Reprinted from

Proceedings of IJCNN International Joint Conference on Neural Networks,
San Diego (CA-USA) pp II-623- II-630 June 1990

Multi-layer Perceptrons with Discrete Weights

M. Marchesi, G. Orlandi(*), F. Piazza, L. Pollonara, A. Uncini,

Dipartimento di Elettronica e Automatica - University of Ancona
Via Brecce Bianche, 60131 Ancona-Italy.
Fax:+39 (071) 2204464 - email: aurel@eealab.unian.it - Internet: http://nnsp.eealab.unian.it/

(*) Dipartimento INFOCOM - University of Rome "La Sapienza"
Via Eudossiana 18, 00184 Rome – Italy.

MULTI-LAYER PERCEPTRONS WITH DISCRETE WEIGHTS

M. Marchesi, G. Orlandi[*]*, F. Piazza, L. Pollonara, A. Uncini*

Dipartimento di Elettronica e Automatica - Università di Ancona
via Brecce Bianche - 60131 Ancona, Italy

[*] Dipartimento INFOCOM - Università di Roma "La Sapienza"
via Eudossiana, 18 - 00183 Roma, Italy

ABSTRACT

Digital VLSI neural networks are currently very promising for implementing high performance versions of such networks. In such digital circuits the multiply phase of neuron activity is crucial for yielding high performance. This paper studies the feasibility of restricting the weight values in multi-layer perceptrons to powers-of-two or sums of powers-of-two. Multipliers could be thus substituted by shifters and adders on digital hardware, saving both time and chip area, under the assumption that the neuron activation function is computed through a look-up table (LUT), and that a LUT may be shared among many neurons.
A learning procedure based on back-propagation to obtain a neural network with such discrete weights is presented. This learning procedure requires full real arithmetic and therefore must be performed off-line. It starts from a multi-layer perceptron with continuous weights learned using back-propagation. Then a weight normalization is made to ensure that the whole shifting dynamics is used and to maximize the match between continuous and discrete weights of neurons sharing the same LUT. Finally, a discrete version of BP algorithm, with automatic learning rate control is applied up to convergence. Some test runs on a simple pattern recognition problem show the feasibility of the approach.

INTRODUCTION

Neural networks are becoming increasingly used in many areas of signal and image processing, owing to their intrinsic parallelism their ability to learn by examples and their fault tolerance [1,2]. The main approaches to their hardware implementation are analog, mixed analog-digital and fully digital VLSI circuits.

Limiting our attention to digital architectures, one can easily realize that one of their bigger implementation problems, regarding performance and chip area, is related to the multiply phase of the neuron activity. In almost every neural model, in fact, the elementary processor (neuron) computes a weighted sum of its inputs and then applies on it a nonlinear activation function. In a digital implementation weights are stored and recalled easily in a local memory and the activation function can as well be easily computed through look-up tables. Sums are performed quickly and require a limited amount of hardware. The bottleneck are the multiplications between inputs and weights: they are slow compared to other operations and multipliers require a lot of chip area, especially if they operate in parallel.

This work has been supported in part by Consiglio Nazionale delle Ricerche of Italy and in part by Ministero della Pubblica Istruzione of Italy.

A possible solution of this problem was already exploited in digital filter design: it consists on using weights that are powers-of-two, or sums of powers-of-two terms [3,4]. In this way, it is possible to substitute multiplications with simple shifts or shift-and-add operations, that are much quicker and require less hardware.

In this paper we both discuss the feasibility of this approach also for multi-layer neural networks with binary outputs and present a learning procedure based on back-propagation (BP) [5] to obtain a neural network with discrete weights. It is worth noting that the learning procedure requires full real arithmetic and therefore must be performed off-line. Once the weights are computed for a specific problem, they can be loaded into the multi-layer neural network that will be run in forward mode, processing the data.

MULTI-LAYER PERCEPTRON

Our work focuses on the so called Multi-Layer Perceptron (MLP), schematically shown in Fig. 1, a neural network model widely studied and used. In the MLP, the neurons are grouped in sequentially connected layers, each layer being numbered from 0 to N. The number of neurons belonging to each layer is N_s.and neurons belonging to layer s are numbered in sequence, from 1 to N_s. Each neuron is connected to all the neurons of the two adjacent layers, and to no other neuron. The neurons of layer 0 (input layer) do not perform computations, but only feed input signals into the neurons of layer 1. Layer N is the output layer, where the response of the network come from. Each neuron of MLP is characterized by one output and many inputs, which are the outputs of the neurons of the preceding layer. Let $net_k^{(s)}$ denote the value of k-th neuron of s-th layer. The weighted sum of the inputs is computed by neurons according to the formula:

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

where $w_{kj}^{(s)}$ is the weight by which the same neuron multiplies the output, $o_j^{(s-1)}$, of the j-th neuron of the previous layer and $\vartheta_k^{(s)}$ is the offset of k-th neuron of s-th layer. The neuron output is computed by applying a non linear, bounded function f (often the sigmoid function), to the weighted sum of its inputs:

$$o_k^{(s)} = f(net_k^{(s)}) \tag{2}$$

When processing data, N_0 input signals are fed into the network by input neurons and for each layer the outputs are computed and fed into the neurons of the next layer, up to the last one, accordingly to equations (1) and (2). The weights are kept constant and their values are precisely what determines the network behavior and its capability to correctly process the signals.

The Back Propagation, is one of the most widely used algorithms in MLP networks to train the network, i.e. to compute the values of the weights in order to obtain a required network behavior. A network behavior consists of a set of input/output pairs, each composed by N_0 input signals i_k ($k=1, \dots, N_0$), and N_N corresponding output signals d_j ($j=1,\dots,N_N$). These pairs are the desired behavior of the system.

In a fast digital implementation, the activation function f(.) is computed through a look-up table (LUT) contained in every processing element (PE). Depending on the granularity of the neural network, 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) a distinct LUT for every neuron;

b) a LUT common to all neurons of each layer;

c) a LUT common to all neurons of some vertical "slice" of the MLP;

d) a LUT common to the whole neural network.

Figs. 2a-2d graphically show the four possible assignments of MLP neurons to PE's and thus possible LUT assignments. Cases "a" and "d" are straightforward, while cases "b" and "c" are derived from architectures implementing MLP found in literature [6,7].

LUT assignment to neurons is important because each LUT allows a distinct rescaling of all the weights of the neurons using that LUT, and rescaling is a crucial step in the proposed learning algorithm (see step 2 of learning algorithm in the next section). A neural network has more degrees of freedom with a greater number of LUT's, making faster convergence in learning, as test cases presented in next sections will show. Of course, different assignments of neurons to PE's can be found, but the presented learning algorithm can be easily adapted to deal with any possible assignment.

THE LEARNING ALGORITHM

The proposed learning procedure works for weights that are simple powers-of-two or sums of two (or more) powers-of-two. Let us denote with $\mathcal{W}_{M,N}$ the set of admissible weights values:

$$\mathcal{W}_{M,N} = \{ x \mid x = \sum_{k=1}^{M} R_k 2^{-p_k} , \ |x| \leq 1; R_k \in \{-1,0,1\}, \ p_k \in \{0,1,..,N\} \} \qquad (3)$$

where M is the number of powers-of-two terms and N is the number of bits that can be shifted. This definition implies that the weights absolute value must be ≤ 1. The offsets ϑ's are real numbers, since they are involved only in sums.

Briefly, the learning procedure consists of the following steps:

1. Make the network learn using the classical BP algorithm, until convergence. Let $\{w_{kj}^{(s)}, j=1,...,N_{s-1}\}$, and $\vartheta_k^{(s)}$ be the weights and offset associated to k-th neuron of layer s after convergence.

2. Normalize weights: for the neurons referring to h-th LUT, compute the maximum weight (in absolute value):

$$W_h = \max \{ |w_{kj}^{(s)}| \}. \qquad (4)$$

where k and s are restricted to span only on neurons using h-th LUT to compute f(.). Divide all these weights and offsets by W_h. In this way, all new weights $w'^{(s)}_{kj}$ will belong to the interval [-1, 1].

3. For k and s restricted to span only on neurons using h-th LUT, we define the scalar function $E_h(A)$:

$$E_h(A) = \sum_s \sum_k \sum_{j=1}^{N_{s-1}} (w'^{(s)}_{kj} - A^{-1} < w'^{(s)}_{kj} A >)^2 \qquad (5)$$

where the operator <.> denotes rounding to the nearest term belonging to $\mathcal{W}_{M,N}$.and A is a suitable scaling factor. Note that $E_h(A)$ is a measure of the matching between real weights and their rounded approximations belonging to $\mathcal{W}_{M,N}$. Following the same approach proposed in [8] for FIR filters design, $E_h(A)$ is minimized in the interval [0.1, 5] and new weights and offsets are computed for the value A_h that minimizes $E_h(A)$:

$$w''^{(s)}_{kj} = < A_h \, w'^{(s)}_{kj} > \qquad (6a)$$

$$\vartheta''^{(s)}_k = A_h \, \vartheta'^{(s)}_k \qquad (6b)$$

According to this normalization, the domain of the activation function f(.) is properly scaled to yield values corresponding to the non-normalized case.

4. Check the behavior of the neural network having the new weights and offsets $w''^{(s)}_{kj}$ and $\vartheta''^{(s)}_k$. If the outputs of the network must be binary, as it is usually the case, the network real outputs are considered to be zero if < 0.4, and to be one if > 0.6. If the new network still performs the desired mapping between inputs and outputs, the design procedure ends. Otherwise, the following steps are performed.

5. Starting from the network having weights and offsets $w''^{(s)}_{kj}$ and $\vartheta''^{(s)}_k$, perform a BP step, adding the contribution of all input-output pairs belonging to the learning set, and compute for every neuron the quantities $\delta^{(s)}_k$ and $\Delta^{(s)}_k$:

$$\delta^{(s)}_k = \Sigma_p \, \delta^{(s)}_{pk} \qquad (7a)$$

$$\Delta^{(s)}_k = \Sigma_p \, \delta^{(s)}_{pk} \, o^{(s)}_{pk} \qquad (7b)$$

where p refers to p-th input-output pair of the learning set, while $o^{(s)}_{pk}$ is the corresponding output of k-th neuron of layer s and $\delta^{(s)}_k$ has been defined in [5] and is associated to.k-th neuron of layer s.

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

$$w''_{kj}{}^{(s)}(t+1) = < w''_{kj}{}^{(s)}(t) + \eta \Delta_k{}^{(s)} > \tag{8a}$$

$$\vartheta''_k{}^{(s)}(t+1) = \vartheta''_k{}^{(s)}(t) + \eta \delta_k{}^{(s)} \tag{8b}$$

If no term is significantly changed after application of (8), η should be increased. After the computation of the new values, go to step 4.

If convergence cannot be achieved, the network topology should be changed adding layers and/or neurons to the existing structure and repeating the whole learning procedure.

RESULTS AND CONCLUSIONS

We report preliminary results concerning a simple pattern recognition problem used as test case. It consists in recognizing ten input patterns representing the ten digits encoded in 8x8 pixel matrices according to the PC CGA graphic character set. The neural network has 64 inputs, 8 intermediate neurons and 4 output neurons (required to output a binary representation of every input digit). In this problem, input and output values are binary. As BP needs a continuous neuron activation function (in our case the sigmoid function), input and output values of the training set are equal to 0.1 and to 0.9 in correspondance of the binary zero and one, respectively.

The problem deals only with the case $M = 1$, that is simple powers-of-two weights (this is also the best situation for hardware implementation). The values of N, however, are varied from 8 to zero in the tests performed. The $N=0$ case means that only the values of $\{-1, 0, +1\}$ are allowed for the weights, thus eliminating also the shift.

We made test runs for each LUT assignment to neurons, as reported in Fig. 2. In the case of Fig. 2c we considered four vertical "slices", each with one output neuron, 2 intermediate neurons and 16 inputs. The error measure is the maximum difference (in absolute value) between computed and desired outputs for each pair of the training set. The continuous BP learning of step 1 has been the same for all test cases: it took about 400 iterations to converge and resulted in an error of 0.094. The discrete learning was stopped when the error became lower than 0.3. The results of the test runs for each different LUT assignment are reported in Tables I and II. The proposed algorithm was able to find a solution in all test cases but those having weights that can only assume values of $\{-1, 0, 1\}$. In some test cases having high values of N the network with discrete weights converges after step 3, without further learning. As the maximum number of bits that can be shifted, N, decreases, the number of iterations needed to find a solution increases, with few exceptions. This is due to the fact that for low values of N, the weights are very coarse and the overall problem is more difficult than for higher values of N.

The number of iterations to reach convergence of another test case is presented in Table III. It regards a MLP of 64-64-4 neurons, trained with the same 10 input patterns of the previous example. Only the most significant cases, those with $N = 1$ and $N = 0$, are presented. Two runs are reported for every value of N, corresponding to different starting values of the learning rate η. Convergence is always achieved, at an higher computational cost for $N = 0$.

Presently, work is carried on considering harder test cases, enhancing the learning rate control strategy and studying the possibility of having the neuron outputs and not the weights as sums of powers-of-two. We are also studying modifications to other neural network models to avoid multiplications both during data processing and learning, thus increasing the speed and the integrability of such models.

REFERENCES

[1] D.E. Rumelhart and J.L. McClelland, "Parallel Distributed Processing", Vol. I, The MIT Press, Cambridge, USA, 1986.
[2] R.P. Lippman, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, April 1987, pp. 4-22.
[3] Y.C. Lim, S.R. Parker, "FIR Filters Design Over a Discrete Powers-of-two Coefficients Space", IEEE Trans. ASSP,, vol. ASSP-31, 1983, pp. 583-591.
[4] Y.C. Lim, B. Liu, "Design of Cascade Form FIR Filters with Discrete Valued Coefficients", IEEE Trans. ASSP, vol. ASSP-36, 1988, pp.1735-1739.
[5] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning Internal Representations by Error Propagation", in [1].
[6] F. Piazza, M. Marchesi, G. Orlandi, A. Uncini, "An Efficient Mapping of Neural Networks to a Linear Architecture of Coarse-Grained Processors", Proc. IEEE ISCAS '90, New Orleans, May 1990 (this issue).
[7] S.Y. Kung and J.N. Hwang, "Parallel Architectures for Artificial neural Nets", Proc. IEEE Int. Conf. on Neural Networks, San Diego, California, 1988, pp. II-165 - II-172.
[8] Q. Zhao, Y. Tadokoro, "A Simple Design of FIR Filters with Powers-of-two Coefficients", IEEE Trans. CAS, vol CAS-35, 1988, pp. 566-570.

Table I. The value of the maximum error after normalization

	Value of N			
LUT assignment	8	4	1	0
Single neuron	0.17	0.16	0.44	0.50
Single layer	0.21	0.18	0.57	0.54
Vertical "slice"	0.32	0.31	0.54	0.67
Whole network	0.42	0.20	0.46	0.71

Table II. Number of iterations to converge to an error ≤ 0.3

	Value of N			
LUT assignment	8	4	1	0
Single neuron	0	0	42	2438
Single layer	0	0	91	NO
Vertical "slice"	5	4	35	NO
Whole network	8	0	38	NO

Table III. Number of iterations to converge to an error ≤ 0.3

	$\eta_{start} = 0.1$		$\eta_{start} = 20$	
LUT assignment	N=1	N=0	N=1	N=0
Single neuron	67	91	16	41
Single layer	62	107	8	51
Vertical "slice"	64	120	10	35
Whole network	65	199	27	66

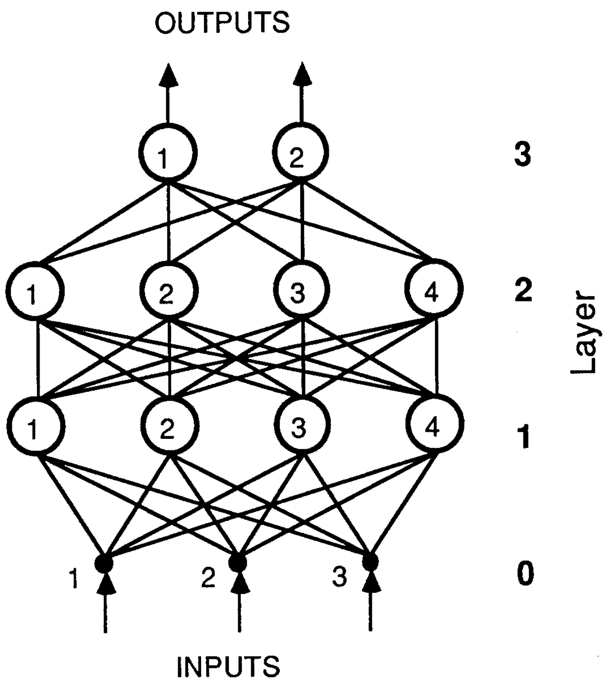

Fig. 1. The Multi Layer Perceptron.

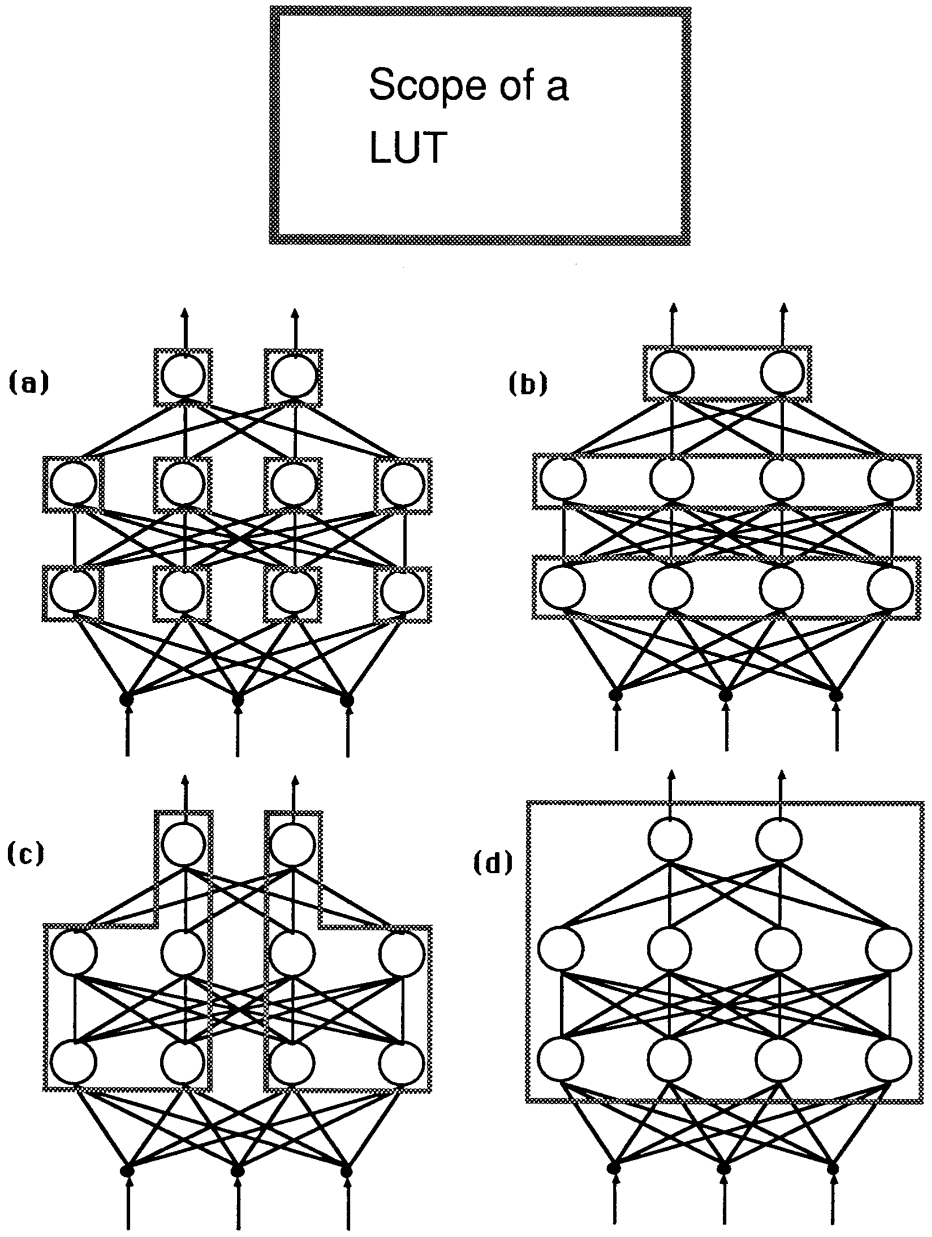

Fig. 2. Possible assignments of MLP neurons to look-up tables.