Scalable Architecture for on-Chip Neural Network Training using Swarm Intelligence

Amin Farmahini-Farahani, Sied Mehdi Fakhraie, Saeed Safari
School of Electrical and Computer Engineering
University of Tehran, Tehran 14395-515, Iran
a.farmahini@ece.ut.ac.ir, {fakhraie,saeed}@ut.ac.ir

Abstract

This paper presents a novel architecture for on-chip neural network training using particle swarm optimization (PSO). PSO is an evolutionary optimization algorithm with a growing field of applications which has been recently used to train neural networks. The architecture exploits PSO algorithm to evolve network weights as well as a method called layer partitioning to implement neural networks. In the proposed method, a neural network is partitioned into groups of neurons and the groups are sequentially mapped to available functional units. Thus, the architecture is reconfigurable for training and implementing different multilayer feedforward neural networks without the need for modifying the architecture. The implementation is intended for real-time applications regarding hardware cost and speed. The results show that the proposed system provides a trade-off between resource requirements and speed.

1. Introduction

Artificial neural networks (ANNs) have been successfully used in a wide range of scientific and engineering applications [7,9,18]. ANNs consist of networks of simple elements called neurons which communicate with each other through weighted connections. By learning or training from examples, a neural network (NN) is capable of exhibiting intelligent behavior and modeling complex non-linear functions which makes it proper to variable conditions. Training is the process of gradually adjusting the weights of connections. Backpropagation algorithm is mainly used to train ANNs for many applications [16, 17, 19]. Since this algorithm is based on gradient descent which demands derivatives, it is complex and prone to get trapped in local optima. It also has a slow convergence rate and high resource requirement on hardware.

Particle swarm optimization (PSO) [12] is one of the evolutionary computation techniques based on swarm intelligence. PSO is a stochastic population-based search algorithm. Recently, PSO algorithm has been employed for training feedforward neural networks. In [1, 8, 14], the authors have proven the speed and accuracy of using PSO to determine neural network weights in comparison to other methods. Furthermore, PSO requires fewer computations than the other methods to obtain the same solution.

During the last two decades, hardware implementation of ANNs has become more popular among researchers due to the major limitations of software implementations. Importantly, software implementations of ANNs which run sequentially are time-consuming and cannot benefit from inherent parallelism of ANNs [2, 15]. For using ANNs in some real-time applications, hardware implementations of ANNs to reach the necessary speed is indispensable. On the other hand, hardware implementation has some drawbacks in comparison to software realization such as lack of flexibility and resource limitations. Thus, the idea of reconfigurable ANNs is introduced to cope with these problems.

Extensive hardware implementations of NNs have been reported in literature. Here, we discuss some recently presented reconfigurable implementations. Himavathi *et al.* [10] have proposed a method to implement large neural networks in FPGAs with a low resource requirement and without much compromise on the speed. However, the area cost of the proposed architecture is directly proportional to the largest layer (layer with the maximum number of neurons). Hence, the architecture is not appropriate for the networks which have a large layer. Li *et al.* [13] have presented a reconfigurable systolic approach. Their architecture is composed of several processing units performing basic neural computations. By changing the number of processing units, different neural networks are implemented. Domingos *et al.* [3] have designed an architecture for neural network training. Although memory bandwidth puts obstacles to scale the architecture, the reported performance is notable. Nevertheless, available architectures generally make use of backpropagation algorithm which can only achieve local so-

978-3-9810801-3-1/DATE08 © 2008 EDAA

lutions, while evolutionary algorithms have the potential to attain a global solution. To the best of our knowledge, none of previous work has proposed an on-chip training approach using PSO.

This paper proposes a reconfigurable architecture for on-chip neural network training using PSO algorithm. The proposed system is able to implement and train multi-layer perceptron (MLP) neural networks without regard to the network size. There are four neural network processing elements (PE) in the system performing NN computation. An NN is partitioned into at most four neurons and then it is mapped to PEs in order to deal with different neural networks. In addition, a PSO hardware core which has low resource requirement is exploited to train network weights. Section 2 gives an overview of PSO algorithm and the problem of NN training. The architecture of a neuron PE is presented in Section 3. The proposed system is described in Section 4. Section 5 explains the implementation results followed by concluding remarks in Section 6.

2. Particle Swarm Optimization

2.1. Particle Swarm Optimizer Algorithm

The PSO utilizes social interaction between agents to optimize a problem. In PSO, a potential solution, called a particle, represents a point in the search space. Each particle flies through the solution space of problem to search for the global optimum according to its own and social historical experiences. A particle consists of five components. The first component, x, is a vector that contains the current location in the solution space. $fitness$ is the quality of the solution represented by the vector x. v is a vector that contains the velocity for each vector of x. $fitness_p$ is the fitness value of the best solution encountered by a particular particle. Finally, $pbest$ is a copy of x for the location of the best solution encountered by a particular particle. Each particle is also aware of $gbest$, the particle reporting the current best fitness, and $fitness_g$, the fitness value of the $gbest$.

The heart of PSO algorithm is the process by which v is modified, forcing the particles to search through the most promising areas of the solution space. PSO concept consists of velocity changes of each particle toward its $pbest$ and $gbest$ locations. The modified velocity and location of each individual particle can be calculated using the following formulas:

$$v_{id}^{k+1} = w \cdot v_{id}^{k} + c_1 \cdot r_1 \cdot (pbest_{id} - x_{id}^{k}) + c_2 \cdot r_2 \cdot (gbest_d - x_{id}^{k}) \quad (1)$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (2)$$

where x_{id}^{k} is the current location of the particle i at the iteration k, which has v_{id}^{k+1} as the velocity and satisfies $V_{min} \leq v_{id}^{k+1} \leq V_{max}$. Besides, there are five parameters: w is the inertia weight factor, c_1 and c_2 are acceleration constants, and r_1 and r_2 are uniform random numbers between 0 and 1.

2.2. Problem Description of NN Training

In NN training, the main goal is to obtain a set of weights that minimizes mean squared error. In order to address the problem of neural network training to PSO, we represent each set of weights and biases of a network by a single particle. Thus, each particle is a string of continuous-valued numbers encoding a candidate solution for weights and biases of all neurons in the network. The length of a particle depends on the network intended to train. A pool of particles is considered as a swarm (population) for PSO. By repetitively updating particles of the swarm, the most suited network weights are gradually determined. Different stopping criteria may be used. One criterion is to update the particles until the error between actual and target outputs is lower than a given threshold. Stopping the training process after a given period without any improvement in training is the next criterion. Another criterion is training the network for a given iteration.

3. Neuron Architecture

In digital implementation of neural networks, the numerical precision of data determines a tradeoff between cost and performance. Higher precision for fixed-point representation leads to larger resource requirement, and so higher cost. While less precision causes more quantization errors that may prevent neural networks from learning. Regarding limited resources available on an FPGA, floating-point implementation is not area-efficient. According to [11], the minimum required fixed-point precision for weights is 16 bits. In order to adapt the proposed hardware for a wide range of applications, the weights and biases of the network are represented with 23 bits (1 bit for sign, 7 bits for integer part, and 15 bits for fraction part). Since the range of the sigmoid function is $[-1.0; +1.0]$, 17 bits for sigmoid LUT output data (2 bits for sign and integer part, and 15 bits for fractional part) are assigned [4].

As shown in Figure 1, the internal structure of a neuron is composed of different functional blocks which are combined to form the hardware implemented neuron. In fact, computations of MLP neurons require a multiply-and-accumulate (MAC) unit plus a sigmoid calculation unit. In the figure, MUL performs multiplication of $neuron\_input$ and $neuron\_weight$ which are 17-bit and 23-bit signed numbers. ADD performs the addition of the result of multiplication and accumulation register. Accumulation register is used to store the partial sum of the products of weights

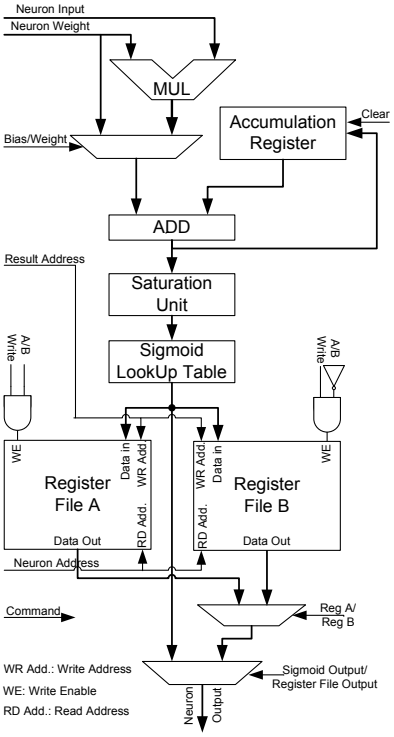

Figure 1. Architecture of a neuron processing element.

and inputs. The multiplication and addition operations are repeated with the appropriate values depending on the number of neuron inputs. After addition of the bias and accumulation register, the result is saturated and converted to a 2's-complement 10-bit data by using saturation unit and the saturated accumulation result is applied to a sigmoid calculation unit.

Here bipolar sigmoid function is used as the activation function. Sigmoid function saturates for large inputs including over +8 or under -8, which implies that bit-width for the integer part of sigmoid input can be selected as 4 bits. The bit-width for the fractional part of the sigmoid input can be selected 6 bits which provides sufficient precision [4]. Sigmoid unit produces the value for the activation function regarding to the address from the output of the saturation unit. The sigmoid unit is implemented as a lookup table to utilize the built-in RAM available in FPGA. The output of the sigmoid unit is written to one of the register files contingent on controlling signals in order to use as $neuron\_input$ for the next layer.

In this architecture, two register files are embedded in each neuron with the purpose of accelerating neural network computations. In computations of each layer of the neural network, one register file provides $neuron\_input$ values for neurons, while another register file stores the output result ($neuron\_output$) of its including neuron. Since the $neuron\_outputs$ of the neurons in the current layer are used as the $neuron\_inputs$ for the next layer, the two register files will exchange their tasks for performing the computation of the next layer using controlling signals. Finally, $neuron\_output$ is selected from newly-computed sigmoid output and previously stored values in the register files. The controlling signals of all the units are obtained by decoding the command input of the neuron.

4. Hardware Implementation of the on-Chip Training Approach

4.1. Training Methodology

We propose an on-chip training approach for feed-forward MLP neural networks. In this paper, PSO algorithm has been employed to evolve a set of weights for a given network. After achieving desired weights, the system is capable of acting as hardware-based neural network realization. It can be seen from Figure 2 that the architecture consists of four major units incorporating together. These units are PSO core, weight temporary storage, neuron mapping besides NN computation, and output error calculation.

4.2. PSO Core

A hardware particle swarm optimization core has been designed and implemented to update the particles. This unit accomplishes all processes which are related to particles and consists of diverse computational blocks. It also contains some memory blocks used to keep vectors and values of PSO such as particle location, particle velocity, and $pbest$ vectors. The internal structure of the core can be found in [5,6]. Location vectors of particles (weights) are updated using this core. Updated weights besides the particle address are forwarded to weight temporary storage unit. The core was designed to attain the throughput of one weight per clock.

4.3. Weight Temporary Storage

This unit consists of two memory blocks, a memory control block, and routing logics. The unit acts as a medium between the weight evolution unit (PSO core) and the neuron mapping unit. Memory blocks are used to temporarily hold received weights in order to feed the network. Having two memory blocks makes it possible for the architecture to operate in parallel and significantly improves the performance. While the PSO core is updating the next training set of weights, it also writes the updated weights to one of the memory blocks. As the network is being computed, neuron mapping unit reads from another memory

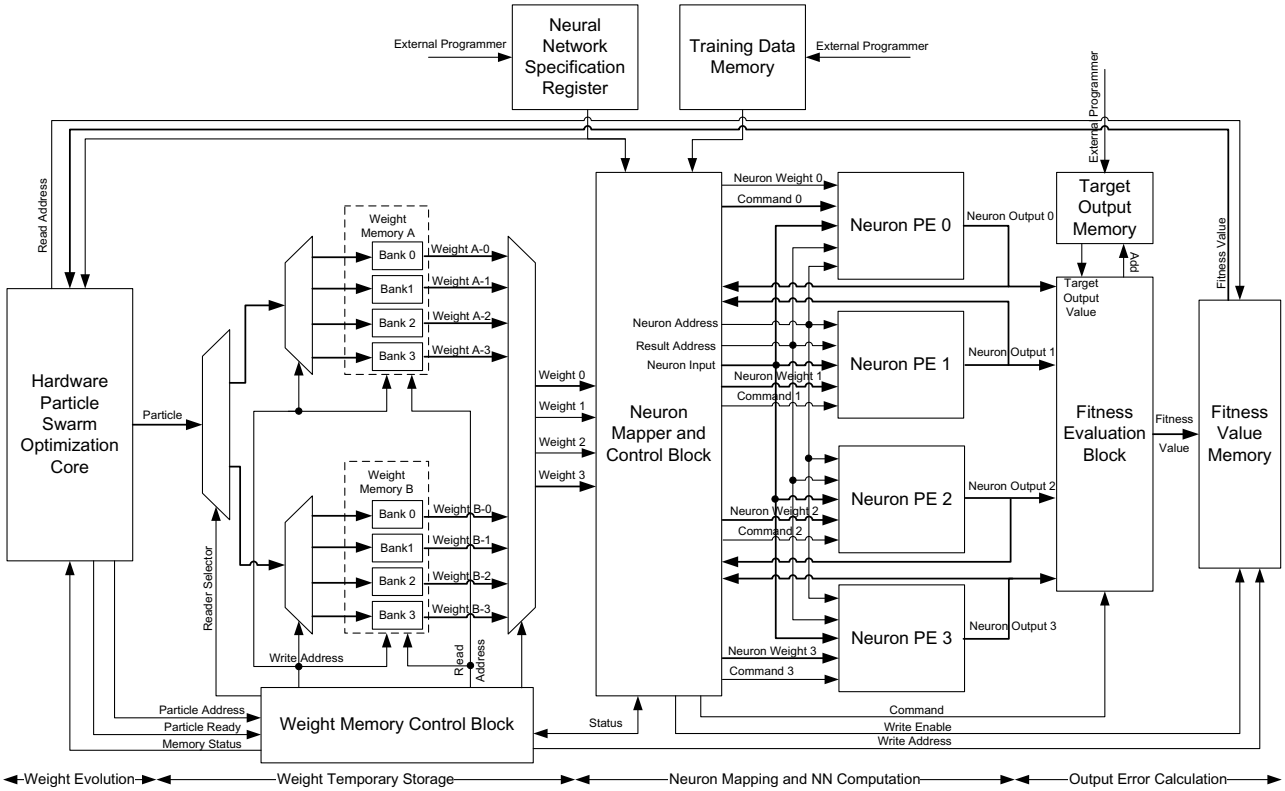

Figure 2. Block diagram of the neural network training architecture.

block which holds the previously computed weights. Each weight memory block has four memory banks. Thus, all neuron PEs are able to read their respective weights concurrently. Weight memory control block is responsible for writing received weights to the appropriate memory block and memory banks. Moreover, it computes read and write addresses according to the status signals received from the adjacent units and places the correct set of weights to neuron PEs.

4.4. Neuron Mapping and NN Computation

The realization of a multilayer network involves the implementation of all the neurons and all layers of the network. The more neurons and the more layers, the more hardware cost it requires. Although computation of the neurons in a layer can be performed in a parallel manner, computation of layers is performed sequentially as the output of each layer is the input to the next layer. Indeed, implementing the single largest layer (i.e., layer with maximum neurons) provides a good method for hardware implementation of ANN with low resource requirement. However, there are some complex and large networks needing a great number of neurons per layer. Realizing a hardware approach that embraces these large networks (i.e., network with many neurons in a layer) demands a considerable hardware cost. In addition, even if a myriad of neuron PEs are provided in hardware, a few of them are exploited to compute conventional networks. An architecture which is adaptable to different networks is presented to realize multilayer networks with minimum resource requirement. Therefore, MLP neural networks with different number of layers and a variety number of neurons per layer can be trained and implemented.

The concept of layer partitioning is proposed herein which enables the system to cope with networks of any number of neurons with the same hardware. Layer partitioning denotes the partitioning of layers and repeatedly mapping of subsets of neurons in a layer to the available functional blocks. Therefore, the layer partitioning leads to both a meaningful hardware cost reduction and a scalable architecture being reconfigurable for any networks.

In the architecture, four neuron PEs as the functional blocks are employed to perform network computations. Hence, a group of neurons in a layer is mapped to four neuron PEs and after storing the results, the next subset is mapped. The mappings are repeated until computation of all neurons in the layer is performed. Then, these steps are repeated until all layers are computed. For instance, consider a 3-9-2 network and four neuron PEs. The first layer

has three neurons, so all three neurons are mapped to the neuron PEs. The next layer containing nine neurons is partitioned into three subsets, two subsets of four neurons and a subset of one neuron. This layer is mapped in three mapping steps. Finally, the last layer is mapped.

The neuron mapping and control unit, which resembles a state machine, directs the execution of the steps of the layer partitioning. To do so, the unit reads neural network specification register which holds number of layers in network as well as number of neurons in each layer. Also, the unit reads from weight memory and training data memory (which holds the training set) and places proper weights, inputs, addresses and command signals to PEs and fitness evaluation unit.

4.5. Output Error Calculation

Fitness evaluation unit calculates fitness value of each particle and writes it to the fitness value memory block. Fitness value is considered as the cumulative error of network output(s) for the entire training set. Every neuron PE calculates its output and passes it to fitness evaluation block. The block reads the desired output from target output memory and calculates the difference (error) between the target output and actual output of the neural network. Finally, all errors are summed up and forwarded to the fitness value memory block.

5. Results

Implementation details of the proposed neural network training architecture are presented in this section. The system has been implemented on a Xilinx Virtex2P FPGA (chip xc2vp7-5fg456). These FPGAs contains embedded 18×18 Multipliers which perform accumulate functions as well as multiply operations. All blocks are developed using synthesizable Verilog hardware description language. Four different neural networks are trained in which each has various numbers of layers and neurons in order to measure the performance of the architecture and show the adaptability of the architecture to different networks. According to [8], the PSO algorithm uses parameter values $w = 0.8$, $c_1 = c_2 = 2$ in all of the networks, and the swarm size is 25. During the run of the algorithm, a maximum velocity $V_{max} = 2$ was applied for every component of the velocity vector. In this design, the maximum operating frequency is 75 MHz. The hardware cost of the proposed architecture is summarized in Table 1.

It is worth noting that due to use of layer partitioning method, the hardware area of the architecture for different networks is constant and only the required memory bits for some units (PSO core and weight temporary storage) are variable. As shown, the total resource requirement of the architecture is fewer than 2200 slices, so it is possible to add more neuron PEs to the design to speed up the computation time of complex networks.

After training the network, the architecture is able to reconfigure as a neural network realization. Table 2 shows the results of the proposed architecture for different neural networks. The first column shows the network structure. The second column represents the performance of the architecture for executing trained networks in terms of connections per second (CPS). It also shows the speed of the architecture for network training in terms of connection updates per second (CUPS) . While PSO core is updating the next training set of weights, neural PEs can perform the network computations using the weights stored in weight temporary storage. Thus, network training and weight updating phases are performed concurrently, reducing training time considerably.

Table 2 also makes a comparison between the proposed design and other designs which have been implemented on Xilinx FPGAs and benefit from higher frequency. DOM [3] and LI [13] have used backpropagation algorithm for network training and HIM [10] has not implemented any on-chip training method. Our design takes advantage of 23-bit fixed point precision and four low cost neural PEs running at 75 MHz. Although our architecture has lower performance in some cases, it has higher precision and less area cost. To gain higher performance for computational intensive networks, more neuron PEs can be added to the architecture. Using a more powerful FPGA is another way to achieve higher frequency.

6. Conclusions

A method to realize large neural networks on available hardware has been proposed which leads to a scalable and fast architecture with low area cost. Moreover, an on-chip approach for neural network training using PSO algorithm has been described. The proposed system provides a way to implement and train various networks without increasing the area cost of the architecture. The results prove the advantages of the system for real-time applications. Adding more PEs to the architecture is our future step to improve the computational power.

7. Acknowledgements

The authors would like to express their gratitude to Iran Telecommunication Research Center (ITRC) for their support during the course of this research. Amin Farmahini-Farahani would also like to thank Mehdi Kamal at Sharif University of Technology for his advice throughout this project.

Table 1. Resource Requirements of the NN Training Approach Using PSO in Xilinx Virtex2P

Unit	Slices	Slice Flip Flops	4-input LUTs	BRAMs	18×18 MULTs
PSO Core	757	605	1,073	6*	10
Four Neuron PEs	1020	796	1716	4	8
Neuron Mapping and Control	248	129	425	0	0
Weight Temporary Storage	183	148	325	8	0
Fitness Evaluation	136	112	242	0	0

* The reported memory blocks are for 8-5-5-5-5-3 network. Required memory bits are dependent on both neural network structure and training data.

Table 2. Performance Results for Four Different Networks

Network Structure	Proposed Architecture	Architectures in Literature (Fixed Point)			
	Performance	Name	Performance	Precision and Freq.	PEs
2-2-1	32 MCPS - 32 MCUPS	DOM [3]	21.4 MCPS - 8.6 MCUPS	16 bits at 100 MHz	12
25-20-10	204 MCPS - 204 MCUPS	DOM [3]	630 MCPS - 226 MCUPS	16 bits at 100 MHz	12
25-10-10	154 MCPS - 154 MCUPS	LI [13]	Not Reported - 432 MCUPS	16 bits at 100 MHz	10
8-5-5-5-5-3	112 MCPS - 112 MCUPS	HIM [10]	35 MCPS - Not Reported	17 bits at 73 MHz	5

References

[1] M. Carvalho and T. B. Ludermir. An analysis of PSO hybrid algorithms for feed-forward neural networks training. In *Proc. Brazilian Symp. on Neural Networks*, pages 2–7, Ribeirao Preto, Brazil, Oct. 2006.

[2] C.-H. Chang, M. Shibu, and R. Xiad. Self organizing feature map for color quantization on FPGA. A. R. Omondi and J. C. Rajapakse, editors, *FPGA Implementations of Neural Networks*, chapter 8, pages 225–245. Springer-Verlag New York Inc., NY, USA, 2006.

[3] P. O. Domingos, F. M. Silva, and H. C. Neto. An efficient and scalable architecture for neural networks with backpropagation learning. In *Proc. Field Programmable Logic and Applications*, pages 89–94, Aug. 2005.

[4] H. Esmaeilzadeh, P. Saeedi, B. N. Araabi, C. Lucas, and S. Fakhraie. Neural network stream processing core (NnSP) for embedded systems. In *Proc. Intl. Symp. on Circuits and Systems*, pages 2773–2776, Island of Kos, Greece, May 2006.

[5] A. Farmahini-Farahani, S. M. Fakhraie, and S. Safari. Sopc-based architecture for discrete particle swarm optimization. In *Proc. IEEE Intl. Conf. on Electronics, Circuits and Systems*, Marrakech, Morocco, Dec. 2007.

[6] A. Farmahini-Farahani, S. Vakili, S. M. Fakhraie, S. Safari, and C. Lucas. Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. submitted for publication.

[7] E. Fiesler and R. Beale. *Handbook of Neural Computation*. IOP Publishing Ltd and Oxford University Press, 1997.

[8] V. G. Gudise and G. K. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proc. IEEE Swarm Intelligence Symp.*, pages 110–117, Indianapolis, Indiana, USA, Apr. 2003.

[9] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, USA, second edition, 1999.

[10] S. Himavathi, D. D. Anitha, and A. Muthuramalingam. Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization. *IEEE Trans. on Neural Networks*, 18(3):880–888, May 2007.

[11] J. L. Holt and J.-N. Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. on Computers*, 42(3):281–290, Mar. 1993.

[12] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. IEEE Intl. Conf. Neural Networks*, volume 1, pages 1942–1948, 1995.

[13] A. Li, Q. Wang, Z. Li, and Y. Wan. A reconfigurable approach to implement neural networks for engineering application. In *Proc. World Cong. on Intelligent Control and Automation*, volume 1, pages 2939–2943, Dalian, China, June 2006.

[14] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural network training. In *Proc. Intl. Joint Conf. on Neural Network*, volume 2, pages 1895–1899, Honolulu, Hawaii, USA, May 2002.

[15] N. Nedjah and L. de Macedo Mourelle. Stochastic reconfigurable hardware for neural networks. In *Proc. Euromicro Symp. on Digital Systems Design*, pages 438–442, Sept. 2003.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, Oct. 1986.

[17] P. J. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, New York, 1994.

[18] B. Widrow, D. E. Rumelhart, and M. A. Lehr. Neural networks: Applications in industry, business and science. *Communications of the ACM*, 37(3):93–105, Mar. 1994.

[19] H.-M. Zhao, J.-J. Xu, and C.-G. Zhou. Reliability prediction of fuze storage based on BP neural network. *J. of Test and Measurement Technology*, 19(1):95–97, 2005.