Journal of Embedded & Digital System Design
2026, VOL. 01, NO. 1, 1–14
http://doi.org/10.5281/zenodo.18397422

A Survey on Different Strategies for Hardware Implementation of Activation Functions

Shirshendu Roy[1] and **Jisy N K**[1]

[1]Electronics and Communication Engineering, Dayananda Sagar University, Bengaluru

Abstract

Activation functions (AFs) are fundamental components of neural networks (NNs), introducing nonlinearity that enables models to learn complex representations and decision boundaries. With the growing adoption of NNs in real-time and embedded systems, there is an increasing demand for efficient hardware implementations. Implementing AFs on hardware poses significant challenges due to the presence of nonlinear operations and transcendental functions, especially exponential computations required by commonly used AFs such as sigmoid, hyperbolic tangent, and softmax. Efficient approximation and computation techniques for exponential functions (EFs) are essential to enable practical and high-performance hardware realizations of AFs.

This paper presents a comprehensive survey of techniques for implementing AFs on digital hardware. The contributions include a concise review of widely used and emerging AFs, a survey of major EF computation methods such as lookup tables, polynomial and piecewise approximations, COordinate Rotation DIgital Compute (CORDIC) based approaches, and iterative algorithms, and a detailed examination of hardware-oriented implementation strategies for AFs. The surveyed techniques are analyzed in terms of accuracy, resource utilization, latency, and power efficiency. This survey aims to provide valuable insights and design guidelines for researchers and practitioners developing efficient NN accelerators on field gate programmable array (FPGA) and other digital hardware platforms.

Keywords: Field Programmable Gate Array (FPGA), Activation Function Architectures, Exponential Function Architectures, Sigmoid Functions, Hyperbolic Tangent.

1 Introduction

Activation functions (AFs) are fundamental components of artificial NNs (ANNs) that introduce nonlinearity into the model. Mathematically, an AF maps the weighted sum of inputs and bias of a neuron to an output signal, thereby determining the activation state of the neuron. Without AFs, a NN would effectively behave as a linear model irrespective of the number of layers, significantly limiting its ability to learn complex patterns.

The output of a neuron in a feedforward ANN is given by

$$\boldsymbol{a}^{(l+1)} = \phi\Big(\mathbf{W}^{(l)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}\Big), \tag{1}$$

where $\boldsymbol{a}^{(l)}$ and $\boldsymbol{a}^{(l+1)}$ are input and output activations at layer l and $l+1$ respectively, $\phi(\cdot)$ is the AF, $\mathbf{W}$ is the weight matrix, and $\boldsymbol{b}$ is the bias vector.

AFs are employed to enable neural networks (NNs) to model complex and nonlinear relationships inherent in real-world data. By introducing nonlinearity, they allow deep networks to approximate arbitrary functions and learn hierarchical feature representations across multiple layers. Furthermore, AFs strongly influence training behavior, including gradient propagation, convergence speed, numerical stability, and susceptibility to issues such as vanishing or exploding gradients. Consequently, the selection of an appropriate AF plays a crucial role in determining network performance, training efficiency, and implementation feasibility.

AFs are used across a wide range of NN architectures and in many machine learning algorithms. They are integral to feedforward NNs, convolutional NNs (CNNs), recurrent NNs (RNNs), transformers, and spiking NNs (SNN). Real-time applications demand for efficient implementation of these networks on digital hardware like field programmable gate array (FPGA). Thus numerous research works have been carried to find efficient techniques to implement the AFs.

Major contributions of this manuscript are

1. All kinds of AFs are briefly discussed.
2. A detailed survey on different techniques to implement exponential function (EF) is presented.
3. Different techniques to implement AFs are discussed in details.

The manuscript is organized in seven sections. Section II discusses about different AFs used in NN architectures. A brief study on techniques to implement EF is discussed in Section III. Section IV discusses about different techniques to implement different AFs. Section V discusses about different estimation metrics and performance analysis of different architectures. Conclusive remarks are made in Section VI.

2 Different Activation Functions

2.1 Binary Activation Functions

Binary AFs limits the output of a node to one of two values. They also called as threshold AFs or Heaviside function. These kind of functions are used in single layer

Manuscript received 16th January 2026; accepted 27th January 2026. Date of publication 28th January 2026.
(Corresponding author: Shirshendu Roy.)
Email ID: (e-mail: shirshenduroy88@gmail.com; jisyshiva1@gmail.com).

NNs. One such function is represented as

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

Bipolar version is also possible which is expressed as

$$f(x) = \begin{cases} 1, & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{3}$$

All these kind of step functions are shown in Figure 1(a).

2.2 Linear Activation Functions

Linear AFs are also used in NNs. Linear AFs are mostly suitable in regression analysis and they are used to increase the range of values. Linear functions can be easily computed in hardware as they do not provide any non-linearity in the system.

In a linear case output $f(x)$ varies proportionately with input variable x as $f(x) = cx$ where the constant c may take any value. Similarly we may have positive or negative linear AFs. Linear AF is defined as

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{4}$$

Similarly saturated linear AF took the expression as

$$f(x) = \begin{cases} 1, & x \geq 1 \\ 0 & x < 0 \\ x & 0 \leq x < 1 \end{cases} \tag{5}$$

Symmetric saturated linear AF is defined as

$$f(x) = \begin{cases} 1, & x \geq 1 \\ -1 & x \leq -1 \\ x & -1 < x < 1 \end{cases} \tag{6}$$

Above mentioned linear functions are shown in Figure 1(b). These functions can be used in situations where neurons are linearly activated or produces linear output within a range. Linear AFs are unbounded and differentiable.

Rectified linear unit (ReLU) AF is one of the variant of linear AFs which is very popular in NNs. It is defined as

$$f(x) = max(0, x) \tag{7}$$

ReLU function actually clips the negative part of input x. If samples of x are negative then this function simply passes zero to the output. A leaky ReLU version is also possible which is defined as

$$f(x) = max(\alpha x, x) \tag{8}$$

where α is a constant. Typically $\alpha = 0.01$ is chosen to provide small gradient for negative values. Another variation is parametric ReLU (PReLU) where parameter α is calculated by back-propagation. In some scenarios, dynamic version of ReLU function also may be used. Dynamic ReLU functions take the form as

$$f(x) = max(c, x) \tag{9}$$

where the constant c may be positive or negative. The ReLU function and its variants are shown in Figure 2(a).

2.3 Non-linear Activation Functions

Binary or linear AFs are not enough to analyse complex signals using NNs with two or more than two layers. Thus different kinds of non-linear AFs are reported in literature. Few popular functions are discussed here.

2.3.1 Sigmoid Function:- Sigmoid function is governed by the following equation

$$f(x) = \frac{1}{1 + e^{-x}} = \sigma(x) \tag{10}$$

Sigmoid function maps any variable within the range $\{0, 1\}$ and mainly used in the output layer. The characteristic of this AF is shown in Figure 1(b). Sigmoid function is differentiable, smooth, and suitable for representing probabilities. Sigmoid is not centred around 0 and thus problem may be encountered in weight update. Sigmoid function suffers from vanishing gradient problem as for higher values sigmoid function saturates.

2.3.2 Hard Sigmoid Function:- Hard sigmoid function is simplified sigmoid function for range $\{0, 1\}$. This function is preferred where speed is more important compared to accuracy. The equation hard sigmoid function is

$$f(x) = max(0, min(1, 0.5 + 0.2x) \tag{11}$$

In many applications, hard-sigmoid function (shown in Figure 2(b)) is used in place of sigmoid function.

2.3.3 Hyperbolic Tangent Function:- Hyperbolic tangent function is an improved AF. The equation for tanh function is shown below

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{12}$$

This AF generally used in the hidden layer of the deep NNs and maps the input samples within the range $\{-1, 1\}$. The characteristic of this AF is shown in Figure 2(b). Compared to sigmoid, tanh has steeper gradient which helps faster convergence. Tangent function is zero value centred which helps smooth gradient update. Tangent function also have problem of vanishing gradient.

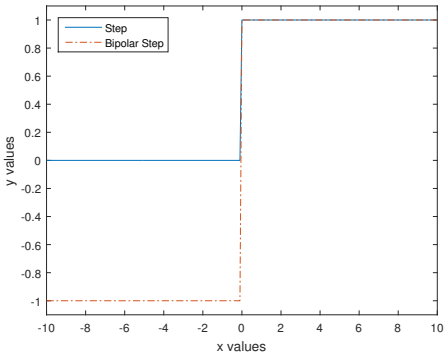
2.3.4 Bent's Identity:- Bent's identity is another AF which has the feature of ReLU function in positive half. But it does not have sharp transition which good for optimization. This function is gradient friendly and offers small non-linearity. The equation is as follows

$$f(x) = x + \frac{\sqrt{x^2 + 1} - 1}{2} \tag{13}$$

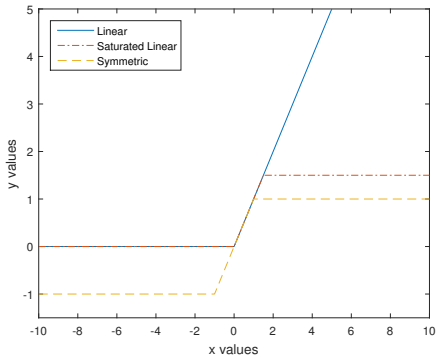
Figure 4(a) shows the characteristic for Bent's identity function.

2.3.5 ARiA-2 Activation Functions Adaptive Richard's curve weighted activation (ARiA) [1] AF is a generalized AF. It is a non-monotonous but allows a precise control over its non-monotonous convexity by varying the hyper-parameters. ARiA-2 is specialized version having two parameters α and β.

$$f(x) = x(1 + e^{-\beta x})^{-\alpha} \tag{14}$$

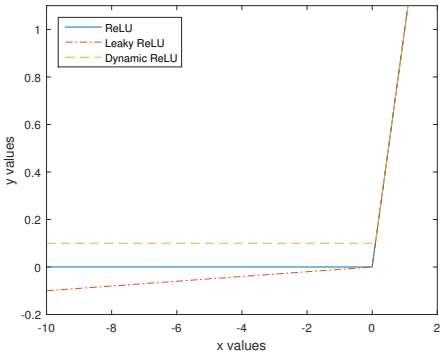


(a) Step functions.

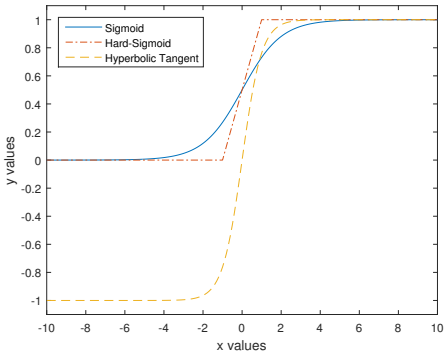


(b) Linear functions.

Figure 1: Different type of step and linear AFs.



(a) Different kind of ReLU functions.



(b) Sigmoid vs hard sigmoid vs tanh.

Figure 2: Different kind of ReLU functions, sigmoid, hard-sigmoid and tanh function.

Varying α and β different kind of functions can be generated. Figure 3 shows ARiA-2 functions generated for different parameter values. Different kind of functions generated based on different values of β is shown in Figure 3(a).

2.3.6 Softmax Function:- Softmax AF is very popular in case of multiple object classification problem in machine learning and mostly used in the output node of a NN. The equation for softmax function is

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}} \tag{15}$$

where x_i is the output from the NN for the i^{th} class for classification of k number of classes. Softmax function finds the probability of different classes.

2.3.7 Softplus Activation Function:- Softplus is smooth version of ReLU AF. Like ReLU function, softplus function also takes account of only positive values. The equation of softplus is

$$f(x) = ln(1 + e^x) \tag{16}$$

A comparison of ReLU and softplus function is shown in Figure 4(a). The slope of both the functions can be varied accordingly.

2.3.8 Softsign Activation Function:- Softsign function is an alternative to the tanh function. Tanh function converges exponentially whereas softsign function converges polynomially. The equation for softsign is as follows

$$f(x) = \frac{x}{1 + |x|} \tag{17}$$

A comparison of softsign and tanh function is shown in Figure 3(b).

2.3.9 Mish Function:- Mish, shown in Figure 4(b), is a novel smooth and non-monotonic neural AF which can be defined as

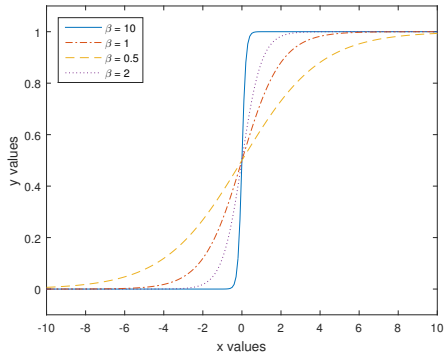
$$f(x) = x \cdot tanh(ln(1 + e^x)) \tag{18}$$

In the positive side this function behaves like ReLU. In the negative side, this function provides slightly more weights to the values which are nearer to 0.

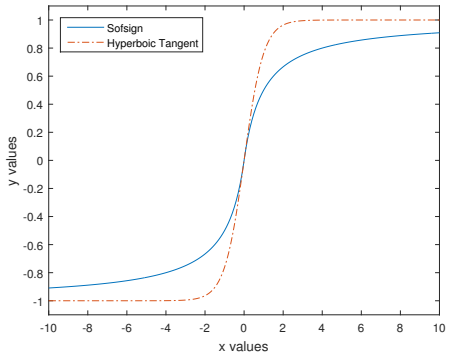
2.3.10 Swish Function:- The swish function is developed to address the disadvantages of ReLU and Sigmoid function. It dose not have sharp cut-off like ReLU and it also does not have problem of vanishing gradient. The equation for Swish function is

$$f(x) = \frac{x}{1 + e^{-x}} \tag{19}$$

Swish function is very similar to Mish function and compared with Mish function in Figure 4(b).

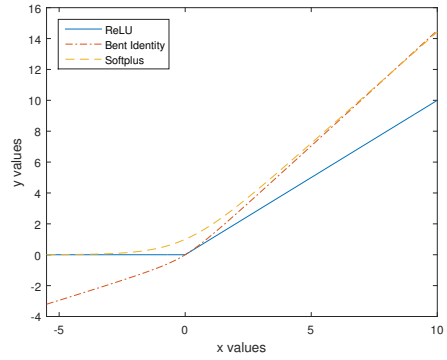


(a) ARiA-2 functions.

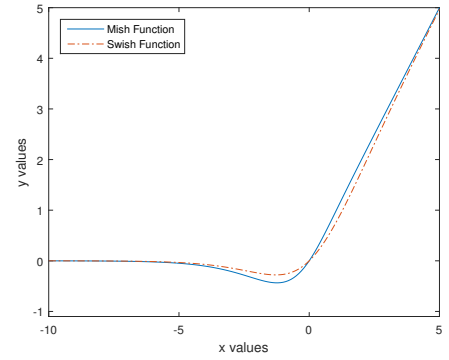


(b) Tanh vs Softsign function.

Figure 3: Different kind of functions based on ARiA-2 and comparison of tanh vs softsign function.

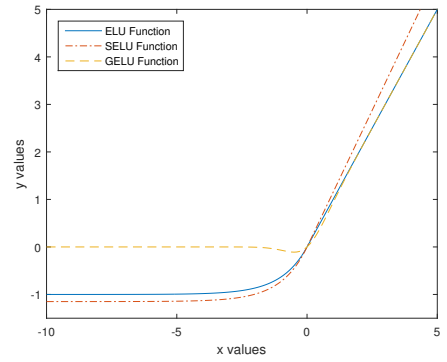


(a) Softplus, ReLU, and Bent's identity.

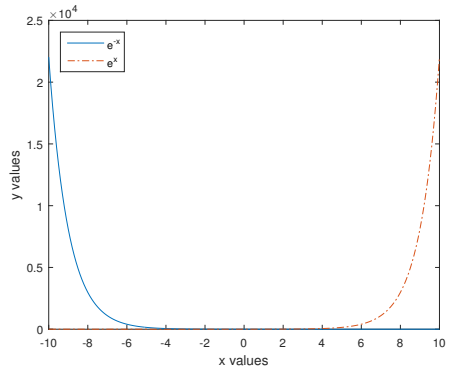


(b) Mish and Swish AFs.

Figure 4: Mish, Swish, ReLU, Bent's identity, and Softplus function.



(a) ELU, SELU, and GELU.



(b) Exponential functions.

Figure 5: Exponential based AFs and EFs.

2.3.11 Exponential Linear Unit:- Exponential linear unit (ELU) is an updated version of ReLU function where small exponential slope is provided for negative samples. The equation as follows

$$f(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x & x > 0 \end{cases} \tag{20}$$

ELU functions saves neurons from dying. Generally the parameter α is chosen as 1. Complexity of ELU increases compared to ReLU function but helps to converge faster.

Another variation of ELU function is scaled ELU (SELU) function and it is defined as

$$f(x) = \lambda \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x & x > 0 \end{cases} \tag{21}$$

here α and λ are predefined constants. Typical values are $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$. In deep NNs (DNN) this activation helps in faster convergence. A comparison of ELU and SELU function is shown in Figure 5 (a).

2.3.12 Gaussian Error Linear Unit:- Gaussian error linear unit (GELU) function designed to improve performance in DNN models by approximating Gaussian like functions. In many cases GELU function performed better compared to ReLU function in DNNs. The equation for GELU is

$$f(x) = 0.5x(1 + tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3))) \quad (22)$$

GELU provides smooth gradient flow as it is smooth and differentiable. Compared to ReLU it doesn't have sharp cut-offs. GELU function is compared with ELU and SELU function in Figure 5(a).

3 Implementation of Exponential Function

Efficient implementation of EF, shown in Figure 5(b), is very crucial to implement different AFs as EF itself many times used as AF. All efficient techniques used for EF implementation are summarized in this section.

3.1 CORDIC Based Computation of Exponential Function

COordinate Rotation DIgital Compute (CORDIC) algorithm [2] based exponential architecture [3] uses rotation mode in hyperbolic path to compute hyperbolic sine and cosine. The CORDIC equations in hyperbolic path are

$$x_{i+1} = x_i + \sigma_i y_i 2^{-i} \quad (23)$$

$$y_{i+1} = y_i + \sigma_i x_i 2^{-i} \quad (24)$$

$$z_{i+1} = z_i - \sigma_i tanh^{-1} 2^{-i} \quad (25)$$

The hyperbolic functions $sinh\theta$ and $cosh\theta$ can be computed by setting initial values as $x_0 = \frac{1}{k_h}$, $y_0 = 1$, and $z_0 = \theta$. The constant $k_h \approx 1.64676$ is for hyperbolic mode of CORDIC. The controlled bit σ_i can be computed as

$$\sigma_i = \begin{cases} 1, & \text{if } z_i \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (26)$$

CORDIC iteratively computes the hyperbolic sine and cosine. The EF is computed as

$$e^\theta = sinh\theta + cosh\theta \quad (27)$$

Accuracy of CORDIC depends on data width used for representing θ. CORDIC based technique is expensive but useful in some specialized ML applications. In [4] authors used fast convergence CORDIC (FC-CORDIC) to implement the AFs.

Many hardware implementations including CORDIC calculate EF for the range $\{0, 1\}$. But input samples may exceed this range and thus we need to extend the range. One obvious solution to extend the range is to express input sample as

$$x = p \times log_e^2 + q \quad (28)$$

Exponential function then can be computed as

$$e^x = 2^p \times e^q \quad (29)$$

It is easy to compute power of 2 in hardware but difficulty is faced to express input x in terms of p and q.

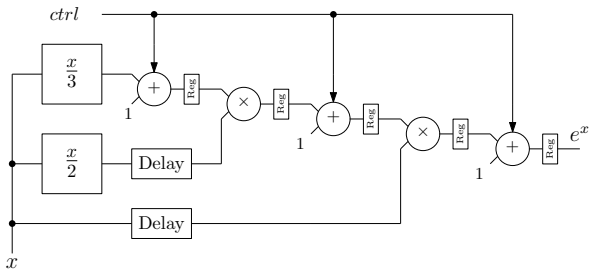


Figure 6: Taylor series based exponential.

3.2 Taylor Series Based Implementation

Many research works used Taylor series expansion method for approximating EF. Taylor series expansion of EF can be written as

$$e^x = 1 \pm x + \frac{x^2}{2!} \pm \frac{x^3}{3!} + \cdots \quad (30)$$

The alternate minus ('-') symbol is used for negative value of x. The accuracy increases if number of terms in Taylor series increases. First three terms gives an acceptable accuracy for many applications and are given as

$$e^x = 1 \pm x(1 + \frac{x}{2}(1 \pm \frac{x}{3})) \quad (31)$$

Figure 6 presents an architecture of EF based on the above equation. Authors in [5] tried to increase the accuracy by dynamically increasing the number of terms of Taylor series.

3.3 Iterative Technique for Exponential

Iterative technique [6, 7] is governed by the following two equations

$$x_{i+1} = x_i - \sigma \cdot ln(1 + \sigma \cdot s_i \cdot 2^{-i}) \quad (32)$$

$$y_{i+1} = y_i \times (1 + \sigma \cdot s_i \cdot 2^{-i}) \quad (33)$$

where i varies from 0 to n denoting total number of iterations. Initially, x_0 is the initial value and y_0 is set equal to 1. The parameter σ is included to make the equations general. $\sigma = 1$ and $\sigma = -1$ for positive and negative value of x respectively. The parameter d is used to find the values of s_i in each iterations. It is computed as $d_i = x_i - \sigma \cdot ln(1 + \sigma \cdot 2^{-i})$. If $\sigma = 1$, s is computed as

$$s_i = \begin{cases} 1, & \text{if } d_i \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (34)$$

If $\sigma = -1$, s is calculated as

$$s_i = \begin{cases} 1, & \text{if } d_i \leq 0 \\ 0, & \text{otherwise} \end{cases} \quad (35)$$

After n iterations final value of x_n becomes 0 or almost equal to 0 and y_n nearly converges to $e^{\sigma x}$. A parallel and pipelined architecture for EF for both positive and negative numbers is reported in [7]. This architecture is accurate and has less latency compared to CORDIC based architecture.

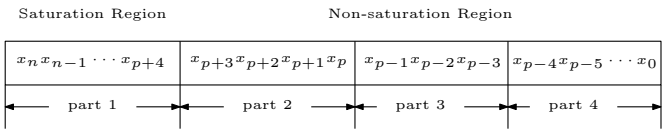


Figure 7: Partition of binary representation of input samples x.

3.4 Combination of LUT and Taylor Series Approximation

Few implementations are based on combination of look-up table (LUT) and Taylor series approximation. Taylor series is very good for approximating EF within range of $0 \leq x < 1$. But computation beyond this range is difficult. Thus few implementations have used LUTs to approximation EF.

Authors in [8] divided input x into three parts, n_1 denotes the integer part, n_2 denotes the upper fractional part ($\geq 2^{-6}$), and n_3 denotes the lower fractional part ($< 2^{-6}$). Then exponential of x can be computed as

$$e^x = e^{n_1} \cdot e^{n_2} \cdot e^{n_3} \tag{36}$$

The components e^{n_1} and e^{n_2} can be computed using LUTs. The component e^{n_3} is computed using 4th order Taylor series approximation based on the following equation

$$fpoly(q) = 1 + q + \frac{1}{2}q^2 + c_3 q^3 + c_4 q^4 \tag{37}$$

where the constants taken as $c_3 = 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-11} + 2^{-13}$ and $c_4 = 2^{-5} + 2^{-7} + 2^{-8}$ for simplification in hardware implementation. Saturation is used if output goes beyond 2^{16} and less than 2^{-15}.

Authors in [9] followed the same procedure by dividing the input x in four parts as shown in Figure 7. Lets say input sample x is represented using fixed point binary representation with word length of n bits and precision of p bits. Range of x can be divided into two parts, saturated and unsaturated region. For, $x \geq 16$ output is saturated to exponential of $(2^p - 16)$. Unsaturated region is divided into 3 parts where part 2 is corresponding to the input values in range $1 \leq x < 16$, part 3 corresponding to the range $0.125 \leq x < 1$, and part 4 is corresponding to the range $x < 0.125$. Output of exponential corresponding to part 2 and 3 is computed based on LUT and exponential output corresponding to part 4 is computed using 3rd order Taylor series approximation using the following expression.

$$e^{-x} = 1 - x(1 - \frac{x}{2}(1 - \frac{2.5x}{8})) \tag{38}$$

The architecture for LUT based approximation [9] is shown in Figure 8. Four bits are used for LUT1 and 3-bits are used for LUT2. Thus depth of LUT1 and LUT2 is 16 and 8 respectively. Authors here implemented the Taylor series approximation using one's complement number system to reduce adder/subtracter blocks.

Authors in [10] reported a template scaling method which is similar to the LUT method discussed previously.

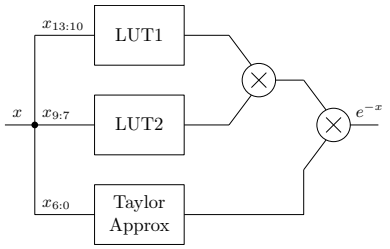


Figure 8: LUT + Taylor series based approximation of exponential.

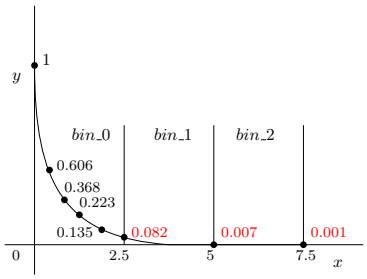


Figure 9: Exponential curve is divided into several bins.

Table 1: Template scaling approximation of exponential.

i	Scaling ($e^{-0.5i}$)	Template ($e^{-0.5i}$)
0	1	1
1	0.082	0.606
2	0.007	0.368
3	0.001	0.223
4	0.000...	0.135

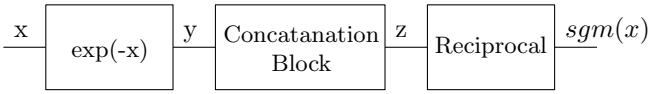


Figure 10: Exponential based sigmoid implementation [7].

Here authors have divided the entire range of exponential into several bins as shown Figure 9. For example, range of (0 - 12.5) is divided in to five bins where each bin has width of 2.5. Exponential values from the first bin are taken as templates (reference bin) and starting values of each bin are considered as scaling factors. Scaling factors and templates for the above example is shown in Table 1. Exponential values in other bins are computed by multiplying corresponding scaling factor and reference template value. Computation of e^{-3} is shown below

$$e^{-3} = e^{-2.5} \cdot e^{-0.5} = 0.0498 \tag{39}$$

Similarly $e^{-3.5}$ is computed as $e^{-2.5} \cdot e^{-1}$. Both templates and scaling factors are required to store in LUTs.

4 Activation Function Implementation Strategies

4.1 Implementation of Sigmoid Function

4.1.1 Exponential Based Sigmoid Function Implementation:- The first approach that comes in mind to compute

Sigmoid function is to compute EF first. Authors in [7], show how efficiently Sigmoid function can be computed using efficient computation of exponential. Here, computation of sigmoid is performed in three steps: Computation of exponential (e^{-x}), computation of $(1 + e^{-x})$ and computation of reciprocal. Computation of all these steps are shown in Figure 10. The input samples should be normalized to take values between 0 to 1.

The output of the exponential block is within the range $\{0, 1\}$ and thus computation of this operation can be optimized. This function can take maximum integer value of 3. This simple addition operation $(1 + e^{-x})$ can be replaced with concatenation operation. This concatenation operation is shown below

$$\begin{cases} z_{13} = 0 \\ z_{12} = y_{10}|y_{11} \\ z_{11} = \ y_{10} \\ z_{9:0} = y_{9:0} \end{cases} \tag{40}$$

where z is the output of the concatenation block and y is the output of the exponential block. Minimum 13-bits are required for correct computation of y and z for precision of 10-bits. The indices at subscripts denote bit positions like $z_{9:0}$ denotes 10-bits from least significant bit (LSB) of z.

One reciprocal operation is required at last stage to compute the sigmoid function. Authors in [7] used partial radix-2 reciprocal unit [11] where number of stages is proportional to the precision. Thus only 11 stages are required for precision of 10-bits. The latency for this reciprocal unit is of 10 clock cycles only.

4.1.2 Look-Up Table Method:- LUT based implementations are popular for simplicity in computation. Authors in [12, 13] took help of LUTs to implement the sigmoid function. Authors in [12] exploited the symmetry property of sigmoid function. Either positive half or negative half can be stored in the LUT. LUT based designs are easy to implement but they need more memory locations.

4.1.3 Direct Approximation:- A very simple approximation of sigmoid function is reported in [14]. The approximation is shown below

$$y = \frac{1}{2}(\frac{x}{1 + |x|} + 1) \tag{41}$$

This approximation is useful in many applications where error within range can be tolerated. With this direct approximation technique, average error of 0.0577 and maximum error of 0.0823 is achieved. Comparison of this technique with ideal sigmoid function is shown in Figure 11(a).

4.1.4 Approximation of Exponential:- Authors in [15] presented a low-complexity architecture for EF. If exponential is converted to base 2, then it can be approximated as

$$e^{-x} = 2^{-1.44x} \tag{42}$$

This approximation is used by authors in [16] to reduce the hardware resources in implementing sigmoid function. Authors here approximated the sigmoid function as

$$y = \frac{1}{1 + 2^{-1.44x}} \approx \frac{1}{1 + 2^{-1.5x}} \tag{43}$$

Table 2: Breakpoints for A-law based approximation of sigmoid function.

x	-0.8	-0.4	-0.2	-0.1	1	2	4.0	8
y	0	0.0625	0.125	0.25	0.75	0.875	0.9375	1

This technique proved quite effective in many applications because of its simplicity and less error in computation. An average error of 0.0026 and maximum error of 0.0087 is achieved with this technique. A comparison of this technique with the actual sigmoid function is shown in Figure 11(b).

4.1.5 Logarithmic Approximation:- Sigmoid function is approximated by logarithm function in [17]. Authors have approximated the sigmoid function within the range $\{-10, 10\}$. If the input sample $x \geq 0$, the output is approximated as

$$y = \frac{log_2(2x + 0.486)}{8} + 0.63 \tag{44}$$

For, $x < 0$ output is approximated as

$$y = \frac{-log_2(-0.5x + 0.008458)}{16} + 0.07 \tag{45}$$

Average error of 0.0678 and maximum error of 0.1899 is estimated with this method. Authors in [17] suggested an approximation technique of logarithm computation based on floating point number system. Logarithm with base 2 of a floating point number ($x = S \cdot M \cdot 2^E$) can be written as

$$\log_2 x = S \cdot (\log_2 M + E) \tag{46}$$

The logarithm of mantissa can be approximated as

$$\log_2 M = \begin{cases} M - 0.92 & 1 < M \leq 1.84 \\ 0.5 * M & 1.84 \leq M < 2 \end{cases} \tag{47}$$

This approximation does not need any multiplication operation which makes logarithmic approximation technique hardware friendly.

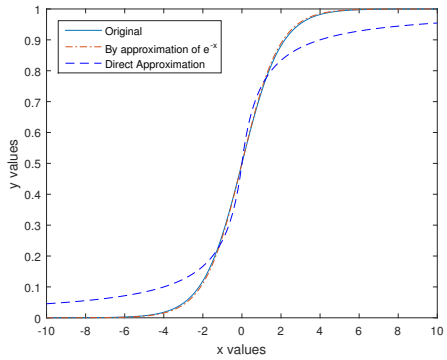
4.1.6 A-law Based Approximation:- A very simple approximation of sigmoid function is presented in [18]. This technique considers many breakpoints and values corresponding to these breakpoints can be stored in LUTs. All these breakpoints are shown in Table 2. The breakpoints have difference of 0.0625. This helps easy binary encoding.

4.1.7 Approximation of Alippi and Storti-Gajani:- Another piecewise linear approximation technique for approximation of sigmoid function is reported in [19]. In this approximation technique, sigmoid function is computed for negative values only. The formula the authors used to approximate sigmoid function is

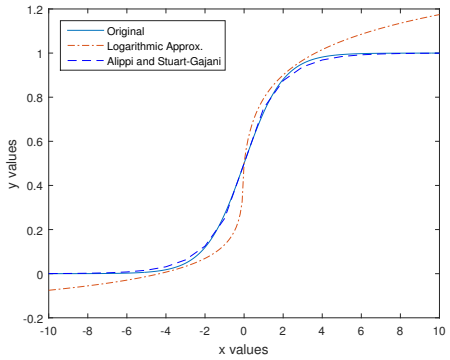
$$y = \frac{0.5 + 0.25\hat{x}}{2^{|(x)|}} \tag{48}$$

where $\hat{x}$ is the decimal part and (x) is the integral part of x. The computation of $\hat{x}$ can be done for negative value of x as

$$\hat{x} = x + |(x)| \tag{49}$$



(a) Actual vs direct approximation vs approx based on exponential.



(b) Original vs logarithmic technique vs Alippi and Stuart-Gajani method.

Figure 11: Comparison of different approximation technique with the original sigmoid function.

This technique involves addition and shifting operations and this suitable for hardware implementation. This approximation method achieves average error of 0.0071 and maximum error of 0.0189.

4.1.8 PLA Approximation Technique:- Sigmoid function can also be computed by piece wise linear approximation (PLA) technique [20, 21, 22, 23]. In this method, sigmoid function is divided into many pieces of linear sections. The approximation of sigmoid function is carried way based on the following equation.

$$y = \begin{cases} 1, & 5 \leq |x| \\ 0.03125 \cdot |x| + 0.84375, & 2.375 \leq |x| < 5.0 \\ 0.125 \cdot |x| + 0.625, & 1.0 \leq |x| < 2.375 \\ 0.25 \cdot |x| + 0.5, & 0 \leq x < 1.0 \end{cases} \quad (50)$$

The mentioned PLA technique approximates sigmoid function for positive value of x. Function y can be approximated for negative value of x by subtracting the result from 1. Average error of 0.0047 and max error of 0.0189 is found in this method. Approximation of sigmoid function using PLA technique is shown in Figure 12(a).

4.1.9 Center Recursive Method:- Authors in [24], demonstrated a centred recursive interpolation (CRI) technique to approximate the sigmoid function. Initially this technique divides the sigmoid curve into three linear segments and then recursively it approximates more linear segments. Initial approximation for CRI technique is

$$y_1(x) = 0 \quad (51)$$
$$y_2(x) = 0.5 + 0.25x \quad (52)$$
$$y_3(x) = 1 \quad (53)$$

The actual CRI technique is shown in Algorithm 1.Here q is the interpolation level, Δ is the depth parameter dependent on q, $h(x)$ is the linear interpolation function, and $g(x)$ is the resulting approximation. Number of segments increased based on the value of q as

$$\text{Number of Segments} = 2^{q+1} + 1 \quad (54)$$

Initially $q = 0$ so there are 3 segments and for $q = 1$, 5 segments were created. For different values of q different optimum values of Δ are taken. These values are shown in Table 3. This technique is efficient in terms of hardware as neither multiplications nor divisions are needed, since they are reduced to shiftings. CRI technique achieves average error of 0.0067 and max error of 0.0196. Approximation of sigmoid function using CRI technique is shown in Figure 12(a). A low cost implementation of sigmoid function is reported in [25]. Here, authors have used optimized equations for piece wise linear approximation technique. The equation they used is given below.

Table 3: Optimum values of Δ in CRI technique.

Δ	0.30895	0.28094	0.26588	0.26380
q	1	2	3	4

Algorithm 1 Centred recursive interpolation of sigmoid function.

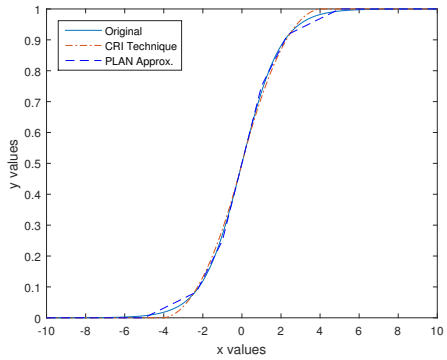
Input: Initially, $g(x) = y_2(x)$ and $h(x) = y_3(x)$.
Output: Approximation of function $g(x)$.

```
for i = 0 : q do
    g'(x) = min{h(x), g(x)}
    h(x) = 1/2 (h(x) + g(x) − Δ)
    g(x) = g'(x)
    Δ = Δ/4
end for
g(x) = min{h(x), g(x)}
```

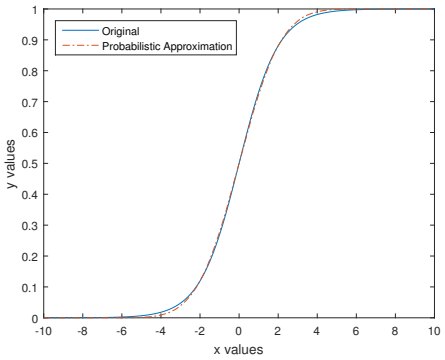
$$y = \begin{cases} 1, & x \leq -2 \\ \frac{x}{4} + 0.5, & -2 < x \leq 2 \\ 1, & x > 2 \end{cases} \quad (55)$$

Similar to this technique, authors in [26] used different linear polynomials in different range of sigmoid function. One of the problem of using different polynomials is that constants vary for different ranges. Thus dedicated multipliers are required.

4.1.10 Non-linear Approximation of Sigmoid Function:- Sigmoid function can also be approximated by non-linear functions. Few literatures [27] have reported implementa-



(a) Actual vs PLA approximation vs CRI technique.



(b) Original vs probabilistic approximation technique.

Figure 12: Comparison of different approximation technique with the original sigmoid function.

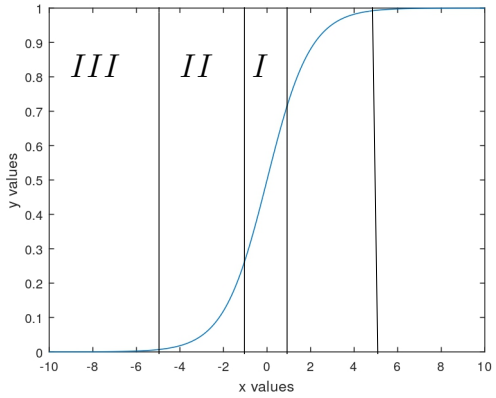


Figure 13: Region wise approximation of sigmoid function.

Table 4: Non-linear approximation of Sigmoid function.

Range	Function
$-1 < x < 1$	$y = c_1x + c_2$
$-5 < x \leq -1$	$c_3x^2 + c_4x + c_5$
$-7.6 \leq x < -5$	c_6
$-\infty \leq x < -7.6$	0
$1 \leq x < 5$	$c_7x^2 + c_8x + c_9$
$5 \leq x < 7.55$	c_{10}
$7.55 \leq x < \infty$	1

tion of sigmoid function using non-linear functions. The sigmoid function is divided into some regions and each region is approximated by one function. Figure 13 shows the segmentation of sigmoid function into different regions. Overall output curve of sigmoid function can be divided into three type of regions. In region I, a linear function can be applied. Similarly in region II a quadratic function can be used for approximation. Table 4 shows different non-linear functions and the ranges of sigmoid function where these functions are applied.

A general second order approximation technique for sigmoid function approximation is reported in [28]. The equations are as follows

$$y = \begin{cases} 1 & x \geq L_h \\ 1 - \frac{1}{2(L_h)^2}(L_h - x)^2, & 0 \leq x < L_h \\ \frac{1}{2(L_h)^2}(L_h - x)^2, & L_l \leq x < 0 \\ 0 & x < 0 \end{cases} \tag{56}$$

where L_h and L_l are the defined upper and lower limits of sigmoid function respectively.

Another non-linear technique to approximate sigmoid function is reported in [29]. Binary version of sigmoid function can be realized as

$$y = \frac{1}{2}Gx + \frac{1}{2} \tag{57}$$

where $G(x)$ defined as

$$G(x) = \begin{cases} 1, & L \leq x \\ H(x), & -L \leq x \leq L \\ -1 & x \leq -L \end{cases} \tag{58}$$

and $H(x)$ function is defined as

$$H(x) = \begin{cases} x(\frac{1}{L} - \frac{1}{L^2}x), & 0 \leq x \leq L \\ x(\frac{1}{L} + \frac{1}{L^2}x), & -L \leq x \leq 0 \end{cases} \tag{59}$$

Here, L is the range within which the sigmoid function is approximated. Authors in [30] provided a special case of the above rule for approximation of sigmoid function by second order equation within range $\{-4, 4\}$. The equations for this approximation is shown below

$$y = \begin{cases} 0.5(1 - |0.25x|)^2, & -4 < x < 0 \\ 1 - 0.5(1 - |0.25x|)^2, & 0 \leq x < 4 \end{cases} \tag{60}$$

This technique is very simple and needs only one multiplier. Absolute error of 0.0111 and max error of 0.0216 are reported for this technique. Similar approximation technique is used in [31].

4.1.11 Combinational Approximation Method:- Authors in [32] suggested a novel bit level mapping method for approximation of sigmoid function. Authors have demonstrated that positive range of sigmoid function can be

Table 5: Notation for *sig_347p* method.

Notation	Significance
s	Sign bit
x	Number of input integer bits
y	Number of input fractional bits
z	Number of output fractional bits
o	a: When all bits of input (+ve and -ve) are mapped. n: When negative bits are only mapped. p: When positive bits are only mapped.

Table 6: Bit level mapping.

x	Binary of x	y	Binary of y
0	000000000	0.5	1000000
0.0625	000000001	0.5156	1000001
0.1250	000000010	0.5312	1000011
⋮	⋮	⋮	⋮
7.9475	011111111	0.9996	1111111

mapped using mapping of binary bits. Fixed point number system is used for this approximation. This technique is famously named as *sig_xyzo* technique. Nomenclature for this technique is shown in Table 5. Table 6 shows approximation of sigmoid function for positive inputs using *sig_347p* format. Here, 3-bits are considered for input integer, 4-bits are used for input fraction, and 7 bits are used for output.

Table 6 shows binary values of different input samples and binary corresponding output values. K-map or any other optimization technique can be used to find the boolean expression for output bits. Valuation of two bits is shown below in case of bit-level mapping technique.

$$y_6 = 1 \tag{61}$$

$$y_5 = x_4x_1 + x_4x_2 + x_4x_3 + x_5 + x_6 \tag{62}$$

Similarly other bits are also can be calculated. Bit-level mapping is an interesting technique where boolean expression for output is directly found.

4.1.12 Probabilistic Model for Sigmoid Function:- Authors used fully probabilistic method to compute sigmoid function in [33]. Let x be the input and η be the independent and identically distributed (IID) Gaussian noise with mean of zero and variance of σ^2, where $\sigma = 1.702$. The output of a sigmoid function can be approximated by the probability of a noisy input $(x + \eta)$ greater than or equal to zero, i.e.,

$$P(x_p + \eta \geq 0) \approx \frac{1}{1 + e^{-x}} \tag{63}$$

The above equation is valid of positive value of x. If x is negative then sigmoid value is estimate by computing $1 - P(x_p + \eta \geq 0)$. If x is positive then following relations

Algorithm 2 Estimation of Sigmoid function using probabilistic method.

Input: The absolute η vector is stored in another vector λ having k samples and $k = 2^s$. Also take initially $idx = \frac{k}{2}$.
Output: Estimation of sigmoid function $(\hat{y})$.

```
for i = 1 : (s − 1) do
  if |x_j| > λ_idx then
    idx = idx + k/2^(i+1)
  else
    if |x_j| < λ_idx then
      idx = idx − k/2^(i+1)
    else
      idx = idx
    end if
  end if
end for

if x_j > 0 then
  ŷ = idx/2k + 0.5
else
  ŷ = 1 − (idx/2k + 0.5)
end if
```

can be written

$$\begin{aligned} P(x_p + \eta \geq 0) &= P(x_p + \eta \geq -x_p) \\ &= P(0 < \eta < x) + 0.5 = \frac{1}{2}P(|\eta| < x) + 0.5 \end{aligned} \tag{64}$$

The following equation can be written using the basic definition of probability.

$$P(|\eta| < x) = \frac{1}{k}\sum_{i=1}^{k} g(\eta_i) \tag{65}$$

$$g(\eta_i) = \begin{cases} 1, & \text{if } x \geq |\eta_i| \\ 0, & \text{otherwise} \end{cases} \tag{66}$$

where η_i is the i^{th} sample of vector η. Authors used binary search [34] algorithm to compute the equation (66). First the Gaussian noise vector η having k samples is computed in software tool and their absolute values are calculated. The absolute values are sorted in ascending order and kept in another vector λ. The value k is large and is power of 2. The method is shown in Algorithm 2. Probabilistic method closely approximates the sigmoid function with average error of 0.0038 and max error of 0.0098. Comparison of probabilistic method and original sigmoid function is shown in Figure 12(b).

4.2 Implementation of Hyperbolic Tangent

Implementation of hyperbolic tangent function can achieved in three ways. First method is based on computation of EF, second method is based on computation of sigmoid function, and in the third method, hyperbolic tangent function is directly approximated. Following equation shows how tanh function can be realized using sig-

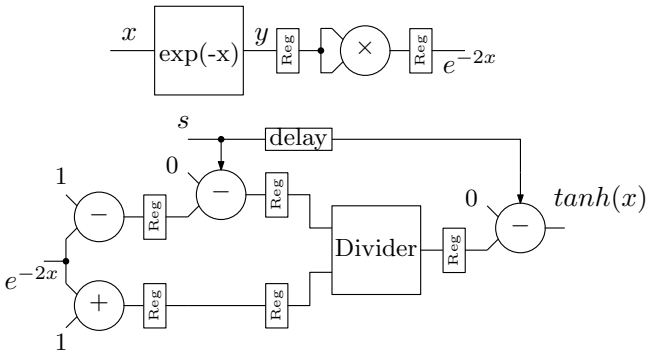


Figure 14: Schematic for tanh function evaluator [7].

moid function

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2 \cdot \sigma(2x) - 1 \tag{67}$$

Exponential based computation of tanh function faces range issues as computation of EF is preferable only in $\{0, 1\}$ range. Instead sigmoid function can be computed by any approximation technique to compute tanh function.

4.2.1 Exponential Based Implementation:- The architecture for tanh AF using exponential architecture is shown in Figure 14. Here, same exponential architecture (shown in Figure 8.7) is used to compute tanh function. The registered output of the exponential block is input to a multiplier to compute square. Architecture for tanh function involves variable data-width as one square operation is involved. Compared to the architecture of the sigmoid function, 1 extra bit is required to accommodate the maximum value (e^2) after square operation.

Output of the multiplier is then supplied to one adder and one subtracter. Finally, outputs of the adder and subtracter are sent to the divider. Divider works in similar fashion as the reciprocal used for sigmoid function but with latency of 11 clock cycles. Divider works for unsigned number and thus inverters are placed at input and output stage. The control s is high when $1 - e^{-2x}$ is negative.

4.2.2 Direct Approximation:- Few research works have directly computed the hyperbolic tangent function. For example hyperbolic tangent function is directly approximated by using polynomials of 1st and 2nd order in [35]. Authors in [35] shows that more accuracy is achieved by 2nd order polynomials but with the cost of increased hardware. Authors in [36] used curve-fitting technique and used a 4th order polynomial technique to approximate the tanh function by the following polynomial

$$f(x) = -0.009x^4 + 0.118x^3 - 0.575x^2 + 1.246x - 0.024 \tag{68}$$

This function can easily implemented by serial step-by-step multiplication as

$$f(x) = (((-0.009x - 0.118)x - 0.575)x + 1246)x - 0.024) \tag{69}$$

The above equation reduces number of multiplication compared to previous equation.

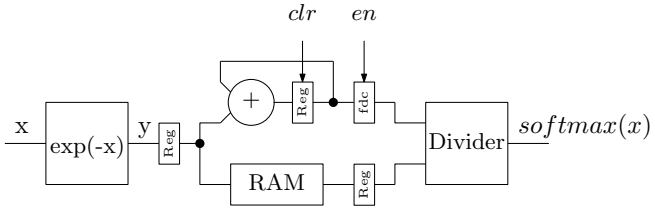


Figure 15: Schematic for softmax function evaluator [7].

Authors in [3] implemented tanh function using CORDIC. Few implementations [22] used sigmoid function to compute tanh function. Few implementations [37] used LUT for tanh function by exploiting the symmetry property. In [38], authors implemented tanh function using DCT interpolation technique. DCT interpolation technique is good for achieving a smooth curve with high accuracy but consumes resources as filter co-efficients are to be stored in memory. Authors in [39] used Chebyshev approximation technique for approximating tanh function. Authors in [40] implemented tanH function using different models including continued fraction and polynomial approximations.

4.3 Implementation of Softmax Function

Many research papers have implemented softmax AF using EF [41]. A direct implementation for softmax function is shown in Figure 15. In majority architectures, output of a NN is serial. Thus output of the exponential block is serially loaded to a random access memory (RAM). So, output for k classifiers will be stored in RAM. Simultaneously, exponential values are also accumulated with the help of an adder and a register. The *clr* signal clears the register at the start of accumulation.

Accumulated value is then loaded to another register (fdc) by another control signal *en*. Once the accumulation is completed, the exponential values are read from the memory and fed to the divider. The architecture for softmax function also requires variable data-width because of accumulation operation. The width of the divider varies with the width of the accumulator. Width of the accumulator depends on number of classifiers.

Direct implementation needs dedicated divider which consumes more hardware. Authors in [42] computed inverse of softmax function and then used reciprocal block at the end. The inverse of the softmax function can be written as

$$\begin{aligned} f_{sm}^{-1}(x_i) &= \frac{\sum_{i=1}^{k} e^{x_i}}{e^{x_i}} \\ &= \frac{e^{x_1} + e^{x_2} + \cdots + e^{x_i} + \cdots + e^{x_k}}{e^{x_i}} \\ &= 1 + \sum_{j=1}^{k} e^{x_j - x_i} \end{aligned} \tag{70}$$

Here, CORDIC block is used to compute EF and then these exponential values are accumulated and added with 1. The use of reciprocal unit instead of divided reduces hardware.

4.4 Implementation of Other Activation Functions

Authors in [43] presents an AF core that supports various functions including Gaussian, Sigmoid linear unit (SiLU), ELU, Softplus in addition to classic ones. The core uses piecewise polynomial approximation for efficient hardware execution, demonstrating flexible support for hardware implementations of non-standard AF. The manuscript [44] specifically targets the Swish AF, which is smoother and often performs better than ReLU. Because Swish is computationally complex (it involves multiplication and a logistic component), the authors propose a hardware-efficient approximation and demonstrate its implementation with improved delay, area, and power characteristics compared to earlier approximations. The manuscript [45], implements and evaluates Gaussian AFs on FPGA as part of RBF networks, providing logic utilization and timing results that highlight hardware design considerations for non-standard activations.

5 Estimation of Function Approximation

This section discusses some metrics estimation the approximation accuracy. Error between the actual function and the approximated function is calculated as

$$e_j = |y_i - \hat{y_j}| \tag{71}$$

where y_i is i^{th} sample of actual function, $\hat{y_j}$ is the i^{th} sample of approximated function, and e_j is the error for i^{th} sample. The average error is then calculated as

$$e_{avg} = \frac{1}{n}\sum_{j=1}^{n} e_j \tag{72}$$

where n denotes the total number of samples. Maximum error can be computed as

$$e_{max} = \underset{1\cdots n}{max}\ e_j \tag{73}$$

Accuracy of an architecture is also measured in terms of relative signal to noise ratio (RSNR). The equation of calculating RSNR is shown below

$$RSNR = 20\log_{10}\left(\frac{\|x\|_2}{\|x-\hat{x}\|_2}\right) \tag{74}$$

Authors in [7], implemented sigmoid, tanh, and softmax function based parallel exponential block on Artix-7 FPGA board (9xc7a100tftg256-2). A comparison of three architectures is shown in Table 7. Here authors achieved RSNR of 51 dB, 45.86 dB, and 46 dB in computation of Sigmoid function, tanH function, and softmax function respectively.

Table 7: FPGA performance comparison [7].

Parameters	sigmoid	tanh	softmax
Slice LUTs	644	712	716
Slice Registers	559	651	670
Occupied Slice	224	258	245
DSP48E1	-	1	-
Maximum Frequency	181.81 MHz	181.81 MHz	181.81 MHz
Dynamic Power	0.052 W	0.057 W	0.052 W

6 Conclusion

This paper presented a brief survey of AFs used in modern NNs, including classical functions such as sigmoid and hyperbolic tangent, widely used ReLU variants, and recently proposed nonlinear functions with their mathematical properties. In addition, this paper reviewed major techniques for computing EFs, which constitute the core computational component of several nonlinear AFs. Approaches such as lookup tables, polynomial and piecewise approximations, CORDIC based methods, and iterative algorithms were analyzed in terms of accuracy, latency, memory overhead, and hardware complexity. This survey focused on a comprehensive review of different techniques for implementing AFs on hardware platforms. AFs can be implemented directly using EFs, by approximation of EF or by using other linear or non-linear approximation techniques. Different design metrics are also discussed which estimates the accuracy in approximation of an AF.

References

[1] N. Patwardhan, M. Ingalhalikar, and R. Walambe, "Aria: utilizing richard's curve for controlling the non-monotonicity of the activation function in deep neural nets," *arXiv preprint arXiv:1805.08878*, 2018.

[2] S. Roy, *CORDIC Algorithm.* Cham: Springer International Publishing, 2024, pp. 207–225. [Online]. Available: https://doi.org/10.1007/978-3-031-41085-7_11

[3] A. Boudabous, F. Ghozzi, M. Kharrat, and N. Masmoudi, "Implementation of hyperbolic functions using cordic algorithm," in *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.*, 2004, pp. 738–741.

[4] B. Wang, Z. Duan, Z. Shen, Y. Zhao, L. Gao, and C. Wang, "A reconfigurable high-precision and energy-efficient circuit design of sigmoid, tanh and softmax activation functions," in *2023 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, 2023, pp. 118–119.

[5] D. Wu, T. Chen, C. Chen, O. Ahia, J. San Miguel, M. Lipasti, and Y. Kim, "Seco: A scalable accuracy approximate exponential function via cross-layer optimization," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).* IEEE, 2019, pp. 1–6.

[6] I. Koren, *Computer arithmetic algorithms.* AK Peters/CRC Press, 2018.

[7] S. Roy, "Fpga implementation of sigmoid, hyperbolic tangent and softmax functions using efficient architecture for exponential function," in *2025 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2025, pp. 1–5.

[8] J. Partzsch, S. Höppner, M. Eberlein, R. Schüffny, C. Mayr, D. R. Lester, and S. Furber, "A fixed point exponential function accelerator for a neuromorphic many-core system," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[9] M. Chandra, "On the implementation of fixed-point exponential function for machine learning and signal-processing accelerators," *IEEE Design & Test*, vol. 39, no. 4, pp. 64–70, 2021.

[10] J. Kim, V. Kornijcuk, and D. S. Jeong, "Tsefa: Resource-efficient high-precision approximation of exponential functions based on template-scaling method," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2020, pp. 358–363.

[11] S. Roy, *Advanced Digital System Design: A Practical Guide to Verilog Based FPGA and ASIC Implementation*. Springer Nature, 2023.

[12] M. T. Ali and B. H. Abd, "An efficient area neural network implementation using tan-sigmoid look up table method based on fpga," in *2022 3rd International Conference for Emerging Technology (INCET)*, 2022, pp. 1–7.

[13] R. Pogiri, S. Ari, and K. Mahapatra, "Design and fpga implementation of the lut based sigmoid function for dnn applications," in *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*. IEEE, 2022, pp. 410–413.

[14] A. Tisan, S. Oniga, D. Mic, and A. Buchman, "Digital implementation of the sigmoid function for fpga circuits," *Acta Technica Napocensis Electronics and Telecommunications*, vol. 50, no. 2, p. 6, 2009.

[15] S. Gomar and A. Ahmadi, "Digital multiplierless implementation of biological adaptive-exponential neuron model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1206–1219, 2014.

[16] S. Gomar, M. Mirhassani, and M. Ahmadi, "Precise digital implementations of hyperbolic tanh and sigmoid function," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, 2016, pp. 1586–1589.

[17] P. W. Zaki, A. M. Hashem, E. A. Fahim, M. A. Mansour, S. M. ElGenk, M. Mashaly, and S. M. Ismail, "A novel sigmoid function approximation suitable for neural networks on fpga," in *2019 15th International Computer Engineering Conference (ICENCO)*. IEEE, 2019, pp. 95–99.

[18] D. Myers and R. Hutchinson, "Efficient implementation of piecewise linear activation function for digital vlsi neural networks," *Electronics Letters*, vol. 25, pp. 1662–1663, 1989. [Online]. Available: https://digital-library.theiet.org/doi/abs/10.1049/el%3A19891114

[19] C. Alippi and G. Storti-Gajani, "Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning," in *1991 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1991, pp. 1505–1508 vol.3.

[20] R. Pogiri, S. Ari, and K. K. Mahapatra, "Design and fpga implementation of the lut based sigmoid function for dnn applications," in *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2022, pp. 410–413.

[21] R. Murmu, M. Bhattacharya, and S. Kumar, "Enhancing performance of sigmoid implementation in fpga using approximate computing," in *2024 International Conference on Informatics Electrical and Electronics (ICIEE)*, 2024, pp. 1–5.

[22] A. Vaisnav, S. Ashok, S. Vinaykumar, and R. Thilagavathy, "Fpga implementation and comparison of sigmoid and hyperbolic tangent activation functions in an artificial neural network," in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, 2022, pp. 1–4.

[23] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware implementation of sigmoid activation functions using fpga," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, 2019, pp. 34–38.

[24] K. Basterretxea and J. M. Tarela, "Approximation of sigmoid function and the derivative for artificial neurons," *advances in neural networks and applications, WSES Press, Athens*, pp. 397–401, 2001.

[25] K. Tatas and M. Gemenaris, "High-performance and low-cost approximation of ann sigmoid activation functions on fpgas," in *2023 12th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2023, pp. 1–4.

[26] H. Faiedh, Z. Gafsi, and K. Besbes, "Digital hardware implementation of sigmoid function and its derivative for artificial neural networks," in *ICM 2001 Proceedings. The 13th International Conference on Microelectronics.* IEEE, 2001, pp. 189–192.

[27] Z. Xie, "A non-linear approximation of the sigmoid function based on fpga," in *2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2012, pp. 221–223.

[28] K. Sammut and S. Jones, "Implementing nonlinear activation functions in neural network emulators," *Electronics Letters*, vol. 27, pp. 1037–1038, 1991. [Online]. Available: https://digital-library.theiet.org/doi/abs/10.1049/el%3A19910645

[29] H. K. Kwan, "Simple sigmoid-like activation function suitable for digital hardware implementation," *Electronics letters*, vol. 28, no. 15, pp. 1379–1380, 1992.

[30] M. Zhang, S. Vassiliadis, and J. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 1045–1049, 1996.

[31] I. Tsmots, O. Skorokhoda, and V. Rabyk, "Hardware implementation of sigmoid activation functions using fpga," in *2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*. IEEE, 2019, pp. 34–38.

[32] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, 2003.

[33] W. Lu, M. Lu, X. Zhang, Z. Lu, M. Sun, B. Dong, and Z. Shu, "A fully probabilistic model for sigmoid approximation and its hardware- efficient implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 8, pp. 3775–3786, 2024.

[34] D. E. Knuth, "Sorting and searching," *The Art of Computer Programmimg*, vol. 422, pp. 559–563, 1973.

[35] S. Bouguezzi, H. Faiedh, and C. Souani, "Hardware implementation of tanh exponential activation function using fpga," in *2021 18th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE, 2021, pp. 1020–1025.

[36] R. K. Yousif, I. A. Hashim, and B. H. Abd, "Low area fpga implementation of hyperbolic tangent function," in *2023 6th International Conference on Engineering Technology and its Applications (IICETA)*. IEEE, 2023, pp. 596–602.

[37] K. Leboeuf, R. Muscedere, and M. Ahmadi, "Performance analysis of table-based approximations of the hyperbolic tangent activation function," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2011, pp. 1–4.

[38] A. M. Abdelsalam, J. M. P. Langlois, and F. Cheriet, "A configurable fpga implementation of the tanh function using dct interpolation," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 168–171.

[39] Z. Hajduk and G. R. Dec, "Very high accuracy hyperbolic tangent function implementation in fpgas," *IEEE Access*, vol. 11, pp. 23 701–23 713, 2023.

[40] C. Hingu, X. Fu, T. Saliyu, R. Hu, and R. Mishan, "Power-optimized field-programmable gate array implementation of neural activation functions using continued fractions for ai/ml workloads," *Electronics*, vol. 13, no. 24, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/24/5026

[41] M. A. Hussain and T.-H. Tsai, "An efficient and fast softmax hardware architecture (efsha) for deep neural networks," in *2021 IEEE 3rd International conference on artificial intelligence circuits and systems (AICAS)*. IEEE, 2021, pp. 1–4.

[42] A. Kagalkar and S. Raghuram, "Cordic based implementation of the softmax activation function," in *2020 24th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 2020, pp. 1–4.

[43] G. González-Díaz Conti, J. Vázquez-Castillo, O. Longoria-Gándara, A. Castillo-Atoche, R. Carrasco-Álvarez, A. Espinoza-Ruiz, and E. Ruíz-Ibarra, "Hardware-based activation function-core for neural network implementations," *Electronics*, vol. 11, no. 1, p. 14, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/14

[44] K. Choi, S. Kim, J. Kim, and I.-C. Park, "Hardware-friendly approximation for swish activation and its implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 10, pp. 4516–4520, 2024.

[45] V. Shymkovych, S. Telenyk, and P. Kravets, "Hardware implementation of radial-basis neural networks with gaussian activation functions on fpga," *Neural Computing and Applications*, vol. 33, no. 15, pp. 9467–9479, 2021.