



TWO-STEPS IMPLEMENTATION OF SIGMOID FUNCTION FOR ARTIFICIAL NEURAL NETWORK IN FIELD PROGRAMMABLE GATE ARRAY

Syahrlanuar Ngah¹, Rohani Abu Bakar¹, Abdullah Embong¹ and Saifudin Razali²

¹Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Malaysia

²Faculty of Electrical and Electronic Engineering, Universiti Malaysia Pahang, Malaysia

syahrlanuar@ump.edu.my

ABSTRACT

The complex equation of sigmoid function is one of the most difficult problems encountered for implementing the artificial neural network (ANN) into a field programmable gate array (FPGA). To overcome this problem, the combination of second order nonlinear function (SONF) and the differential lookup table (dLUT) has been proposed in this paper. By using this two-steps approach, the output accuracy achieved is ten times better than that of using only SONF and two times better than that of using conventional lookup table (LUT). Hence, this method can be used in various applications that required the implementation of the ANN into FPGA.

Keywords: sigmoid function, second order nonlinear function, differential lookup table, FPGA.

INTRODUCTION

The Artificial neural network (ANN) has become a growing research focus over the last few decades. It have been used broadly in many fields to solve great variety of problems such as speed estimation [1], image processing [2] and pattern recognition [3]. ANN have been implemented mostly in software, however hardware version are gaining importance [4]. Software implementation offers flexibility for emulating wide range of neural networks model while the hardware implementation is vital in taking advantages of inherent parallelism [5]. The implementation of artificial neural network in field programmable gate array is attracting more attention now due to two reasons. Firstly, though the performance of conventional processor is continually improved, the fastest sequential processor is still unable to provide a real-time response for the ANN with a large number of neurons. Secondly, most implementations have been done in the form of simulations running on a personal computer (PC) or workstation owing to the lack of portable software for running the ANN [6]. Alternatively, the ANN can be run on field programmable gate array (FPGA) since it provides more portable platform for numerous real-world applications. FPGA is completely programmable and efficient device to replace the microprocessor yet faster to develop the dedicated chips [7]. At the same time, parallel structure of ANN makes it potentially fast for the computation of certain task and resulted reduce the processing time [4][8][9]. However, implementing the ANN into FPGA is not as simple as implementing in software especially when involving a large number of neuron and the calculation of complex equation such as activation function[9]. There are several common types of activation function used in ANN define, respectively as linear, bipolar threshold, sigmoidal (sigmoid function) and hyperbolic tan [7][10][11]. Table-1 lists a few of linear and nonlinear activation functions. Comparison between various nonlinear activation function

namely Gaussian, logarithmic, hyperbolic tangent and sigmoid function has been done by T. Tan *et al.* They found that, the optimal configuration for their controller can be achieved by using the sigmoid function [12]. In this paper, sigmoid function is being used as an activation function. It is an important component of ANN, mainly to limit down the neuron output. Since the hardware is unable to calculate the floating number as compared to software especially when involving the activation function, new method is needed to do the calculation and at the same time produce the output as similar as software calculation.

Table-1. Some common activation function.

Name	Function
Binary threshold (Step function)	$f(x)=1; x \geq 1$ $f(x)=0; x < 0$
Bipolar threshold (sign function)	$f(x)=1; x \geq 1$ $f(x)=-1; x < 0$
Linear	$f(x)=x$
Saturating linear	$f(x)=1; x \geq 1$ $f(x)=0; x < 0$ $f(x)=x; 0 \leq x < 1$
Hyperbolic tangent	$f(x)=(e^x - e^{-x}) / (e^x + e^{-x})$
Sigmoid function	$f(x)= 1/(1+e^{-x})$

ARTIFICIAL NEURAL NETWORK (ANN)

NNs is one of the techniques in the area of artificial intelligence known as nature inspired. It can be defined as a model of reasoning based on the human brain. It resembles the human brain in two aspects i) the acquired knowledge by the network is gain from the environment through a learning process and ii) the strength of connection between neuron (weight connection) are used to store the acquired knowledge. The ANNs consists of a huge class of different architectures. There are several



factors need to be considered before implementing the ANN in solving certain problems. This is because ANN will only perform excellently if the selections are perfectly matched to the problems [13]. The general structure of ANN model is depicted in Figure-1. This model is composed of input, hidden and output layers. The number of neurons in the input and output layers are determined by system itself. For example, let used Figure-1 for positioning the object. Figure-1 have 4 neurons since it need to receive 4 distance enter from the user/system and 2 neuron in output layer to produce 2 output of coordinate x and y. Figure-2 depicts one of the neurons where the calculation of activation function is involved.

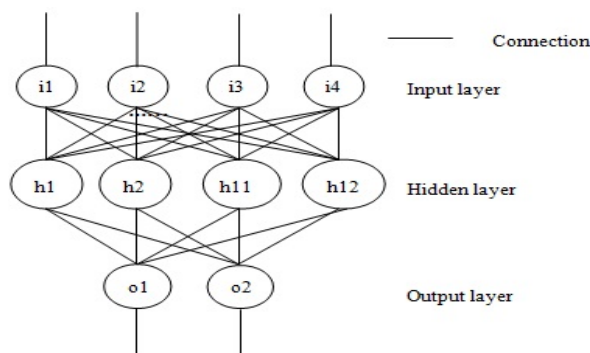


Figure-1. General structure of neural network.

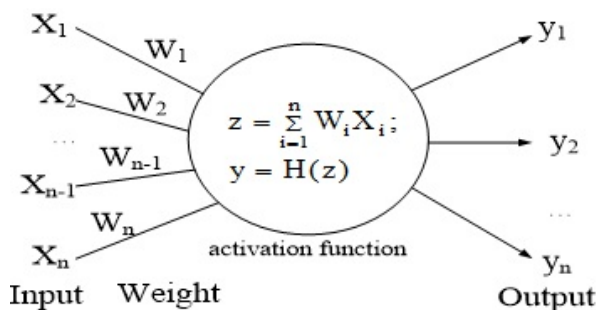


Figure-2. Abstracted model of neuron with connection.

HARDWARE IMPLEMENTATION OF ANN

ANN has been mostly implemented in the software [14][15]. The benefit is that the designer does not need to understand how the neural network elements work. The parallel structure of the ANN makes it potentially fast for hardware implementation. Basically, the implementation of the ANN into hardware can be classified into three categories [15] which are digital signal processing (DSP) [16], application specific integrated circuits (ASIC)[17] and FPGA. ANN can be performed in parallel, but this feature cannot be exploited in DSP since DSP chips perform the tasks given in serial manner. On the other hand, if the ANN is implemented in ASIC, the network configuration cannot be updated or modified to any further extent. It is fixed into ASIC

hardware. Both problem can be solved by using the FPGA [18].

However, in order to implement the ANN on FPGA, a special attention must be given on the activation function, particularly for the nonlinear activation function. A high accuracy of ANN output requires a high precision in the output of the activation function. Since the output of the activation function will become the input for the next layer, any error that exists at the output of activation function are accumulated and amplified by the ANN. Hence, the final results of the ANN will contain a very large error.

As aforementioned, the sigmoid function as the activation function is one of the most frequently used in back-propagation neural networks applications. Sigmoid function provides smooth transition between input and output thereby improving the neural response when compared to other activation function like step function or saturating linear activation function [19]. It can be described by the following mathematical equation:

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

Figure-3 shows the output of the eq.1 (curve of the sigmoid function). Nevertheless, this function is not suitable for direct hardware implementation. A straightforward implementation of the sigmoid function is unfeasible, since both division and exponentiation are very demanding operations in terms of area of resources and long clock cycle [20]. A more practical solution is needed to overcome this problem. The used of approximation approach/method is one of work has been done on the hardware implementation of the sigmoid function [21][22].

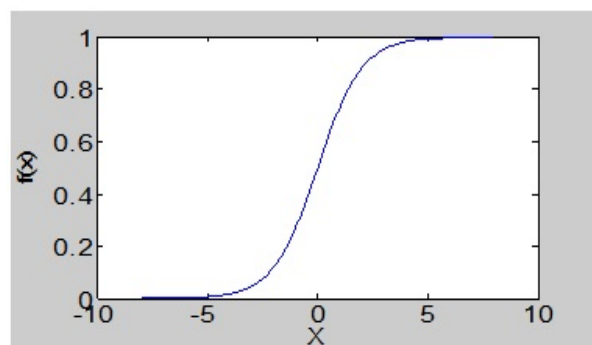


Figure-3. Sigmoid function curve.

a) Conventional sigmoid function implementation

It has been reported that several methods can be employed for implementing the sigmoid function in hardware, such as look-up table (LUT), coordinate rotation digital computer (CORDIC) algorithm, piecewise linear approximation (PWL) and piecewise nonlinear approximation[23][24][25]. All these methods will be discussed in details in the next sub-section.



i. Look-up table (LUT)

The simplest implementation of sigmoid function is the use of a LUT. Many implementations used a LUT for the approximation of the sigmoid function [26][27][28]. The value of the sigmoid function curve are divided into several segments and stored into table as shown in Figure-4. However the amount of hardware memory required for LUT implementation can be quite large especially if one requires higher accuracy in approximation since the value of the sigmoid function curve need to be divided into more segments. To create a LUT, a limited operating range should be determined.

Based on Figure-4, the sigmoid function is nearly 1 for $x > 8$ and 0 for $x < -8$. The non-saturation range is between -8 and 8. Since the sigmoid function has a symmetry point at (0, 0.5), only half of the x-y pairs have to be computed [20] and can be represented by:

$$f_{x>0} = 1 - f_{x\leq 0} \quad \text{or} \quad (2)$$

$$f_{x<0} = 1 - f_{x\geq 0} \quad (3)$$

In other words, the LUT only needs to operate between 0 and 8 [25][29]. Then, the LUT size then can be decreased into half of the desired range. The resulting table has 11 address bits selecting the 8 bits fractional portion, using $2^8 \times 8 \text{ bits} = 16 \text{ Kbits}$ of memories. The deviation between the LUT and the sigmoid function are ranges from -0.005 to 0.005.

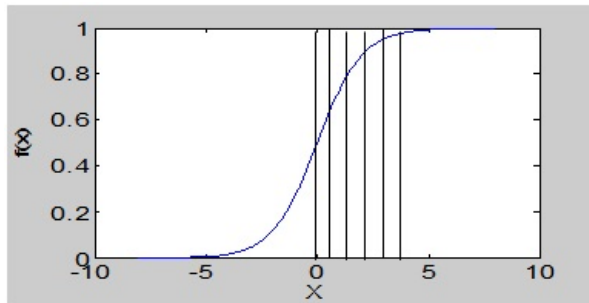


Figure-4. LUT implementation for sigmoid function.

ii. CORDIC algorithm

The modern CORDIC algorithm was first described in 1959 by Jack E. Volder and commonly used for FPGAs [30][31]. This algorithm is another alternative to calculate the sigmoid function. The idea of CORDIC algorithm is based on the rotating of vector on a plane. It starts from the initial position to until it coincides with the desired target position. The rotation of vector is done by using fixed number of angular steps which are executed in sequence [32]. A fundamental problem with CORDIC is that it is inherently a sequential approach. This implies that the CORDIC function gains accuracy at the cost of latency; where latency is defined as the number of clock cycles required from the start of the calculation until the resulting data is gained. Each additional stage in the CORDIC calculation simultaneously increases the

accuracy and the time required to complete the calculation by one or more clock cycles [25]. The 11-stage CORDIC function has a latency of 50 clock cycles. The deviation resulted from the CORDIC function ranges from -0.005 to 0.005.

iii. Piecewise linear approximation (PWL)

There are few PWL methods proposed by researches such a-law based companding technique [33], PWL of a nonlinear function (PLAN) [34] and recursive algorithm based on the centred recursive interpolation (CRI) [21]. Basically, all the proposed PWL methods obtain the low value of both maximum and average error with low computational complexity. They are different in numbers and locations of start and end point of the approximating lines. A-Law based technique used a modified curve so that the gradient of each linear segment is expressed by the power of two. Meanwhile PLAN approximation that proposed by Amin et al. [34] divided the sigmoid function into 5 segments as shown in Table-2. The calculations only need to be performed on the absolute value of the input x. After simplifying the shift and addition operations implicit in Table.2, the bit-level logic equations become effective for implementation. CRI methods try to improve the accuracy recursively. Initially, three straight lines are used:

$$f_1(x) = 0$$

$$f_2(x) = \frac{1}{2} * \left(1 + \frac{x}{2}\right) \quad (4)$$

$$f_3(x) = 1$$

The number of linear segments increased exponentially in order to improve the accuracy of the sigmoid function output. The summary of the CRI implementation is shown in Table-3. All of the proposed PWL approximation methods that have been discussed do not require multipliers. Hence, it is suitable for implementing into hardware.

Table-2. Implementation of PLAN.

Operation	Condition
$f(x)=1$	$ x \geq 5$
$f(x)=0.03125 * x + 0.84375$	$2.375 = < x < 5$
$f(x)=0.125 * x + 0.625$	$1 = < x < 2.375$
$f(x)=0.25 * x + 0.25$	$0 = < x < 1$
$f(x) = 1 - Y$	$x < 0$

Table-3. CRI approximation.

Interpolation Level	Optimum depth parameter	Number of segments
q=0		3
q=1	$\Delta_1, \text{opt} = 0.30895$	5
q=2	$\Delta_2, \text{opt} = 0.28094$	9
q=3	$\Delta_3, \text{opt} = 0.26588$	17



iv. Piecewise nonlinear approximation

In general, piecewise nonlinear approximation used the eq.5 to calculate the sigmoid function.

$$f(x) = c_0 + c_1 * x + c_2 * x^2 \quad (5)$$

Eq.5 can be simplified to become an eq.6 as shown.

$$f(x) = c_0 + x(c_1 + c_2 * x) \quad (6)$$

However the need of two multiplier in the eq.6 become the main shortcoming of this method [20][35] which result in low conversion speed.

Another method, a simple second order nonlinear function (SONF) to approximate the sigmoid function has been presented by Zhang et al. [23] with no cost of memory. Zhang has dividing the sigmoid function into 4 segments, which is given as follows:

$$f(x) = \begin{cases} 1 - \frac{1}{2} \left(1 - \frac{1}{4} |x|\right)^2, & 0 < x \leq 4 \\ \frac{1}{2} \left(1 - \frac{1}{4} |x|\right)^2, & -4 < x \leq 0 \\ 1, & 4 < x \\ 0, & x \leq -4 \end{cases} \quad (7)$$

Similarly to CORDIC function, this nonlinear function can be directly implemented using digital techniques. The SONF can be implemented with one multiplier, two shifters and two XORs. Its operation speed is higher than that in the PWL and CORDIC function. Yet this method has a limited accuracy. Figure-5 shows the comparison of the output produced by SONF (eq.7) and sigmoid function (eq.1). The deviation produced by the SONF method ranges from -0.022 to 0.022 as shown in Figure-6. Hariprasath et al increased the range of the eq.7 to -8 and 8. However, the result produces is not much differences with the maximum absolute error (deviation) is 6.59% [36].

In summary, both LUT and CORDIC function can achieve a higher accuracy but the LUT requires huge amount of hardware resources and CORDIC function suffers with long computation time since it required more iteration in order to achieve it. Meanwhile PWL and piecewise nonlinear approximation need less hardware resources and are faster in computation time since both methods eliminate the need for multiplication and addition operation. However, in term of accuracy, both are still low as compared to LUT and CORDIC.

b) The proposed method

This subsection proposed a new sigmoid function method for the hardware implementations. The main idea came from the advantage of the SONF and LUT. As explained before, the output deviation produced by the SONF ranges from -0.025 to 0.025. This deviation values then are used to create a LUT. This new LUT is called differential-LUT. The Differential-LUT has the same criteria like conventional LUT. The only differences is the

value to create the differential-LUT itself. Instead of using the input value between 0 and 8 for indexing the sigmoid function output between 0 and 1 for conventional LUT, the differential-LUT just stored the input value between 0 and 6 and divided into 64 segments, for indexing the sigmoid function output between 0 and 0.025 (the deviation range achieved by the SONF). Basically, more segments can be used with the cost of higher hardware resources. Meanwhile less segments will compromise the sigmoid function output. As a result of dividing the input value into 64 segments, the differential LUT provides 5 bits of output data. That is the reason why the differential-LUT need less hardware resources (320 bits) compared to conventional LUT.

The implementation concept of this two-steps approach is illustrated in Figure-7. Firstly, run the SONF by using eq.7 to calculate the neuron input to produce initial sigmoid function output. The next step, this initial output value is added/deducted to the differential-LUT value based on the indexing value. If the input values for SONF are smaller than zero, then the value stored in differential-LUT is deducted from the output value of the SONF. Otherwise the output value of the SONF is added to the value that is stored in the differential LUT. By using this new approaches, (a combination of SONF and differential-LUT) the deviation range is reduced to between -0.0022 and 0.0022. Figure-8 indicates that the deviation achieved by the proposed method compared to SONF. It is clearly shows that the proposed method can reduce the deviation 10 times better than that the SONF method.

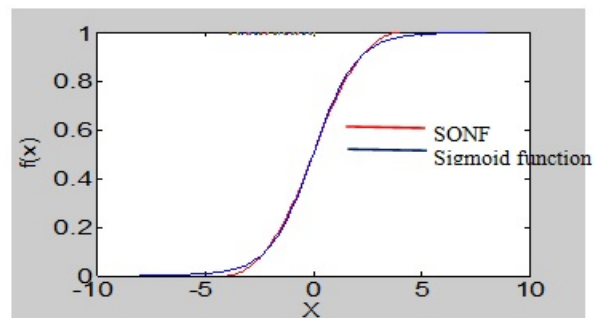


Figure-5. Output of SONF and sigmoid function.

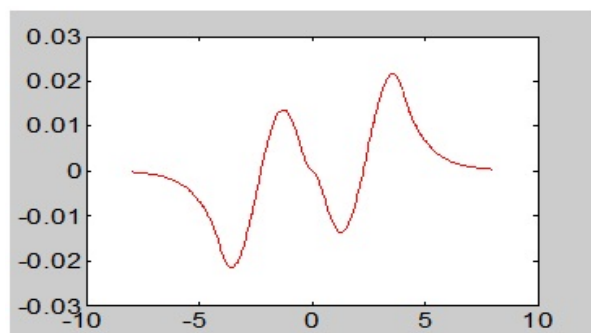


Figure-6. Deviation range between eq.1 and eq.7.

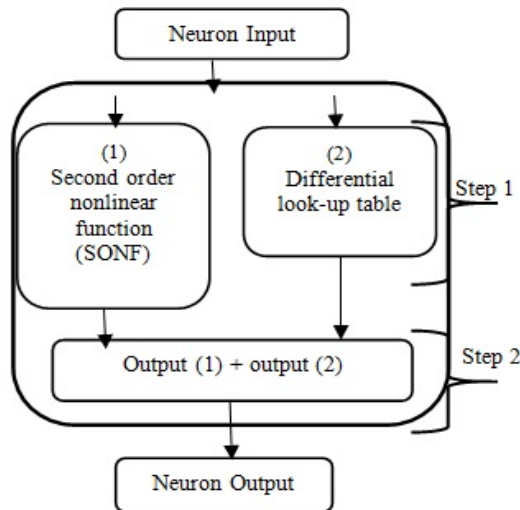


Figure-7. Proposed two-steps implementation of the sigmoid function.

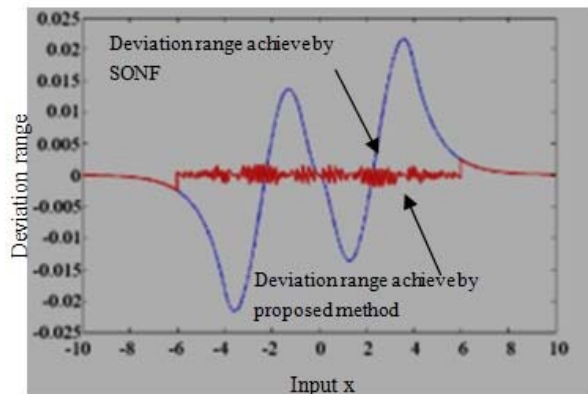


Figure-8. Deviation range achieved by the proposed method is 10 times better than the SONF.

RESULTS AND DISCUSSION

Table-4 shows a comparison of the different sigmoid function implementations. The table lists the memory, the clock cycle and the deviation of each method compared to sigmoid function. The LUT has the lowest clock cycle and produces second lowest of the deviation. Contrary to CORDIC function, it uses the highest clock cycle (50) to achieve the same deviation as LUT. Meanwhile, SONF has the worse deviation as compared to the other methods. The proposed method needs three extra clock cycles than the SONF and achieves the smallest deviation among the four methods in the table.

Based on the above discussion, it can be concluded that, the deviation in output produced by the proposed method is two times better than those in the conventional LUT and CORDIC, 10 times better than that of the SONF method. The proposed method combines the SONF with a differential LUT to achieve a higher accuracy hardware implementation of the sigmoid function.

Table-4. Summary deviation of sigmoid function implementation.

Implementation	Clock Cycle	Range different with Eq.3	Memory used
LUT[25]	3	-0.005 to 0.005	16Kbits
CORDIC[25]	50	-0.005 to 0.005	0
SONF	10	-0.022 – 0.022	0
The propose method	13	-0.0022 to 0.0022	320 bits

CONCLUSIONS

This paper has introduced a combination of SONF and differential-LUT as a sigmoid function to improve performances of implementing the ANN in FPGA. This idea is based on replacement of the complex sigmoid function with hardware-based sigmoid function in neuron. Slow response and portable issue for real-times application that arise in the software-based ANN can be solved by implementing the whole ANN in FPGA.

With SONF and 320 bits of ROM size for storing the differential LUT (only 2% compared with conventional LUT), this new method will reduce the differences in outputs of sigmoid function between hardware-based and software-based ANN up to 0.0022 at maximum. Accuracy achieved by the proposed idea is 10 times better than that of using only SONF and 2 times better than that of using conventional LUT that uses 16 Kbits of ROM. The proposed idea can be implemented in various applications of real-time hardware-based ANN with higher speed and accuracy.

ACKNOWLEDGEMENTS

This work was supported by Universiti Malaysia Pahang under grant no. RDU1303116.

REFERENCES

- [1] T. Orlowska-Kowalska and M. Kaminski. 2011. FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system. IEEE Trans. Ind. Informatics, vol. 7, no. 3, pp. 436–445.
- [2] M. Krips, T. Lammert, and Kummert. 2002. FPGA implementation of a neural network for a real-time hand tracking system. Proc. First IEEE Int. Work. Electron. Des. Test Appl., pp. 313–317.
- [3] V. Saichand, N. D. M., A. S., and N. Mohankumar. 2008. FPGA Realization of Activation Function for Artificial Neural Networks. Eighth Int. Conf. Intell. Syst. Des. Appl., pp. 159–164.



- [4] T. M. Jamel and B. M. Khammas. 2012. Implementation of a Sigmoid Activation Function for Neural Network Using FPGA. In: 13th Scientific Conference of Al-Ma'moon University College.
- [5] C. P. U. Rückert, A. Funke. 1993. Accelerator board for neural associative memories. *Neurocomputing*, vol. 5, no. 1, pp. 39–49.
- [6] P. Ferreira, P. Ribeiro, A. Antunes, and F. M. Dias. 2007. A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function. *Neurocomputing*, vol. 71, no. 1–3, pp. 71–77.
- [7] K. P. Lakshmi and M. Subadra. 2013. A Survey on FPGA based MLP Realization for On-chip Learning. *Int. J. Sci. Eng. Res.*, vol. 4, no. 1, pp. 1–9.
- [8] L. Kim, S. Asaad, and R. Linsker. 2014. A Fully Pipelined FPGA Architecture of a Factored Restricted Boltzmann MCHINE Artificial Neural Network. vol. 7, no. 1.
- [9] E. Z. Mohammed and H. K. Ali. 2013. Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array. *Int. J. Comput. Theory Eng.*, vol. 5, no. 5, pp. 780–783.
- [10] B. K. Bose. 2007. Neural Network Applications in Power Electronics and Motor Drives - An Introduction and Perspective. *Ind. Electron. IEEE Trans.*, vol. 54, no. 1, pp. 14–33.
- [11] Michael Negnevitsky. 2006. Artificial Intelligence - A Guide to Intelligence Systems. In: *Artificial Intelligence-2nd Edition*, Addison Wesley, pp. 165 – 170.
- [12] T. G. Tan, J. Teo, and Patricia Anthony. 2014. A comparative investigation of non-linear activation functions in neural controllers for search-based game AI engineering. *Artif. Intell. Rev.*, vol. 41, no. 1, pp. 1–25.
- [13] G. Dreyfus. 2005. *Neural Networks: Methodology and Applications*. Springer Berlin Heidelberg.
- [14] I. Aybay. 1996. Classification of neural network hardware. *Neural Netw. World, IDG Co.*, vol. 6, no. 1, pp. 11 – 29.
- [15] Sahin and I. Koyuncu. 2012. Design and implementation of neural networks neurons with radbas, logsig, and tansig activation functions on FPGA. *Electron. Electr. Eng.*, vol. 4, no. 4, pp. 51–54.
- [16] W. Yonggang, D. Junwei, Z. Zhonghui, Y. Yang, Z. Lijun, and P. Bruyndonckx. 2–11. FPGA based electronics for PET detector modules with neural network position estimators. *IEEE Trans. Nucl. Sci.*, vol. 58, no. 1 PART 1, pp. 34–42.
- [17] J. Juang, L. Chien, and F. Lin. 2011. Automatic Landing Control System Design Using Adaptive Neural Network and Its Hardware Realization. *System Journal*, vol. 5, no. 2, pp. 266–277.
- [18] S. Himavathi, D. Anitha, and Muthuramalingam. 2007. Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization. *IEEE Trans. Neural Networks*, vol. 18, no. 3, pp. 880–888.
- [19] X. Chen, G. Wang, W. Zhou, S. Chang, and S. Sun. 2006. Efficient Sigmoid Function for Neural Networks Based FPGA Design. In: *International Conference on Intelligent Computing, ICIC* pp. 672–677.
- [20] M. T. Tommiska. 2003. Efficient digital implementation of the sigmoid function for reprogrammable logic. *IEE Proc. - Comput. Digit. Tech.*, vol. 150, no. 6, p. 403.
- [21] K. Basterretxea, J. M. Tarela, and I. del Campo. 2004. Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. *IEE Proc. Circuits, Devices Syst.*, vol. 151, no. 1, p. 18.
- [22] H. Faiedh, Z. Gafsi, and K. Besbes. 2003. Digital hardware implementation of sigmoid function and its derivative for artificial neural networks. *ICM 2001 Proceedings. 13th Int. Conf. Microelectron.*
- [23] M. Zhang, S. Vassiliadis, J. G. Delgado-Frias, and S. V. and J. G. D. F. M. Zhang. 1996. Sigmoid generators for neural computing using piecewise approximations. *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049.
- [24] Z. Xie. 2012. A non-linear approximation of the sigmoid function based on FPGA In: *IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI)*, pp. 221–223.
- [25] R. W. Duren, R. J. Marks, P. D. Reynolds, and M. L. Trumbo. 2007. Real-time neural network inversion on the SRC-6e reconfigurable computer. *IEEE Trans. Neural Networks*, vol. 18, no. 3, pp. 889–901.
- [26] A. Muthuramalingam, S. Himavathi, and E. Srinivasan. 2008. Neural Network Implementation Using FPGA: Issues and Application. *Int. J. Inf. Technol.*, vol. 4, no. 2, pp. 396–402.



www.arnjournals.com

- [27] F. Piazza, A. Uncini, and M. Zenobi. 1993. Neural networks with digital LUT activation functions. In: Proceedings of International Joint Conference on. Vol. 2. IEEE, no. 2, pp. 1401–1404.
- [28] A. R. Omondi and J. C. Rajapakse. 2002. Neural networks in FPGAs. In: Proceedings of the 9th International Conference on Neural Information Processing, ICONIP, vol. 2, pp. 954–959.
- [29] A. Armato, L. Fanucci, G. Pioggia, and D. De Rossi. 2009. Low-error approximation of artificial neuron sigmoid function and its derivative. *Electron. Lett.*, vol. 45, no. 21, p. 1082.
- [30] J. E. Volder. 1959. The CORDIC Trigonometric Computing Technique. *Electron. Comput. IRE Trans.*, vol. EC-8, no. 3, pp. 330–334.
- [31] Y. H. Hu and S. Naganathan. 1990. Novel implementation of a chirp Z-transform using a CORDIC processor. *IEEE Trans. Acoust.*, vol. 38, no. 2, pp. 352–354.
- [32] M. P. K. Chandrakanth Y, “Low Latency & High Precision CORDIC Architecture Using Improved Parallel Angle Recoding,” in International Conference on Signal Processing, Communication, Computing and Networking Technologies, 2011, no. Icscn, pp. 498–501.
- [33] D. J. Myers and R. a. Hutchinson. 1989. Efficient implementation of piecewise linear activation function for digital VLSI neural networks. *Electron. Lett.*, vol. 25, no. 24, p. 1662.
- [34] H. Amin, K. M. Curtis, and B. R. Hayes-Gill. 1997. Piecewise linear approximation applied to nonlinear function of a neural network. In: *Circuits, Devices and Systems, IEE Proceedings*, vol. 144, no. 6, pp. 313–317.
- [35] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi. 2009. Efficient hardware implementation of the hyperbolic tangent sigmoid function. *IEEE Int. Symp. Circuits Syst.*, pp. 2117–2120.
- [36] S. Hariprasath and T. N. Prabakar. 2012. FPGA Implementation of Multilayer Feed Forward Neural Network Architecture using VHDL. In: International Conference on Computing, Communication and Applications (ICCCA), pp. 1–6.