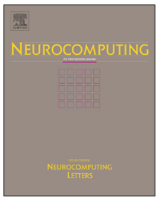
Neurocomputing 216 (2016) 718–734

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems

Shan Sung Liew [a,*], Mohamed Khalil-Hani [a], Rabia Bakhteri [b]

[a] VeCAD Research Laboratory, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[b] Machine Learning Developer Group, Sightline Innovation, #202, 435 Ellice Ave, Winnipeg, MB, Canada R3B 1Y6

ARTICLE INFO

Article history:
Received 13 January 2016
Received in revised form
9 August 2016
Accepted 10 August 2016
Communicated by Jiwen Lu
Available online 14 August 2016

Keywords:
Activation function
Output boundary
Generalization performance
Training stability
Deep neural network
Convolutional neural network

ABSTRACT

This paper focuses on the enhancement of the generalization ability and training stability of deep neural networks (DNNs). New activation functions that we call bounded rectified linear unit (ReLU), bounded leaky ReLU, and bounded bi-firing are proposed. These activation functions are defined based on the desired properties of the universal approximation theorem (UAT). An additional work on providing a new set of coefficient values for the scaled hyperbolic tangent function is also presented. These works result in improved classification performances and training stability in DNNs. Experimental works using the multilayer perceptron (MLP) and convolutional neural network (CNN) models have shown that the proposed activation functions outperforms their respective original forms in regards to the classification accuracies and numerical stability. Tests on MNIST, *mnist-rot-bg-img* handwritten digit, and AR Purdue face databases show that significant improvements of 17.31%, 9.19%, and 74.99% can be achieved in terms of the testing misclassification error rates (MCRs), applying both mean squared error (MSE) and cross-entropy (CE) loss functions This is done without sacrificing the computational efficiency. With the MNIST dataset, bounding the output of an activation function results in a 78.58% reduction in numerical instability, and with the mnist-rot-bg-img and AR Purdue databases the problem is completely eliminated. Thus, this work has demonstrated the significance of bounding an activation function in helping to alleviate the training instability problem when training a DNN model (particularly CNN).

1. Introduction

Since the introduction of artificial neural networks (ANNs) in the last few decades, many research works have been conducted to explore the capability of ANNs in various reasoning and decision based applications. Inspired by how a biological brain works, an ANN possesses the ability to learn from experience, enabling it to solve sophisticated problems that are deemed too difficult for conventional methods.

The expressiveness and approximation properties of a neural network (NN) often rely heavily on its structure. It is believed that deeper models are more powerful in approximating functions, and can capture more information due to their large learning capacities. This is motivated from the biological point of view, that in general our brains and visual systems are composed of multiple stages of information processing. This leads to the development of the deep neural network (DNN) model, which can provide better and deeper abstraction.

In general, there are three main features that characterize a DNN: its model or architecture (connection pattern between neurons), the corresponding learning algorithm (method to search for the optimum weights) and its activation function (transfer function that provides the nonlinearity behavior [1,2]). Different connection patterns can create different types of neuron layers that exhibit various characteristics. Examples of neuron layers include convolutional and pooling layers. A suitable learning algorithm is usually decided upon the DNN model based on how it can lead to a better convergence within the shortest possible time period.

However, the impact of an activation function on the generalization performance and training stability of a DNN is often ignored. There is a lack of consensus on how to select a good activation function for an NN model, and a specific one may not be suitable for all applications. This is especially true for problem domains where the numerical boundaries of the inputs and outputs are the main considerations.

In addition, the training process is heavily dependent on the

* Corresponding author.
E-mail addresses: ssliew2@live.utm.my (S.S. Liew), khalil@fke.utm.my (M. Khalil-Hani), rbakhteri@sightlineinnovation.com (R. Bakhteri).

http://dx.doi.org/10.1016/j.neucom.2016.08.037
0925-2312/© 2016 Elsevier B.V. All rights reserved.

choice of the activation function. As most supervised learning algorithms are based on the backward propagation of the error gradients, the tendency at which an activation function saturates during the back-propagation is one of the main concerns. Output saturation can result in poorer convergence, which is undesirable. Also, since an activation function is applied to the outputs of all neurons in most cases, its computational complexity will contribute heavily to the overall execution time.

Training stability of an NN model defines the magnitude of diversity detected during the training process [3]. A good training stability refers to the condition where the learning process moves consistently (with small dispersion) towards the convergence state. This requires proper design of the NN model and the choice of learning algorithm. The numerical stability during a training process is also crucial to avoid numerical underflow or overflow problem that can render the whole training useless [4].

Many previous works have been reported that stressed on the importance of numerical stability during an NN training, and various approaches have been proposed to alleviate the problem. In general, most approaches ensure the numerical stability by bounding the variables within a certain range. Typical examples include input normalization that scales the input data to a range that is suitable to be processed by the NN [4–6], and output normalization to eliminate the overflow errors [7]. As for the weights, appropriate initialization methods [8] or weight normalization [9] are performed to prevent any potential accumulation overflow. Weight regularization techniques are also incorporated into loss functions to control the maximum amplitudes that the weights can have through the training process [10].

A typical learning algorithm for the NN model involves weight updates through gradient computations, and requires a loss function to calculate the errors. Many works have been reported on modifying the loss functions [11–15] (especially the ones that involve exponential operations) to alleviate the numerical overflow problem by limiting the magnitude of the errors produced, which are then used to calculate the gradients through back-propagations. Some works have proposed to clip the gradients with respect to a bounded range [16,17] (to avoid the ‘exploding’ gradients [18]) or limit the maximum update sizes prior to the weight updates. Careful selection of hyperparameters that are involved in a training process can affect the overall numerical stability as well [19]. For instance, improper learning rate values (that are too large) can cause the training to diverge and overflow [20–22].

Most research works on the activation functions are focused on the complexity of the nonlinearity that an activation function can provide [1], how well it can propagate errors [10], or how fast it can be executed [23], but often neglect its impact on the overall training stability. The numerical stability during a training process is largely dependent on the input and output boundaries of the activation function as well as the numerical representation of the physical computing machine [24]. Larger boundary values allow for more efficient propagation of neurons' values, but with higher risk of getting into the numerical overflow problem, which causes unstable outputs in a trained NN model. This should be taken into considerations as well when designing a suitable activation function for an NN model.

In this paper, we propose new activation functions for DNNs that are applied in visual pattern recognition problems. Convolutional neural network (CNN) has been chosen as our baseline DNN model, and the proposed activation functions are evaluated on a number of case studies that include classification of handwritten digits and face recognition problem. The contributions of this paper are as follows:

- New bounded activation functions for DNNs are proposed. The functions, which include bounded ReLU, bounded leaky ReLU, and bounded bi-firing, improve the training stability of the corresponding unbounded activation functions while achieving better generalization performances; and
- Derivation of a set of coefficient values for the scaled hyperbolic tangent activation function that leads to a more efficient network convergence and higher classification accuracy.

The paper is organized as follows. Section 2 discusses on common activation functions that are applied in DNN models. Section 3 proposes new activation functions based on the desired characteristics of a good activation function. Section 4 describes the experimental design and methodology of performing the analysis. Section 5 analyzes the experimental results, and Section 6 concludes the results and suggests future work.

2. Activation functions

An activation function is a transfer function that transforms the net input of a neuron into an output signal. It determines the total strength (amplitude) that a neuron will produce and receive, thus affecting the overall behavior of an NN. In addition, it is a vital component that usually provides nonlinearity [2], which is important for NNs to possess the ability of approximating functions as defined by the universal approximation theorem (UAT) [25].

In general, there are several universal approximation properties (UAPs) to be fulfilled by an activation function based on the UAT [1]:

1. The output is non-constant for all ranges of inputs;
2. Bounded within a range of values, where there exists a real number P such that $|f(x)| \le P$;
3. Continuous over all values of point c of its domain, where $\lim_{x\to c} f(x) = f(c)$;
4. Monotonically increasing, where $x \le y$ and $f(x) \le f(y)$; and
5. Differentiable everywhere, i.e. the slope of the tangent line of points from the left is approaching the same value as another slope of the tangent of points from the right.

The fifth property is not necessary for the existence of UAP, but is valid for learning algorithms that involve the backpropagation (BP) algorithm, since the error gradients are computed through partial differentiations. However, an activation is not necessary to be differentiable everywhere, as long as it is partially differentiable to be able to propagate the error gradients (as in many recent activation functions [10,23,26,27]).

Realizing the importance of the activation function in ANNs, many functions have been proposed for various specific applications. Most of them are designed based on the UAT, while others partially fulfill the properties, yet still achieve outstanding learning performance [10,26]. Typically, activation functions can be categorized into two basic families based on their shape of curves, i.e. sigmoidal or non-sigmoidal [28].

2.1. Sigmoidal functions

A sigmoidal function refers to any mathematical function having an S-shaped curve (sigmoid curve). It is also known as squashing function, since the permissible range of an output value is usually squashed into a finite boundary. Sigmoidal functions are usually continuous and differentiable for all real-valued inputs, which makes them ideal for NNs, especially when trained using the BP algorithm.

The most common form of a sigmoidal function is logistic (*logsig*). A logistic function typically has an output range of [0, 1], and can be defined as in Eq. (1):

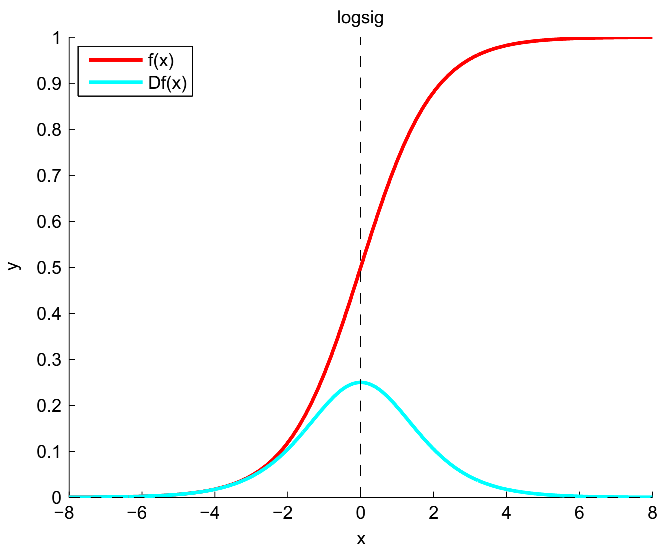


Fig. 1. Activation and gradient curves of the logistic function.

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

where x is the input. Fig. 1 illustrates the logistic function.

A hyperbolic tangent (*tanh*) function is a ratio between hyperbolic sine and cosine functions with respect to the input x:

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{2}$$

In comparison to the logistic function, hyperbolic tangent function has a wider output range of $[-1, 1]$, with its curve midpoint at the origin. As a result, it has more gradual gradients than the logistic function, which is more preferable in NN learning. Also, it has odd symmetricity about the origin, which may yield faster convergence [29].

However, hyperbolic tangent has large saturation regions on both sides of the output boundary, as the output can hardly approach to the boundary values even when the input is large due to its mathematical representation (see Fig. 2). Therefore, scaled hyperbolic tangent (*stanh*) has been proposed to alleviate this problem by manipulating its amplitude and slope [29]:

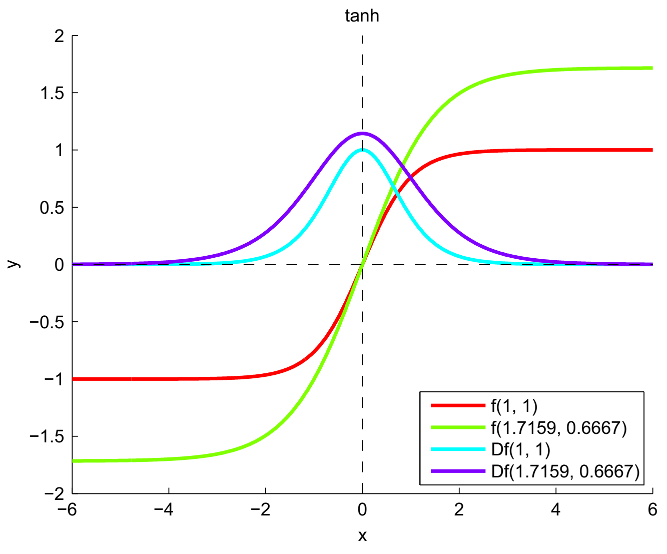


Fig. 2. Activation and gradient curves of the hyperbolic tangent function.

$$f(x) = A\tanh(Bx) \tag{3}$$

where A represents the amplitude, and B denotes the slope. The authors in [29] stated that the function is more suitable to be used with ground truth labels of 1 and -1 during the training process by setting $A = 1.7159$ and $B = \frac{2}{3}$. Also, the maximum absolute values of its second order derivative are set at $+1$ and -1, which can improve the overall convergence rate [29]. However, the authors did mention that the particular choice of parameters is merely a convenience that does not affect the result. This work aims to determine the exact parameter values (coefficients) and validate if the difference between these values brings any impact on the learning performance or not.

Many variants of the sigmoidal activation functions have been derived from the logistic and hyperbolic tangent functions by adding more nonlinearity to them [1,30–35]. However, this often comes with an additional computational cost. Near-sigmoidal functions [36] are designed based on approximations to the sigmoid curve. Common approaches include piecewise linear (PWL) approximation [27,37] and piecewise quadratic (PWQ) interpolation [23]. PWL and PWQ are among the favorite choices of approximation for hardware implementations of activation functions due to their efficient resource consumption.

Despite their popularity and wide acceptance in ANNs, sigmoidal functions can suffer badly from gradient diffusion problem. Gradient diffusion, also known as vanishing gradient, is the training condition when the proportion of error gradients being propagated backward decreases over a number of layers [10]. This occurs mainly due to the saturation problem of an activation function, where changes on the neurons' outputs can be hardly observed for any inputs near to or within a saturation region [38]. In the case of a sigmoidal function, the saturation region lies on both ends of the sigmoid curve. The nearer an input is to the saturation region, the higher the tendency of its output to stay within the corresponding region. This can be problematic, since the smaller gradient is produced, and this gradient is continuously reduced when being propagated from layer to layer. Small gradients can hurt overall training performance, as most learning algorithms rely on these gradients to perform optimization and parameter tuning. Careful initialization of weights, especially in hidden layers is essential in order to prevent significant saturation problem during the early training stages.

2.2. Non-sigmoidal functions

Non-sigmoidal functions refer to any mathematical functions that produce outputs other than the S-shaped curves. Their designs are mainly motivated by the drawbacks of a sigmoidal function, and are often influenced by their targeted applications. A typical example is a linear function which can be represented by straight lines. A neuron with a standard linear function is essentially a neuron without any activation functions. A ramp function is a non-negative real-valued function, with zero output value for any inputs less than zero. This function has not been widely used in ANNs in the early years, until researchers discovered its true potential in large ANNs [26]. Rectified linear unit (ReLU) was proposed based on the ramp function, which is defined as:

$$f(x) = \max(0, x) \tag{4}$$

There are arguments than the ReLU (*relu*) is more biologically plausible than the standard sigmoidal functions, since it is often rare to have the cortical neurons in their maximum saturation regime [26]. This is often proven by the state-of-the-art performances of neural networks (especially CNNs) using ReLU as their activation functions [39].

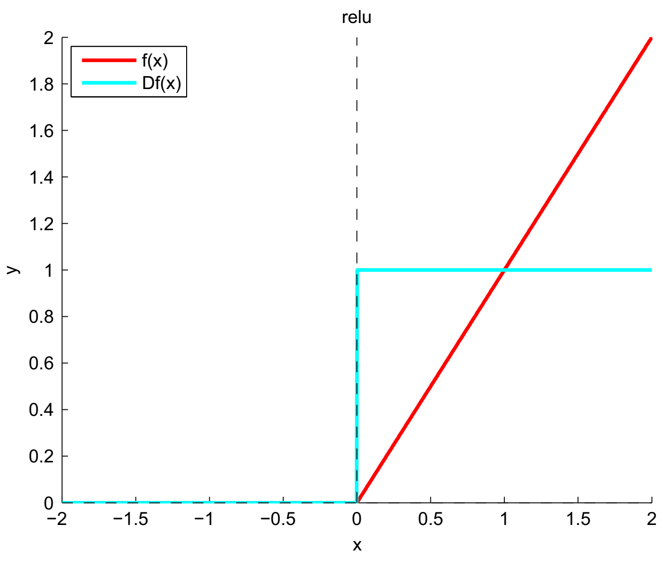


Fig. 3. Activation and gradient curves of the ReLU function.

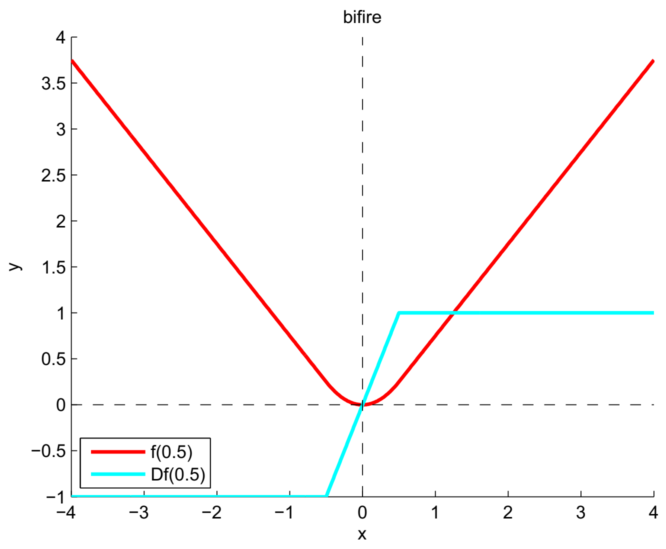


Fig. 4. Activation and gradient curves of the bi-firing function.

In general, the ReLU function is compute efficient, since it requires only a simple comparison between two values. ReLU has a sparse activation probability that creates sparse representation of data, which is beneficial for classification purposes. Gradient backward propagation is simple and ReLU does not suffer from the gradient diffusion problem as much as sigmoidal functions do. It is differentiable at any point in the function domain except at the origin (piecewise differentiable).

Some modifications have been proposed to improve the conditioning of the ReLU function. For instance, ReLU produces only zero gradient for any negative input values (see Fig. 3). Therefore, a leaky version of ReLU (*lrelu*) function has been proposed to allow a small, non-zero gradient value whenever the neuron is inactive [40]:

$$f(x) = \begin{cases} x & x > 0 \\ 0.01x & \textit{otherwise} \end{cases} \tag{5}$$

Overall, all variants of the ReLU function are susceptible to training instability due to their unbounded outputs, which will be discussed in detail later in this section.

A new type of non-sigmoidal activation function, i.e. the bi-firing (*bifire*) function was proposed in [10] to alleviate the gradient diffusion problem. It is basically an even symmetric and piecewise function as given in Eq. (6):

$$f(x) = \begin{cases} -x - \dfrac{A}{2} & x < -A \\ x - \dfrac{A}{2} & x > A \\ \dfrac{x^2}{2A} & \textit{otherwise} \end{cases} \tag{6}$$

where A is a smoothing parameter that determines the convex-shaped curve's sharpness when the input is approaching to zero. Fig. 4 depicts the bi-firing function and its derivative.

Bi-firing activation function improves the condition for efficient error propagation by having a very small saturation region near the origin. Its computational cost is also cheaper than most sigmoidal functions that involve exponential operations. However, the authors stated that it can cause numerical problems due to its unbounded nature, thus it requires a weight penalty mechanism to control the maximum output that it produces [10]. This induces more computational complexity to the training process, especially when the NN size is larger and more data samples are involved. In addition, it still requires more computations than a ReLU activation function.

Many recent state-of-the-art works focused on building larger and deeper NN models using ReLU as the main activation function [4,8,11,12,18,39,41]. These systems do not emphasize on the importance of the bounded outputs, since many NN libraries automatically detect and stabilize common numerically unstable expressions due to the unbounded values (e.g. Theano [42], Caffe [43], and cuDNN [44]). They also support high floating-point precision number representation that alleviates the numerical stability issue, thus this issue is not of the concern in these cases.

However, the output boundary issue is very important, especially in embedded system applications or dedicated hardware design where the number representation is limited, since it can bring significant impact on the computational complexity and resource consumption of the system. This explains why many previous works on the hardware design of NN models still choose common sigmoidal functions as their activation functions [45–50]. Also, there are some previous works reported on the numerical instability issue due to the unbounded nature of the ReLU function [18,51,52].

3. Proposed activation functions

The generalization performance and training stability of an NN model are heavily dependent on the choice of activation function. This work aims to improve the existing activation functions for DNN models, particularly CNNs. Several existing non-sigmoidal activation functions (i.e. ReLU, leaky ReLU and bi-firing) are chosen based on their reported performances in the previous works [10,26,40].

Realizing the importance of bounding the output of activation function for better training stability (as discussed in Section 2), a new set of activation functions are defined from these functions, with UAT as the underlying theory (especially the second property) to support the idea of having a bounded output range. This is done by alleviating the numerical instability problem while still retaining their strengths. In addition, this work also proposes a new set of coefficients for the scaled hyperbolic tangent function that matches the desired characteristics as discussed in [29].

3.1. Bounded ReLU activation function

The conventional ReLU function has unbounded outputs for non-negative inputs. According to the UAT, a function should be bounded within a range of inputs. Therefore, an upper boundary condition is added to Eq. (4) to produce a bounded version of the ReLU function:

$$f(x) = \min\left(\max(0, x), A\right) = \begin{cases} 0 & x \le 0 \\ x & 0 < x \le A \\ A & x > A \end{cases} \tag{7}$$

where A defines the maximum output value the function can produce. This only requires an additional minimum operator, and has an additional interesting effect of creating a sigmoidal-like activation pattern.

3.2. Bounded leaky ReLU activation function

A leaky ReLU function has non-zero gradient values for all negative input values (as in Eq. (5)). Similar to the bounded ReLU function, another limiting condition is introduced as its output approaches to a positive value A. Instead of hard-limiting any output values greater than A to be equal to A, a slightly oblique line with a constant small positive gradient is added to create the odd symmetricity about the origin, and its gradient value is set to be identical to the line region where $x \le 0$:

$$f(x) = \begin{cases} 0.01x & x \le 0 \\ x & 0 < x \le A \\ 0.01x + c & x > A \end{cases} \tag{8}$$

In order to create a continuous function over all regions, c is set as $0.99A$ by solving the equation $x = 0.01x + c$ when $x{=}A$, which produces the following equation:

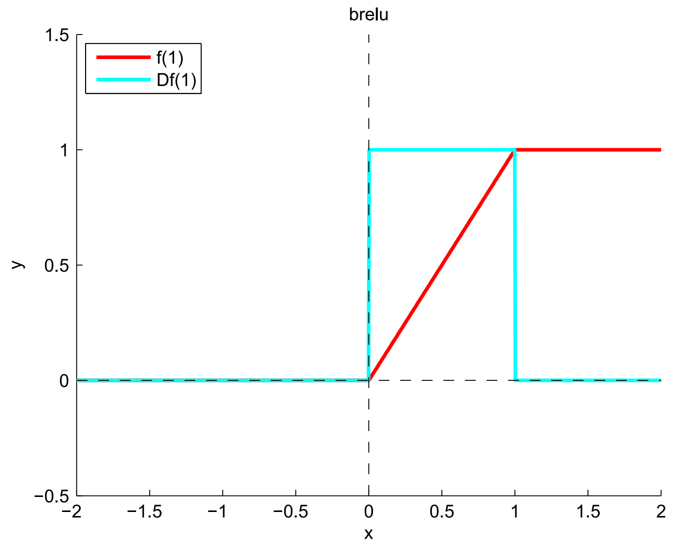
$$f(x) = \begin{cases} 0.01x & x \le 0 \\ x & 0 < x \le A \\ 0.01x + 0.99A & x > A \end{cases} \tag{9}$$

The bounded leaky ReLU function has a similar sigmoidal-like activation pattern as the bounded ReLU function, except that small gradients exist at both sides of the region where an input is not within the region of $[0, A]$. Fig. 5 illustrates the activation and gradient curves of the proposed bounded ReLU (*brelu*) and bounded leaky ReLU (*blrelu*) functions.

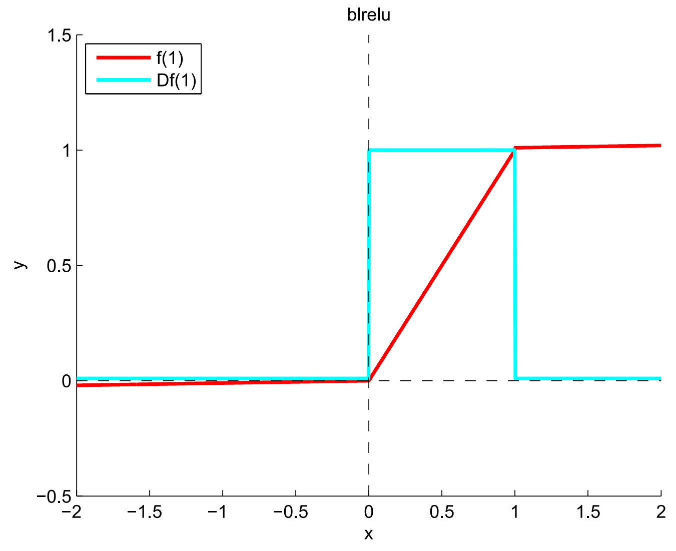
3.3. Bounded bi-firing activation function

As aforementioned previously, a bi-firing function has unbounded outputs, thus requires some weight penalty techniques during the training process to reduce the risk of training instability. This work attempts to avoid the necessity of using any weight penalty techniques when training an NN model with the bi-firing activation function. This is accomplished by restricting the maximum activation value that the function can produce by adding hard limits on both sides using an upper boundary value B to preserve the even symmetricity of the function:

$$f(x) = \begin{cases} B & x < D_L \\ -x - \dfrac{A}{2} & D_L \le x < -A \\ \dfrac{x^2}{2A} & -A \le x \le A \\ x - \dfrac{A}{2} & A < x \le D_U \\ B & x > D_U \end{cases} \tag{10}$$



(a)



(b)

Fig. 5. Activation and gradient curves of proposed activation functions: (a) bounded ReLU and (b) leaky bounded ReLU.

According to the UAT, an activation function should be continuous. Therefore, by solving both equations $-x - \frac{A}{2} = B$ when $x = D_L$ and $x - \frac{A}{2} = B$ when $x = D_U$, the final equation is obtained as:

$$f(x) = \begin{cases} B & x < -B - \dfrac{A}{2} \\ -x - \dfrac{A}{2} & -B - \dfrac{A}{2} \le x < -A \\ \dfrac{x^2}{2A} & -A \le x \le A \\ x - \dfrac{A}{2} & A < x \le B + \dfrac{A}{2} \\ B & x > B + \dfrac{A}{2} \end{cases} \tag{11}$$

The new bounded bi-firing (*bbifire*) function has a near inverse bell-shaped activation curve which is even symmetrical about the origin (as illustrated in Fig. 6).

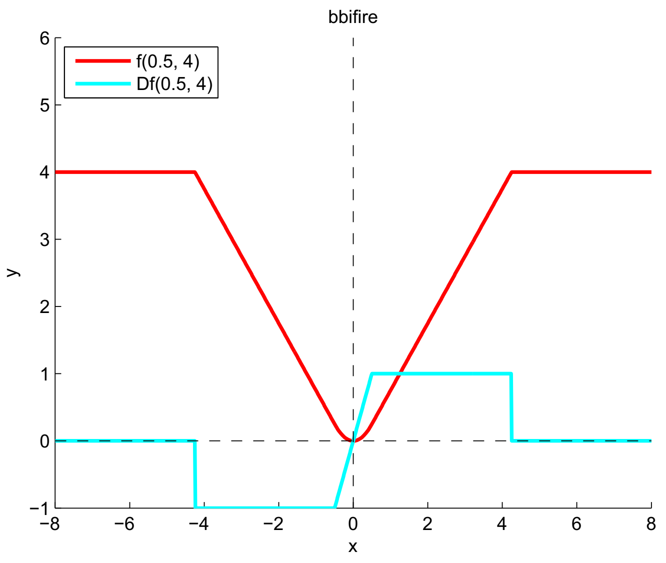


Fig. 6. Activation and gradient curves of the proposed bounded bi-firing activation function.

3.4. *Propositions based on the UAT*

The UAT provides a mathematical justification for the approximation power of an arbitrary function to emulate the exact representation of a given problem. The following propositions are established for these three new activation functions to evaluate their characteristics: .

Proposition 1. The bounded ReLU and bounded bi-firing functions have non-constant outputs for a range of inputs, while bounded leaky ReLU has non-constant outputs for all input values.

Proposition 2. All three functions are bounded within a range of input values.

Proposition 3. All three functions are continuous for all ranges of inputs.

Proposition 4. Both bounded ReLU and bounded leaky ReLU functions are monotonically increasing functions for any value of x, while the bounded bi-firing function is monotonically increasing only for $x \geq 0$.

Proposition 5. All three functions are piecewise differentiable over all input values.

Bounded ReLU function: The first derivative of the function is defined as:

$$\frac{df(x)}{dx} = \begin{cases} 1 & 0 < x \leq A \\ 0 & otherwise \end{cases} \tag{12}$$

Bounded leaky ReLU function: Its first derivative is defined as:

$$\frac{df(x)}{dx} = \begin{cases} 1 & 0 < x \leq A \\ 0.01 & otherwise \end{cases} \tag{13}$$

Bounded bi-firing function: The first derivative of the function is:

$$\frac{df(x)}{dx} = \begin{cases} -1 & -B - \frac{A}{2} \leq x < -A \\ \frac{x^2}{2A} & -A \leq x \leq A \\ 1 & A < x \leq B + \frac{A}{2} \\ 0 & otherwise \end{cases} \tag{14}$$

From the Propositions 1–5, it is noted that both bounded ReLU and bounded leaky ReLU functions are non-constant, limited, continuous, monotonically increasing and piecewise differentiable for all values of input x. The bounded bi-firing function has all the above characteristics as well, with an exception of being monotonically increasing only when $x \geq 0$. It is important to note that the UAT is primarily used in this work to support the importance of having a bounded output range for an activation function rather than designing a function to satisfy every property in the UAT.

As aforementioned previously, gradient diffusion problem occurs mainly due to the large saturation region of an activation function, where smaller gradients are produced which lead to inefficient error propagation. However, the proposed bounded activation functions are designed to alleviate the problem by having constant gradient values within the active regions of the functions (as shown in Eqs. (12), (13) and (14)). This ensures that the gradient values are not suppressed when back propagating through these activation functions. Also, as the active region is dependent on the choice of the hyperparameter value for the upper output boundary, the idea is that as long as the output boundary is large enough to alleviate the vanishing gradient problem, but at the same time small enough to avoid the numerical instability problem, then the NN training will produce faster and more stable convergence.

3.5. *Better coefficient values for the scaled hyperbolic tangent function*

The coefficients for the scaled hyperbolic tangent function as proposed in [29] were decided merely due to convenience, which they claimed would not bring any impact on the learning performance of an NN model. However, this work argues otherwise that small changes in the characteristics of the function can still be affecting the NN's training convergence.

Based on the scaled hyperbolic tangent activation function proposed in [29], the desired characteristics of a good hyperbolic tangent function are:

1. $f(-1) = -1$ and $f(1) = 1$; and
2. Both $f''(-1)$ and $f''(1)$ have maximum absolute values.

The first order derivative of the activation function (Eq. (3)) is defined as:

$$\frac{df(x)}{dx} = AB\left(1 - (\tanh(Bx))^2\right) \tag{15}$$

Then its second order derivative is:

$$\frac{d^2f(x)}{dx^2} = -2AB^2\left(\tanh\left(Bx\right) - (\tanh(Bx))^3\right) \tag{16}$$

For simplicity, let $p = \frac{d^2f(x)}{dx^2}$, then p is differentiated with respect to x to find a maximum or minimum value:

$$\frac{dp}{dx} = -2AB^3\left(3\left(\tanh(Bx)\right)^4 - 4\left(\tanh(Bx)\right)^2 + 1\right) \tag{17}$$

Since $\frac{dp}{dx}$ represents the gradient curve of function p, when the gradient of a point is zero (i.e. $\frac{dp}{dx} = 0$), the function p has a local maximum or minimum at that point. Therefore, $\frac{dp}{dx} = 0$. Let $C = \left(\tanh(Bx)\right)^2$. By substituting C into Eq. (17) and setting $\frac{dp}{dx} = 0$ gives:

$$3C^4 - 4C^2 + 1 = 0 \tag{18}$$

By solving Eq. (18), $C = \frac{1}{3}$ or $C=1$ (invalid), i.e. $\tanh(Bx) = \sqrt{\frac{1}{3}}$ or $\tanh(Bx) = -\sqrt{\frac{1}{3}}$. The positive value for $\tanh(Bx)$ is chosen to find the maximum absolute value. By substituting the value of $\tanh(Bx)$ into Eq. (3) while setting $x=1$ and $f(x) = 1$ (based on the rationale in the previous work [29] that the output of the function becomes 1 when its input is 1, and −1 when the input is −1 as well, hence the ground truth label can be set to either 1 or −1 for simplicity purposes) gives:

$$A = 1.7321 \textit{ and } B = 0.6585 \tag{19}$$

Finally, the definition of a better scaled hyperbolic tangent function becomes:

$$f(x) = 1.7321 \tanh(0.6585x) \tag{20}$$

Fig. 7 illustrates the activation and gradient curves for the hyperbolic tangent function with different sets of coefficients A and B.

4. Experimental design

The activation functions proposed in this work are evaluated on several case studies that include:

1. Classification of basic handwritten digits using the MNIST database;
2. Classification of complex handwritten digits using the *mnist-rot-bg-img* database; and
3. Face recognition using the AR Purdue database.

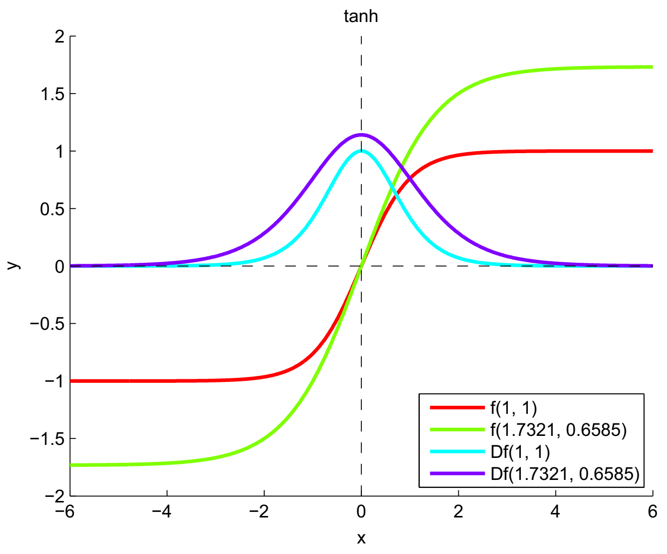


Fig. 7. Activation and gradient curves of the hyperbolic tangent function with different sets of coefficients.

All experiments are carried out using the identical MLP and CNN models for fair comparisons and better understanding on how well the proposed activation functions perform.

4.1. Dataset preparation and partitioning

The dataset in the first case study comes from the Mixed National Institute of Standards and Technology (MNIST) database which consists of 70,000 28 × 28 pixel 8-bit grayscale images of handwritten digits. The default data partitioning scheme is applied, which results in 60,000 training and 10,000 testing samples respectively. This dataset can be considered as a moderately large one. The only preprocessing techniques to be applied on the images is *z*-score normalization, which requires the computation of the mean pixel value of an input image:

$$\overline{X} = \frac{\sum_R \sum_C x_{rc}}{R \times C} \tag{21}$$

where x_{rc} is the value of pixel (r, c) in the image, R and C are the height and width of the image, respectively. Then the standard deviation is:

$$\sigma_X = \sqrt{\frac{\sum_R \sum_C \left(x_{rc} - \overline{X}\right)^2}{R \times C - 1}} \tag{22}$$

and finally the normalized image pixel value is computed as:

$$x_{rc_{norm}} = \frac{x_{rc} - \overline{X}}{\sigma_X} \tag{23}$$

These samples are flattened to single-dimensional vectors only in experiments involving the MLP models. Fig. 8 illustrates some examples of the MNIST handwritten digit images.

This MNIST dataset is chosen as the primary dataset to be evaluated in this work, since it is often the standard dataset used for the benchmarking of new algorithms (NN models and learning algorithms). In addition, it contains tens of thousands of samples that allow for realistic benchmarking of the proposed activation functions as compared to small and simple databases.

In the second case study, the *mnist-rot-bg-img* database is used. It also comes from MNIST. It is created by selecting the original handwritten digit images randomly to create complex perturbations [53]. The images are rotated randomly between 0 and 2π radians, and random image patches with high pixel variance are added to the digits as the backgrounds. This results in a total of 12,000 training and 50,000 testing samples respectively. The complex perturbations on the handwritten digit images makes them hard to be trained and classified correctly. Also, the smaller ratio of the total training to testing samples serves as a difficult challenge for the NN models to learn and generalize well on the unseen samples. The same z-score normalization is applied to all the samples. Fig. 9 shows some examples of the *mnist-rot-bg-img* images. This dataset constitutes a complex, moderately large classification problem.

In the face recognition case study, we use the AR Purdue face database which consists of over 4000 RGB face images of 126 subjects. This work uses the cropped version of the AR Purdue face database that contains a total of 100 subjects with 26 165 × 120 color images per subject [54]. There are variations in these face images, which include different facial expressions (neutral, simile, anger, scream), partial occlusions (with sunglasses or scarf), and taken under different illumination conditions (left light on, right

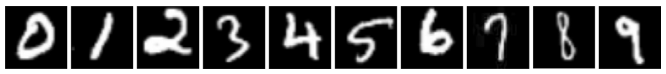


Fig. 8. Examples of the MNIST handwritten digit images.

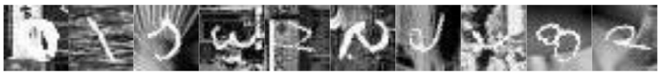


Fig. 9. Examples of the handwritten digit images in the *mnist-rot-bg-img* database.

light on, both sides lights on). Fig. 10 depicts the possible variations of the facial images from a single subject. This creates a challenging dataset which is ideal to evaluate the performance of the proposed activation functions in this paper.

The database is partitioned by having 20 out of 26 total images per subject for training, and the remaining 6 images for testing. This results in a total of 2000 training and 600 testing samples. All images are converted from RGB to grayscale color space, and are resized to 56×46. Normalizations are performed on these images by applying the z-score normalization method. No other pre-processing techniques (e.g. histogram equalization, affine transformations) or data augmentation is performed on all the aforementioned datasets.

4.2. Neural network models used in the experiments

NN models applied in our experimental work include a multi-layer perceptron (MLP) and two convolutional neural network (labeled CNN1 and CNN2) models. The MLP is used for benchmarking with previous works, while the CNNs are used to evaluate the learning performance of the NN models with the proposed activation functions for the rest of the experiments.

The MLP model is implemented based on the work in [10]. Table 1 shows the details of the seven-layered MLP model, which is used for the benchmarking of the activation functions on the *mnist-rot-bg-img* dataset.

The CNN1 model consists of seven layers in total, with alternating layers of convolutions and max-pooling in the feature extraction stage, followed by a fully-connected layer and a softmax layer for the classification purpose unless otherwise stated. Table 2 shows the details of CNN1 model, which is depicted as in Fig. 11. This model is applied in the experiments using the MNIST and *mnist-rot-bg-img* datasets.

The CNN2 model is derived from the work reported for face verification using the AR Purdue dataset [55]. It has the same total number of feature maps per layer as the CNN1 model (except for the fully-connected and softmax layers). The architectural details of the CNN2 model are tabulated in Table 3.

All convolutional layers perform convolutions with single step-sized correlation filtering operation [56]. Meanwhile, the max-pooling layers perform subsampling by selecting the maximum values within a window as the outputs. A fully-connected layer works similar to a hidden layer of an MLP. The outputs of all layers are passed through an activation function, except for the max-pooling and softmax layers. In terms of the free parameters, all convolutional layers have shared weights and biases, which result in fewer parameters [57]. No weights or biases are included in the max-pooling layers for simpler and faster computations.

Table 1
The MLP model for the experiments using the *mnist-rot-bg-img* dataset.

Layer	Type	Total neurons	Activation function	Free parameters
N0	Input	784	No	0
F1	Fully-connected	500	Yes	392,500
F2	Fully-connected	500	Yes	250,500
F3	Fully-connected	500	Yes	250,500
F4	Fully-connected	500	Yes	250,500
F5	Fully-connected	10	No	5010
X6	Softmax	10	No	0
Total				1,149,010

Table 2
The CNN1 model for the experiments using the MNIST and *mnist-rot-bg-img* datasets.

Layer	Dimension	Operation	Activation function	Free parameters
N0	1@28×28	–	No	0
C1	4@24×24	5×5 convolution	Yes	104
S1	4@12×12	2×2 max-pooling	No	0
C2	16@8×8	5×5 convolution	Yes	1616
S2	16@4×4	2×2 max-pooling	No	0
C3	128@1×1	4×4 convolution	Yes	32,896
F4	10@1×1	Full connection	Yes	1290
X5	10@1×1	Softmax	No	0
Total				35,906

4.3. Training methodology

In this work, a training procedure consists of 3 training repetitions for each experiment. An NN model is trained for 50 epochs for each repetition unless otherwise stated, to follow the settings of the previous work [10] for the benchmarking purposes (i.e. 100 epochs). At the beginning of every training epoch, the training samples are shuffled to ensure that the NN model can learn robustly and produces better learning performance.

Every new training repetition starts with the weight initialization process. Normalized weight initialization is performed throughout all experiments to generate the initial weights [58].

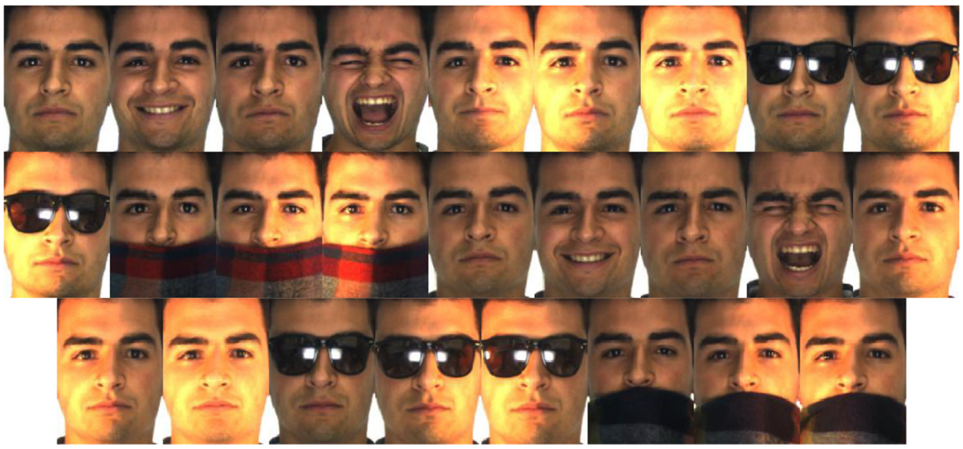


Fig. 10. The face images of a single subject in the AR Purdue face database.

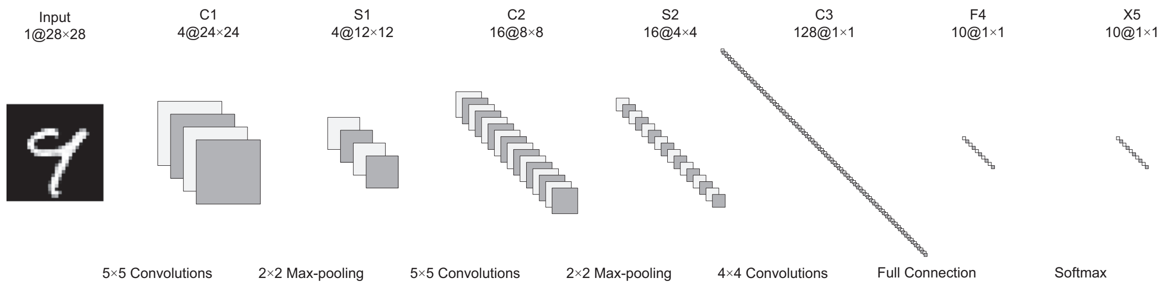


Fig. 11. The baseline CNN1 model for handwritten digit classification using the MNIST and *mnist-rot-bg-img* datasets.

Table 3
The CNN2 model for the experiments using the AR Purdue dataset.

Layer	Dimension	Operation	Activation function	Free parameters
N0	$1@56 \times 46$	–	No	0
C1	$4@50 \times 40$	7×7 convolution	Yes	200
S1	$4@25 \times 20$	2×2 max-pooling	No	0
C2	$16@20 \times 15$	6×6 convolution	Yes	2320
S2	$16@5 \times 5$	4×3 max-pooling	No	0
C3	$128@1 \times 1$	5×5 convolution	Yes	51,328
F4	$100@1 \times 1$	Full connection	Yes	12,900
X5	$100@1 \times 1$	Softmax	No	0
Total				66,748

For the normalized weight initialization, it takes into account both a neuron's fan-in $\mathcal{F}^{(l-1)}$ and fan-out $\mathcal{F}^{(l+1)}$, which can be expressed as in Eq. (24):

$$W_{ji}^{(l)} \sim U\left[-\frac{\sqrt{6}}{\sqrt{\mathcal{F}^{(l-1)}+\mathcal{F}^{(l+1)}}}, \frac{\sqrt{6}}{\sqrt{\mathcal{F}^{(l-1)}+\mathcal{F}^{(l+1)}}}\right] \tag{24}$$

This initialization method does not require any additional hyperparameter and each weight is initialized individually based on the model structure, which tend to produce better learning performance.

As for the learning algorithm, the L-SDLM algorithm [59] is applied in all training procedures unless otherwise stated. L-SDLM is chosen as the default learning algorithm since it requires tuning of a single hyperparameter only, and it can converge faster than the conventional SGD algorithm. All variants of the SDLM algorithms require a small subset of the training set to be used for the Hessian estimation. In this work, 1% of the training samples are randomly selected for the Hessian estimation for all experiments.

A typical NN learning process requires a loss function that evaluates the fitness of the training outcome. In this paper, we ran all experiments using two different sets of loss functions, i.e. MSE and CE. The objective of a MSE loss function is to minimize the distance between actual and desired outputs as small as possible to achieve discrimination among two or more classes in a classification problem:

$$(E_{MSE})_m = \frac{1}{2N^{(L)}} \sum_{j \in N^{(L)}} \left(\left(Y_j^{(L)}\right)_m - (d_j)_m\right)^2 \tag{25}$$

where $\left(Y_j^{(L)}\right)_m$ is the value of j^{th} neuron at output layer L for m^{th} sample, $(d_j)_m$ is the desired (target) value of j^{th} output neuron for m^{th} sample, and $N^{(L)}$ denotes the total output neurons.

Since the calculation of the MSE loss function requires specific target values for each sample which directly correspond to an activation function's output boundary values, this can be problematic for the unbounded activation functions. Hence, the upper output boundary P_U of these functions is set empirically to some finite value to avoid any numerical overflow condition due to a very large value of P_U, i.e. $P_U \in \{4, 64\}$. This is, however exceptional for the *stanh* function, as we fix the lower and upper boundary values to be −1 and 1 respectively for all sets of coefficient values as recommended in [29].

The cross-entropy (CE) loss function, on the other hand, is applied together with the softmax layer to evaluate the fitness of the NN model [60,61]. A softmax layer evaluates the probability of an input sample to be categorized into one of the predefined classes:

$$\left(p_j^{(L)}\right)_m = \frac{\exp\left(Y_j^{(l-1)}\right)_m}{\sum_{i \in N^{(l-1)}} \exp\left(Y_i^{(l-1)}\right)_m} \tag{26}$$

where $\left(Y_j^{(l-1)}\right)_m$ is the output of j^{th} neuron in the preceding fully-connected layer $(l-1)$ for m^{th} sample, $N^{(l-1)}$ are total neuron outputs in layer $(l-1)$, and $\left(p_j^{(L)}\right)_m$ denotes the probability of j^{th} class in the softmax layer for m^{th} sample. On the other hand, the CE error is computed as

$$\left(e_j^{(L)}\right)_m = \left(p_j^{(L)}\right)_m - (d_j)_m \tag{27}$$

where $\left(e_j^{(L)}\right)_m$ is the error for the m^{th} sample, and $(d_j)_m$ is the ground truth label for the m^{th} sample:

$$(d_j)_m = \begin{cases} 1 & \textit{correct class for } m^{th} \textit{ sample} \\ 0 & \textit{otherwise} \end{cases} \tag{28}$$

The CE loss function for the m^{th} sample is defined as

$$(E_{CE})_m = -\sum_{J}\left((d_j)_m \ln\left(p_j^{(L)}\right)_m\right) \tag{29}$$

Training is performed to minimize the average loss value over all training samples:

$$E = \frac{\sum_M E_m}{M} \tag{30}$$

where M are total samples in the training set, and E_m can be an MSE or CE loss function. In this work, the softmax function is not applied when evaluating the training process using the MSE loss function. This work utilizes the winner-takes-all (WTA) mechanism [62] that compares among all output values to perform NN classification (regardless of which loss function to be applied during the training process).

In terms of the global learning rate, we run the experiments using the following values: 0.001, 0.0025, 0.005, 0.0075 and 0.01. Table 4 summarizes the hyperparameter values for some activation functions that are evaluated in this work. By referring to this table, the values of hyperparameters A for both *brelu* and *blrelu* functions are set heuristically to the powers of 2. As for the *bifire* and *bbifire* functions, the values of A are adopted from the previous

Table 4
Hyperparameters for the activation functions and the corresponding search range during the training.

Activation function	Boundary Lower ($\mathbf{P_L}$)	Boundary Upper ($\mathbf{P_U}$)	Hyperparameters A	Hyperparameters B
brelu	0	*A*	1, 2, 4, 8, 16, 32, 64	–
blrelu	0	*A*	1, 2, 4, 8, 16, 32, 64	–
bifire	0	P_U	0.01, 0.1, 0.3, 0.5	–
bbifire	0	*B*	0.01, 0.1, 0.3, 0.5	1, 4, 16
stanh	−*A*	*A*	1, 1.7159, 1.7321	0.6585, 0.6667, 1

work in [10], while the values of *B* are set heuristically to the powers of 4. Finally, the values of *A* and *B* for the *stanh* function are derived from the previous work in [29].

In theory, the upper boundary of the proposed bounded activation functions can be set according to the desired range of the number representation for the NN model, especially in the hardware design where there is limited numerical precision due to the resource constraint [41]. No weight regularization techniques are applied in all the experiments to exclude the effect of the weight penalty on the training outcomes [57], hence evaluating the sole effect of bounding an activation function on the generalization performance and training stability of the NN models. The best results are presented in the next section.

4.4. Implementation details

In this work, all training procedures, learning algorithms and NN models are developed from scratch. All codes are written in C/C++. Native compilation is performed on Ubuntu 14.04 64-bit LTS OS using G++ compiler with the highest code optimization level (O3). Real-valued data are represented by the single-precision floating data type throughout the experiments. All program codes and experiments are executed on a general purpose computing platform with an overclocked 4.5 GHz Intel Core i7 4790 K CPU.

5. Results and discussions

In this section, the experimental analysis of the proposed activation functions is discussed according to the following aspects: (a) benchmarking with previous works, (b) training efficiency, (c) classification performance, and (d) training stability. The results are reported based on the lowest MCR values within the 50 training epochs (or 100 in the case of benchmarking with the previous work [10]). As for some activation functions with hyperparameters (see Table 4), the best results are reported from all the experimental settings using different hyperparameter values for these functions.

5.1. Benchmarking with previous works

The proposed activation functions are first benchmarked using the baseline experimental setup in [10] that uses the MLP model (as described in Table 1). Their performances are evaluated in terms of the testing MCRs on the *mnist-rot-bg-img* dataset.

Table 5 summarizes the testing MCRs of the proposed activation functions together with other functions as evaluated in [10]. Both MLPs with the proposed bounded ReLU (*brelu*) and bounded leaky ReLU (*blrelu*) functions outperform the ones with the ReLU (*relu*) and leaky ReLU (*lrelu*) functions (i.e. 50.09% versus 51.98%, and 49.88% versus 50.52%, respectively).

The MLP with the proposed bounded bi-firing (*bbifire*) function

Table 5
Testing MCRs of the MLPs with different activation functions on the *mnist-rot-bg-img* dataset. The bolded functions denote the proposed functions.

Activation function	Testing MCR (%)
logsig [10]	62.64
relu [10]	51.98
brelu	**50.09**
lrelu	50.52
blrelu	**49.88**
bifire [10]	48.75
bbifire	**48.97**
tanh [10]	53.17
stanh	50.12
stanh	**49.89**

achieves a slightly higher testing MCR than the bi-firing (*bifire*) function (48.97% versus 48.75%), but with a very small difference of 0.22% only. Tuning the amplitude and slope of the hyperbolic tangent (*tanh*) function to create the scaled hyperbolic tangent (*stanh*) function [29] is indeed advantageous, as the MLP with the *stanh* function scores a much lower testing MCR (50.12%) compared to the original *tanh* function (53.17%). Moreover, applying the proposed coefficient values to the *stanh* function manages to achieve an even lower MCR of 49.89% only.

There is an important point to note though: the results obtained in this work are based on an uneven experimental setup as compared to the work in [10]. The results published in [10] were the best results obtained from multiple training attempts with different MLPs and hyperparameter values. Their training procedure incorporates the weight regularization techniques, and most probably takes ≥200 training epochs.

Meanwhile, the MLP model used in this work is fixed, and is trained for only 100 epochs with a fixed global learning rate of 0.005. No weight regularization techniques are applied during the whole training procedure. Still, MLPs with the proposed activation functions are able to perform comparably well or even better than other activation functions, which clearly demonstrates their superior performances in achieving better generalization ability of a DNN model.

5.2. Training efficiency

The proposed activation functions are evaluated with respect to their original forms in terms of the training efficiencies using both the MSE and CE loss functions. This is done by observing both the training losses and testing MCRs on the MNIST dataset.

Fig. 12(a) shows the training MSE losses for both the *relu* and *brelu* functions on the MNIST dataset. The CNN with the *brelu* function has a lower initial MSE loss (0.0068 versus 0.1634) that steadily decreases and achieves lower MSE loss (0.0019) than the *relu* function (0.0033). It is able to achieve a higher testing accuracy than the *relu* function (99.06% versus 99.03%), which is depicted in Fig. 12(b). This is due to the better control of the outputs' distribution through the additional boundary condition, which encourages the network convergence.

Training a CNN with the *brelu* function using the CE loss function results in a slower decrease of the training loss compared to the *relu* function (Fig. 12(c)). This is most probably that the CE loss function is better suited for unbounded activation functions. However, this does not affect its generalization performance on the testing set. In fact, it is able to reach far lower testing MCR than the *relu* function, i.e. 0.92% versus 1.02% (as depicted in Fig. 12(d)). Learning better on the training set does not necessarily mean a better accuracy on the unseen samples prior to the training.

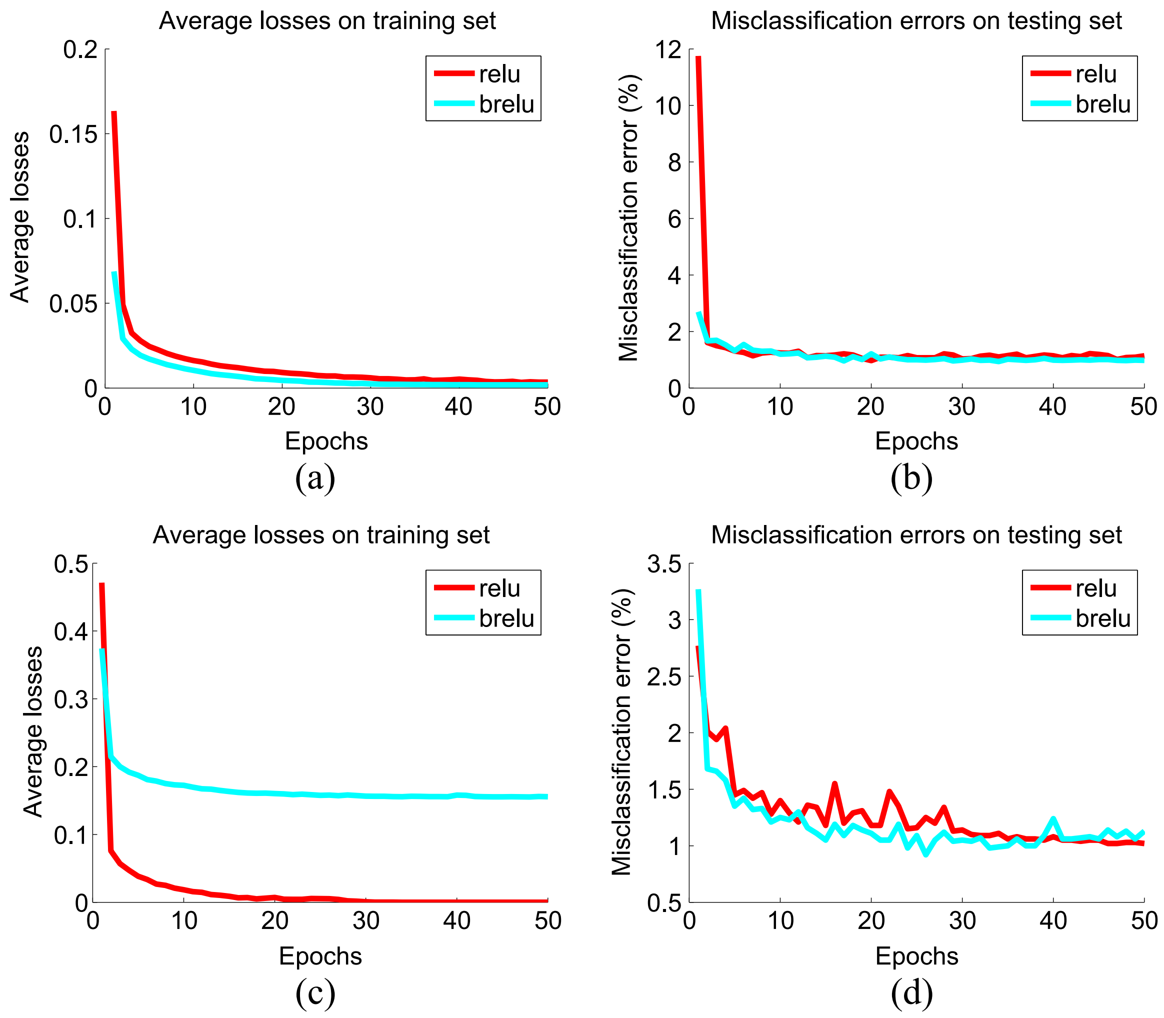


Fig. 12. Training efficiencies of the CNNs with the *relu* and *brelu* functions: (a) training losses and (b) testing MCRs using the MSE loss function; and (c) training losses and (d) testing MCRs using the CE loss function.

The results have shown a very good training convergence of a CNN with the proposed *blrelu* function than the *lrelu* function when trained using the MSE loss function (0.0001 versus 0.1553). This is clearly indicated by the two training loss curves that are clearly separated by a distance (Fig. 13(a)). The CNN model with the *blrelu* function also produces more stable MCRs over training epochs (Fig. 13(b)), and eventually achieves a testing MCR comparable to the *lrelu* function. (0.96% versus 0.95%).

A similar pattern is observed for the training outcome using the CE loss function, where the training loss of the CNN with the *blrelu* function decreases faster and reaches less than 0.0007 within 18 epochs (Fig. 13(c)). It also produces a much smoother MCR curve than the model with the *lrelu* function, and is able to achieve only 0.96% testing MCR (Fig. 13(d)) as compared to the *lrelu* function (1.06%).

Experiments on the *bifire* and the proposed *bbifire* activation functions have shown the clear advantage of having a safe output boundary condition for better training stability. The differences between the training MSE losses of these two functions can be hardly observed (Fig. 14(a)). However, by referring to Fig. 14(b), the trained CNN model with the *bbifire* function produces a more stable MCR curve and achieves the testing MCR comparable to the *bifire* function, i.e. 0.90% and 0.89%, respectively.

The positive effects of the bounded outputs are shown clearly in Fig. 14(c), where the training CE loss of the CNN with the *bbifire* function decreases steadily as the training proceeds. In contrast, it decreases initially for the CNN with the *bifire* function, but starts to increase after 30 training epochs. This clearly indicates the training instability due to the unbounded nature of the *bifire* function. Another evidence is clearly observed in Fig. 14(d), where the MCR curve of the CNN with the *bifire* function continues to fluctuate over time as more training epochs have passed. On the contrary, training of the CNN with the proposed *bifire* function makes smoother progress that leads to a very low testing MCR of 0.86% on the MNIST dataset.

As for the hyperbolic tangent (*tanh*) function, setting its output boundary values as the target values when training using the MSE loss function gives the best training loss as illustrated in Fig. 15(a). However, no direct relationship is observed between the training loss and its corresponding testing MCR (Fig. 15(b)), as the CNN with the *tanh* function using the proposed coefficients (*stanh*) is able to achieve a slightly lower testing MCR (0.99%) than the conventional *tanh* function (1.01%).

The effect of having different amplitudes and slopes for the *tanh* function is clearly depicted in Fig. 15(c), where the training on the CNNs using the *stanh* functions results in significantly lower training CE losses than the one with the *tanh* function. More importantly, the *stanh* function with the proposed coefficients outperforms the other two functions again by achieving 99.01% testing accuracy (Fig. 15(d)). The experimental results have proven the validity of the desired properties of a hyperbolic tangent activation function as proposed in [29]. Table 6 summarizes the classification

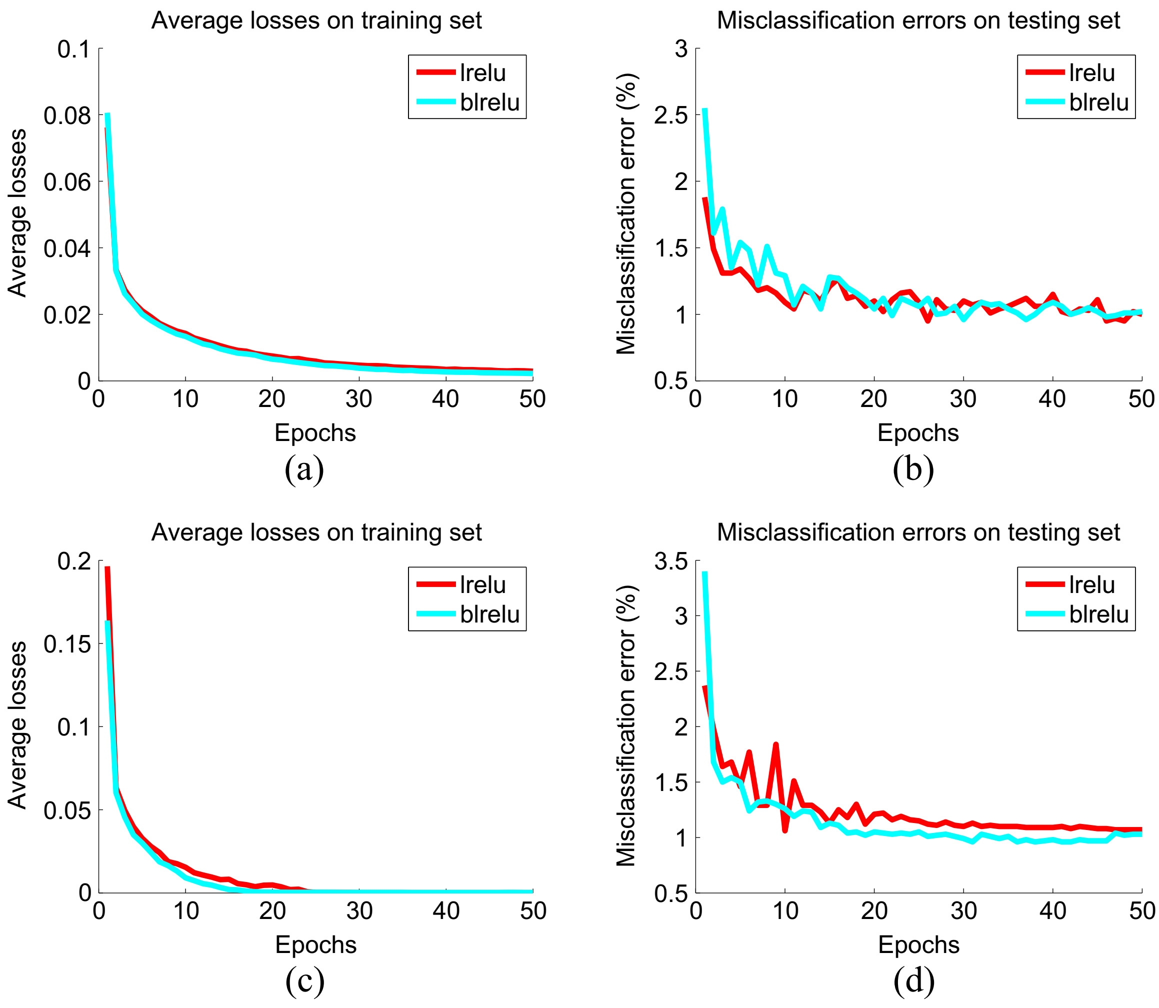


Fig. 13. Training efficiencies of the CNNs with the *lrelu* and *blrelu* functions: (a) training losses and (b) testing MCRs using the MSE loss function; and (c) training losses and (d) testing MCRs using the CE loss function.

performances of the proposed activation functions on the MNIST dataset.

Based on this table, it is observed that most of the proposed activation functions reduce the average execution time per epoch under the MSE training criterion. Having a bounded activation function allows for the target outputs to be set at its output boundary values when training using the MSE loss function. This encourages the weights to be tuned such that the outputs of the neurons are either zero (lower boundary) or a certain positive value (upper boundary that depends on the value of hyperparameter A or B). More zero output values contribute to more sparse representation of the data in the NN model, which reduces the computational time due to more operations on the zero-valued variables.

Training using CE loss function, however, requires that a softmax layer to be added to the output layer of the NN model which increases the overall computation time. The softmax layer squashes the output values of the activation function into the range of $[0, 1]$ regardless of its original output range (as shown in Eq. (26)). However, the larger the output range of an activation function, the more possibility that the softmax output probability will be suppressed to zero due to the exponential operations. As aforementioned previously, more sparse representation of the data reduces the execution time, which explains why NN models with unbounded activation functions perform faster than the proposed bounded functions under the CE training criterion.

5.3. *Classification performance*

The classification performances of the CNN models using various activation functions are also reported in terms of the training and testing MCRs on the *mnist-rot-bg-img* and AR Purdue datasets. These results indicate how an activation function will affect the generalization performance of a CNN model.

Table 7 shows the training and testing MCRs of the CNN models with various activation functions on the *mnist-rot-bg-img* dataset. Overall, the CNN models benefit more from training using the softmax function and CE loss function. Regardless of which loss function to be used, all the proposed activation functions outperform their original forms in terms of the testing MCRs. Improvements up to 6.42% (*brelu*), 9.19% (*blrelu*), 7.71% (*bbifire*), and 3.45% (*stanh*) are achieved respectively, while requiring comparable execution time as compared to their original forms (with the proposed *brelu* being the fastest, i.e. 18.59 s). Lowest testing MCRs are obtained by training the CNN models with the proposed *bbifire* and *brelu* functions, i.e. 23.09% and 22.59% when trained using the MSE and CE loss functions, respectively.

The results in Table 8 further demonstrate the superiority of the proposed activation functions by achieving significant improvements in terms of the testing MCRs, i.e. 16.12% (*brelu*), 74.99% (*blrelu*), 9.00% (*bbifire*), and 44.72% (*stanh*) respectively. These improvements are obtained without sacrificing the average execution time, with even the proposed *brelu* completing a training

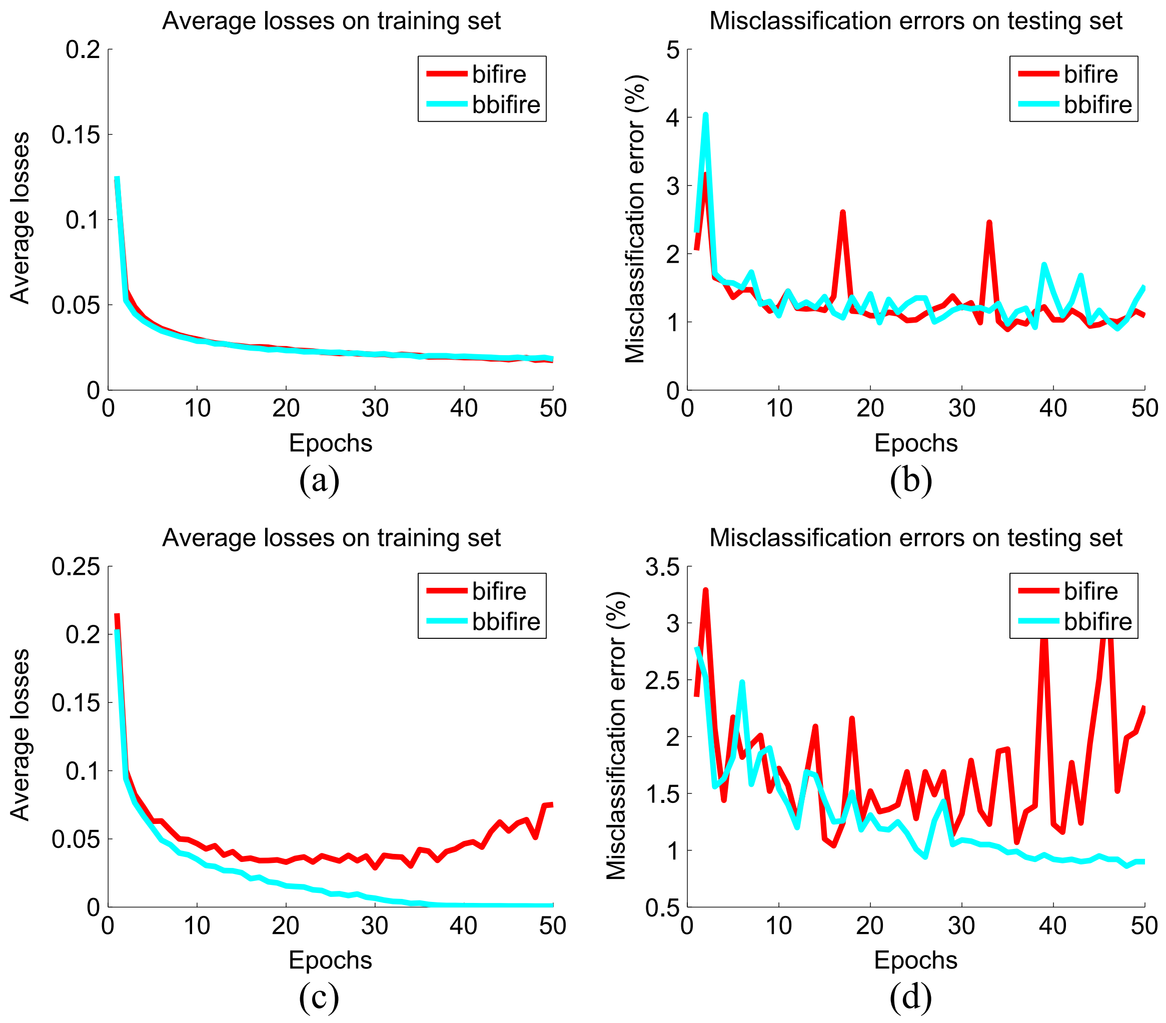


Fig. 14. Training efficiencies of the CNNs with the *bifire* and *bbifire* functions: (a) training losses and (b) testing MCRs using the MSE loss function; and (c) training losses and (d) testing MCRs using the CE loss function.

epoch within the shortest time (7.84 s). Comparisons among various activation functions reveal that the lowest testing MCRs are achieved by training the CNN models with the proposed *stanh* and *bbifire* functions using the MSE and CE functions, respectively (i.e. 2.67% and 6.67%). These results indicate that the MSE loss function is more suitable to be applied to this particular dataset.

By comparing the results in Table 6–8, we note that the performance differences between different variants of the *stanh* function are relatively small as compared to the bounded activation functions (except on the AR Purdue dataset). This is due to the fact that the value differences between the original and proposed coefficients for the *stanh* function are very small (i.e. 1.7159 and 0.6667 versus 1.7321 and 0.6585). The aim here is to prove the importance of having the correct coefficients for the *stanh* function based on the properties as defined in [29] instead of proposing a new sigmoidal function that outperforms it significantly.

There is an important thing to note though, that training the CNN model with the *bifire* function using the MSE loss function encounters the numerical instability problem consistently even with different sets of hyperparameter values. This signifies clearly the impact of having an unbounded activation function to the overall training stability, which serves as the main motivation of studying the effect of various activation functions on the training stability (in terms of the numerical stability), particularly on deep-layered NN models.

5.4. *Training stability*

It is important to evaluate the training stability during a training process based on its numerical stability, since it dictates whether a CNN model will be successfully trained or not. A numerical instability problem occurs whenever a “not a number” (NaN) or infinity value is detected, or a sudden drastic increase in the instantaneous training loss (output of the MSE or CE loss function) to a very large value is observed.

The probability of a numerical instability problem is evaluated by calculating the proportion of the experimental settings for an activation function that result in the numerical instability during an NN training. A training using a specific experimental setting is perceived to encounter the instability problem whenever the numerical instability occurs during at least a single training repetition.

By referring to Table 9, training of the CNN models with the proposed bounded activation functions reduces the probability of numerical instability significantly as opposed to the ones with the unbounded functions. Analysis of the results reveals big improvements of 78.58% (both *brelu* and *blrelu*) and 71.93% (*b*bifire) on the MNIST dataset. This justifies the importance of manipulating the output boundary values for a more stable training process. Note that the *stanh* function with the proposed coefficients is not evaluated here, since its small bounded range guarantees the numerical stability in most cases.

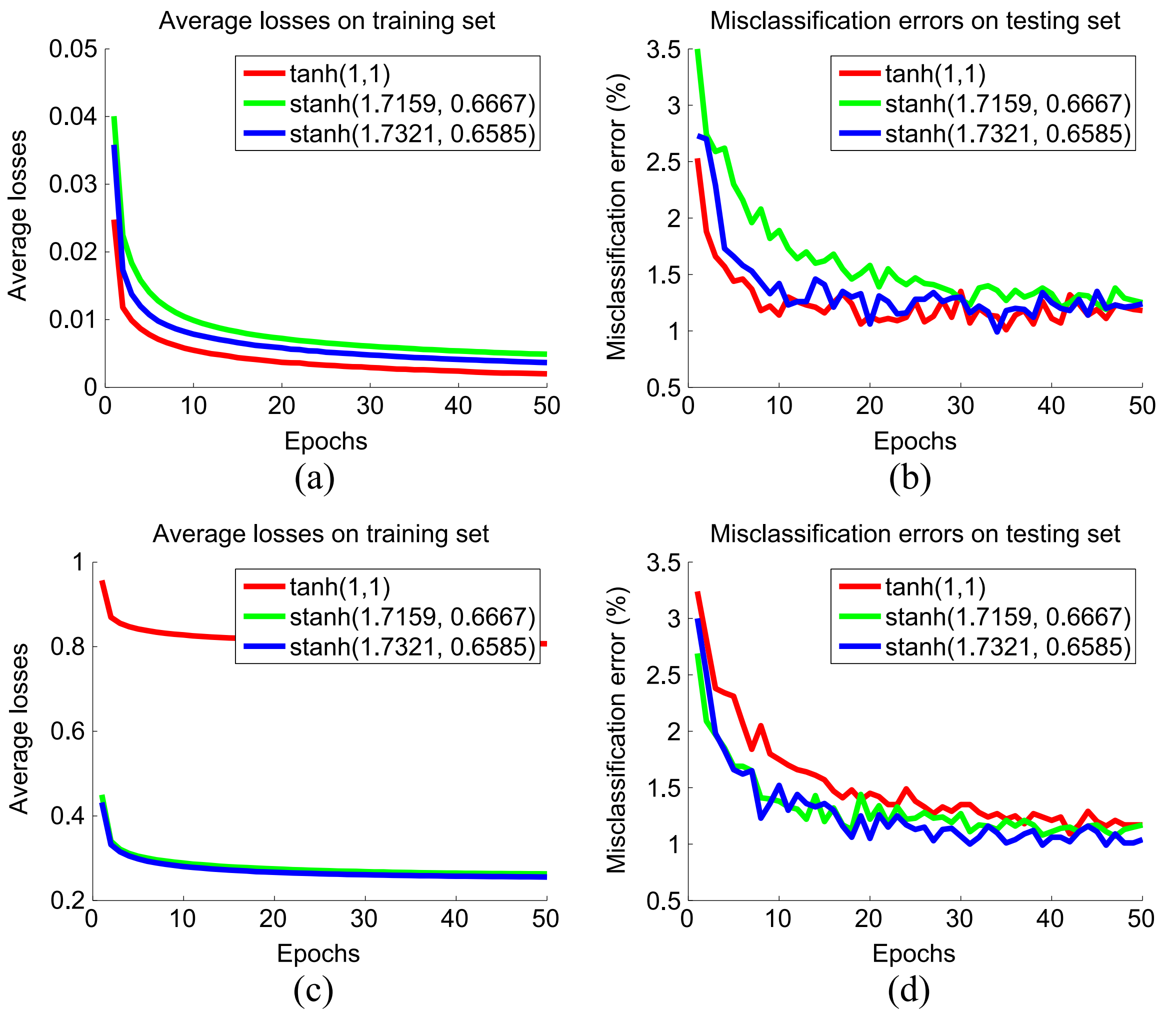


Fig. 15. Training efficiencies of the CNNs with the *tanh* function with different sets of coefficient values: (a) training losses and (b) testing MCRs using the MSE loss function; and (c) training losses and (d) testing MCRs using the CE loss function.

Table 6
Results of the CNN models using different activation functions on MNIST dataset. The bolded functions denote the proposed functions.

Activation function	WTA+MSE			Softmax+CE		
	MCR (%)		Time	MCR (%)		Time
	Training	Testing	(s)	Training	Testing	(s)
relu	0.14	**0.97**	49.97	0.00	1.02	24.58
brelu	**0.18**	**0.94**	**32.60**	**0.10**	**0.92**	**40.61**
lrelu	0.07	0.95	52.48	0.01	1.06	26.50
blrelu	**0.11**	**0.96**	**44.08**	**0.01**	**0.96**	**48.48**
bifire	0.71	0.89	58.10	0.79	1.04	33.14
bbifire	**0.81**	**0.90**	**57.17**	**0.01**	**0.86**	**52.04**
tanh	0.19	1.01	62.01	0.57	1.09	58.80
stanh	0.87	1.19	62.84	0.29	1.07	59.19
stanh	**0.56**	**0.99**	**63.33**	**0.29**	**0.99**	**61.86**

Table 7
Results of the CNN models using different activation functions on the *mnist-rot-bg-img* dataset. The bolded functions denote the proposed functions.

Activation function	WTA+MSE			Softmax+CE		
	MCR (%)		Time	MCR (%)		Time
	Training	Testing	(s)	Training	Testing	(s)
relu	3.53	25.22	21.94	5.72	22.97	19.91
brelu	**4.65**	**23.60**	**18.59**	**2.72**	**22.59**	**21.12**
lrelu	2.85	25.70	21.92	0.37	25.79	21.43
blrelu	**9.37**	**24.44**	**22.10**	**0.10**	**23.42**	**20.50**
bifire	15.58	25.02	21.67	13.82	25.61	22.10
bbifire	**12.06**	**23.09**	**22.93**	**3.17**	**24.05**	**22.56**
tanh	8.53	31.89	28.38	8.06	32.63	29.32
stanh	9.73	29.99	28.28	5.18	29.94	29.07
stanh	**10.43**	**29.48**	**28.14**	**4.98**	**29.02**	**29.31**

The proposed bounded activation functions are also evaluated on the *mnist-rot-bg-img* and AR Purdue datasets, and the results are tabulated as in Table 10,11. Both results exhibit the similar pattern, where CNN models with either *relu* or *brelu* function does not suffer from the numerical instability problem during the training process. However, significant improvements in the numerical instability problem are observed from the experiments using the proposed *blrelu* function, where training of CNN models using the CE loss function does not encounter any numerical instability condition with the proposed *blrelu* function instead of *lrelu*.

Experiments on the bi-firing function even clearly indicate the necessity of bounding the output of an activation function, where replacing the original *bifire* function with the proposed *bbifire* function eliminates any numerical instability problem during the training process on both datasets, regardless of which type of the

Table 8
Results of the CNN models using different activation functions on the AR Purdue dataset. The bolded functions denote the proposed functions.

Activation function	WTA+MSE			Softmax+CE		
	MCR (%)		Time	MCR (%)		Time
	Training	Testing	(s)	Training	Testing	(s)
relu	16.50	**20.50**	8.93	11.85	19.67	8.47
brelu	**12.90**	**17.33**	**7.84**	**8.95**	**16.50**	**8.19**
lrelu	5.50	16.67	9.06	0.05	7.67	8.65
blrelu	**0.90**	**4.17**	**8.90**	**0.15**	**7.00**	**8.40**
bifire	OF		N/A	0.00	7.33	8.85
bbifire	**3.70**	**7.17**	**9.22**	**0.10**	**6.67**	**9.45**
tanh	1.45	**10.00**	10.38	0.25	10.83	10.36
stanh	0.25	4.83	9.85	0.00	8.33	10.25
stanh	**0.00**	**2.67**	**9.87**	**0.00**	**7.00**	**10.27**

Table 9
Probability of numerical instability for the CNN models using different activation functions on the MNIST dataset. The bolded functions denote the proposed functions.

Activation function	Probability of numerical instability (%)	
	WTA+MSE	Softmax+CE
relu	0.00	40.00
brelu	**0.00**	**8.57**
lrelu	60.00	40.00
blrelu	**51.43**	**8.57**
bifire	95.00	93.33
bbifire	**26.67**	**53.33**

Table 10
Probability of numerical instability for the CNN models using different activation functions on the *mnist-rot-bg-img* dataset. The bolded functions denote the proposed functions.

Activation function	Probability of numerical instability (%)	
	WTA+MSE	Softmax+CE
relu	0.00	0.00
brelu	**0.00**	**0.00**
lrelu	60.00	60.00
blrelu	**25.71**	**0.00**
bifire	**92.50**	0.00
bbifire	**0.00**	**0.00**

Table 11
Probability of numerical instability for the CNN models using different activation functions on the AR Purdue dataset. The bolded functions denote the proposed functions.

Activation function	Probability of numerical instability (%)	
	WTA+MSE	Softmax+CE
relu	0.00	0.00
brelu	**0.00**	**0.00**
lrelu	80.00	0.00
blrelu	**31.43**	**0.00**
bifire	100.00	45.00
bbifire	**0.00**	**0.00**

loss function to be applied. This proves the importance of having a bounded activation function as defined in the UAT.

There have been some recent works on improving the training stability of DNN models, notably batch normalization [63]. This method focuses on the internal covariate shift problem (i.e. the change in the distribution of neuron inputs that leads to the difficulty of training the DNN effectively) by normalizing these neuron inputs based on the mini-batches in the training set. This results in an accelerated training process and higher training accuracy since larger learning rates can be used without deteriorating the learning outcome due to the training divergence. Also, the authors stated that the method can be applied on both sigmoidal and non-sigmoidal activation functions, hence the proposed bounded functions as well.

However, batch normalization does not address directly the impact of unbounded activation functions to the training stability of DNN models due to the numerical instability problem. It is suited for mini-batch learning algorithms only, requires more parameters to be trained per activation, and hence introduces far more computations to the training process. Nevertheless, we are positive that the combination of the proposed bounded activation functions and batch normalization can yield even better overall training outcome, since batch normalization adjusts the inputs' distributions for more effective training, while bounded activation functions alleviate the numerical instability problem by bounding the activation outputs. This corresponds to a relatively promising future work to improve the DNN training.

6. Conclusions

In this paper, we have presented three bounded activation functions for neural networks that deal with the generalization performance and training stability issues. Our proposed functions, named bounded ReLU (*brelu*), bounded leaky ReLU (*blrelu*), and bounded bi-firing (*bbifire*), are defined from the existing activation functions that excel in the DNN models [10,40,26] based on the desired properties of the UAT. We have also proposed a better set of coefficient values for the scaled hyperbolic tangent (*stanh*) function that is derived based on the rationales as discussed in the previous work [29].

To evaluate the performance of our activation functions, a series of experiments on MLP and CNN models are performed on three case studies that include classification of basic handwritten digits in MNIST database, complex handwritten digits from the *mnist-rot-bg-img* database, and face recognition using AR Purdue database. The effect of using different loss functions for the activation functions is analyzed by incorporating either MSE or CE loss function into the training process. Benchmarking with the previous work using MLP shows that the proposed activation functions perform comparably or better than their original forms, even with possibly smaller MLP model, less training epochs, and without any weight regularization techniques.

Analysis of the learning curves on the MNIST database reveals that CNN models with the proposed functions converge faster and more stable, signifying better training efficiency and with the proposed *bbifire* function achieving the lowest testing MCR (i.e. 0.86%) on the MNIST dataset. Experimental results on all the three datasets demonstrate the superiority of all the proposed activation functions, with significant improvements up to 17.31%, 9.19%, and 74.99% on MNIST, *mnist-rot-bg-img*, and AR Purdue databases respectively in terms of the testing MCRs. Training of the CNN models with the *brelu* function results in the fastest execution time per epoch on both the *mnist-rot-bg-img* and AR Purdue databases.

In terms of the training stability, bounding the output of an activation function results in a significant reduction in the probability of numerical instability problem (i.e. 78.58%) on the MNIST dataset. Experiments on the *mnist-rot-bg-img* and AR Purdue databases reveals even bigger impact of having a bounded activation function in a DNN model, as using the *bbifire* function completely eliminates the numerical instability problem. This provides a solid evidence that the bounded output of an activation function is essential to alleviate the training instability problem when training a DNN model (particularly CNN) regardless of which loss function is to be applied.

Future work of this research involves extensive analysis of the proposed activation functions for different NN models as well as case studies in pattern recognition. This can be performed together with a wide range of sigmoidal and non-sigmoidal activation functions to evaluate the performance of NN models with various activation functions on different case studies. Batch normalization is also a promising method to be applied in conjunction with the proposed activation functions to achieve higher accuracies as discussed previously.

Also, we do acknowledge the need of tuning the hyperparameters for the proposed activation functions manually despite of their superior performances over the unbounded ones. Therefore, another research direction is to incorporate the hyperparameter selection of the activation functions into the training process [64], where suitable hyperparameter values for the activation function of each neuron are tuned automatically to achieve better learning performance.

Acknowledgments

This work is supported by Universiti Teknologi Malaysia (UTM) and the Ministry of Science, Technology and Innovation of Malaysia (MOSTI) under the Science Fund Grant no. 4S116.

References

[1] S.S. Sodhi, P. Chandra, Bi-modal derivative activation function for sigmoidal feedforward networks, Neurocomputing 143 (0) (2014) 182–196, http://dx.doi.org/10.1016/j.neucom.2014.06.007.
[2] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (2) (1991) 251–257, http://dx.doi.org/10.1016/0893-6080(91)90009-T.
[3] I. Belič, Neural networks and modelling in vacuum science, Vacuum, 80(10), 2006, pp. 1107–1122. The World Energy Crisis: Some Vacuum based Solutions ⟨http://dx.doi.org/10.1016/j.vacuum.2006.02.017⟩.
[4] W. Wang, X. Yang, B. Ooi, D. Zhang, Y. Zhuang, Effective deep learning-based multi-modal retrieval, VLDB J. (2015) 1–23, http://dx.doi.org/10.1007/s00778-015-0391-4.
[5] A. Sadrmomtazi, J. Sobhani, M. Mirgozar, Modeling compressive strength of {EPS} lightweight concrete using regression, neural network and {ANFIS}, Constr. Build. Mater. 42 (2013) 205–216, http://dx.doi.org/10.1016/j.conbuildmat.2013.01.016.
[6] L. Wen, B. Yang, C. Cui, L. You, M. Zhao, Ultrasound-assisted extraction of phenolics from longan (Dimocarpus longan lour.) fruit seed with artificial neural network and their antioxidant activity, Food Anal. Methods 5 (6) (2012) 1244–1251, http://dx.doi.org/10.1007/s12161-012-9370-1.
[7] T. Desell, S. Clachar, J. Higgins, B. Wild, Evolving neural network weights for time-series prediction of general aviation flight data, in: Parallel Problem Solving from Nature, Vol. 8672 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 771–781.
[8] Y. Ioannou, D.P. Robertson, J. Shotton, R. Cipolla, A. Criminisi, Training cnns with low-rank filters for efficient image classification, CoRR abs/1511.06744.
[9] F. Mamalet, S. Roux, C. Garcia, Real-time video convolutional face finder on embedded platforms, EURASIP J. Embed. Syst. 2007 (1) (2007) 1–8, http://dx.doi.org/10.1155/2007/21724.
[10] J.-C. Li, W. Ng, D. Yeung, P. Chan, Bi-firing deep neural networks, Int. J. Mach. Learn. Cybern. 5 (1) (2014) 73–83, http://dx.doi.org/10.1007/s13042-013-0198-9.
[11] X. Liu, S. Li, M. Kan, J. Zhang, S. Wu, W. Liu, H. Han, S. Shan, X. Chen, Agenet: deeply learned regressor and classifier for robust apparent age estimation, 2015.
[12] L. Zhang, L. Lin, X. Wu, S. Ding, L. Zhang, End-to-end photo-sketch generation via fully convolutional representation learning, CoRR abs/1501.07180.
[13] P.H.O. Pinheiro, R. Collobert, Weakly supervised object segmentation with convolutional neural networks, Idiap-RR Idiap-RR-13-2014, Idiap (8 2014).
[14] J.J. Tompson, A. Jain, Y. LeCun, C. Bregler, Joint training of a convolutional network and a graphical model for human pose estimation, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, 27, Curran Associates, Inc., United States, 2014, pp. 1799–1807.
[15] I.J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V.D. Shet, Multi-digit number recognition from street view imagery using deep convolutional neural networks, CoRR abs/1312.6082.
[16] Y. Miao, M. Gowayyed, F. Metze, EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding, CoRR abs/1507.08240.
[17] A. Graves, Generating sequences with recurrent neural networks, CoRR abs/1308.0850.
[18] K. Kurach, M. Andrychowicz, I. Sutskever, Neural random-access machines, CoRR abs/1511.06392.
[19] W. Bian, X. Chen, Neural network for nonsmooth, nonconvex constrained minimization via smooth approximation, IEEE Trans. Neural Netw. Learn. Syst. 25 (3) (2014) 545–556, http://dx.doi.org/10.1109/TNNLS.2013.2278427.
[20] S. Jain, S. Singh, Low-order dominant harmonic estimation using adaptive wavelet neural network, IEEE Trans. Ind. Electron. 61 (1) (2014) 428–435, http://dx.doi.org/10.1109/TIE.2013.2242414.
[21] N. Wang, J. Melchior, L. Wiskott, Gaussian-binary restricted boltzmann machines on modeling natural image statistics, CoRR abs/1401.5900.
[22] S. Jain, S. Singh, Fast harmonic estimation of stationary and time-varying signals using ea-awnn, IEEE Trans. Instrum. Meas. 62 (2) (2013) 335–343, http://dx.doi.org/10.1109/TIM.2012.2217637.
[23] V.P. Nambiar, M.K. Hani, R. Sahnoun, M.N. Marsono, Hardware implementation of evolvable block-based neural networks utilizing a cost efficient sigmoid-like activation function, Neurocomputing 140 (2014) 228–241, http://dx.doi.org/10.1016/j.neucom.2014.03.018.
[24] M. Puheim, L. Nyulaszi, L. Madarasz, V. Gaspar, On practical constraints of approximation using neural networks on current digital computers, in: Proceedings of the 2014 18th International Conference onIntelligent Engineering Systems (INES), 2014, pp. 257–262. ⟨http://dx.doi.org/10.1109/INES.2014.6909379⟩.
[25] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366, http://dx.doi.org/10.1016/0893–6080(89)90020-8.
[26] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G.J. Gordon, D.B. Dunson (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11), Vol. 15, J. Mach. Learn. Res. Workshop and Conference Proceedings, 2011, pp. 315–323.
[27] C. Wen, X. Ma, A max-piecewise-linear neural network for function approximation, Neurocomputing 71 (4) (2008) 843–852.
[28] P. Chandra, U. Ghose, A. Sood, A non-sigmoidal activation function for feedforward artificial neural networks, in: Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1–8 ⟨http://dx.doi.org/10.1109/IJCNN.2015.7280440⟩.
[29] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 86(11), 1998, pp. 2278–2324 ⟨http://dx.doi.org/10.1109/5.726791⟩.
[30] J.A. Bonnell, Implementation of a new sigmoid function in backpropagation neural networks, Master's thesis, East Tennessee State University (8 2011).
[31] G. da, S. Gomes, T. Ludermir, L. Lima, Comparison of new activation functions in neural network for forecasting financial time series, Neural Comput. Appl. 20 (3) (2011) 417–439, http://dx.doi.org/10.1007/s00521-010-0407-3.
[32] P. Chandra, S. Sodhi, A skewed derivative activation function for sffanns, in: Recent Advances and Innovations in Engineering (ICRAIE), 2014, 2014, pp. 1–6. http://dx.doi.org/10.1109/ICRAIE.2014.6909324.
[33] P. Chandra, Y. Singh, A case for the self-adaptation of activation functions in {FFANNs}, Neurocomputing 56 (2004) 447–454, http://dx.doi.org/10.1016/j.neucom.2003.08.005.
[34] P. Chandra, Y. Singh, An activation function adapting training algorithm for sigmoidal feedforward networks, Neurocomputing 61 (2004) 429 – 437, hybrid Neurocomputing: Selected Papers from the 2nd International Conference on Hybrid Intelligent Systems. http://dx.doi.org/10.1016/j.neucom.2004.04.001.
[35] Y. Singh, P. Chandra, A class $+1$ sigmoidal activation functions for ffanns, J. Econ. Dyn. Control 28 (1) (2003) 183–187.
[36] D. van den Bout, P. Franzon, J. Paulos, T. Miller, W. Snyder, T. Nagle, W. Liu, Scalable vlsi implementations for neural networks, J. VLSI Signal Process. Syst. Signal, Image Video Technol. 1 (4) (1990) 367–385, http://dx.doi.org/10.1007/BF00929928.
[37] X. Nie J. Cao S. Fei Multistability and instability of delayed competitive neural networks with nondecreasing piecewise linear activation functions Neurocomputing 119 (2013) 281–291, http://dx.doi.org/10.1016/j.neucom.2013.03.030. (intelligent Processing Techniques for Semantic-based Image and Video Retrieval).
[38] J.Y. Yam, T.W. Chow, A weight initialization method for improving training speed in feedforward neural network, Neurocomputing 30 (1) (2000) 219–232.
[39] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[40] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proceedings of ICML, vol. 30, 2013.
[41] S.G. andAnkur Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, CoRR abs/1502.02551.
[42] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I.J. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, Y. Bengio, Theano: new features and speed improvements, CoRR abs/1211.5590.

[43] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, MM'14, ACM, New York, NY, USA, 2014, pp. 675–678. ⟨http://dx.doi.org/10.1145/2647868.2654889⟩.
[44] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, cudnn: Efficient primitives for deep learning, CoRR abs/1410.0759.
[45] M. Peemen, R. Shi, S. Lal, B. Juurlink, B. Mesman, H. Corporaal, The neuro vector engine: Flexibility to improve convolutional net efficiency for wearable vision, in: Proceedings of the 2016 Design, Automation Test in Europe Conference Exhibition (DATE), 2016, pp. 1604–1609.
[46] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, O. Temam, Shidiannao: Shifting vision processing closer to the sensor, in: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA'15, ACM, New York, NY, USA, 2015, pp. 92–104. ⟨http://dx.doi.org/10.1145/2749469.2750389⟩.
[47] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, Y. Chen, Pudiannao: A polyvalent machine learning accelerator, in: Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'15, ACM, New York, NY, USA, 2015, pp. 369–381. ⟨http://dx.doi.org/10.1145/2694344.2694358⟩.
[48] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, Dadiannao: A machine-learning supercomputer, in: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-47, IEEE Computer Society, Washington, DC, USA, 2014, pp. 609–622. ⟨http://dx.doi.org/10.1109/MICRO.2014.58⟩.
[49] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, O. Temam, Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'14, ACM, New York, NY, USA, 2014, pp. 269–284. ⟨http://dx.doi.org/10.1145/2541940.2541967⟩.
[50] T. Moreau, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, Approximate computing on programmable socs via neural acceleration, Technical report, 2014.
[51] J. Ma, R.P. Sheridan, A. Liaw, G.E. Dahl, V. Svetnik, Deep neural nets as a method for quantitative structure-activity relationships, J. Chem. Inf. Model. 55 (2) (2015) 263–274.
[52] X. Zhang, J. Trmal, D. Povey, S. Khudanpur, Improving deep neural network acoustic models using generalized maxout networks, in: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 215–219. http://dx.doi.org/10.1109/ICASSP.2014.6853589⟩.
[53] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, in: Proceedings of the 24th International Conference on Machine Learning, ICML '07, ACM, New York, NY, USA, 2007, pp. 473–480. ⟨http://dx.doi.org/10.1145/1273496.1273556⟩.
[54] A.M. Martinez, A. Kak, Pca versus lda, IEEE Trans. Pattern Anal. Mach. Intell. 23 (2) (2001) 228–233, http://dx.doi.org/10.1109/34.908974.
[55] S. Chopra, R. Hadsell, Y. LeCun, Learning a similarity metric discriminatively, with application to face verification, in: Proceeedings of the Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, 2005, pp. 539–546 . ⟨http://dx.doi.org/10.1109/CVPR.2005.202⟩.
[56] S.S. Liew, M. Khalil-Hani, A. Syafeeza, R. Bakhteri, Gender classification: a convolutional neural network approach, Turk. J. Elec. Eng. 24 (3) (2016) 1248–1264.
[57] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, M.S. Lew, Deep learning for visual understanding: a review, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.09.116.
[58] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 9 (2010) 249–256. ⟨http://dx.doi.org/10.1.1/207.2059⟩.
[59] M. Milakov, Convolutional Neural Networks in Galaxy Zoo Challenge, April 2014, pp. 1–7.
[60] Y. Dong, Y. Liu, S. Lian, Automatic age estimation based on deep learning algorithm, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.09.115.
[61] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, Deepface: Closing the gap to human-level performance in face verification, in: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1701–1708. ⟨http://dx.doi.org/10.1109/CVPR.2014.220⟩.
[62] M. Negnevitsky, Artificial Intelligence: A Guide to Intelligent Systems, 1st ed., Addison-Wesley Longman Publishing Co., Inc.,, Boston, MA, USA, 2001.
[63] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 448–456.
[64] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, CoRR abs/1502.01852.

Shan Sung Liew was born in Kuching, Sarawak, Malaysia in 1989. He received his B.Eng. (Hons.) degree in computer engineering in 2012, and his Ph.D. degree in electrical engineering in 2016 from Universiti Teknologi Malaysia. His Ph.D. research focuses on artificial neural networks, deep learning, distributed machine learning, and computer vision.

Mohamed Khalil-Hani received his Bachelor's degree in Communications Engineering from the University of Tasmania, Hobart, Australia, in 1978 and his M.Eng. in computer architecture from Florida Atlantic University, Boca Raton, USA, in 1985. He obtained his Ph.D degree from Washington State University, Pullman, USA, in 1992, specializing in digital systems and computer engineering. He is a senior member of IEEE. Currently, he is a professor in digital systems, computer architecture and FPGA system-on-chip (SoC); serving in the University of Technology, Malaysia (UTM) since 1981. His current research interests include real-time digital computing, hardware-software codesign and FPGA SoC architectures for applications in pattern recognition, image processing, neuro-intelligent hardware systems, biometrics, and quantum computing emulation. He has worked and published extensively in these research areas for more than 2 years. He has supervised and graduated more than 35 postgraduate research students (Ph.D and Masters), and is extremely dedicated to teaching and the pursuit of knowledge.

Rabia Bakhteri received her B.Eng. in Computer Engineering in 2006, her Master's degree in Electronics and Telecommunication Engineering in 2008, and her Ph.D. in Computer Engineering 2011 from Universiti Teknologi Malaysia. She is currently employed as a Graduate Faculty Member at Universiti Teknologi Malaysia. After serving as Senior Lecturer at UTM for four years, she now works at Sightline Innovation, Canada developing machine learning systems for industrial applications. Her research focus is on embedded systems and System-on-Chip (SoC) design, ESL modeling and verification, biometrics image processing, data security, and artificial neural networks.