general, for every other Toeplitz network, the diameter is the ceiling of $w/2+1$.

To show that there exists at least such a path between any two points i and j, consider the set of points $i-w/2$ to $i+w/2$. One of these points has to be adjacent to j. This is so because the width of the Toeplitz network is w (meaning that there are exactly $w-1$ zeros between any two ones in the adjacency matrix). Hence, the next theorem follows:

Theorem 4: The diameter of the Toeplitz network is the ceiling of $w/2+1$.

Because of the symmetric nature of the construction of the Toeplitz network, the network is reliable in case of point and edge failures. Fault diameter is yet another useful indicator of the reliability of a network. Fault diameter is the length of the longest path when $k-1$ points fail, k being the point connectivity of a network.

Theorem 5: The fault diameter of the network is w.

Proof: From Lemma 3, we see that each of the $s+1$ paths is at most of length w. Hence, if any of the s points fail, the length of the longest path is w. The length of the longest path is a ceiling of $w/2+1$, even when $s/2$ points fail. This implies that we could still apply the lemma and there will be 2 points adjacent to every point.

As stated in the earlier section, graph theorists are often interested in obtaining Moore graphs. The Toeplitz network has less number of points than the Moore graphs for the same regular degree and diameter. However, the Toeplitz network satisfies some additional useful properties, such as bipartiteness. As against group graphs, the diameter and the fault diameter calculations are explicit, and hence, the routing is simpler.

IV. Conclusion

In this paper, a new class of interconnection networks are presented and the spectral properties of these networks are studied. It is shown that these networks possess many desirable properties, such as small diameter, high connectivity, and high reliability. It would be interesting to derive algorithms which exploit the Toeplitz network interconnection scheme.

Acknowledgment

The authors wish to thank both the referees for their suggestions and Joyce Brock for editing.

References

[1] S. B. Akers and B. Krishnamurthy, "Group graphs as interconnection networks," in *Proc. 14th Fault Tolerance Computing Symp.*, pp. 422–427, 1984.
[2] B. W. Arden and H. Lee, "A multi-tree structure network," in *Proc. Compcon. '78 Fall*, pp. 201–210, Sept. 1978.
[3] J.-C. Bermond, N. Homobono, and C. Peyrat, "Large fault-tolerant interconnection networks," in *Proc. Japan Conf. Graph Theory and its Applications*, Hakone, June 1986.
[4] F. Boesch and R. Tindell, "Circulants and their connectivities," *J. Graph Theory*, vol. 8, pp. 487–499, 1984.
[5] F. Boesch and J. F. Wang, "Reliable circulant networks with minimum transmission delay," *IEEE Trans. Circuits Syst.*, vol. 32, pp. 1286–1291, 1985.
[6] ____, "Super line connectivity properties of circulant graphs," *Siam J. Algebraic Discrete Methods*, vol. 7, pp. 89–98, 1986.
[7] S. H. Bokhari and A. D. Raza, "Augmenting computer networks," 84-33, ICASE Rep. No. NASA Langley Research Center, Hampton, VA, 1984.
[8] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications.* New York: American Elsevier, 1976.
[9] F. R. K. Chung, "Diameters of graphs: Old problems and new results," in *Proc. 18th Southeastern Conf. Combinatorics, Graph Theory and Computing*, Boca Raton, LA, 1987.
[10] N. Deo and M. J. Quinn, "Pascal graphs and their properties," *The Fibonacci Quarterly*, vol. 21, pp. 203–214, Aug. 1983.
[11] F. Harary, *Graph Theory.* Reading, MA: Addison-Wesley, 1969, pp. 94–96.
[12] D. K. Pradhan and S. M. Reddy, "A fault-tolerant communication architecture for distributed systems," *IEEE Trans. Comput.*, vol. C-31, pp. 863–870, Sept. 1982.
[13] F. P. Preparata and J. K. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. Ass. Comput. Math.* pp. 300–309, 1981.
[14] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed, parallel machine: I," in *Proc. Fourth Symp. Computer Architecture*, pp. 105–117, 1977.
[15] L. Uhr, *Algorithm-Structured Computer Arrays and Networks.* Orlando, FL: Academic, 1984, pp. 113–127.
[16] R. S. Wilkov, "Analysis and design on reliable computer networks," *IEEE Trans. Commun.*, vol. COM-20, pp. 660–678, June 1972.

An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks

STEFANOS KOLLIAS AND DIMITRIS ANASTASSIOU

Abstract—**A novel learning algorithm is developed for the training of multilayer feedforward neural networks, based on a modification of the Marquardt–Levenberg least squares optimization method. The algorithm updates the input weights of each neuron in the network in an effective parallel way. An adaptive distributed selection of the convergence rate parameter is presented, using suitable strategies of optimization techniques. The algorithm has better convergence properties than the conventional backpropagation learning technique. Its performance is illustrated, using examples from digital image halftoning and logical operations such as the XOR function.**

I. Introduction

An important class of neural networks is feedforward multilayer perceptrons. These networks consist of layers of neurons, which are connected in a feedforward way. In the simplest case each neuron produces its output by computing the inner product of its input signal and weight vectors and by passing the result through a nonlinear function, as is shown in Fig. 1. One commonly used nonlinear monotonic function is the sigmoidal one, which can be defined as follows:

$$g(x)=\frac{1}{1+e^{-x}}. \tag{1.1}$$

A three-layer network is shown in Fig. 2. The intermediate layers between the output layer and the network's input are usually called hidden layers. A crucial property of these networks is their ability to improve their performance by learning new information. This is accomplished by modifying the interconnection strengths among neurons, i.e., the weights of the network, according to some prespecified rules. The "knowledge" of the network lies therefore in its inter-neuron connections and their

Manuscript received February 10, 1988; revised August 29, 1988 and March 2, 1989. This paper was recommended by Associate Editor H. Gharavi.

S. Kollias was with Department of Electrical Engineering and the Center for Telecommunications Research, Columbia University, New York, NY 10027-6699, on leave from the Electrical Engineering Department, National Technical University of Athens, Greece.
D. Anastassiou is with the Department of Electrical Engineering and the Center for Telecommunications Research, Columbia University, New York, NY 10027-6699.

IEEE Log Number 8928270.

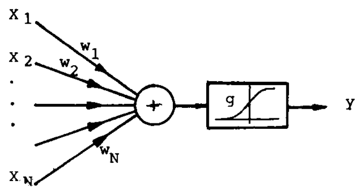

Fig. 1. The diagram of a single neuron.

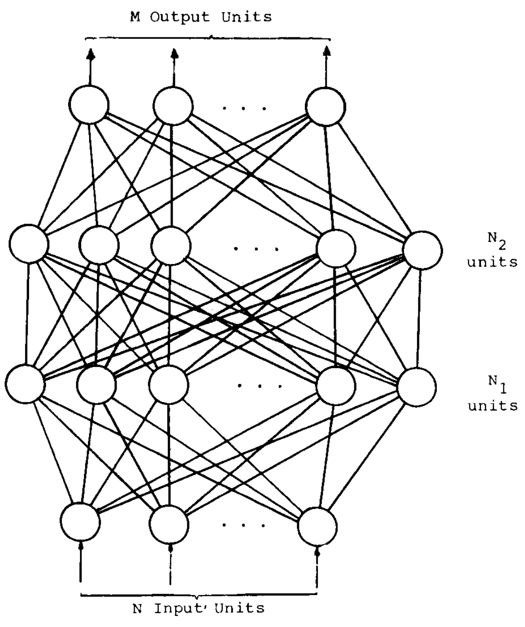

Fig. 2. A three-layer neural network. Circles are used to represent the neurons of the network.

weights, in contrast to rule-based artificial intelligence applications. Furthermore, the network architecture permits high computational rates to be obtained through the massively parallel distributed processing which is performed. It is also due to this kind of processing that such networks have a great degree of robustness or fault tolerance to local damages. In this paper we are concerned with the problem of supervised learning of multilayer networks. Various learning algorithms have been recently introduced, the most famous being the backpropagation algorithm [1]-[3]. Many successful applications of this method have been developed in areas such as speech and image recognition and modeling of human learning [4], [5].

This paper presents a neural network learning algorithm, which is based on an efficient modification of the Marquardt–Levenberg least squares optimization method [6]-[8] and has better convergence properties than the conventional backpropagation method. The paper is organized as follows. Section II formulates the nonlinear least squares minimization problem and discusses various unconstrained optimization techniques for its solution. The proposed algorithm is developed in Section III, while an adaptive selection of the convergence rate parameter is presented in Section IV. Section V presents various applications of the proposed learning technique in digital image halftoning and in performing logical operations such as the XOR function. Some conclusions and suggestions for further research are given in Section VI.

II. The Nonlinear Least Squares Minimization Problem

The error criterion which is generally used in the training of neural networks is the minimization of a sum of squares error function. For a network with M outputs and after the presentation of K pairs of input and desired output patterns, this function is

$$E_K = \frac{1}{2}\sum_{r=1}^{K}\sum_{m=1}^{M}\left[d(m)-y(m)\right]_r^2 \tag{2.1}$$

where the symbols d and y denote the desired and actual outputs of the network, and the subscript r shows their dependence on the rth input presentation.

When an input pattern is presented, it propagates forward through the network and each neuron computes its output value using the prespecified set of its input weights. Let us consider the case of a network consisting of L layers, the lth of which contains N_l neurons. Then each neuron computes its output according to the following equations for $l=1,\cdots,L$ and $i=1,\cdots,N_l$:

$$x_i^l = g\left\{\sum_{n=1}^{N_{l-1}}\left(w_{ni}^l x_n^{l-1}\right)+\theta_i^l\right\} \tag{2.2}$$

where x_i^l denotes the output of the ith neuron belonging to the lth layer and w_{ni}^l is the weight of the connection between the nth neuron of the $(l-1)$ layer and the ith neuron of the lth layer. The Lth layer is the output layer, while layer zero is the input layer. Thus N_L and N_0 are equal to M and N, respectively, while x_i^L and x_i^0 denote the output $y(i)$ and input $x(i)$ sequences. The function g is usually the sigmoid defined in (1.1) and θ_i^l is the threshold of the ith neuron of the lth layer, which controls the neuron's output when there are no signals to its input. In the following, we treat the thresholds as weights that connect an input, which is always *on*, i.e., its value is always one, to the neuron.

Let us define the vector w_i^l of all input weights to the ith neuron of layer l, including the threshold θ_i^l

$$w_i^l = \left[w_{1i}^l \cdots w_{N_{l-1}i}^l \quad \theta_i^l\right]^T \tag{2.3}$$

and let the vector x^l contain the outputs of all neurons in layer l and one more element equal to one

$$x^l = \left[x_1^l \cdots x_{N_l}^l \quad 1\right]^T. \tag{2.4}$$

Then (2.2) can be written using the above symbol notations as follows:

$$x_i^l = g\left\{\left(w_i^l\right)^T\left(x^{l-1}\right)\right\}. \tag{2.5}$$

Let us also define a vector w, consisting of all the weights w_i^l in the network. A necessary condition at a minimum, global or local, of (2.1) is that the derivatives of this function with respect to w be zero. Forming these derivatives we get the following system of nonlinear equations for $l=1,\cdots,L$ and $i=1,\cdots,N_l$:

$$f_i^l = \sum_{r=1}^{K}\sum_{m=1}^{M}\left[d(m)-y(m)\right]_r^T\left[-\frac{\partial y(m)}{\partial w_i^l}\right]_r = 0 \tag{2.6}$$

where the dimension of vector f_i^l is $(N_{l-1}+1)$. Let us also form a global vector F, consisting of all the vectors f_i^l.

A simple technique for the solution of such a system is steepest descent [9], [10]. This method, which is the basis for the derivation of the backpropagation algorithm or the momentum version of it, is suitable for parallel updating of all the weights of the network, or at least of those in the same layer, during each iteration of it. Thus its computational complexity is $O(N_{l-1})$ operations per iteration and neuron, or $O(n_p)$ operations per iteration in total, where,

$$n_p = \sum_{l=1}^{L} N_l(N_{l-1}+1). \tag{2.7}$$

However, this method has many drawbacks. It converges linearly and may be inefficient, particularly as the minimum is approached. Another well known iterative technique for the solution of the system of nonlinear equations (2.6) based on a linearization procedure, is Newton's method [9], [10]. This method is attractive, because it converges quadratically, provided that a good, sufficiently close to a minimum of the error function, initial estimate of the unknown weight vector is available. Computation of the increments $\Delta \boldsymbol{w}$, which bring the initial estimate $\boldsymbol{w}_0$ closer to a minimum, is performed by solving the following linear system:

$$H\Delta \boldsymbol{w} = -\boldsymbol{F} \tag{2.8}$$

where the Hessian matrix H contains the second derivatives of the error function with respect to the network weights. The application of standard quasi-Newton algorithms [9]–[11] has been examined as a solution to the learning problem [12], but no efficient parallel implementation in the form of an artificial neural network was found. Application of such methods in the solution of (2.8) requires $O[(n_p)^2]$ operations per iteration. An approach has been recently developed [13], which implements Newton's method as a second-order backpropagation algorithm with complexity $O(n_p)$. This approach uses, however, an approximation of matrix H, which, as is also mentioned in [13], is inefficient for real life applications and can be useful only in cases of uncorrelated input signals. Furthermore, it introduces more tuning parameters in its attempt to approximate the second derivatives in (2.8), making therefore its application more difficult.

In this paper we follow a different approach. This approach permits the derivation of an efficient neural network learning algorithm, based on the Marquardt–Levenberg least squares method. The algorithm possesses quadratic convergence close to a minimum, where it approximates the Newton method. Moreover it can converge even if the initial estimates are relatively poor, as does the method of steepest descent.

III. Analysis of the Least Squares Learning Algorithm

The Marquardt–Levenberg technique approximates matrix H by a simpler one

$$H = \boldsymbol{F}\boldsymbol{F}^T + \lambda\Omega \tag{3.1}$$

where λ is a positive factor and Ω is an appropriately chosen matrix. A version of the Marquardt–Levenberg method, which has successfully been used in many applications [8], [14], was to let Ω be diagonal, with elements equal to the diagonal terms of $\boldsymbol{F}\boldsymbol{F}^T$. The main idea followed in this paper in order to derive an effective algorithm, which can efficiently be applied to distributed neural network training, is to let matrix Ω have the following near-diagonal form:

$$\Omega = \begin{bmatrix} f_1^1(f_1^1)^T & \cdots & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & f_{N_1}^1(f_{N_1}^1)^T & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \cdots & f_1^L(f_1^L)^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} & \cdots & f_{N_L}^L(f_{N_L}^L)^T \end{bmatrix}. \tag{3.2}$$

This form contains the diagonal elements of $\boldsymbol{F}\boldsymbol{F}^T$, but it also contains the rest of the $f_i^l(f_i^l)^T$ elements around its diagonal ones, keeping thus second order information for the input weights $\boldsymbol{w}_i^l$ of each neuron in the network.

The system of equations defined by (2.8) and (3.1)–(3.2) can easily be proved to result from the minimization of the following error criterion:

$$E_K'(\boldsymbol{w}) = \mu E_K(\boldsymbol{w}) + \Delta \boldsymbol{w}^T \Omega \Delta \boldsymbol{w} \tag{3.3}$$

where $\mu = 1/\lambda$ and E_K was defined in (2.1). Let the increment $\Delta \boldsymbol{w}_1 = \boldsymbol{w}_1 - \boldsymbol{w}_0$ be provided by the solution of (2.8) using the definitions (3.1)–(3.2), i.e., by minimizing the error criterion E_K' in (3.3). Then it can easily be shown, using (3.3), that

$$\mu E_K(\boldsymbol{w}_1) < \mu E_K(\boldsymbol{w}_1) + \Delta \boldsymbol{w}_1^T \Omega \Delta \boldsymbol{w}_1$$
$$= E_K'(\boldsymbol{w}_1) < E_K'(\boldsymbol{w}_0) = \mu E_K(\boldsymbol{w}_0) \tag{3.4}$$

the last equality being true, because $\Delta \boldsymbol{w}_0$ equals zero. In the first inequality in (3.3), it was assumed that the symmetric and semi-positive definite matrix Ω is positive definite. This condition is always fulfilled, if a very small positive number, such as 0.01 [15], is added to the diagonal elements of Ω in (3.2). Thus (3.4) shows that minimization of the error criterion E_K' also diminishes the sum of squares error function E_K. Let now the increment $\Delta \boldsymbol{w}_2 = \boldsymbol{w}_2 - \boldsymbol{w}_0$ be provided by solving the general form of (2.8), i.e.,by minimizing the original error function E_K in (2.1). It can be shown, by using (3.3) and the increments $\Delta \boldsymbol{w}_1$ and $\Delta \boldsymbol{w}_2$ as in [6], that it is

$$\Delta \boldsymbol{w}_1^T \Omega \Delta \boldsymbol{w}_1 < \Delta \boldsymbol{w}_2^T \Omega \Delta \boldsymbol{w}_2. \tag{3.5}$$

Equation (3.5) proves that minimization of the error criterion in

(3.3) also reduces the weighted sum of squares of the increments Δw, improving therefore the linearization procedure used to derive (2.8). By letting $\lambda \gg 1$, matrix H in (3.1) is written in terms of (3.2)

$$H = \lambda \Omega \tag{3.6}$$

and due to the special form of matrix Ω, (2.8)–(3.1) take the following form in the case of the ith neuron of the lth layer, for example,

$$H_i^l \Delta w_i^l = -\mu f_i^l \tag{3.7}$$

where

$$H_i^l = \sum_{r=1}^{K} \sum_{m=1}^{M} \left[\frac{\partial y(m)}{\partial w_i^l}\right]_r \left[\frac{\partial y(m)}{\partial w_i^l}\right]_r^T. \tag{3.8}$$

As a result, the algorithm avoids the computation and storage of second-order information about the cross-products between input weights of different neurons and each weight vector Δw_i^l can be computed independently of all others during each iteration in a distributed parallel form. The simple Gauss–Newton or weighted Gauss–Newton least squares algorithm [8], [9] can be used to solve each (3.7).

In the neural network formulation the system of equations (3.7)–(3.8) implies that each neuron in the network tries to minimize the error function with respect to its own weight vector w_i^l and that all the neurons of the network, or at least these of the same layer, may update their input weights during each iteration independently of all others. The convergence factor μ prevents successive estimates from oscillating. The algorithm has the distributed form of the backpropagation technique. It performs two sweeps through all the weights of the network at each iteration of it. On the first sweep the input signals are propagated forward, so that each hidden and output neuron computes its own output. On the second sweep the error signals are computed at the top of the network as the difference between the desired and actual outputs. This information is then propagated down to the bottom layer and each neuron updates its input weights using (3.7)–(3.8). In the case ofthe weight vector w_i^l in a multilayer network, the proposed algorithm can be written in sequential form, for $r = 1, \cdots, K$:

$$w_i^l(r) = w_i^l(r-1) + P_i^l(r) e_i^l(r) (x^{l-1})_r \tag{3.9}$$

$$P_i^l(r) = P_i^l(r-1) - (1/d_i^l) P_i^l(r-1) x^{l-1} (x^{l-1})^T P_i^l(r-1) \tag{3.10}$$

where the matrix $P_i^l(K)$ is the inverse of H_i^l defined in (3.8) and the scalars d and e are computed, as follows, for $l = 1, \cdots, L$ and $i = 1, \cdots, N_L$:

$$d_i^l = (b_i^l)^{-1} + (x^{l-1})^T P_i^l(r-1) x^{l-1} \tag{3.11}$$

$$e_i^l(r) = \mu \epsilon_i^l - b_i^l (x^{l-1})^T \left[w_i^l(r-1) - w_i^l(0) \right]. \tag{3.12}$$

In the above equations, the indication of dependence of some quantities on r was ignored for notational simplicity. During each iteration of the algorithm in (3.9)–(3.12), the input data are processed sequentially and the outputs of all neurons are computed using the weight vector $w_i^l(0)$. In the end the current weight vector estimate $w_i^l(K)$ replaces $w_i^l(0)$, so that the next iteration can begin. The procedure stops when the changes in all weights are less than a very small threshold. At the beginning of each iteration matrix $P_i^l(0)$ is chosen diagonal with relatively large values. The algorithm can also be applied in recursive form. In this case the outputs of all neurons are computed, in each recursion of the algorithm, using the weight vector $w_i^l(r-1)$ instead of $w_i^l(0)$. The addition of an over-relaxation term, exactly as in the momentum technique, is sometimes necessary to ensure convergence in this case [16]. The updating of matrix $P_i^l(r)$ in (3.10) requires $O[(N_{l-1}+1)^2]$ operations per iteration [17], thus the computational complexity of the above described algorithm is $O[(N_{l-1})^2]$ per neuron and iteration, or $O(n_p')$ in total, where,

$$n_p' = \sum_{l=1}^{L} N_l (N_{l-1}+1)^2. \tag{3.13}$$

Even in a massively parallel environment, the increased, compared to standard backpropagation, per neuron complexity $O[(N_{l-1})^2]$ constitutes a problem. However, this problem is less serious, for implementation purposes, than the $O[(n_p)^2]$ computational and storage requirement for the solution of the general form of (2.8), using for example the technique presented in [12]. Moreover, the complexity of the algorithm can be reduced to $O(n_p)$, if fast algorithms of the least squares method [17], [18] are used. These efficient schemes do not calculate matrix P_i^l, but directly update vector $P_i^l x^{l-1}$, which is required in (3.9), having a computational complexity similar to the backpropagation one. Use of such algorithms is, however, feasible only when the training is performed using consecutive samples of the input data, i.e., when a sliding window passes over the data and constitutes the input to the network in each iteration. This happens in most signal and image processing applications, as is discussed later in the paper.

Since the updating of the input weight vectors of all neurons takes place in the second sweep of the algorithm, appropriate formulas which update b_i^l and ϵ_i^l recursively from the top to the bottom of the network are derived next. The formulas for the ϵ_i^l are the same as the ones derived for the backpropagation algorithm [2]

$$\epsilon_i^{l-1} = c_i^{l-1} \sum_{m=1}^{N_l} \epsilon_m^l w_{im}^l \tag{3.14}$$

for $l = L-1, L-2, \cdots, 2$ with

$$\epsilon_i^{L-1} = c_i^{L-1} \sum_{m=1}^{N_L} c_m^L w_{im}^L. \tag{3.15}$$

Equations (3.14)–(3.15) provide a computationally efficient way of computing the backward errors, because they require only $O(n_p)$ operations per iteration. The proposed algorithm requires the computation of the nonnegative quantities b_i^l in (3.11)–(3.12) as well. A similar, but more elaborate, recursive way of computing these quantities is given next and proved in Appendix A, by defining the following sequence:

$$\beta_{ij}^{l-1} = c_i^{l-1} c_j^{l-1} \sum_{m=1}^{N_l} w_{im}^l \sum_{n=1}^{N_l} w_{jn}^l \beta_{mn}^l \tag{3.16}$$

for $l = L-1, L-2, \cdots, 2$ and $1 \leqslant i, j \leqslant N_{l-1}$, with

$$\beta_{ij}^{L-1} = c_i^{L-1} c_j^{L-1} \sum_{m=1}^{N_L} w_{im}^L w_{jn}^L (c_m^L)^2 \tag{3.17}$$

and by letting

$$b_i^{l-1} = \beta_{ii}^{l-1}. \tag{3.18}$$

In the above equations it is

$$c_i^l = x_i^l [1 - x_i^l] \tag{3.19}$$

while the ϵ_i^L and b_i^L quantities at the output layer are

$$\epsilon_i^L = \sum_{m=1}^{M} c_m^L[d(m)-y(m)] \tag{3.20a}$$

$$b_i^L = \sum_{m=1}^{M} (c_m^L)^2. \tag{3.20b}$$

It can be shown that the updating of b_i^l using (3.16)–(3.20) requires $O(n_p')$ operations per iteration in the case of networks with more than two layers. In this case we can use an approximate formula by ignoring the cross-neuron products in (3.16) and by updating b_i^l, for $i=1,\cdots,N_{l-1}$ and $l=1,\cdots,L$, as follows:

$$b_i^{l-1} = (c_i^{l-1})^2 \sum_{m=1}^{N_l} (w_{im}^l)^2 b_m^l. \tag{3.21}$$

As a result (3.21) has the same computational complexity as (3.14). The above equations together with (3.9)–(3.12) form the proposed learning algorithm.

IV. Adaptive Selection of the Convergence Rate Factor

The selection of the parameter μ in (3.9) is a crucial problem in the application of the proposed algorithm. This problem is examined next, by using suitable techniques, which have been developed for the general form of the Marquardt–Levenberg optimization method. Two techniques, which can be combined with the proposed algorithm, are the model trust region approach [19], [20] and the line search strategy [19], [21]. Both techniques try to determine a region, in which the nonlinear problem is adequately represented by the quadratic model (3.3), so that convergence of the algorithm to a local minimum of the error function be guaranteed. A criterion is used, in the second for example technique, for selecting an acceptable value of μ at the end of each iteration of the algorithm. In the case of the distributed system of (3.7) this criterion can be written as follows:

$$\psi = \frac{E_K(w)-E_K(w+\Delta w)}{\sum_{l=1}^{L}\sum_{i=1}^{N_l}\mu(f_i^l)^T(H_i^l)^{-1}f_i^l} > \sigma \tag{4.1}$$

where w contains all vectors $w_i^l(0)$ used in each iteration, while Δw contains all the increments $\Delta w_i^l(K)$ computed at the end of each iteration and σ is a small positive constant, such as 10^{-4}. A backtracking strategy is used, by selecting an initial value of μ and then reducing it, until (4.1) is satisfied. Appropriate rules for reducing μ are given in [19]. The computation of $E_K(w+\Delta w)$, required in each application of (4.1), does not seriously augment the complexity of the method, due to the massive parallelism of the network.

Since (3.7)–(3.8) are solved separately for each neuron of the network, it would be more effective to use different values of μ, i.e., μ_i^l, for updating the input weights of each neuron. This would, however, require the selection of each parameter μ_i^l in the network, which would be a quite difficult task. An adaptive distributed updating of these parameters is presented next, by approximately computing the contribution of each neuron to the minimization of the error function. Let us consider the error $E_K(w+\Delta w)$ as a function of the parameter μ. We may write, for small values of the difference between two parameter values μ and μ',

$$E_K(\mu') = E_K(\mu) + (\mu'-\mu)\left(\frac{\partial E_K}{\partial \mu}\right). \tag{4.2}$$

The derivative of the error E_K in (2.1) with respect to μ can be approximated by using the definition (2.6) and the block-iterative form of (3.9), resulting from repeated summation of (3.9) for $r=1,\cdots,K$ [8]:

$$\left(\frac{\partial E_K}{\partial \mu}\right) = \left[\sum_{l=1}^{L}\sum_{i=1}^{N_L}\left(\frac{\partial E_K}{\partial w_i^l}\right)^T\left(\frac{\partial w_i^l}{\partial \mu}\right)\right] = -2\left[\sum_{l=1}^{L}\sum_{i=1}^{N_L} R_i^l(K)\right] \tag{4.3}$$

where

$$R_i^l(K) = \left[\sum_{r=1}^{K}\epsilon_i^l(x^{l-1})_r^T\right]P_i^l(K)\left[\sum_{r=1}^{K}\epsilon_i^l(x^{l-1})_r\right]. \tag{4.4}$$

From (4.2) to (4.4) it is seen that the reduction of the error function in each iteration of the algorithm, due to a small change in the convergence parameter μ, is proportional to the sum of the positive scalars $R_i^l(K)$, which tend to zero as the minimum is approached [8].

It should, however, be stressed that different neurons will generally give different values of the $R_i^l(K)$ quantities and consequently different reductions of the error function. Using this fact we show next that we may let the convergence factor μ vary with iteration and neuron. If we call the resulting factors in each iteration μ_i^l and use them instead of μ' in (4.2), then it is easy to show, using (4.3), that the error reduction due to the differences $(\mu_i^l-\mu)$ will be equal to

$$2\left[\sum_{l=1}^{L}\sum_{i=1}^{N_L}(\mu_i^l-\mu)R_i^l(K)\right]. \tag{4.5}$$

It is desired that this reduction be nonnegative, therefore it should be

$$\sum_{l=1}^{L}\sum_{i=1}^{N_L}[\mu_i^l R_i^l(K)] \geqslant \mu\sum_{l=1}^{L}\sum_{i=1}^{N_L}R_i^l(K). \tag{4.6}$$

An obvious solution to (4.6) could be to let all μ_i^l's be greater than μ, but this would eventually lead to an unstable behavior of the algorithm. A choice of μ_i^l's which satisfies (4.6), keeping also the sum of all μ_i^l's constant and equal to $\mu[\sum_{l=1}^{L}N_l]$, is

$$\mu_i^l = \mu\frac{[\sum_{l=1}^{L}(N_l)]R_i^l(K)}{\sum_{l=1}^{L}\sum_{i=1}^{N_l}R_i^l(K)} \tag{4.7}$$

as can be verified by using the above values of μ_i^l in (4.6) and the following inequality:

$$n\sum_{i=1}^{n}a_i^2 - \left(\sum_{i=1}^{n}a_i\right)^2 = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_i-a_j)^2 \geqslant 0 \tag{4.8}$$

where the a's represent the R_i^l's and the summation over i in (4.8) corresponds to the summation over i and l in (4.7). Following the above analysis, each iteration of the algorithm is performed using (3.9)–(3.12) and the same convergence factor μ for all neurons in the network. The $R_i^l(K)$ quantities are also computed, by using a sequential form of (4.4). Then the weights $w_i^l(K,\mu_i^l)$ are computed in the end of each iteration in terms of the already determined $w_i^l(K,\mu)$

$$w_i^l(K,\mu_i^l) - w_i^l(0) = (\mu_i^l/\mu)[w_i^l(K,\mu)-w_i^l(0)]. \tag{4.9}$$

Derivation of (4.9) is based on the fact that all quantities, apart from the parameters μ, which are used to calculate both $w_i^l(K,\mu_i^l)$

and $w_i^l(K,\mu)$, are computed in terms of $w_i^l(0)$. Thus they are the same and, apart from the μ's, they cancel each other.

Let us now replace the parameter μ in (4.1) by its distributed values μ_i^l from (4.7). The selection of the μ_i^l parameters is then equivalent to the selection of the single parameter μ in (4.7). The criterion (4.1) can therefore be applied to determine an acceptable value of μ, using the already computed values of the quantities $R_i^l(K)$. It should be added that the exact number of the required hidden units in most neural network applications is unknown and a larger number of neurons is generally used. In these cases (4.7) and (4.9), which update the weights of each neuron by examining its contribution to the minimization of the error function, can be very useful. For ease of implementation in cases of massive layered networks, it would be more preferable to compute the μ_i^l using only the R_i^l's of the same layer l. Equation (4.7) would then be written

$$\mu_i^l = \mu \frac{N_l R_i^l(K)}{\sum_{i=1}^{N_l} R_i^l(K)}. \tag{4.10}$$

V. Examples

A. Digital Image Halftoning

The accuracy of the proposed algorithm in estimating the weights of a single layer network is investigated first, by examining the use of a neural network in the problem of digital image halftoning. Image halftoning techniques create a bilevel image, whose average density mimics the one of a given continuous image. Let $u_{m,n}$ be the given continuous image with values in the interval [0,1] and $x_{m,n}$ the resulting bilevel one with values either 0 or 1. In the case where a sigmoidal nonlinearity is used, the values 0.1 and 0.9 are more appropriate than the 0 and 1 ones. The values -1 and 1 can also be used instead of 0 and 1. An error function for the image halftoning problem can depend on the difference between the continuous level image and an average density of the bilevel one

$$u_{m,n} - \sum_{(i,j)\in R} w_{ij} x_{m-i,n-j} \tag{5.1}$$

where R is the model support region and w_{ij} are the weight coefficients. If the above difference is equal to zero, then

$$x_{m,n} = \sum_{(i,j)\in R-(0,0)} (-w'_{ij}) x_{m-i,n-j} + w'_{00} u_{m,n} \tag{5.2}$$

where

$$w'_{ij} = \frac{w_{ij}}{w_{00}}, \qquad w'_{00} = \frac{1}{w_{00}}. \tag{5.3}$$

However, $x_{m,n}$ may take only bilevel values, thus a thresholding nonlinearity g, such as a sharp sigmoidal one, can be included in the right-hand side of (5.2). Then the error function is, in case of an $(N*N)$ image, the following sum of squares:

$$Q = \sum_{m=1}^{N} \sum_{n=1}^{N} \epsilon_{mn}^2 \tag{5.4}$$

where

$$\epsilon_{mn} = x_{m,n} - g\left(\sum_{(i,j)\in R-(0,0)} (-w'_{ij}) x_{m-i,n-j} + w'_{00} u_{m,n} \right) = 0. \tag{5.5}$$

Let us assume that we are given a pair of an original image u and a good halftoned version x of it. If we correspond the summation over r in (2.1)–(2.2) with summation over i and j in (5.4)–(5.5) and let $M=1$ in (2.1), we may easily see that (5.4)–(5.5) represent a single neuron (perceptron) network, which is trained using as input and desired response the bilevel image x and as threshold the original image u. A nonsymmetric half plane [18] model region R is used in this paper. This model region permits a space-recursive generation of the bilevel image in the usual pixel-by-pixel form, using (5.2), after the weights w'_{ij} have been estimated. The same technique can determine the weights of non-causal models, which lead to parallel implementations using Hopfield-type networks [22].

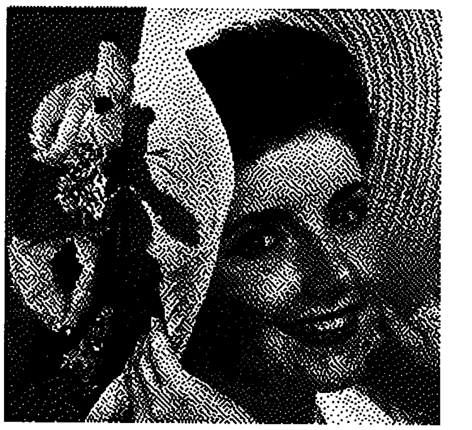
Fig. 3. A bilevel image produced by a standard halftoning technique.

An 8-bit gray-scale (256*256) image and a bilevel one, which was provided by an error diffusion halftoning algorithm [23] were used as the $u_{m,n}$ and $x_{m,n}$ images in (5.5), respectively. The bilevel image is shown in Fig. 3. The proposed algorithm, as well as the momentum-backpropagation, were used to compute the unknown coefficients w' in (5.5). A part of the above-mentioned images, the square between pixels (170,170) and (210,210) was used to provide 1600 samples of input and desired output sequences. The error measure Q, defined in (5.4) and normalized over the sum of the squares of the corresponding bilevel pixel values, was used to test the performance of the learning algorithms after each iteration. This error, expressed in decibels, is plotted in Fig. 4 as a function of the number of iterations. The number of errors and the corresponding values of Q after 100 iterations of both algorithms are given in Table I. The momentum algorithm was applied with parameters μ in the interval [0.1,0.5] and a momentum factor equal to 0.9, its final performance being similar in all cases. For a value of $\mu \geqslant 0.5$, problems of stability appeared. The iterative and recursive versions of the least squares algorithm were used. The recursive version seemed to converge slightly faster than the iterative one, but the latter was more accurate and its overall performance was slightly better. It can easily be seen that in this example both versions outperformed the backpropagation algorithm in accuracy. Moreover, the computational complexity of all methods was comparable in this case, because a fast scheme of the least squares method [18] was feasible. It can be added that fast least squares schemes can also be useful in multilayer networks, if the inputs to hidden or output units also follow a sliding window rule. This may be the case, when former values of these inputs, either in the spatial or in the temporal domain, are also used in the form of additional neuron outputs, for the training of the network.

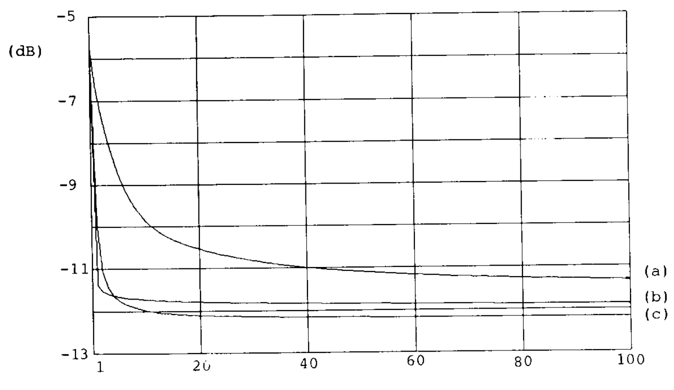

Fig. 4. Performance of the least squares and backpropagation algorithms.

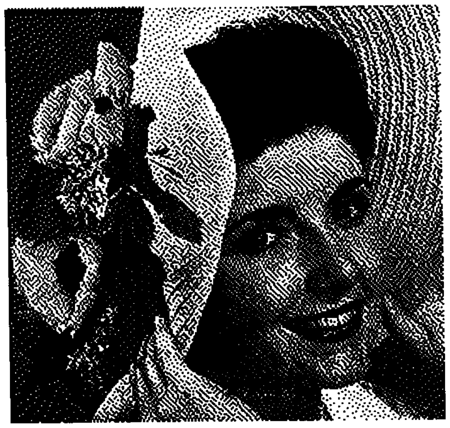

Fig. 5. A bilevel image produced by the neural network halftoning method.

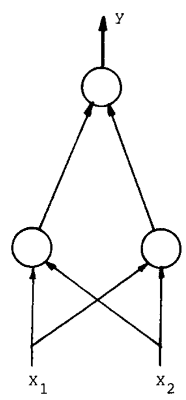

Fig. 6. A three neuron neural network able to perform the XOR function.

TABLE I
PERFORMANCE OF THE LEAST SQUARES AND BACKPROPAGATION ALGORITHM
(IMAGE HALFTONING)

METHOD(100 ITERATIONS)	NUMBER OF ERRORS	NORMALIZED ERROR(dB)
Iterative LS	114	- 12.2
Recursive LS	122	- 11.9
Backpropagation(μ=0.1)	146	- 11.3
Backpropagation(μ=0.3)	152	- 10.7

We used the whole image as testing data. Using the weight estimates w' provided by the least squares algorithm and (5.2), the bilevel image shown in Fig. 5 was generated. A comparison of the images in Figs. 3 and 5 shows that the neural network has quite successfully learnt to perform image halftoning. The above-described recursive implementation of image halftoning, of course, has a higher complexity than the error-diffusion technique. In this paper it was used as an interesting example to examine the properties of the learning algorithm. Application of this algorithm to the estimation of the weights of other image models, which lead to efficient parallel implementations, is included in a forthcoming publication.

B. The XOR Problem

This classical learning problem requires the use of at least one hidden layer in the network. The simplest form of such a network comprises three neurons, as is shown in Fig. 6. Four pairs of input and desired output patterns are used for its training

$$a\colon (0,0,0) \quad b\colon (0,1,1) \quad c\colon (1,0,1) \quad d\colon (1,1,0). \quad (5.6)$$

The following sequence of 20 patterns was generated and used as

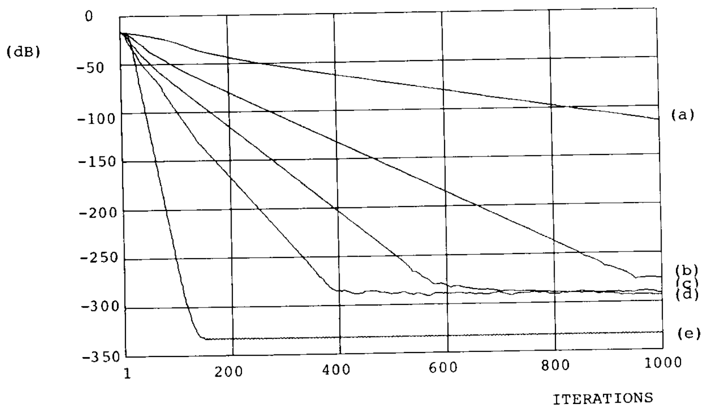

Fig. 7. Average performance (10 cases) of the momentum-backpropagation algorithm with (a) $\mu = 0.1$; (b) $\mu = 0.3$; (c) $\mu = 0.5$; (d) $\mu = 0.75$; and (e) of the adaptive least squares algorithm.

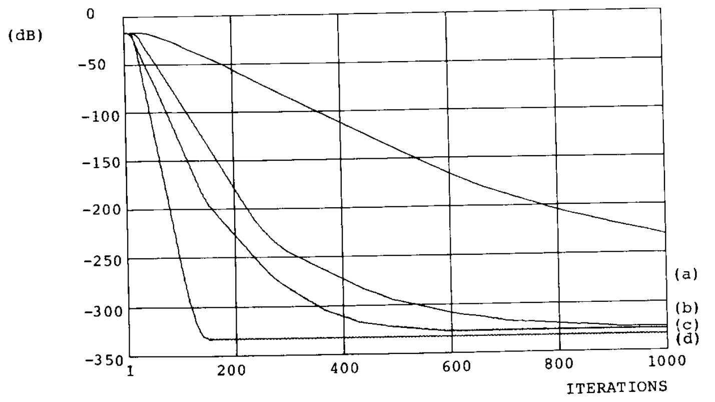

Fig. 8. Average performance of the iterative least squares algorithm with (a) $\mu = 0.1$; (b) $\mu = 0.3$; (c) $\mu = 0.5$; and (d) of the adaptive version of it.

TABLE II
PERFORMANCE OF THE LEAST SQUARES AND BACKPROPAGATION ALGORITHM (XOR FUNCTION)

METHOD	Backpropagation					Iterative LS			Adaptive LS
μ	0.1	0.3	0.5	0.75	0.9	0.1	0.3	0.5	—
ERROR (dB) 100 Iterations	-28	-52	-71	-96	-104	-32	-86	-131	-256
ERROR (dB) 500 Iterations	-72	-158	-246	-284	-299	-139	-281	-322	-332
ERROR (dB) 999 Iterations	-114	-276	-292	-293	-303	-226	-325	-327	-333

input data for each iteration of the learning algorithms:

$$bdacadcbdaaccbddacbb. \qquad (5.7)$$

Ten different sets of initial weights were generated in the range $(-0.3, 0.3)$ by feeding a random number generator with different initial seeds. The backpropagation algorithm (Fig. 7(a)–(d)) was examined in the above cases with a momentum factor = 0.9 and different values of μ. The iterative version of the least squares algorithm was also examined using the corresponding values of μ. An initial choice $P_i^l(0) = 100$ was used, when applying the latter algorithm.

The performance of the least squares algorithm using the adaptive selection of μ was also investigated. The criterion (4.1) was used, based on a simple halving technique for reducing μ, which proved equivalent to using a constant value of $\mu = 0.5$. Figs. 7 and 8 show the average normalized output error provided by the above techniques, expressed in decibels, as a function of the number of iterations. Table II summarizes the output error values after 100, 500, and 1000 iterations of the above algorithms. A value of $\mu > 0.75$ often led to an unstable behavior of the backpropagation algorithm. This result is in agreement with similar results reported in [2]. The case of $\mu = 0.9$ was included in Table II for comparison purposes, by averaging the eight (out of the ten) cases, when convergence was obtained. In this example, the least squares algorithm had a better performance than backpropagation, being also less sensitive to initial conditions. A way to measure the convergence speed of the algorithms is to multiply the number of iterations required for convergence by the number of computations, either per iteration, or per iteration and neuron in a massively parallel environment, given in (2.7) and (3.13) for the backpropagation and the least squares algorithm respectively. It is very interesting to see that the adaptive selection of μ_i^l's has greatly increased the convergence speed of the least squares algorithm, without causing any stability problems, as would have caused the use of a higher constant value of μ. Furthermore, its performance was independent of the selected initial conditions, in contrast to backpropagation, the performance of which heavily depended on the choice of initial conditions. It can also be mentioned that the above-described results favorably compare with corresponding results of the algorithm presented in [13].

VI. Conclusions

The development of a neural network learning algorithm, which is more powerful than conventional backpropagation, has been investigated in this paper. The algorithm was based on a modification of the Marquardt–Levenberg least-squares optimization method, which was suitable for the parallel updating of the input weights of each neuron in the network. A distributed, adaptive computation of the convergence rate parameter μ was presented and combined with a search criterion, used in optimization techniques. A subject of further research, which is also suggested in a recent paper [24], is to examine the application of other powerful optimization methods, such as the conjugate gradient method [9]–[10], in efficient neural network forms and to further investigate the use of other efficient convergence criteria [19] in the massively parallel neural network optimization problem. Various examples from digital image halftoning and logical operations were used to illustrate the performance of the proposed algorithm. Similar results have been obtained, when examining other problems, such as the recognition of handwritten digits. The application of the proposed technique in more complex real-world problems needs to be further examined. The least squares technique is more elaborate than backpropagation. It was shown that for an important class of applications its computational complexity can be made similar to that of backpropagation, by using fast versions of it. A study of such applications is under investigation and constitutes another topic of research.

Appendix

We will prove (3.17) first by considering the $(L-1)$ layer. Let us define β_{ij}^{L-1} as follows:

$$\sum_{r=1}^{K}\sum_{m=1}^{M}\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_i^{L-1}}\right]_r\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_j^{L-1}}\right]_r^T = \beta_{ij}^{L-1}\boldsymbol{x}^{L-1}(\boldsymbol{x}^{L-1})^T. \quad (A.1)$$

Let us also note that each $\boldsymbol{w}_i^{L-1}$ affects the outputs $y(m)$ through the hidden unit outputs x_i^{L-1} and the connections with weights w_{im}^L. If we use definition (3.19) in (A.1), we can easily derive (3.17) and then let b_i^{L-1} equal β_{ii}^{L-1}.

We will prove next (3.16) by considering the $(L-2)$ layer and using the chain rule to compute the required derivatives.

$$\sum_{m=1}^{M}\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_i^{L-2}}\right]\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_j^{L-2}}\right]^T = \sum_{m=1}^{M}\left\{\sum_{n=1}^{N_{L-1}}\left[\frac{\partial y(m)}{\partial x_n^{L-1}}\right]\left[\frac{\partial x_n^{L-1}}{\partial x_n^{L-2}}\right]\left[\frac{\partial x_n^{L-2}}{\partial \boldsymbol{w}_i^{L-2}}\right]\right\}\left\{\sum_{n'=1}^{N_{L-1}}\left[\frac{\partial y(m)}{\partial x_{n'}^{L-1}}\right]\left[\frac{\partial x_{n'}^{L-1}}{\partial x_{n'}^{L-2}}\right]\left[\frac{\partial x_{n'}^{L-2}}{\partial \boldsymbol{w}_j^{L-2}}\right]^T\right\}$$
$$= c_i^{L-2}c_j^{L-2}\sum_{m=1}^{M}\left(c_m^L\right)^2\sum_{n=1}^{N_{L-1}}c_n^{L-1}w_{nm}^L w_{in}^{L-1}\sum_{n'=1}^{N_{L-1}}c_{n'}^{L-1}w_{n'm}^L w_{jn'}^{L-1}\left[\boldsymbol{x}^{L-3}(\boldsymbol{x}^{L-3})^T\right]. \quad (A.2)$$

Therefore,

$$\beta_{ij}^{L-2} = c_i^{L-2}c_j^{L-2}\sum_{n=1}^{N_{L-1}}w_{in}^{L-1}\sum_{n'=1}^{N_{L-1}}w_{jn'}c_n^{L-1}c_{n'}^{L-1}\sum_{m=1}^{M}\left(c_m^L\right)^2 w_{nm}^L w_{n'm}^L \quad (A.3)$$

or

$$\beta_{ij}^{L-2} = c_i^{L-2}c_j^{L-2}\sum_{n=1}^{N_{L-1}}w_{in}^{L-1}\sum_{n'=1}^{N_{L-1}}w_{jn'}^{L-1}\beta_{nn'}^{L-1} \quad (A.4)$$

where the definition (3.17) for $\beta_{nn'}^{L-1}$ was used.

Generalization of this procedure in the case of the lth layer is straightforward. In the case of the $(L-3)$ layer, for example,

$$\sum_{m=1}^{M}\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_i^{L-3}}\right]\left[\frac{\partial y(m)}{\partial \boldsymbol{w}_j^{L-3}}\right]^T = c_i^{L-3}c_j^{L-3}\sum_{m=1}^{M}\left(c_m^L\right)^2\left[\sum_{n=1}^{N_{L-1}}c_n^{L-1}w_{nm}^L\sum_{k=1}^{N_{L-2}}c_k^{L-2}w_{kn}^{L-1}w_{ik}^{L-2}\right]$$
$$\cdot\left[\sum_{n'=1}^{N_{L-1}}c_{n'}^{L-1}w_{n'm}^L\sum_{k'=1}^{N_{L-2}}c_{k'}^{L-2}w_{k'n}^{L-1}w_{jk'}^{L-2}\right]\left[\boldsymbol{x}^{L-4}(\boldsymbol{x}^{L-4})^T\right]. \quad (A.5)$$

Therefore,

$$\beta_{ij}^{L-3} = c_i^{L-3}c_j^{L-3}\sum_{k=1}^{N_{L-2}}w_{ik}^{L-2}\sum_{k'=1}^{N_{L-2}}w_{jk'}^{L-2}$$
$$\cdot\left[c_k^{L-2}c_{k'}^{L-2}\sum_{m=1}^{M}\left(c_m^L\right)^2\sum_{n=1}^{N_{L-1}}c_n^{L-1}w_{nm}^L w_{kn}^{L-1}\sum_{n'=1}^{N_{L-1}}c_{n'}^{L-1}w_{n'm}^L w_{k'n'}^{L-1}\right] \quad (A.6)$$

and using (A.2)–(A.3) we get

$$\beta_{ij}^{L-3} = c_i^{L-3}c_j^{L-3}\sum_{k=1}^{N_{L-2}}w_{ik}^{L-2}\sum_{k'=1}^{N_{L-2}}w_{jk'}^{L-2}\beta_{kk'}^{L-2}. \quad (A.7)$$

References

[1] P. Werbos, "Beyond regression: New tool for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1974.

[2] D. Rumelhart, G. Hinton, and G. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1, (D. Rumelhart and J. McCleland, Eds.). Cambridge, MA: MIT Press, 1986.
[3] D. Parker, "Learning logic," MIT Tech. Rep. TR-47, Center for Comp. Res. in Econ. and Manag. Sci., 1985.
[4] T. Sejnowski and C. Rosenberg, "NET talk: A parallel network that learns to read aloud," *Comp. Syst.*, vol. 1, pp. 145-168, 1987.
[5] P. Werbos, "Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, Jan./Feb. 1987.
[6] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. Appl. Math.*, pp. 164-168, 1944.
[7] D. Marquardt, "An algorithm for least squares estimation of non-linear parameters," *J. Soc. Ind. Appl. Math.*, pp. 431-441, 1963.
[8] J. Beck and K. Arnold, *Parameter Estimation in Engineering and Science*. New York: Wiley, 1976.
[9] D. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.
[10] J. Ortega and W. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic, 1970.
[11] R. Fletcher and T. Freeman, "A modified Newton method for minimization," *J. Opt. Theory Appl.*, vol. 23, pp. 357-372, 1977.
[12] R. Watrous, "Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization," Tech. Rep. MS-CIS-87-51, LINC LAB 72, Univ. of Pennsylvania, 1986.
[13] D. Parker, "Second order backpropagation: Implementing an optimal $O(n)$ approximation to Newton's method as an artificial neural network," presented at the IEEE Conf. on Neural Information Processing Systems, Denver, CO, Nov. 1987.
[14] M. Davies and J. Whitting, "A modified form of Levenberg correction," in *Numerical Methods for Nonlinear Optimization*. (Ed. F. A. Lootsma). London, England: Academic, 1972.
[15] J. Cadzow, "Recursive digital filters synthesis via gradient based algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 349-355, 1976.
[16] G. Watson, *Approximation Theory and Numerical Methods*. New York: Wiley, 1980.
[17] G. Carayannis, N. Kalouptsidis, and D. Manolakis, "A fast sequential algorithm for filtering and prediction," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 1394-1402, 1983.
[18] Y. Boutalis, S. Kollias, and G. Carayannis, "A fast multichannel approach to adaptive image estimation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, July 1989.
[19] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[20] J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, (G. A. Watson, Ed.) Lecture Notes in Math., 630, Springer Verlag, Berlin, 1977.
[21] M. Osborne, "Nonlinear least squares: The Levenberg algorithm revisited," *J. Australian Math. Soc.*, vol. 19 (Series B), pp. 343-357, 1976.
[22] D. Anastassiou, "Digital image halftoning: A neural based approach," in *Proc. IEEE Int. Symp. on CAS*, Helsinki, Finland, June 1988.
[23] J. Jarvis, C. Judice, and W. Ninke, "A new technique for displaying continuous-tone images on bi-level displays," *Computer Graphics and Image Processing*, vol. 5, pp. 13-40, 1976.
[24] P. Werbos, "Backpropagation: Past and future," in *Proc. IEEE Int. Conf. on Neural Networks*, San Diego, CA, July 1988.

A Digital Controlled Oscillator Based on Controlled Phase Shifting

GIULIANO DONZELLINI, DANIELE D. CAVIGLIA, GIANCARLO PARODI, DOMENICO PONTA, AND PAOLO REPETTO

***Abstract*—A new architecture for digital controlled oscillators is presented. Based on a controlled phase shifting principle, the novel oscillator shows phase and frequency characteristics comparable to those of simple accumulator-type rate multiplier based circuits. The strongest point of the proposed configuration is the possibility of a high clock frequency, also with conventional low-cost technologies. Due to its characteristics, it is especially suitable for applications involving synchronization of different systems. Its transfer characteristic, frequency resolution, and phase performance are evaluated and discussed. Results from computer simulation and functional verification are reported.**

Manuscript received February 26, 1988; revised November 8, 1988 and February 24, 1989. This paper was recommended by Associate Editor T. Matsumoto.

The authors are with the Department of Biophysical and Electronic Engineering, University of Genoa, 16145 Genoa, Italy.

IEEE Log Number 8928271.

I. Introduction

The continuous advances in signal processing require the use of digital structures whose speed is of paramount importance. As an example, the transfer of standard coded signals between different systems is a common need, requiring synchronization among the systems themselves. Phase-locked loops (PLL) are often used to achieve this goal [1]–[4].

For technological reasons, PLL's are almost always implemented as analog circuits (APLL) or, sometimes, as hybrid circuits (ADPLL). As a result, such analog structures, connected to a usually completely digital system, constitute the weakest part of the system itself, because of the well-known problems of stability, sensitivity to noise, drift, etc.

Clearly, the trend in PLL's is toward the use of completely digital systems: Lindsey and Chie [5] offer a wide review of general problems and design options connected with the implementation of digital PLL (DPLL). In synchronization problems, the main obstacle is represented by the speed required to the digital components of the DPLL; often the specific structures used for its design constitute another important limitation in reaching high operating frequencies.

The aim of this paper is to contribute to the solution of these problems: a new architecture for oscillators employed in DPLL, the phase shifting digitally controlled oscillator (PSDCO), is presented.

Usually a square wave digital controlled oscillator (DCO) is based on a programmable down counter, used as a divider circuit [5], [6]. Frequencies generated are precise and stable, but their values are excessively discretized, as they are controlled by an integer parameter in an inversely proportional mode. In some cases, the designer has to resort to large ratio programmable dividers to reach a good frequency resolution, which would require impractically high input clock frequencies for operation.

When minimum rms phase jitter is not a primary requirement, DCO's can be implemented with accumulator-type rate-multiplier circuits [7]. Output frequency is obtained taking up the carry signal of a circuit calculating the phase: a good frequency resolution can be reached by increasing the length of the phase accumulator register. In order to minimize phase irregularities, proportional to the master clock period time, an high input clock frequency is needed: this is limited, however, by the speed of the phase calculating circuit, which works at master clock rate. In addition, the output waveform is almost regular only if an output by-two divider is employed: so, for a given application, the master clock frequency must double.

In the same class of rms phase jitter behavior, we find DCO's based on "add/delete" one clock cycle techniques, where phase can be easily controlled [8]–[10]. Phase control is obtained by adding or deleting one master clock cycle, then dividing down to reach an acceptable time-step/total-period ratio and duty cycle. Reported implementations usually do not provide direct control of DCO output frequency, as this structure is normally integrated in complete DPLL's. The basis add/delete circuit has to work