Implementation with FPGAs of a Pipelined On-Line Backpropagation

Rafael Gadea Gironés

Dpto. Ing. Electrónica U.P.V
46020 Valencia
rgadea@eln.upv.es

Antonio Mocholí Salcedo

Dpto. Ing. Electrónica U.P.V
46020 Valencia
amocholi@eln.upv.es

ABSTRACT

The paper describes the implementation of a systolic array for a multilayer perceptron on ALTERA FLEX10KE FPGAs with a hardware-friendly learning algorithm. A pipelined adaptation of the on-line backpropagation algorithm is shown. It better exploits the parallelism because both the forward and backward phases can be performed simultaneously. As a result, a combined systolic array structure is proposed for both phases. Analytic expressions show that the pipelined version is more efficient than the non-pipelined version. The design is implemented and simulated using VHDL at different levels of abstraction and finally mapped on FPGAs.

1. INTRODUCTION

In recent years it has been shown that neural networks are capable of providing solutions to many problems in the areas of pattern recognition, signal processing, time series analysis, etc. While software simulations are very useful for investigating the capabilities of neural network models and creating new algorithms, hardware implementations are essential to take full advantage of the inherent parallelism of neural networks.

To organize the ideas described below a careful examination of the parallelism inherent in artificial neural networks (ANN) is useful. A casual inspection of the standard equations used to describe backpropagation reveals two obvious degrees of parallelism in an ANN. Firstly, there is parallel processing by many nodes in each layer. Secondly, there is parallel processing in the many training examples.

The former comes to mind most easily when parallel aspects of ANNs are being considered. The network may be partitioned by distributing the synaptic coefficients and neurons throughout a processor network. Again, there are two variations of this technique: “neuron-oriented parallelism” – and “synapse-oriented parallelism”.

In the first variation, the neurons are distributed among available processors. However, it is difficult to place the neurons in such a way as to produce efficient implementations, which require both an evenly distributed computational load (easy) and reduced data communications (difficult) [1].

The second variation is based on the fact that the computations in a neural network are basically matrix products [2],[3], [4] and [5]. The advantage of this approach is the amount of data communicated between processors is moderate and evenly distributed, although in a multilayer perceptron the synaptic matrix is lower triangular. It can be more interesting to perform the matrix in an implementation distributed by layers (matrix-vector representation) [6].

The latter, which we will refer to as “training set parallelism”, is perhaps the most useful. In backpropagation, this latter parallel aspect is the result of the linear combination of the individual contributions made by each training pattern to the adjustment of the network weights. The linearity implies that the patterns can be processed independently and hence, simultaneously. However, this implementation requires that the weights be updated after all the parallel processed training patterns have been seen:– so-called “batch” updating [7].

A third, but less obvious, aspect of backpropagation stems from the fact that forward and backward passes of different training patterns can be processed in parallel. In a work by Rosemberg and Belloch [8] with Connection Machine, the authors noted this possibility in their implementation, though it remained unimplemented. Later, A. Petrowski et.[9], describe a theoretical analysis and experimental results with transputers. However, only the batch-line version of backpropagation algorithm was shown. The possibility of an on-line version was noted by the authors in general terms, but it was not implemented with systematic experiments and theoretical investigations. In [10] we show that this parallelism, we will refer to it as "forward-backward parallelism", has a good performance in convergence time and generalization rate and we begin to show the better hardware performance of the pipelined on-line backpropagation in terms of speed of learning. Now in this paper our main purpose will be to concrete this improvement of speed in a hardware implementation on Altera FLEX10K50 and to show the hardware costs of this pipelined on-line backpropagation always compared to standard backpropagation.

In section 2 pipelined on-line backpropagation is presented and proposed. The methodology of design using VHDL is described in Section 3. Also in this section the implementation properties when we compile on FLEX10K FPGAs from Altera will be presented .

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)

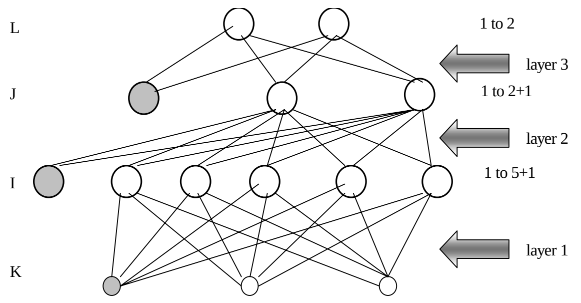

Figure 1

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).

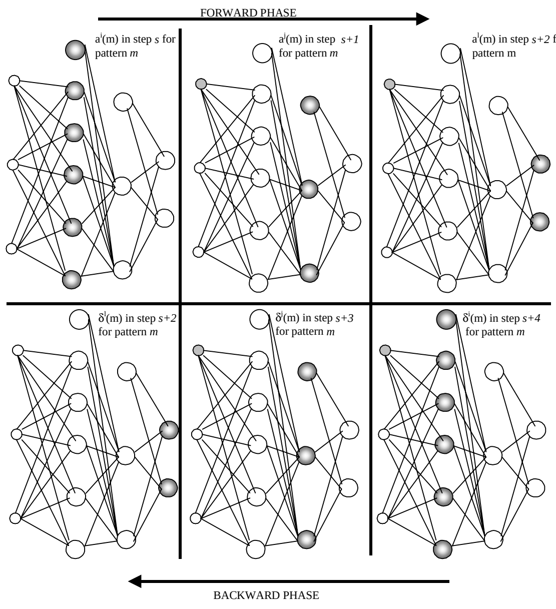
FORWARD PHASE
a^i(m) in step s for pattern m
a^i(m) in step s+1 for pattern m
a^i(m) in step s+2 for pattern m
δ^i(m) in step s+2 for pattern m
δ^i(m) in step s+3 for pattern m
δ^i(m) in step s+4 for pattern m
BACKWARD PHASE

Figure 2

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.

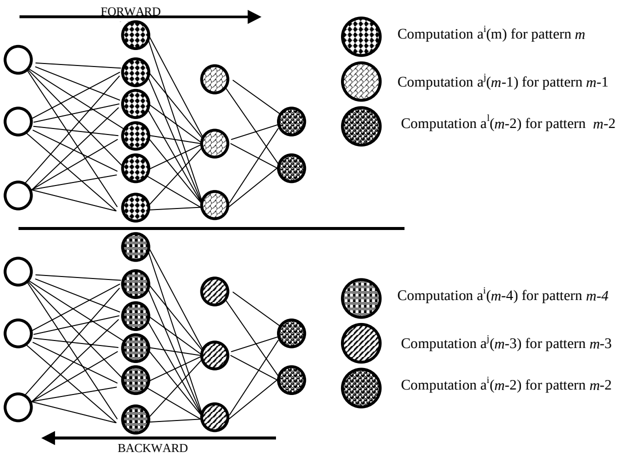
FORWARD
Computation a^i(m) for pattern m
Computation a^j(m-1) for pattern m-1
Computation a^l(m-2) for pattern m-2
Computation a^i(m-4) for pattern m-4
Computation a^j(m-3) for pattern m-3
Computation a^l(m-2) for pattern m-2
BACKWARD

Figure 3

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 the following section we will see how, in spite of this problem, the pipeline version for the proposed systolic array is more efficient than the non-pipeline version.

Evidently, the pipeline carries an important modification of the original backpropagation algorithm [11],[12]. 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.

It is important that the equations for the two phases perform in the same manner as the non-pipeline version. For this, it will be necessary to store the values of sigmoids and their derivatives (see the following section). 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. IMPLEMENTATION AND VERIFICATION

This section compares directly the implementation properties of pipelined on-line BP with the standard BP algorithm when we use the same technology: ALTERA FLEX10KE FPGAs .

3.1 Design entry with VHDL

The design entry of the pipelined on-line BP, and classical on-line BP for performing comparatives, is accomplished in VHDL. It is very important to make these system descriptions independents of the physical hardware because our objetive in the future will be to test our descriptions on others FPGAs and even on ASICs.

We have made eight VHDL testbenches to perform the simulations: four for the pipeline version and four for the non-pipeline version. The VHDL description of the "alternating orthogonal systolic array" (always the unit under test) is totally configurable by means of generics and generates statements whose values are obtained from three ASCII files:

- Database file: number of inputs, number of outputs, the training and validation patterns
- Learning file: number of neurons in the first hidden layer, number of neurons in the second hidden layer, type of learning (on-line, batch-line or BLMS), value of learning rate η, value of momemtum rate, type of sigmoid
- Resolution file: resolution of weights (integer and decimal part), resolution of activations, resolution of accumulator (integer and decimal part),etc.

3.2 Speed and resource usage

The speed performance of pipeline version is much better than non pipeline version in the number of Connections Updated Per Second (Table 1).

	pipeline	non pipeline
TOTAL PERFORMANCE	198,9 MCPS 198,9 MCUPS (12 MHz)	281,7 MCPS 77,84 MCUPS (17 MHz)

Table 1

We show that the increment of hardware cost for pipelining the backpropagation algorithm is higher in the synapses (until 300 logic cells) than in the neurons (until 70 logic cells) and is produced fundamentally because the pipeline version needs different multipliers and accumulators for the forward and backward phases.

4. CONCLUSIONS

This paper evaluates the hardware performance of the pipelined on-line backpropagation algorithm. 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 contributes new data for the classical contention between researchers who update network weights continuously (on-line) and those updating weights only after some subset, or often after the entire set, of training patterns has been presented to the network (batch-line). Until now, batch updating after the entire training set has been processed (i.e. after each epoch) was preferred in order to best exploit "training set parallelism" and "forward backward parallelism." Now, we can see that to exploit all the degrees of parallelism, we can use the on-line version of backpropagation without degradation of its properties.

[1] A. Singer, "Implementations of artificial neural networks on the connection machine", *Parallel Computing*, vol. 14, 1990, pp. 305-315.

[2] S. Shams, and J.L. Gaudiot, "Implementing Regularly Structured Neural Networks on the Dream Machine" *IEEE transactions on neural networks,* vol. 6, no.2 , March 1995, pp. 408-421.

[3] W-M lin, V. K. Prasanna, and K. W. Przytula, "Algorithmic Mapping of Neural Network Models onto Parallel SIMD Machines", *IEEE Transactions on Computers,* Vol. 40, no. 12, December 1991, pp. 1390-1401.

[4] S.R. Jones, K.M. Sammut, and J. Hunter, "Learning in Linear Systolic Neural Network Engines: Analysis and Implementation", *Transactions on Neural Netwoks,* Vol. 5, no. 4, July 1994, pp. 584-593.

[5] D. Naylor, S. Jones , and D. Myers, " Backpropagation in Linear Arrays- A performance Analysis and Optimization", *IEEE Transactions on Neural Networks,* Vol. 6, no. 3, May 1995, pp. 583-595.

[6] 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.

[7] X. Zhang, M. McKenna, J.P. Mesirov, and D Waltz, "An efficient implementation of the backpropagation algorithm on the conection Machine CM-2, *Advances in Neural Information Porcessing Systems 2,* D.S. Touretzky,Ed. San Mateo, CA:Morgan Kaufmann, 1990, pp. 801-809.

[8] 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.

[9] 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.

[10] R.Gadea, A. Mocholí, "Systolic Implementation of a Pipelined On-Line Backpropagation", *Proc. Conf. om Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems* ,Granada, Spain, April 1998, pp. 387-394.

[11] 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.

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