Artificial Neural Network Implementation on a single FPGA of a Pipelined On-Line Backpropagation

Rafael Gadea[1] , Joaquín Cerdá[2], Franciso Ballester[1], Antonio Mocholí[1]

[1] *Department of Electronic Engineering, Universidad Politecnica de Valencia*
46022 Valencia, Spain
{rgadea, fballest, amocholi}@eln.upv.es
[2] *Department of Electronic Engineering, Universidad Politecnica de Valencia*
46022 Valencia, Spain
joacerbo@teleco.upv.es

Abstract

The paper describes the implementation of a systolic array for a multilayer perceptron on a Virtex XCV400 FPGA with a hardware-friendly learning algorithm. A pipelined adaptation of the on-line backpropagation algorithm is shown. Parallelism is better exploited because both forward and backward phases can be performed simultaneously. We can implement very large interconnection layers by using large Xilinx devices with embedded memories alongside the projection used in the systolic architecture. These physical and architectural features – together with the combination of FPGA reconfiguration properties with a design flow based on generic VHDL – create an easy, flexible, and fast method of designing a complete ANN on a single FPGA. The result offers a high degree of parallelism and fast performance.

1. Introduction

In recent years it has been shown that neural networks can provide solutions to many problems in the areas of pattern recognition, signal processing, time series analysis, etc. Software simulations are useful for investigating the capabilities of neural network models and creating new algorithms; but hardware implementations remain essential for taking full advantage of the inherent parallelism of neural networks.

Traditionally, ANN's have been implemented directly on special-purpose digital and analogue hardware. More recently, ANN's have been implemented with re-configurable FPGA's. Although FPGA's do not achieve the power, clock rate, or gate density, of custom chips; they do provide a speed-up of several orders of magnitude compared to software simulation [1]. Until now a principal restriction in this approach has been the limited logic density of FPGA's.

Although some current commercial FPGA's maintain very complex array logic blocks, the processing element (PE) of an artificial neural network is not likely to be mapped onto a single logic block. Often, a single PE could be mapped into a entire FPGA device, and if a larger FPGA is chosen it would be possible to implement some PEs – perhaps a small layer of neurons – but never a complete neural network. In this way, we can understand the implementations in [2] and [3] in which simple multilayer perceptrons (MLPs) are mapped using arrays of almost 30 Xilinx XC3000 family devices. These ANN's perform the training phase off-chip because considerable space can be saved.

A second solution for overcoming the problem of limited FPGA density is the use of pulse-stream arithmetic. With this technique the signals are stochastically coded in pulse sequences and therefore can be summed and multiplied using simple logic gates. This type of arithmetic can be observed with fine-grained FPGA's, for example: the ATMEL AT6005 [4]; or with coarse-grained FPGAs, for example: the Xilinx XC4005 [5] and XC4003 [6]. These implementations use an off-chip training phase, however, a simple ANN's can be mapped in a single device. In the same way, [7] presents an FPGA prototyping

implementation of an on-chip backpropagation algorithm that uses parallel stochastic bit-streams.

A third solution is to implement separate parts of the same system by time-multiplexing a single FPGA chip through run-time reconfiguration. This technique has been used mainly in standard backpropagation algorithms; dividing the algorithm in three sequential stages: forward, backward and update stage. When the stage computations are completed, the FPGA is reconfigured for the following stage. We can observe this solution by using the Xilinx XC3090 [8], or the Altera Flex10K [9]. Evidently, the efficiency of this method depends on the reconfiguration time when compared to computational time.

Finally, another typical solution is to use time-division multiplexing and a single shared multiplier per neuron [12]. This solution enables mapping a MLP for the XOR problem (3-5-2) on a single Xilinx XC4020 with the training phase off-chip.

This paper presents an advance in two basic respects with regard to previously reported neural implementations on FPGA's. The first is the use of an aspect of backpropagation and stems from the fact that forward and backward passes of different training patterns can be processed in parallel [13][14]. In [10] we show that this parallelism, referred to as "forward-backward parallelism", performs well in convergence time and generalisation rate. The better hardware performance of the pipelined on-line backpropagation is shown in terms of speed of learning. In [11] we specify this improvement of speed in a hardware implementation on an Altera FLEX10KE and show the hardware costs of this pipelined on-line backpropagation compared to standard backpropagation. The calculus was made by adding the obtained results with isolated neurons and synapses. In this paper, our main purpose will be to implement on a Xilinx XCV400 a completed MLP in which we can perform both versions of the algorithm with the same structure. The second point we contribute is to produce a completed ANN with on-chip training, and good throughput for the recall phase – on a single FPGA. This is necessary, for example, in industrial machine vision [2][3], and for the training phase, with continually online training (COT) [15].

In section 2 pipelined on-line backpropagation is presented and proposed. The methodology of design using VHDL is described in Section 3. Section 4 reviews some of the physical design issues that arose when mapping the ANN onto the Xilinx XCV400, and appraises the performance of the network.

2. Pipeline and Backpropagation algorithm

2.1 Initial point

The starting point of this study is the backpropagation algorithm in its on-line version. We assume we have a multilayer perceptron with three layers: two hidden layers and the output layer (i.e. 2-5-2-2 of Fig. 1)

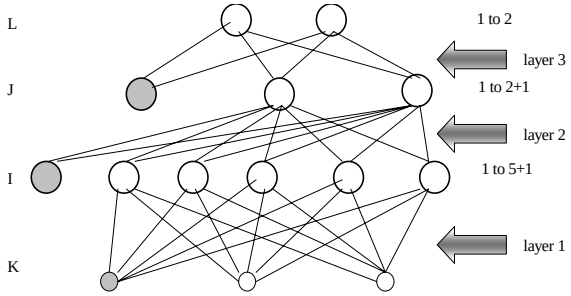

Figure 1. Multilayer perceptron

The phases involved in backpropagation taking one pattern *m* at a time and updating the weights after each pattern (on-line version) are as follows:

a) Forward phase. Apply the pattern a_i^K to the input layer and propagate the signal forwards through the network until the final outputs a_i^L have been calculated for each *i* and *l*

$$a^l{}_i = f(u_i{}^l)$$

$$y_i{}^l = u_i{}^l = \sum_{j=0}^{N_{l-1}} w^l{}_{ij} a^{l-1}{}_j \qquad (1)$$

$$1 \le i \le N_l \quad , 1 \le l \le L$$

b) Error calculation step. Computer the δ'*s* for the output layer *L* and compute the δ'*s* for the preceding layers by propagating the errors backwards using

$$\delta_i^L = f'(u^L{}_i)(t_i - y_i)$$

$$\delta_i^{l-1} = f'(u^{l-1}{}_i) \sum_{j=1}^{N_l} w_{ij} \delta^l{}_j \quad (2)$$

$$1 \le i \le N_l \quad , 1 \le l \le L$$

c) Weight update step. Update the weights using

$${}^m w_{ij}^l = {}^{m-1} w_{ij}^l + {}^m \Delta w_{ij}^l$$

$${}^m \Delta w_{ij}^l = \eta \, {}^m \delta_i{}^l y_j^{l-1} \qquad (3)$$

$$1 \le i \le N_l \quad , 1 \le l \le L$$

All the elements in (3) are given at the same time as the necessary elements for the error calculation step; therefore it is possible to perform these two last steps simultaneously

(during the same clock cycle) in this on-line version and to reduce the number of steps to two: forward step (1) and backward step (2) and (3). However in the batch-line version, the weight update is performed at the end of an epoch (set of training patterns) and this approximation would be impossible.

2.2 Pipeline versus non-pipeline

Non- pipeline: Non-pipeline: The algorithm takes one training pattern m . Only when the forward step is finished in the output layer can the backward step for this pattern occur. When this step reaches the input layer, the forward step for the following training pattern can start (Figure 2).

In each step *s* only the neurons of each layer can perform simultaneously, and so this is the only degree of parallelism for one pattern. However, this disadvantage means we can share the hardware resources for both phases because these resources are practically the same (matrix-vector multiplication).

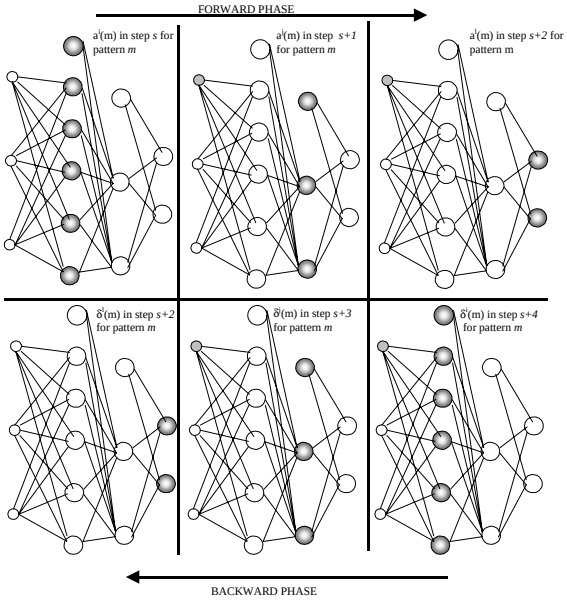

Figure 2. Non-pipeline version

Pipeline: The algorithm takes one training pattern m and starts the forward phase in layer *i*. The following figure shows what happens at this moment (in this step) in all the layers of the multilayer perceptron.

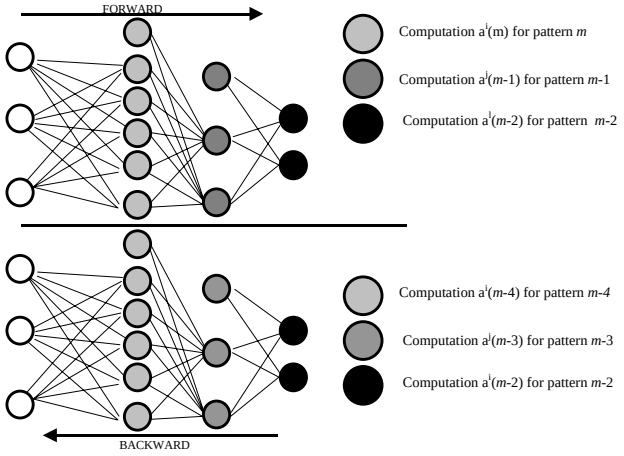

Figure 3. Pipeline version

Figure 3 shows that in each step, every neuron in each layer is busy working simultaneously, using two degrees of parallelism: synapse-oriented parallelism and forward-backward parallelism. Of course, in this type of implementation, the hardware resources of the forward and backward phases cannot be shared in one cycle. In the section 3 we will see how, in spite of this problem, the high-level tradeoff study of the implementation of the synapses in order to find the best solution in pipeline version for the proposed systolic array.

Evidently, the pipeline carries an important modification of the original backpropagation algorithm [16][17]. This is clear because the alteration of weights at a given step interferes with computations of the states a_i and errors δ_i for patterns taken at different steps in the network. For example, we are going to observe what happens with a pattern *m* on its way to the network during the forward phase (from input until output). In particular, we will take into account the last pattern that has modified the weights of each layer . We can see:

1. For the layer *I* the last pattern to modify the weights of this layer is the pattern *m*-5.
2. When our pattern *m* passes the layer *J*, the last pattern to modify the weights of this layer will be the pattern *m*-3.
3. Finally, when the pattern reaches the layer *L* the last pattern to modify the weights of this layer will be the pattern *m*-1.

Of course, the other patterns also contribute. The patterns which have modified the weights before patterns *m-5*, *m-3* and *m-1*, are patterns *m-6*, *m-4* and *m-2* for the layers *I, J* and *L* respectively. In the pipeline version, the pattern *m-1* is always the last pattern to modify the weights of the all layers. It is curious to note that when we use the momentum variation of the backpropagation algorithm with the pipeline version, the last six patterns before the

current pattern contribute to the weight updates, while with the non-pipeline version, only the last two patterns contribute before the current pattern.

Therefore, we have a variation of the original on-line backpropagation algorithm that consists basically in a modification of the contribution of the different patterns of a training set in the weight updates, and in the same line as the momentum variation.

3. Design and verification

3.1 Design flow and tools

Figure 4 shows all the information relative to design flow and tools. It is important to know that our design implements two modes of operation: with forward-backward parallelism (pipeline mode); or without this degree of parallelism.

We wish to emphasise that this design flow can be totally carried out on a PC without limitations and with a good performance. In this design flow, we have resorted to three different software vendors to obtain the maximum power and technology independence. The difficulty for this type of option is always the interfaces between the tools, but in this case, these interfaces (principally in VHDL) are problem-free.

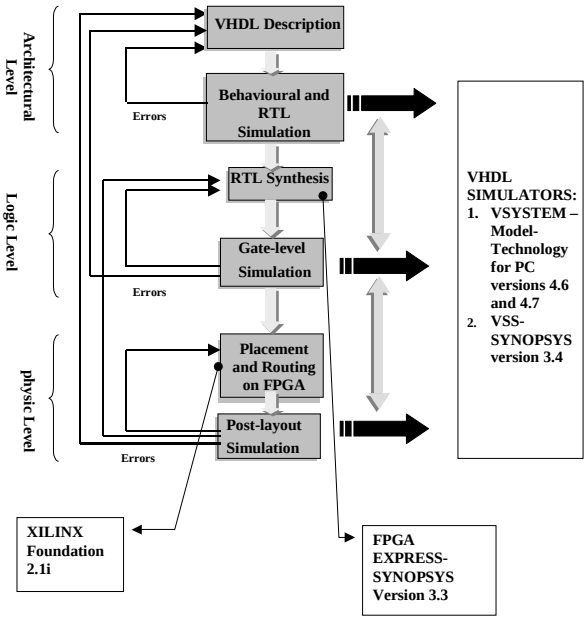

Figure 4. Design Flow

3.2 Digital architecture of the ANN

We suppose that we have a MLP (multilayer perceptron) with three layers (Figure.1) and the following characteristics:

NE = number of inputs.
N1O = number of neurons in the first hidden layer.
N2O = number of neurons in the second hidden layer.
NS = number of outputs.

Figure 5 shows the "alternating orthogonal systolic array" of an MLP with two hidden layers [18]. This architecture of this figure can implement the following structure (2-N1O-3-NS) and is useful for the XOR problem.

We can observe that NE (2) determines the number of vertical synapses (SV) in the first layer. N2O (3) determines the dimensions of the horizontal layer and the last vertical layer; that is to say, the number of horizontal synapses (SH) and horizontal neurons (NH) and the number of last vertical synapses (Svu). The size of N1O will determine the size of the weight memories of the vertical and horizontal synapses, and the size of NS will determine the size of the weight memories of the synapses in the last vertical layer.

In our implementation, these weight memories are mapped in the block SelecRAM +. For example, a Xilinx XCV400 device has 20 blocks that can implement 256 weights with 16 bits of resolution. The architecture shown in figure 5 only needs 10 of these embedded RAMs; one for each synapse. This supposes that the size of N1O and NS has a possible range from 1 to 256, and therefore we can implement a (2-256-3-256) network with the same hardware resources.

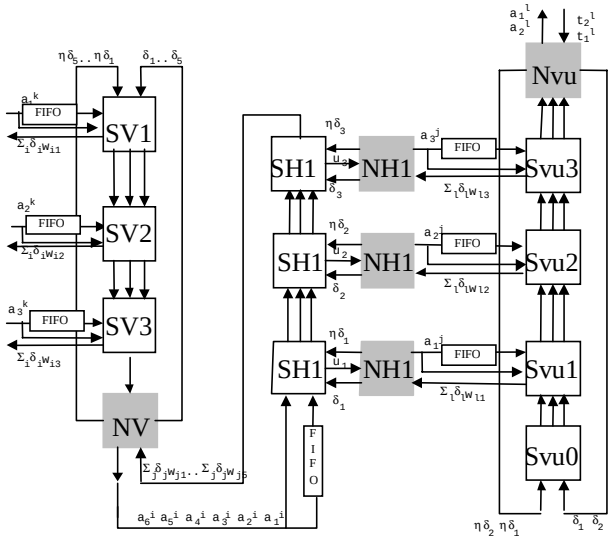

Figure 5. Architecture of MLP

3.3 High level synthesis in synapses

The main problem in the incorporation of forward-backward parallelism is the design of the synapses (white blocks in figure 5). The size of the synapses is increased by 40% when we want to manage the forward and backward phases simultaneously. We can see the necessary hardware resources for a synapse in figure 6 when working in one cycle.

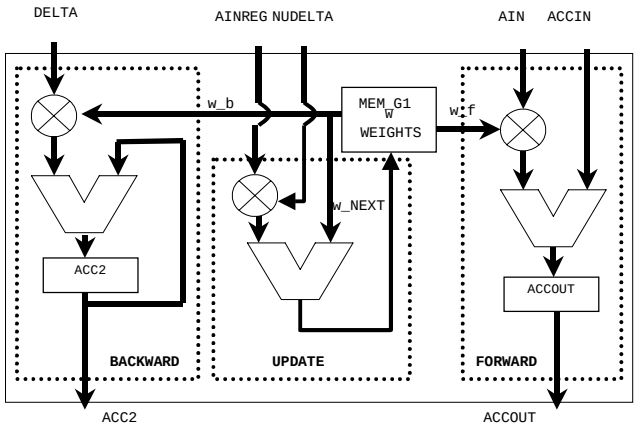

Figure 6. Hardware resources in synapse.

We have realised a high-level synthesis with a behavioural compiler of synopsys to find different solutions with different latencies and frequencies between 10MHz and 100MHz. The results are shown in figure 7 and the 20MHz solution (the best in most cases) is highlighted in red.

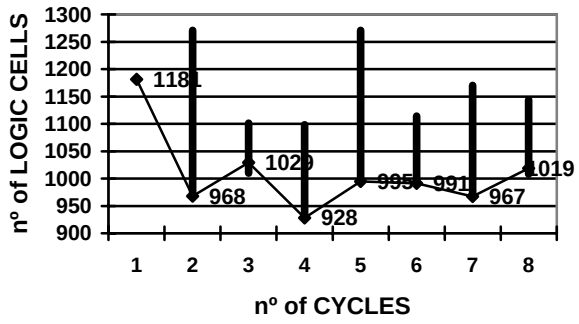

Figure 7. High-level synapse trade-off.

We observe that we can reduce the hardware resources of a synapse by 20% (30% at the most in the complete ANN) if we work with a latency of 4 cycles. Evidently, we would work with the whole architecture (the complete ANN) with throughputs divided by 4, but the designer can quickly decide with this type of tool the best solution for the specifications of the application of the neural network.

3.4 Simulation results

Although the architecture has been verified for different databases [10] we only show the simulation results for the XOR problem (our example in this paper). These results are obtained with the (2-6-3-2) network and with a resolution of 8 bits for activations and deltas; and 16 bits for weights.

In figure 6. we show the *convergence time* T_c, expressing the number of epochs from the beginning of learning up to the moment when the RMS error training set goes below a given threshold E_{out}=0.05. The used mode is the pipeline version.

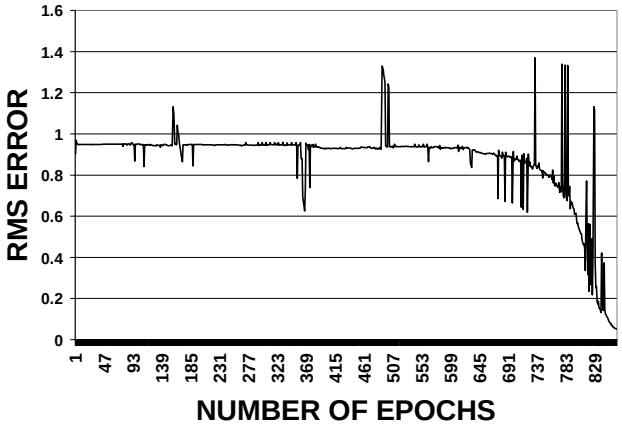

Figure 8. Simulation with XOR problem. Pipeline Mode.

4 Implementation on the Virtex XCV400

4.1 Speed and resource usage

The ANN is mapped onto the XCV400BG560-6 package. Part of map report of Xilinx Foundation 2.1i is reproduced in figure 9. This implementation can work at up to 10 MHz; but we performed the synthesis, placement and routing, without timing constraints.

Number of Slices:	3,473 out of	4,800	72%
Slice Flip Flops:	786		
4 input LUTs:	5,925 (41 used as a route-thru)		
32x1 RAMs:	88		
Number of bonded IOBs:	44 out of	404	10%
Number of Block RAMs:	10 out of	20	50%
Total equivalent gate count for design: 248,822			

Figure 9. Design summary produced by Xilinx software.

4.2 Results and performance

The results of the above implementation are summarised in table 1. The ANN has been analysed in pipeline mode and in non-pipeline mode. The pipeline mode only affects the training and performance parameters that measure the number of connections updated per second (CUPS).

10 MHz	pipeline	non pipeline
Throughput training phase	0.7 us	3.3 us
Throughput recall phase	0.7us	0.7 us
TOTAL PERFORMANCE	81 MCPS 81 MCUPS	81 MCPS 17 MCUPS

Table 1

The values of throughput for this implementation are satisfactory for most real-time operating systems.

5. Conclusions

This paper evaluates the performance of the pipelined on-line backpropagation algorithm implemented in FPGA's. This algorithm removes some of the drawbacks that traditional backpropagations suffer when implemented on VLSI circuits. It may go on to offer considerable improvements, especially with respect to hardware efficiency and speed of learning, although the circuitry is more complex.

We believe this paper also contributes new data for the classical contention between researchers who work with specific hardware implementation for artificial neural networks and those working with software approaches and general purpose processors. Until now, software solution was preferred in order to get quick, flexible designs for different topologies, algorithms, connectivity, activation and base functions, etc. Now, we can see that to exploit all the degrees of parallelism and fault tolerance, we can use hardware designs with several fine-grained processors without degradation of flexibility, quick design, and reusability – thanks to the combination of the reconfiguration properties of FPGA's and a design flow based in VHDL on a PC.

[1] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems" *Proceedings of the IEEE,* 86(4), April 1998, pp. 615-638.

[2] C.E. Cox, W.E. Blanz, "GANGLION- A Fast Field-Programmable Gate Array Implementation of a Connectionist Classifier" *Journal of Solid State Circuits,* Vol.27, no. 3, March 1992, pp. 288-299.

[3] V. Jean, B. Patrice, R. Didier, S. Mark, T. Hervé, B. Philippe. "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Transactions on VLSI Systems,* Vol 4, No 1, March 1996, pp. 56-69.

[4] P. Lysaght, J. Stockwood, J. Law and D. Girma, "Artificial Neural Network Implementation on a Fine Grained FPGA".

[5] V. Salapura, M. Gschwind, and O. Maischberger, " A Fast FPGA Implementation of a General Purpose Neuron", *Porc. of the Fourth International Workshop on Field Programmable Logic and Aplications,* September 1994.

[6] S.L Bade and B.L. Hutchings, "FPGA-Based Stochastic Neural Networks Implementation", *IEEE Workshop on FPGAs for Custom Computing Machines,* April 1994, pp. 189-198.

[7] K. Kollmann, K. Riemschneider, and H.C. Zeider, "On-Chip Backpropagation Training Using Parallel Stochastic Bit Streams" *Proceedings of the IEEE International Conference on Microelectronics for Neural Networks and Fuzzy Systems MicroNeuro'96,* 1996, pp. 149-156.

[8] J.G. Elredge and B.L. Hutchings, " RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs", *Proc. IEEE Int. Conf. on Neural Networks"*, June 1994.

[9] J-L. Beuchat, J-O. Haenni and E. Sanchez, "Hardware Reconfigurable Neural Networks".

[10] R. Gadea, A. Mocholí, "Systolic Implementation of a Pipelined On-Line Backpropagation", *Proc.of the NeuroMIcro'99,* April 1999, pp. 387-394.

[11] R. Gadea, A. Mocholí, "Forward-backward Parallelism in On-Line Backpropagation", *International Work- Conference on Artificial and Natural Neural Networks,* June 1999, pp. 157-165.

[12] N. Izeboudjen, A. Farah, S. Titri, H. Boumeridja, "Digital Implementation of Artificial Neural Networks: From VHDL Description to FPGA Implementation", *Proceedings International Work-Conference on Artificial and Natural Neural Networks, IWANN'99"*, vol.2, June 1999, pp. 139-148.

[13] C.R. Rosemberg, and G. Belloch, "An Implementation of Network Learning on the Connection Machine", *Connectionist Models an their Implications,* D. Waltz and J Feldman, eds., Ablex, Norwood, NJ. 1988

[14] A. Petrowski, G. Dreyfus, and C. Girault, "Performance Analysis of a Pipelined Backpropagation Parallel Algorithm", *IEEE Transaction on Neural Networks,* Vol.4 , no. 6, November 1993, pp. 970-981.

[15] B. Burton, R.G. Harley, G. Diana, and J.R. Rodgerson, "Reducing the Computational Demands of Continually Online-Trained Artificial Neural Networks for System Identification and Control of Fast Processes" *IEEE transaction on Industry Applications,*Vol 34. no.3, May/June 1998, pp. 589-596.

[16] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error backpropagation, *Parallel Distributed processing,* Vol. 1, MIT Press. Cambridge, MA, 1986, pp. 318-362.

[17] S.E. Falhman, "Faster learning variations on backpropagation: An empirical study", *Proc. 1988 Connectionist Models Summer School,* 1988 , pp. 38-50.

[18] P. Murtagh, A.C. Tsoi, and N. Bergmann, " Bit-serial array implementation af a multilayer perceptron ", *IEEE Proceedings-E,* Vol. 140, no. 5, September 1993, pp. 277-288.