Microprocessors and Microsystems 39 (2015) 373–381

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

Hardware implementation of neural network with Sigmoidal activation functions using CORDIC

Vipin Tiwari *, Nilay Khare

Department of Computer Science & Engineering, MANIT, Bhopal, India

ARTICLE INFO

Article history:
Available online 8 June 2015

Keywords:
CORDIC
Field Programmable Gate Array (FPGA)
Hardware
Neural network

ABSTRACT

Activation function is the most important function in neural network processing. In this article, the field-programmable gate array (FPGA)-based hardware implementation of a multilayer feed-forward neural network, with a *log* sigmoid activation function and a *tangent* sigmoid (*hyperbolic tangent*) activation function has been presented, with more accuracy than any other previous implementation of a neural network with the same activation function. Accuracy is enhanced through the implementation of both the sigmoidal functions using COordinate Rotation DIgital Computer (CORDIC) algorithm. The CORDIC algorithm is a simple and effective method for calculation of the *trigonometric* and *hyperbolic* functions. Simulations and experiments have been performed on the ISim simulation engine of the Xilinx Framework, using the Very High Speed Integrated Circuit Hardware Description Language (VHDL) as the programming language. The results show accuracy for a 32-bit and 64-bit input/output, compromising with speed.

1. Introduction

Accuracy and speed are two important issues to be considered when designing a system. Maximum speed can be accomplished by using hardware rather than a practicing software, as the software is platform and operating-system dependent. Accuracy depends on the algorithm used and the availability of computer memory. In recent years Neural Networks have attracted the attention of many researchers owing to its power of parallel and distributed computation, learning and fault tolerance. Many software implementations of neural networks have been reported to date, which are time consuming and platform-dependent being run on. The maximum speed can be achieved using hardware with a suitable algorithm. Hardware is particularly useful in applications where speed and accuracy matter, such as medical science and many others. Neural hardware is especially useful in image and text processing and in classification and clustering applications.

The COordinate Rotation DIgital Computer (CORDIC) algorithm, first proposed by Volder [1], is a simple and efficient method to calculate trigonometric and *hyperbolic* function through coordinate transformation. Although the CORDIC algorithm is used especially where computation involves trigonometric calculation, with minor modifications, this algorithm can be used to calculate other useful operations such as arithmetic operations, exponentiation, and logarithmic and hyperbolic function calculations, as listed by Andraka in [2]. Subsequently, with the role of *pseudo* multiplication and *pseudo* division, as proposed by Meggitt [3], this algorithm has become the base algorithm for the HP-35 first scientific calculator designed by Cochran [4].

The remaining article has been organized as follows. Sections 2 and 3 give a brief introduction of the neural networks and the CORDIC algorithm, respectively. Section 4 reviews the previous approaches to hardware implementation of neural networks. Section 5 proposes a new methodology to implement neural networks, Section 6 shows the results of experiments, and finally Section 7 concludes the study.

2. Artificial neural network

An Artificial Neural Network (ANN) is an information paradigm that is inspired by the biological nervous system, such as the brain. The main feature of this is the unique structure of all information processing units, known as neurons, forming a network to solve a particular problem. ANN attempts to imitate the working of the human brain in a digital environment rather than using algorithmic procedures. Similar to a human brain, ANN provides a massive parallel distributed processing environment that has the innate

* Corresponding author.
E-mail addresses: vipintiwari1@gmail.com (V. Tiwari), nilay.khare@rediffmail.com (N. Khare).

http://dx.doi.org/10.1016/j.micpro.2015.05.012
0141-9331/© 2015 Elsevier B.V. All rights reserved.

capability of storing and retrieving experimental knowledge and data, and making it available whenever necessary. The ANN acquires knowledge through a learning (or training) process and stores this knowledge in the form of synaptic weights (weighted links between the neurons of different or same layers). This stored knowledge is used as an experience when solving a problem.

2.1. Neural network architecture

The architecture of the ANN is defined as the organization of neurons into layers and the pattern of connections within and in between layers. The neurons within a layer may be fully or partially interconnected. The number of layers of the weighted interconnections indicates the number of layers in the ANN. An ANN that has a single layer of weighted interconnection is known as a simple network, with only one input and one output layer. If two layers of weighted interconnections are present, then the Neural Network (NN) will definitely have single hidden layers. Likewise, other NNs may have bodies of different sizes and materials, according to the problem they are going through. A model of single neurons is shown in Fig. 1.

3. CORDIC algorithm

The CORDIC algorithm is a simple and iterative convergence algorithm that offers effective and low-cost implementation of a variety of applications, as discussed in Section 1. The general equations of the CORDIC algorithm are as follows.

$$x_{i+1} = x_i - m\sigma_i y_i \rho^{-S_{m,i}}$$
$$y_{i+1} = y_i + \sigma_i x_i \rho^{-S_{m,i}}$$
$$z_{i+1} = z_i - \sigma_i \alpha_{m,i}$$

where σ_i represents the direction of micro-rotation, either clockwise or anticlockwise, ρ represents the radix of the number system, m represents the type of coordinate system, *circular (m = 1), linear (m = 0)* or *hyperbolic (m = –1)*. $S_{m,i}$ is the non-decreasing integer shift sequence and $\alpha_{m,i}$ is the elementary rotation angle.

CORDIC works in two modes, namely, the rotation mode and vectoring mode.

In the *rotation* mode, the given vector is rotated through an angle θ, where θ is decomposed using a finite number of small elementary angles

$$\theta = \sigma_0\alpha_0 + \sigma_1\alpha_1 + \sigma_2\alpha_2 + \ldots\ldots + \sigma_{n-1}\alpha_{n-1}$$

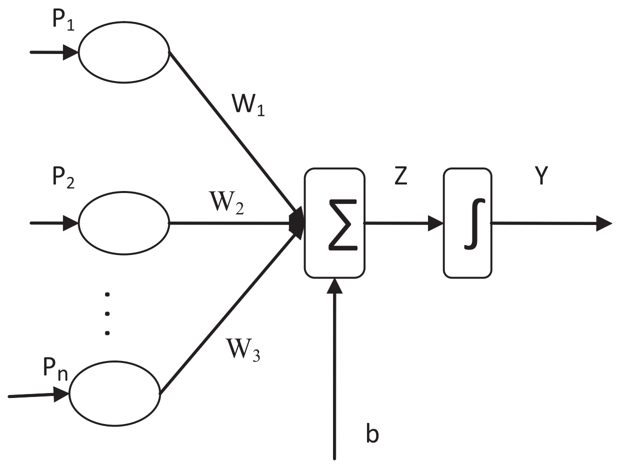

Fig. 1. Architecture of a single neuron.

where n is the number of micro-rotations, α_i is the elementary angle of the ith micro-rotation, and σ_i is the direction of the angle in the ith micro-rotation. In the *rotation* mode, the angle accumulator z_0 is initialized with the input rotation angle. In every iteration the direction of the angle is determined, to bring down the magnitude of the residual angle in the angle accumulator, using the sign of the residual angle obtained in the former iteration. The coordinates of the vector after n micro-rotations are,

$$x_n = K(x_{in}\cos\theta - y_{in}\sin\theta),$$
$$y_n = K(x_{in}\sin\theta + y_{in}\cos\theta),$$
$$z_n \rightarrow 0$$

In the *vectoring* mode, the unknown angle of a vector is computed by performing a finite number of micro-rotations satisfying the relation,

$$-\theta = \sigma_0\alpha_0 + \sigma_1\alpha_1 + \sigma_2\alpha_2 + \ldots + \sigma_{n-1}\alpha_{n-1}$$

The vectoring mode rotates the input vector through a predetermined set of n elementary angles, to reduce the y coordinate of the final vector to zero, as closely as possible. The direction of the rotation is determined based on the sign of the residual y coordinate obtained in the previous iteration. After n micro-rotations, the coordinates in the vectoring mode are given by

$$x_n = K\sqrt{x_{in}^2 + y_{in}^2}$$
$$y_n \rightarrow 0$$
$$z_n = \tan^{-1}\left(\frac{x_{in}}{y_{in}}\right)$$

4. Related work

Many researchers have been working on neural networks and they have implemented these networks using both software and hardware. As hardware is faster than its software counterpart, nowadays, greater focus is on hardware implementation. NNs have been implemented on both digital and analog platforms, but because digital circuits are easy to design, cheaper, and have noise immunity, they are preferred over analog implementation. On account of the fact that they are reconfigurable, there is a great focus on FPGA-based hardware realization of neural networks at present. In [5], an architecture Refined Manufacturing Acceleration Process Network (REMAP) has been introduced, where a highly parallel neural computer has been implemented using only FPGAs. In [6], using FPGA, a high precision neural network processor has been introduced. An FPGA implementation of ANN with learning capability has been presented in [7]. Learning capability has been provided with particle swarm optimization (PSO). Authors in [8] use a network of systolic arrays for real-time enhancement of geospatial and aerial remote sensing imagery. Spiking neurons have been used for hardware implementation of a Liquid State Machine (LSM) in [9]. The LSM is a recurrent neural network to implement real-time isolated digit speech recognition. In [10], an FPGA-based implementation of the Hopfield neural network has been presented. A neural stochastic model for solving the constrained NP-hard problems has been discussed in [11]. A multilayer perceptron has been implemented in [12] with the log sigmoid function as an activation function. The log sigmoid function has been approximated with a linear function, as discussed [13]. Hardware for a neural network with an on-chip learning capability and highly reconfigurable with the log sigmoid activation function has been presented in [14]. In [15], an FPGA-based neural controller of the induction machine, to observe electric vehicles has been proposed. Another FPGA-based stator flux oriented vector controller for an induction motor drive, with log sigmoid activation function has been presented in [16], where