Efficient Training of Neural Nets for Nonlinear Adaptive Filtering Using a Recursive Levenberg–Marquardt Algorithm

Lester S. H. Ngia, *Student Member, IEEE,* and Jonas Sjöberg, *Member, IEEE*

***Abstract*—The Levenberg—Marquardt algorithm is often superior to other training algorithms in off-line applications. This motivates the proposal of using a recursive version of the algorithm for on-line training of neural nets for nonlinear adaptive filtering. The performance of the suggested algorithm is compared with other alternative recursive algorithms, such as the recursive version of the off-line steepest-descent and Gauss—Newton algorithms. The advantages and disadvantages of the different algorithms are pointed out. The algorithms are tested on some examples, and it is shown that generally, the recursive Levenberg—Marquardt algorithm has better convergence properties than the other algorithms.**

***Index Terms*—Gauss–Newton, Levenberg–Marquardt, neural networks, nonlinear adaptive filtering, off-line training, on-line training, steepest descent.**

I. Introduction

NEURAL nets can be trained to learn from their environment. They are known to be universal approximators, e.g., [1]–[4], such that they can be used to approximate any smooth nonlinear functions. Neural net training can be performed off-line or batch, where all data are used in each training iteration of the parameter update, e.g., [5]–[8]. The other possibility is to train the net with an on-line or a recursive algorithm, where a new data sample is available in each training iteration, e.g., [9]–[12]. Such on-line algorithms will be discussed in this article, and an on-line Levenberg–Marquardt algorithm for neural net filters will be specially derived.

Consider a regressor vector $\varphi(t)$, which is based on measured input—output signals up to time $t-1$ and a parameter vector θ. A linear regression model is then expressed as $\theta^T\varphi(t)$ and a pseudo-linear regression model as $\theta^T\varphi(t, \theta)$. Such models or filter structures are described in [13] and [14]. The specific type of filter structure depends on the contents of the regressor, and the FIR and IIR filters are included in these two classes of models.

There exist several approaches to describe recursive training algorithms. In the Bayesian approach, the parameter vector is described as a stochastic time-varying variable with a certain prior probability distribution. Then, the recursive training can be described as a tracking problem, where the algorithm tries to follow the time-varying parameter vector. Another approach is to present the algorithms as modifications of off-line algorithms so that it is possible to use and generalize insights from the off-line training. An overview of the types of approaches and recursive algorithms for the linear and pseudo-linear regression models can be found in [14]–[17].

The chosen approach to derive recursive algorithms for *nonlinear regression* models is from the modifications of off-line training algorithms. The selected technique in this category of approach is similar to the one used to obtain the recursive algorithms for a pseudo-linear regression model in [14]. The *nonlinear regression* models can be described as $g(\varphi(t), \theta)$, where $g(\cdot, \cdot)$ is a nonlinear function. In this case, $g(\cdot, \cdot)$ can be a feed-forward or radial basis function neural net.

The main purpose of this paper is to introduce an on-line Levenberg–Marquardt algorithm to train neural nets for nonlinear adaptive filtering. This is motivated by the fact that Levenberg–Marquardt algorithm is often the best method in off-line training. Indeed, as shown in the examples, the recursive Levenberg–Marquardt algorithm has similar advantages over other recursive algorithms, exactly as their corresponding off-line algorithms.

The applications of neural nets as nonlinear adaptive filters are not new. For example, neural nets are trained by a variant of a recursive steepest descent algorithm in [11] and a recursive linearized least squares algorithm in [10]. The extended Kalman filter algorithm is also used to train neural nets, e.g., [18]–[20], which is derived from the earlier mentioned Bayesian approach. This algorithm is close to a recursive Gauss–Newton algorithm [14]. To the authors' knowledge, the only work that is close to this paper is found in [21], but their algorithm cannot be used to estimate time-varying systems. The derivation of their algorithm ignores the second-order information (or off-diagonal elements) of the Hessian matrix, which is a modification to simplify the computational complexity. This simplification causes slow convergence rate. On the contrary, the proposed recursive Levenberg–Marquardt algorithm can estimate time-varying systems, and it uses the second-order information in a way that does not add large computational complexity.

For the sake of clarity, the derivation is constrained to the single-input single-output (SISO) and the nonlinear regression, i.e., $g(\varphi(t), \theta)$, case. It is straightforward to generalize the result to the multi-input multi-output (MIMO) and the pseudo-nonlinear regression, i.e., $g(\varphi(t, \theta), \theta)$, case.

Manuscript received November 30, 1998; revised November 11, 1999. This work was supported in part by the Swedish Research Council for Engineering Sciences (TFR). The associate editor coordinating the review of this paper and approving it for publication was Prof. Lang Tong.

The authors are with the Signal Processing Group, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden (e-mail: ngia@s2.chalmers.se).

Publisher Item Identifier S 1053-587X(00)04936-9.

Section II formulates the problem and presents the neural nets as filter structures. Section III describes briefly the batch algorithms for off-line training to define the concepts used to describe the recursive algorithms for on-line training. The proposed recursive Levenberg–Marquardt algorithm is explained in Section IV and compared with other recursive algorithms in three system identification examples in Section V. Conclusions are provided in Section VI.

II. Problem Definition

Consider the problem of predicting an output signal $y(t)$ using past values of $y(t)$ and possibly past values of an input signal $u(t)$. It is assumed that the data can be described by a time-varying relation

$$y(t) = g(\varphi(t), \theta(t)) + \nu(t) \tag{1}$$

where

$g(\cdot, \cdot)$	memoryless function parameterized by a time-varying vector $\theta(t)$ of dimension d;
$\varphi(t)$	regressor based on measured input–output signals up to time $t-1$;
$\nu(t)$	disturbance uncorrelated with the input signal.

The time dependence of the data-generating system in (1) is assumed to be described by the time-varying parameter vector $\theta(t)$. It is assumed that $\theta(t)$ changes slowly in time as compared with $\varphi(t)$.

The goal is to have the output estimate of a model

$$\hat{y}(t, \theta) = g(\varphi(t), \theta) \tag{2}$$

to be close to the true value $y(t)$ in (1). Note that $g(\varphi(t), \theta)$ will be replaced with a simpler notation of $g(t, \theta)$ for the rest of this paper. Then, the modeling design problems can be described by the decisions on the following factors:

- model structure $g(\cdot, \cdot)$;
- regressor $\varphi(t)$;
- training algorithm to estimate parameter vector $\theta(t)$.

Since this paper considers time-varying problems where $\theta(t)$ is changing with time, the training algorithms must have a tracking capability, i.e., the parameter vector estimate $\hat{\theta}(t)$ should track the true parameter vector $\theta(t)$.

A. Filter Structures

Depending on the choice of the regressor, one can obtain the following standard models:

$$\varphi(t) = [u(t-n_k), \cdots, u(t-n_k-n_b+1)]^T \quad \text{FIR} \tag{3a}$$

$$\varphi(t) = [-y(t-1), \cdots, -y(t-n_a)]^T \quad \text{AR} \tag{3b}$$

$$\varphi(t) = [-y(t-1), \cdots, -y(t-n_a), u(t-n_k), \cdots, u(t-n_k-n_b+1)]^T \quad \text{ARX} \tag{3c}$$

where n_k is a delay, and n_a, n_b are chosen to match the orders of a dynamic system.

In the case where $g(.,.)$ in (2) is a linear function, the resulted filter can be expressed as

$$\hat{y}(t, \theta) = \theta^T \varphi(t) \tag{4}$$

which has a linear regression structure. Then, the filter structures of models in (3) are linear FIR, AR, and ARX models, respectively [13], [14]. Recursive estimation of such models is extensively covered in [14].

This paper investigates the case when the memoryless function $g(\cdot, \cdot)$ is chosen to be a feedforward or a radial basis function neural net

$$\hat{y}(t, \theta) = g(t, \theta). \tag{5}$$

However, the presented theory is generally valid for any smooth function $g(\cdot, \cdot)$. Choosing $g(\cdot, \cdot)$ to be a neural net makes it possible to describe nonlinear relations in the data, in contrast with the linear regression case in (4). From a training or estimation point of view, it is important to realize that the model (5) is nonlinear in the parameter vector θ, i.e., it has a *nonlinear regression* structure. Then, the corresponding nonlinear filter structures of models in (3) are named as NFIR, NAR, and NARX models, respectively [22]. To derive the recursive algorithms for the nonlinear regression case, the same technique applied for a pseudo-linear regression case in [14] is used, where $g(\cdot, \cdot)$ is linear, but $\varphi(t)$ depends on θ, i.e., $\hat{y}(t, \theta) = \theta^T \varphi(t, \theta)$.

Fig. 1 shows a SISO NARX model where the nonlinear dynamics consist of a feedforward neural net with $M-1$ hidden layers of sigmoidal neuron units. The model is described in Appendix A.

III. Off-Line Training

Here, a short survey of off-line training is given. It will then be used to explain the recursive algorithms in Section IV. For a more thorough treatment of the topic, some standard references are [8], [23], and [24].

Given a fixed set of data $[y(t)\,\varphi(t)]_{\{t=1\}}^N$, the parameter vector estimate is defined as minimizing the argument of a criterion

$$\hat{\theta}_N = \arg\min_{\theta} V_N(\theta). \tag{6}$$

In this paper, the following weighted mean squared error criterion is used:

$$V_N(\theta) = \frac{1}{2N} \sum_{t=1}^{N} \beta(N, t) \varepsilon^2(t, \theta) \tag{7}$$

where

$$\varepsilon(t, \theta) = y(t) - \hat{y}(t, \theta) \tag{8}$$

and $\beta(N, t)$ is a weighting sequence that can be used to give different weight to different data. If all data are equally weighted, then $\beta(N, t) = 1$. The weighting sequence will be especially useful in describing time-varying systems where recent data are more representative of the system than old data.

Starting at an initial estimate $\hat{\theta}_N^{(0)}$, most off-line algorithms can be described by the iteration of

$$\hat{\theta}_N^{(i+1)} = \hat{\theta}_N^{(i)} - \mu_N^{(i)} \left[R_N^{(i)}\right]^{-1} V_N'\left(\hat{\theta}_N^{(i)}\right) \tag{9}$$

where

$\hat{\theta}_N^{(i)}$	parameter vector estimate at iteration i;
$R_N^{(i)}$	matrix that modifies the local search direction defined by the gradient $V_N'(\hat{\theta}_N^{(i)})$;

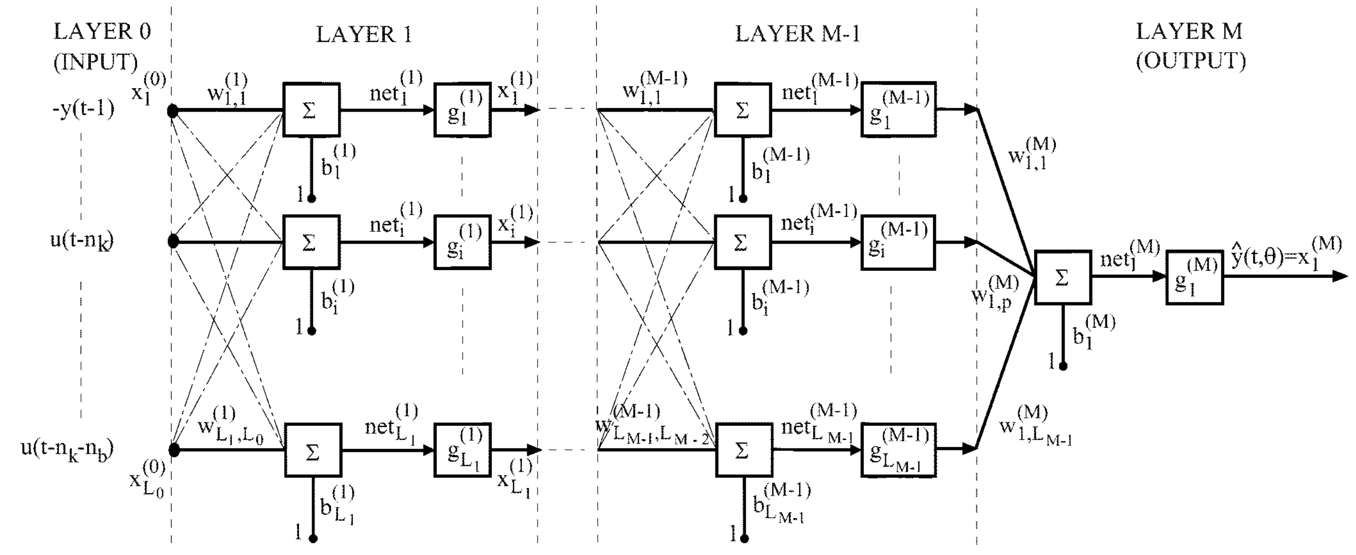

Fig. 1. Nonlinear ARX model that uses a feedforward neural net, with $M-1$ hidden layers of sigmoidal neuron units, as the nonlinear dynamics. Descriptions of the model is given in Appendix A.

$\mu_N^{(i)}$ step-length to assure that $V_N(\hat{\theta}_N^{(i)})$ decreases in every iteration.

A. Search Directions

The basis for the local search is the gradient

$$V_N'(\theta) = -\frac{1}{N}\sum_{t=1}^{N}\beta(N,t)\varepsilon(t,\theta)\psi(t,\theta) \tag{10}$$

where

$$\psi(t,\theta) = \frac{d}{d\theta}g(t,\theta). \tag{11}$$

It is well known that methods that use only the local gradient, i.e., the steepest-descent search direction, perform inefficient minimization especially in ill-conditioned problems that are close to the minimum. Then, it is optimal to use the *Newton search direction*

$$R_N^{-1}(\theta)V_N'(\theta) \tag{12}$$

where $R_N(\theta)$ is the second derivative or Hessian matrix

$$\begin{aligned} R_N(\theta) &= V_N''(\theta)\\ &= \frac{1}{N}\sum_{t=1}^{N}\beta(N,t)\psi(t,\theta)\psi(t,\theta)^T\\ &\quad - \frac{1}{N}\sum_{t=1}^{N}\beta(N,t)\varepsilon(t,\theta)\frac{d^2}{d\theta^2}g(t,\theta). \end{aligned} \tag{13}$$

The true Newton direction will thus require that the second derivative

$$\frac{d^2}{d\theta^2}g(t,\theta)$$

be computed. In addition, far from the minimum, $R_N(\theta)$ need not be positive semi-definite. Therefore, alternative search directions are usually used in practice. The following are some common off-line algorithms that are described by (9).

- *Steepest Descent Algorithm*: Take
$$R_N^{(i)} = I. \tag{14}$$
If $\mu_N^{(i)}$ is constant, then the steepest descent algorithm becomes the traditional back-propagation error algorithm [7].
- *Gauss–Newton Algorithm*: Neglect the second term in (13) to obtain
$$R_N^{(i)} = H_N^{(i)} = \frac{1}{N}\sum_{t=1}^{N}\beta(N,t)\psi\left(t,\hat{\theta}^{(i)}\right)\psi\left(t,\hat{\theta}^{(i)}\right)^T. \tag{15}$$
The neglected term can be expected to be close to zero near the minimum. In addition, by omitting this term, $R_N^{(i)}$ is guaranteed to be positive semi-definite. However, $R_N^{(i)}$ can be almost singular, and this leads to a long Gauss–Newton step in the unimportant parameter space. This is the case in an overparameterized model or when the data sets are not informative enough.
- *Levenberg–Marquardt Algorithm*: Set $\mu_N^{(i)} = 1$, and use
$$R_N^{(i)} = H_N^{(i)} + \delta^{(i)}I \tag{16}$$
where $H_N^{(i)}$ is defined by (15). Now, $\delta^{(i)} \geq 0$ is used instead of $\mu_N^{(i)}$ to assure a decrease in each iteration. Choosing $\delta^{(i)} = 0$ gives a full Gauss–Newton step, which is the largest possible update with the algorithm. Taking $\delta^{(i)} \to \infty$ will cause the diagonal elements of $R_N^{(i)}$ to dominate and thus lead to the steepest descent direction with a short step length.

It is well known that the Levenberg–Marquardt algorithm is often the best choice in many neural net training problems, e.g., [6], [8]. The reason is that the neural net minimization problems are often ill-conditioned as viewed in [25] and [26]. The Levenberg–Marquardt algorithm disregards nuisance directions in the parameter spaces that influence the criterion marginally. At the same time, it still performs an update close to the efficient Gauss–Newton directions within the subset of the important parameters. This is further explained in [27].

IV. Recursive Training

Consider the situation where in each iteration of (9), a new data sample is available, and the system is time-varying such that a weighting sequence $\beta(t, \tau)$ should be chosen to weight higher on more recent data at time t. This can be achieved, for example, by using an exponential forgetting mechanism in the error criterion

$$\begin{aligned} V_t(\theta) &= \tfrac{1}{2} \sum_{\tau=1}^{t} \beta(t, \tau)\varepsilon^2(\tau, \theta) \\ &= \tfrac{1}{2} \sum_{\tau=1}^{t} \lambda^{t-\tau}\varepsilon^2(\tau, \theta) \end{aligned} \tag{17}$$

where $0 < \lambda \leq 1$ is the *forgetting factor*[1] that decides how fast the relevance of past data should decrease. Note that the normalization of the criterion is changed as compared with (7). The influence of λ can be described by the bias-variance error or overfitting and underfitting perspective.

- A small value of λ implies that only recent data are included in the criterion. This allows a fast-varying $\theta(t)$ to be tracked properly, which results into a low bias error in the estimate. The drawback is that the estimate $\hat{\theta}(t)$ will be more influenced by the noise term $\nu(t)$, which gives a high variance error in the estimate.
- A value of λ close to 1 gives the opposite effects. Due to the time-varying system, old data that are generated by different values of θ are included in the criterion. This gives an estimate $\hat{\theta}(t)$, which is biased toward $\theta(\tau)$, where $\tau < t$. Hence, $\lambda \to 1$ includes more old data and causes a larger bias contribution to the estimation error.

The size or complexity of the filter structure $g(\cdot, \cdot)$ is also related to the choice of λ. If more old data are used in the estimation, then a more complex $g(\cdot, \cdot)$ containing more parameters can be used. Likewise, a smaller λ implies that a simpler filter structure $g(\cdot, \cdot)$ must be considered. Therefore, the optimum value of λ is a tradeoff between how fast the filter should track the changes in θ and the complexity of the filter structure. The bias-variance error aspects in connection to nonlinear black-box modeling is further discussed in [22].

A. Recursive Algorithm

The following derivation of recursive algorithm is close to the one called the *recursive maximum likelihood* algorithm in [14] and [28], which is used in the references to obtain a recursive algorithm for a pseudo-linear regression model.

Let $\hat{\theta}(t-1)$ be the estimate at time $t-1$, and assume that it minimizes $V_{t-1}(\theta)$. The goal is to compute $\hat{\theta}(t)$. Consider a Taylor expansion of $V_t(\theta)$ at the current estimate

$$\begin{aligned} V_t(\theta) =& V_t\left(\hat{\theta}(t-1)\right) + V_t'\left(\hat{\theta}(t-1)\right)\left[\theta - \hat{\theta}(t-1)\right] \\ &+ \tfrac{1}{2}\left[\theta - \hat{\theta}(t-1)\right]^T V_t''\left(\hat{\theta}(t-1)\right)\left[\theta - \hat{\theta}(t-1)\right] \\ &+ o\left(\left|\theta - \hat{\theta}(t-1)\right|^2\right). \end{aligned} \tag{18}$$

[1]Typically, the choice of λ is $0.95 \leq \lambda \leq 1$.

Neglecting the remainder $o(|\theta - \hat{\theta}(t-1)|^2)$, the parameter update can be chosen as the minimizer of (18)

$$\hat{\theta}(t) = \hat{\theta}(t-1) - \left[V_t''\left(\hat{\theta}(t-1)\right)\right]^{-1} V_t'\left(\hat{\theta}(t-1)\right)^T. \tag{19}$$

Differentiating (17) with respect to θ gives

$$\begin{aligned} V_t'(\theta)^T &= -\sum_{\tau=1}^{t} \lambda^{t-\tau}\psi(\tau, \theta)\varepsilon(\tau, \theta) \\ &= \lambda V_{t-1}'(\theta)^T - \psi(t, \theta)\varepsilon(t, \theta) \end{aligned} \tag{20}$$

and differentiating once more

$$V_t''(\theta) = \lambda V_{t-1}''(\theta)^T + \psi(t, \theta)\psi(t, \theta)^T - \varepsilon''(t, \theta)\varepsilon(t, \theta). \tag{21}$$

According to the assumption that $\hat{\theta}(t-1)$ is indeed the optimal estimate at time $t-1$, then $V_{t-1}'(\hat{\theta}(t-1)) = 0$. This paper uses the approximation that $\varepsilon''(t, \hat{\theta}(t-1))\varepsilon(t, \hat{\theta}(t-1)) = 0$. This approximation should be good when the estimate $\hat{\theta}(t-1)$ is close to its true value since then, $\varepsilon(t, \hat{\theta}(t-1))$ is almost a white noise and uncorrelated with $\varepsilon''(t, \hat{\theta}(t-1))$.

Thus, (19) can be rewritten as

$$\hat{\theta}(t) = \hat{\theta}(t-1) + H(t)^{-1}\psi\left(t, \hat{\theta}(t-1)\right)\varepsilon\left(t, \hat{\theta}(t-1)\right) \tag{22a}$$

$$H(t) = \lambda H(t-1) + \psi\left(t, \hat{\theta}(t-1)\right)\psi\left(t, \hat{\theta}(t-1)\right)^T \tag{22b}$$

where $H(t)$ is the approximation of the Hessian matrix $V_t''(\theta)$.

By comparing the off-line algorithm in (9) with the recursive algorithm in (22), a general recursive algorithm to update the parameter vector can be defined as

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \mu(t)R(t)^{-1}\psi(t, \theta)\varepsilon(t, \theta) \tag{23}$$

where the design variable $\mu(t)$ is a gain, and $R(t)$ is similar to $H(t)$ at time t. The initial value $\hat{\theta}(0)$ can be considered to be a prior estimate of $\hat{\theta}(t)$. If prior information about θ is unavailable, then $\hat{\theta}(0)$ can be chosen to be small random values with uniform distribution.

Select $R(t)$ and $\mu(t)$ as follows.

- *Steepest descent*: Choose $R(t) = I$ to provide

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \mu(t)\psi(t, \theta)\varepsilon(t, \theta) \tag{24}$$

  and $\mu(t)$ can be a given sequence or normalized as

$$\mu(t) = \frac{\mu}{|\varphi(t)|^2} \tag{25}$$

  where μ is a constant gain. In a linear regression model (4), this algorithm is the same with the normalized least mean square (NLMS) algorithm in, e.g., [17].
- *Gauss–Newton*: Choose $R(t) = (1-\lambda)H(t)$ to normalize $H(t)$ in (22b) and $\mu(t) = (1-\lambda)$ to give

$$\hat{\theta}(t+1) = \hat{\theta}(t) + (1-\lambda)R(t)^{-1}\psi(t, \theta)\varepsilon(t, \theta) \tag{26a}$$

$$R(t) = \lambda R(t-1) + (1-\lambda)\psi(t, \theta)\psi(t, \theta)^T. \tag{26b}$$

  In the linear case, this algorithm is identical to the recursive least square (RLS) algorithm in, e.g., [17].

- *Levenberg–Marquardt*: Choose $R(t) = (1-\lambda)H(t) + \delta I$ and $\mu(t) = (1-\lambda)$ to yield

$$\hat{\theta}(t+1) = \hat{\theta}(t) + (1-\lambda)R(t)^{-1}\psi(t,\theta)\varepsilon(t,\theta) \tag{27a}$$

$$R(t) = \lambda R(t-1) + (1-\lambda)[\psi(t,\theta)\psi(t,\theta)^T + \delta I]. \tag{27b}$$

If $g(.,.)$ is linear, then this algorithm is equivalent to the regularized RLS in [29].

1) Avoiding the Matrix Inversion: A direct solution of (23) is not practical for the recursive Gauss–Newton and Levenberg–Marquardt algorithms because the inversion of $R(t)$ requires a computation complexity of $O(d^3)$ ($d = \dim\theta$). Using the *matrix inversion lemma* can reduce the computation complexity from $O(d^3)$ to $O(d^2)$. The *lemma* expresses the following inversion:

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B\left[DA^{-1}B + C^{-1}\right]^{-1}DA^{-1}. \tag{28}$$

For the recursive Gauss–Newton algorithm, this paper chooses $A = \lambda R(t-1)$, $B = (1-\lambda)\psi(t,\theta)$, $C = 1$, and $D = \psi(t,\theta)^T$. After introducing $P(t) = (1-\lambda)R(t)^{-1}$, the recursive Gauss–Newton algorithm in (26) is rewritten as

$$\hat{\theta}(t+1) = \hat{\theta}(t) + P(t)\psi(t,\theta)\varepsilon(t,\theta) \tag{29a}$$

$$P(t) = \left\{P(t-1) - \frac{P(t-1)\psi(t,\theta)\psi(t,\theta)^T P(t-1)}{\lambda + \psi(t,\theta)^T P(t-1)\psi(t,\theta)}\right\}\Big/\lambda. \tag{29b}$$

The matrix $P(t)$ is the covariance matrix of the estimate $\hat{\theta}(t)$, and $P(0)$ should be chosen with large positive elements if there is little confidence in $\hat{\theta}(0)$. For example, $P(0) = 1 \cdot 10^4 I$. Now, the recursion (29) contains an inversion of a matrix of only dimension 1 because the matrix $[DA^{-1}B + C^{-1}] = 1 + \lambda^{-1}\psi(t,\theta)^T P(t-1)\psi(t,\theta)$ has rank 1. The reason is that the updating quantity $(1-\lambda)\psi(t,\theta)\psi(t,\theta)^T$ in (26b) is of rank 1.

Unfortunately, the *lemma* cannot be used directly on the recursive Levenberg–Marquardt algorithm in (27) since the updating quantity in (27b) is of rank d. A possible remedy to this problem is to add δd to $\psi(t,\theta)\psi(t,\theta)^T$ at one diagonal element at a time, as suggested in [14]. Then, (27b) is approximated by

$$R(t) = \lambda R(t-1) + (1-\lambda)[\psi(t,\theta)\psi(t,\theta)^T + \delta d Z_d(t)] \tag{30}$$

where $Z_d(t)$ is a $d \times d$ zeros matrix, except for one of its diagonal elements at $(t \bmod d) + 1$, which is 1. After a lapse of d time samples, (30) is virtually the same as (27b).

Expression (30) can be rewritten as

$$R(t) = \lambda R(t-1) + (1-\lambda)\left[\psi^*(t,\theta)\Lambda^*(t)^{-1}\psi^*(t,\theta)\right]^T \tag{31}$$

where $\psi^*(t,\theta)$ is a $d \times 2$ matrix where column 1 contains $\psi(t,\theta)$ and column 2 contains a $d \times 1$ zeros vector except for its $(t \bmod d) + 1$ element, which is 1

$$\psi^*(t,\theta) = \begin{pmatrix} & & \psi(t,\theta)^T & & & \\ 0 & \cdots & 0 & 1 & \cdots & 0 \end{pmatrix}^T \tag{32}$$

$$\Uparrow$$
$$\text{position} = (t \bmod d) + 1$$

TABLE I
NUMBER OF FLOPS PER RECURSION THAT ARE USED TO UPDATE $P(t)$

	Recursive Gauss-Newton	Recursive Levenberg-Marquardt
Flops	$\approx d(3.5d + 2.5)$	$\approx d(4.5d + 9.5)$

and the weighting matrix is

$$\Lambda^*(t)^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \delta d \end{pmatrix}. \tag{33}$$

The *lemma* can now be applied to (31) with $A = \lambda R(t-1)$, $B = (1-\lambda)\psi^*(t,\theta)$, $C = \Lambda^*(t)^{-1}$, and $D = \psi^*(t,\theta)^T$. Using the substitution $P(t) = (1-\lambda)R^{-1}(t)$, the following recursive Levenberg–Marquardt algorithm is obtained, which replaces (27):

$$\hat{\theta}(t+1) = \hat{\theta}(t) + P(t)\psi(t,\theta)\varepsilon(t,\theta) \tag{34a}$$

$$P(t) = \{P(t-1) - P(t-1)\psi^*(t,\theta) S(t)^{-1}\psi^*(t,\theta)^T P(t-1)\}/\lambda \tag{34b}$$

$$S(t) = \psi^*(t,\theta)^T P(t-1)\psi^*(t,\theta) + \lambda\Lambda^*(t). \tag{34c}$$

The definition of matrix $P(t)$ in (34b) is the same as (29b) in the recursive Gauss–Newton algorithm. The dimension of $S(t)$ is now only 2 instead of d; therefore, a much smaller matrix needs to be inverted every time.

The only model structure dependent quantity in (24), (29), and (34) is the gradient vector $\psi(t,\theta)$. The computation of the vector $\psi(t,\theta)$ is the same for all algorithms. It is straightforward to derive $\psi(t,\theta)$ by using the same technique as in the backpropagation error algorithm, e.g., [7], which is summarized in Appendix B.

A comparison concerning the number of flops required to update the matrix $P(t)$ in (29b) and (34b) for the respective recursive Gauss–Newton and Levenberg–Marquardt algorithms is given in Table I. Note that the stated arithmetic complexity does not include the computation of $\psi(t,\theta)$. Thus, the recursive Levenberg–Marquardt algorithm needs $d^2 + 7d$ flops more than the recursive Gauss–Newton algorithm to update the matrix $P(t)$.

2) Choice of Value for δ: A good choice of value for δ is important in the off-line Levenberg–Marquardt algorithm [27]. In the parameter directions corresponding to eigenvalues of the Hessian matrix larger than δ, the parameter update is close to that of the efficient Gauss–Newton step. In the parameter directions corresponding to eigenvalues smaller than δ, only a marginal update is taken in this unimportant parameter space. These effects from choosing the correct value for δ are similar in the recursive case. Hence, it is crucial to have a good technique to choose the value for δ in the recursive Levenberg–Marquardt algorithm, which is discussed in the following.

From the off-line training or minimization perspective, the Levenberg–Marquardt method is the solution to the trust region approach problem [23], [24]

$$\hat{\theta}_N^{(i+1)} = \arg\min_{\hat{\theta}_N^{(i)}} V_N(\hat{\theta}_N^{(i)})$$
$$\text{subject to } \left|\hat{\theta}_N^{(i+1)} - \hat{\theta}_N^{(i)}\right| < \xi \tag{35}$$

where ξ is a trust region radius, which is related to δ in a complicated manner. A good value of δ (or ξ) must be chosen such that the off-line criterion is decreased.

There are many ways to implement an off-line Levenberg–Marquardt algorithm that solves (35). One way is to have an algorithm that adjusts the value of δ according to a relationship between the actual and predicted change in the criterion, e.g., [30]. Alternatively, ξ may be adjusted, and an implicitly defined value of δ can be computed according to the actual reduction in the criterion, e.g., [31]. The proposed recursive Levenberg–Marquardt algorithm modifies the off-line adjustment algorithm for δ in [30] to allow δ to be adapted over time.

The new algorithm for the adaptive $\delta(t)$ adjusts the value once after every m time samples. Due to the remedy in (30), this paper sets $m = d(\dim\theta)$ so that each diagonal element in $\psi(t, \theta)\psi(t, \theta)^T$ has been added with δd before the value for $\delta(t)$ is adjusted. From (34), it follows that the predicted reduction $r_p(t)$ of the on-line criterion in (17) at time t is[2]

$$r_p(t) = [\psi(t, \theta)\varepsilon(t, \theta)]^T P(t)\psi(t, \theta)\varepsilon(t, \theta). \tag{36}$$

To monitor the actual reduction $r_a(t)$ in the residual variance over the window of d time samples, a new cost function is introduced.

$$r_a(t) = \frac{1}{2d}\left(\sum_{\tau=t-2d+1}^{t-d} \varepsilon^2\left(\tau, \hat{\theta}(\tau)\right) - \sum_{\tau=t-d+1}^{t} \varepsilon^2\left(\tau, \hat{\theta}(\tau)\right)\right). \tag{37}$$

Then, the values for $\delta(t)$ can be adjusted with the following strategy.

- Increase $\delta(t-1)$ by a factor κ if $r_a(t)/r_p(t)$ is smaller than a threshold ζ.
- Decrease $\delta(t-1)$ by a factor $1/\kappa$ if $r_a(t)/r_p(t)$ is larger than a threshold $(1-\zeta)$.

This is summarized as

$$\delta(t) = \begin{cases} \kappa\delta(t-1), & \text{if } r_a(t) < \zeta r_p(t) \\ \delta(t-1)/\kappa, & \text{if } r_a(t) > (1-\zeta) r_p(t) \\ \delta(t-1), & \text{otherwise} \end{cases} \tag{38}$$

where $\delta(0)$, κ, and ζ are design variables.

The following are the choice of values for the design variables.

- $\delta(0)$: It can be quite a small positive value, e.g., $10^{-4} - 10^{-6}$, as compared with the magnitude of the elements in $\psi(t, \theta)$. The choice is motivated in the same way as using large positive elements in $P(0)$, i.e., to have a large parameter update in the beginning to reflect the little confidence in $\hat{\theta}(0)$.
- κ: It can be any arbitrary positive value of $\kappa > 1$, e.g., 5.
- ζ: It can be any arbitrary positive value constrained to be $0 < \zeta < 0.5$. A typical value is 0.25.

[2]Note that $r_p(t)$ is computed once after every d time samples. An alternative method is to compute an average value of $r_p(t)$ over the window of d time samples. This alternative method requires more computations but can give lesser fluctuation in $\delta(t)$ than the method in (36). However, experiences shows that both methods give similar adjustments in $\delta(t)$. Therefore, the method of computing $r_p(t)$ in (36) is chosen because of its low computational load.

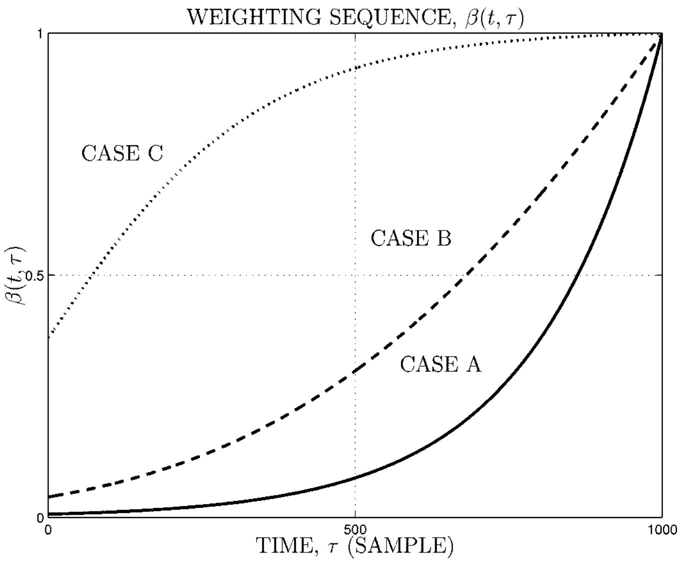

Fig. 2. Weighting sequence $\beta(t, \tau)$ as an increasing function of time τ when $\lambda(0) = 0.995$, $\lambda_r = 1$ (Case a), $\lambda_r = 0.999$ (Case b) and $\lambda_r = 0.995$ (Case c).

The computation of (36)–(38) does not seriously augment the complexity of the recursive Levenberg–Marquardt algorithm because they can be obtained from previous calculations. For example, the computation of $r_p(t)$ in (36) requires only about $2d$ flops after every d time samples or an average of two flops per recursion. This is because the term $P(t)\psi(t, \theta)\varepsilon(t, \theta)$ can be obtained directly from (34a).

3) Choice of λ in a Slow Time-Varying System: In a system with almost constant parameters, it is natural to have $\lambda = 1$. However, the estimates of the variables in the gradient are rather poor in the beginning of the recursion and should therefore carry a lower weight in the criterion compared with later measurements. Thus, it is useful to let $\lambda(t)$ grow exponentially to 1 over time, i.e., $\lambda(t) \to 1$ as $t \to \infty$ by using

$$\lambda(t) = \lambda_r\lambda(t-1) + (1-\lambda_r) \tag{39}$$

where the rate λ_r and initial value $\lambda(0)$ are design variables. A typical value for λ_r is $0.995 < \lambda_r < 1$ and for $\lambda(0)$ is $0.95 < \lambda(0) < 1$. If $\lambda_r = 1$, then $\lambda(t)$ is a constant value, i.e., $\lambda(t) = \lambda(0)$.

Now, the weighting sequence $\beta(t, \tau)$ is

$$\beta(t, \tau) = \begin{cases} \prod_{j=\tau}^{t-1}\lambda(j) & \text{for } 1 \le \tau \le t-1 \\ 1 & \tau = t \end{cases} \tag{40}$$

and it is plotted in Fig. 2 for different values of λ_r with $\lambda(0) = 0.995$ and $t = 1 \cdot 10^3$ samples. Note the sensitivity of $\beta(t, \tau)$ with respect to λ_r. Case a discounts more old data from the on-line error criterion (17) than Case b and Case c, although the differences in the value of λ_r between these cases are small. This sensitivity will be inherited in the recursive Gauss–Newton and Levenberg–Marquardt algorithms if (39) and (40) are applied to their error criterion (17).

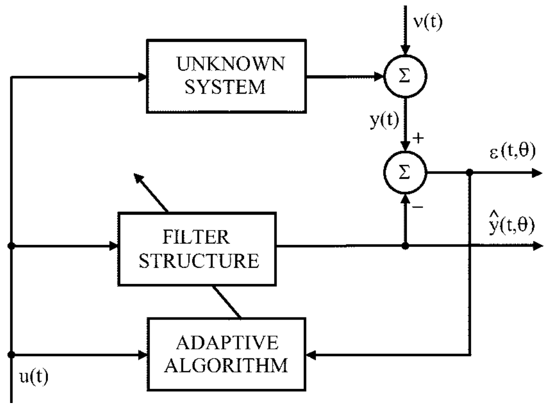

Fig. 3. Adaptive filter used in a system identification mode.

TABLE II
COMPARISON OF THE NUMBER OF FLOPS PER RECURSION TO THE IMPLEMENT EACH ALGORITHM

	RecSD	RecGN	RecLM
Flops	$\approx d(4N_c+2)K_n$	$\approx 5.5d(d+1)$	$\approx d(6.5d+12.5)$

The overall proposed recursive Levenberg–Marquardt algorithm as described in (34)–(40) is summarized in Appendix C.

V. EXAMPLES

In the examples, the adaptive filter is used as a system identifier as shown in Fig. 3. The adjustable filter has an input signal $u(t)$ and produces an output signal $\hat{y}(t, \theta)$. The adaptive algorithm adjusts the parameters in the filter to minimize the error $\varepsilon(t, \theta) = y(t) - \hat{y}(t, \theta)$, where $y(t)$ is the response from the unknown system that is corrupted by a disturbance $\nu(t)$.

The proposed recursive Levenberg–Marquardt algorithm is tested on three examples. To provide some comparisons, two other recursive algorithms are also applied to the same filter structure in each example. All filter structures belong to the general SISO NARX model as shown in Fig. 1. The algorithms follow.

- *Recursive Steepest-Descent Algorithm (RecSD)*: The algorithm suggested in [11] is used, which is a variant of (24) and (25). It has a rectangular sliding data window of length N_c and at each time t, K_n iterations of parameter update are performed. The use of $N_c > 1$ and $K_n > 1$ helps to improve the tracking and convergence rate at the expense of increased computational complexity. If $N_c = K_n = 1$, then the suggested algorithm is the same as the one in (24) and (25).
- *Recursive Gauss–Newton Algorithm (RecGN)*: It uses the algorithm in (29), which is the same as a recursive linearized least squares algorithm in [10].
- *Recursive Levenberg–Marquardt Algorithm (RecLM)*: It applies the algorithms described in (34)–(40), which is summarized in Appendix C.

Table II compares the number of flops required to implement the RecSD, RecGN, and RecLM algorithms. Similarly with Table I, the number of flops does not include the computation of $\psi(t, \theta)$.

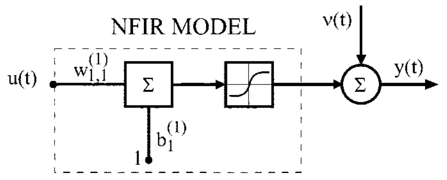

Fig. 4. Time-invariant NFIR system with 1 layer neural nets for Example 1.

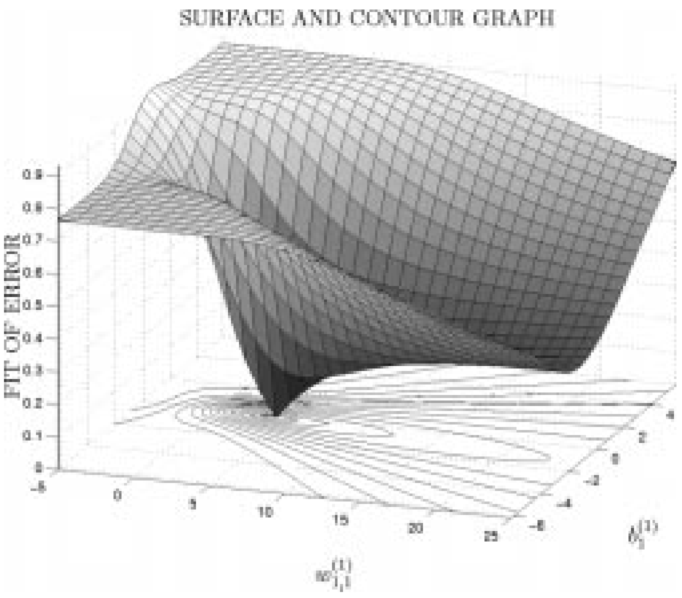

Fig. 5. Surface and contour graph for the fit of the off-line error criterion (7) with $\beta(N, t) = 1$.

A. Example 1—Influence of Design Variable $\delta(t)$ in a Simulated Time-Invariant System

This example illustrates the performance of the algorithms in a time-invariant system. A neural net as shown in Fig. 4 is the system to be identified. It has a constant $\theta = [w^{(1)}_{1,1}, b^{(1)}_1]^T = [4, -0.2]^T$. The input signal is a sequence of $5 \cdot 10^4$ samples of white noise with mean 0 and variance 1. Fig. 5 shows the surface and contour graph for the fit of the off-line error criterion (7) with $\beta(N, t) = 1$ as a function of the bias $b^{(1)}_1$ and weight $w^{(1)}_{1,1}$.

An NFIR filter structure, which is the same as the NFIR model in the system shown in Fig. 4, is used. It has $n_b = 1$ and $n_k = 0$ in the regressor $\varphi(t)$ of (3a), i.e., $\varphi(t) = u(t)$. The values of the design variables in RecSD, RecGN, and RecLM are shown in Table III, except for $\delta(t)$ in RecLM. Sets of different values for $\delta(t)$ will be given in the example. Two different initial values of $\hat{\theta}(0)$ are chosen: Case 1 uses $\hat{\theta}(0) = [17, 4]^T$, and Case 2 uses $\hat{\theta}(0) = [7, -2]^T$. In each case, two kinds of experiments are performed on RecLM. The first experiment uses two different values of constant δ, and the second one uses the adaptive $\delta(t)$, as described in (36)–(38).

In Case 1, RecLM begins with a constant δ of $1 \cdot 10^{-3}$ and $5 \cdot 10^{-3}$. Fig. 6 shows the locus of $\hat{\theta}(t)$ on the same error contour of Fig. 5. The following are the main points.

- Among the algorithms, RecSD will converge the slowest due to its zig-zag path, and RecGN will converge the fastest due to its shortest path to the target.

TABLE III
VALUES OF DESIGN VARIABLES IN EACH ALGORITHM FOR EXAMPLE 1

RecSD	RecGN	RecLM
$N_c = 1$	$P(0) = 1 \cdot 10^4$	$P(0) = 1 \cdot 10^4$
$K_n = 1$	$\lambda = 0.999$	$\lambda(0) = 0.999$
$\mu = 0.5$		$\lambda_r = 1$

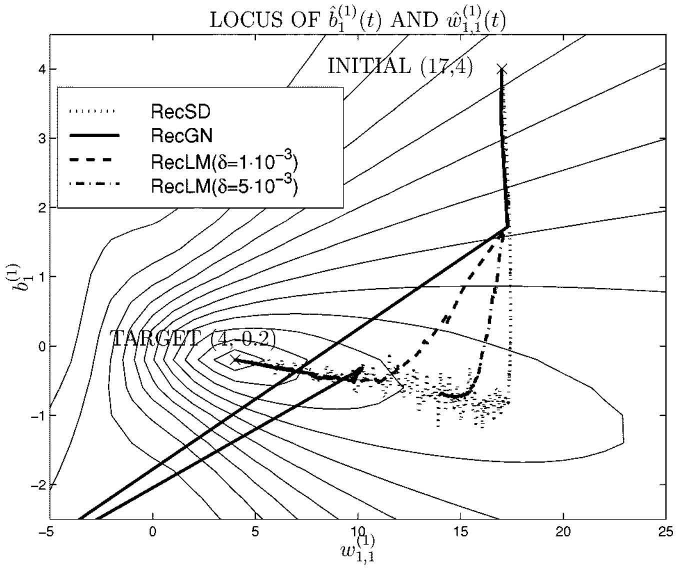

Fig. 6. Locus of parameter estimate $\hat{\theta}(t)$ on the error contour for Case 1 ($\hat{\theta}(0) = [17, 4]^T$). Note that a small part of the locus for RecGN is outside the figure.

- In the beginning, RecGN has, however, taken few long steps in the nuisance parameter space before quickly returning to the correct path toward the target.
- A careful choice of value for δ in RecLM with nonadaptive $\delta(t)$ can improve the convergence.

Fig. 7 depicts the convergence of $\hat{\theta}(t)$ over time.

- Clearly, RecSD converges the slowest and RecGN converges the fastest among the algorithms.
- RecLM with a constant $\delta = 5 \cdot 10^{-3}$ initially converges faster but finally converges slower than RecSD. This re-emphasizes that δ should not be fixed to a constant value.
- If δ is allowed to be adapted over time, a good choice for the value of $\delta(0)$ may not be crucial. This is shown by the RecLM that uses the adaptive $\delta(t)$ with $\delta(0) = 5 \cdot 10^{-3}$. It converges faster than the RecLM with a constant $\delta = 5 \cdot 10^{-3}$, and it has a similar fast convergence rate as that of RecGN.

In Case 2, RecLM begins with two different values of constant δ (i.e., $1 \cdot 10^{-4}$ and $1 \cdot 10^{-3}$). Fig. 8 shows the following results.

- In the beginning, RecSD has taken the same path as other algorithms but later descends quickly toward the target. Thus, RecSD will converge the fastest among the algorithms.

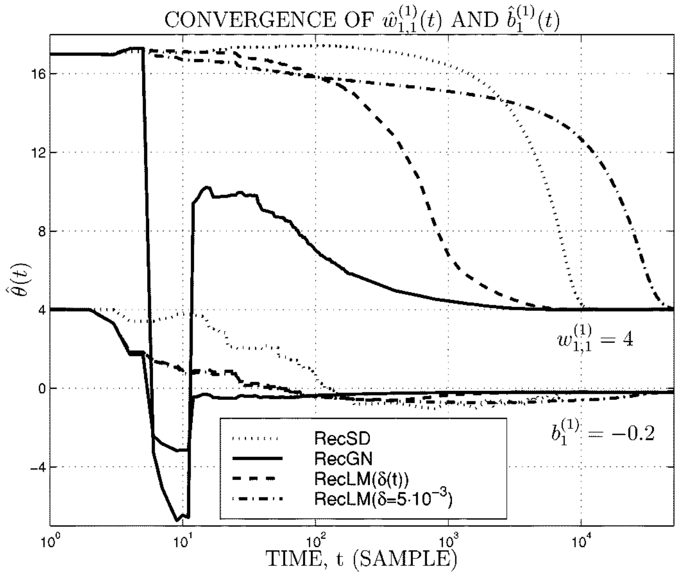

Fig. 7. Convergence of parameter estimate $\hat{\theta}(t)$ for Case 1 ($\hat{\theta}(0) = [17, 4]^T$).

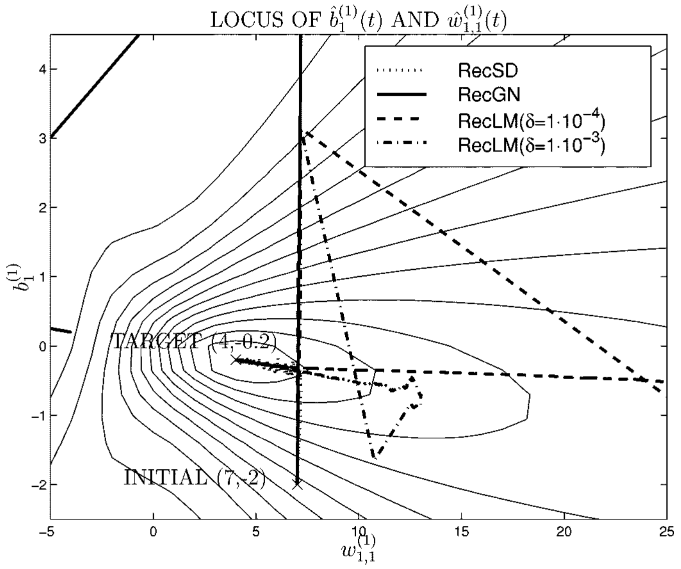

Fig. 8. Locus of parameter estimate $\hat{\theta}(t)$ on the error contour for Case2 ($\hat{\theta}(0) = [7, -2]^T$). Note that a small part of the locus for RecGN and RecLM ($\delta = 1 \cdot 10^{-4}$) is outside the figure.

- RecGN has taken few long steps in the unimportant direction of the parameters space and converged to a local minima.

Fig. 9 illustrates that a RecLM which uses the adaptive $\delta(t)$ with $\delta(0) = 1 \cdot 10^{-4}$ has similar fast convergence rate as RecSD. Note that the convergence of RecGN is not plotted in Fig. 9 because it converged to a bad local minima.

The results shown in Figs. 6–9 indicate that the recursive Levenberg–Marquardt algorithm is indeed a method in between the recursive steepest-descent and Gauss–Newton algorithms. These results are similar to the ones obtained by their corresponding off-line training algorithms. In addition, the adaptive $\delta(t)$ is preferable for obtaining fast convergence in the recursive Levenberg–Marquardt algorithm. In the next two examples, the recursive Levenberg–Marquardt algorithm will use only the adaptive $\delta(t)$.

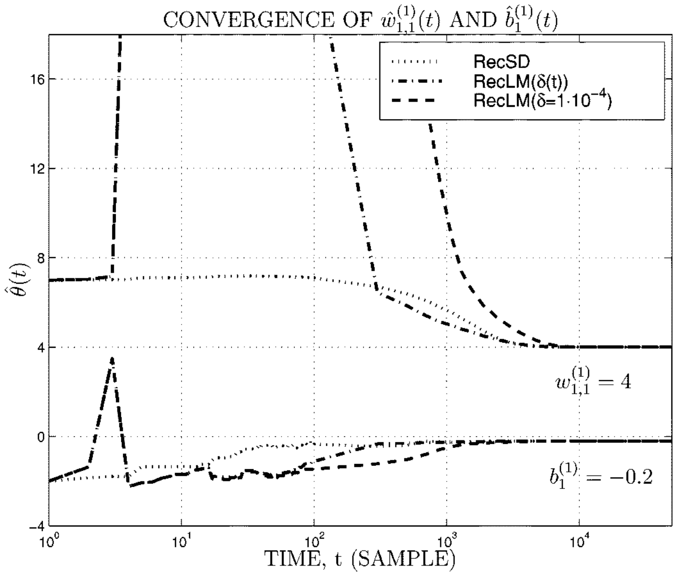

Fig. 9. Convergence of parameter estimate $\hat{\theta}(t)$ for Case2 ($\hat{\theta}(0) = [7, -2]^T$). Note that a small part of the convergence for RecLM ($\delta(t)$) and RecLM ($\delta = 1 \cdot 10^{-4}$) is outside the figure.

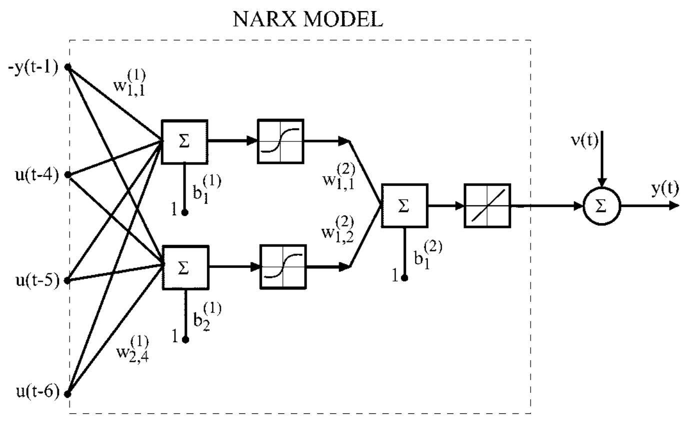

Fig. 10. Time-varying NARX system with two layers neural nets for Example 2.

B. Example 2—Influence of Design Variable $\lambda(t)$ in a Simulated Time-Varying System

This example illustrates the performance of the algorithms in a time-varying system. A NARX system is shown in Fig. 10. The input signal is a sequence of $1.5 \cdot 10^4$ samples of white noise with mean 0 and variance 1. The parameter θ at time $t = 0$ are chosen as uniformly distributed random values in $[-5, 5]$, which are tabulated in Table IV. At time $t = 5 \cdot 10^3$ samples, all parameters are changed by random values with uniform distribution in $[-0.5, 0.5]$. At $t = 1 \cdot 10^4$ samples, the parameters are changed to the original values at $t = 0$.

The algorithms use the same filter structure as the NARX model that is used to generate the output data, as depicted in Fig. 10. It has $n_a = 1$, $n_b = 3$, and $n_k = 4$ in the regressor $\varphi(t)$ of (3c), i.e., $\varphi(t) = [-y(t-1), u(t-4), u(t-5), u(t-6)]$. Table V shows the values of the design variables in the recursive algorithms. Note the weighting sequence $\beta(t, \tau)$, as plotted in Fig. 2, for the two cases in RecLM. The sensitivity of $\beta(t, \tau)$ with respect to λ_r is discussed in Section IV-A3. In this configuration, RecLM requires similar number of flops as in RecSD and 25% more flops than RecGN. The initial values of $\hat{\theta}(0)$ are the same in all algorithms, which are selected randomly with a uniform distribution in $[-1, 1]$.

TABLE IV
VALUES OF TRUE PARAMETER θ IN A TIME-VARYING SYSTEM FOR EXAMPLE 2

	$t = 0$	$t = 5 \cdot 10^3$	$1 \cdot 10^4$
$w^{(2)}_{1,1}$	-1.6480	-1.4472	-1.6480
$w^{(2)}_{1,2}$	1.5553	2.0176	1.5553
$b^{(2)}_1$	-1.0810	-0.8304	-1.0810
$w^{(1)}_{1,1}$	-1.3225	-1.1485	-1.3225
$w^{(1)}_{1,2}$	2.3128	2.7744	2.3128
$w^{(1)}_{1,3}$	4.0481	3.9084	4.0481
$w^{(1)}_{1,4}$	1.3179	1.0797	1.3179
$b^{(1)}_1$	0.4878	0.0371	0.4878
$w^{(1)}_{2,1}$	1.2080	1.7075	1.2080
$w^{(1)}_{2,2}$	-3.0611	-3.5022	-3.0611
$w^{(1)}_{2,3}$	0.6921	0.7406	0.6921
$w^{(1)}_{2,4}$	-2.6559	-2.5585	-2.6559
$b^{(1)}_2$	4.3158	4.3869	4.3158

TABLE V
VALUES OF THE DESIGN VARIABLES IN EACH ALGORITHM FOR EXAMPLE 2

RecSD	RecGN	RecLM
$N_c = 4$	$P(0) = 1 \cdot 10^4$	$P(0) = 1 \cdot 10^4$
$K_n = 6$	$\lambda = 0.995$	$\lambda(0) = 0.995$
$\mu = 0.5$		$\lambda_r = 1$ (Case a)
		$\lambda_r = 0.999$ (Case b)
		$\delta(t)$ with $\delta(0) = 10^{-5}$

Fig. 11 exhibits the convergence of the normalized mean square error (NMSE) of the algorithms, which is defined as

$$\text{NMSE(dB)} = 10 \log_{10} \frac{E[\varepsilon(t, \theta)^2]}{E[y(t)^2]}. \tag{41}$$

The NMSE(dB) is approximated by smoothing $y(t)^2$ and $\varepsilon(t, \theta)^2$ using a first-order filter with a decay factor of 0.95.

Fig. 11 illustrates the following points.

- RecSD converges slower than RecGN and RecLM (Case a) at every time.
- RecGN converges slower than RecLM (Case a) between $t = 0$ to $t = 5 \cdot 10^3$ samples, although they have the same discounting effects, i.e., $\lambda(t)$ in RecLM (Case a) has the same constant 0.995 as λ in RecGN.
- RecGN and RecLM have an almost similar convergence rate after $t = 5 \cdot 10^3$ samples because they track equally well at $t = 5 \cdot 10^3$ and $t = 1 \cdot 10^4$ samples.
- RecLM (Case b) converges slower and tracks poorer than RecLM (Case a). This is quite natural because the estimates of the variables in the gradient are rather poor

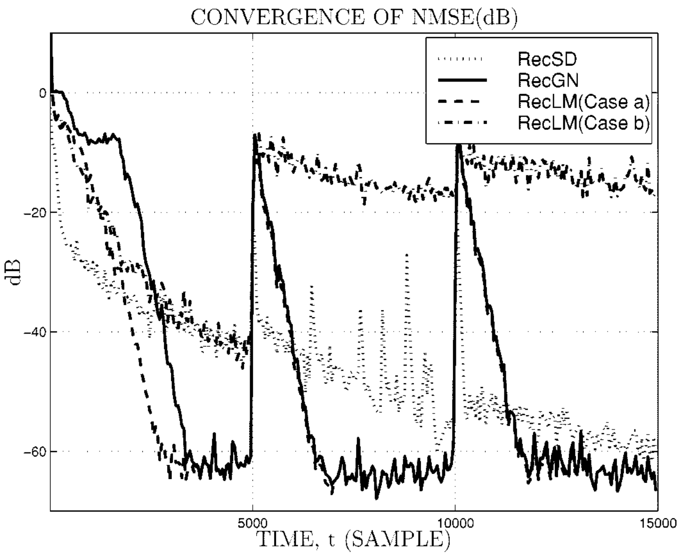

Fig. 11. Convergence of normalized mean square error (NMSE) in decibels. The two different cases of RecLM corresponds to $\lambda(0) = 0.995$ and $\lambda_r = 1$ (Case a) and 0.999 (Case b).

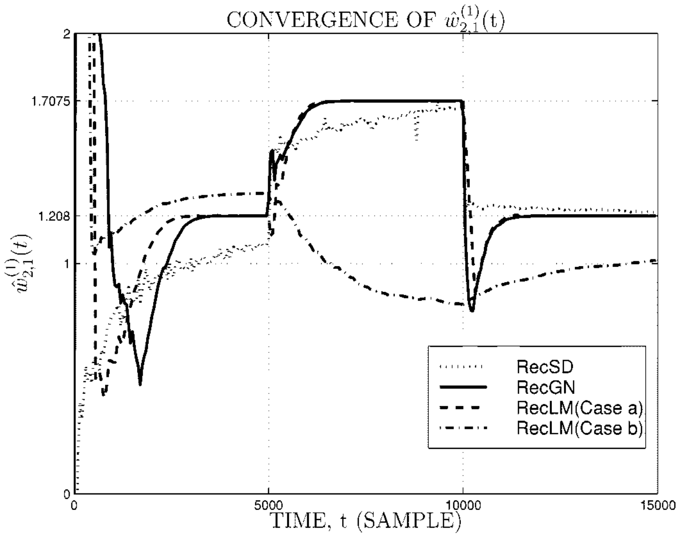

Fig. 12. Convergence of $\hat{w}^{(1)}_{2,1}(t)$ with its true value $w^{(1)}_{2,1}$, as shown in Table IV.

in the transient phase. RecLM (Case a) discounts more of these poor estimates and thus converges faster than RecLM (Case b). RecLM (Case a) also discounts more old data, and thus, it is more alert to parameter changes than RecLM (Case b). The differences in the convergence performance are quite significant between both cases, although they differ only by 0.001 in λ_r. This is due to the sensitivity of $\beta(t, \tau)$ to λ_r. Note that if RecGN uses the same $\lambda(t)$ as in RecLM (Case b), it will have similar slow convergence and poor tracking as in RecLM (Case b); this is not shown in Fig. 11.

A similar result is shown in Fig. 12, which illustrates the convergence of one of the parameter estimates $\hat{w}^{(1)}_{2,1}(t)$.

- Among the algorithms, RecLM (Case a) converges the fastest to the true parameter $w^{(1)}_{2,1} = 1.2080$ during the transient phase between $t = 0$ to $t = 5 \cdot 10^3$ samples.

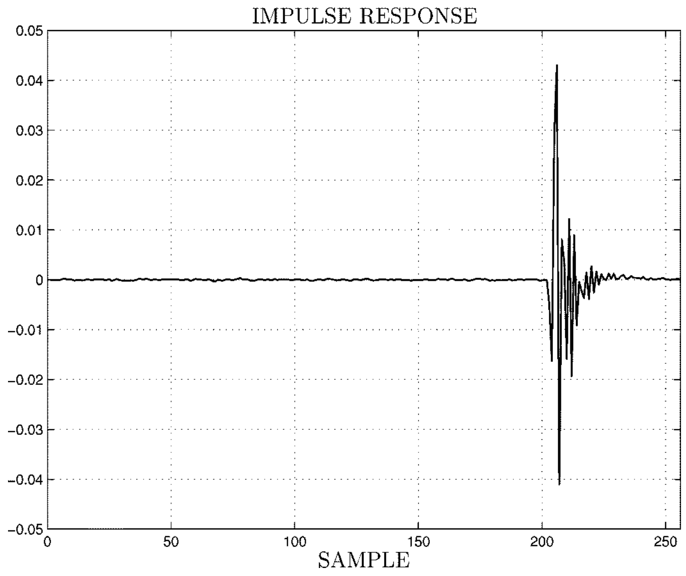

Fig. 13. Estimated impulse response of the network echo channel.

- RecGN and RecLM (Case a) can track the changes of $w^{(1)}_{2,1}$ at $t = 5 \cdot 10^3$ and $t = 1 \cdot 10^4$ samples better than RecSD and RecLM (Case b).

The results depicted in Figs. 11 and 12 imply that the recursive Levenberg–Marquardt algorithm with the adaptive $\delta(t)$ and a correct choice of $\lambda(t)$ has better convergence and tracking properties than the recursive steepest-descent and Gauss–Newton algorithms.

C. Example 3—Echo Cancellation Performance in a Real Network Echo Channel

The data used in the example is an echo of a telephone network channel, which is recorded at a mobile switching center in Sweden. Network echo is a phenomenon in which a delayed and distorted version of an original signal is reflected back to the source due to the impedance mismatch at the hybrid of a local telephone exchange. In practice, the network echo is only nearly linear. The input signal is a white noise sequence, and the sampling rate is 8 kHz.

The algorithms RecSD, RecGN, and RecLM are applied to a NFIR filter structure, which uses the regressor $\varphi(t)$ of (3a). To demonstrate the degree of nonlinearity in the network echo channel, a linear FIR filter is also compared. It has a linear regression structure as in (4) and uses the same $\varphi(t)$ of (3a). Its filter parameters are estimated using the NLMS algorithm[3] with $\mu = 0.5$, which is similar to (24) and (25), except that $\psi(t) = \varphi(t)$ in this linear case. An earlier analysis on the recorded data shows that the network echo channel has an estimated impulse response, as shown in Fig. 13. Thus, the chosen values are 203 for n_k and 25 for n_b in the regressor $\varphi(t)$, which are sufficient to represent the energy contained in the true impulse response of the echo channel.

The values of design variables in RecSD, RecGN, and RecLM are shown in Table VI. They use one hidden layer with five neurons as a trade-off between the achieved ERLE and

[3]Analysis shows that the performances of the NLMS and RLS algorithms on the linear FIR filter are almost similar.

TABLE VI
VALUES OF DESIGN VARIABLES IN EACH ALGORITHM FOR EXAMPLE 3

RecSD	RecGN	RecLM
$N_c = 1$	$P(0) = 1 \cdot 10^4$	$P(0) = 1 \cdot 10^4$
$K_n = 1$	$\lambda = 0.999$	$\lambda(0) = 0.999$
$\mu = 0.5$		$\lambda_r = 1$

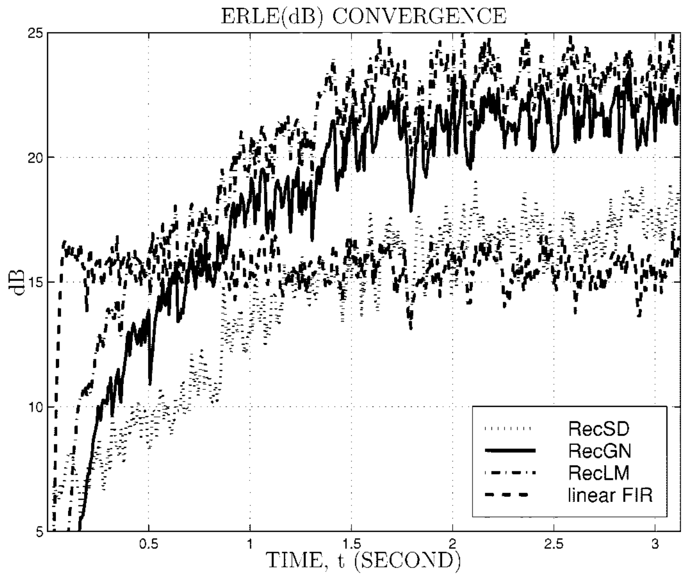

Fig. 14. Comparison of ERLE convergence. Note that the linear FIR filter structure is estimated by the NLMS algorithm with $\mu = 0.5$.

computational complexity. The resulting number of parameters is $d = 136$. In this configuration, the computational complexity of RecLM is similar to RecSD, and RecLM needs 15% more flops than RecGN. The linear FIR filter has the lowest number of parameters (with $d = 25$) to match the predetermined value of n_b, after considering that the input signal is delayed by n_k. All algorithms have the same initial parameter estimate $\hat{\theta}(0)$, which are chosen to be uniformly distributed random values in $[-1, 1]$. Note that the linear FIR filter uses only the first 25 out of 136 initial estimates in $\hat{\theta}(0)$.

The four different adaptive filters operate as echo cancelers to mimic the transfer function of the feedback coupling path. The figure of merit for the effectiveness of an echo canceler is the echo return loss enhancement (ERLE), which is defined as

$$\text{ERLE(dB)} = -\text{NMSE(dB)}. \tag{42}$$

Fig. 14 shows the following main results.

- Among the algorithms, the linear FIR filter has the lowest ERLE because the echo path is nonlinear, and it converges the fastest because it uses the least number of parameters.
- Compared with the linear filter, RecSD has the least ERLE gain of ≈ 1.5 dB, and RecLM has the highest ERLE gain of ≈ 8 dB.
- RecLM converges ≈ 0.3 s faster and has an ERLE of ≈ 2.5 dB higher than RecGN.

To an extent, the results show the superior efficiency of the recursive Levenberg–Marquardt algorithm.

VI. CONCLUSION

The Levenberg–Marquardt numerical search is known to be more efficient than the steepest-descent and Gauss–Newton methods in the off-line training of neural nets. This paper proposes a recursive Levenberg–Marquardt algorithm for on-line training of neural nets for nonlinear adaptive filtering.

The recursive Levenberg–Marquardt algorithm is compared with the recursive steepest-descent and Gauss–Newton algorithms in three system identification examples. All algorithms use the same filter structures in each example. The results indicate that the recursive Levenberg–Marquardt algorithm has similar advantages as its corresponding off-line algorithm. It is an efficient algorithm for adapting nonlinear systems in both simulated and real practical data.

Although the recursive Levenberg–Marquardt algorithm consumes a higher number of flops than the recursive steepest-descent and Gauss–Newton algorithms, it is compensated abundantly by its increased efficiency. The high efficiency is also unmatched by the recursive steepest-descent algorithm when its computational complexity is allowed to be equal to the one of the recursive Levenberg–Marquardt algorithm by increasing the length of data window N_c and the iteration of parameter update at each time t, K_n.

APPENDIX A
DESCRIPTION OF THE SISO NARX MODEL

Fig. 1 illustrates the nonlinear ARX model with $M-1$ hidden layers of sigmoidal neuron units as the nonlinear dynamics. The net input to unit i at layer $l = 1, \cdots, M$ is

$$net_i^{(l)} = \sum_{p=1}^{L_{l-1}} w_{i,p}^{(l)} x_p^{(l-1)} + b_i^{(l)}, \qquad i = 1, \cdots, L_l \tag{43}$$

where

L_l	number of units at layer l;
$w_{i,p}^{(l)}$	weight term;
$b_i^{(l)}$	bias term.

The parameter vector θ consists of all the weights and bias terms

$$\theta = \Big[w_{1,1}^{(M)}, \cdots, w_{1,L_{M-1}}^{(M)}, b_1^{(M)}, \cdots, w_{1,1}^{(1)}, \cdots \\ w_{1,L_0}^{(1)}, b_1^{(1)}, \cdots, w_{L_1,1}^{(1)}, \cdots, w_{L_1,L_0}^{(1)}, b_{L_1}^{(1)}\Big]^T \tag{44}$$

with dimension $d = \sum_{l=0}^{M-1} L_{l+1}(L_l + 1)$. The input–output relationship of the unit i at layer l is

$$x_i^{(l)} = g_i^{(l)}\left(net_i^{(l)}\right) \tag{45}$$

where $g_i^{(l)}(.)$ is a nonlinear function of the unit. At layer 0, $[x_1^{(0)}, \cdots, x_{L_0}^{(0)}]^T$ is equivalent to $\varphi(t)$, and at layer M, $x_1^{(M)}$ is equivalent to $\hat{y}(t, \theta)$.

APPENDIX B
DERIVATION OF GRADIENT VECTOR $\psi(t)$

Consider the neural nets filter structure as shown in Fig. 1. The gradient vector $\psi(t, \theta)$ in (11) is reproduced as

$$\psi(t, \theta) = \frac{d}{d\theta} g(t, \theta) \tag{46}$$

where θ is defined in (44). Thus, (46) can be derived if the gradients for the weight and bias terms are solved.

Define a backpropagated term

$$G_i^{(l)} = \frac{\partial g(.,.)}{\partial net_i^{(l)}} \tag{47}$$

where $net_i^{(l)}$ is defined in (43). Using the chain rule of calculus, the gradients for the weight terms are

$$\frac{dg(.,.)}{dw_{i,p}^{(l)}} = \frac{\partial g(.,.)}{\partial net_i^{(l)}} \frac{dnet_i^{(l)}}{dw_{i,p}^{(l)}} = G_i^{(l)} x_p^{(l-1)} \tag{48}$$

and the bias terms are

$$\frac{dg(.,.)}{db_i^{(l)}} = \frac{\partial g(.,.)}{\partial net_i^{(l)}} \frac{dnet_i^{(l)}}{db_i^{(l)}} = G_i^{(l)} \tag{49}$$

where $x_p^{(l-1)}$ is the input–output relationship of the unit p at layer $l-1$, which can be defined in (45).

At the output layer M, (48) and (49) become

$$\frac{dg(.,.)}{dw_{i,p}^{(M)}} = G_i^{(M)} x_p^{(M-1)} \tag{50}$$

$$\frac{dg(.,.)}{db_i^{(M)}} = G_i^{(M)} \tag{51}$$

where $i = 1$ at layer M for the SISO case.

At a hidden layer, i.e., $1 \le l \le M-1$, (47) can be rewritten using the chain rule of calculus as

$$G_i^{(l)} = \left(\sum_{k=1}^{L_{l+1}} \frac{\partial g(.,.)}{\partial net_k^{(l+1)}} \frac{dnet_k^{(l+1)}}{dx_i^{(l)}} \right) \frac{\partial x_i^{(l)}}{\partial net_i^{(l)}} \tag{52}$$

and it can be easily shown that (52) equals

$$G_i^{(l)} = \left(\sum_{k=1}^{L_{l+1}} G_k^{(l+1)} w_{k,i}^{(l+1)} \right) \frac{\partial g_i^{(l)}(net_i^{(l)})}{\partial net_i^{(l)}}. \tag{53}$$

Now, the backpropagated term $G_i^{(l)}$ can be iteratively computed, starting from $G_i^{(M)}$ at the output layer. Thus, (48) and (49) are solved for $1 \le l \le M-1$ using $G_i^{(l)}$ in (53).

APPENDIX C
RECURSIVE LEVENBERG–MARQUARDT ALGORITHM

Here is the summary of the proposed algorithms as described in (34)–(40).

$$\varepsilon(t) = y(t) - \hat{y}(t, \theta) \tag{54a}$$

$$\hat{\theta}(t+1) = \hat{\theta}(t) + P(t)\psi(t, \theta)\varepsilon(t, \theta) \tag{54b}$$

$$P(t) = \{P(t-1) - P(t-1)\psi^*(t, \theta) S(t)^{-1}\psi^*(t, \theta)^T P(t-1)\}/\lambda(t) \tag{54c}$$

$$S(t) = \psi^*(t, \theta)^T P(t-1)\psi^*(t, \theta) + \lambda(t)\Lambda^*(t) \tag{54d}$$

where $\hat{y}(t, \theta)$ is the estimated output, the gradient vector $\psi(t, \theta)$ is

$$\psi(t, \theta) = \frac{d}{d\theta} g(t, \theta) \tag{55}$$

the modified gradient vector is

$$\psi^*(t, \theta) = \begin{pmatrix} & & \psi(t, \theta)^T & & \\ 0 & \cdots & 0 & 1 & \cdots & 0 \end{pmatrix}^T$$
$$\Uparrow$$
$$\text{position} = (t \bmod d) + 1 \tag{56}$$

the modified weighting matrix is

$$\Lambda^*(t)^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \delta(t)d \end{pmatrix} \tag{57}$$

the instantaneous forgetting factor is

$$\lambda(t) = \lambda_r \lambda(t-1) + (1 - \lambda_r) \tag{58}$$

and the design variable $\delta(t)$ is adapted once after every d time samples to ensure that the error criterion $V_t(\theta)$ in (17) is decreased by using

$$r_a(t) = \frac{1}{2d} \left(\sum_{\tau=t-2d+1}^{t-d} \varepsilon^2(\tau, \hat{\theta}(\tau)) - \sum_{\tau=t-d+1}^{t} \varepsilon^2(\tau, \hat{\theta}(\tau)) \right) \tag{59a}$$

$$r_p(t) = [\psi(t, \theta)\varepsilon(t, \theta)]^T P(t)\psi(t, \theta)\varepsilon(t, \theta) \tag{59b}$$

$$\delta(t) = \begin{cases} \kappa\delta(t-1), & \text{if } r_a(t) < \zeta r_p(t) \\ \delta(t-1)/\kappa, & \text{if } r_a(t) > (1-\zeta) r_p(t) \\ \delta(t-1), & \text{otherwise.} \end{cases} \tag{59c}$$

Choice of values for the design variables $\hat{\theta}(0)$, $P(0)$, $\lambda(0)$, $\delta(0)$, ζ and κ are described in Section IV.

ACKNOWLEDGMENT

The authors thank Ericsson Radio Systems AB, Sweden, for the kind contribution of telephone network echo data.

REFERENCES

[1] G. Cybenko, "Approximations by superposition of a sigmoidal function," *Math. Contr., Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[2] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Comput.*, vol. 2, no. 2, pp. 210–215, Mar. 1990.

[3] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990.

[4] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, no. 2, pp. 246–257, 1991.

[5] E. Barnard, "Optimization for training neural nets," *IEEE Trans. Neural Networks*, vol. 3, pp. 232–240, Mar. 1992.

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

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 32, no. 9, pp. 533–536, Oct. 1986.

[8] P. P. van der Smagt, "Minimization methods for training feedforward neural networks," *Neural Networks*, vol. 7, no. 1, pp. 1–11, 1994.

[9] A. N. Birkett and R. A. Goubran, "Fast nonlinear adaptive filtering using a partial window conjugate gradient algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1996, pp. 3541–3544.

[10] S. C. Douglas and T. H. Meng, "Linearized least-squares training of multilayer feedforward neural networks," in *Proc. Int. Joint Conf. Neural Networks*, 1991, pp. 307–312.

[11] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Drefys, and S. Marcos, "Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms," *Neural Comput.*, vol. 5, no. 2, pp. 165–199, Mar. 1993.

[12] L. Yin, J. Astola, and Y. Neuvo, "A new class of nonlinear filters-neural filters," *IEEE Trans. Signal Processing*, vol. 41, pp. 1201–22, Mar. 1993.

[13] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[14] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*. Cambridge, MA: MIT Press, 1983.

[15] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[16] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[17] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[18] Y. Iiguni and S. Sakai, "A real time learning algorithm for a multilayered neural network based on the extended Kalman filter," *IEEE Trans. Signal Processing*, vol. 40, pp. 959–966, Apr. 1992.

[19] H. Kinjo, S. Tamaki, and T. Yamamoto, "Linearized adaptive filter for a nonlinear system using neural network based on the extended Kalman filter," *Trans. IEE Jpn.*, vol. 117, no. 3, pp. 279–286, Mar. 1997.

[20] M. B. Matthews and G. S. Moschytz, "Neural-network nonlinear adaptive filtering using the extended Kalman filter algorithm," in *Proc. Int. Neural Networks Conf.*, 1990, pp. 115–118.

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

[22] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, Dec. 1995.

[23] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[24] R. Fletcher, *Practical Methods of Optimization*. Wiltshire, U.K.: Wiley, 1997.

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

[26] Q. J. Zhang, Y. J. Zhang, and W. Ye, "Local-sparse connection multilayer networks," in *Proc. IEEE Int. Conf. Neural Networks*, 1995, pp. 1254–1257.

[27] J. Sjöberg and M. Viberg, "Separable nonlinear least-squares minimization—Possible improvements for neural net fitting," in *Proc. IEEE Workshop Neural Networks Signal Process.*, 1997, pp. 345–354.

[28] T. Söderström, "An on-line algorithm for approximate maximum likelihood identification of linear dynamic systems," Lund Inst. Technol., Lund, Sweden, Tech. Rep. 7308, 1973.

[29] S. Gunnarsson, "On covariance modification and regularization in recursive least squares identification," in *Proc. IFAC Symp. Syst. Ident.*, 1995, pp. 935–940.

[30] J. J. Moré, "The Levenberg–Marquardt algorithm: Implementation and theory," in *Numerical Analysis*, ser. Springer Lecture Notes in Mathematics, G. A. Watson, Ed. Berlin, Germany, 1977, vol. 630, pp. 105–116.

[31] M. D. Hebden, "An algorithm for minimization using exact second derivatives," Atomic Energy Res. Establish., Harwell, U.K., Tech. Rep. TP515, 1973.

Lester S. H. Ngia (S'97) was born in Malaysia in 1965. He received the B.E.E. degree from the University of Technology of Malaysia, Kuala Lumpur, in 1988, the M.B.A. degree from the Edinburgh Business School, Heriot-Watt University, Edinburgh, U.K., in 1995, and the M.Sc.E.E. degree from Chalmers University of Technology (CTH), Gothenburg, Sweden in 1997. Since February 1997, he has been working toward the Ph.D. degree in electrical engineering at CTH.

He was with Intel Corporation in Malaysia as a Product Engineer from 1988 to 1992 and a Senior Product, Quality and Reliability Engineer from 1992 to 1995, working in the area of testing and reliability for speech and data communication VLSI products. Since February 1997, he has been a Teaching and Research Assistant with the Department of Signals and Systems, CTH. His research interests include mathematical modeling of linear and nonlinear dynamic systems, with applications on network and acoustic echo cancellation problems.

Jonas Sjöberg (M'97) was born in Sweden. He received the M.Sc. degree from the Uppsala University, Uppsala, Sweden, in 1989 and the Ph.D. degree in electrical engineering from Linköping University, Linköping, Sweden, in May 1995.

He was a Postdoctoral Fellow at the Swiss Federal Institute of Technology, Zürich, and at the Vienna University of Technology, Vienna, Austria. He was a Visiting Researcher at the Technion—Israel Institute of Technology, Haifa, in 1998. Since 1996, he has been an Assistant Professor with the Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden. His current research interests include algorithms and applications of system identification, especially on nonlinear systems. He is currently an Associate Editor of *Control Engineering Practice*.