al., 2018), Fashion-MNIST (FMNIST) (Xiao et al., 2017), CIFAR (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015).

5.1.1 Ablation experiments with LAUs

In this subsubsection, in order to show the superriority of LAUs, we will demonstrate the image classification performance of different components in the network such as batch size, learning rate with LAUs and other activation functions. ShuffleNet-V2 network was used as the backbone and all the experiments are trained and evaluated on CIFAR-10 dataset.

Analysis of different batch size. Batch size is the number of samples selected for a training set, which can affect the degree of model optimization. Generally, the larger the batch size is, the more accurate the descent direction and the smaller the training shock will be. However, if the batch size increases to a certain extent, the descent direction almost no longer changes, In addition, batch size also affects the usage of GPU memory and the training speed. Batch size is usually set to a power of 2, which can speed up the binary calculation and GPU computing and storage. So, we will test the performance of LAUs with different batch sizes in the same network structure. Here, we select samples with batch size of 16, 32, 64, 128, 256, 512 respectively, and the image classification test Top-1 accuracy results with LAUs, Logmoid-1, Swish, Mish and ReLU activation functions are shown in Fig.10. From Fig.10, we can konw that LAUs has the best performance with all batch sizes compared with other activation functions. When batch size is less than 256, the test accuracy increases significantly as the batch size grows, and becomes steady when batch size is too large.

Analysis of different learning rate. The learning rate represents the information accumulation speed in the neural network training. In an ideal situation, we will start with a large learning rate, and with the increasing of iteration number, the learning rate will gradually decrease, so as to ensure

Table 3: Overall architecture of VGG-8-Dilated.

No.	Output size	k/s	p/d	#params
1	Image: 1@32×32			0
2	Convolutional: 64@32×32	3/1	1/1	640
3	Activation			2
4	Max-Pooling: 64@16×16	2/2	0/1	0
5	Convolutional: 128@16×16	3/1	1/1	73,856
6	Activation			2
7	Max-Pooling: 128@8×8	2/2	0/1	0
8	Convolutional: 256@6×6	3/1	1/2	295,168
9	Activation			2
10	Convolutional: 512@4×4	3/1	1/2	1180,160
11	Activation			2
12	Convolutional: 512@2×2	3/1	1/2	2,359,808
13	Activation			2
14	Linear: 10			20,490

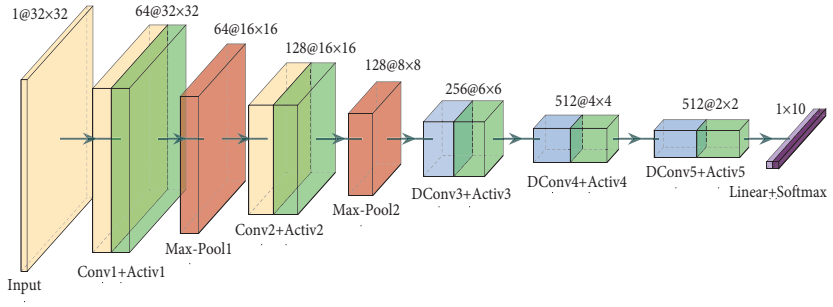


Fig. 11: Schematic VGG-8-Dilated network. DConv refers to the dilated convolutional.

that the model will not fluctuate too much in the later training stage. Table 2 illustrates the image classification Top-1 accuracy comparable results of LAUs, Logmoid-1, Swish, Mish and ReLU activation functions in ShuffleNet-V2 network with four different learning rates.

As can be seen from Table 2, if the learning rate is 10^{-1}, LAUs has the highest 88.97% test accuracy. If the learning rate is 10^{-2}, the test accuracy of the three nonmonotonic activation functions are almost 88%, which proves that different nonmonotonic function have the similar performance in the small learning rate. And if the learning rate is 10^{-3}, Mish has the best test accuracy performance. If the learning rate is 10^{-4}, the accuracy of LAUs is higher than ReLU, Swish, Mish and Logmoid-1. All these results demonstrate that LAUs has best performance in almost comparable learning rates, and the expression ability of LAUs at a smaller learning rate is significantly better than that of Logmoid-1, Swish, Mish and ReLU.

5.1.2 Empirical results on Fashion-MNIST

We take use of two basic architectures: LeNet (Lecun et al., 1998) and VGG-8 (Simonyan & Zisserman, 2014) to evaluate Logmoid on FMNIST. The learnable Logmoid virtual networks are used to replace the activation functions in LeNet. The network of LeNet contains 61,706 trainable parameters while LAUs adds 8 additional parameters. We investigate whether there are smaller subnetworks that make its performance similar to the original network. Here, we replace some blocks of the VGG-8 network with dilated convolutions, which is named as VGG-8-Dilated network with 3,930,132 parameters, shown in Fig.11. Its architecture is shown in Table 3.

From Table 4, LAUs improves 1.6% Top-1 accuracy compared with the runner-up result on VGG-8. By comparing the baseline activation functions on the VGG-8-Dilated and LeNet architectures,

there is no clear activation option for optimal performance. However, LAUs always matches or outperforms the best performance of baseline activation functions. This shows LAU is a learnable activation function, which does not require network designers to perform suboptimal experiments.

Table 4: Performance comparison of activation functions on FMNIST. The input size of train and test is 32×32. The results are averaged over 5 reruns Top-1 classification accuracy (in %) across various deep neural architectures. The best (•) and runner-up (○) results per architecture are shown with bold numbers.

Activation	LeNet #0.06M	VGG-8 #9.2M	VGG-8-Dilated #3.9M
ReLU	○**89.84**	90.19	90.96
Swish	89.37	90.15	90.80
Mish	88.87	90.09	○**91.21**
TanhExp	89.47	○**90.72**	90.45
Lgomoid-1	88.99	90.56	90.25
LAUs	•**89.90**	•**92.35**	•**91.84**

5.1.3 LEARNING ACTIVATION FUNCTIONS ON FASHION-MNIST

LAU as a flexible activation function presents similarities to standard functions by checking up the activation functions learned from the data. In fact, from Fig.12 some learned activations are smooth versions of Logmoid-1 because their shapes can be controlled by hyper-parameters α and β. Hence, we initialize LAUs with coefficients that match Logmoid-1. This provides evidences that LAUs can learn new activation functions per layer automatically.

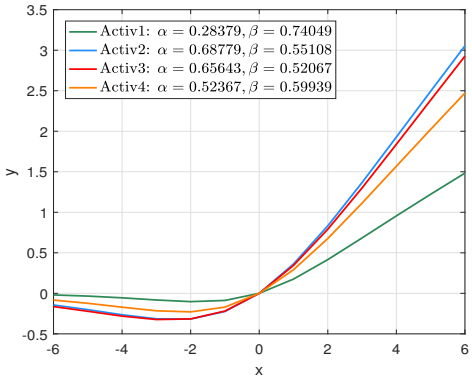


Fig. 12: The estimated activation functions after training the LeNet network in Fig.11 with LAUs on FMNIST.

5.1.4 EMPIRICAL RESULTS ON CIFAR

We explore the performance of LAUs on a difficult dataset, CIFAR (Krizhevsky et al., 2009). There are two types: CIFAR-10 and CIFAR-100. These 3-channel images require a neuron network with stronger learning ability. Here, we evaluate the performance of LAUs as well as other baseline activation functions in both datasets on networks of ResNet-20 (RN-20) (He et al., 2016), MobileNet (MN) (Howard et al., 2017), MobileNet-V2 (MN-V2) (Sandler et al., 2018), ShuffleNet (SN) (Zhang et al., 2018), ShuffleNet-V2 (SN-V2) (Zhang et al., 2018), SqueezeNet (SQ) (Iandola et al., 2016), SeNet-18 (SN-18) (Hu et al., 2018) and EfficientNet-B0 (EN-B0) (Tan & Le, 2019).

The dataset CIFAR-10 has 10 classes with 5,000 training images and 1,000 test images per class. From Table 5 LAUs outperforms the baseline activation functions in most cases. Especially, LAUs improves the Top-1 classification accuracy by around 5% compared LAUs other baseline activation functions on EfficientNet-B0. For the predictive performance of LAUs on the networks ResNet-20 and SqueezeNet, the present learnable activation function is still competitive and close to the best.

Table 5: Performance comparison of activation functions on CIFAR-10 (train and test on 32×32 input size). The results are averaged over 5 reruns Top-1 accuracy (in %) across various deep neural architectures. The best (•) and runner-up (○) results per architecture are shown with bold numbers.

Model	ReLU	Swish	Mish	TanhExp	Logmoid-1	LAUs
RN-20	90.96	91.08	•**91.33**	•**91.26**	90.97	91.12
MN	86.04	87.93	○**88.27**	88.24	87.91	•**88.53**
MN-V2	84.92	85.87	○**86.83**	85.83	86.39	•**87.95**
SN	87.55	88.87	88.18	○**89.2**	87.22	•**89.91**
SN-V2	87.42	87.62	○**87.70**	87.38	87.18	•**88.97**
SQ	•**88.50**	87.04	88.11	87.86	87.63	○**88.22**
SN-18	88.81	90.53	90.29	90.19	○**90.95**	•**91.43**
EN-B0	76.17	77.35	77.47	77.19	○**78.23**	•**82.52**

The dataset CIFAR-100 has 100 classes with 500 training images and 100 test images per class. From Table 6 LAUs is in the lead or close to the best when it comes to the predictive performance on most network architectures. Particularly, LAUs always outperforms ReLU on all network architectures. Additionally, LAUs improves the top-1 classification accuracy by 4.6%, 3.3%, 4.4%, 2.5% and 2.9% compared with ReLU, Logmoid-1, Swish, Mish and TanhExp on EfficientNet-B0. LAUs reduces the classification accuracy by 1.4% compared with the best on ShuffleNet-V2. This may be improved by exploring different initialization and optimization parameters.

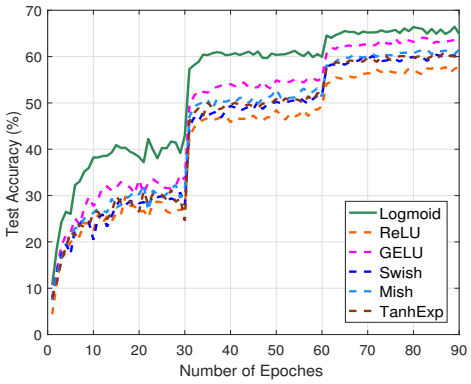


Fig. 13: ShuffleNet-V2 Top-1 test accuracy for multiple activation functions on ImageNet-200.

Table 6: Performance comparison of activation functions on CIFAR-100 (train and test on 32×32 input size). The results are averaged over 5 reruns Top-1 accuracy (in %) across various deep neural architectures. The best (•) and runner-up (○) results per architecture are shown with bold numbers.

Model	ReLU	Swish	Mish	TanhExp	Logmoid-1	LAUs
RN-20	66.16	67.73	67.36	•**67.79**	67.68	○**67.77**
MN	59.40	62.11	•**62.48**	○**62.21**	61.81	62.05
MN-V2	59.40	61.32	○**61.94**	60.76	60.09	•**63.66**
SN	64.53	65.45	66.20	66.7	○**66.76**	•**68.01**
SN-V2	63.65	○**65.88**	65.72	•**66.18**	65.69	64.73
SQ	62.64	63.97	○**63.98**	62.38	63.64	•**64.26**
SN-18	66.26	○**67.85**	67.82	67.68	67.45	•**68.96**
EN-B0	45.81	46.01	○**47.89**	47.52	47.08	•**50.38**

5.1.5 EMPIRICAL RESULTS ON IMAGENET-200

ImageNet 2012 (Russakovsky et al., 2015) is considered to be one of the most challenging and significant classification tasks in computer vision. We compared LAUs with other baseline activation functions on this dataset. For small sample case, we choose 200 classes and randomly extracted 500 training images, 50 testing images each class from ImageNet, named as ImageNet-200. From Table 7 LAUs leads in both the top-1 and Top-5 accuracies. In contrast, we get a relatively bad result by using ReLU. Moreover, LAUs shows faster learning compared to the other activation functions as can be seen in Fig.13.

Table 7: ShuffleNet-V2 (SN-V2) Top-1 and Top-5 accuracies (in %) on ImageNet-200 (train and test on 224×224 input size) for different activations. The best (•) and runner-up (○) results are shown with bold numbers.

SN-V2	ReLU	Swish	Mish	TanhExp	Logmoid-1	LAUs
Acc@1	57.76	60.38	61.5	60.05	○**63.07**	•**64.93**
Acc@5	81.02	83.15	84.09	83.07	○**85.02**	•**86.03**

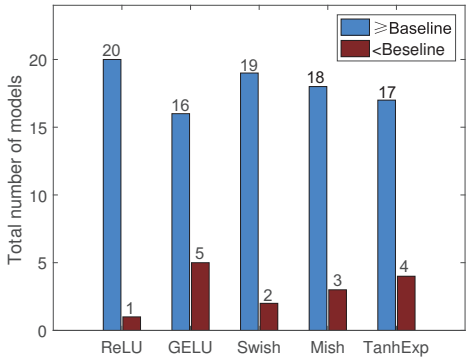


Fig. 14: Total number of models on which Logmoid outperforms or under-performs each baseline activation function in experiments.

5.1.6 SUMMARIZE RESULTS

Finally, we compare LAUs with each baseline activation function that has been considered and aggregated the number of occurrences. The results are shown in Fig.14, which are simulated on FMNIST, CIFAR and ImageNet-200. LAUs outperforms Logmoid-1, ReLU, Swish and Mish in almost all 21 models, and about 75% in terms of TanhExp. This means the present LAUs beats other baseline activation functions in most cases, and under-performs marginally on rare occasions.

6 CONCLUSIONS

In this work, we firstly presented a novel family of activation function, named as Logmoid family and a subfamily called Logmoid-1. Similar to Swish and Mish, it is also a smooth nonmonotonic function with bounded below and no bounded above. Secondly, a class of FNN operators with special Logmoid-1 function is constructed and used to approximate any continuous function. The quantitative approximation errors are estimated by using the continuous modulus of the function. This has resolved the approximation complexity of the constructed FNN operators. Thirdly, a learnable Logmoid Activation Unit called LAU is proposed. LAUs are initialized using Logmoid-1, trainable in an end-to-end fashion using backpropagation. This makes it easy for practitioners to replace standard activation functions with LAU units in any neural networks. The simulations have shown that LAUs can learn new activation functions and in turn novel neural networks that are competitive to state-of-the-art networks with fixed activation functions. Actually, LAUs shows the best performance across all activation functions and architectures. This shows the effectiveness of present learnable activation functions. Further work will be performed to apply LAUs to other computer vision tasks.

REFERENCES

Koushik Biswas, Sandeep Kumar, Shilpak Banerjee, and Ashish Kumar Pandey. Tanhsoft–a family of activation functions combining tanh and softplus. *arXiv:2009.03863*, 2020.

Pierre Cardaliaguet and Guillaume Euvrard. Approximation of a function and its derivative with a neural network. *Neural Networks*, 5(2):207–220, 1992.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv:1812.01718*, 2018.

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

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

M.P. Hassell, J.H. Lawton, and J.R. Beddington. Sigmoid functional responses by invertebrate predators and parasitoids. *The Journal of Animal Ecology*, pp. 249–262, 1977.

Kai-Ming He, Xiang-Yu Zhang, Shao-Qing 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.

Kai-Ming He, Xiang-Yu Zhang, Shao-Qing 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.

Xin-Gui He and Shao-Hua Xu. *Process neural networks: Theory and applications*. Springer Science & Business Media, 2010.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 (2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

Andrew G Howard, Meng-Long Zhu, Bo Chen, Dmitry Kalenichenko, Wei-Jun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018.

Guang-Bin Huang and Haroon A Babri. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE transactions on neural networks*, 9(1):224–229, 1998.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *arXiv:1602.07360*, 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.

Barry L Kalman and Stan C Kwasny. Why tanh: choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pp. 578–581. IEEE, 1992.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Stefan Klein, Josien PW Pluim, Marius Staring, and Max A Viergever. Adaptive stochastic gradient descent optimisation for image registration. *International journal of computer vision*, 81(3):227–239, 2009.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017. ISSN 0001-0782. doi: 10.1145/3065386.

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

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Yann LeCun, Corinna Cortes, and J.C. Burges. Mnist handwritten digit database, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in Neural Information Processing Systems*, 31, 2018.

Shi-Yu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv:1610.04161*, 2016.

Xin-Yu Liu and Xiao-Guang Di. Tanhexp: A smooth activation function with high convergence speed for lightweight neural networks. *arXiv:2003.09855*, 2020.

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.

Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv:1908.08681*, 2019.

Alejandro Molina, Patrick Schramowski, and Kristian Kersting. Padé activation units: End-to-end learning of flexible activation functions in deep networks. *arXiv:1907.06732*, 2019.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv:1811.03378*, 2018.

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhi-Heng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Mark Sandler, Andrew Howard, Meng-Long Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Zuo-Wei Shen, Hai-Zhao Yang, and Shi-Jun Zhang. Deep network approximation characterized by number of neurons. *arXiv:1906.05497*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Bharath K Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert RG Lanckriet. Hilbert space embeddings and metrics on probability measures. *The Journal of Machine Learning Research*, 11:1517–1561, 2010.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ming-Xing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

Bing Xu, Nai-Yan Wang, Tian-Qi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853*, 2015.

Xiang-Yu Zhang, Xin-Yu Zhou, Meng-Xiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

Yuan Zhou, Dan-Dan Li, Shu-Wei Huo, and Sun-Yuan Kung. Soft-root-sign activation function. *arXiv:2003.00547*, 2020.

Hegui Zhu, Huimin Zeng, Jinhai Liu, and Xiangde Zhang. Logish: A new nonlinear nonmonotonic activation function for convolutional neural network. *Neurocomputing*, 458:490–499, 2021.