Reprinted from

IEE Proceedings Circuits Devices and Systems, Vol. 143, No 4, August 1996.

Backpropagation without multiplier for multilayers neural networs

M.L. Marchesi(*), Francesco Piazza(**) and Aurelio Uncini(**)

(*) Dipartimento di Ingegneria Biofisica ed Elettronica – University of Genova, Italy

(**) Dipartimento di Elettronica e Automatica – University of Ancona, Italy
Via Brecce Bianche – I60131 Ancona, Italy

Backpropagation without multiplier for multilayer neural networks

M.L. Marchesi
F. Piazza
A. Uncini

Indexing terms: Neural networks, VLSI Technology, Systems engineering, Learning algorithms

Abstract: When multilayer neural networks are implemented with digital hardware, which allows full exploitation of the well developed digital VLSI technologies, the multiply operations in each neuron between the weights and the inputs can create a bottleneck in the system, because the digital multipliers are very demanding in terms of time or chip area. For this reason, the use of weights constrained to be power-of-two has been proposed in the paper to reduce the computational requirements of the networks. In this case, because one of the two multiplier operands is a power-of-two, the multiply operation can be performed as a much simpler shift operation on the neuron input. While this approach greatly reduces the computational burden of the forward phase of the network, the learning phase, performed using the traditional backpropagation procedure, still requires many regular multiplications. In the paper, a new learning procedure, based on the power-of-two approach, is proposed that can be performed using only shift and add operations, so that both the forward and learning phases of the network can be easily implemented with digital hardware.

1 Introduction

Digital neural network architectures have received increasing attention during recent years, due to the immediate availability of the VLSI and ULSI technology for their design and implementation [1]. Very recently, some multiplierless digital architectures have been proposed for the implementation of some neural network (NN) models [1–4]. Such architectures are based on the property that, in binary representation, the multiplication between two integer numbers can be substituted by a shift, if one of them is a power of two. If a representation with a sum of two power-of-two terms is preferred, which is obviously more accurate, a multiplication with such a number can still be substituted by two shifts and an addition. The advantages of such implementations, which were first introduced in digital filter design [5], that they are much faster and have a much smaller chip area, whereas multipliers are slow and require large areas on chip with respect to shift registers.

A digital multiplierless architecture, the digi-neocognitron, able to implement the complete neocognitron NN model has already been proposed [1]. On the other hand, multiplierless implementations of the multilayer perceptron (MLP) NN model have been proposed only for its forward phase, when data are processed, but not for its learning phase, which must be performed off-line on a computer able to perform multiplications [2, 4].

The learning algorithm for the MLP that is almost universally used for its training is backpropagation (BP) [6], which has the LMS algorithm for adaptive filters as the linear counterpart. For such linear adaptive filters, a complete multiplierless implementation has been already proposed [7].

In this paper we propose a MLP learning algorithm based on BP, which does not use multiplications, but only shift-and-add and rounding operations, and can be performed on fast digital hardware. The whole approach is mostly devised for a VLSI implementation of the network using many parallel processing elements. An example of such architecture could be the linear data-driven array of processors proposed in [8], or other similar architectures [9].

2 Method

2.1 Multilayer perceptrons

In the followings we describe briefly the well-known MLP neural model [6]. Refer to [3] for a more detailed description of such a model and of the notation used. In forward mode, data are processed by each neuron according to the following formulae:

$$net_k^{(s)} = \sum_{j=1}^{N_{s-1}} w_{kj}^{(s)} o_j^{(s-1)} + \theta_k^{(s)} \qquad k = 1, .., N_s;\ s = 1, .., M \tag{1}$$

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

where s is the layer index (from 0 to M); N_s is the number of neurons of the sth layer; $o_k^{(s)}$ denotes the output of the kth neuron of the sth layer and $net_k^{(s)}$ is its weighted sum, $w_{kj}^{(s)}$ is the weight by which the same neuron multiplies the output $o_j^{(s-1)}$ of the jth neuron of

IEE Proceedings online no. 19960336

Paper first received 8th March and in revised form 11th December 1995

M.L. Marchesi is with the Dipartimento di Ingegneria Biofisica ed Elettronica, Università di Genova, Italy

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

the previous layer; $\theta_k^{(s)}$ is the offset of the kth neuron of the sth layer; $f(.)$ is the activation function.

If MLP weight values are restricted to power-of-two, and if the activation function is computed through a look-up table (LUT), no explicit multiplication is needed in the forward phase. To see how it is possible to avoid multiplications in the overall learning, let us consider the backward move:

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

$$\delta_k^{(s)} = \sigma_k^{(s)} f'(net_k^{(s)}) \quad k = 1,..,N_s;\ s = M-1,..,1 \quad (4)$$

$$\begin{cases} \Delta w_{kj}^{(s)} = \eta \delta_k^{(s)} o_j^{(s-1)} \\ \qquad k = 1,..,N_s;\ j = 1,..,N_{s-1},\ s = M-1,..,1 \\ \Delta \theta_k^{(s)} = \eta \delta_k^{(s)} \end{cases} \quad (5)$$

where $f'(.)$ is the derivative of the activation function, d_k is the target output of the kth neuron of the output layer (Mth layer); $o^{(0)}$ is the vector of the inputs, η is the learning rate, $\Delta w_{kj}^{(s)}$ is the weight update relative to the BP move. We assume a scaled integer representation for all quantities involved in the computation.

2.2 Power-of-two representation

In the formulae of Section 2.1, if the weights are constrained to be power-of-two the multiplication in eqn. 3 between $w_{kj}^{(s+1)}$ and $\delta_j^{(s+1)}$ needs only shift-and-add operations. In eqn. 4, $f'(.)$ can be computed from a LUT, and thus $\sigma_k^{(s)}$ must be a power-of-two term to avoid hardware multiplication. In eqn. 5, with $o_j^{(s-1)}$ being a real number, both η and $\delta_k^{(s)}$ must be power-of-two terms to avoid multiplication. While η is an external term easily restricted to power-of-two, both $\sigma_k^{(s)}$ and $\delta_k^{(s)}$ are terms computed during the backward move, and must be rounded to the closest allowable power-of-two value before multiplication.

Let us denote $W_{m,n}$ to be the set of admissible single-term power-of-two values:

$$W_{m,n} \equiv \{x | x = R2^{-p};\ R \in \{-1,0,1\};\ m \le p \le n\} \quad (6)$$

where m and n are integers with $m \le n$. Extending admissible values to the sum of two power-of-two terms leads to the set:

$$W_{m,n}^2 \equiv \{x | x = R'2^{-p} + R''2^{-q};\ R', R'' \in \{-1,0,1\} \quad m \le p \le n,\ m \le q \le n\} \quad (7)$$

In the proposed algorithm, the choice whether to use $W_{m,n}$ or $W_{m,n}^2$ to approximate the involved items depends on the problem and must be made *a priori*. The chosen W set is then used for all values that must be discretised. In the following, we denote with $<y>$ the operation of rounding a number y, to the closest power-of-two or sum of power-of-two terms belonging to proper W set. We consider only rounding and not truncation or 'ceiling' operation.

During the learning, expressions eqns. 4 and 5 are repeatedly evaluated. In consequence, to avoid multiplications we need to round to power-of-two terms one operand of each expression at every iteration. It is worth noting that the multiplierless MLP presented in [3], which is able to work only in forward mode with fixed power-of-two weights, did not require such continuous rounding.

The proposed algorithm is viable only in the hypothesis that rounding a number to the closest power-of-two or sum of powers-of-two is an operation that can be efficiently implemented using hardware, and, together with a shift or a shift-and-add operation, is still more convenient with respect to multiplication. To this purpose, a recent paper on decomposition of binary integers into power-of-two terms has demonstrated that this operation can be accomplished very quickly and with a limited amount of circuitry [10]. The gain with respect to multiplication is more than one order of magnitude both in speed and in chip area, and so this operation is negligible with respect to both multiplication and addition. The proposed approach is then well suitable for a VLSI implementation of MLP with BP learning algorithm.

2.3 Proposed algorithm

The learning algorithm we propose closely resembles BP, but with rounding to avoid explicit multiplications. As already mentioned, the forward move does not need multiplications, as weight values are restricted to power-of-two. The offsets $\theta_k^{(s)}$ are not involved in multiplications and their values could still be real numbers, providing more 'degrees of freedom' to the network. The learning rate η must be restricted to a power-of-two term. The backward move is performed as follows:

Step 1: $\sigma_k^{(s)}$ are computed for each neuron of the Mth layer, and rounded to give

$$\sigma_k^{(M)} =< d_k - o_k^{(M)} > \quad (8)$$

Step 2: Eqn. 4 is applied to compute $\delta_k^{(s)}$; $f'(net_k^{(s)})$ is computed through a LUT and the multiplication is reduced to a shift or a shift-and-add operation:

$$\delta_k^{(s)} = \sigma_k^{(s)} f'(net_k^{(s)}) \quad (9)$$

Step 3: $\sigma_k^{(s)}$ are computed for each neuron of the previous layer. As weights values are power-of-two, no multiplication is involved:

$$\sigma_k^{(s)} =< \sum_{j=1}^{N_{s+1}} w_{jk}^{(s+1)} \delta_j^{(s+1)} > \quad (10)$$

Step 4: The weight and offset increments Δw and $\Delta\theta$ are computed. Rounding of $\delta_k^{(s)}$ is necessary to avoid multiplications in the first expression:

$$\begin{cases} \Delta w_{kj}^{(s)} = \eta < \delta_k^{(s)} > o_j^{(s-1)} & \text{if } < \delta_k^{(s)} > \neq 0 \\ \Delta w_{kj}^{(s)} = \eta 2^{-n} \text{sign} < \delta_k^{(s)} > o_j^{(s-1)} & \text{if } < \delta_k^{(s)} >= 0 \\ \Delta \theta_k^{(s)} = \eta \delta_k^{(s)} \end{cases} \quad (11)$$

Step 5: Weights and offsets are updated, preserving the power-of-two representation of weights:

$$\begin{cases} w_{kj}^{(s)} :=< w_{kj}^{(s)} + \Delta w_{kj}^{(s)} > \\ \theta_k^{(s)} := \theta_k^{(s)} + \Delta \theta_k^{(s)} \end{cases} \quad (12)$$

Step 6: The procedure is iterated from Step 2 and up to the first layer.

It is worth noting that in Step 4 the weight increment is forced to the minimum allowable power-of-two step if $\delta_k^{(s)}$ is rounded to zero. In this way, the algorithm is given a chance to proceed toward the minimum of MSE, even if the magnitudes of the quantities are reduced during learning.

The learning procedure can also be cumulative, i.e. the weight increments can be cumulated through the

whole learning set, and the new weights obtained rounding the old weights plus the cumulative increment, except for offsets which are not rounded. However, in the tests made we found a better behaviour of the algorithm with classical noncumulative weight updating and restricting the offsets to be power-of-two values. In the case of binary inputs (restricted to the values 0 and 1, or -1 and 1), or generally in the case of inputs which can be restricted to power-of-two, $\delta_k^{(s)}$ does not need to be rounded in Step 4 for the weights of the first layer, as $o_j^{(0)}$ is itself a power-of-two. This exception can be particularly useful in the case of MLP with many discrete inputs and fewer hidden and output neurons.

The proposed learning algorithm resembles BP, but the performed rounding implies that the weight increments do not exactly follow the direction of minus the gradient of the cost function, as in BP. The high number of rounding and their randomness, however, should guarantee an average convergence, which could constitute even an advantage to escape from local minima. Clearly, such algorithms cannot guarantee the convergence of learning, because traditional BP cannot. In the case of convergence problems, selective learning can be used, presenting more often those members of the learning set which exhibit the largest errors. If the arithmetic is performed with single power-of-two terms, the hardware speed and simplicity is highest, but the convergence of learning is more problematic. With two powers-of-two, convergence is much more likely for most problems, but the advantages of the approach are reduced.

3 Results and conclusion

The proposed learning algorithm has been tested on the classical pattern recognition problem comprising the recognition of noisy characters. Such a problem is not easy and has been already used for testing network performance [3, 11].

Table 1: Performance of 5 networks trained with the standard BP algorithm (Case: Real) and with the proposed algorithm (Case: PW2) to recognise the 10 character set (see text)

Test: A10; Case: PW2, $m = -1$, $n = 14$			
	Number of wrong patterns	Hit rate	MSE
1	32	99.68	0.1542
2	52	99.48	0.1397
3	29	99.71	0.1541
4	42	99.58	0.1419
5	39	99.61	0.1324
	mean = 38.6	mean = 99.61	mean = 0.1446
	std. dev. = 7.76	std. dev. = 0.0809	std. dev. = 0.00851
Test: A10; Case: Real			
1	53	99.47	0.0395
2	32	99.68	0.0370
3	38	99.62	0.0481
4	45	99.55	0.0520
5	66	99.34	0.0571
	mean = 46.8	mean = 99.53	mean = 0.04674
	std. dev. = 11.89	std. dev. = 0.1189	std. dev. = 0.00754

The performance has been evaluated in forward mode using 10 × 1000 noisy characters with 5% salt and pepper noise. The networks have 49 inputs, 10 hidden neurons and 4 outputs

In the first test, the 10 digit character set, encoded in a 7 × 7 pixel matrix, has been used as inputs of the networks and the target patterns were a 4-bit code sequence. The input patterns have been corrupted by adding salt and pepper noise consisting in 5% flipped pixels. MLP networks with 49 inputs, 10 hidden neurons and 4 outputs have been used to recognise the characters. The first group of trials has been performed using 5 networks with different initial weights and without any constraints, trained by the standard backpropagation procedure (Case: Real). The second group of trials has been performed using the same networks, whose weights were constrained to be sums of two powers-of-two belonging to $W_{m,n}^2$ with $m = -1$ and $n = 14$ (see eqn. 7), trained by the proposed learning algorithm (Case: PW2). All networks have been trained for 10000 iterations with constant learning rate equal to 0.5. Typical results obtained in this experiment using 10 × 1000 noisy characters, different from those used during training and with salt and pepper noise comprising 5% flipped pixels, are shown in Table 1. The performance of the proposed algorithm is roughly equivalent to the standard BP unconstrained case in terms of error rate (wrong patterns), although the mean square error (MSE) is consistently higher.

Table 2: Performance of 5 networks trained with the standard BP algorithm (Case: Real) and with the proposed algorithm (Case: PW2) to recognise the 64 character set (see text)

Test: A64 Case: PW2, $m = -1$, $n = 14$			
	Number of wrong patterns	Hit rate	MSE
1	170	99.73	0.2781
2	174	99.72	0.2847
3	182	99.71	0.2682
4	153	99.76	0.2543
5	225	99.65	0.2648
	mean = 180.8	mean = 99.71	mean = 0.270
	std. dev. = 24.04	std. dev. = 0.011	std. dev. = 0.010
Test: A10; Case: Real			
1	144	99.78	0.1146
2	156	99.76	0.1200
3	146	99.77	0.1182
4	155	99.76	0.1258
5	110	99.83	0.1398
	mean = 142.2	mean = 99.78	mean = 0.1236
	std. dev. = 16.78	std. dev. = 0.026	std. dev. = 0.008

The performance has been evaluated in forward mode using 64 × 1000 noisy characters with 0.5% salt and pepper noise. The networks have 56 inputs, 64 hidden neurons and 7 outputs

Other experiments performed on larger character sets, using the 7 × 8 IBM PC CGA standard characters, confirm the capability of the proposed learning algorithm. As in the previous experiment, we used 5 different networks for each trial. MLP networks with 56 inputs, 64 hidden neurons and 7 outputs have been used to recognise a set of 64 characters corresponding to the ASCII codes from 32 to 96. The training phase consisted of 10000 iterations with learning rate equal to 0.5, plus 3000 iterations with a reduced learning rate equal to 0.18750. The input patterns has been corrupted by adding salt and pepper noise consisting of 0.5% flipped pixels. Table 2 summarises the perfor-

mance of the MLP networks after the training with the standard BP (Case: Real) and with the power of two learning algorithm (Case: PW2), obtained using 64 × 1000 noisy characters different from those used during training, again with salt and pepper noise consisting of 0.5% flipped pixels.

Following the last experiment, the networks trained on the 64 character set were tested in forward mode with 64 × 1000 character patterns corrupted by different salt and pepper noise levels. Fig. 1 shows the number of wrong classifications obtained when the noise level has been progressively increased from 0.5% to 2%.

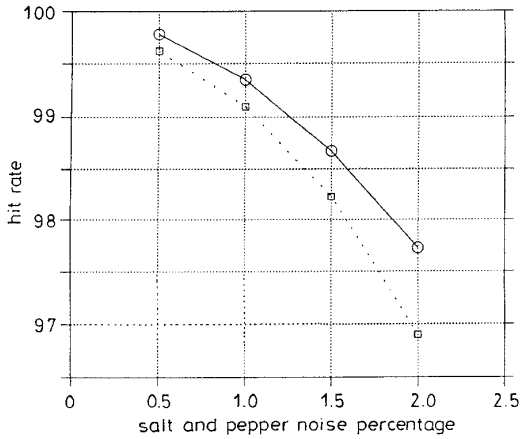

Fig. 1 *Hit rate obtained by the networks of Table 2 when the salt and pepper noise level has been progressively increased from 0.5% to 2%*
The test set consisted in 64 × 1000 noisy character patterns
○——○ real networks
□- - -□ power of two networks

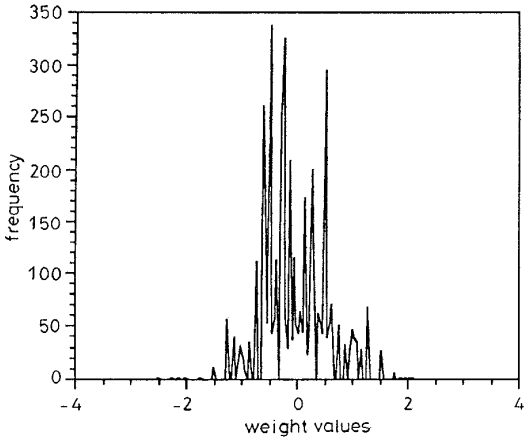

Fig. 2 *Distribution of the discretised weights, for a typical network used in Table 2, belonging to the $W^2_{-1,14}$ set*

In conclusion, all the performed experiments indicate that performance of the proposed power-of-two learning algorithm is good. In particular, the discretised networks show performance close to that obtained by the real-valued networks in terms of hit rate, i.e. the number of wrong recognitions with respect to the total number of patterns, while they exhibit worse performance in terms of mean square error. This result suggests that the proposed learning algorithm could be well suited for most pattern recognition and classification problems where the inputs and outputs are binary or discretised in some way.

Sometimes, in the experiments, the performance of the power-of-two network is better than that obtained by the corresponding standard network; this fact can be explained by the high number of rounding operations to the nearest power-of-two and their randomness, and that can be an advantage to escape from local minima. Also the weights rounding operation can be considered as an additive noise on the input data; therefore it could improve the generalisation performance of the feedforward layered perceptrons [12].

The obtained performance is confirmed by the distribution of the discretised weights, shown in Fig. 2 for a typical case, and reveals how the proposed learning algorithm is able to make a correct use of the available discrete values.

From an hardware point of view, the presented experiments show that it can be obtained by networks trained with the proposed algorithm which require only 16-bit shifters and adders, an error rate performance slightly worse than with standard networks, which, however, require 16-bit multipliers and adders, much more demanding in terms of chip area and execution time. More advantage could be obtained using only one power-of-two, trading performance with hardware complexity.

4 References

1 WHITE, B.A., and EL-MASRY, M.I.: 'The digi-neocognitron: a digital neocognitron neural network model for VLSI', *IEEE Trans.*, 1992, **NN–3**, (1), pp. 73–85
2 BENVENUTO, N., MARCHESI, M., ORLANDI, G., PIAZZA, F., and UNCINI, A.: 'Design of multi-layer neural networks with powers-of-two weights'. Proc. ISCAS-90 IEEE Intl. Symp. *Circuits & Systems*, New Orleans, LA, USA, May 1990, pp. 2951–2954
3 MARCHESI, M., ORLANDI, G., PIAZZA, F., and UNCINI, A.: 'Fast neural networks without multipliers', *IEEE Trans.*, 1993, **NN–4**, (1), pp. 53–62
4 TANG, C.Z., and KWAN, H.K.: 'Feedforward neural networks with powers-of-two weights'. Proc. Intl. Joint Conf. *Neural Networks* — Vol. I, Beijing, China, November 1992, pp. 93–98
5 LIM, Y.C., and LIU, B.: 'Design of cascade form FIR filters with discrete valued coefficients', *IEEE Trans.*, 1988, **ASSP–36**, pp. 1735–1739
6 RUMELHART, D.E., HINTON, G.E., and WILLIAMS, R.J.: 'Learning internal representations by error propagation' *in* RUMELHART, D.E., and McCLELLAND, J.L. (Eds.): 'Parallel distributed processing, Vol. I: Foundations' (MIT Press, Cambridge, MA, 1986)
7 XUE, P., and LIU, B.: ' Adaptive equalizer using finite-bit power-of-two quantizer', *IEEE Trans.*, 1986, **ASSP–34**, pp. 1603–1611
8 MARCHESI, M.L., ORLANDI, G., PIAZZA, F., and UNCINI, A.: 'Linear data-driven architectures implementing neural network models', *Int. J. Neural Networks*, 1992, **3**, (3)
9 KUNG, S.Y., and HWANG, J.N.: 'Parallel architectures for artificial neural nets'. Proc. IEEE Intl. Conf. *Neural Networks*, San Diego, California, 1988, Vol. II, pp. 165–172
10 LIM, Y.C., LIU, B., and EVANS, J.B.: 'VLSI circuits for decomposing binary integers into power-of-two terms'. Proc. of IEEE Intl. Symp. *Circuits & Syst.*, New Orleans, May 1990, pp. 2304–2307
11 TANG, C.Z., and KWAN, H.K.: 'Multilayer feedforward neural networks with single power-of-two weights', *IEEE Trans.*, 1993, **SP–41**, (8), pp. 2724–2727
12 REED, R., MARKS, R.J., and OH, S.: 'Similarities of error regularization, Sigmoide gain scaling, target smoothing, and training with jitter', *IEEE Trans.*, 1995, **NN–6**, (3), pp. 529–538