International Journal of Control
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713393989

Parallel recursive prediction error algorithm for training layered neural networks

S. Chen [a]; C. F. N. Cowan [a]; S. A. Billings [b]; P. M. Grant [a]
[a] Department of Electrical Engineering, University of Edinburgh, Edinburgh, Scotland, U.K.
[b] Department of Control Engineering, University of Sheffield, Sheffield, England, U.K

Online Publication Date: 01 January 1990
To cite this Article: Chen, S., Cowan, C. F. N., Billings, S. A. and Grant, P. M. (1990) 'Parallel recursive prediction error algorithm for training layered neural networks', International Journal of Control, 51:6, 1215 - 1228
To link to this article: DOI: 10.1080/00207179008934127
URL: http://dx.doi.org/10.1080/00207179008934127

INT. J. CONTROL, 1990, VOL. 51, NO. 6, 1215–1228

Parallel recursive prediction error algorithm for training layered neural networks

S. CHEN†, C. F. N. COWAN†, S. A. BILLINGS‡ and P. M. GRANT†

A new recursive prediction error algorithm is derived for the training of feedforward layered neural networks. The algorithm enables the weights in each neuron of the network to be updated in an efficient parallel manner and has better convergence properties than the classical back propagation algorithm. The relationship between this new parallel algorithm and other existing learning algorithms is discussed. Examples taken from the fields of communication channel equalization and non-linear systems modelling are used to demonstrate the superior performance of the new algorithm compared with the back propagation routine.

1. Introduction

Multi-layer neural networks are usually trained to perform a particular task by the back propagation algorithm (Rumelhart *et al.* 1986) which is a simple version of the smoothed stochastic gradient algorithm. This type of algorithm uses the negative gradient of some chosen criterion as the search direction and, because the computation can be distributed to each weight in the network, the algorithm is coherent with the massively parallel nature of the network. It is well-known however that this type of algorithm suffers from the drawback of slow convergence.

More recent work has produced improved learning strategies based on an extended Kalman algorithm (Singhal and Wu 1989) and a recursive prediction error routine (Chen *et al.* 1989 b). Although these two algorithms were each derived independently based on a different approach they are actually equivalent. They both use the same search direction called the Gauss–Newton direction, for which the negative gradient is multiplied by the inverse of an approximate hessian matrix of the given criterion. This is a more efficient search direction than the steepest-descent approach of back propagation and it significantly improves the convergence performance. It does however also increase the computational complexity and the weight updating requires a centralized computing facility with the effect that the parallel structure of the network is not exploited.

It is known from non-linear optimization theory that the inverse hessian is not the only matrix that can be used to modify the negative gradient. This result is exploited in the present study, where a near-diagonal matrix is carefully chosen to modify the negative gradient direction. The advantage of this approach is that the updating of this matrix can be decomposed such that the computation can be distributed to each neuron in the network. This results in a new parallel recursive prediction error algorithm which utilizes the parallel computing potential of the network and which

Received 17 October 1989.

† Department of Electrical Engineering, University of Edinburgh, Mayfield Road, Edinburgh EH9 3JL, Scotland, U.K.

‡ Department of Control Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, England, U.K.

0020-7179/90 $3.00

also cuts the computational requirement to a small proportion of that needed by the conventional recursive prediction error algorithm or the extended Kalman algorithm. Whilst the idea used in the derivation of this learning algorithm is similar to that employed by Kollias and Anastassiou (1989) the parallel algorithm introduced in the present study is quite different because it is a truly recursive algorithm. The training of neural networks as adaptive channel equalizers and the recursive identification of non-linear systems based on a neural network model are used as examples to compare the convergence performance of the new learning algorithm with the back propagation routine.

2. Layered neural networks

The neural networks considered in this paper are feedforward type networks with one or more hidden layers between the inputs and outputs. Each layer consists of some computing units known as neurons. Figure 1 shows the structure of a multi-layered neural network. Inputs to the network are passed to each neuron in the first layer. The outputs of the first layer neurons then become inputs to the second layer, and so on. The outputs of the network are therefore the outputs of the neurons lying in the final layer. Usually, all the neurons in a layer are fully connected to the neurons in adjacent layers but there is no connection between neurons within a layer and no connection bridging layers. The input–output relationship of each neuron is determined by connection weights w_i, a threshold parameter μ and the neural activation function $a(\cdot)$ such that

$$y = a\left(\sum w_i x_i + \mu\right) \tag{1}$$

where x_i are the neural inputs and y is the neural output. Two typical examples of the neural activation function are

$$a(y) = \tanh\left(\frac{y}{2}\right) = \frac{1-\exp(-y)}{1+\exp(-y)} \tag{2}$$

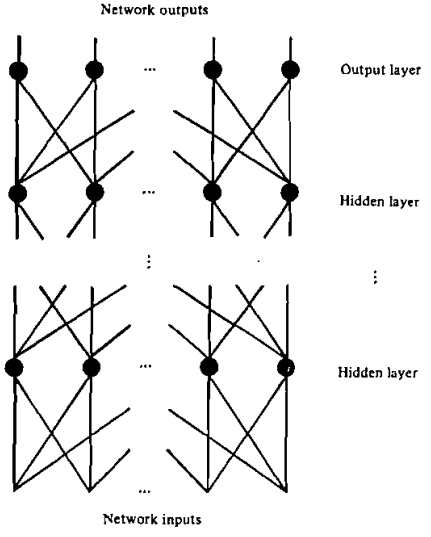

Figure 1. Multi-layered neural network.

and

$$a(y) = \frac{1}{1 + \exp(-y)} \tag{3}$$

Assume that all the weights and thresholds of the network are ordered in an n_Θ-dimensional vector

$$\Theta = [\theta_1 \quad \dots \quad \theta_{n_\Theta}]^T \tag{4}$$

The overall input–output relationship of an m-input r-output network can be described by the following non-linear relationship:

$$\hat{y}(t, \Theta) = f(v(t); \Theta) \tag{5}$$

where $\hat{y}(t, \Theta)$ is the r-dimensional network output vector, $v(t)$ is the m-dimensional network input vector and $f(\cdot)$ is a vector valued non-linear function. Training a multi-layer neural network to perform a given task involves supplying the network with an input sequence $\{v(t)\}$ and determining Θ so that the network output sequence $\{\hat{y}(t, \Theta)\}$ approximates a desired sequence $\{d(t)\}$. In the terminology of system identification, the discrepancy between $d(t)$ and $\hat{y}(t, \Theta)$

$$\varepsilon(t, \Theta) = d(t) - \hat{y}(t, \Theta) \tag{6}$$

is called the prediction error. The gradient of $\hat{y}(t, \Theta)$ with respect to Θ is the $n_\Theta \times r$ matrix

$$\Psi(t, \Theta) = \left[\frac{d\hat{y}(t, \Theta)}{d\Theta}\right]^T = g(v(t); \Theta) \tag{7}$$

which plays an important role in determining Θ. The combination of (5) and (7):

$$\begin{bmatrix} \hat{y}(t, \Theta) \\ \Psi(t, \Theta) \end{bmatrix} = \begin{bmatrix} f(v(t); \Theta) \\ g(v(t); \Theta) \end{bmatrix} \tag{8}$$

will be referred to as the extended network model. When Θ is partitioned in the form of (4), $\Psi(t, \Theta)$ can accordingly be written as

$$\Psi(t, \Theta) = \begin{bmatrix} \psi_1(t, \Theta) \\ \vdots \\ \psi_n(t, \Theta) \end{bmatrix} \tag{9}$$

where $\psi_i(t, \Theta)$, a $1 \times r$ row vector, is the gradient of $\hat{y}(t, \Theta)$ with respect to θ_i and $i = 1, \dots, n_\Theta$.

For notational convenience, a different partition of Θ and $\Psi(t, \Theta)$ is also introduced. Assume that neurons in the network are numbered from 1 to p and the weights and threshold of the ith neuron are arranged in an n_{Θ_i}-dimensional vector Θ_i, $i = 1, \dots, p$. Θ and $\Psi(t, \Theta)$ can then be represented as

$$\Theta = \begin{bmatrix} \Theta_1 \\ \vdots \\ \Theta_p \end{bmatrix}, \quad \Psi(t, \Theta) = \begin{bmatrix} \Psi_1(t, \Theta) \\ \vdots \\ \Psi_p(t, \Theta) \end{bmatrix} \tag{10}$$

where $\Psi_i(t, \Theta)$, an $n_{\Theta_i} \times r$ matrix, is the gradient of $\hat{y}(t, \Theta)$ with respect to Θ_i and $i = 1, \ldots, p$.

3. Batch prediction error algorithms

A good measure of the closeness between $d(t)$ and $\hat{y}(t, \Theta)$ is the quadratic form of $\varepsilon(t, \Theta)$. Assume that a block of data $\{v(t), d(t)\}_{t=1}^{N}$ is available. The best Θ may then be selected by minimizing the loss function

$$J(\Theta) = \frac{1}{2N} \sum_{t=1}^{N} \varepsilon^T(t, \Theta)\varepsilon(t, \Theta) \tag{11}$$

Such a method of obtaining a desired Θ is known as the prediction error estimation method in systems identification (Goodwin and Payne 1977, Ljung and Söderström 1983). Minimization of (11) is usually achieved iteratively according to

$$\Theta^{(k)} = \Theta^{(k-1)} + \alpha \Xi(\Theta^{(k-1)}) \tag{12}$$

where the superscript (k) denotes the iteration step in the minimization procedure, $\Xi(\Theta^{(k-1)})$ is a search direction based on information about $J(\Theta)$ acquired at a previous iteration, and α is a positive constant which is appropriately chosen to guarantee convergence of the iterative procedure.

3.1. *Steepest-descent algorithm*

The simplest search direction $\Xi(\Theta)$ is the negative gradient of the criterion (11) with respect to Θ, that is

$$\Xi(\Theta) = -\nabla J(\Theta) = \frac{1}{N} \sum_{t=1}^{N} \Psi(t, \Theta)\varepsilon(t, \Theta) \tag{13}$$

This algorithm, often called the steepest-descent algorithm, is known to converge at least to a local minimum of the loss function (11) but will normally exhibit a fairly slow convergence rate. For a multi-layer neural network, however, choosing such a gradient direction has an advantage. The algorithm can be integrated into the parallel structure of the network by decomposing the vector equation (12) into n_Θ scalar equations

$$\theta_i^{(k)} = \theta_i^{(k-1)} + \alpha \xi_i(\Theta^{(k-1)}) \quad i = 1, \ldots, n_\Theta \tag{14}$$

with

$$\xi_i(\Theta) = \frac{1}{N} \sum_{t=1}^{N} \psi_i(t, \Theta)\varepsilon(t, \Theta) \quad i = 1, \ldots, n_\Theta \tag{15}$$

3.2. *Gauss–Newton algorithm*

To improve the efficiency of the minimization procedure, a common strategy is to modify the negative gradient direction with some positive definite $n_\Theta \times n_\Theta$ matrix $M(\Theta)$ and this gives rise to a general search direction

$$\Xi(\Theta) = M(\Theta)(-\nabla J(\Theta)) \tag{16}$$

Different choices of $M(\Theta)$ will result in algorithms with different convergence rates. A very efficient algorithm called the Gauss–Newton algorithm is obtained by choosing

$M(\Theta)$ as the inverse of the approximate hessian of (11), that is

$$M(\Theta) = H^{-1}(\Theta) \tag{17}$$

where

$$H(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Psi^T(t, \Theta) \tag{18}$$

In case $H(\Theta)$ is near singular, (18) can be modified to

$$H(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Psi^T(t, \Theta) + \rho I \quad \rho > 0 \tag{19}$$

for some small value of ρ, where I is the identity matrix with appropriate dimension. The Gauss–Newton procedure, although very powerful, requires much more computation and can only be implemented with a centralized structure when applied to neural networks.

3.3. *Parallel prediction error algorithm*

If however $M^{-1}(\Theta)$ is chosen to be the following near-diagonal matrix $\bar{H}(\Theta)$:

$$\bar{H}(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \begin{bmatrix} \Psi_1(t, \Theta)\Psi_1^T(t, \Theta) & 0 & \dots & 0 \\ 0 & \Psi_2(t, \Theta)\Psi_2^T(t, \Theta) & & \vdots \\ \vdots & & & 0 \\ 0 & \dots & 0 & \Psi_p(t, \Theta)\Psi_p^T(t, \Theta) \end{bmatrix} \tag{20}$$

the search direction $\Xi(\Theta)$ can be decomposed into p smaller vectors:

$$\Xi_i(\Theta) = \left[\frac{1}{N}\sum_{t=1}^{N} \Psi_i(t, \Theta)\Psi_i^T(t, \Theta)\right]^{-1} \frac{1}{N}\sum_{t=1}^{N} \Psi_i(t, \Theta)\varepsilon(t, \Theta) \quad i = 1, \dots, p \tag{21}$$

The minimization algorithm (12) can thus be decomposed into p sub-algorithms:

$$\Theta_i^{(k)} = \Theta_i^{(k-1)} + \alpha\Xi_i(\Theta^{(k-1)}) \quad i = 1, \dots, p \tag{22}$$

and each of these sub-algorithms corresponds to a neuron in the network. ρI may also be added to $\bar{H}(\Theta)$ to guarantee positive definiteness without affecting the decomposition. The choice (20) forms a basis for the derivation of a new parallel recursive prediction error algorithm. $\bar{H}(\Theta)$ can be regarded as a simplified form of $H(\Theta)$ where all the blocks $\Psi_i(t, \Theta)\Psi_j^T(t, \Theta)$, $i \neq j$, in $H(\Theta)$ are substituted by zero blocks. The steepest-descent algorithm represents the extreme case where $H(\Theta)$ is simply replaced by the identity matrix.

4. Recursive prediction error algorithms

Recursive approximations of the prediction error method have been studied systematically (Ljung and Söderström 1983, Chen *et al.* 1990), and these results can be utilized here. Denote $\hat{\Theta}(t)$ as the estimate of Θ at t, introduce a time-varying version of the extended network model (8)

$$\begin{bmatrix} \hat{y}(t) \\ \Psi(t) \end{bmatrix} = \begin{bmatrix} f(v(t);\, \hat{\Theta}(t-1)) \\ g(v(t);\, \hat{\Theta}(t-1)) \end{bmatrix} \tag{23}$$

and define

$$\varepsilon(t) = d(t) - \hat{y}(t) \tag{24}$$

4.1. *Conventional recursive prediction error algorithm*

The conventional recursive prediction error algorithm can be summarized as follows (Chen *et al.* 1990):

$$\Delta(t) = \gamma_m \Delta(t-1) + \gamma_g \Psi(t)\varepsilon(t) \tag{25}$$

$$P(t) = \frac{1}{\lambda}\{P(t-1) - P(t-1)\Psi(t)[\lambda I + \Psi^T(t)P(t-1)\Psi(t)]^{-1}\Psi^T(t)P(t-1)\} \tag{26}$$

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + P(t)\Delta(t) \tag{27}$$

where γ_g and γ_m are the adaptive gain and momentum parameter respectively, and λ is the usual forgetting factor. $\Delta(t)$, an n_Θ-dimensional vector, can be viewed as a recursive form of the negative gradient $-\nabla J(\Theta)$ and is often called the smoothed stochastic gradient. If the momentum parameter γ_m is zero, it reduces to a stochastic gradient. $P(t)$, an $n_\Theta \times n_\Theta$ matrix, is a recursive approximation of the inverse hessian $H^{-1}(\Theta)$. The algorithm (25) to (27) can thus be regarded as a recursive Gauss–Newton algorithm.

$P(t)$ determines the asymptotical accuracy of the estimate and, therefore, is referred to as the covariance matrix (of the estimate). In adaptive applications, $0 < \lambda < 1$. If the covariance matrix $P(t)$ is implemented in its basic form as shown in (26), a phenomenon known as 'covariance wind-up' may occur under certain circumstances. That is, $P(t)$ may become explosive. Many numerical measures have been developed to overcome this problem (Salgado *et al.* 1988). A technique called constant trace adjustment is given here:

$$\left.\begin{aligned} \bar{P}(t) &= P(t-1) - P(t-1)\Psi(t)[\lambda I + \Psi^T(t)P(t-1)\Psi(t)]^{-1}\Psi^T(t)P(t-1) \\ P(t) &= \frac{K_0}{\text{trace}\,[\bar{P}(t)]}\bar{P}(t), \quad K_0 > 0 \end{aligned}\right\} \tag{28}$$

This will set an upper bound for the eigenvalues of $P(t)$ matrix.

It has been demonstrated (Chen *et al.* 1989 b), in the context of adaptive channel equalization using neural networks, that this recursive prediction error algorithm achieves far better convergence performance and is less sensitive to the initial values of network weights compared with the back propagation algorithm. The main drawbacks of the algorithm are that it can only be implemented in a centralized structure and the computational complexity of the algorithm may be a limitation in certain applications.

4.2. *Back propagation algorithm*

The widely used back propagation algorithm is represented by

$$\left.\begin{aligned} \Delta(t) &= \gamma_m \Delta(t-1) + \gamma_g \Psi(t)\varepsilon(t) \\ \hat{\Theta}(t) &= \hat{\Theta}(t-1) + \Delta(t) \end{aligned}\right\} \tag{29}$$

or, in the decomposed form,

$$\left.\begin{aligned}\delta_i(t) &= \gamma_m \delta_i(t-1) + \gamma_g \psi_i(t)\varepsilon(t)\\ \hat{\theta}_i(t) &= \hat{\theta}_i(t-1) + \delta_i(t)\end{aligned}\right\} \quad i = 1, \ldots, n_\Theta \tag{30}$$

The algorithm is clearly a recursive approximation of the steepest-descent algorithm. It can also be obtained from the recursive Gauss–Newton algorithm by replacing $P(t)$ in (27) by the identity matrix. The main strength of the algorithm lies in its computational simplicity and parallel structure. Slow convergence behaviour, on the other hand, is its well-known disadvantage.

4.3. *Parallel recursive prediction error algorithm*

Using the same procedure for deriving the conventional recursive prediction error algorithm, it is straightforward to obtain the following recursive approximation of the batch algorithm (21) and (22)

$$\left.\begin{aligned}\Delta_i(t) &= \gamma_m \Delta_i(t-1) + \gamma_g \Psi_i(t)\varepsilon(t)\\ \hat{\Theta}_i(t) &= \hat{\Theta}_i(t-1) + P_i(t)\Delta_i(t)\end{aligned}\right\} \quad i = 1, \ldots, p \tag{31}$$

where $\Delta_i(t)$ and $P_i(t)$ are the n_{Θ_i}-dimensional smoothed stochastic gradient and the $n_{\Theta_i} \times n_{\Theta_i}$ covariance matrix for the ith neuron, respectively. The formula for updating each $P_i(t)$ is identical with that used for $P(t)$. The technique of constant trace adjustment, for example, will give

$$\left.\begin{aligned}\bar{P}_i(t) &= P_i(t-1) - P_i(t-1)\Psi_i(t)\\ &\quad \times [\lambda I + \Psi_i^T(t)P_i(t-1)\Psi_i(t)]^{-1}\Psi_i^T(t)P_i(t-1)\\ P_i(t) &= \frac{K_0}{\operatorname{trace}[\bar{P}_i(t)]}\bar{P}_i(t), \quad K_0 > 0\end{aligned}\right\} \quad i = 1, \ldots, p \tag{32}$$

This algorithm may be viewed as simply applying the conventional recursive prediction error algorithm to each neuron in the network. It is obvious that the number of the elements in $P(t)$ and the number of all the elements in $P_i(t)$, $i = 1, \ldots, p$, satisfy

$$(n_\Theta)^2 = \left(\sum_{i=1}^{p} n_{\Theta_i}\right)^2 \gg \sum_{i=1}^{p} (n_{\Theta_i})^2 \tag{33}$$

The computational requirement of this new recursive algorithm is therefore only a small proportion of that needed for the conventional recursive prediction error algorithm. The main attraction of the new algorithm is that it can naturally be integrated into the parallel structure of the network and, like the back propagation algorithm, it performs two sweeps through the network at each recursion in an efficient parallel manner. The input signals are propagated forward on the first sweep and, as the signals pass through a layer in the network all the neurons in that layer compute their own outputs simultaneously. The error signal $\varepsilon(t)$ obtained at the top of the network is then propagated down the network on the second sweep. As this signal, accompanied by the gradient information, propagates back through a layer, all the neurons in that layer again update their own weights simultaneously.

This parallel learning algorithm is of course computationally more complex than the back propagation algorithm. It is likely, however, that the former will be more

efficient in terms of convergence rate than the latter. This aspect is illustrated using examples taken from two different adaptive applications.

5. Communication channel equalization

Consider the digital communications system depicted in Fig. 2. A binary sequence $x(t)$ is transmitted through a linear dispersive channel modelled as a finite impulse response filter whose transfer function is given by

$$C_n(z) = \sum_{i=0}^{n} c_i z^{-i} \tag{34}$$

The channel output $\hat{o}(t)$ is corrupted by an additive gaussian white noise $e(t)$. The task of the equalizer at the sampling instant t is to reconstruct the input symbol $x(t-d)$ using the information contained in the channel output observations $o(t)$, ..., $o(t-m+1)$, where m is known as the order of the equalizer and $d \geqslant 0$ is a delay parameter. $x(t)$ is assumed to be an independent sequence taking values of either 1 or -1 with equal probability.

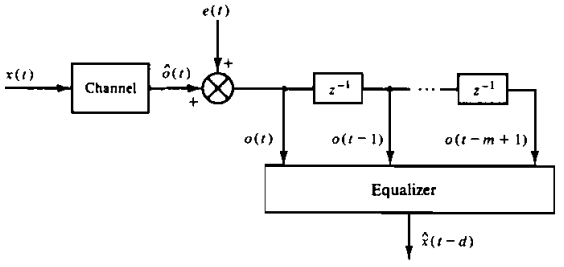

Figure 2. Schematic diagram of data transmission system.

The need for equalizers arises owing to the fact that the channel introduces intersymbol interference and noise. An equalizer tries to remove these undesired channel effects on the transmitted signal. In this sense, channel equalization may be regarded as a form of inverse modelling. Traditionally, communication channel equalization is based on linear finite filter techniques. The problem is, however, an inherently non-linear one (Gibson *et al.* 1989) and non-linear structures are required in order to achieve fully or near optimal performance.

It can be shown that the minimum bit-error-rate equalizer is defined as follows (Gibson *et al.* 1989):

$$\hat{x}(t-d) = \operatorname{sgn}(f_{de}(v(t))) \tag{35}$$

where

$$v(t) = [o(t) \quad \ldots \quad o(t-m+1)]^T \tag{36}$$

is the channel observation vector, and

$$\operatorname{sgn}(y) = \begin{cases} 1 & y \geqslant 0 \\ -1 & y < 0 \end{cases} \tag{37}$$

represents a slicer. $f_{de}(\cdot)$ is known as the decision function, and it partitions the input space $\mathbb{R}^m$ into two sets, one corresponding to the decision $\hat{x}(t-d) = 1$ and the other

$\hat{x}(t-d) = -1$. The boundary that separates these two sets in the input space is called the decision boundary and is often highly non-linear. The channel transfer function $C_n(z)$ and the channel noise distribution function, together with equalizer order m and delay d, completely specify the optimal decision function $f_{de}(\cdot)$.

Because $f_{de}(\cdot)$ for a communications channel is generally not available and can be time varying, a means of adaptively approximating this function or generating the corresponding decision boundary is essential to realize the optimal equalizer solution. The multi-layer neural network offers an ideal solution and it is known that a three-layer (two hidden layers and one output layer) neural network with sufficient neurons can generate an arbitrarily complex decision boundary (Lippmann 1987). The architecture of a three-layer neural network equalizer will be described as $m - n_1 - n_2 - 1$, where m is the dimension of the input space, n_1 and n_2 are the numbers of neurons in the respective hidden layers, and the last number indicates that the output dimension is one and the output layer consists of a single neuron. The aim of a neural network equalizer is to use the input–output function $\hat{y}(t, \Theta) = f(v(t); \Theta)$ to realize the optimal decision function $f_{de}(v(t))$. The activation function for each neuron in the equalizer is given in (2). During training, the error signal is defined as

$$\varepsilon(t) = x(t-d) - f(v(t); \hat{\Theta}(t-1)) \tag{38}$$

In the following simulation study, initial network weights were set randomly to values between $-0{\cdot}5$ to $0{\cdot}5$. The mean square error at the sampling instant t_1 was defined as $2J(\hat{\Theta}(t_1))$, i.e. the variance of $\varepsilon(t, \hat{\Theta}(t_1))$ achieved for an N data point sequence when the equalizer's parameter vector is fixed at $\hat{\Theta}(t_1)$. The bit error rate at t_1 was defined as the ratio of the number of error decisions obtained with the equalizer's parameter vector fixed at $\hat{\Theta}(t_1)$ to the transmitted data length.

Example 1

Channel transfer function $C_1(z) = 1 + 0{\cdot}5z^{-1}$ and additive gaussian white noise variance $0{\cdot}2$; equalizer structure 2–9–5–1 ($n_\Theta = 83$) with zero lag ($d = 0$). Parameters in the two training algorithms were set to:

back propagation algorithm: $\gamma_g = 0{\cdot}01$, $\gamma_m = 0{\cdot}8$

parallel recursive prediction error algorithm: $\gamma_g = 0{\cdot}01$, $\gamma_m = 0{\cdot}8$, $\lambda = 0{\cdot}99$, $K_0 = 40{\cdot}0$, $P_i(0) = 100{\cdot}0I \ \forall\, i$

The evolution of the mean square error and bit error rate using the two learning algorithms are shown in Fig. 3.

Example 2

The channel transfer function $C_2(z) = 0{\cdot}3482 + 0{\cdot}8704z^{-1} + 0{\cdot}3482z^{-2}$ and additive gaussian white noise variance $0{\cdot}01$; equalizer structure 3–9–3–1 ($n_\Theta = 70$) with lag one ($d = 1$).

Parameters in the two training algorithms were identical with those for Example 1 and the evolutions of the mean square error and bit error rate using the two learning algorithms are given in Fig. 4.

The superior convergence performance of the new parallel recursive prediction error algorithm is apparent in the results for both examples.

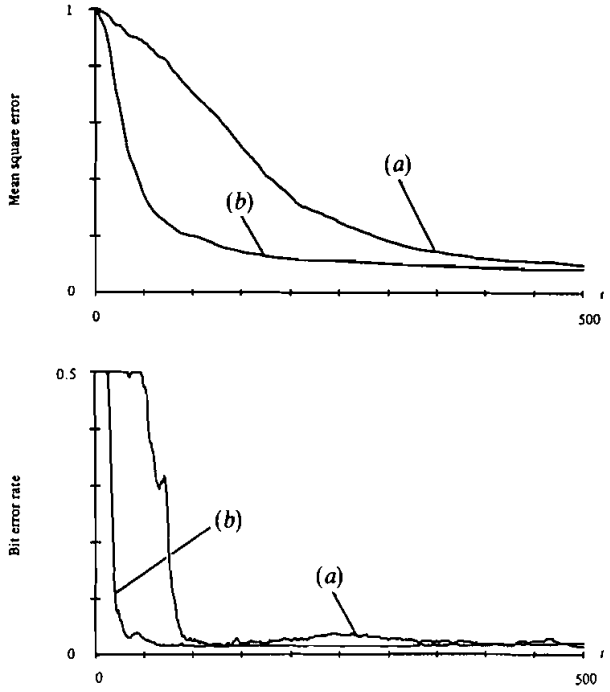

Figure 3. Evolution of mean square error and bit error rate (Example 1): (*a*) back propagation algorithm, (*b*) parallel recursive prediction error algorithm.

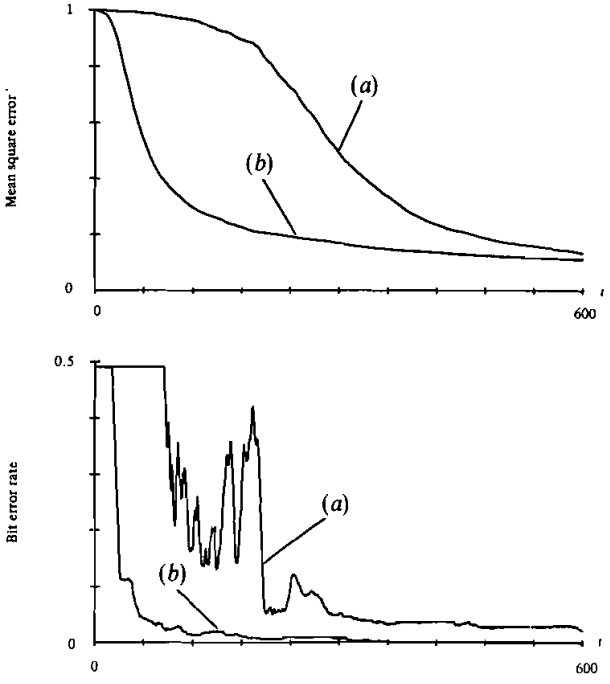

Figure 4. Evolution of mean square error and bit error rate (Example 2): (*a*) back propagation algorithm, (*b*) parallel recursive prediction error algorithm.

6. Non-linear systems modelling

It is known that a wide class of discrete-time non-linear systems can be represented by the non-linear autoregressive moving average with exogenous inputs (NARMAX) model (Leontaritis and Billings 1985, Chen and Billings 1989). The NARMAX model provides a description of the system in terms of some non-linear functional expansion of lagged inputs, outputs and prediction errors, and it is valid for multi-input multi-output systems. Here, a simplified version of the single-input single-output NARMAX system is considered:

$$y(t) = f_s(y(t-1), \ldots, y(t-n_y), u(t-1), \ldots, u(t-n_u)) + e(t) \tag{39}$$

where n_y and n_u are the maximum lags in the system output and input respectively, $e(t)$ is a zero-mean independent system noise sequence, and $f_s(\cdot)$ is some non-linear function.

A schematic diagram representing the non-linear system (39) is illustrated in Fig. 5. Using a neural network to model the non-linear system (39) has been investigated by Chen *et al.* (1990). The underlying idea is to use the neural network model $\hat{y}(t, \Theta) = f(v(t); \Theta)$ as the one-step-ahead predictor for $y(t)$, where

$$v(t) = [y(t-1) \ldots y(t-n_y) u(t-1) \ldots u(t-n_u)]^T \tag{40}$$

The prediction error is accordingly given as

$$\varepsilon(t, \Theta) = y(t) - \hat{y}(t, \Theta) \tag{41}$$

We only consider a neural network model that consists of one hidden layer with n_h neurons and a single output neuron (because the output dimension is one). The activation function for each neuron in the hidden layer is specified by (3). The activation function for the output neuron is chosen to be linear and no threshold parameter is introduced in the output neuron. In the following simulation study, initial network weights were set randomly to values between $-0{\cdot}3$ to $0{\cdot}3$.

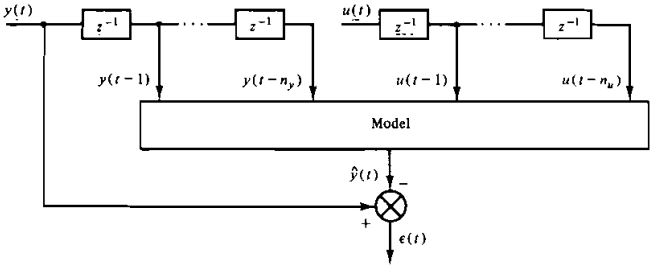

Figure 5. Non-linear systems modelling.

Example 3

This is a simulated time series process. The series was generated by

$$y(t) = (0{\cdot}8 - 0{\cdot}5 \exp(-y^2(t-1)))y(t-1) - (0{\cdot}3 + 0{\cdot}9 \exp(-y^2(t-1)))y(t-2) + e(t)$$

where $e(t)$ was a gaussian white sequence with mean zero and variance $0{\cdot}1$. The structure of the model is given by $m = n_y = 2$ and $n_h = 5$ ($n_\Theta = 20$).

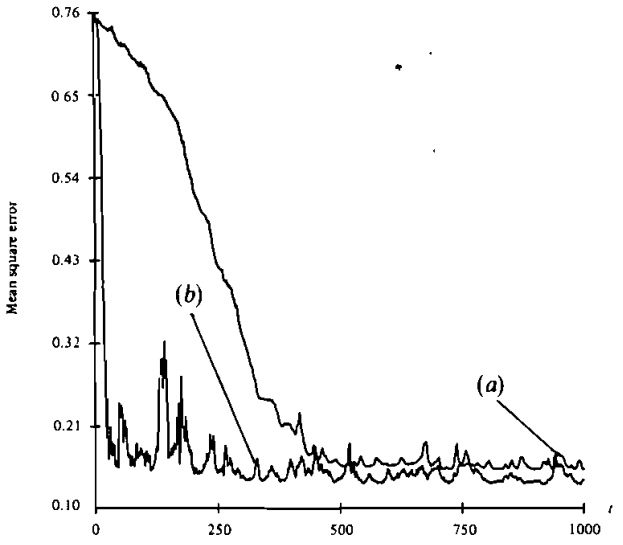

Figure 6. Evolution of mean square error (Example 3): (*a*) back propagation algorithm, (*b*) parallel recursive prediction error algorithm.

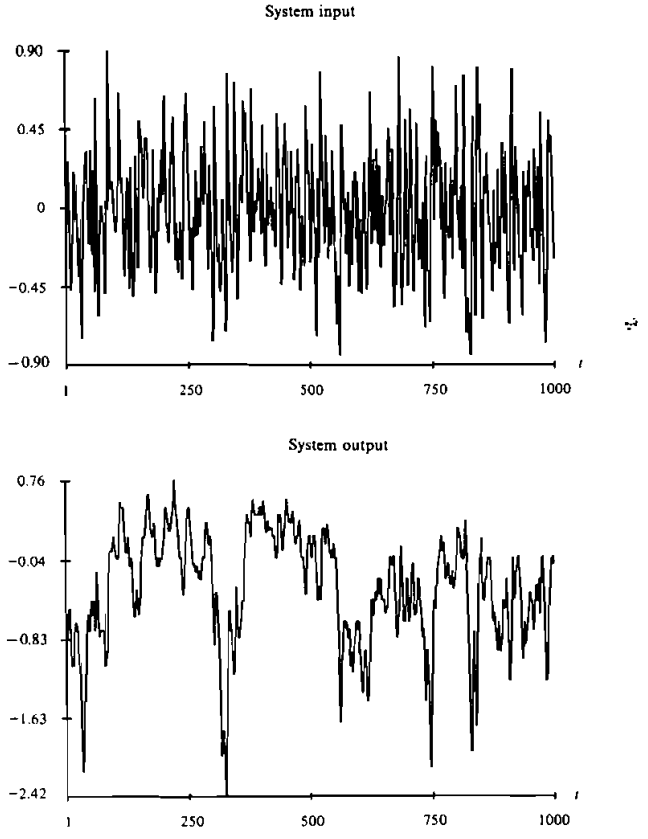

Figure 7. Inputs and outputs of Example 4.

The parameters in the two recursive identification algorithms were chosen to be

back propagation algorithm: $\gamma_g = 0{\cdot}01, \quad \gamma_m = 0{\cdot}8$

parallel recursive prediction error algorithm: $\gamma_g = 1{\cdot}0, \quad \gamma_m = 0{\cdot}0,$
$\lambda = 0{\cdot}99, \quad K_0 = 5{\cdot}0,$
$P_i(0) = 1000{\cdot}0I \ \forall \ i$

The evolution of the mean square error obtained using the two algorithms is plotted in Fig. 6.

Example 4

The data generated from a non-linear liquid level system is shown in Fig. 7. The neural network model was chosen to be $m = n_y + n_u = 3 + 3$ and $n_h = 5$ ($n_\theta = 40$).

The parameters in the two recursive algorithms were set to the same values as those for Example 1, with the exception of $K_0 = 60{\cdot}0$. The evolution of the mean square error obtained in the recursive procedure is shown in Fig. 8.

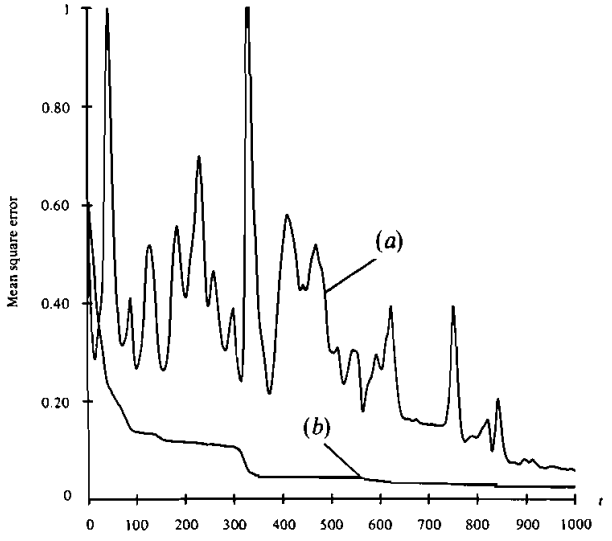

Figure 8. Evolution of mean square error (Example 4): (*a*) back propagation algorithm, (*b*) parallel recursive prediction error algorithm.

7. Conclusions

Neural network learning strategies based on the prediction error estimation principle have been investigated and a new parallel recursive prediction error algorithm has been derived. The new training algorithm can be implemented in an efficient parallel structure and is more powerful than the classical back propagation algorithm. Better convergence performance of this new learning algorithm over the back propagation algorithm has been demonstrated using examples of adaptive channel equalization and recursive non-linear systems identification.

ACKNOWLEDGMENT

This work was supported by the U.K. Science and Engineering Research Council.

References

CHEN, S., and S. A. BILLINGS, 1989, Representation of non-linear systems: the NARMAX model. *International Journal of Control*, **49**, 1013–1032.

CHEN, S., BILLINGS, S. A., and GRANT, P. M., 1990, Non-linear system identification using neural networks. *International Journal of Control*, **51**, 1191–1214.

CHEN, S., GIBSON, G. J., COWAN, C. F. N., and GRANT, P. M., 1989, Recursive prediction error algorithm for training multilayer perceptrons. *Proceedings of the I.E.E.E. Colloquium on Adaptive Algorithms for Signal Estimation and Control*, Edinburgh, Scotland.

GIBSON, G. J., SIU, S., and COWAN, C. F. N., 1989, Application of multilayer perceptrons as adaptive channel equalisers. *Proceedings of the I.E.E.E. International Conference on Acoustics, Speech, and Signal Processing*, Glasgow, Scotland, pp. 1183–1186.

GOODWIN, G. C., and PAYNE, R. L., 1977, *Dynamic System Identification: Experiment Design and Data Analysis* (New York: Academic Press).

KOLLIAS, S., and ANASTASSIOU, D., 1989, An adaptive least squares algorithm for the efficient training of artificial neural networks. *I.E.E.E. Transactions on Circuits and Systems*, **36**, pp. 1092–1101.

LEONTARITIS, I. J., and BILLINGS, S. A., 1985, Input–output parametric models for non-linear systems. Part I: deterministic non-linear systems; Part II: stochastic non-linear systems. *International Journal of Control*, **41**, 303–344.

LIPPMANN, R. P., 1987, An introduction to computing with neural nets. *I.E.E.E. ASSP Magazine*, **4**.

LJUNG, L., and SÖDERSTRÖM, T., 1983, *Theory and Practice of Recursive Identification* (Cambridge, Mass: MIT Press).

RUMELHART, D. E., HINTON, G. E., and WILLIAMS, R. J., 1986, Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, edited by D. E. Rumelhart and J. L. McClelland (Cambridge, Mass: MIT Press), pp. 318–362.

SALGADO, M. E., GOODWIN, G. C., and MIDDLETON, R. H., 1988, Modified least squares algorithm incorporating exponential resetting and forgetting. *International Journal of Control*, **47**, 477–491.

SINGHAL, S., and WU, L., 1989, Training feed-forward networks with the extended Kalman algorithm. *Proceedings of the I.E.E.E. International Conference on Acoustics, Speech, and Signal Processing*, Glasgow, Scotland, pp. 1187–1190.