Fast Neural Networks Without Multipliers

Michele Marchesi, Gianni Orlandi, *Member, IEEE,* Francesco Piazza, *Member, IEEE,* and Aurelio Uncini, *Member, IEEE*

Abstract—**The paper introduces multilayer perceptrons with weight values restricted to powers-of-two or sum of power-of-two. In a digital implementation, these neural networks do not need multipliers but only shift registers when computing in forward mode, thus saving chip area and computation time. A learning procedure, based on back-propagation, is presented for such neural networks. This learning procedure requires full real arithmetic and therefore must be performed off-line.**

Some test cases are presented, concerning MLP's with hidden layers of different size, on pattern recognition problems. Such tests demonstrate the validity and the generalization capability of the method and give some insight into the behavior of the learning algorithm.

I. Introduction

In the past years, artificial neural networks (NN) have been applied to a variety of problems in many fields, most of them calling for an efficient hardware implementation. Using electronic or optical techniques, several different approaches have been proposed to face this problem. Among them, the digital approach is the most popular. It consists in the numerical simulation of NN's using digital VLSI hardware, parallel computers, or traditional computers. Using this approach, it is possible to take advantage of state-of-the-art VLSI and ULSI implementation techniques, with several advantages over the analog implementation [1].

One of the major problems of digital architectures implementing NN's, affecting both performance and chip area, is the presence of multipliers. In almost every neural model, in fact, the elementary processor (neuron) computes an activation function on the weighted sum of its inputs. While weights are stored and recalled easily in a local memory, the activation function can also be easily computed through and look-up table and sums are performed quickly using a limited amount of hardware; the multiplications between inputs and weights can be the bottle-neck of the system. They are slow compared to other operations and the multipliers require of lot of chip area if a direct VLSI or ULSI implementation is planned. A possible solution to this problem has already been exploited in digital filter design: it consists in avoiding multiplications using integer weights whose values are power-of-two or sums of power-of-two [1], [2]. Using such weights, it is possible to substitute multiplications with simple shifts or shift-and-add operations, which are much faster and require less hardware.

In this paper we demonstrate the feasibility of this approach for the multilayer perceptron (MLP) [3] with binary outputs, and present a learning procedure based on backpropagation (BP) [4] to obtain a MLP with such weights. This learning procedure requires full real arithmetic and therefore must be performed off-line. Once the weights are computed a specific problem, the corresponding power-to-two values can be loaded into the multilayer neural network that will be run in forward mode, processing the data. In the following, after briefly presenting a notation for MLP description, we shall discuss the problem of assigning the neurons to the available processing elements, describe in detail the learning algorithm, and present several test cases concerning MLP's of different sizes and with different numbers of processing elements. These tests confirm the validity of previous results [5], indicate the validity of the method, and give some insight into the behavior and generalization capabilities of the learning algorithm.

II. Multilayer Perceptron (MLP)

Many NN models have been proposed in technical literature; among them the MLP is one of the most widely used because a simple and powerful learning algorithm, known as back-propagation has been devised for it. Here we shall briefly describe this model and we shall introduce a notation for its description that will be followed throughout the paper.

In the MLP, all neurons are grouped in sequentially connected layers, numbered from 0 to M. Let N_s be the number of neurons belonging to the generic sth layer. Each neuron is connected to all neurons of the two adjacent layers. The neurons of layer 0 (input layer) do not perform computations but only feed input signals into neurons of layer 1. Layer M is the output layer, from where the network response comes.

Each MLP neuron is characterized by one output and many inputs, which are the neuron outputs of the preceding layer. Let $o_k{}^{(s)}$ denote the output of kth neuron of the sth layer, then the computation performed by each neuron can be expressed as:

$$\mathrm{net}_k{}^{(s)} = \sum_{j=1}^{N_{s-1}} w_{kj}{}^{(s)} o_j{}^{(s-1)} + \vartheta_k{}^{(s)} \tag{1}$$

$$o_k{}^{(s)} = f\left(\mathrm{net}_k{}^{(s)}\right) \tag{2}$$

where $\mathrm{net}_k{}^{(s)}$ is the weighted sum of the k neuron of the sth layer, $w_{kj}{}^{(s)}$ is the weight by which the same neuron multiplies the output $o_j{}^{(s-1)}$ of the jth neuron of the previous

Manuscript received March 5, 1991; revised April 21, 1992. This work was supported in part by MURST funding and by CNR PF "Materiali e disposivi per Elettronica a Stato Solido."

M. Marchesi is with the Dipartimento di Ingegneria Biofisica e Elettronica, Università di Genova, 16145 Genova, Italy.

G. Orlandi is with the Dipartimento INFOCOM, Università di Roma "La Sapienza," 00183 Roma, Italy.

F. Piazza and A. Uncini are with the Dipartimento di Elettronica e Automatica, Università di Ancona, 60131 Ancona, Italy.

IEEE Log Number 9201748

layer, $\vartheta_k{}^{(s)}$ is the offset of the kth neuron of the sth layer and $f(.)$ is a nonlinear, bounded function, often the sigmoid function.

When processing data, the MLP accepts an input vector $\boldsymbol{i}$ and produces an output vector $\boldsymbol{o}$. From the above considerations, $\boldsymbol{i}$ is equal to $\boldsymbol{o}^{(0)}$, whose components are $o_k{}^{(0)}$, $k = 1, \cdots, N_0$. Applying iteratively equations (1) and (2), starting from the first ($s = 1$) to the last ($s = M$) layer, an output vector $\boldsymbol{o}$, equal to $\boldsymbol{o}^{(M)}$, is obtained, whose components are $o_k{}^{(M)}, k = 1, \cdots, N_M$.

The backpropagation [4] is the most widely used training algorithm in MLP networks. Using an iterative steepest descent technique, it is able to compute the values of the weights and offsets which approximate a desired network behavior. BP requires a number of forward moves (computation of $\boldsymbol{o}^{(M)}$ for given input vector), each followed by a backward move, during which weight and offset adjustments occur to adapt the network to a desired output vector $\boldsymbol{d}$. This operation can be expressed in its simplest form by the following formulae:

$$\sigma_k{}^{(s)} = \begin{cases} d_k - o_k{}^{(s)} & s = M; \\ \sum_{j=1}^{N_{s+1}} w_{jk}{}^{(s+1)} \delta_j{}^{(s+1)} & s = 1, \cdots, M-1 \end{cases} \tag{3}$$

$$\delta_k{}^{(s)} = \sigma_k{}^{(s)} f'\left(\text{net}_k{}^{(s)}\right) \qquad s = 1, \cdots, M \tag{4}$$

$$\Delta w_{kj}{}^{(s)} = \eta \delta_k{}^{(s)} o_j{}^{(s-1)} \qquad k = 1, \cdots, N_s, \; j = 1, \cdots, N_{s-1} \tag{5}$$

where $f'_{(\cdot)}$ is the derivative of the activation function $f(\cdot)$, η is a constant called the learning rate and $\Delta w_{kj}{}^{(s)}$ is the weight change. Similar formulate are used to adjust offsets. By applying iteratively equations (3)–(5) from the last ($s = M$) to the first ($s = 1$) layer, the backward move is completed.

III. Neurons and Neural Networks Without Multipliers

A. Weight Values

The starting point of our approach to the realization of fast NN without multipliers are (1) and (2). The basic activity of a single neuron consists in calculating the weighted sum $\text{net}_k{}^{(s)}$ of its inputs and then computing the activation function $f(\cdot)$. The multiplications concern only the weights and the corresponding inputs, not the offset. To avoid numeric multiplications, the weight values must be restricted to power-of-two (or sums of power-of-two).

In this case, the multiplications can be substituted by shift, or shift-and-add operations whose hardware implementation is very simple and much faster than multiplication. The number of bit-shifts that can be performed equals the difference between the maximum and the minimum power-of-two term. In the case of the sum of two power-of-two terms, two shift registers and one adder are needed.

Fig. 1 shows the degree of discretization and nonuniformity of the value distribution for both single power-of-two and sum of two power-of-two terms, with a maximum bit-shift of six.

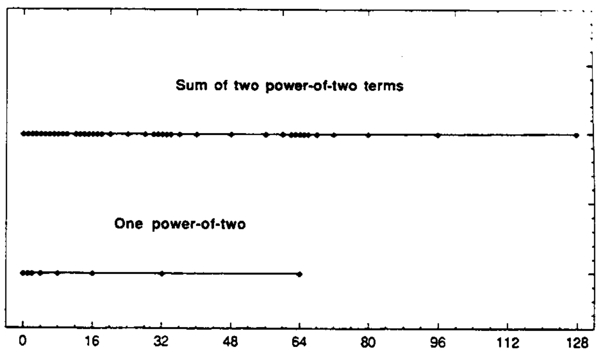


Fig. 1. The discretization obtained using power-of-two terms. The permitted values actually used in the presented method are divided by 64.

Throughout the paper, however, we will use a normalization procedure yielding weights, whose absolute values are less than or equal to a given value (usually one).

As they are not involved in multiplications, the offset values can still be real numbers, thus providing more "degrees of freedom" in learning. The dual case in which inputs to the neurons and not weights are restricted to power-of-two will not be considered in the paper, although it could be a matter for further research.

In the presented results we shall consider the case of a single power-to-two, which represents the most favorable case for hardware implementation, as additional sums are eliminated. It also represents the most difficult case as far as learning is concerned because the discretization is extreme. The learning algorithm proposed in the next section, however, is applicable also to more general NN's, whose weights are sums of two or more power-of-two terms. Let us denote with W_s the set of admissible weight values:

$$W_s \equiv \{x | x = R \quad 2^{-p}; R \in \{-1, 0, 1\}; p \in \{0, 1, \cdots, S\}\} \tag{6}$$

where S is the number of bits that can be shifted. It can be noted that this definition constrains the weights absolute value to be less than or equal to one, which is easily obtained by scaling weights and offsets.

Extending admissible weight values to the sum of two power-of-two terms leads to the set:

$$W_{S,T} \equiv \{x | x = R' 2^{-p} + R'' 2^{-q}; R', R'' \in \{-1, 0, 1\}; \; p \in \{0, 1, \cdots, S\}; q \in \{0, 1, \cdots, T\}\} \tag{7}$$

where S and T are the maximum number of shifts in each shift register, respectively, and the maximum weights absolute value is less than or equal to two.

It is also worth noting that increasing the number of power-of-two terms of the sum enables the approximation of any number in a given interval to a given error. The expansion in the sum of more than two power-of-two terms has been recently proposed in digital filter design [6], limited to a small subset of weights to fulfil the filter requirements. Such expansion, however, would substantially weaken the advantages of this technique, owing to the increased complexity of weight management and the higher number of operations.

B. Neurons and Processing Elements

In a digital implementation of NN's, one of the basic problems is the assignment of the neurons to the actual processing elements (PE) that constitute the available digital hardware. Depending on the number of PE's, the possible choices span from the assignment of a single neuron to each PE to a single PE performing the simulation of the whole NN. At the former extreme there are VLSI implementations of high granularity, while at the latter there are software simulations on a single CPU computer. In the middle there are many possible solutions, implemented directly on VLSI chips, or on many connected microprocessors running in parallel, or on parallel or multi-CPU computers, characterized by assigning the simulation of more than one neuron to every PE with various possible levels of granularity.

In a fast digital implementation the activation function $f(.)$ is often computed through a look-up table (LUT) contained in every PE. Depending on the granularity of the NN, i.e. on the number of neurons that are assigned to each PE, and on the way neurons are assigned to PE's, we may consider different cases:

- a distinct LUT for every neuron;
- a LUT common to the neurons of each layer;
- a LUT common to the neurons of each vertical "slice" of the MLP;
- a LUT common to the whole NN.

Fig. 2(a)-(d) graphically show these four possible assignments of MLP neurons to PE's and thus the possible LUT assignments to neurons. Cases (a) and (d) are straightforward, while cases (b) and (c) are derived by architectures implementing MLP and found in the literature [7], [8]. Of course, other assignments of neurons to PE's can be found, and the presented learning algorithm can be easily adapted to deal with any possible assignment. Since each LUT allows a distinct scaling of all the weights of the neurons using that LUT, the NN has more degrees of freedom with a greater number of LUT's. This means a better convergence in learning, as test cases presented in the next sections will show.

IV. Learning Algorithm

A. Basic Ideas

Given a mapping between input and output spaces of the MLP and a training set of pairs $(\boldsymbol{i}_p, \boldsymbol{d}_p)$ of inputs and outputs representing or sampling the mapping, the problem is to find the right MLP architecture (number of layers and of neurons per layer) and its weights (limited to some W_S or $W_{S,T}$ ensemble) and offsets values able to perform the desired mapping with minimum error.

The starting point for the learning algorithm is the solution of the same mapping with a classical MLP having continuous weights, applying BP as the learning algorithm. We shall not address here the problem of finding the right number of layers and neurons per layer, but we shall assume that a continuous solution can be found.

The main ideas behind the proposed learning algorithm are two: the first is to scale the weights of the neurons referring to the same LUT in such a way as to maximize the match

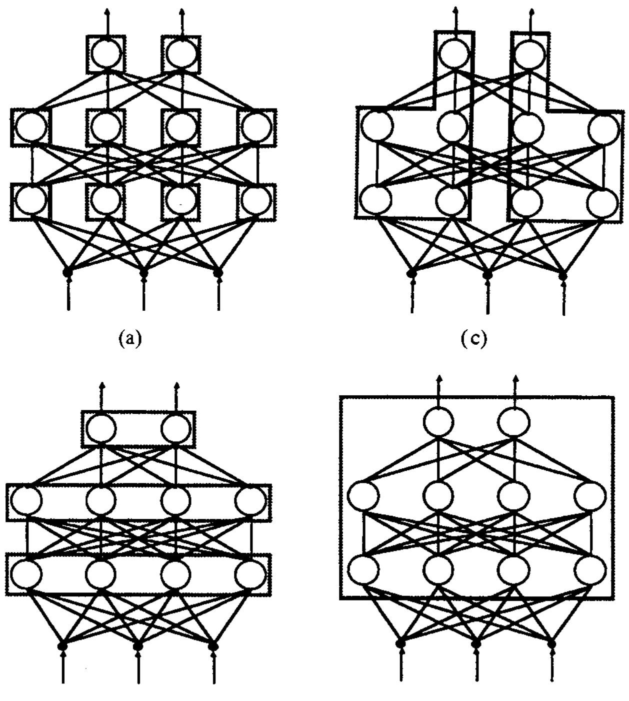


Fig. 2. Various possible assignments of neurons to processing elements having one look-up table: (a) single; (b) "slice"; (c) layer; (d) global.

between the continuous solution and the discrete one. During this scaling, weight absolute values are also restricted to be less than or equal to one. This approach has been proposed in [9] for digital filter design to find the starting point for discrete minimization of filter coefficients. Since every weight and offset is scaled down by the same constant value, the LUT has to be scaled up accordingly: if weights and offsets are scaled by the common factor A then the domain of the function to be computed using the LUT has to be scaled by the factor A^{-1}.

The second idea is to perform the learning, starting from the discretized continuous solution obtained by BP, again applying BP. It is well known that the BP algorithm computes the gradient of a quadratic error function that is the sum of the squares of the differences between the actual and desired outputs for each input/output pair of the training set [4]. The BP algorithm used in this phase estimates the gradient of this error function and updates weights and offsets applying a discretized BP formula. Offset values, which are continuous, are freely changed toward the steepest descent direction, while weights are changed only if the learning move prescribes a new value that, rounded to the closer value belonging to W_S or $W_{S,T}$, differs from the previous one.

Clearly, a good control of the learning rate η is crucial for the good behavior of the algorithm. In such a way, it is possible to reduce the error function and, hopefully, to reach convergence. We have seen that best results are obtained if the starting point is computed by iterating continuous BP learning up to the highest possible degree of convergence.

B. Formal Description

More formally, given a MLP with a certain number of layers and neurons per layer and a set (training set) of P input–output pairs $\{\boldsymbol{i}_p\boldsymbol{d}_p; p = 1, \cdots, P\}$, the problem is to find the weight and offset values that minimize the error between desired and computed outputs for all input–output pairs of the training set. Usually, the measures of such error are the average quadratic error E_2:

$$E_2 = \frac{\sum_{p=1}^{P} \left|\boldsymbol{d}_p - \boldsymbol{o}_p{}^{(M)}\right|^2}{PN_M} \tag{8}$$

and the minmax error E_X:

$$E_X = \max_{p,k}\left\{\left|d_{p,k} - o_{p,k}{}^{(M)}\right|\right\} \tag{9}$$

where $d_{p,k}$ is the kth component of $\boldsymbol{d}_p$ (the desired output of the kth neuron of the last layer, referring to the pth input–output pair), and $o_{p,k}{}^{(M)}$ is the kth component of $\boldsymbol{o}_p{}^{(M)}$.

In the following we assume that the whole NN is assigned to L LUT's and that suffix r refers to weights and offsets of neurons assigned to rth LUT, $r = 1, \cdots, L$. We also assume that the weight values belong to W_S, i.e., they are single power-to-two. The extension to $W_{S,T}$ (sum of two-power-of-two) is straightforward.

The learning procedure consists in the following step.

1) Make the network learn using the classical BP algorithm until convergence and until no more significative improvement can be achieved. Let $\vartheta'_k{}^{(s)}$ be the offset of the kth neuron of the sth layer, and $w'_{kj}{}^{(s)}$ be the weight related to the link from the jth neuron of the layer $(s-1)$ to the kth neuron of the sth layer of such a converged MLP.

2) Normalize weights: compute the maximum weight, W_r, (in absolute value) of each LUT:

$$W_r = \max_{r,j,k,s}\left\{\left|w'_{r,kj}{}^{(s)}\right|\right\} \tag{10}$$

where index r refers to the LUT and indexes k and s refer to neurons that use the rth LUT. Divide all weights and offsets by W_r:

$$w''_{r,kj}{}^{(s)} = \frac{w'_{r,kj}{}^{(s)}}{W_r}, \quad \vartheta''_{r,k}{}^{(s)} = \frac{\vartheta'_{r,k}{}^{(s)}}{W_r}. \tag{11}$$

In this way, all new weights $w''_{r,kj}{}^{(s)}$ will belong to the interval $[-1, 1]$.

3) For each LUT of the digital architecture, define the scalar function $e_r(B)$:

$$e_r(B) = \max_{r,k,j,s}\left\{\left|w''_{r,kj}{}^{(s)} - \frac{Q\left[Bw''_{r,kj}{}^{(s)}\right]}{B}\right|\right\} \tag{12}$$

where index r again refers to the LUT, indexes k and s refer to neurons that use the rth LUT, and the operator $Q\,[\cdot]$ denotes rounding to the nearest term belonging to W_S. B a further scale factor; $e_r(B)$ is a measure of the match between real weights and their rounded approximations belonging to W_S. Following an approach similar to that proposed in [9] for FIR filters design, $e_r(B)$ is minimized with respect to B and new weights and offsets are computed for the value B_r that minimizes $e_r(B)$:

$$w_{r,kj}{}^{(s)} = Q\left[B_r w''_{r,kj}{}^{(s)}\right], \quad \vartheta_{r,k}{}^{(s)} = B_r\vartheta''_{r,k}{}^{(s)}. \tag{13}$$

The cumulative scale factor for rth LUT is thus:

$$A_r = \frac{B_r}{W_r}. \tag{14}$$

rth LUT is then scaled accordingly to yield values corresponding to the nonscaled case.

4) Check the behavior of the neural network having the new weights and offsets $w_{r,kj}{}^{(s)}$ and $\vartheta_{r,k}{}^{(s)}$. If the new network performs the desired mapping between inputs and outputs with an error E_X within a given tolerance, the design procedure ends. If not, the following steps are performed.

5) Starting from the network having weights and offsets $w_{r,kj}{}^{(s)}$ and $\vartheta_{r,k}{}^{(s)}$, perform a cumulative BP step [10], adding the contribution of all input–output pairs belonging to the training set and compute for each neuron the quantities $d_k{}^{(s)}$ and $D_{kj}{}^{(s)}$:

$$d_k{}^{(s)} = \sum_{p=1}^{P} \delta_{p,k}{}^{(s)} \qquad s = M, \cdots, 1$$
$$D_{kj}{}^{(s)} = \sum_{p=1}^{P} \delta_{p,k}{}^{(s)} o_{p,j}{}^{(s-1)} \qquad k = 1, \cdots, N_s$$
$$j = 1, \cdots, N_{s-1} \tag{15}$$

where p refers to the pth input–output pair of the training set, $o_{p,j}{}^{(s-1)}$ is the corresponding output of the jth neuron of layer $(s-1)$, and $\delta_{p,k}{}^{(s)}$ has the meaning defined in (4), referring to the pth input–output pair of the training set. It is now possible to drop the index r, referring to the LUT, as it does not carry any more information.

6) Change the weights and offsets according to the formulae:

$$w_{kj}{}^{(s)} := Q\left[w_{kj}{}^{(s)} + \eta D_{kj}{}^{(s)}\right] \qquad s = M, \cdots, 1$$
$$\vartheta_k{}^{(s)} := \vartheta_k{}^{(s)} + \eta d_k{}^{(s)} \qquad k = 1, \cdots, N_s \tag{16}$$

where the right-hand-side term refers to the old value, while the left one is the new value. If the error E_2 is decreased after application of (16), η is increased by a factor M_η. On the contrary, if E_2 is increased, η is decreased by a factor D_η. In this way, η is automatically adapted to the problem, similarly to the approach proposed in [10]. The values of η control parameters actually used are:

$$M_\eta = 1.05, \quad D_\eta = 0.7. \tag{17}$$

This learning rate control strategy (a slight increase of η in case of success and a bigger decrease in case of failure) is fairly robust, and has been successfully tested by the authors, also in continuous BP learning. As the learning rate is self-adaptive, the actual values of M_η and D_η are not critical, provided that they do not differ too much from the given ones.

Fig. 3(a) shows a typical learning curve of the method (E_2 versus the number of iterations), while Fig. 3(b) shows the

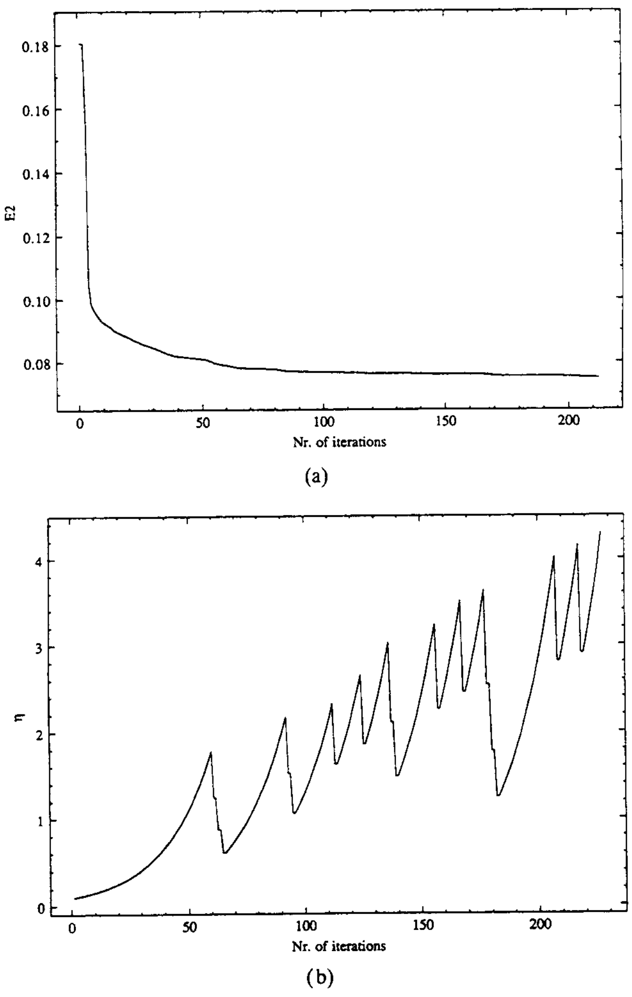


Fig. 3. (a) E_2 versus the number of iterations of the discrete learning algorithm during a typical learning; (b) Typical behavior of η value versus the number of iterations of the learning algorithm.

typical behavior of η during learning. It can be seen that η, starting from its initial value, quickly adapts to the problem, allowing the number of iterations to be reduced.

7) If no weights have been changed in the latest L_I iterations, the single weight having a maximum value of $\Delta w_{kj}{}^{(s)}$ is forced to change to the direction of $\Delta w_{kj}{}^{(s)}$. If this change causes a percentage increase in E_2 of more than P_2, the weight is reset to its old value. The values used for these additional parameters are:

$$L_I = 10, \quad P_2 = 15\%. \tag{18}$$

These figures proved successful in all test runs performed, most of which, however, converged without resorting to this forced change. For other problems it is possible that these parameters need a "tuning" depending on the particular problem. After the computation of steps 6 and 7, go to step 4.

If convergence cannot be achieved, two alternatives are still possible:

a) Locate the most "critical" weights and use for them an approximation with a higher number of power-of-two terms. A similar procedure has been presented for FIR filter design [6]. The most critical weights could be defined as those weights having the highest values of $\left|D_{kj}{}^{(s)}\right|$, according to (15) and (16).

b) Change the network topology, adding layers and/or neurons to existing layers, trying to find a better starting point for the discretization procedure.

However, convergence cannot be guaranteed, as in MLP's with continuous weights using the BP algorithm. The following section shows indeed that convergence has been achieved in most test cases considered.

V. Results

The design procedure has been tested using a pattern recognition problem, consisting in the recognition of character patterns encoded in a 8×8 pixle matrix, according to the IBM PC CGA character set. 95 different characters had to be recognized, corresponding to ASCII codes between 32 and 126. The number of input–output pairs, P, is thus 95, and for each pair the input is a vector of 64 binary values, corresponding to the pixel matrix representing a character, while the output is a vector of 8 binary values, representing its coded ASCII value. Therefore, the used MLP's have 64 inputs and 8 outputs.

All performed tests deal with binary values, while BP requires real values. We set all input–output pairs of values of the training set to 0.1 and 0.9 rather than to zero and one, respectively, to improve convergence [11].

Let us note that, although the CGA character set has been designed to maximize the difference between characters, there are still many characters very similar to each other. For instance: colon and semicolon; full stop and comma, the characters 1, I, [, |; and many others. Therefore, this pattern recognition problem is not straightforward.

A reduced test problem was also used, consisting in the recognition of the characters representing the ten digits. In this case, only four network outputs are significant.

In the tests, we considered various MLP's, with various topologies, various assignments of LUT's to neurons, and different values of S (see (6)).

A. MLP's Used in the Tests

In the tests we used two MLP's: the first with 64 inputs, 64 hidden neurons and 8 outputs (64/64/8) and the second with 32 hidden neurons (64/32/8). The assignments of PE's (and thus LUT's) to neurons, as discussed in Section III-B, are of four kinds: single (S), slice (C), layer (L), and global (G). In the case (C), we have 8 slices, each including 8 inputs, 8 or 4 hidden neurons and one output neuron. The considered values of the number of shifts, S, are 8, 4, and 1.

For the reduced problem with 10 input–output pairs, we used three different MLP topologies (with 64/64/4, 64/8/4 and 64/4 neurons, respectively), and many different values of S, from 8 to 0.

B. Continuous Learning

Step 1 of the learning algorithm prescribes to perform a continuous BP learning, until no further significant improve-

TABLE I
RESULTS OF CONTINUOUS BP LEARNING ON DIFFERENT MLP'S AND WITH TEST PROBLEMS INVOLVING THE RECOGNITION OF 10 AND 95 CHARACTERS OF CGA PC CHARACTER SET. THE LEARNING WAS STOPPED WHEN E_X REACHED THE VALUE OF 0.1.

MLP	Neurons in Layers			input–output	approx. n	
name	Input	Hidden	Output	pairs	of iterations	E_2
64/64/8 A	64	64	8	95	10000	0.0299
64/64/8 B	64	64	8	95	10000	0.0361
64/64/8 C	64	64	8	95	10000	0.0311
64/64/8 D	64	64	8	95	10000	0.0324
64/64/8 E	64	64	8	95	10000	0.0315
64/32/8 F	64	32	8	95	11000	0.0391
64/64/8 G	64	32	8	95	11000	0.0235
64/32/8 H	64	32	8	95	11000	0.0292
64/64/8	64	64	4	10	1000	0.0337
64/8/8	64	8	4	10	400	0.0325

ment can be made. In the presented cases, the learning stopped when E_X was less than 0.1 to limit the learning time.

For the 64/64/8 case, we considered five different networks, starting from different initial random weights, denoted by letters A, B, C, D, and E. Similarly, for the 64/32/8 case, we considered three different networks, denoted by letters F, G, and H.

Table I shows data about this continuous learning, referring both to 95 and 10 input–output pairs. The errors E_2 and E_X—see (8) and (9)—are reported for each case.

It is well known that the optimal number of hidden neurons of a MLP is difficult to determine. In the 95 characters problem, we found that the 64/64/8 MLP converges easily, while in some cases the 64/32/8 MLP has convergence problems, getting stuck in local minima.

C. Results with 95 Characters

In this tests, the tolerance ε, as defined in point 4) of Section IV-B, was set to 0.3, meaning that a value less than 0.4 is considered to be zero, and a value greater than 0.6 is considered to be one. Table II summarizes all test runs, reporting the average number of iterations and their standard deviation versus S, for all LUT assignments to neurons. Table III reports the starting value of E_X and the total number of iterations to converge for all test runs concerning the case with $S = 4$.

The results concerning the 64/64/8 MLP are very noisy, being characterized by a high standard deviation. However, convergence was always achieved in a number of iterations ranging from 48 to 472, lowering the absolute error E_X from an initial value of about 0.8 to less than 0.3. All kinds of LUT assignments to neurons seem to be equivalent, within one standard deviation, in the number of iterations, at least for S greater than one. Moreover, the number of iterations needed to converge seems lower for $S = 4$ than for $S = 8$, while one could expect the contrary, as the number of degrees of freedom decreases with S. Therefore, these results confirm the redundancy of the 64/64/8 MLP for this problem.

On the contrary, the results concerning the 64/32/8 MLP, limited to three tests for each value of S, show a clear

TABLE II
NUMBER OF ITERATIONS TO CONVERGE TO $E_X = 0.3$. (AVERAGE AND STANDARD DEVIATION) ON FIVE MLP'S 64/64/8 (A..E CASES) AND THREE MLP'S 64/32/8 (F..H CASES) VARYING LUT ASSIGNMENT TO NEURONS AND NUMBER OF MAXIMUM BIT SHIFTS S.

LUT Assignment	Value of S					
to Neurons	8		4		1	
64/64/8	avg.	s.d.	avg.	s.d.	avg.	s.d.
Single (S)	206	115	114	51	426	141
Slice (C)	107	48	305	140	==	==
Layer (L)	142	91	200	106	==	==
Global (G)	182	137	179	120	640	213
64/32/8	avg.	s.d.	avg.	s.d.	avg.	s.d.
Single (S)	180	37	332	110	==	==
Slice (C)	224	43	333	106	==	==
Layer (L)	216	52	355	111	==	==
Global (G)	194	105	397	142	==	==

TABLE III
STARTING VALUES OF E_X AFTER STEPS 2) AND 3) OF THE ALGORITHM, AND NUMBER OF ITERATIONS TO REACH $E_X = 0.3$ WITH THE DISCRETIZED BP PROCEDURE.

	N. of iterations for $E_X = 0.3$				Starting value of E_X			
LUT:	S	C	L	G	S	C	L	G
64/64/8;$S = 4$								
A	69	472	198	59	0.780	0.806	0.878	0.657
B	206	120	386	48	0.793	0.811	0.817	0.767
C	82	190	95	160	0.740	0.796	0.797	0.757
D	80	454	223	360	0.532	0.718	0.841	0.691
E	132	291	100	268	0.583	0.854	0.837	0.851
64/32/8;$S = 8$								
F	212	278	290	326	0.873	0.883	0.933	0.912
G	128	221	178	190	0.975	0.980	0.976	0.997
H	201	173	181	68	0.792	0.812	0.875	0.956
64/32/8;$S = 4$								
F	436	450	430	538	0.876	0.901	0.931	0.956
G	382	357	437	452	0.967	0.957	0.949	0.996
H	180	193	198	201	0.802	0.870	0.862	0.961

dependence of the MLP behavior on its degrees of freedom given by LUT assignment and shift levels. Convergence in the case with $S = 8$ is much better than in the case with $S = 4$, while with $S = 1$, the MLP's always fail to converge.

These results suggest that, as redundance becomes critical, convergence becomes slower and cannot be reached with extreme discretization.

These results, however, confirm the validity of the proposed algorithm and the realizability of NN's with power-of-two weights. In the case of extreme weight discretization, of course, a redundant MLP is needed to achieve convergence.

D. Asymptotic Learning

Beside the ability of the algorithm to reach prescribed value of E_X, it is also interesting to find the smallest reached value of E_X. As these tests need a lot of computer time, we performed most of them using the reduced test cases with 10 input/output pairs. The used MLP's span from a very redundant one (64/64/4 MLP) to the least redundant possible (64/4 MLP). In the tests, we considered also the extreme

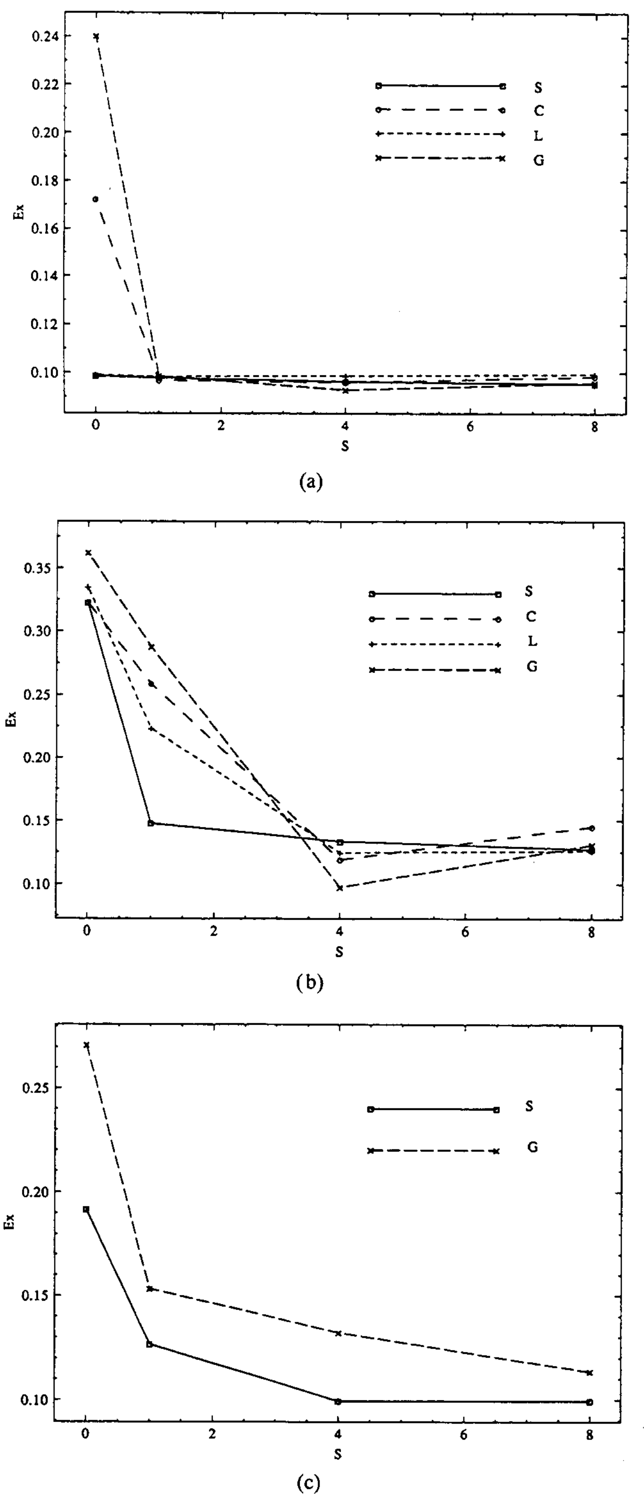


Fig. 4. Minimum values of E_X after asymptotic learning versus N, for the possible assignments of LUT's to neurons: single (S), "slice" (C), layer (L) and global (G); (a) MLP 64/64/4 with 10 input–output pairs in the training set; (b) MLP 64/8/4 with 10 input–output pairs in the training set; (c) MLP 64/64/8 with 95 input–output pairs in the training set.

discretization with $S = 0$, meaning that weights values must belong to the set $\{-1, 0, 1\}$.

Fig. 4(a)–(c) show the results for the MLP's 64/64/4, 64/8/4, and 64/4, respectively. Each figure reports the value of the minimum reached E_X versus S, for the various assignments of LUT's to neurons. It can be seen that, when the redundance of the MLP is high (high number of neurons and/or high value of S), the minimum value of E_X reached is approximately the same. As the redundance diminished, the minimum value of E_X increases and becomes higher as S and as the number of LUT's decreases, with the exception of the "layer" assignment (L), that performs in most cases better than the "slice" one (C), which has more LUT's. Presently, we have no explanation for this fact.

TABLE IV
MINIMUM VALUES OF E_X AND E_2 AFTER ASYMPTOTIC LEARNING FOR MLP 64/64/8 WITH 95 INPUT–OUTPUT PAIRS IN THE LEARNING SET.

LUT Assignment to neurons	Error	Value of N: 8	4
Single (S)	E_X	0.207	0.226
Single (S)	E_2	0.056	0.060
Global (G)	E_X	0.280	0.252
Global (G)	E_2	0.056	0.067

Table IV shows the results of similar test cases, for the 64/64/8 MLP with 95 characters, which confirm this view.

VI. GENERALIZATION CAPABILITY

To perform further tests on the performance of the proposed algorithm, and to verify the generalization capability of the discretized MLP, another experiment has been carried out. This experiment concerns the well known problem of classifying points inside and outside a circular region in the plane, as reported in [12].

Formally, the problem consists in the building of a MLP with 2 inputs and 1 output, which, given the x, y coordinates of a point in the Cartesian plane, is able to recognize if this point lies inside or outside a unit-radius circle centered at the origin. The network output is zero if the point lies outside, one if it lies inside. In practice, the experiment is performed choosing a suitable MLP topology, creating a finite-size learning set composed of randomly selected points inside and outside the circle, and applying the BP algorithm until the network converges within a given accuracy. As in [12], a MLP with one hidden layer of 8 neurons has been used in this experiment, while the learning set is composed of 200 points, 100 points randomly selected inside the circle, and 100 points randomly selected outside the circle, but within the square region bounded by $|x| < 5$ and $|y| < 5$.

To test the generalization capabilities of the resulting network, the MLP has been requested to classify all the points inside the region described by $|x| < 5$ and $|y| < 5$. In practice, a dense grid of points was used to this purpose. The obtained set of output values can be used to produce a contour plot, where each curve corresponds to a given output level. Any point inside the curve produces an output level which is equal to, or greater than the given level. A typical plot for the MLP after convergence is reported in Fig. 5, where the four curves represent the levels 0.2, 0.4, 0.6, and 0.8 from the outer to the inner. As can be seen, the contour plots resemble very much the target unit circle, especially the 0.8 level innermost curve.

The proposed algorithm has been tested on this problem varying LUT assignment and discretization levels. For each

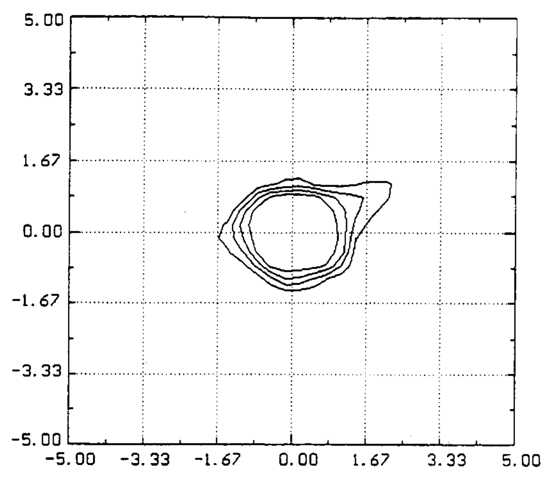


Fig. 5. Contour plot for the unit-circle recognition problem after continuous learning. The curves represent the MLP output levels 0.2, 0.4, 0.6, and 0.8, from the outer to the inner (an output of 0.1 means "outside" while 0.9 means "inside").

LUT assignment and discretization level, many trials were performed with different initial sets of weights. In all test runs, a good accordance between the overall degree of freedom (LUT number and discretization levels) and the generalization capability of the network has been obtained.

For the sake of brevity, only a few graphical results are reported. Fig. 6(a) shows the performance of the MLP discretized using a global LUT (the worst case of LUT assignment) and one power-of-two term with $S = 0$ (the worst case of level discretization, the weights being constrained to be -1, 0, or 1). Although the network behavior is worse than that of the original MLP (see Fig. 5), the discretized network obtained with the proposed algorithm retains a good generalization capability as far as the inner levels are concerned. A comparison of this plot with Fig. 6(b) which reports the performance of the network after steps 2) and 3) of the algorithm, but before the additional discretized BP procedure of steps 5) to 7) clearly highlights the importance of this phase to the overall performance.

Fig. 7 reports the result of the MLP discretized using a global LUT and a single power-of-two term with $S = 3$. The generalization capability and the effectiveness of the proposed algorithm are confirmed.

VII. Discussion and Conclusions

In this paper, we proposed the use of weights with values limited to power-of-two for a fast digital implementation of NN's, and presented a learning algorithm for MLP's with such weights. The proposed discretized MLP and the learning algorithm performed well in all test cases considered and demonstrated a good generalization capability. The main drawback of the algorithm is the need for off-line computation, since learning requires multiplications.

A discrete power-of-two MLP is, of course, less powerful than a classical, real-valued MLP of the same size. The higher the number of admissible power-of-two values, and the more different LUT's are available, the more powerful the discrete MLP will be. These considerations are confirmed by the tests

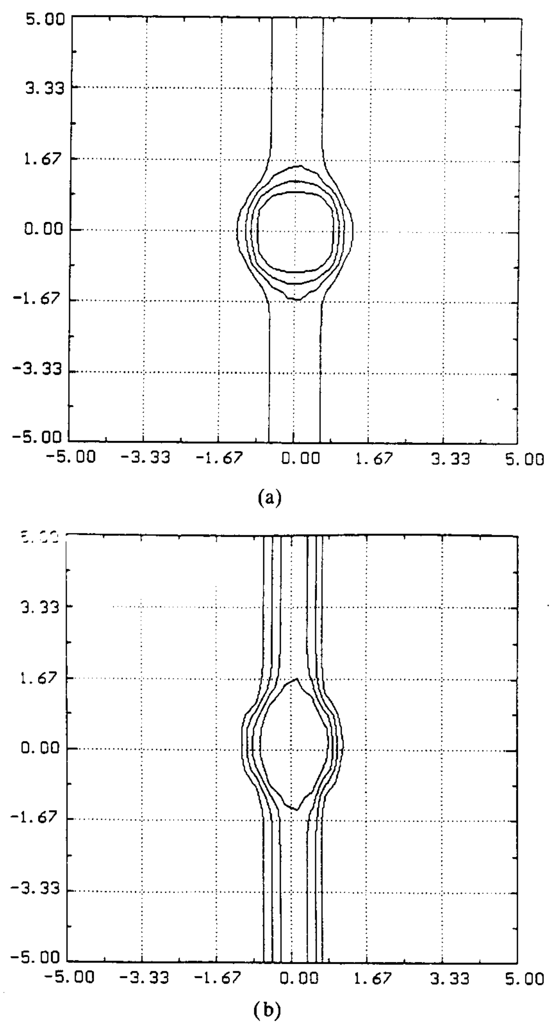


Fig. 6. Unit-circle recognition problem solved with a power-of-two MLP with $S = 0$ (the weight values can only be: -1, 0, or 1) and global LUT. (a) Contour plot after discrete learning; (b) contour plot after rounding off the continuous solution. (The curves have the same meaning as Fig. 5).

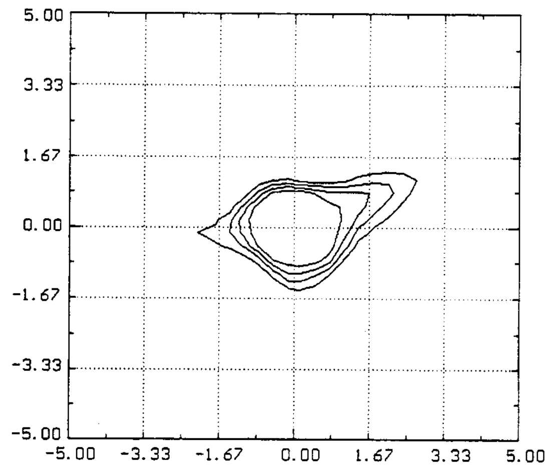


Fig. 7. Unit-circle recognition problem solved with a power-of-two MLP with $S = 3$ and global LUT. (The curves have the same meaning as Fig. 5).

made: if the real-valued MLP is not redundant at all for solving a given problem, its power-of-two counterpart will exhibit a difficult convergence, if not at all. If the real-valued MLP is redundant, the power-of-two counterpart exhibits a clear improvement in its performance, increasing the number of admissible power-of-two, S, and the number of LUT's. Eventually, when the real-valued MLP is very redundant, the corresponding power-of-two network shows the typical learning behavior of very redundant MLP's: the number of iterations needed to learn is very variable and often worse than for a less redundant network. In this latter case, discrete MLP's exhibit a limiting behavior that does not depend on S or on the LUT number, being affected by "noise" due to discretization. On the other hand, a MLP which is redundant enough can learn a given behavior using only one or a few LUT's, even with extreme weight discretization ($S = 0$).

In the presented tests we concentrated on single power-of-two weights, although the learning algorithm can deal also with multiple power-of-two. In fact, we also performed some tests on the character recognition problem using MLP's with weights whose values were the sum of two power-of-two terms, and S was varied between 7 and 9. Such weight values are much closer to continuous ones than single power-of-two weights, and we could expect a discretized MLP behavior very similar to a real-valued MLP. All test cases computed confirmed this view, presenting a much better convergence to the solution, and often showing a satisfactory behavior immediately after weight normalization and rounding, with no need to perform further learning iterations.

The idea of using power-of-two weights to avoid multiplications in NN's was exploited also for the Neocognitron neural model [1]. Owing to Neocognitron characteristics, it was possible to avoid multiplications also in the learning phase, thus taking full advantage of this technique.

Reference [13] reports a analysis of the effects of quantization on learning and generalization in NN's. The test cases presented there refer to a classification problem, where the performance of the network is measured in terms of the percentage of patterns correctly classified. However, the qualitative relationship among quantization weight level, number of hidden units (network redundancy), and network performance is similar to that previously presented and discussed.

Other approaches to NN's with discrete weights are reported in [14], where an analog VLSI model of MLP (feedforward and recurrent) with discrete weights is presented, together with a discrete learning algorithm (weight perturbation); and in [15], where extremely discretized NN's with binary weights are proposed, together with a learning procedure (CHIR) based on choosing internal representations. These approaches are not easily comparable with our approach, since in essence they are neural models not based on back propagation. Moreover, the former is based on an analog VLSI implementation, while our proposal is aimed at a digital implementation for the reasons explained in the introduction.

In conclusion, we believe that the power-of-two weight MLP could be good VLSI implementation alternative for applications requiring extremely fast data processing, but not requiring in-line learning or relearning. Although restricted to feed-forward processing, the area-delay product of such neural VLSI chips could be improved by two orders of magnitude, or more, over traditional implementations using multipliers [1]. Examples of possible fields of application could be the same as power-of-two digital filters: DSP of wide-band signals, nonlinear channel equalization, real-time image processing, and others.

ACKNOWLEDGMENT

The authors wish to thank Prof. N. Benvenuto who first introduced them to the study of powers-of-two quantization, and Mr. L. Pollonara who wrote and debugged most of the programs used to perform the tests.

REFERENCES

[1] B. A. White and M. I. Elmasry, "The digi-neocognitron: a digital neocognitron neural network model for VLSI," *IEEE Trans. Neural Networks*, vol. 3, pp. 73–85, Jan. 1992.

[2] Y. C. Lim and B. Liu, "Design of cascade from FIR filters with discrete valued coefficients," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-36, pp. 1735–1739, 1988.

[3] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing, vol. 1, Foundations*. Cambridge, MA: MIT Press, 1986.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing vol. 1, Foundations. Cambridge, MA: MIT Press, 1986.

[5] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Multi-layer perceptrons with discrete weights," in *Proc. IEEE IJCNN*, San Diego, CA. 1990, vol. II, pp. 623–630.

[6] Z. Jiang, "FIR filter design and implementation with powers-of-two coefficients," in *Proc. IEEE ICASSP'89*, Glasgow, U.K., May 1989, pp. 1239–1242.

[7] S. Y. Kung and J. N. Hwang, "Parallel architectures for artificial neural nets," in *Proc. IEEE Int. Conf. Neural Networks*, San Diego, CA, 1988, vol. II, pp. 165–172.

[8] F. Piazza, M. Marchesi, G. Orlandi, and A. Uncini, "Coarse-grained processor array implementing the multi-layer neural network model," in *Proc. IEEE ISCAS'90*, New Orleans, LA, May 1990, pp. 2963–2965.

[9] Q. Zhao and Y. Tadokoro, "A simple design of FIR filters with powers-of-two coefficients," *IEEE Trans Circ. Syst.*, vol. CAS-35, pp. 566–570, 1988.

[10] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of the back-propagation method," *Biol. Cybern.*, vol. 59, pp. 257–263, 1988.

[11] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[12] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, 1987.

[13] Y. Xie and M. A. Jabri, "Analysis of the effects of quantization in multilayer neural networks using a statistical model," *IEEE Trans. Neural Networks*, vol. 3, pp. 334–38, 1992.

[14] M. A. Jabri and B. Flower, "Weight Perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer network," *Neural Computation*, vol. 3, pp. 546–565, 1991.

[15] D. Nabutowsky, T. Grossman, and E. Domany, "Learning by CHIR without storing internal representations," *Complex Systems*, vol. 4, pp. 519–541, 1990.

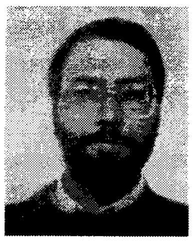
Michele Marchesi graduated from the University of Genova, Italy, in 1975 with a degree in electronic engineering and in 1980 with a degree in applied mathematics.

From 1976 to 1987 he was a Researcher at the Institute for Electronic Circuits, C.N.R., Genova. From 1987 to 1990 he was an Associate Professor at University of Ancona, Italy. Since November, 1990, he has been an Associate Professor of Network Theory in the Department of Biophysical and Electronic Engineering at the University of Genova. His main fields of interest are optimization techniques applied to circuit design and neural networks.

Gianni Orlandi (S'69, M'82) was born in Rome, Italy, in 1947. He received the degree in electrical engineering from the University of Rome "La Sapienza" in 1972.

From 1972 to 1978 he was with "U. Bordoni" Foundation, in Rome, where he worked on circuit theory and digital signal processing. In 1978 he joined the Department of Information and Communication of the University of Rome "La Spaienza," where he was first Assistant Professor and then in 1983, Associated Professor of electrical engineering. From 1986 to 1989 he was Full Professor in the Department of Electronics and Automatics at the University of Ancona, Italy, and since 1989 has been Full Professor of electrical engineering in the Department of Informations and Communications at the University of Rome "La Sapienza." His research interests are in the areas of circuit theory, digital signal processing, parallel algorithms, VLSI parallel architectures, and neural networks. He is author of over 100 publications on these topics.

Professor Orlandi is member of INNS.



Francesco Piazza (M'87) was born in Jesi, Italy, in 1957. He received the degree with honors in electrical engineering in 1981 from the University of Ancona, Italy.

From 1981 to 1983 he worked on CCD-image processing in the Physics Institute of the University of Ancona. From 1983 to 1984 he worked at the Olivetti OSAI software development center and was a consultant in the field of microprocessor-based systems and radar displays. In 1985 he joined the Department of Electronics and Automatics of the University of Ancona, as Researcher in electrical engineering. He is presently, an associate professor of electrical engineering. His research interests are in the areas of circuit theory, digital signal processing, parallel algorithms, VLSI architectures, and neural networks.

Aurelio Uncini (M'88) was born in Cupra Montana, Italy, in 1958. He received a "laurea" degree in electrical engineering from the University of Ancona, Italy, in 1983.

From 1984 to 1986 he was with the Fondazione U. Bordoni, Rome, Italy, engaged in research on digital speech processing. Since 1987 he has been a researcher associated with the Department of Electronics and Automatics at the University of Ancona. His research interests include digital signal processing, digital circuit theory, spectrum analysis, and neural networks.