Sensors & Transducers, Vol. 227, Issue 11, November 2018, pp. 67-72

www.sensorsportal.com

Sensors & Transducers

Published by IFSA Publishing, S. L., 2018
http://www.sensorsportal.com

Power and Area Efficient Intelligent Hardware Design for Water Quality Applications

[1] Abheek GUPTA, [2] Anu GUPTA and [3] Rajiv GUPTA
[1, 2] Department of Electrical and Electronics Engineering, BITS Pilani, 333031, India
[3] Department of Civil Engineering, BITS Pilani, 333031, India
[1] Tel.: +91-7665283283
[1] E-mail: p2016423@pilani.bits-pilani.ac.in

Received: 31 July 2018 /Accepted: 28 September 2018 /Published: 30 November 2018

Abstract: The paper presents a power efficient and computationally less intensive intelligent hardware using artificial neural network for water quality applications. A compact Hardware Neural Network algorithm has been developed that takes four water quality parameters as the input vector and perform classification of the parameters using a Multilayer Perceptron Network. The computational complexity in the implementation of logistic function has been reduced at a mathematical level by use of approximation methods such as Padé approximation for exponential function and non-linear approximation for sigmoid function. The network improves accuracy of the output by learning by back-propagation of the error. Results show that non-linear approximation method is 34.13 % power efficient and utilizes 15.53 % less number of hardware resources in comparison to Padé. ASIC implementation is compact and has 99 % less power consumption as compared to FPGA implementation of the same algorithm.

Keywords: Artificial Neural network, Activation function, ASIC, Power efficient, Water quality, FPGA, Computational complexity.

1. Introduction

The aim of the paper is to present a power efficient and computationally less intensive intelligent hardware using artificial neural network for water quality applications. The whole system is designed to be power efficient and will be contained in a handheld device which gives an output in a very easily legible way by classifying the water sample being tested into one of the three classes – potable, agricultural and non-usable.

The classifications and the parameters boundaries are taken as per the standards given by the World Health Organisation Drinking Water Quality Standards [1]. A Hardware Neural Network algorithm has been developed that takes four water quality parameters, viz., pH, Oxidation Reduction Potential (ORP), Dissolved Oxygen (DO) and Total Dissolved Solids (TDS) as the input vector and perform classification of the parameters using a Multilayer Perceptron (MLP) Network.

Artificial Neural networks (ANN) have traditionally been implemented using software only or hardware-software co-design approaches for computational simplicity and accuracy [2]. A complete hardware based design, on application specific integrated circuit (ASIC) design approach, involves a trade-off between the accuracy of the calculation vs. the complexity, and power consumption.

A complete hardware based implementation of ANN has been explored by Faiedh, *et al.* [3] using

asynchronous handshaking technique which gives a delayed response. Generally ASIC design approach has proven to be computationally complex but power efficient and area required for design is also reduced drastically as the application specificity of the integrated circuit rules out unnecessary or redundant components from the IC making it suitable for a handheld device. In the present design, both ASIC and field programmable gate array (FPGA) based design methodologies are implemented and performance is compared.

ANNs are most commonly expected to perform one of the four tasks:

- *Classification* of input data sets into predefined classes.

- *Prediction* of future output based on a set of current input-output mappings.

- *Clustering* data into groups based on prior knowledge about the data.

- An ANN can be trained to remember particular data patterns and then *associate* input data pattern with the ones in the memory or discard the data pattern.

An ANN neuron is modelled as:

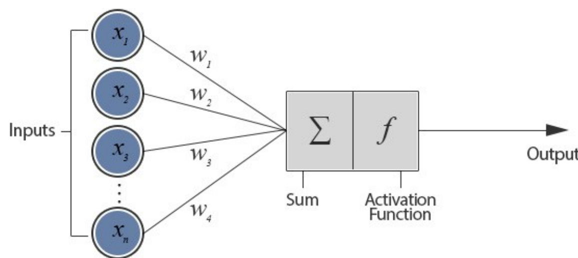

Fig. 1. Model of a neuron of an ANN. Image Source: [4].

The input points on above figure are analogous to synaptic connections on a nerve cell. For an n-dimension input vector $[x_1, x_2, x_n]$ each input is multiplied by a synaptic weight $[w_{1,}w_2, \ldots\ldots\ldots w_n]$. These products are hereafter summed up in the nerve centre and the final sum is passed through an activation function (a threshold function, stepwise linear function or sigmoid function) that defines the final output of the neuron. A multitude of such neurons form a layer of parallel processing centres which can work on a huge range of inputs. The outputs of one such layer of neurons serves as the input to many such subsequent layers of neurons, thus implementing a huge parallel processing capability to the system. The outputs are then compared with the expected outputs and the errors are measured. The weights of the synapses are thus altered in accordance with the error. This is the basic learning process of a neuron.

Fig. 2 shows a typical architecture for an ANN.

- **Input Layer** – It contains those neurons that receive the input data that is to be processed by the ANN.

- **Hidden Layer(s)** – These are the layers of units between the input and the output layers. The hidden layers take the data from the input layer and perform the computations such that it can give useful information to the other hidden or output layer.

- **Output Layer** – This consists of the neurons or units that give output depending on the learning that has taken place inside the ANN.

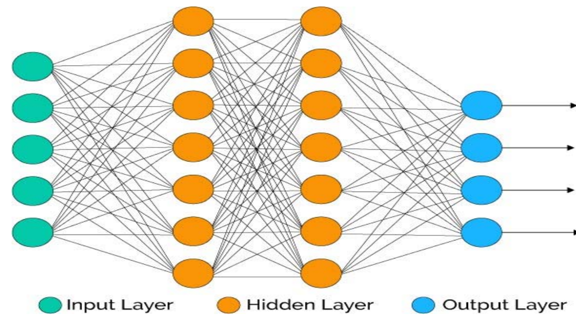

Fig. 2. A typical ANN Architecture; Image Source [13].

2. ANN Implementation

2.1. Software vs. Hardware

Traditionally, ANNs have been implemented on software which requires a complete processing system. That means a lot of unnecessary hardware is engaged but not utilised. Also when the software code has to decode down to a hardware level to be able to interact with the real world, it has to go through multiple abstractions of computer architecture. This renders the process to become time consuming and also limits the usability of the ANN. Thus, an Application Specific Integrated Circuit (ASIC) or System-on-Chip (SoC) based implementation of an ANN algorithm makes the system much more resource and time efficient because of the application specific nature of the hardware. A direct hardware implementation also improves the speed of the system. An application specific IC is also economical because of its specificity.

Hardware implementation of ANN's offer a simpler development cycle for powerful machine intelligence. Application specific nature of these hardware offer better performance, but at the cost of programmability. Moreover, physical resource (Silicon area) utilisation is also minimal in this approach. The trade-off between application specificity and program flexibility is a part and parcel of VLSI design.

While implementing a logic on hardware, there are two approaches: - *Analog and Digital.* Morgan*, et al.* [5] present the key advantages that a digital approach has over analog.

Biological mechanisms which function on inaccurate components serve as models for ANN algorithms. Thus, analog circuits having infinite resolution (continuous sampling) should serve as the better implementation option. Practically though, there is a limit to the representation of the smallest resolution on a circuit. i.e. the sampling rate of a circuit

must be finite. Moreover, a multitude of ANN algorithms model biology very poorly and thus frequently need wide dynamic range to bring about convergence to useful solutions. Popular stochastic algorithms such as backpropagation (BP) require 12 – 16 bit of range or roughly four orders of magnitude between the largest weight value and the smallest weight change [6-7]. Resolution required to achieve convergence cannot be obtained from analog circuits [5].

Calculation of functions can be done by device physics by clever analog designing, but these designs are more dependent on circuit size than scalable CMOS (Complementary Metal Oxide Semiconductor) digital designs. Particularly, thermal noise power can be assumed to be proportional to inverse of the length unit (λ). Thus, for very small circuits the noise levels can be very high and render the circuit unreliable. Hence, as circuits sizes decrease, digital approach offers better prospects for performance improvement.

Connectivity between elements on a chip is a big limitation. Particularly, for ANN's, they require huge multiplexing. Multiplexing analog signals is a design and resource intensive process. Interconnecting analog circuits at board level is pretty tricky because of radiated noise, power supply isolation problems, crosstalk, etc. Digital connections, on the other hand, tend to be rather reliable, comparatively. Since, a neural network generally consists of more than one neuron, crosstalk needs special consideration when implementing computational multiplexing. Digital circuitry demands more Silicon area.

When implementing changes in algorithm, analog circuit requires major redesign efforts. Digital circuits can accommodate such changes by implementing a different logic array.

Since most other computational units are digital, nowadays. A digital circuit also improves compatibility to other systems, as compared to an analog circuit. Also ANN's are subject to Amdahl's Law[1], as per which, the speed improvements of an analog neuron will stay untapped if the remaining network is implemented by slower digital implementations.

2.2. ANN Architectures

On most neural networks, each neuron in a hidden layer is connected to each unit in the previous (input) layer and the subsequent (output) layers. ANNs can be implemented in a number of architectures. In 1943, McCulloch and Pitts presented the first ever model of an artificial neuron, called the perceptron [8]. A layer of perceptrons can perform some tasks. Thus a single layer of perceptrons can form a network. We term such a network as a Single Layer Perceptron. An arrangement of a series of a Single Layer Perceptron is called a Multi-Layer Perceptron (MLP). MLPs are also called feedforward networks.

2.3. Activation Function Design

The most computationally complex part in hardware implementation of an ANN is the implementation of the activation function as it involves complex non-linear mathematical calculations.

Various techniques have been used in proposed ANN design to cut down on computational complexity in order to achieve less power consumption. Accuracy is improved by the learning cycle of the network.

The computational complexity in the implementation of logistic function has been reduced at a mathematical level by use of approximation methods such as Padé approximation for exponential function and non-linear approximation for sigmoid function.

(1) In first method, we use logistic activation function Eq. (1) as the activation function implemented using IEEE 754 floating point representation for Multi-Layer Perceptron architecture.

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

Padé approximation Eq. (2), as proposed by ZbigniewHajduk's [9] work has given fairly accurate network outputs despite compromising marginally on mathematical accuracy as compared to other expansions such as Taylor series or McLaurin series. However, the Padé approximation for exponential function is valid only for the input values lying in the interval $0 \leq x \leq 1$.

$$e^x = \frac{1680+840x+180x^2+20x^3+x^4}{1680-840x+180x^2-20x^3+x^4} \quad (2)$$

This limits the application of the approximation for our project. Fig. 3, shows schematic diagram for Padé approximation using Look up table (LUT) blocks of Xilinx Zynq700 board.

(2) In second method, we implement an approximation of the Sigmoid function as proposed by Zhenzhen Xi [10]. Herein, a Look up table (LUT) based approach is followed, which is particularly suited for FPGA implementation of the sigmoid function. The total domain of the sigmoid function is broken up into shorter intervals and the curve in those intervals is approximated by curve fitting method to simpler polynomials. For our application, since we normalise the input values in the range of [-1, 1], hence, we take up the intervals [-2, -1], (-1, 1)

[1]Amdahl's Law – The performance gain of a faster execution mode is limited to the time for which the execution mode is being used.

and [1, 2). The polynomials for the said intervals are given Eq. (3), Eq. (4) and Eq. (5), respectively:

$$y = 0.0467 * x^2 + 0.1239 * x + 0.2969 \quad (3)$$

$$y = 0.2383 * x + 0.5 \quad (4)$$

$$y = -0.0467 * x^2 + 0.2896 * x + 0.4882 \quad (5)$$

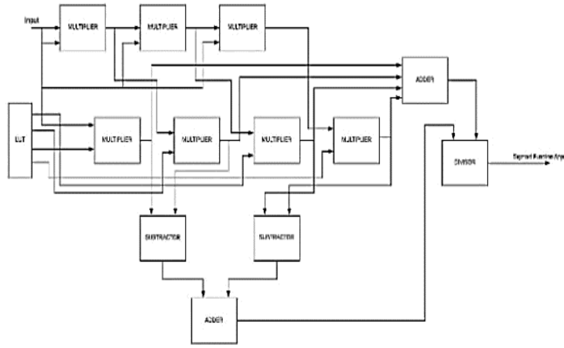

Fig. 3. Schematic Diagram of neuron using Padé Approximation.

Fig. 4 above shows schematic diagram for Non-linear approximation using the LUT blocks of a Xilinx Zynq 7000 series board.

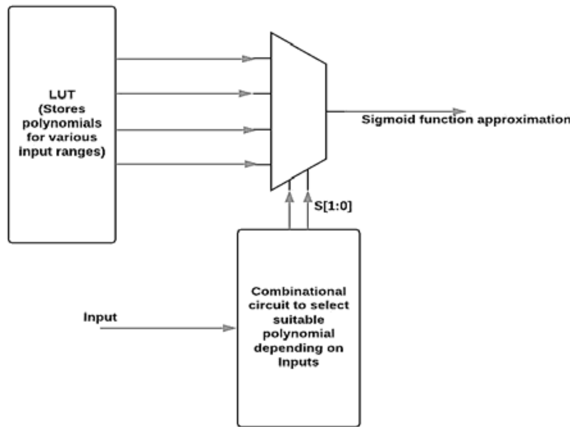

Fig. 4. Schematic diagram of for nonlinear approximation.

2.4. Learning Algorithm

The learning algorithm for this network is chosen to be backpropagation. As propounded by Rumelhart, *et al.* [11], the aim of backpropagation is to obtain a set of weights so as to minimise the difference between the desired output and the actual output of each neuron, given a particular input vector. The total error E is given by:

$$E = \frac{1}{2}\sum_{c}\sum_{j}(y_{j,c} - d_{j,c})^2, \quad (6)$$

where c indicates the case (input-output pair), and j indicates the output unit, y is the actual state of the output unit while d is the desired state of that output unit. The partial derivative of E with respect to each weight in the network is computed to minimise the error using gradient descent algorithm. The partial derivative is simply the sum of the partial derivatives of each case. Thus, we calculate $\partial E/\partial y$ for each case:

$$\frac{\partial E}{\partial y_j} = y_j - d_j \quad (7)$$

Applying chain rule to compute $/\partial x_j$:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{dx_j} \quad (8)$$

Differentiating equation (1) for dy_j/dx_j we get:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot y_j(1 - y_j) \quad (9)$$

The total input is a linear function of the states of the previous levels and also a linear function of the weights on the connections. Hence, it becomes easy to compute how the error is affected by a change in the state and the weights. Given a weight w_{ji}, from i to j the derivative is given as:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ji}} = \partial E/\partial x_j \cdot y_i \quad (10)$$

And the contribution of $\partial E/\partial y_i$ for the ith output unit because of the effect of i on j is:

$$\frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial y_i} = \partial E/x_j \cdot w_{ji} \quad (11)$$

Considering all the synapses emanating from i, we get:

$$\frac{\partial E}{\partial y_i} = \sum_{j} \frac{\partial E}{x_j} \cdot w_{ji} \quad (12)$$

So now we have the $\partial E/\partial y$ for a unit in penultimate layer, given the $\partial E/\partial y$ for all the output neurons. To compute this term for previous layers, weuse the same principle successively while computing $\partial E/\partial w$ for the weights.

$\partial E/\partial w$ is used to change the weights in every epoch. The $\partial E/\partial w$ over the network are accumulated and then at the end of the epoch, we change the weights of each neuron by an amount proportional to the accumulated $\partial E/\partial w$:

$$\Delta w = -\varepsilon \partial E/\partial w \quad (13)$$

As stated by Rumelhart, the method has a low convergence rate as compared to methods that make use of second derivative but it is easy to implement on parallel hardware. The method is improvised by accelerating method where current gradient modifies

the velocity of the point in weight space instead of its position:

$$\Delta w(t) = -\frac{\varepsilon \partial E}{\partial w(t)} + \alpha \Delta w(t-1) \quad (14)$$

where t denotes the time-step or epoch and α denotes an exponential decay factor between 0 and 1 that controls contribution of current and previous gradients to the change in weights. Thus, α is also called the learning index.

3. Power Efficiency and Accuracy

The dynamic power consumption of a single neuron is reduced by replacing sequential units like counters and MAC units by parallel multipliers and adders that perform the calculation in parallel. This also reduces the requirement of storage elements within each neuron.

The network improves accuracy of the output by learning by back-propagation of the error. As the final output of one epoch is generated, an FSM is designed that would generate a signal to start a back-propagation algorithm [12]. This algorithm measures the difference between the desired output and the actual output. The error in the final output is back propagated to all the neurons in the preceding layer while the weights of the current layer are updated. The control of the learning mechanism being synchronised using an FSM gives a time multiplexing to the learning mechanism so as to reduce switching power consumption of the complete network.

4. Results and Comparison

The algorithms have been coded in Verilog and implemented on ZynQ7000 FPGA using Xilinx Vivado. The results of the implementation are shown in Table 1.

Table 1. Comparison of two implementations of Activation Function in FPGA based design.

Implemen-tation	MUX	LUT	DSP	Power (FPGA) (Watts)	Power (ASIC) (Watts)
Padé approximation	15	3578	18	5.95	3.75×10^{-4}
Non-linear approximation	14	3144	12	5.3	2.47×10^{-4}

The power consumption of the FPGA implementation were very high. Thus, we synthesized the design using ASIC design methodology. The ASIC was synthesized using Cadence Encounter RTL Compiler tool with UMC 90 nm standard cell library. The power consumption of a neuron using Padé approximation dropped from **5.95 Watts on FPGA to 3.75×10^{-4} Watts in ASIC synthesis**. Similarly for the Non-linear approximation method the power consumption of a single neuron drops from **5.3 Watts on FPGA to 2.47×10^{-4} Watts for ASIC synthesis.**

Padé approximation method makes use of 25538 Cells covering 241897 nm^2 of the library as compared to 21571 cells covering 196997 nm^2 for nonlinear approximation method.

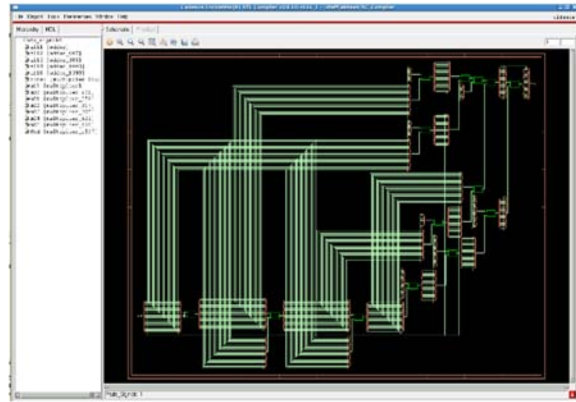

Fig. 5. ASIC implementation of Padé approximation.

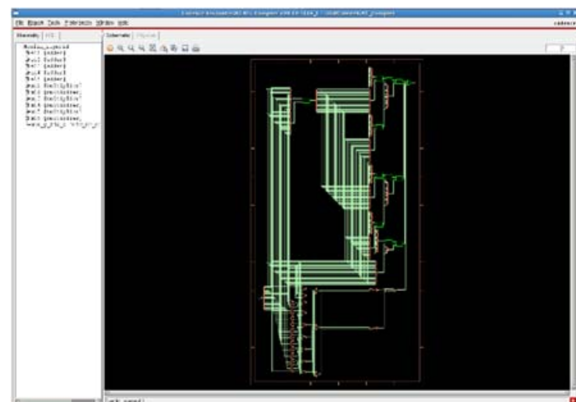

Fig. 6. ASIC Implementation of a Nonlinear Approximation of Sigmoid function.

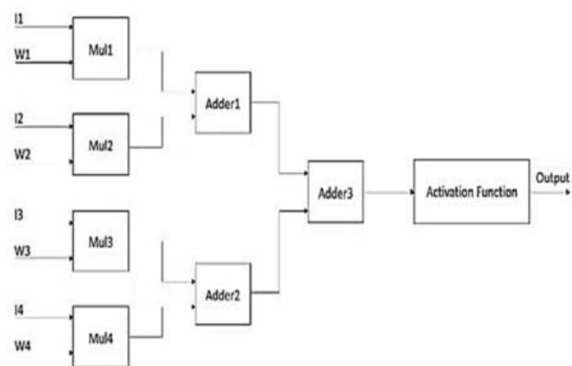

Fig. 7. Basic structure of a Neuron.

5. Conclusions

From the results shown in Table 1, non-linear approximation method is found to be more suited to proposed design for water quality application. It also is much more power efficient and utilises much less number of hardware resources. However, FPGA implementations are consuming much higher power. Thus we have changed to a compact ASIC implementation of the network designed. The ASIC implementation shows drastic drops in power consumption of the system as compared to FPGA implementation.

Acknowledgements

Financial support for project entitled "Conjunctive use of Rainwater and Groundwater to provide safe drinking water with intervention of Advanced Technology" No. DST/TM/WTI/2K16/204(G), Government of India, Ministry of Science and Technology, Department of Science and Technology, New Delhi, India.

References

[1]. World Health Organization, Guidlines for Drinking-water Quality, 4th edition, Incorporating the First Addendum, *World Health Organization*, Geneva, Switzerland, 2017.

[2]. P. Ferreira, P. Ribeiro, A. Antunes, F. M. Dias, Artificial Neural Networks Processor – A Hardware Implementation Using a FPGA, in *Proceedings of the 14th International Conference on Field Programmable Logic and Application (FPL'04)*, Leuven, 2004.

[3]. H. Faiedh, Z. Gafsi, K. Torki, K. Besbes, Digital hardware implementation of a neural network used for classification, in *Proceedings of the IEEE 16th International Conference on Microelectronics Proceedings (ICM'04)*, Tunis, Tunisia, 2004, pp. 16-19.

[4]. Introduction to Artificial Neural Network (ANN) | secret mind control in Sweden and worldwide, *mindcontrolinsweden.wordpress.com*, 30 January 2015. Available: https://mindcontrolinsweden.wordpress.com/2015/01/30/introduction-to-artificial-neural-networks/

[5]. N. Morgan, K. Asanovic, B. Kingsbury, J. Wawrzynek, Developmets in Digital VLSI Design for Artificial Neural Networks, *University of California at Berkeley*, Berkeley, California, 1990.

[6]. N. Morgan, Artificial Neural Networks: Electronic Implementations, *IEEE Computer Society Press*, Washington, *D.C.*, 1990.

[7]. T. Baker, D. Hammerstrom, Modifications to Artificial Neural Networks Models for Digital Hardware Implementation, *CSETech,* Department of Computer Science and Engineering, Oregon Graduate Centre, Washington County, Oregon, USA, 1988.

[8]. W. S. McCulloch, W. Pitts, A logical calculus of ideas immanent in nervous activity, *The Bulletin of Mathematical Biolphysics,* Vol. 5, Issue 4, 1943, pp. 115-133.

[9]. Z. Hajduk, High accuracy FPGA activation function implementation for neural networks, *Neurocomputing,* Vol. 247, 2017, pp. 59-61.

[10]. Z. Xie, A Non-linear Approximation of the Sigmoid Function based on FPGA, in *Proceedings of the IEEE 5th International Conference on Advanced Computational Intelligence (ICACI'12),* 2012, pp. 221-223.

[11]. D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, *Letters to Nature,* Vol. 323, 1986, pp. 533-536.

[12]. A. Savich, M. Moussa, S. Areibi, A scalable pipelined architecture for real-time computation of MLP-BP neural networks, *Microprocessors and Microsystems,* Vol. 36, Issue 2, 2012, pp. 138-150.

[13]. Navdeep Singh Gill, Overview of Artificial Neural Networks and its Applications, *Xenonstack*: *A Stack Innovator*, 2017. Available: https://www.xenonstack.com/blog/overview-of-artificial-neural-networks-and-its-applications

___

Published by International Frequency Sensor Association (IFSA) Publishing, S. L., 2018 (http://www.sensorsportal.com).