Mechanical Systems and Signal Processing (1991) 5(3), 233-255

A COMPARISON OF THE BACKPROPAGATION AND RECURSIVE PREDICTION ERROR ALGORITHMS FOR TRAINING NEURAL NETWORKS

S. A. BILLINGS AND H. B. JAMALUDDIN

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

AND

S. CHEN

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

(*Received* 11 *January* 1990, *accepted* 29 *June* 1990)

A new recursive prediction error routine is compared with the backpropagation method of training neural networks. Results based on simulated systems, the prediction of Canadian lynx data, and the modelling of an automotive diesel engine indicate that the recursive prediction error algorithm is far superior to backpropagation.

1. INTRODUCTION

A neural network is a highly parallel dynamical system with the topology of a directed graph that can carry out information processing by means of its state response. Neural networks have been the subject of intense study by researchers from many diverse disciplines. Neurobiologists, psychobiologists and those who study the brain and memory are working to establish the ground rules of how the central nervous system actually works. Computer scientists and researchers in the cognitive sciences are pursuing the study of intelligence and the massively parallel distributed information processing done by biological systems. The engineering community are more involved in image processing, signal processing, optimisation and control problems. The most important factors in employing neural networks for any application include the choice of architecture for the network and appropriate learning algorithms. Various neural network models and learning algorithms have been reviewed by Lippmann [1].

The present study investigates the use of neural networks to model non-linear systems. Non-linear systems can be modelled using the functional series description of Volterra and Wiener [2]. Alternatively, a more concise representation for a wide class of non-linear systems can be achieved by using Non-linear AutoRegressive Moving Average models with eXogeneous inputs or NARMAX models [3]. More recently neural networks have been considered as an alternative functional representation [4, 5]. The use of neural networks in this context is supported by the results of Cybenko [6] and Funahashi [7] who have proved that any continuous function can be uniformly approximated by a neural network with only one internal layer.

The most popular and successful learning algorithm used to train multilayer neural networks in areas such as speech, natural language processing, pattern recognition and system modelling is currently the backpropagation routine [4, 8-10]. Backpropagation however is based on the steepest descent algorithm which is known to be an inefficient

0888-3270/91/030233+23 $03.00/0

optimisation procedure and Brady *et al.* [11] and Sutton [12] have recently reported additional problems with this method of training neural networks. In the present study a new recursive prediction error routine which was introduced by Chen *et al.* [5] and extended to work in parallel by Chen *et al.* [13] is extensively tested and compared with the backpropagation routine. The influence of the momentum and the learning rate constants on the convergence of the backpropagation algorithm are studied and model validation methods are introduced as a measure of network performance. Convergence analysis of the recursive prediction error algorithm is discussed and the method is shown to produce an improved convergence rate and prediction accuracy compared with the backpropagation routine. Simulated examples, the prediction of Canadian lynx data and the identification of a model relating fuel rack position to engine speed of an automotive diesel engine are used to compare the two algorithms.

2. NON-LINEAR SYSTEM REPRESENTATION

Multilayered neural networks considered in the present study are feed-forward networks. A multilayered neural network is made up of one or more hidden layers between the input and output layers. Each layer consists of computing units called nodes or neurons connected together in the structure of a layered network as illustrated in Fig. 1. The functionality of the network is determined by specifying the strength of the connection paths called weights and the threshold parameter of each neuron.

The input layer usually acts as an input data holder and it distributes inputs to the first hidden layer. The inputs then propagate forward through the network and each neuron

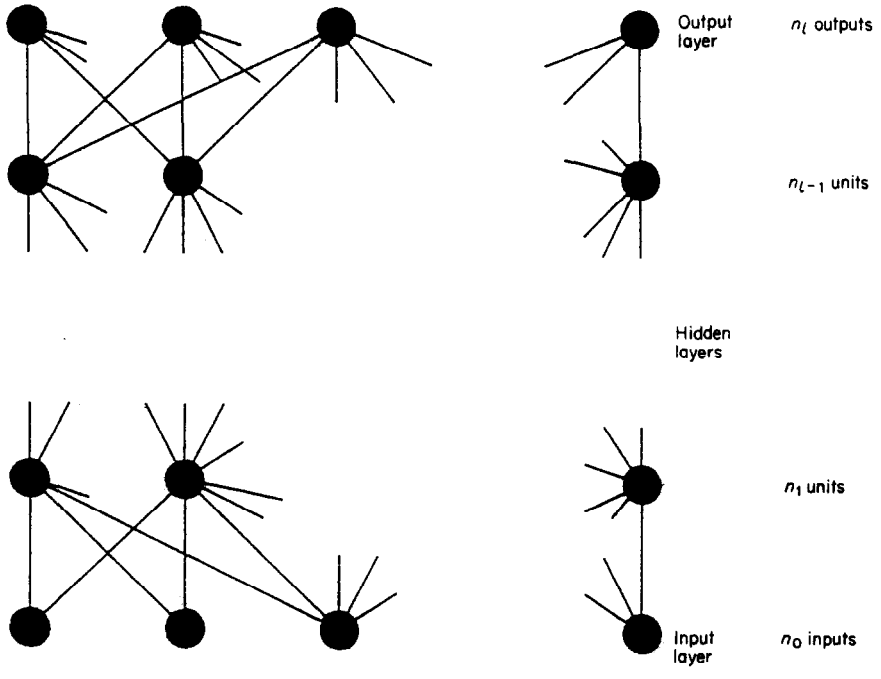


Figure 1. A multilayered neural network.

computes its output according to

$$x_i^k = g\left(\sum_{j=1}^{n_{k-1}} w_{ij}^k x_j^{k-1} + b_i^k\right) \tag{1}$$

for $i = 1, \ldots, n_k$ and $k = 1, \ldots, l$, where x_i^k is the output of the ith neuron in the kth layer, w_{ij}^k is the weight connection between the jth neuron of the $(k-1)$th layer and the ith neuron of the kth layer, and b_i^k is the threshold of the ith neuron in the kth layer. As shown in Fig. 1, the lth layer is the output layer and the input layer may be labelled as layer zero. Thus n_0 and n_l refer to the numbers of network inputs and outputs respectively. x_i^0 and x_i^k will also be denoted as x_i and $\hat{y}_i$. The function $g(.)$ is called the node activation function. The argument

$$z_i^k = \sum_{j=1}^{n_{k-1}} w_{ij}^k x_j^{k-1} + b_i^k \tag{2}$$

is therefore the activation for the ith neuron in the kth layer. The function $g(.)$ is assumed to be differentiable and to have a strictly positive first derivative.

For the neurons in the hidden layers, the activation function is often chosen to be

$$g(z) = \frac{1}{1+e^{-z}}. \tag{3}$$

Since in system modelling applications the dynamic range of the output data may be greater than 1, the activation function of the output nodes is chosen to be linear. Thus the ith output node performs a weighted sum of its inputs as follows

$$\hat{y}_i = \sum_{j=1}^{n_{l-1}} w_{ij}^l x_j^{l-1}. \tag{4}$$

Only networks with one hidden layer are considered in the present study because the results of Cybenko [6] and Funahashi [7] show that this is sufficient to approximate all continuous functions. Let $\Theta = [\theta_1 \ \cdots \ \theta_{n_\theta}]^T$ represent all the unknown weights and thresholds of the network where n_θ denotes the dimension of the parameter vector Θ and is defined as $n_\theta = \sum_{j=1}^{l-1} n_j(n_{j-1}+1) + n_{l-1} \times n_l$. A network with a single layer of hidden units can then be defined by the model

$$\hat{y}_i(t, \Theta) = \sum_{j=1}^{n_1} w_{ij}^2 x_j^1(t) = \sum_{j=1}^{n_1} w_{ij}^2 g\left(\sum_{k=1}^{n_0} w_{jk}^1 x_k(t) + b_j^1\right), \qquad 1 \leqslant i \leqslant n_2 \tag{5}$$

where $x(t) = [x_1(t) \cdots x_{n_0}(t)]^T$ is the input vector to the network.

3. LEARNING ALGORITHMS

Two learning algorithms are considered. The backpropagation algorithm is a gradient algorithm designed to minimise the mean square error between the actual output of the network and the desired output. The recursive vesion of this algorithm is discussed in the present study. The recursive prediction error or RPE algorithm is based on the class of unified recursive parameter estimation methods which minimise the prediction error over the model set using an approximation of the Gauss–Newton search direction. Both approaches require differentiable non-linearities, and the backpropagation algorithm can be viewed as a simplified version of the RPE algorithm.

3.1. BACKPROPAGATION ALGORITHM

The backpropagation algorithm has been described in Rumelhart and McClelland [10]. Define the energy function as

$$J = \tfrac{1}{2}\sum_i (y_i(t) - \hat{y}_i(t))^2 \tag{6}$$

where $y_i(t)$ is the ith desired output. This energy function is to be minimised with respect to all the unknown parameters. In the steepest descent approach the parameter vector Θ is adjusted using the increment vector $[\Delta\theta_1 \quad \Delta\theta_2 \cdots \Delta\theta_{n_\theta}]^T$ defined along the negative gradient direction of J

$$\Delta\theta_i = -\eta \frac{\partial J}{\partial \theta_i} = -\eta \nabla_i J. \tag{7}$$

Although the one-hidden-layer model is used in the present application it is useful to derive the gradient of J for the general case and the result for the one-hidden-layer model can readily be obtained as a special case.

Starting from the top of the network, the application of the chain rule gives rise to

$$\frac{\partial J}{\partial w_{ij}^l} = \frac{\partial J}{\partial \hat{y}_i} \times \frac{\partial \hat{y}_i}{\partial w_{ij}^l}. \tag{8}$$

From equation (6)

$$\frac{\partial J}{\partial \hat{y}_i} = -(y_i - \hat{y}_i) = -\delta_i^l \tag{9}$$

where δ_i^l is called the error signal of the ith neuron in the lth layer. From equation (4)

$$\frac{\partial \hat{y}_i}{\partial w_{ij}^l} = x_j^{l-1}. \tag{10}$$

Thus

$$\frac{\partial J}{\partial w_{ij}^l} = -\delta_i^l x_j^{l-1}. \tag{11}$$

Next consider the $(l-1)$th layer. Using the chain rule yields

$$\frac{\partial J}{\partial w_{ij}^{l-1}} = \sum_k \frac{\partial J}{\partial \hat{y}_k} \times \frac{\partial \hat{y}_k}{\partial x_i^{l-1}} \times \frac{\partial x_i^{l-1}}{\partial z_i^{l-1}} \times \frac{\partial z_i^{l-1}}{\partial w_{ij}^{l-1}}. \tag{12}$$

It is easy to see that

$$\frac{\partial \hat{y}_k}{\partial x_i^{l-1}} = w_{ki}^l \tag{13}$$

$$\frac{\partial x_i^{l-1}}{\partial z_i^{l-1}} = g'(z_i^{l-1}) \tag{14}$$

$$\frac{\partial z_i^{l-1}}{\partial w_{ij}^{l-1}} = x_j^{l-2} \tag{15}$$

where

$$g'(z) = \frac{\partial g(z)}{\partial z}. \tag{16}$$

By defining the error signal for the ith neuron of the $(l-1)$th layer as

$$\delta_i^{l-1} = \sum_k \left(-\frac{\partial J}{\partial \hat{y}_k} \right) \times \frac{\partial \hat{y}_k}{\partial x_i^{l-1}} \times \frac{\partial x_i^{l-1}}{\partial z_i^{l-1}} = g'(z_i^{l-1}) \sum_k \delta_k^l w_{ki}^l \tag{17}$$

equation (12) can be rewritten as

$$\frac{\partial J}{\partial w_{ij}^{l-1}} = -\delta_i^{l-1} x_j^{l-2}. \tag{18}$$

Similarly it can be shown that

$$\frac{\partial J}{\partial b_i^{l-1}} = -\delta_i^{l-1}. \tag{19}$$

By carrying on this procedure the following general results are obtained for $m = l, l-1, \ldots, 2$

$$\delta_i^{m-1} = g'(z_i^{m-1}) \sum_k \delta_k^m w_{ki}^m \tag{20a}$$

$$\frac{\partial J}{\partial w_{ij}^{m-1}} = -\delta_i^{m-1} x_j^{m-2} \tag{20b}$$

$$\frac{\partial J}{\partial b_i^{m-1}} = -\delta_i^{m-1}. \tag{20c}$$

Equation (20a) indicates how the error signals propagate down the network, hence, "backpropagation".

The steepest-descent minimisation of the energy function defined in equation (6) produces the following increments for updating Θ

$$\Delta w_{ij}^k(t) = \eta_w \delta_i^k(t) x_j^{k-1}(t) \tag{21a}$$

$$\Delta b_i^k(t) = \eta_b \delta_i^k(t) \tag{21b}$$

where

$$\delta_i^l(t) = y_i(t) - \hat{y}_i(t) \tag{22a}$$

and

$$\delta_i^k(t) = g'(z_i^k(t)) \sum_m \delta_m^{k+1}(t) w_{mi}^{k+1}(t-1), \qquad k = l-1, \ldots, 2, 1. \tag{22b}$$

The constants η_w and η_b represent the learning rates for the weights and thresholds respectively. In practice, a large value of the learning rate would be preferable because this would result in rapid learning. Unfortunately a large value of learning rate can also lead to oscillation or even divergence. To overcome this problem, a momentum term is usually included so that equations (21a) and (21b) become

$$\Delta w_{ij}^k(t) = \eta_w \delta_i^k(t) x_j^{k-1}(t) + \alpha_w \Delta w_{ij}^k(t-1) \tag{23a}$$

$$\Delta b_i^k(t) = \eta_b \delta_i^k(t) + \alpha_b \Delta b_i^k(t-1) \tag{23b}$$

where α_w and α_b are momentum constants which determine the effect of past changes of $\Delta w_{ij}^k(t-1)$ and $\Delta b_i^k(t-1)$ on the current updating direction in the weight and the threshold space respectively. This effectively filters out high frequency variations in the error surface.

To summarise, the backpropagation algorithm updates the weights and thresholds of the network according to

$$w_{ij}^k(t) = w_{ij}^k(t-1) + \Delta w_{ij}^k(t) \tag{24a}$$

$$b_i^k(t) = b_i^k(t-1) + \Delta b_i^k(t) \tag{24b}$$

with the increments $\Delta w_{ij}^k(t)$ and $\Delta b_i^k(t)$ given in equations (23a) and (23b).

3.1.1. *Implementation*

The implementation of the backpropagation algorithm involves two phases. Weights and thresholds are usually initialised with small random values. In the first phase, inputs are presented to the first hidden layer and these signals then propagated through the network to the output layer. The actual network outputs obtained from the output neurons are compared with the desired outputs to produce the prediction errors or residuals. The second stage involves propagating error signals backwards from the output layer to the first layer. Weights and thresholds are updated based on these error signals. This process is repeated until weights and thresholds converge.

For the activation function defined in equation (3) it can be shown that

$$g'(z_i^k) = x_i^k(1 - x_i^k). \tag{25}$$

The implementation for the one-hidden-layer model equation (5) can be summarised as follows:

i. Initialise weights and thresholds for the network with small random values and select values for the learning rates η_w and η_b and the momentum constants α_w and α_b.

ii. Present the input data to the network and compute the outputs of the network according to

$$\hat{y}_i(t) = \sum_{j=1}^{n_1} w_{ij}^2(t-1) g\left(\sum_{k=1}^{n_0} w_{jk}^1(t-1) x_k(t) + b_j^1(t-1) \right).$$

iii. Compare the outputs of the network with the desired outputs to produce the prediction errors $\varepsilon_i(t)$ which are equal to the error signals $\delta_i^2(t)$ for the output neurons

$$\varepsilon_i(t) = \delta_i^2(t) = (y_i(t) - \hat{y}_i(t)).$$

iv. Compute the error signals for the hidden neurons. As an example consider hidden neuron j whose output is transmitted to two output neurons as illustrated in Fig. 2. The error signal for this hidden neuron is then given by

$$\delta_j^1(t) = x_j^1(t)(1 - x_j^1(t))(\delta_k^2(t) w_{kj}^2(t-1) + \delta_i^2(t) w_{ij}^2(t-1)).$$

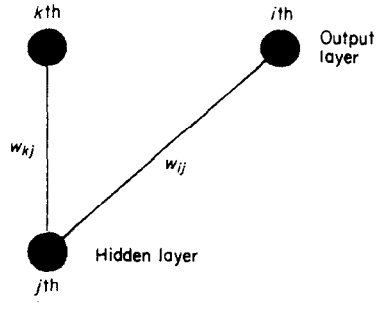


Figure 2. Hidden neuron j connected to output neurons k and i.

v. Update the weight connections between the hidden layer and the output layer

$$\Delta w_{ij}^2(t) = \eta_w \delta_i^2(t) x_j^1(t) + \alpha_w \Delta w_{ij}^2(t-1)$$

$$w_{ij}^2(t) = w_{ij}^2(t-1) + \Delta w_{ij}^2(t).$$

vi. Update the weights and thresholds for the hidden neurons according to

$$\Delta w_{ij}^1(t) = \eta_w \delta_i^1(t) x_j(t) + \alpha_w \Delta w_{ij}^1(t-1)$$

$$w_{ij}^1(t) = w_{ij}^1(t-1) + \Delta w_{ij}^1(t)$$

$$\Delta b_i^1(t) = \eta_b \delta_i^1(t) + \alpha_b \Delta b_i^1(t-1)$$

$$b_i^1(t) = b_i^1(t-1) + \Delta b_i^1(t).$$

Steps (ii) to (vi) are repeated until convergence.

3.1.2. *Convergence*

It is well-known, from standard non-linear optimisation theory, that the block-data version of the backpropagation algorithm will converge to a local minimum of the mean square error surface. The backpropagation algorithm discussed in the present study is a recursive version. Notice that this algorithm is known as the smoothed stochastic algorithm in system identification. If the learning rates η_w and η_b are made t-dependent and tend to zero as $t \to +\infty$, a unified method developed by Ljung [14] for convergence analysis of recursive algorithms can readily be applied. Under the above assumption on learning rates, it can be shown that the recursive version of the backpropagation algorithm has the same convergence properties as its off-line counterpart. This means that the parameter vector $\hat{\Theta}(t)$ will converge to a local minimum of the mean square error surface as the sample points $t \to \infty$. Furthermore convergence of the backpropagation algorithm can be discussed within the same framework for the RPE algorithm. However it is generally true that the convergence of the backpropagation algorithm is fairly slow compared with the RPE algorithm. It is reasonable to believe that these results will have relevance for the case of smaller constant learning rates.

If all the neurons have a linear activation function, the network will degenerate to a linear one. In this extreme case, the convergence results for the adaptive linear combiner [15] are valid.

3.2. RECURSIVE PREDICTION ERROR (RPE) ALGORITHM

The off-line prediction error algorithm (PEM) is derived based on the criterion

$$J(\Theta) = \frac{1}{N} \sum_{t=1}^{N} Q(\varepsilon(t), \Theta) \tag{26}$$

where $Q(.,.)$ is the measure of fit and $\varepsilon(t)$ is the prediction error vector. A good measure of fit is a quadratic function of $\varepsilon(t)$ [16]

$$Q(\varepsilon(t), \Theta) = \tfrac{1}{2} \varepsilon^T(t, \Theta) \Lambda^{-1} \varepsilon(t, \Theta) \tag{27}$$

where Λ is some $n_l \times n_l$ symmetric positive definite matrix and n_l is the number of output nodes. If the prediction errors are Gaussian with zero mean, $Q(.,.)$ can be chosen as the negative log likelihood function

$$Q(\varepsilon(t), \Theta) = \text{constant} + \tfrac{1}{2} \log |\Lambda| + \tfrac{1}{2} \varepsilon^T(t, \Theta) \Lambda^{-1} \varepsilon(t, \Theta). \tag{28}$$

When the covariance matrix Λ is known (independent of Θ), the first two terms in $Q(.,.)$ have no effect on the minimisation which then becomes equivalent to equation (27). In this case the prediction error estimate is equal to the maximum likelihood estimate with

the advantage that the estimation accuracy satisfies the Cramer-Rao bound. In general the prediction error will not be Gaussian. However the asymptotic properties of PEM with $Q(.,.)$ given in equation (27) are very similar to those of maximum likelihood and it can be shown that under weak assumptions the PE estimates are consistent, asymptotically normally distributed and asymptotically efficient.

The minimisation of the criterion (26) can be performed using the Gauss-Newton algorithm

$$\Theta^{(k)} = \Theta^{(k-1)} + s^{(k)}\mu(\Theta^{(k-1)}) \tag{29}$$

where superscript (k) denotes the iteration step, $\mu(\Theta)$ is the Gauss-Newton search direction defined by

$$\mu(\Theta) = -[H(\Theta)]^{-1}\nabla J(\Theta) \tag{30}$$

$\nabla J(\Theta)$ is the gradient of $J(\Theta)$ and $H(\Theta)$ is the second derivative, known as the Hessian matrix of $J(\Theta)$. The gradient of $J(\Theta)$ can be derived by substituting equations (27) into the criterion (26) to yield

$$\nabla J(\Theta) = -\frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Lambda^{-1}\varepsilon(t, \Theta) \tag{31}$$

where

$$\Psi(t, \Theta) = \left[\frac{d\hat{y}(t, \Theta)}{d\Theta}\right]^T \tag{32}$$

is the first derivative of the one step ahead prediction with respect to Θ. The Hessian of $J(\Theta)$ is obtained by differentiating $\nabla J(\Theta)$

$$H(\Theta) = \frac{\partial^2 J}{\partial \Theta^2} = \frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Lambda^{-1}\Psi^T(t, \Theta) - \frac{1}{N}\sum_{t=1}^{N} \frac{d\Psi(t, \Theta)}{d\Theta}\Lambda^{-1}\varepsilon(t, \Theta). \tag{33}$$

At a minimum point of the criterion (26), the prediction errors form an uncorrelated sequence, and it can be shown that the expected value of the last term in equation (33) goes to zero so that equation (33) can be approximated as

$$H(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Lambda^{-1}\Psi^T(t, \Theta). \tag{34}$$

In case the matrix $H(\Theta)$ may be close to singular, the following approximate Hessian can be taken

$$H(\Theta) = \frac{1}{N}\sum_{t=1}^{N} \Psi(t, \Theta)\Lambda^{-1}\Psi^T(t, \Theta) + \beta I \tag{35}$$

where I is the identity matrix and β is a non-negative small scalar. $s^{(k)}$ is obtained by minimising $J(\Theta^{(k-1)} + s\mu(\Theta^{(k-1)}))$ over $0 < s < 1$ using a linear search technique.

The off-line prediction error algorithm can be used as a basis to derive a recursive algorithm to minimise the criterion (26). The derivation is given by refs. [16, 17] and yields the standard recursive prediction error algorithm

$$\varepsilon(t) = y(t) - \hat{y}(t) \tag{36}$$

$$R(t) = R(t-1) + \gamma(t)[\Psi(t)\Lambda^{-1}\Psi^T(t) + \beta I - R(t-1)] \tag{37}$$

$$\hat{\Theta}(t) = \hat{\Theta}(t-1) + \gamma(t)R^{-1}(t)\Psi(t)\Lambda^{-1}\varepsilon(t) \tag{38}$$

where $\hat{\Theta}(t)$ is the estimate of the parameter vector and $\gamma(t)$ is the gain at sample t. The negative gradient of $Q(\varepsilon(t))$ is $\Psi(t)\Lambda^{-1}\varepsilon(t)$ which is a stochastic gradient and $R^{-1}(t)\Psi(t)\Lambda^{-1}\varepsilon(t)$ can be regarded as an approximation of the Gauss-Newton search direction.

The simplest choice of Λ is I [17]. In practice the RPE algorithm is not implemented with an inversion of $R(t)$ in each step, but the algorithm [16] is updated in terms of

$$P(t)=\gamma(t)R^{-1}(t) \tag{39}$$

to give for $\beta=0$

$$\varepsilon(t)=y(t)-\hat{y}(t) \tag{40}$$

$$P(t)=\frac{1}{\lambda(t)}[P(t-1)-P(t-1)\Psi(t)[\lambda(t)I+\Psi^T(t)P(t-1)\Psi(t)]^{-1}\Psi^T(t)P(t-1)] \tag{41}$$

$$\hat{\Theta}(t)=\hat{\Theta}(t-1)+P(t)\Psi(t)\varepsilon(t) \tag{42}$$

where

$$\lambda(t)=\frac{\gamma(t-1)}{\gamma(t)}(1-\gamma(t)) \tag{43}$$

$\lambda(t)$ is called the forgetting factor. When it is required to weight out the old data, it is required that $\lambda(t)<1$. It is desirable to set $\lambda(t)<1$ at the initial stage so that rapid adaptation takes place and then to let $\lambda(t)\to 1$ as $t\to\infty$ [17]. These objectives can be achieved using the scheme

$$\lambda(t)=\lambda_0\lambda(t-1)+(1-\lambda_0) \tag{44}$$

where λ_0 and the initial forgetting factor $\lambda(0)$ are design values.

The gradient $\Psi(t,\Theta)$ is an $n_\theta\times n_l$ matrix. For networks with a single hidden layer ($n_l=n_2$), the number of parameters to be estimated is given by

$$n_\theta=(n_0+1)\times n_1+n_1\times n_2 \tag{45}$$

The elements of Ψ for the one-hidden-layer model can be obtained by differentiating equation (5) with respect to θ_i to yield

$$\Psi_{ij}=\frac{d\hat{y}_j}{d\theta_i}=\begin{cases}x_k^1, & \text{if } \theta_i=w_{jk}^2,\ 1\leqslant k\leqslant n_1\\ x_k^1(1-x_k^1)w_{jk}^2, & \text{if } \theta_i=b_k^1,\ 1\leqslant k\leqslant n_1\\ x_k^1(1-x_k^1)w_{jk}^2x_m, & \text{if } \theta_i=w_{km}^1,\ 1\leqslant k\leqslant n_1,\ 1\leqslant m\leqslant n_0\\ 0, & \text{otherwise}\end{cases} \tag{46}$$

where the activation function $g(.)$ is assumed to be equation (3). If the output layer consists of a single neuron, $n_2=1$, the index j in equation (46) can then be removed. Consider for example the network shown in Fig. 3 where $n_0=3$, $n_1=2$, $n_2=1$ and the input vector is given by

$$x(t)=[x_1(t)\quad x_2(t)\quad x_3(t)]^T=[u(t-1)\quad u(t-2)\quad y(t-1)]^T.$$

Using equation (45), the number of parameters to be estimated is ten. The parameter vector Θ is then

$$\Theta=[\theta_1\quad \theta_2\quad \cdots\quad \theta_{10}]^T$$

If the following equivalences

$$\theta_1=w_{11}^1,\quad \theta_2=w_{12}^1,\quad \theta_3=w_{13}^1,\quad \theta_4=b_1^1,\quad \theta_5=w_{21}^1,$$
$$\theta_6=w_{22}^1,\quad \theta_7=w_{23}^1,\quad \theta_8=b_2^1,\quad \theta_9=w_1^2,\quad \theta_{10}=w_2^2$$

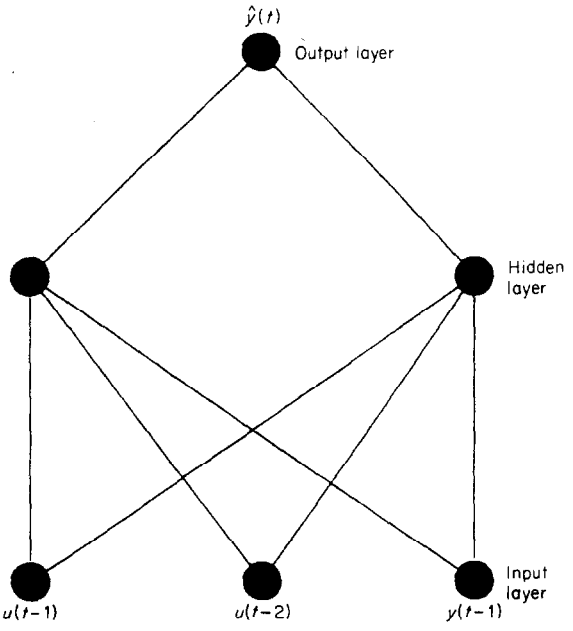


Figure 3. A three-layered neural network with one output neuron, two hidden neurons and three input neurons.

are introduced, the elements of Ψ in this case are given by

$$\psi_1 = x_1^1(1-x_1^1)w_1^2x_1, \quad \psi_2 = x_1^1(1-x_1^1)w_1^2x_2, \quad \psi_3 = x_1^1(1-x_1^1)w_1^2x_3,$$
$$\psi_4 = x_1^1(1-x_1^1)w_1^2, \quad \psi_5 = x_2^1(1-x_2^1)w_2^2x_1, \quad \psi_6 = x_2^1(1-x_2^1)w_2^2x_2,$$
$$\psi_7 = x_2^1(1-x_2^1)w_2^2x_3, \quad \psi_8 = x_2^1(1-x_2^1)w_2^2, \quad \psi_9 = x_1^1, \quad \psi_{10} = x_2^1.$$

Finally it is worth pointing out that, if $P(t)$ in equation (42) is replaced by the identity matrix I, the RPE algorithm reduces to the backpropagation algorithm written in a vector form. This can be verified easily. The matrix $P(t)$ determines the asymptotical accuracy of the estimate and therefore is also called the covariance matrix.

3.2.1. *Implementation*

The implementation of the RPE algorithm involves two phases. The first phase is the same as that of the backpropagation algorithm, that is to obtain the output $\hat{y}(t)$. The second phase includes computing the covariance matrix $P(t)$ and the negative gradient $\Psi(t)\varepsilon(t)$ and updating the parameter vector. The matrix $P(t)$ in particular requires matrix algebra. The RPE algorithm is therefore computationally more demanding than the simple backpropagation method. The RPE algorithm for one-hidden-layer model equation (5) can be summarised as follows:

i. Initialise the weights and thresholds with small random values; choose $P(0)$ as a diagonal matrix with large diagonal values typically in the range of 10^2I to 10^4I; assign values to λ_0 and $\lambda(0)$ typically $\lambda_0 = 0{\cdot}99$ and $\lambda(0) = 0{\cdot}95$.
ii. Present inputs to the network and compute the network outputs.
iii. Compare the network outputs with the desired outputs to give the prediction errors $\varepsilon(t)$; then compute elements of $\Psi(t)$ according to equation (46). Notice this is done from the top output layer down to the hidden layer.
iv. Compute $P(t)$ according to equation (41).
v. Adjust parameter vector Θ according to equation (42).

Steps (ii) to (v) are repeated until convergence.

3.2.2. *Convergence*

It has been shown [14] that the RPE algorithm used in linear systems identification has the same convergence properties as its off-line counterpart. One of these properties is that the estimate $\hat{\Theta}(t)$ converges with probability one either to a local minimum of $E[\varepsilon^T(t, \Theta)\varepsilon(t, \Theta)]$ or to a boundary point of the set of the parameter vector describing the model set as t tends to infinity, where $E[.]$ is the expectation operator. The proof of convergence is based upon the differential equation associated with the RPE algorithm

$$\frac{d}{d\tau}\hat{\Theta}(\tau) = R^{-1}(\tau) f(\hat{\Theta}(\tau)) \tag{47a}$$

$$\frac{d}{d\tau}R(\tau) = G(\hat{\Theta}(\tau)) - R(\tau) \tag{47b}$$

where

$$f(\hat{\Theta}) = E[\Psi(t, \Theta)\varepsilon(t, \Theta)] \tag{48a}$$

$$G(\hat{\Theta}) = E[\Psi(t, \Theta)\Psi^T(t, \Theta)]. \tag{48b}$$

The convergence of the RPE algorithm is connected with the stability of the above differential equations. The stability of equations (47a) and (47b) can be proved by applying Lyapunov stability theory. It can also be shown that the estimate $\hat{\Theta}(t)$ should follow the trajectories of the above differential equations asymptotically. These results can be extended to non-linear systems [5, 18].

Under the assumption that the learning rates tend to zero as $t \to +\infty$, the convergence properties of the backpropagation algorithm can be similarly analysed. The differential equation associated with the backpropagation algorithm in this case can easily be obtained by replacing $R(\tau)$ in equation (47a) with the identity matrix to yield

$$\frac{d}{d\tau}\hat{\Theta}(\tau) = f(\hat{\Theta}(\tau)). \tag{49}$$

4. MODEL VALIDITY TESTS

The main objective in system modelling is to produce an acceptable model which accurately represents the system. The identified model will only produce acceptable predictions over different data sets if it is unbiased. If the model structure and the estimated parameters are correct then the prediction error sequence $\varepsilon(t)$ should be unpredictable from all linear and non-linear combinations of past inputs and outputs. This condition will hold if and only if [19]

$$\begin{aligned}
&\phi_{\varepsilon\varepsilon}(\tau) = 0, && \tau \neq 0\\
&\phi_{u\varepsilon}(\tau) = 0, && \text{for all } \tau\\
&\phi_{\varepsilon\varepsilon u}(\tau) = 0, && \tau \geq 0\\
&\phi_{u^2\varepsilon^2}(\tau) = 0, && \text{for all } \tau\\
&\phi_{u^2\varepsilon}(\tau) = 0, && \text{for all } \tau
\end{aligned} \tag{50}$$

where ϕ represents the standard correlation function. If these functions fall within the 95% confidence intervals ($\pm 1{\cdot}96/\sqrt{N}$), the model can be regarded as adequate.

5. SIMULATION STUDY

Five examples of non-linear systems were considered. The first three examples called $S1$, $S2$ and $S3$ represent simulated non-linear systems. The fourth example is the prediction

of Canadian lynx data and the fifth example is the identification of an automotive diesel engine. The fourth example is a scalar time series and the rest of the examples are all single-input single-output systems. Therefore the output layer of neural networks requires only one neuron ($n_2 = 1$).

5.1. SIMULATED SYSTEMS: $S1$, $S2$ AND $S3$

Neural network models with a single layer of hidden units were used to model these systems. The hidden layer consisted of five hidden neurons ($n_1 = 5$) and the network input vector is defined by

$$x(t) = [u(t-1) \quad u(t-2) \quad u(t-3) \quad y(t-1) \quad y(t-2)]^T$$

giving $n_0 = 5$. In each of the three cases, the input $u(t)$ was a zero mean uniformly distributed white noise sequence and 500 data points were generated. The backpropagation routine and the RPE algorithm were used to train the networks. Comparisons of these algorithms were made using the following models:

$$S1: y(t) = 0{\cdot}8y(t-1) + 0{\cdot}4(u(t-1) + u^2(t-1) + u^3(t-1))$$

$$S2: y(t) = 0{\cdot}4y^2(t-1) + 0{\cdot}2u(t-1) + 0{\cdot}6u^2(t-1)$$

$$S3: y(t) = 0{\cdot}8y(t-1) + 0{\cdot}5u^2(t-1)y(t-1) + u^3(t-1).$$

These polynomial models are chosen such that $S1$ contains power non-linearities in the input, $S2$ contains power non-linearities both in the input and output and $S3$ contains a cross-multiplication term and power non-linearity.

The measure of closeness between the predicted output and the measured output used is the error index defined as the normalised root mean square of the residual given by

$$\text{Error index} = \sqrt{\frac{\sum (\hat{y}(t) - y(t))^2}{\sum y^2(t)}}.$$

The main result obtained from the simulations was that the RPE algorithm converged faster and gave a better prediction than the backpropagation algorithm. This is shown in Figs 4, 5 and 6 for simulations $S1$, $S2$ and $S3$ respectively. The error index curves obtained

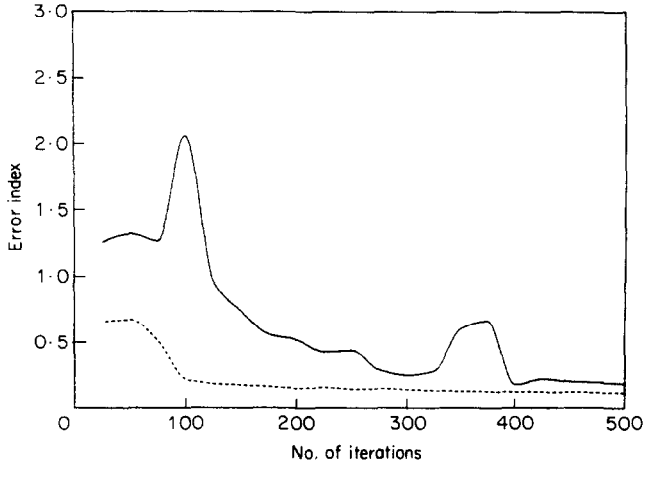


Figure 4. Evolution of the error index for the backpropagation (——) and RPE (- - -) algorithms for simulation $S1$.

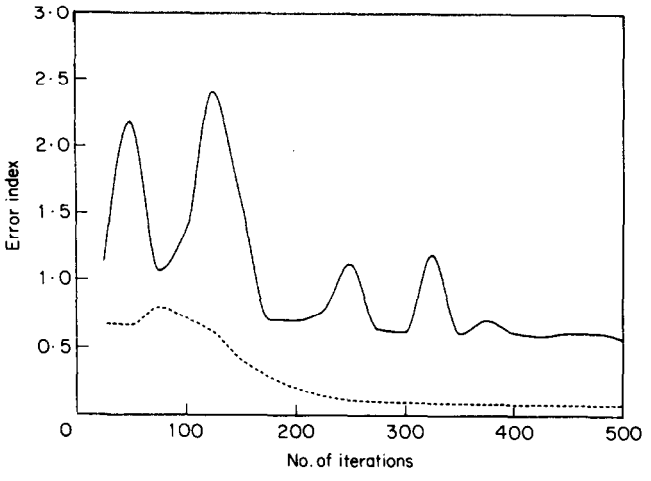


Figure 5. Evolution of error index for the backpropagation (——) and RPE (- - -) algorithms for simulation *S*2.

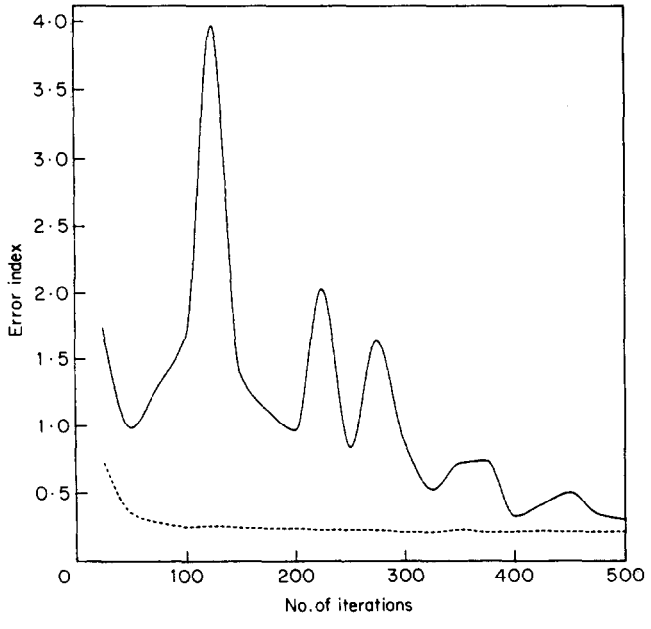


Figure 6. Evolution of error index for the backpropagation (——) and RPE (- - -) algorithms for simulation *S*3.

for the RPE algorithm were smoother than those of the backpropagation algorithm. This is easily understood because the covariance matrix $P(t)$ has the effect of smoothing the random variations in the gradient vector.

It was found that the choice of learning rates and the momentum constants for the backpropagation algorithm influenced the evolution of the error indexes and their corresponding final values. Normally, small values of learning rates, η_w and η_b are chosen and momentum constants, α_w and α_b are chosen to be less than but near to 1. The effect of varying one of these parameters whilst keeping the others constant is illustrated in Fig. 7. The final values of the error indexes were 0·487, 0·165, 0·312, 1·292 and 0·243 for

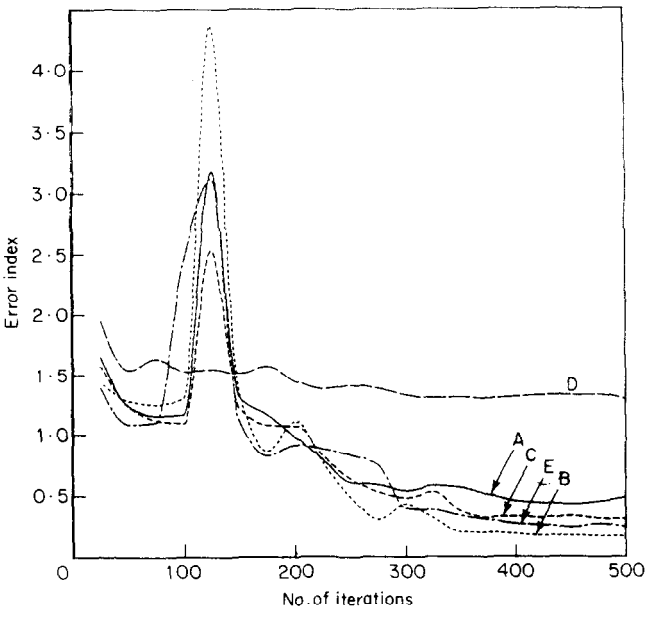


Figure 7. Effect of user selectable parameters on the evolution of the error index for backpropagation for simulation $S1$. A, $\eta_w = \eta_b = 0{\cdot}4$, $\alpha_w = \alpha_b = 0{\cdot}8$; B, $\eta_w = \eta_b = 0{\cdot}5$, $\alpha_w = \alpha_b = 0{\cdot}8$; C, $\eta_w = \eta_b = 0{\cdot}3$, $\alpha_w = \alpha_b = 0{\cdot}8$; D, $\eta_w = \eta_b = 0{\cdot}4$, $\alpha_w = \alpha_b = 0{\cdot}9$; E, $\eta_w = \eta_b = 0{\cdot}4$, $\alpha_w = \alpha_b = 0{\cdot}7$.

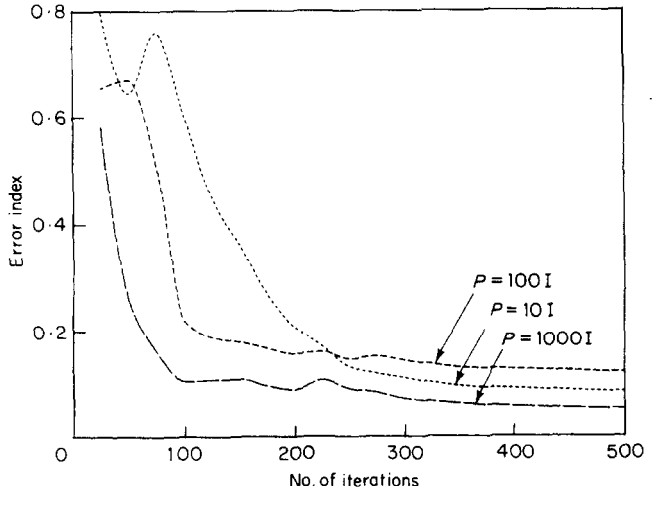
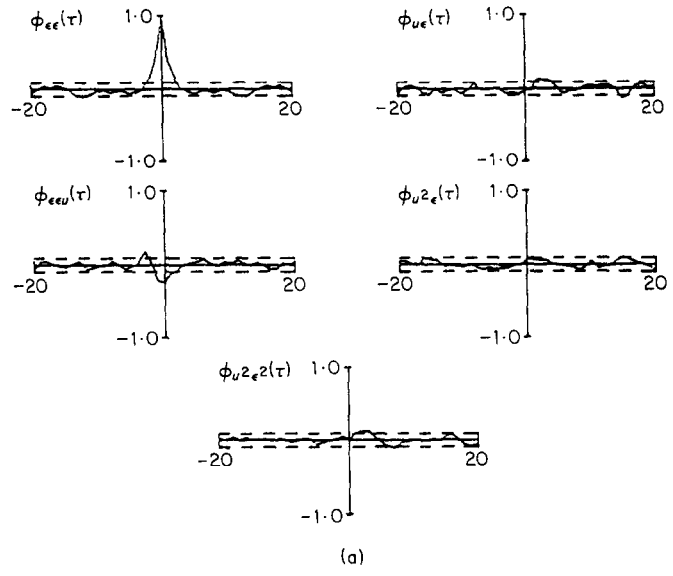


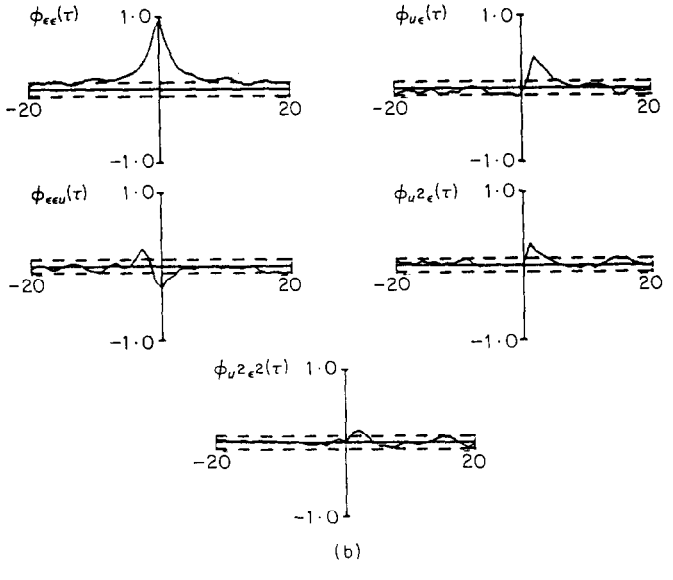
Figure 8. Effect of initial value of P on the evolution of error index for the RPE algorithm for simulation $S1$.

the cases A, B, C, D and E respectively. This shows the sensitivity of the backpropagation algorithm to the user selectable parameters.

For each example considered, a different combination of these parameters was needed. If an inappropriate choice was made, there was a possibility that the model would not produce a satisfactory prediction for the same amount of training data. This sensitivity can create difficulties whenever backpropagation routine is employed.



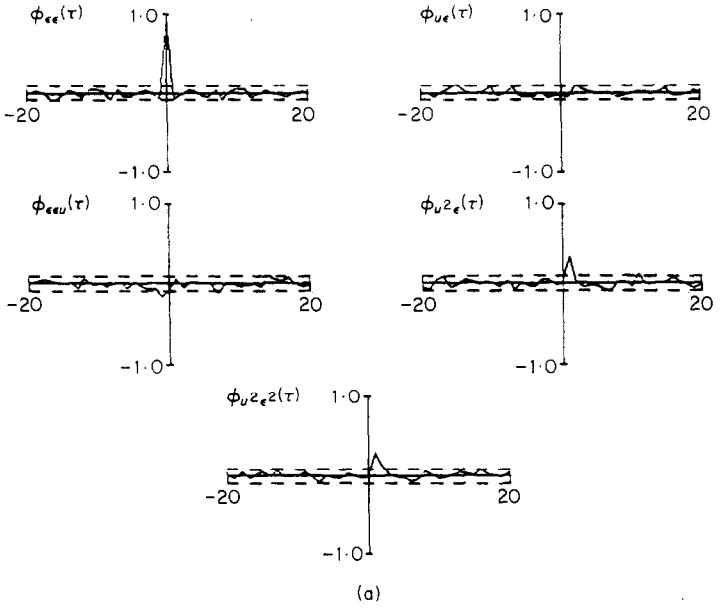
(a)



(b)

Figure 9. Correlation tests for simulation $S1$ using (a) the RPE algorithm and (b) the backpropagation algorithm.

It was also difficult to satisfy the model validity tests of equation (50) using the backpropagation routine. Figures 9(b), 10(b) and 11(b) illustrate the tests for simulations $S1$, $S2$ and $S3$ respectively using the backpropagation algorithm. Whilst neither RPE nor backpropagation satisfy these tests completely the results for the latter algorithm are inferior to those for RPE. Larger networks may well have produced improved results but a longer training period will be required. The selection of the number of nodes in a network is still an open question which will not be addressed in the current study.



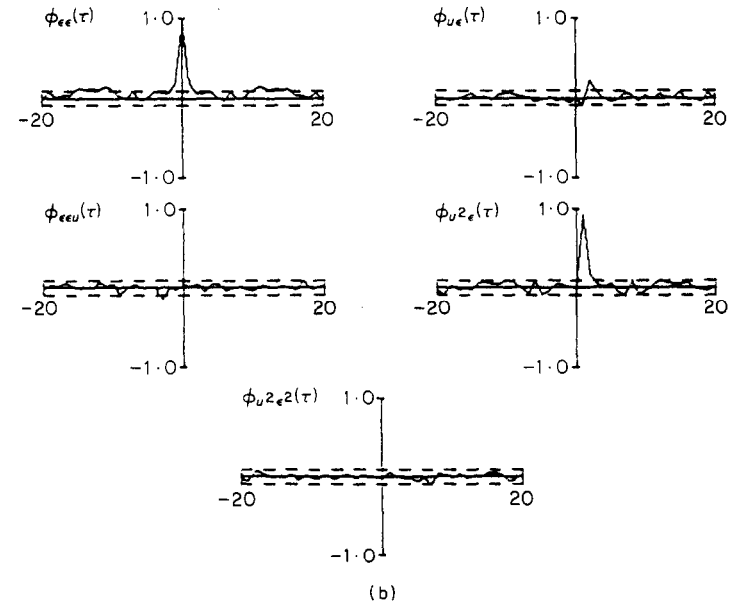
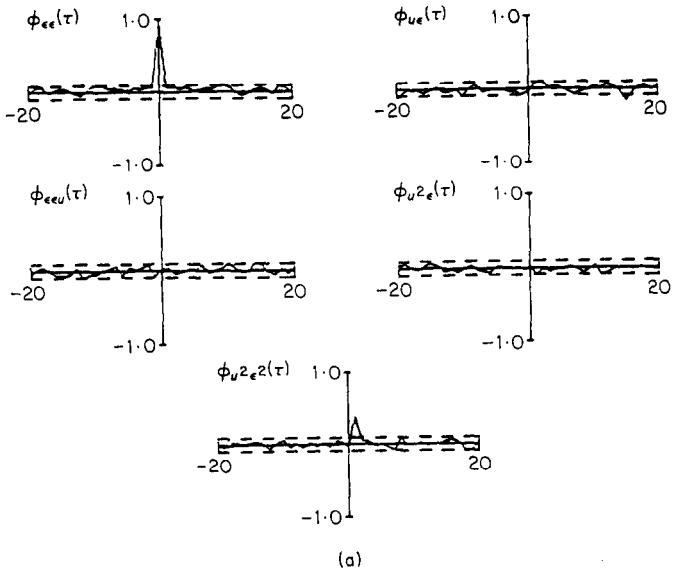


Figure 10. Correlation tests for simulation *S2* using (a) the RPE algorithm and (b) the backpropagation algorithm.



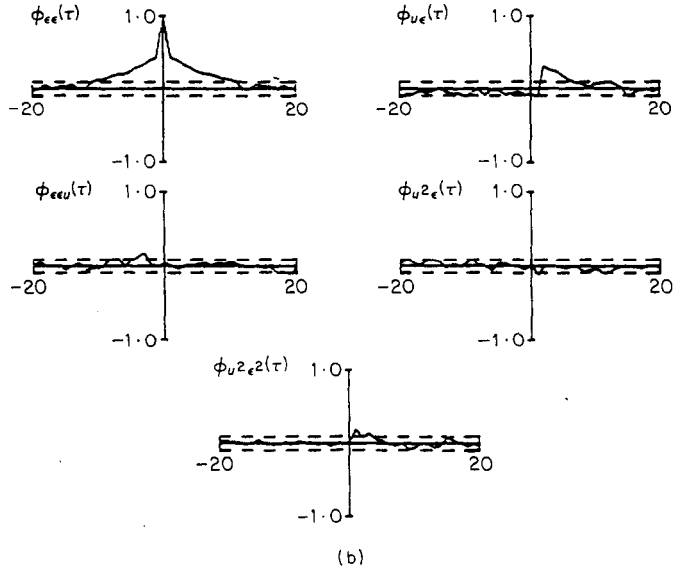


Figure 11. Correlation tests for simulation *S*3 using (a) the RPE algorithm and (b) the backpropagation algorithm.

For the RPE algorithm, it is known that while the choice of the initial covariance matrix $P(0) = \rho I$, $\rho > 0$ affects the initial stage of parameter estimation it should not significantly influence the final parameter estimates. A larger ρ will generally produce a more rapid change at the initial stages. These expectations were confirmed in the simulation as shown

in Fig. 8. The recursive prediction error algorithm produced model validity tests which were closer to the 95% confidence limits for the same structure of network and gave better predictions compared to the backpropagation algorithm as indicated in Figs 4 to 11. These advantages of the RPE algorithm are achieved at the expense of more computations at each iteration. Notice that the RPE algorithm described above can be reformulated into a parallel algorithm [13] so that the parallel computing potential of the network is exploited whilst maintaining the efficiency of RPE.

5.2. CANADIAN LYNX DATA

The fourth example concerns the modelling of a time series which representing Canadian lynx data. A neural network model with $n_0 = 7$ and $n_1 = 5$ was fitted using both the backpropagation and the RPE algorithms. The input vector in this case is

$$x(t) = [y(t-1) \quad y(t-2) \quad \ldots \quad y(t-7)]^T.$$

The lynx data consists of the annual record of the numbers of Canadian lynx trapped in the Mackenzie River district of North-Western Canada. This data was collected over a period of 114 years between 1821–1934 [20]. It has been noticed that the data exhibits an apparently regular 10 year cycle. Since the peaks of the data are sharp, while the troughs are relatively smooth, Moran [21] recommended that the logarithm to the base 10 of the data should be used for estimation.

For the lynx data, the backpropagation routine failed to give a reasonable model using the data set consisting of 114 data points although many combinations of learning rates, momentum constants and initial weights and thresholds were tried. This failure was probably due to the slow convergence of the backpropagation routine. A length of 114 data points might not be enough for the backpropagation algorithm to train such a network of 45 parameters so that significant changes to the initial parameter values were obtained.

Figure 12 shows the logarithmically transformed lynx data, the one step ahead prediction and the residuals obtained from the model trained by the RPE algorithm using the data set of 114 points. The one step ahead prediction superimposed on the actual data is illustrated in Fig. 13, where it is shown that the resulting model was capable of predicting the observations reasonably well. Various alternative network structure were trained but no significant improvement was achieved.

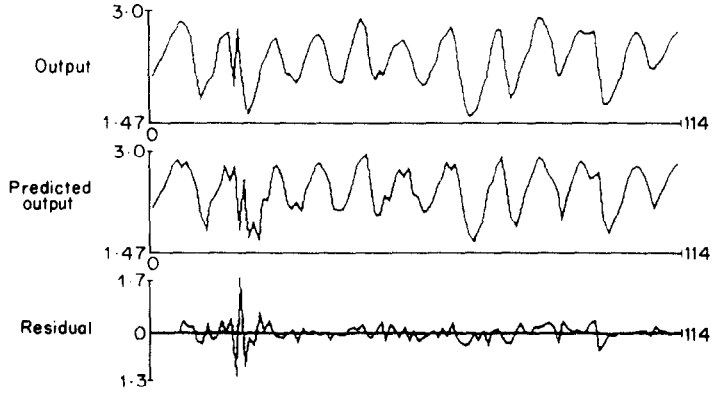


Figure 12. Output, one step ahead prediction and residual for the lynx data using the RPE algorithm.

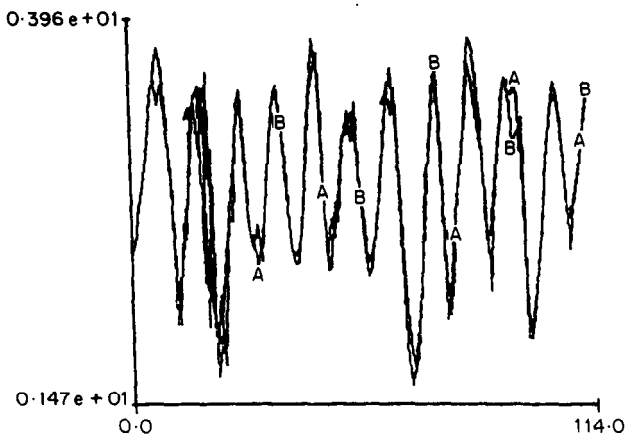


Figure 13. One step ahead prediction superimposed on observed lynx data using the RPE algorithm.

5.3. A TURBOCHARGED AUTOMOTIVE DIESEL ENGINE

The fifth example involves the identification of a model of a turbocharged automotive diesel engine. A neural network with $n_0 = n_1 = 5$ was used to model this system. The input vector was defined by

$$x(t) = [u(t-1) \quad u(t-2) \quad u(t-3) \quad y(t-1) \quad y(t-2)]^T.$$

The description of the experimental procedure and data collection are described in [22]. The low engine speed data set, b6shv2.dta was used in the present analysis. The input to the system was fuel rack position and the observed output was engine speed. The fuel rack was perturbed by a square wave signal of large amplitude with a superimposed PRBS.

The input and output data together with the predicted outputs and the residuals of the model trained by the RPE algorithm are shown in Fig. 14. Figure 15 shows the one step

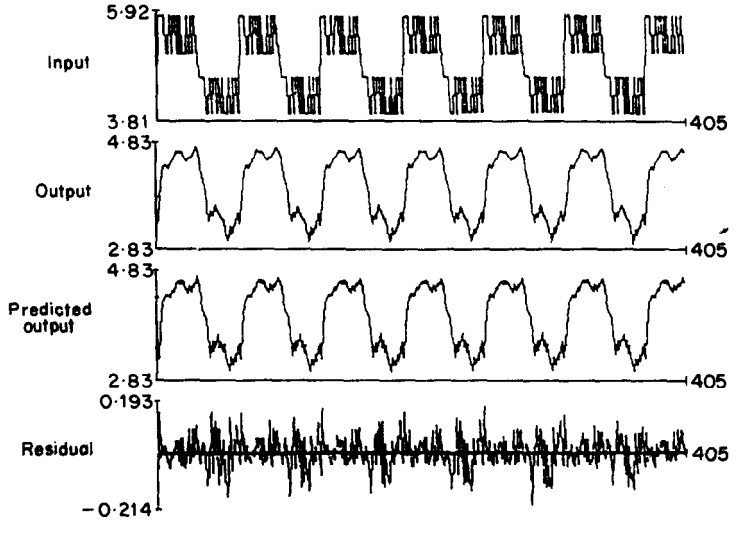


Figure 14. Input, output, one-step-ahead prediction and residual for the diesel engine. The model was trained using the RPE algorithm.

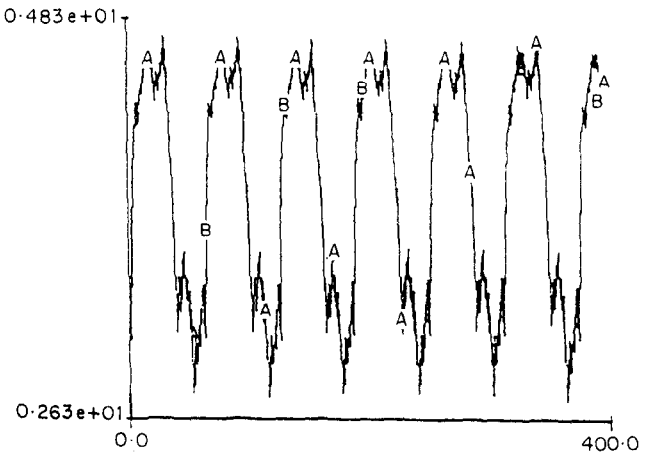


Figure 15. One step ahead prediction superimposed on observed output for the diesel engine using the RPE algorithm.

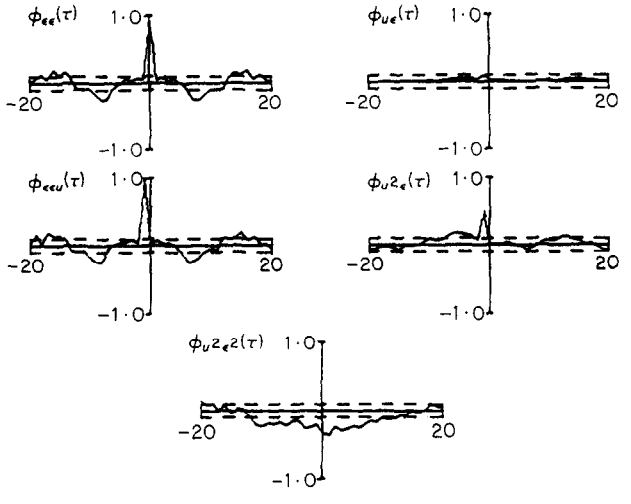


Figure 16. Correlation tests for the diesel engine using a neural network with five hidden and five input neurons using the RPE algorithm.

ahead predicted output superimposed on the actual output. The correlation validity tests shown in Fig. 16 suggest the model is inadequate. Increasing the number of hidden neurons from five to seven gave the validity tests shown in Fig. 17 which compared with Fig. 16 show no significant improvement. However on increasing the number of input nodes from five to seven that is the input vector was now defined by

$$x(t)=[u(t-1) \quad u(t-2) \quad u(t-3) \quad u(t-4) \quad y(t-1) \quad y(t-2) \quad y(t-3)]^T$$

the correlation tests were improved as shown in Fig. 18. The data set was shown to be highly non-linear [22]. Four hundred points of data might not be sufficient for the RPE algorithm to obtain a finely tuned parameter estimate so that the model validity tests are all within the confidence bands. Presenting the same set of data to the network for a second time resulted with more acceptable model validation plots of Fig. 19. It is therefore

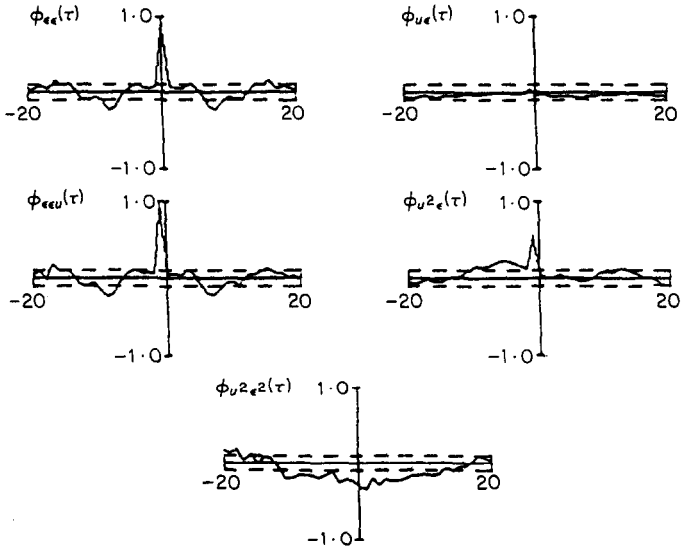


Figure 17. Correlation tests for the diesel engine using a neural network with seven hidden and five input neurons using the RPE algorithm.

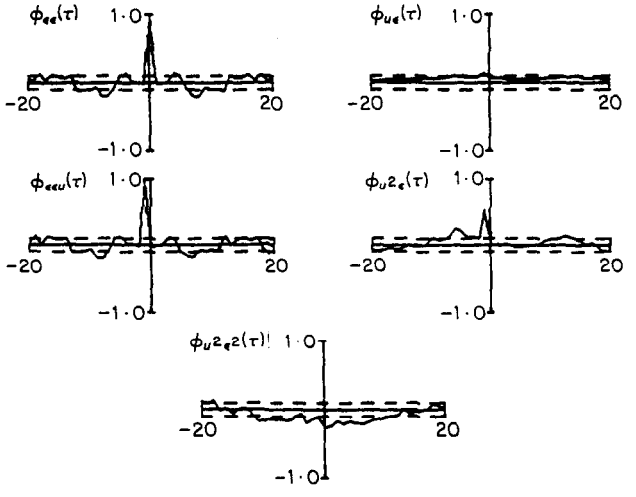


Figure 18. Correlation tests for the diesel engine using a neural network with five hidden and seven input neurons using the RPE algorithm.

not surprising that for this system, despite many attempts, the backpropagation routine failed to produce a reasonable estimate.

A NARMAX polynomial model has been estimated for the diesel engine but a direct comparision of these results with the neural network model are not possible because a noise model was estimated as part of the NARMAX analysis.

6. CONCLUSIONS

A new recursive prediction error estimator (RPE) has been compared with backpropagation as a method of training neural network models of non-linear systems. It has been demonstrated that the recursive prediction error algorithm often yields better predictions

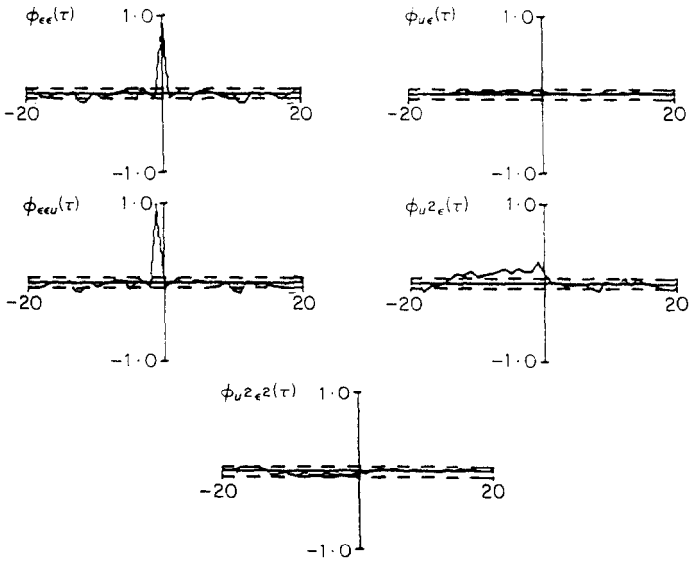


Figure 19. Correlation tests for the diesel engine using a neural network with five hidden and seven input neurons using the RPE algorithm after a second presentation of the data.

and faster convergence than backpropagation. The backpropagation algorithm appears to be dependent on the user selectable parameters to the extent that if an inappropriate combination of the learning rates and momentum constants are chosen the algorithm performs badly. The new RPE algorithm appears to provide an effective method of training neural networks at the expense of increased computational load compared with back-propagation.

REFERENCES

1. R. P. LIPPMAN 1987 *IEEE ASSP Magazine.* An Introduction to computing with neural nets.
2. P. Z. MARMARELIS and V. Z. MARMARELIS 1978 *White Noise Analysis of Physiological Systems* New York: Plenum.
3. I. J. LEONTARITIS and S. A. BILLINGS 1985 *International Journal of Control* **41,** 303-344. Input-output parametric models for non-linear systems, Part I: Deterministic non-linear systems; Part II: Stochastic non-linear systems.
4. A. LAPEDES and R. FARBER 1987 *Preprint LA-UR*-87-2662. Nonlinear signal processing using neural networks: prediction and system modelling.
5. S. CHEN, S. A. BILLINGS and P. M. GRANT 1990 *International Journal of Control,* **51,** 1191-1214. Non-linear systems identification using neural networks.
6. G. CYBENKO 1989 *Mathematics of Control, Signals and Systems,* **2,** 303-314. Approximations by superpositions of a sigmoidal function.
7. K. FUNAHASHI 1989 *Neural Networks* **2,** 183-192. On the approximate realization of continuous mappings by neural networks.
8. T. SEJNOWSKI and C. R. ROSENBERG 1986 *Johns Hopkins University Technical Report JHU/EECS*-86/01. NETtalk: a parallel network that learns to read aloud.
9. R. B. ALLEN 1987 *IEEE First International Conference on Neural Networks* **2,** 335-341. Several studies on natural language and backpropagation.
10. D. E. RUMELHART and J. L. MCCLELLAND (Eds.) 1986 *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Vol. 1 *Foundations,* Cambridge, MA: MIT Press.
11. M. L. BRADY, R. RAGHAVAN and J. SLAWNY 1989 *IEEE Transactions on Circuits and Systems,* **36,** 665-674. Backpropagation fails to separate where perceptrons succeed.
12. R. S. SUTTON 1986 *8th Annual Conference of the Cognitive Science Society* pp. 823-831. Hillsdale, NJ: Lawrence Erlbaum. Two problems with backpropagation and other steepest-descent learning procedures for networks.

13. S. CHEN, C. F. N. COWAN, S. A. BILLINGS and P. M. GRANT 1990 *International Journal of Control* **51**, 1215–1228. A parallel recursive prediction error algorithm for training layered neural networks.
14. L. LJUNG 1977 *IEEE Transactions on Automatic Control* **AC–22**, 551–575. Analysis of recursive stochastic algorithms.
15. B. WIDROW and S. D. STEARNS 1985 *Adaptive Signal Processing*, Englewood Cliffs, NJ: Prentice Hall.
16. L. LJUNG 1981 *Automatica* **17**, 89–99. Analysis of a general recursive prediction error identification algorithm.
17. L. LJUNG and T. SODERSTROM 1983 *Theory and Practice of Recursive Identification.* Cambridge, MA: MIT Press.
18. S. CHEN and S. A. BILLINGS 1989 *International Journal of Control* **49**, 569–594. Recursive prediction error parameter estimator for non-linear models.
19. S. A. BILLINGS and W. S. F. VOON 1986 *International Journal of Control* **44**, 235–244. Correlation based model validity tests for non-linear models.
20. T. S. RAO and M. M. GABR 1984 *An Introduction to Bispectral Analysis and Bilinear Time Series Models.* Berlin: Springer-Verlag.
21. P. A. P. MORAN 1953 *Australian Journal of Zoology* **1**, 163–173; 291–298. The statistical analysis of the Canadian lynx cycle.
22. S. A. BILLINGS, S. CHEN and R. J. BACKHOUSE 1989 *Mechanical Systems and Signal Processing* **3**, 123–142. The identification of linear and non-linear models of a turbocharged automotive diesel engine.