Two Highly Efficient Second-Order Algorithms for Training Feedforward Networks

Nikolaos Ampazis and Stavros J. Perantonis

Abstract—**In this paper, we present two highly efficient second-order algorithms for the training of multilayer feedforward neural networks. The algorithms are based on iterations of the form employed in the Levenberg–Marquardt (LM) method for nonlinear least squares problems with the inclusion of an additional adaptive momentum term arising from the formulation of the training task as a constrained optimization problem. Their implementation requires minimal additional computations compared to a standard LM iteration which are compensated, however, from their excellent convergence properties. Simulations to large scale classical neural-network benchmarks are presented which reveal the power of the two methods to obtain solutions in difficult problems, whereas other standard second-order techniques (including LM) fail to converge.**

Index Terms—**Algorithms, multilayer feedforward neural networks, nonlinear least-squares, optimization, supervised learning.**

I. Introduction

METHODS originating from the field of optimization theory have played an important role in developing training algorithms for artificial neural networks. Indeed, the realization that the training of multilayer feedforward networks can be considered as an unconstrained optimization problem has led to the introduction of a plethora of first– and second-order algorithms in the neural-networks literature [1], [2]. However, even to date, there is still a great number of problems that cannot be solved efficiently by the majority of the training algorithms that have been proposed over the years, using standard simple feedforward network architectures. In this paper, we concentrate on the development of optimization methods that can lead to powerful algorithms for the training of such networks. The existence of efficient learning methods is very important, since it is well known that the representational ability of these networks is a function of their size and architecture [3] and, therefore, limitations of the learning algorithms may prevent that potential from being fully explored [4]. Besides such an obvious disadvantage, limitations of the training algorithms can also influence additional desired network properties as, for example, the network's generalization ability, since for a given network and a set of data there may be an optimal solution which gives the best generalization, but cannot be reached by the learning algorithm. Hence the development of training algorithms that are powerful enough to find such solutions may prove to be beneficial for these additional desired properties as well, even though the original focus of the development may be on the consideration of convergence issues and not on the improvement of these properties *per se*.

One of the most powerful algorithms that have been proposed for the training of feedforward networks is undoubtedly the Levenberg–Marquardt (LM) method [5]–[8] which combines the excellent local convergence properties of Gauss–Newton method near a minimum with the consistent error decrease provided by (a suitably scaled) gradient descent far away from a solution. The LM algorithm has been compared with backpropagation (BP) and conjugate gradient (CG) in [8]. A variation of the algorithm is described in [9] where it is combined with adaptive stepsize which is heuristicaly determined. From these studies it becomes evident that the LM algorithm can be extremely effective in medium to large scale problems (up to several hundred weights) since it can train the same network from 10 to 100 times faster than BP.

A disadvantage of the LM method, however, is its increased memory requirements arising from the demand to calculate the Jacobian matrix of the error function and the need to invert matrices with dimensions equal to the number of the weights of the neural network. However this disadvantage is usually compensated for by the increased rate of convergence of the algorithm which becomes quadratic as the iterations converge toward a solution. Another disadvantage originates from the fact that, since LM is a local optimization method, it is not guaranteed to converge to the global minimum of the cost function, but is globally convergent in the sense that it is guaranteed to converge to a minimizer (local or global) of the cost function where the necessary and sufficient conditions for optimality hold [10]. Therefore, in the case that the algorithm's iterations converge toward a local minimum, there is no way of escaping and a suboptimal solution will be obtained. If such a solution is unacceptable the whole training process should be restarted with the hope that in the next trial the trajectory of the iterations will not approach a local minimizer. The increased memory requirements of the LM algorithm, however, render such a practice clearly unacceptable. Therefore, it would be extremely beneficial if the algorithm was able to handle local minimizers with increased robustness, but nevertheless maintain its fast convergence rate in the vicinity of the global miminum. In first-order methods (such as gradient descent) this problem has been dealt with the inclusion of a momentum term which in some cases might help to overshoot a local minimizer. The momentum term actually inserts second-order information in the training process and provides iterations whose form is similar to the

Manuscript received December 6, 2001.

The authors are with the Institute of Informatics and Telecommunications, National Center for Scientific Research "DEMOKRITOS," Athens, Greece (e-mail: abazis@iit.demokritos.gr; sper@iit.demokritos.gr).

Publisher Item Identifier S 1045-9227(02)05574-1.

i method. The major difference with the CG method, how-
:r, is that the coefficients regulating the weighting between
: gradient and the momentum term are heuristically selected,
ıereas in the CG algorithm these coefficients are adaptively
:ermined.

From the above discussion, it should be obvious that an
alogous methodology for including a momentum term in
:ond-order methods would be highly beneficial for dealing
th the inability of second-order methods (and consequently
LM) to escape from local minima. Our aim, of course, is
t to incorporate such a term in the weight update rule by
nply adding the momentum term multiplied with a coeffi-
:nt whose value is heuristicaly determined. On the contrary,
e purpose of this paper is to illustrate that this methodology
n be achieved by formulating the training task as a con-
:ained optimization problem whose solution effectively of-
rs the necessary framework for successfuly incorporating the
omentum term into the learning rule. We propose two very
)werful training algorithms, for training multilayer feedfor-
ard neural networks, called Levenberg–Marquardt with adap-
ve momentum (LMAM) and optimized Levenberg–Marquardt
ith adaptive momentum (OLMAM) that satisfy the desired
rgets by simultaneously combining the merits of the LM and
G techniques in order to enhance the very good properties
f LM.

In the experimental section, the proposed algorithms are
)mpared with other well-known second-order algorithms for
aining multilayer feedforward networks on training tasks
ıat are well known for their difficulty. Conventional training
lgorithms fail in solving these tasks in the majority of cases,
/hereas the proposed algorithms are shown to solve these tasks
/ith exceptionally high success rates.

This paper is organized as follows: In Section II, a descrip-
on of the LM algorithm is presented. In Sections III and IV,
ur proposed algorithms LMAM and OLMAM are introduced
espectively. In Section V, convergence issues concerning the
roposed algorithms are discussed. Section VI presents an eval-
ation of the performance of our algorithms in comparison to
ther well-known training algorithms for feedforward networks.
inally, conclusions are drawn in Section VII.

II. LM Algorithm

Let us consider a multilayer feedforward neural network
vhich consists of an input layer of neurons, an arbitrary number
)f hidden layers and an output layer, all containing neurons
vith sigmoid activation functions $f(s) = 1/(1 + \exp(-s))$.
For this network, and a set of P training patterns, the mean
square error (MSE) cost function is defined as

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{p}\sum_{i}\left(d_i^{(p)} - y_i^{(p)}\right)^2 \tag{1}$$

where $y_i^{(p)}$ and $d_i^{(p)}$ denote the output activations and desired responses of each output node i, given a pattern p, respectively, and $\boldsymbol{w}$ is the column vector containing all the weights and thresholds of the network. The functional dependence of the MSE cost function on the synaptic weights can be clearly seen if we write explicitly the expression giving the output activations $y_i^{(p)}$ (due to the forward signal propagations) as

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{p}\sum_{i}\left(d_i^{(p)} - f\left(\sum_j w_{ij}^L f\left(\sum_j w_{ij}^{L-1} y_j^{(p)}\right)\right)\right)^2 \tag{2}$$

where w_{ij}^L are the weight connections between each node i in layer L and node j in the immediately preceding layer $(L-1)$, and $y_j^{(p)}$ is the signal from node j.

The main idea in second-order methods is the local approximation of the cost function by a quadratic form given by

$$E(\boldsymbol{w}_t + \boldsymbol{dw}_t) = E(\boldsymbol{w}_t) + \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_t + \frac{1}{2}\boldsymbol{dw}_t^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_t \tag{3}$$

where $\nabla E(\boldsymbol{w}_t)$ and $\nabla^2 E(\boldsymbol{w}_t)$ are the *Gradient* vector and the matrix of second derivatives (or *Hessian* matrix) of the cost function, respectively. The optimal step (or Newton step) $\boldsymbol{dw}_t$ is obtained by the first optimality condition [10] and is given by

$$\boldsymbol{dw}_t = -[\nabla^2 E(\boldsymbol{w}_t)]^{-1}\nabla E(\boldsymbol{w}_t). \tag{4}$$

Due to the special form (sum of squares) of (1) the Hessian matrix can be written as

$$\nabla^2 E(\boldsymbol{w}_t) = \left(\boldsymbol{J}_t^T \boldsymbol{J}_t + \boldsymbol{S}_t\right) \tag{5}$$

where $\boldsymbol{J}_t$ is the *Jacobian* matrix of first derivatives of the *residuals* $(d_i^{(p)} - y_i^{(p)})$ (details of how these derivatives can be evaluated with the standard BP chain rule can be found in [8]) and $\boldsymbol{S}_t$ denotes the second-order information in $\nabla^2 E(\boldsymbol{w}_t)$ [11]. If one simply ignores the $\boldsymbol{S}_t$ term in the above expression for the Hessian, then (4) becomes the Gauss–Newton method. Near the solution, the second term is indeed approximately equal to zero [11] and, therefore, the Gauss–Newton method can achieve the quadratic convergence of Newton's method using information from first derivatives only. However, far away from the solution the term $\boldsymbol{S}_t$ is not neglibigle and the approximation to the Hessian matrix is poor, resulting to slow convergence rates and problems to the solution of (4) due to the ill-conditioning of the Jacobian matrix [12]. The ill-conditioning of the Jacobian becomes even more prominent in the case of multilayer feedforward networks due to possible redundancy of the synaptic weights and also due to the saturation of the sigmoid activation functions [13].

The LM method [5]–[8] is based on the assumption that such an approximation for the Hessian matrix is valid only inside a trust region of small radius. Therefore, under such an assumption, the optimal step can be selected by solving the following constrained optimization problem:

$$\begin{aligned} \text{Minimize} \quad & m(\boldsymbol{w}_t + \boldsymbol{dw}_t) = E(\boldsymbol{w}_t) + \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_t \\ & + \frac{1}{2}\boldsymbol{dw}_t^T \left(\boldsymbol{J}_t^T \boldsymbol{J}_t\right)\boldsymbol{dw}_t \\ \text{subject to} \quad & \|\boldsymbol{dw}_t\| \le \Delta_t \end{aligned} \tag{6}$$

where $m(\boldsymbol{w}_t + \boldsymbol{dw}_t)$ represents the local quadratic approximation of the cost function and Δ_t is the current trust region radius.

The solution of this problem is given by [5]

$$\boldsymbol{dw}_t = -\left[\left(\boldsymbol{J}_t^T \boldsymbol{J}_t + \mu_t \boldsymbol{I}\right)\right]^{-1} \nabla E(\boldsymbol{w}_t) \tag{7}$$

where $\boldsymbol{I}$ is the identity matrix and μ_t is a scalar which (indirectly) controls the size of the trust region. This means that the overall approximation to the Hessian matrix is given by

$$\nabla^2 E(\boldsymbol{w}_t) = \left(\boldsymbol{J}_t^T \boldsymbol{J}_t + \mu_t \boldsymbol{I}\right). \tag{8}$$

It can be shown that as μ_t varies between zero and ∞ then $\boldsymbol{dw}_t$ varies continuously, in a curved trajectory, between the Gauss–Newton step and a submultiple of the negative gradient [11].

Due to the difficulty in obtaining a closed form expression for the evaluation of the paramater μ_t in terms of the desired trust region radius Δ_t [14], a common implementation of the LM method in neural-network training is based on the selection of a small μ_0 which is adapted as follows during every epoch [8]: If a successful step is taken (i.e., $E(\boldsymbol{w}_t + \boldsymbol{dw}_t) < E(\boldsymbol{w}_t)$) then μ_t is decreased by a factor of ten biasing, therefore, the iteration toward the Gauss–Newton direction. On the other hand if for the current μ_t the step is unsuccessful ($E(\boldsymbol{w}_t + \boldsymbol{dw}_t) > E(\boldsymbol{w}_t)$) then μ_t is increased by the same factor until a successful step can be found (since the increase of μ_t drives $\boldsymbol{dw}_t$ to the negative gradient).

III. LMAM Algorithm

Before we proceed with the formulation of the algorithm, we wish to note that in order to minimize the cost function of (1) we will adopt an "epoch-by-epoch" optimization framework with the following basic objectives.

- At each epoch, the cost function must be decremented by a quantity δQ_t, so that at the end of learning E is rendered as small as possible. To first order, we can substitute the change in E by its first differential and demand that

$$\boldsymbol{dE}(\boldsymbol{w}_t) = \delta Q_t < 0. \tag{9}$$

- At each epoch of the learning process, the vector $\boldsymbol{w}_t$ is to be incremented by $\boldsymbol{dw}_t$ so that

$$\boldsymbol{dw}_t^T \nabla^2 E(\boldsymbol{w}_t) \boldsymbol{dw}_t = (\delta P)^2 \tag{10}$$

where δP is a small constant. Therefore the search for an optimum new point in the space of $\boldsymbol{w}$ is restricted to a small *hyperellipse* centered at the point defined by the current $\boldsymbol{w}_t$. It is well known that the extremely complex shape of the cost function landscape, which usually consists of many flat areas and elongated narrow valleys, renders gradient descent techniques very ineffective. Various methods (including the incorporation of momentum) have been proposed that deal with the problem of alleviating long jumps when the value of the gradient is high, while simultaneously avoiding the deceleration of movement when the gradient is very small. In [15] we proposed an effective optimization method where the movement of the weight vector is restricted within a small hypersphere. However a disadvantage of such an approach is that the hypersphere has the same shape in all directions which results in ignoring the underlying geometry of the space defined by the synaptic weights. In contrast, the movement within the limits of a hyperellipse, which has the same shape with the local quadratic approximation of the cost function, reflects the scaling of the problem and allows for the correct weighting among all possible directions. This has the effect that directions for which the model may differ most from the true function are restricted more than those directions for which the curvature is small [16].

The idea of restricting the weight adaptation vector within a hyperellipse has a long history in optimization theory since its was first proposed by Levenberg for the solution of the following constrained optimization problem [5]:

$$\begin{aligned} \text{Minimize} \quad & m(\boldsymbol{w}_t + \boldsymbol{dw}_t) = E(\boldsymbol{w}_t) + \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_t \\ & \qquad + \frac{1}{2} \boldsymbol{dw}_t^T \left(\boldsymbol{J}_t^T \boldsymbol{J}_t\right) \boldsymbol{dw}_t \\ \text{subject to} \quad & \boldsymbol{dw}_t^T \boldsymbol{D} \boldsymbol{dw}_t \leq \Delta_t \end{aligned} \tag{11}$$

where $m(\boldsymbol{w}_t + \boldsymbol{dw}_t)$ is the local quadratic approximation of the cost fnction, Δ_t is the current trust region radius, and $\boldsymbol{D}$ a suitably selected positive definite *diagonal* matrix.

The inclusion of the matrix $\boldsymbol{D}$ into the above formalism contributes to the simplification of the solution of the problem since diagonal trust region problems are easier to solve [16]. However, an important consideration has always been the proper selection of the matrix $\boldsymbol{D}$, so that the shape of the hyperellipse takes into account the geometry of the model. It should be obvious that the selection $\boldsymbol{D} = |\nabla^2 E(\boldsymbol{w})|$ satisfies the requirement for the consideration of the more important directions. The only disadvantage of such a choice is of course the cost of spectral factorization of the matrix $\nabla^2 E(\boldsymbol{w})$ so that the problem can be transformed to a diagonal trust region problem.

Our aim here is not to obtain an exact solution to the constrained optimization problem stated above, but to investigate the application of the weight movement restriction condition within the particular hyperellipse and then to incorporate it into a more general optimization formalism. Indeed, within the framework of the formalism, we will make use of the fact that the LM method produces positive definite approximations to the Hessian and, therefore, we can drop the absolute value dependency for estimating the hyperellipses. Hence, at this point we should just point out that if δP is small enough, then we can assume that the changes induced to the cost function $E(\boldsymbol{w}_t)$, due to the changes of the weights can be approximated by the first-order differential $dE(\boldsymbol{w}_t)$.

Having determined our basic objectives, we now focus our attention to the problem of selecting appropriate functional conditions that represent the aims that we set at the end of Section II. The main idea in the formulation of the proposed algorithm is

it a one-dimensional minimization in the direction $\boldsymbol{dw}_{t-1}$, fol-wed by a second minimization in the direction $\boldsymbol{dw}_t$ does not arantee that the function has been minimized on the subspace anned by both of these directions. A solution to this problem to choose minimization directions which are noninterfering d linearly independent. This can be achieved by the selection *conjugate* directions which form the basis of the CG method 7]. Two vectors $\boldsymbol{dw}_t$ and $\boldsymbol{dw}_{t-1}$ are noninterfering or mutu-y conjugate with respect to $\nabla^2 E(\boldsymbol{w}_t)$ when

$$\boldsymbol{dw}_t{}^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1} = 0. \tag{12}$$

Therefore, our objective is to reach a minimum of the cost nction of (1) with respect to the synaptic weights, while si-ultaneously trying to maintain the conjugacy between succes-ve weight changes by maximizing the quantity

$$\Phi_t = \boldsymbol{dw}_t{}^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1} \tag{13}$$

ithout compromising the need for a decrease in the cost func-on. Hence, in every epoch we wish to achieve the maximum ossible change in the quantity Φ_t and also to respect the basic nditions (9) and (10).

The strategy, which we adopt for the solution of this con-ined optimization problem, follows the methodology for in-rporating additional knowledge in the form of constraints in ural-network training proposed in [15] and [18]. This opti-ization problem can be solved analytically by introducing two agrange multipliers λ_1 and λ_2 to take account of (9) and (10), spectively. We introduce the function ϕ_t, which is defined as llows:

$$\begin{aligned} = \Phi_t &+ \lambda_1(\delta Q_t - dE(\boldsymbol{w}_t) \\ &+ \lambda_2\left[(\delta P)^2 - \boldsymbol{dw}_t^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_t\right]. \end{aligned} \tag{14}$$

n evaluating the differentials involved in the right-hand side, id substituting Φ_t we readily obtain

$$\begin{aligned} = \boldsymbol{dw}_t{}^T &\nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1} + \lambda_1(\delta Q_t - \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_t) \\ &+ \lambda_2\left[(\delta P)^2 - \boldsymbol{dw}_t^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_t\right]. \end{aligned} \tag{15}$$

o maximize ϕ_t at each iteration, we demand that

$$\begin{aligned} d\phi_t &= \boldsymbol{d^2 w}_t^T \cdot (\nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1} - \lambda_1 \nabla E(\boldsymbol{w}_t) \\ &\quad - 2\lambda_2 \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_t) \\ &= 0 \end{aligned} \tag{16}$$

id

$$d^2\phi_t = -2\lambda_2\left[\boldsymbol{d^2 w}_t^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{d^2 w}_t\right] < 0. \tag{17}$$

ence, from (16) we obtain

$$\boldsymbol{dw}_t = -\frac{\lambda_1}{2\lambda_2}[\nabla^2 E(\boldsymbol{w}_t)]^{-1}\nabla E(\boldsymbol{w}_t) + \frac{1}{2\lambda_2}\boldsymbol{dw}_{t-1}. \tag{18}$$

he above equation constitutes the weight update rule for the eural network. Note that (18) is similar to (4), with the im-ortant differences that in (18) there is an additional adaptive omentum term, and that the Newton step is multiplied with an daptive factor which controls its size. Due to the special form f the cost function (1), the Hessian matrix can be also approxi-mated by (8), therefore, yielding a weight update rule equivalent to the LM algorithm with an additional term of adaptive momentum (LMAM), as it is obvious from the following relation:

$$\boldsymbol{dw}_t = -\frac{\lambda_1}{2\lambda_2}\left[(\boldsymbol{J}_t^T \boldsymbol{J}_t + \mu_t \boldsymbol{I})\right]^{-1}\nabla E(\boldsymbol{w}_t) + \frac{1}{2\lambda_2}\boldsymbol{dw}_{t-1}. \tag{19}$$

By making such an approximation for the Hessian matrix, the quantity μ_t can be selected in a similar way to the one described at the end of Section II, whereas we should also note that this approximation ensures the positive definiteness for the Hessian [11]. In particular, for the implementation of the LMAM algorithm, we slightly modify the methodology proposed in [8] and we change μ_t as follows.

If a successful step is taken then μ_t is decreased by a factor of ten biasing, therefore, the iteration toward the Gauss–Newton direction. However, a step is considered successful only when

$$E(\boldsymbol{w}_t + \boldsymbol{dw}_t) < E(\boldsymbol{w}_t) + \sigma_1 \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_t \tag{20}$$

with $\sigma_1 = 0.1$. We should note that the above inequality is known as the *first Wolfe condition* [10] which actually states that the cost function should be *sufficiently* decreased. In Section V, we will show that this simple modification in the selection of μ_t ensures that the resulting algorithm is globally convergent from any starting point, that is it will converge to a stationary point of the cost function where the optimality conditions hold [10]. On the other hand, if for the current μ_t the step is unsuccessful (i.e., the above inequality does not hold) then μ_t is increased by the same factor until a successful step can be found.

Equation (18) is useful provided that λ_1 and λ_2 can be evaluated in terms of known quantities. This can be achieved as follows.

From (9) and (18), we obtain

$$\delta Q_t = \frac{1}{2\lambda_2}(I_{GF} - \lambda_1 I_{GG}) \tag{21}$$

with I_{GG} and I_{GF} given by

$$I_{GG} = \nabla E(\boldsymbol{w}_t)^T [\nabla^2 E(\boldsymbol{w}_t)]^{-1} \nabla E(\boldsymbol{w}_t) \tag{22}$$

and

$$I_{GF} = \nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_{t-1}. \tag{23}$$

Equation (21) can be readily solved for λ_1, giving

$$\lambda_1 = \frac{-2\lambda_2 \delta Q_t + I_{GF}}{I_{GG}}. \tag{24}$$

It remains to evaluate λ_2. To this end, we substitute (18) into (10) to obtain

$$4\lambda_2^2(\delta P)^2 = I_{FF} + \lambda_1^2 I_{GG} - 2\lambda_1 I_{GF} \tag{25}$$

where I_{FF} is given by

$$I_{FF} = \boldsymbol{dw}_{t-1}^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1}. \tag{26}$$

Finally, we substitute (24) into (25) and solve for λ_2 to obtain

$$\lambda_2 = \frac{1}{2}\left[\frac{I_{FF} I_{GG} - I_{GF}^2}{I_{GG}(\delta P)^2 - (\delta Q_t)^2}\right]^{1/2} \tag{27}$$

where the positive square root value has been chosen for λ_2 in order to satisfy (17) (we wish to *maximize* Φ_t) for a positive-definite Hessian matrix.

Note also the bound $|\delta Q_t| \leq \delta P\sqrt{I_{GG}}$ set on the value of δQ_t by (27) (since it is easy to see that the numerator of the fraction can never become negative, due to the Cauchy–Schwartz inequality). We always use the value

$$\delta Q_t = -\xi \delta P \sqrt{I_{GG}} \tag{28}$$

where ξ is a constant between zero and one.

Thus, the final weight update rule has only two free parameters, namely δP and ξ. The value chosen for the free parameter ξ determines the contribution of the constraints to the weight update rule. A large value of ξ means that the weight update rule is biased toward the LM step, while a small value of ξ has the opposite effect. In our simulations, which are presented in Section VI, the values recorded for δP and ξ are those giving the best performance. However, similar performances were recorded with $0.85 < \xi < 0.95$ and $0.01 < \delta P < 0.6$. The range of optimal values for ξ indicates that it is a good practice not to deviate much from the LM step, which actually predicts the maximum possible decrease in the error function, whereas the range of optimal δP values shows that the size of the trust region should be conservatively selected.

IV. OLMAM Algorithm

We have seen that the LMAM algorithm has two free parameters δP and ξ that should be externally determined for the evaluation of the adaptation of the weights according to (19). The second novel algorithm that we present in this paper with the name OLMAM is a modification of the LMAM algorithm in order to achieve independency from externally provided parameter values.

In Section III, we emphasized that our main objective is to reach a minimum of the cost fucntion with respect to the weight vector $\boldsymbol{w}$ while simultaneously trying to maintain the conjugacy between successive minimization directions, through the maximization of the quantity Φ_t given by relation (13). Since this conjugacy can be achieved only when $\Phi_t = 0$, this means that we have already made the assumption that Φ_t is bounded above by zero, that is

$$\Phi_t \leq 0. \tag{29}$$

In order to test the validity of this assumption, we can substitute (13), (18), (23), and (26) in the above relation, so that we can directly obtain

$$-\frac{\lambda_1}{2\lambda_2}\nabla E(\boldsymbol{w}_t)^T \boldsymbol{dw}_{t-1} + \frac{1}{2\lambda_2}\boldsymbol{dw}_{t-1}^T \nabla^2 E(\boldsymbol{w}_t)\boldsymbol{dw}_{t-1} \leq 0$$
$$-\lambda_1 I_{GF} + I_{FF} \leq 0. \tag{30}$$

Lagrange multiplier λ_1 is given by (24) in which the expression for the second Lagrange multiplier λ_2 is involved. Therefore, by substituting (28) into (27) we can obtain

$$\lambda_2 = \frac{1}{2}\left[\frac{A}{I_{GG}(\delta P)^2(1-\xi^2)}\right]^{1/2} \tag{31}$$

where

$$A = I_{FF}I_{GG} - I_{GF}^2. \tag{32}$$

Based on the relations (28), (31), and (32), (24) can, therefore, be written as

$$\lambda_1 = \frac{I_{GF} + \left[\frac{A}{(1-\xi^2)}\right]^{1/2}\xi}{I_{GG}}. \tag{33}$$

Substituting the above expression into relation (30) and taking into account (32) we can obtain the result

$$I_{GF}\left[\frac{A}{(1-\xi^2)}\right]^{1/2}\xi \geq A. \tag{34}$$

Due to the fact that A and ξ are positive, the above inequality can hold only when $I_{GF} > 0$. The quantity I_{GF} is the inner product between the current gradient vector $\nabla E(\boldsymbol{w}_t)$ and the vector of weight changes at the immediately preceding epoch $\boldsymbol{dw}_{t-1}$. If, at every epoch, the size of the weight changes was determined by an *exact* line minimization technique then this inner product would be equal to zero. In our case where the size of the step is limited within a hyperellipse, due to (10), the sign of I_{GF} is determined by the value of the parameter δP. If this value is large then the movement along the direction $\boldsymbol{dw}_{t-1}$ overshoots the minimum resulting in the sign of I_{GF} being negative. On the other hand if the size of δP is conservatively selected (as it is the case with LMAM), then I_{GF} is always positive and hence (34) holds.

Having justified the attempt for maximization of the quantity Φ_t in every epoch of the LMAM algorithm, we will next show the way by which the values of the parameters δP and ξ can be adaptively determined, so that the conjugacy between the vectors of weight changes in successive epochs is ensured. Hence, we should take into account the different combinations that arise from (34).

First, since we wish $\Phi_t = 0$ to hold, it is obvious that (34) implies that

$$I_{GF}\left[\frac{A}{(1-\xi^2)}\right]^{1/2}\xi = A. \tag{35}$$

Solving the above equation with respect to ξ (substituting simultaneously (32) for A) we can directly obtain the following expression which determines the optimal value of ξ at every epoch:

$$\xi = \sqrt{1 - \frac{I_{GF}^2}{I_{GG}I_{FF}}}. \tag{36}$$

We note, of course, that due to the Cauchy–Schwartz inequality it is obvious that $\xi \leq 1$.

Due to the simultaneous demand for the automatic determination of parameter δP, we should take extra care at every epoch for the monitoring of the sign of I_{GF}, since in this case it cannot be guaranteed that δP is always small. Therefore, in case that the sign is positive then the weight update rule is given again by (19), due to the demand for maximization of Φ_t. On the contrary, if due to the adaptivity of δP the sign of I_{GF} is negative,

en it should be obvious that in this case we demand the mimization of the quantity Φ_t. The effect that this demand has the learning rule is actually minimal since the only expression that changes in the optimization formalism is the sign on e right-hand side of (27), which gives Lagrange multiplier λ_2, hich in this case should be negative. Therefore, in this way, e not only ensure that the quantity Φ_t is maximized (or minized) appropriately, but also that the minimum (or maximum) at we seek is equal to zero.

So far we have discussed the way by which it is possible to sure the conjugacy between successive minimization directons through the automatic evaluation of the value of ξ from 6) and the monitoring of the sign of I_{GF}, but we have not yet fined the way by which we can automatically select the size δP. We know that the value of this parameter defines the size the hyperellipse within the limits of which the search for an ptimum new point is allowed. Since the LM step given by (7) is e solution of the trust region problem, i.e., the region in which e trust the approximation $(\boldsymbol{J}_t^T \boldsymbol{J}_t)$ for the Hessian matrix, we an safely assume that the limits of the hyperellipse should not ceed those predicted by the LM method.

Therefore, substituting (7) into (10) and taking, of course, into count the overall approximation to the Hessian matrix given y (8) we can obtain

$$\delta P = \sqrt{\nabla E(\boldsymbol{w}_t)^T [\nabla^2 E(\boldsymbol{w}_t)]^{-1} \nabla E(\boldsymbol{w}_t)} = \sqrt{I_{GG}}. \quad (37)$$

he above expression provides an estimate that can be useful or the automatic adaptation of the parameter δP based on the ptimal LM step for the constrained optimization problem (6). lowever, we should note that the calculation of the parameter $_t$ appearing in the expression for the Hessian matrix is based n the composite weight update rule of (19) and not on the pure M step of (7). Due to the fact that the value of the parameter $_t$ affects indirectly (in combination with the parameter δP) the ize of the trust region, in the implementation of the OLMAM lgorithm we use for the parameter δP a fraction of the optmal value given by (37)). In our simulations which are presented in Section VI the values recorded for the parameter δP re those giving the best performance. However similar performances were recorded with $\sqrt{I_{GG}}/64 < \delta P < \sqrt{I_{GG}}/8$.

V. Convergence Issues

In this section, we examine the convergence properties of the proposed algorithms. We show that the LMAM algorithm always converges to a stationary point of the cost function. For the OLMAM algorithm in its present form, we have not been able o show convergence conclusively, but we present our views on his issue. We shall base our analysis on convergence theory of lgorithms from the field of numerical analysis. From this field, we must first refer to Wolfe's conditions and Zoutendijk's theorem, which are necessary for our discussion.

Wolfe's conditions are imposed on the weight update $d\boldsymbol{w}$ and are stated as follows [10]:

$$E(\boldsymbol{w}_t + \alpha_t d\boldsymbol{w}_t) \leq E(\boldsymbol{w}_t) + \sigma_1 \alpha_t \nabla E(\boldsymbol{w}_t)^T d\boldsymbol{w}_t \quad (38)$$

$$\nabla E(\boldsymbol{w}_t + \alpha_t d\boldsymbol{w}_t)^T d\boldsymbol{w}_t \geq \sigma_2 \nabla E(\boldsymbol{w}_t)^T d\boldsymbol{w}_t \quad (39)$$

where α_t is the steplength, and $0 < \sigma_1 < \sigma_2 < 1$.

The first of the above conditions guarantees that the cost function reduces *sufficiently*, while the second prevents the steps $d\boldsymbol{w}$ from being too small. It can be shown that if $d\boldsymbol{w}$ is a descent direction and E is everywhere differentiable and bounded from below along $\boldsymbol{w} + d\boldsymbol{w}$, it is indeed possible to find values for the length of the vector $d\boldsymbol{w}$ that satisfy the conditions (38) and (39) [19].

The following theorem is useful for examining the convergence properties of the proposed algorithms and is due to Zoutendijk. A proof of this theorem can be found in [19].

Theorem 1: Suppose that E is bounded from below in $\Re^N$ and continuously differentiable in a neigborhood N of the level set $L: = \{E(\boldsymbol{w}) \leq E(\boldsymbol{w}_0)\}$. Also suppose that the gradient is Lipschitz continuous, i.e., there exists a constant $\epsilon > 0$ such that

$$\|\nabla E(\boldsymbol{w}) - \nabla E(\bar{\boldsymbol{w}})\| \leq \epsilon \|\boldsymbol{w} - \bar{\boldsymbol{w}}\| \quad (40)$$

for all $\boldsymbol{w}, \bar{\boldsymbol{w}} \in N$. If the iterations follow descent directions and the length of the step satisfies Wolfe's conditions (38) and (39), then

$$\sum_{t \geq 1} \cos^2 \theta_t \|\nabla E(\boldsymbol{w}_t)\|^2 < \infty \quad (41)$$

where θ_t is the angle between the actual update vector $d\boldsymbol{w}_t$ and the steepest descent direction:

$$\cos \theta_t = \frac{-\nabla E(\boldsymbol{w}_t)^T d\boldsymbol{w}_t}{\|\nabla E(\boldsymbol{w}_t)\| \|d\boldsymbol{w}_t\|} \quad (42)$$

and t is the iteration index.

An immediate corollary of Zoutendijk's theorem is the following:

Corollary 1: If all hypotheses made in Zoutendijk's theorem hold and all algorithm iterations are such that

$$\cos \theta_t \geq \delta > 0 \quad \forall t \quad (43)$$

then

$$\lim_{t \to \infty} \|\nabla E(\boldsymbol{w}_t)\| = 0 \quad (44)$$

i.e., the algorithm converges to a stationary point of the cost function.

It follows from the above considerations that in order to guarantee convergence of the proposed algorithms to a stationary point of the cost function, we should ensure the following:

1) All directions followed by the algorithm are descent directions with respect to the cost function.
2) Wolfe's conditions are satisfied throughout training.
3) There exists a constant $\delta > 0$ such that the angle θ_t between the directions followed by the algorithm and the corresponding steepest descent (gradient) directions obeys $\cos \theta_t \geq \delta$.

For both LMAM and OLMAM algorithms it is easy to show that the directions followed in the space defined by the parameters of the cost function are descent directions. Indeed, from relation (9), to first order, it follows that

$$\nabla E(\boldsymbol{w}_t)^T d\boldsymbol{w}_t = \delta Q_t < 0 \quad (45)$$

This relation guarantees that the cost function decreases along the direction defined by $\boldsymbol{dw}$ when a sufficiently small step is used.

Given that the directions of motion are descent directions, a second important step is to ensure that Wolfe's conditions are valid. These are ensured for both LMAM and OLMAM algorithms by the way μ_t is updated. Indeed, the first condition (38) follows directly from relation (20). The second condition (39) also holds for the following reason: As it is evident from the update rule for μ_t, the direction on which the first condition (38) is satisfied can only be within striking distance to a previous candidate direction, which has been already rejected for violating the sufficient decrease condition, that is, because the length of the step $\boldsymbol{dw}$ was too large. Hence, the selection of the actual direction among candidate directions is made starting from larger steps and moving slowly toward smaller steps. This prevents steps from becoming arbitrarily small and hence the second condition (39) is not violated. The situation is similar to the backtracking line minimization method, where steps, but not directions, are selected in a similar way and where Wolfe's conditions also hold (see [10] for an analysis of this point).

It remains to study the angle θ_t between the steepest descent direction and the actual direction of motion for the proposed algorithms.

Consider second-order algorithms that follow the Newton direction corresponding to (4) and assume that the condition number $k(\nabla^2 E(\boldsymbol{w}_t))$ of the matrices of second derivatives is uniformly bounded, i.e., for all iterations (t) there exists a constant $\Delta > 0$ such that

$$k(\nabla^2 E(\boldsymbol{w}_t)) = \|\nabla^2 E(\boldsymbol{w}_t)\|\|\nabla^2 E(\boldsymbol{w}_t)^{-1}\| \leq \Delta \tag{46}$$

with

$$\|\nabla^2 E(\boldsymbol{w}_t)\| = \lambda_{\max} \tag{47}$$

and

$$\|\nabla^2 E(\boldsymbol{w}_t)^{-1}\| = 1/\lambda_{\min} \tag{48}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalue of the matrix $\nabla^2 E(\boldsymbol{w}_t)$, respectively. For the LM algorithm in particular, it is reasonable to suppose that relation (46) is valid because of the presence of the variable μ_t which is introduced in order to alleviate the ill-conditioning of the Jacobian matrix of first derivatives [11]. The presence of this variable in both our proposed algorithms LMAM and OLMAM allows us to make the same assumption concerning the boundedness of the condition number.

We also consider the following property of $\nabla^2 E(\boldsymbol{w}_t)$ which follows from a well known spectral property of positive definite matrices.

Lemma 1: Consider the positive definite matrix $\nabla^2 E(\boldsymbol{w}_t)$ with maximum and minimum eigenvalues $0 < \lambda_{\min} < \lambda_{\max}$. Then $\forall z \in \mathcal{R}^N$

$$\frac{\|z\|^2}{\lambda_{\max}} \leq z^T \nabla^2 E(\boldsymbol{w}_t)^{-1} z \leq \frac{\|z\|^2}{\lambda_{\min}}. \tag{49}$$

As an interesting exercise, let us first study $\cos\theta_t$ for the LM algorithm. From the matrix inequality

$$\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\| \tag{50}$$

and from (42) it follows that

$$\begin{aligned}\cos\theta_t &\geq \frac{\nabla E(\boldsymbol{w}_t)^T \nabla^2 E(\boldsymbol{w}_t)^{-1} \nabla E(\boldsymbol{w}_t) \|\nabla^2 E(\boldsymbol{w}_t)\|}{\|\nabla E(\boldsymbol{w}_t)\|^2 \|\nabla^2 E(\boldsymbol{w}_t)\| \|\nabla^2 E(\boldsymbol{w}_t)^{-1}\|} \\ &\geq \frac{\nabla E(\boldsymbol{w}_t)^T \nabla^2 E(\boldsymbol{w}_t)^{-1} \nabla E(\boldsymbol{w}_t) \|\nabla^2 E(\boldsymbol{w}_t)\|}{\|\nabla E(\boldsymbol{w}_t)\|^2} \frac{1}{\Delta} \\ &\geq \frac{1}{\Delta}\end{aligned} \tag{51}$$

since

$$\frac{\nabla E(\boldsymbol{w}_t)^T \nabla^2 E(\boldsymbol{w}_t)^{-1} \nabla E(\boldsymbol{w}_t) \|\nabla^2 E(\boldsymbol{w}_t)\|}{\|\nabla E(\boldsymbol{w}_t)\|^2} \geq 1 \tag{52}$$

because of (49).

Moreover, Dennis and Moré [20] have shown that if the iterations follow or approach the Newton direction, the step length can be set equal to the total Newton step given by (4), since this step satisfies the Wolfe conditions. Hence, from (51) and Zoutendijk's theorem it readily follows that

$$\lim_{t\to\infty} \|\nabla E(\boldsymbol{w}_t)\| = 0 \tag{53}$$

and, therefore, the sequence of gradients converges to zero, i.e., the LM algorithm converges to a stationary point of the cost function.

Returning to our proposed algorithms, we first examine the special case whereby $\xi \to 1$. We shall show for both proposed methods that in the limit $\xi \to 1$, the iterations approach the LM step and hence convergence is guaranteed.

From (31) it is obvious that if $\xi \to 1$ then λ_2 tends to infinity and therefore the second term on the right-hand side of (19) tends to zero. Hence the weight update rule is given by

$$\boldsymbol{dw}_t = -\frac{\lambda_1}{2\lambda_2} \left[\left(\boldsymbol{J}_t^T \boldsymbol{J}_t + \mu_t \boldsymbol{I}\right)\right]^{-1} \nabla E(\boldsymbol{w}_t). \tag{54}$$

We must, therefore, examine the behavior of the fraction $\lambda_1/2\lambda_2$ as ξ tends to one. From (33) and (31) we obtain

$$\begin{aligned}&\lim_{\xi\to 1} \left(\frac{\lambda_1}{2\lambda_2}\right) \\ &\quad = \lim_{\xi\to 1} \left(\frac{I_{GF}\delta P}{\sqrt{I_{GG}} \left[\frac{A}{(1-\xi^2)}\right]^{1/2}} + \frac{\left[\frac{A}{(1-\xi^2)}\right]^{1/2} \xi \delta P}{\sqrt{I_{GG}} \left[\frac{A}{(1-\xi^2)}\right]^{1/2}} \right) \\ &\quad = \frac{\delta P}{\sqrt{I_{GG}}}.\end{aligned} \tag{55}$$

This result shows that for the LMAM algorithm, as $\xi \to 1$, the iterations tend to a submultiple of the LM step. Moreover, based on (37)) we conclude that the iterations of the OLMAM algorithm coincide with those of the LM algorithm. Therefore, the convergence of these algorithms is guaranteed in the case $\xi \to 1$. Unfortunately, this observation is not sufficient to guarantee convergence in the general case, particularly for the OLMAM algorithm where ξ is updated adaptively. To advance our anal-

is, we return to the study of the angle θ_t between the steepest escent direction and the current weight update vector given by 9).

To obtain a bound for $\cos\theta_t$ we proceed in a way similar to e one we followed for the LM algorithm. Taking into account 2), (28) and (45), it follows from (42) that

$$\cos\theta_t = \frac{\xi\delta P\sqrt{\nabla E(\boldsymbol{w}_t)^T\nabla^2 E(\boldsymbol{w}_t)^{-1}\nabla E(\boldsymbol{w}_t)}}{\|\nabla E(\boldsymbol{w}_t)\|\|\boldsymbol{dw}_t\|}. \tag{56}$$

ote that, due to (10), the magnitude of the weight update vector $\boldsymbol{w}_t$ cannot exceed the magnitude of the major axis of the hyperellipse. Hence, the following equation holds:

$$\|\boldsymbol{dw}_t\| \le 1/\sqrt{\lambda_{\min}} = \sqrt{\|\nabla^2 E(\boldsymbol{w}_t)^{-1}\|}. \tag{57}$$

rom (57) and (46) it follows that

$$\sqrt{\|\nabla^2 E(\boldsymbol{w}_t)\|}\|\boldsymbol{dw}_t\| \le \sqrt{\|\nabla^2 E(\boldsymbol{w}_t)\|\|\nabla^2 E(\boldsymbol{w}_t)^{-1}\|} \le \sqrt{\Delta}. \tag{58}$$

y combining this inequality with (50) and (52), we obtain from 56)

$$\begin{aligned}\cos\theta_t &= \frac{\xi\delta P\sqrt{\nabla E(\boldsymbol{w}_t)^T\nabla^2 E(\boldsymbol{w}_t)^{-1}\nabla E(\boldsymbol{w}_t)}\sqrt{\|\nabla^2 E(\boldsymbol{w}_t)\|}}{\|\nabla E(\boldsymbol{w}_t)\|\sqrt{\|\nabla^2 E(\boldsymbol{w}_t)\|}\|\boldsymbol{dw}_t\|}\\ &\ge \sqrt{\frac{\nabla E(\boldsymbol{w}_t)^T\nabla^2 E(\boldsymbol{w}_t)^{-1}\nabla E(\boldsymbol{w}_t)\|\nabla^2 E(\boldsymbol{w}_t)\|}{\|\nabla E(\boldsymbol{w}_t)\|^2}}\frac{\xi\delta P}{\sqrt{\Delta}}\\ &\ge \frac{\xi\delta P}{\sqrt{\Delta}}.\end{aligned} \tag{59}$$

It is interesting to observe that the right-hand side of the above nequality is bounded from above by one, since from (10), (46), 50), and (57) it follows that

$$\delta P \le \|\boldsymbol{dw}_t\|\sqrt{\|\nabla^2 E(\boldsymbol{w}_t)\|} \le \sqrt{\Delta} \tag{60}$$

and therefore

$$\delta P \le \frac{\sqrt{\Delta}}{\xi} \tag{61}$$

since $\xi < 1$.

Hence, for the LMAM algorithm, whereby the quantities ξ and δP are constant, it readily follows from (59) and Zoutendijk's condition that

$$\lim_{t\to\infty}\|\nabla E(\boldsymbol{w}_t)\| = 0. \tag{62}$$

Therefore the sequence of gradients converges to zero, i.e., the LMAM algorithm converges to a stationary point of the cost function.

Unfortunately, it is not possible to guarantee global convergence for the OLMAM algorithm by a similar argument, since ξ and δP are updated adaptively. Of course, we could select δP as follows:

$$\delta P \ge \rho\frac{\sqrt{\Delta}}{\xi} \tag{63}$$

with $\rho < 1$, which would immediately lead to

$$\cos\theta_t \ge \rho \tag{64}$$

ensuring convergence to a stationary point by Zoutendijk's theorem. Although this would involve estimation of the condition number of the Jacobian matrix, which is a computationally expensive task, we have implemented such a scheme, but the experimental results did not justify such a choice. Other schemes that relate δP and ξ were also not successful. As we have already pointed out and shall show in the experimental section, the best choice for δP from the practical point of view is equal to a submultiple of $\sqrt{I_{GG}}$, but it remains an open question to show convergence using this choice.

VI. Experimental Results

The two algorithms proposed in this paper were tested on the training of standard multilayer networks with sigmoid activation functions on a higher order parity problem (8-bit parity) and on two well known classification benchmarks, namely the Sonar [21] and 2-Spirals [22] problems. The data sets corresponding to these benchmarks are publicly available from the CMU Repository of Neural Network Benchmarks at http://www.boltz.cs.cmu.edu. Details on the network architectures for each of these benchmarks are mentioned on the corresponding paragraphs of this section dealing with the discussion of the algorithms's performances for each of these problems. The performance of the proposed algorithms was compared to that of the following well known second-order algorithms: LM [8], BFGS [1], Inverse-BFGS [23] and CG/PR (Polak-Ribiére version with restarts) [24]. All simulations were carried out on a Pentium III 450 MHz with 64 MB RAM PC, using the BILLNET neural network simulator which has been developed in our laboratory and is publically available at http://www.iit.demokritos.gr/~vasvir/billnet. MATLAB versions of each algorithm's source code have also been implemented and can found in the form of a complete MATLAB Toolbox which we have made available at http://www.iit.demokritos.gr/~abazis/toolbox. In all cases 100 training trials were performed (with uniformly random initialization of the weights in $[-0.1, 0.1]$). The maximum number of epochs was set to 5000 and training was considered successful whenever Fahlman's "40-20-40" criterion was satisfied [25] (i.e., values in the lowest 40% of the output range were treated as logical zero, values in the highest 40% were treated as logical one, and values in the middle 20% were treated as indeterminate and therefore considered as incorrect).

It is well known that parity problems are difficult tasks for feedforward networks especially as the order of the problem increases. Table I shows the results of training an 8-8-1 (eight inputs, one hidden layer with eight nodes, and one output node) network on the 8-bit parity problem. It is interesting to note that all conventional training algorithms (LM, BFGS, Inverse BFGS, CG/PR) failed to converge in all trials. On the other hand, the LMAM algorithm was able to solve the problem at least in 14% of the trials exhibiting a quite reasonable mean number of epochs, considering the size of the problem. The OLMAM algorithm exhibits a very high success rate (94%) along with

TABLE I
RESULTS IN TERMS OF NUMBER OF EPOCHS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS), COMPUTATIONAL TIME IN SECONDS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS) AND SUCCESS RATES FOR THE 8-BIT PARITY PROBLEM. NC DENOTES FAILURE OF CONVERGENCE IN ALL TRIALS

	LMAM	OLMAM	LM	BFGS	Inverse BFGS	CG/ PR
	$\delta P=0.03$ $\xi=0.95$	$\delta P=\sqrt{I_{GG}}/16$				
Epochs (Std. Dev.)	169 (114.96)	117 (90.15)	-	-	-	-
CPU Time (Std. Dev.)	19.25 (10.1)	13.96 (8.5)	-	-	-	-
Successes (%)	14	94	NC	NC	NC	NC

TABLE II
RESULTS IN TERMS OF NUMBER OF EPOCHS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS), COMPUTATIONAL TIME IN SECONDS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS) AND SUCCESS RATES FOR THE SONAR DATA PROBLEM. NC DENOTES FAILURE OF CONVERGENCE IN ALL TRIALS

	LMAM	OLMAM	LM	BFGS	Inverse BFGS	CG/ PR
	$\delta P=0.6$ $\xi=0.95$	$\delta P=\sqrt{I_{GG}}/8$				
Epochs (Std. Dev.)	21 (6.52)	49 (5.75)	10 (0.83)	-	-	-
CPU Time (Std. Dev.)	1.34 (0.42)	3.25 (0.55)	0.7 (0.1)	-	-	-
Successes (%)	69	97	7	NC	NC	NC

a smaller mean value of epochs than LMAM which, obviously, constitute the best training results. In addition, from the reported CPU times and mean number of epochs it is evident that the computational cost per epoch of the OLMAM algorithm is very similar to that of the LMAM algorithm. This is, of course, an expected result since the cost of the adaptive evaluation of the parameters ξ and δP is practically negligible compared with all other computations needed to implement the weight update rules of these algorithms.

The Sonar benchmark is a very well-known classification problem. The task is to classify reflected sonar signals in two categories (metal cylinders (mines) and rocks). The related data set comprises 208 input vectors, each with 60 components. Recently, it has been pointed out that this problem is linearly separable [26], [27]. Despite this fact, Gorman and Sejnowski report a success rate of only 85% for a single-layered perceptron, rising to 100% only when 12 hidden nodes are introduced in the feedforward neural-network architecture [21], [28]. It has been argued in [27] that the solution of this problem without hidden nodes is a difficult task because of the highly nonuniform distribution of data points in the input space. Therefore conventional algorithms may take very long training times to converge and this explains Gorman and Sejnowski's results. Hence, a challenging task is to apply the proposed algorithms to the sonar problem using a network without hidden nodes. Table II shows the results obtained for such a network, that is a network with 60 inputs, one output unit, and no hidden nodes. The BFGS

TABLE III
RESULTS IN TERMS OF NUMBER OF EPOCHS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS), COMPUTATIONAL TIME IN SECONDS (MEAN VALUE AND STANDARD DEVIATION IN PARENTHESIS) AND SUCCESS RATES FOR THE 2-SPIRALS PROBLEM. NC DENOTES FAILURE OF CONVERGENCE IN ALL TRIALS

	LMAM	OLMAM	LM	BFGS	Inverse BFGS	CG/ PR
	$\delta P=0.1$ $\xi=0.95$	$\delta P=\sqrt{I_{GG}}/64$				
Epochs (Std. Dev.)	179 (19.43)	330 (54.35)	239 (54.48)	-	-	2514 (457.76)
CPU Time (Std. Dev.)	48.51 (5.98)	88.07 (6.03)	65.89 (24.42)	-	-	361.32 (53.23)
Successes (%)	89	90	11	NC	NC	5

(both in the standard and inverse version) and CG/PR methods failed to converge in all trials, while the LM algorithm solved the problem in only 7% of the trials. On the other hand, the LMAM algorithm exhibits a relatively high success rate (69%) along with a small mean value of epochs, while the total CPU time does not exceed significantly that of the LM algorithm. The OLMAM algorithm exhibits an increase in the mean number of epochs needed to achieve convergence, but this drawback is counterbalanced by a remarkable increase in the success rate (the algorithm converged successfully in 97% of the trials).

The 2-Spirals benchmark is a two-dimensional classification problem. The task is to classify 194 data points lying along two spiral curves into two categories, one for each curve. This problem was originally proposed by *A*. Wieland as a very difficult benchmark for feedforward networks. Wieland reports that a modified version of the BP algorithm required 150 000 to 200 000 epochs to solve the problem, while conventional BP failed in all trials. Lang and Witbrock [22] used a 2-5-5-5-1 feedforward network architecture (three hidden layers with five nodes in each layer) with shortcut connections between nodes in nonadjacent layers. With this architecture, BP required on average 20 000 epochs, a version of BP with a modified cost function required around 12000 epochs, while the Quickprop algorithm required about 8000 epochs. The same authors reported that they were also able to solve the problem with a 2-5-5-1 architecture using Quickprop in 60 000 epochs. Fahlman and Lebiere [29] have used their Cascade-Correlation algorithm to solve the problem. This is a constructive method for obtaining the network architecture in the course of training. This type of network also involves shortcut connections between each new node and all previous layers (hidden and input). With this nonconventional architecture Fahlman and Lebiere were able to solve the problem with networks comprising 12 to 19 hidden nodes in 1700 epochs on average.

In this paper, we use a conventional feedforward network with only one hidden layer containing 30 nodes, without any shortcut connections between nonadjacent layers, that is we used a standard 2-30-1 feedforward network (two inputs, 30 hidden nodes, one output unit). Results are presented in Table III. The BFGS and Inverse BFGS algorithms failed to solve the problem in all trials, while the CG/PR algorithm converged successfully only in 5% of the trials exhibiting relatively

arge average values of epochs and computing time (the latter ecause of the iterative line minimization required for each poch). The LM algorithm solved the problem in 11% of the ials with a relatively satisfactory number of epochs given the fficulty of the problem. The proposed LMAM algorithm exbits a remarkable success rate of 89% as well as the smallest verage value of epochs, which, to the best of our knowledge, the smallest mean number of epochs ever reported in the edforward networks literature for this problem. Moreover, ie computing time required by LMAM is comparable to that equired by the LM algorithm, confirming the relatively small dditional computational overhead per epoch required by the roposed LMAM method. The OLMAM algorithm exhibits a uccess rate of 90% which, as far as we know, is again the ighest success rate ever reported for a conventional feedforard network attempting to solve the 2-spirals problem. Rearding the average number of epochs, we observe an increase ompared to the LMAM algorithm (330 epochs compared to 79). However, it is still very important that the fully adaptive)LMAM algorithm achieved these results without the need or careful selection of training parameters and this justifies its otential to be established as a very attractive choice among econd-order training algorithms.

VII. Conclusion

Two powerful second-order algorithms have been proposed or the training of feedforward neural networks. The algorithms ave been derived from the formulation of the training task as a onstrained optimization problem attempting to introduce conugate directions of motion within a framework similar to that of the LM algorithm. Both algorithms involve iterations simlar to the LM rule with an additional adaptive momentum term. LMAM involves two free parameters which must be tuned by the user, while OLMAM is adaptive, requiring minimal input from the end user. The convergence properties of both algorithms have been studied and the conclusion was reached that LMAM is globally convergent (in the sense that it will always converge to a stationary point of the cost function). For the convergence of OLMAM no definitive conclusion was reached, but partial results were obtained and may lead to further productive ideas. The proposed algorithms were tested on training tasks that are well known for their difficulty. Many state-of-the-art second-order training algorithms failed in solving these tasks in the majority of cases, whereas the proposed algorithms were able to solve these tasks with very high success rates. In particular, the success rates of LMAM and OLMAM were both shown to be the highest ever reported in the literature of feedforward networks. These results point to the conclusion that the proposed methods stand as very promising new tools for the efficient training of neural networks whenever the employment of second-order methods is required.

References

[1] R. Battiti, "First and second-order methods for learning: Between steepest descent and Newton's method," *Neural Computa.*, vol. 4, no. 2, pp. 141–166, 1992.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1996.

[3] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[4] R. D. Reed and R. J. Marks II, *Neural Smithing; Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA: MIT Press, 1999.

[5] K. Levenberg, "A method for the solution of certain problems in least squares," *Quart. Appl. Math.*, vol. 5, pp. 164–168, 1944.

[6] D. Marquardt, "An algorithm for least squares estimation of nonlinear parameters," *SIAM J. Appl. Math.*, vol. 11, pp. 431–441, 1963.

[7] J. J. Moré, "The Levenberg-Marquardt algorithm: Implementation and theory," in *Lecture Notes in Mathematics*. New York: Springer-Verlag, 1978, pp. 105–116.

[8] M. T. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, vol. 5, pp. 989–993, Nov. 1994.

[9] S. Kollias and D. Anastassiou, "An adaptive least squares algorithm for the efficient training of multilayered networks," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1092–1101, 1989.

[10] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer-Verlag, 1999.

[11] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Philadelphia, PA: SIAM, 1996.

[12] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. New York: Academic, 1999.

[13] S. Saarinen, R. Bramley, and G. Cybenco, "Ill-conditioning in neural network training problems," *SIAM J. Sci. Comput.*, vol. 14, no. 3, pp. 693–714, 1993.

[14] D. M. Gay, "Computing optimal locally constrained steps," *SIAM J. Sci. Statist. Comput.*, vol. 2, pp. 186–197, 1981.

[15] N. Ampazis, S. J. Perantonis, and J. G. Taylor, "Dynamics of multilayer networks in the vicinity of temporary minima," *Neural Networks*, vol. 12, no. 1, pp. 43–58, 1999.

[16] N. I. M. Gould and J. Nocedal, "On the modified absolute-value factorization norm for trust-region minimization," in *High Performance Algorithms and Software in Nonlinear Optimization*. Boston, MA: Kluwer, 1998, pp. 225–241.

[17] J. C. Gilbert and J. Nocedal, "Global convergence properties of conjugate gradient methods for optimization," *SIAM J. Optimiz.*, vol. 2, no. 1, 1992.

[18] S. J. Perantonis and D. A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, vol. 8, no. 2, pp. 237–239, 1995.

[19] J. Nocedal, "Theory of algorithms for unconstrained optimization," *Acta Numerica*, vol. 1, pp. 199–242, 1992.

[20] J. E. Dennis and J. J. Moré, "Quasi-Newton methods, motivation and theory," *SIAM Rev.*, vol. 19, pp. 46–89, 1977.

[21] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, pp. 75–89, 1988.

[22] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proc. 1988 Connectionist Models Summer School*. San Mateo, CA, pp. 52–59.

[23] R. Battiti, "Accelerated backpropagation learning: Two optimization methods," *Complex Syst.*, vol. 3, pp. 331–342, 1989.

[24] E. M. Johansson, F. U. Dowla, and D. M. Goodman, "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method," *Int. J. Neural Syst.*, vol. 2, no. 4, pp. 291–301, 1991.

[25] S. E. Fahlman, "Faster learning variations on back propagation: AN empirical study," in *Proc. 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann, pp. 38–51.

[26] J. M. Torres Moreno and M. B. Gordon, "Characterization of the sonar signals benchmark," *Neural Processing Lett.*, vol. 7, no. 1, pp. 1–4, 1998.

[27] S. J. Perantonis and V. Virvilis, "Input feature extraction for multilayered perceptrons using supervised principal components analysis," *Neural Processing Lett.*, vol. 10, no. 3, pp. 243–252, 1999.

[28] R. P. Gorman and T. J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Trans. Neural Networks*, vol. 36, pp. 1135–1140, 1988.

[29] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

Nikolaos Ampazis received the B.Eng. (Hons.) degree from the Department of Electrical and Electronic Engineering, Imperial College of Science Technology and Medicine, University of London, London. U.K., in 1992, the M.Sc. degree with distinction in information processing and neural networks from the Department of Mathematics, King's College, London, in 1995, and the Ph.D. degree with distinction in neural computing from the Department of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Athens, Greece, in 2001.

Since 1996, he has been with the National Center for Scientific Research (NCSR) "Demokritos," Athens, as a member of the Neural Networks Laboratory, Institute of Informatics and Telecommunications, participating in European and national research programs. His main research interests are in the theory, learning algorithms, and real-world applications of artificial neural networks. He has more than 20 publications in journals, books, and international conference proceedings.

Stavros J. Perantonis received the B.Sc. degree from the Department of Physics, the University of Athens, Athens, Greece, in 1984, the M.Sc. degree in computer science from the Department of Computer Science, University of Liverpool, Liverpool, U.K., in 1991, and the D.Phil. degree in computational physics from the Department of Theoretical Physics, University of Oxford, Oxford, U.K., in 1987.

He is currently Senior Researcher with the Neural Networks Laboratory, Institute of Informatics and Telecommunications, National Center for Scientific Research "Demokritos," Athens, Greece. His research interests include the theoretical analysis of neural-network systems, the development of learning algorithms and related applications, with emphasis on pattern recognition and image processing. He has authored or coauthored more than 80 publications in journals, books, and international conference proceedings, and has been involved in numerous European or national research and development projects concerning industrial or web-based neural-network applications.