Model-Based Recurrent Neural Network for Modeling Nonlinear Dynamic Systems

Chengyu Gan and Kourosh Danai

Abstract—**A model-based recurrent neural network (MBRNN) is introduced for modeling dynamic systems. This network has a fixed structure that is defined according to the linearized state-space model of the plant. Therefore, the MBRNN has the ability to incorporate the analytical knowledge of the plant in its formulation. With its original topology intact, the MBRNN can then be trained to represent the plant nonlinearities through modifying its nodes' activation functions, which consist of contours of Gaussian radial basis functions (RBF's). Training in MBRNN involves adjusting the weights of the RBF's so as to modify the contours representing the activation functions. The performance of the MBRNN is demonstrated via several examples. The results indicate that it requires much shorter training than needed by ordinary recurrent networks. This efficiency in training is attributed to the MBRNN's fixed topology which is independent of training.**

I. Introduction

The need for modeling complex industrial systems demands modeling methods that can cope with high dimensionality, nonlinearity, and uncertainty. As such, alternatives to traditional linear and nonlinear modeling methods are needed. One such alternative is neural network modeling. Artificial neural networks are powerful empirical modeling tools that can be trained to represent complex multi-input multi-output nonlinear systems. Neural networks are also pattern classifiers, so they provide robustness to parameter variations and noise.

Various types of neural networks have been used for modeling dynamic systems. Multi-layer perceptrons have been used to provide an input-output representation of the plant in auto-regressive moving average (ARMA) form for cases where the past values of the plant inputs and outputs are available as network inputs [1]. Another network used for modeling dynamic systems is the cerebellar model articulation controller (CMAC) [2], noted for its ability to represent the spacial aspects of the system and for its fast learning that is desired for on-line applications [3]. Multilayer perceptrons and CMAC networks, however, are feedforward networks that only provide static mapping between inputs and outputs. A more suitable network for representing dynamic systems is the recurrent network which has the inherent format to internally represent the autoregressive aspect of dynamic systems [4]–[7].

Regardless of their type, however, neural networks are generally disadvantaged by their "black box" format. They need training for defining their structure (number of nodes) as well as their connection weights [7]. Therefore, these networks require prohibitively extensive training, and are hard to interpret once trained. One variant to these networks is the diagonal recurrent neural network proposed by Ku and Lee [8] that provides a simpler structure than the fully connected network, and therefore is an easier network to train. Another variant is a simplified network that is initiated based on representative patterns in the training data [9].

While the above solutions have led to reduced demand for training, they have not been significant, mostly due to their reliance on empiricism. Ideally, neural networks should be less empirical, to reduce the demand for training, and more transparent, to provide insight into their representation. A step toward such an ideal solution for modeling dynamic systems is using sub-networks to represent separately the autoregressive and moving average parts of the plant, and selecting the complexity of these networks according to linearity/nonlinearity of these parts [10]. Although this modular solution takes advantage of the existing knowledge of the plant to reduce the complexity of the overall network and, consequently, reduces the demand for training, it does not contribute to improving the transparency of the network, as it still uses a black box format for the individual sub-networks.

Perhaps a more fundamental step towards the ideal solution has been in the area of neuro-fuzzy inference systems [11], [12] and knowledge-based artificial neural networks (KBANN) [13], [14]. In the neuro-fuzzy approach, the heuristic knowledge of the process is expressed in the form of rules between inputs and outputs, and incorporated in a feedforward network with several layers, where each layer performs a step of inference. An important feature of neuro-fuzzy networks is their equivalence to Gaussian radial basis function (RBF) networks [11], which makes possible the use of learning algorithms associated with RBF networks to modify the rules. KBANN's are formulated by symbolic rules in the form of propositional logic [13]. Both the neuro-fuzzy system and KBANN provide methods for incorporating the existing knowledge of the plant in a neural network. However, they require that this knowledge be formulated as heuristics or propositional logic. Given that most plant models are developed from first principles and are in analytical form, these methods would either ignore the existing analytical models or require the extra step of converting these models into heuristics or propositional logic.

The objective of this paper is to introduce a method of incorporating the analytical knowledge of the plant in a recurrent neural network. This model-based recurrent neural network (MBRNN) will be formulated according to a linearized state-space model of the dynamic system. It will then be trained to adapt its activation functions to the nonlinearities of the plant as reflected in the training data. As such, the topology of the MBRNN will remain intact, and adaptation to nonlinearities will be confined to the activation functions of individual nodes. The activation functions in MBRNN are defined as contours of the RBF's that comprise the node, therefore, adaptation entails changing the weight of individual RBF's within each node.

II. Methodology

The MBRNN is structured according to a linearized state-space model of the plant. It, therefore, assumes that an analytical model of the plant is available which can be linearized about an operating point. If the discrete-time nonlinear state-space model of the plant is defined as

$$\mathbf{x}(k+1) = \Phi[\mathbf{x}(k), \mathbf{u}(k)] \tag{1}$$

$$\mathbf{y}(k) = \Psi[\mathbf{x}(k), \mathbf{u}(k)] \tag{2}$$

where $\mathbf{u}(k)$, $\mathbf{x}(k)$, and $\mathbf{y}(k)$ represent the sampled values of the inputs, states, and outputs at time k, respectively, then the linearized state-space model has the form

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{3}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k). \tag{4}$$

Manuscript received December 25, 1998; revised November 20, 1999. This work was supported by the National Science Foundation under Grant CMS-9523087. This paper was recommended by Associate Editor S. Lakshmivarahan.

The authors are with the Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA 01003 USA.

Publisher Item Identifier S 1083-4419(00)02972-1.

The above model can be formulated as a recurrent neural network, having the same number of inputs and outputs. An example of such a network is shown in Fig. 1 for the second-order model

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} u(k) \quad (5)$$

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} u(k). \quad (6)$$

Note that the nodes of this initial MBRNN have a gain of 1 (i.e., their output equals their input) to initially represent the linearized model (see Fig. 1), but the activation functions of these nodes are subsequently adapted during training into nonlinear functions so as to represent the plant nonlinearity. As such, adaptation in MBRNN is performed by only changing the form of the activation functions, leaving the connection weights intact. In order to provide adaptability, the activation functions in MBRNN are defined as contours of RBF's that comprise the node. If the output of each RBF is defined as

$$o_i = \exp(-|x_j - c_i|^2/\sigma^2) \quad (7)$$

to represent a normal distribution with localized characteristics, then the activation function of the node will have the form

$$O_j = \sum_{i=1}^{N} \theta_i o_i = \sum_{i=1}^{N} \theta_i \exp(-|x_j - c_i|^2/\sigma^2). \quad (8)$$

While the activation function O_j can be composed of an array of any basis function that can be adapted on-line, it is desirable that the basis function should have a localized characteristic so that the activation function can be modified locally without affecting the value of the function at neighboring points.

The above formulation represents a modular format for MBRNN where each module consists of an RBF network denoting a node. Training consists of adjusting the weights θ_i of individual RBF's to shape the node's activation function. The above second-order network with an adaptable set of activation functions has the form

$$\begin{Bmatrix} x_1(k+1) \\ x_2(k+1) \end{Bmatrix} = \begin{bmatrix} a_{11} \sum_{i=1}^{N_{a11}} \theta_i^{a11} \exp\left(-|x_1(k) - c_i^{a11}|^2/\sigma^2\right) \\ \quad + a_{12} \sum_{i=1}^{N_{a12}} \theta_i^{a12} \exp\left(-|x_2(k) - c_i^{a12}|^2/\sigma^2\right) \\ a_{21} \sum_{i=1}^{N_{a21}} \theta_i^{a21} \exp\left(-|x_1(k) - c_i^{a21}|^2/\sigma^2\right) \\ \quad + a_{22} \sum_{i=1}^{N_{a22}} \theta_i^{a22} \exp\left(-|x_2(k) - c_i^{a22}|^2/\sigma^2\right) \end{bmatrix} + \begin{bmatrix} b_1 \sum_{i=1}^{N_{b1}} \theta_i^{b1} \exp\left(-|u(k) - c_i^{b1}|^2/\sigma^2\right) \\ b_2 \sum_{i=1}^{N_{b2}} \theta_i^{b2} \exp\left(-|u(k) - c_i^{b2}|^2/\sigma^2\right) \end{bmatrix} \quad (9)$$

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} c_{11} \sum_{i=1}^{N_{c11}} \theta_i^{c11} \exp\left(-|x_1(k) - c_i^{c11}|^2/\sigma^2\right) \\ \quad + c_{12} \sum_{i=1}^{N_{c12}} \theta_i^{c12} \exp\left(-|x_2(k) - c_i^{c12}|^2/\sigma^2\right) \\ c_{21} \sum_{i=1}^{N_{c21}} \theta_i^{c21} \exp\left(-|x_1(k) - c_i^{c21}|^2/\sigma^2\right) \\ \quad + c_{22} \sum_{i=1}^{N_{c22}} \theta_i^{c22} \exp\left(-|x_2(k) - c_i^{c22}|^2/\sigma^2\right) \end{bmatrix} + \begin{bmatrix} d_1 \sum_{i=1}^{N_{d1}} \theta_i^{d1} \exp\left(-|u(k) - c_i^{d1}|^2/\sigma^2\right) \\ d_2 \sum_{i=1}^{N_{d2}} \theta_i^{d2} \exp\left(-|u(k) - c_i^{d2}|^2/\sigma^2\right) \end{bmatrix}. \quad (10)$$

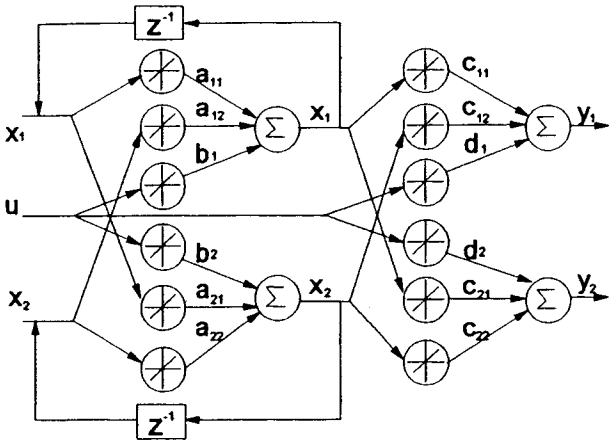

Fig. 1. Model-based recurrent neural network (MBRNN) representing a second-order plant.

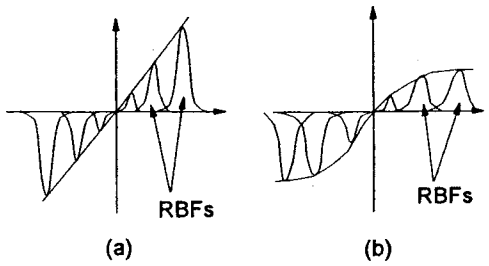

Fig. 2. Initial linear activation function of MBRNN formed from the contour of RBF's in the node (a), and the trained activation function obtained by modifying the weights of various RBF's (b).

The RBF's in each node of MBRNN are evenly distributed across the expected range of state or input space. For the initial MBRNN, the values of parameters θ_i are selected to produce linear activation functions as their contours, so as to depict the linearized model of the plant. Subsequently, these parameters are modified during a training phase to adapt the MBRNN to plant nonlinearities. Note that MBRNN is formulated such that the activation functions are linear in terms of parameters θ_i, so training is tractable. Based on (9) and (10), if the states are measurable, then training of the parameters associated with each of the states and inputs can be carried out separately by the least-squares method. In cases where the states are not accessible, as is often the case, alternative forms of training such as dynamic backpropagation or extended Kalman filtering can be used. In either case, after training the activation functions may have forms very different from their initial linear form, as shown in Fig. 2 for a hypothetical case. The construction of the MBRNN's nodes by normally distributed RBF's provides it with the distinct characteristic of "spacial localization" [15]. Because of one-to-one relationship between input-output pairs, each adaptation iteration in MBRNN is confined to a limited number of θ_i that are associated with the excitation input. This, in addition to providing efficient training, enables MBRNN to accommodate localized nonlinearities within limited ranges of the input space.

In its present form, the initial MBRNN represents the first-order approximation of the nonlinear plant, and it maintains this first-order representation while allowing the individual components of the linearized model to incorporate nonlinearity through training. As such, this format does not allow inclusion of bilinear terms. For example, if the actual x_1 in our example of (5) and (6) had the form $x_1(k+1) =$

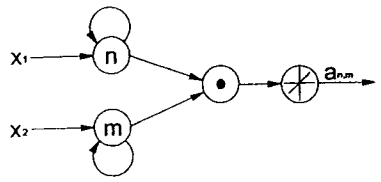

Fig. 3. Possible modification to MBRNN to accommodate coupling between x_1 and x_2. The nodes with circular arrows pointing to themselves represent the repetitive multiplication of the input variable, i.e., the node with x_1 as input to the node enclosing n would output x_1^n, and the node with a dot inside signifies the product of the input variables.

$F(x_1(k), x_2(k))$, and was then approximated by a Taylor Series Expansion, as

$$F(x_1, x_2) = F(x_1, x_2)|_0 + \sum_{r=0}^{1} \binom{1}{r} \frac{\partial F}{\partial x_1^r \partial x_2^{1-r}} \times (x_1, x_2)(x_1 - x_1|_0)^r (x_2 - x_2|_0)^{1-r} + \cdots + \sum_{r=0}^{n} \binom{n}{r} \frac{\partial^n F}{\partial x_1^r \partial x_2^{n-r}} \times (x_1, x_2)(x_1 - x_1|_0)^r (x_2 - x_2|_0)^{n-r} + \cdots$$

where $\binom{n}{r} = (n!/r!(n-r)!)$ denote the binomial coefficients and all the derivatives are evaluated at $(x_1, x_2)|_0$, then the MBRNN constructed according to the first-order approximation of function F, would exclude the coupling between x_1 and x_2, represented by $\sum_{r=1}^{n} \binom{n}{r} (\partial^n F / \partial x_1^r \partial x_2^{n-r})(x_1, x_2)$. Therefore, when this coupling is not negligible, MBRNN, in its present form, will not be able to approximate the nonlinear plant satisfactorily. In order to include such significant coupling terms, additional nodes may be added into the feedforward path of MBRNN to represent specific couplings. One stand-alone node that can represent a coupling term is shown in Fig. 3, where the nodes with circular arrows pointing to themselves represent the repetitive multiplication of the input variable (e.g., the node with x_1 as input to the node enclosing n would output x_1^n), and the node with a dot inside signifies the product of the input variables. Although the inclusion of such coupling terms in MBRNN is straightforward, it poses two constraints: 1) it requires a knowledge of the dominant coupling between the plant variables and 2) it adds to the complexity of MBRNN and its demand for learning.

A central issue in MBRNN is the number of RBF's needed within each node. Initially, the activation functions should be linear with slopes of 1 so as to provide a uniform gain of unity at all inputs for an accurate representation of $x = y$. A perfect linear activation function, however, would require a large number of RBF's. Therefore, a method needs to be devised to limit the number of RBF's within the expected ranges of inputs and states. One method for selecting the number of RBF's is cross validation, where the available input-output data is divided into two parts, one part used for training and the other for testing. The selection criterion in this case is the generalization ability of the network based on the test set [16], [17]. For each modular RBF network representing a node, the average of mean-square-error over the training set can be defined as [18]

$$\hat{\sigma}^2_{\mathrm{GCV}_j} = \frac{p \hat{\mathbf{y}}_j^T \mathbf{P}_j^2 \hat{\mathbf{y}}_j}{\mathrm{trace}(\mathbf{P}_j)^2} \tag{11}$$

where

$\hat{\sigma}^2_{\mathrm{GCV}_j}$ generalized cross-validation (GCV);

p number of patterns used in training;

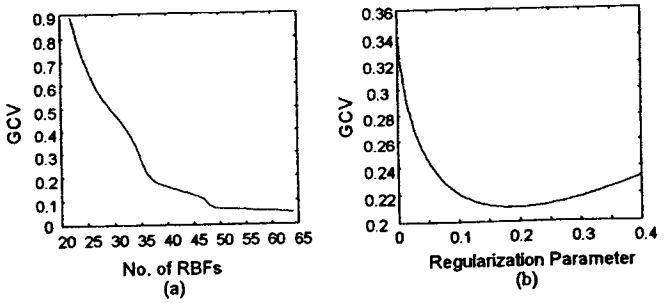

Fig. 4. Effect of the number of RBF's (a) and the regularization parameter (b) on the GCV value.

$\mathbf{P}_j$ square matrix which projects p-dimensional vectors onto the m_j-basis subspace spanned by the RBF's within the node;

$\hat{\mathbf{y}}_j$ output of the node.

Since each node in MBRNN can be perceived as a separate RBF network, GCV can be used to determine the number of RBF's within each MBRNN node. For this, each set of outputs from the p patterns corresponding to each node is projected to the m_j-dimension spanned by the RBF's within that node, so as to represent its optimal representation in least-squares sense. The projection matrix $\mathbf{P}_j$ for MBRNN is defined as

$$\mathbf{P}_j = \mathbf{I}_p - \mathbf{O}_j \mathbf{A}_j^{-1} \mathbf{O}_j^T \tag{12}$$

where $\mathbf{O}_j$ represents the matrix of RBF's o_i for each node [see (7)]

$$\mathbf{O}_j = \begin{bmatrix} o_1(s_1) & o_2(s_1) & \cdots & o_{m_j}(s_1) \\ o_1(s_2) & o_2(s_2) & \cdots & o_{m_j}(s_2) \\ \vdots & \vdots & \ddots & \vdots \\ o_1(s_p) & o_2(s_p) & \cdots & o_{m_j}(s_p) \end{bmatrix} \tag{13}$$

where s_i represents the input value associated with pattern i, and $\mathbf{A}_j^{-1}$ denotes the variance matrix of $\mathbf{O}_j$ obtained as

$$\mathbf{A}_j^{-1} = \left(\mathbf{O}_j^T \mathbf{O}_j + \lambda_j \mathbf{I}_\mathbf{p} \right)^{-1}. \tag{14}$$

In the above equation, λ_j denotes the regularization parameter associated with each node. Regularization parameters are integral parts of training the activation functions by dynamic backpropagation, so they are included in the formulation of GCV to define the rationale for their selection. GVD is a measure of the network's ability in mapping a set of inputs to a desired set of outputs. As such, GCV can be used to determine the value of the regularization parameter and the number of RBF's for each node to provide an accurate representation of $y = x$ initially. The typical values of GCV obtained for different numbers of RBF's, with a constant regularization parameter, are shown in Fig. 4(a). The results indicate that the increasing number of RBF's has a diminishing effect on the GCV value, so the GCV value can be used as the basis of selecting the RBF numbers. Similarly, the effect of the regularization parameter on the GCV value of the node, with a fixed number of RBF's, is shown in Fig. 4(b). The results indicate that the accuracy of each node's output (as represented by the GCV value) is directly affected by the regularization parameter, and that an "optimal" value for it can be selected based on the GCV value. Of course, one should note that there is a correlation between the number of RBF's and the value of regularization parameter, and that by determining each independent of the other there is no assurance that their optimal values have been selected. Experience, however, indicates that selecting the number of

RBF's and the regularization parameter separately leads to satisfactory results.

III. Training

An important feature of MBRNN is the linearity of its activation functions in terms of the RBF weights, which makes possible adapting the RBF weights by various linear regression methods. An important objective of MBRNN design is to adhere to its initial format which is structured according to the state-space model of the plant. In order to satisfy this objective, limited changes should be made to the shape of the activation functions during training, so as to 1) avoid drastic deviations from the initial structure of MBRNN and 2) avoid creation of limit cycles. Two adaptation methods that can implement such a restriction, based on dynamic backpropagation [19], [20] and extended Kalman filter (EKF) [7], [15], [21], are described here, to estimate the network parameters θ_Φ and θ_Ψ in the MBRNN representation of the nonlinear plant, as

$$\hat{\mathbf{x}}(k+1) = \Phi_{\text{net}}[\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi] \tag{15}$$

$$\hat{\mathbf{y}}(k) = \Psi_{\text{net}}[\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Psi] \tag{16}$$

where $\mathbf{x} \in \mathcal{R}^n$, $\mathbf{u} \in \mathcal{R}^m$, and $\mathbf{y} \in \mathcal{R}^q$.

A. Training by Dynamic Backpropagation

For the p input-output training pairs, the objective function can be defined as

$$J = \sum_{i=1}^{p} (\hat{\mathbf{y}}_i - \mathbf{y}_i)^2 \tag{17}$$

where $\hat{\mathbf{y}}$ denotes the vector of network outputs and $\mathbf{y}$ the corresponding plant outputs. Although this objective function is suitable for adaptation of the network parameters, it does not satisfy the goal of preserving its structure. This restriction on the extent of parameter changes can be imposed by including a regularization term in the objective function, as

$$J = \sum_{j=1}^{p} (\hat{\mathbf{y}}_j - \mathbf{y}_j)^2 + \sum_{j=1}^{N} \lambda_j \sum_{i=1}^{m_j} (\theta_i - \theta_{io})^2 \tag{18}$$

where

- N number of nodes in MBRNN;
- m_j number of RBF's in individual nodes;
- θ_i weight of each RBF during training;
- θ_{io} initial weight of each RBF;
- λ_j regularization parameter of each node [see (14)].

MBRNN consists of two parts (see Fig. 1): 1) a recurrent part, which emulates the state equations [(1)] and a 2) feedforward part, that represents the output equations [(2)]. Training of the recurrent part can be performed by backpropagation (BP) over time (dynamic BP), to estimate θ_Φ, whereas the the static part can be trained by regular BP, to estimate θ_Ψ. The objective of training is to determine the vector θ^* which minimizes the objective function J in (18) by moving the parameter vector along the negative gradient of the objective function with respect to θ, as [22]

$$\theta(k+1) = \theta(k) - \eta \nabla_\theta J(k) \tag{19}$$

where η denotes the learning rate, $\nabla_\theta J$ represents the gradient of the objective function, and $\theta(k)$ represents the current value of the RBF's weight vector.

For the dynamic part of MBRNN, the change in a weight at time k will result in a change in the output $\hat{\mathbf{y}}(t)$ for all $t \geq k$. This means that the present value of the output $\hat{\mathbf{y}}(t)$ is not only affected by the current value of the weight vector $\theta_\Phi(k)$, for $k = t$, but by all the past values of $\theta_\Phi(k)$, for $0 \leq k \leq t$, as well. This implies that weight adaptation should be performed with regard to past adaptations, as reflected in the estimate of the gradient of the objective function, computed from (15) and (16)

$$\frac{\partial \hat{\mathbf{x}}(k+1)}{\partial \theta_\Phi(k)} = \frac{\partial \Phi_{\text{net}}(\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi(k))}{\partial \hat{\mathbf{x}}(k)} \frac{\partial \hat{\mathbf{x}}(k)}{\partial \theta_\Phi(k)} + \frac{\partial \Phi_{\text{net}}(\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi(k))}{\partial \theta_\Phi(k)} \tag{20}$$

$$\frac{\partial \hat{\mathbf{y}}(k)}{\partial \theta_\Phi(k)} = \frac{\partial \Psi_{\text{net}}(\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi(k))}{\partial \hat{\mathbf{x}}(k)} \frac{\partial \hat{\mathbf{x}}(k)}{\partial \theta_\Phi(k)} + \frac{\partial \Psi_{\text{net}}(\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi(k))}{\partial \theta_\Phi(k)}. \tag{21}$$

The components of the above equations can be computed at each instant k to yield $\partial \hat{\mathbf{y}} / \partial \theta_\Phi$ as the gradient of the objective function with respect to the RBF weights. Dynamic BP can then be used to adapt the parameter vector θ_Φ according to the learning rule in (19). Note that (20) and (21) define a state-space representation of the gradient of the objective function $\nabla_{\theta_\Phi} J$ used in (19). The drawback of backpropogation is that it is often too slow for practical application. There are many variations on backpropogation that speed up the training, such as the Levenberg-Marquardt algorithm, BP with Momentum, etc. [23]. Also, in order to eliminate the spikes during training, a low-pass filter can be incorporated in training.

Another issue in training by backpropagation is the proper choice of the learning rate η [see (19)]. Usually small values of η guarantee convergence, but at a very low speed. Larger values of η, on the other hand, may lead to instability. It is possible to determine analytically the range of η to guarantee stability.

B. Training by the Extended Kalman Filter

The MBRNN can also be trained by the EKF [7], [15], which has the added appeal of being applicable to both the recurrent and feedforward parts of the network. EKF-based training is also faster than dynamic BP, and is less prone to noise.

In application to MBRNN, the network states are augmented with auxiliary states that correspond to the RBF weights. As such, EKF-based training of MBRNN consists of simultaneous parameter and state estimation. Since the network inputs lie in $\Re^{m+n}$ [see (15) and (16)], the augmented input space of the network $\mathbf{w}^T = [\mathbf{x}\ \mathbf{u}]$ belongs to $\mathcal{R}^{m+n}$. In this case, part of the augmented space $\mathbf{u}$ will be extraneous, and $\mathbf{x}$ will be constrained by the network dynamics. In the EKF algorithm, the system dynamics are first linearized about the current state and parameter estimates. The traditional Kalman filter is then used to update the state and parameter estimates of this linearized system.

The formulation of the EKF algorithm as applied to the MBRNN is as follows:

$$\hat{\mathbf{x}}(k+1) = \Phi_{\text{net}}[\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi] + \zeta(k) \tag{22}$$

$$\hat{\mathbf{y}}(k) = \Psi_{\text{net}}[\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Psi] + \nu(k) \tag{23}$$

where $\zeta(k)$ and $\nu(k)$ are zero-mean, independent Gaussian random variables representing the process and measurement noise, respectively. If the augmented parameter vector θ is defined as $\theta^T = [\theta_\Phi\ \theta_\Psi]$, and the augmented state vector $\mathbf{z}$ representing the unknown parameters

of the network as $\mathbf{z}^T = [\hat{\mathbf{x}}\ \theta]$, then the network model can be defined as

$$\begin{aligned}\mathbf{z}(k+1) &= \begin{bmatrix}\boldsymbol{\Phi}_{\text{net}}[\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Phi(k)]\\ \theta(k)\end{bmatrix} + \begin{bmatrix}\zeta(k)\\ \beta(k)\end{bmatrix}\\ &\equiv \mathbf{F}(\mathbf{z}(k), \mathbf{u}(k)) + \gamma(k) \qquad (24)\\ \hat{\mathbf{y}}(k) &= \boldsymbol{\Psi}_{\text{net}}(\hat{\mathbf{x}}(k), \mathbf{u}(k), \theta_\Psi(k)) + \nu(k)\\ &\equiv \mathbf{H}(\mathbf{z}(k), \mathbf{u}(k)] + \nu(k) \qquad (25)\end{aligned}$$

where $\beta(k)$ denotes a vector of fictitious white noise, and $\gamma(k) = [\zeta(k)\ \beta(k)]^T$. Based on the above representation, the covariance matrix $\mathbf{Q}(k)$ can be defined as

$$\mathbf{Q}(k) = \begin{bmatrix}\mathbf{Q}_x(k) & 0\\ 0 & \mathbf{Q}_\theta(k)\end{bmatrix} \qquad (26)$$

where $\mathbf{Q}_x(k) = E[\zeta(k)\ \zeta(k)^T]$ and $\mathbf{Q}_\theta(k) = E[\theta(k)\ \theta(k)^T]$. The above equations can now be linearized about some nominal $\bar{\mathbf{z}}(k)$ and $\bar{\mathbf{y}}(k)$ as

$$\begin{aligned}\delta\mathbf{z}(k+1) &= \mathbf{z}(k+1) - \bar{\mathbf{z}}(k+1)\\ &= \mathbf{F}(\mathbf{z}(k), \mathbf{u}(k)) + \mathbf{w}(k) - \mathbf{F}(\bar{\mathbf{z}}(k), \mathbf{u}(k))\\ &= \mathbf{F}_0(k)\delta\mathbf{z}(k) + \mathbf{w}(k) \qquad (27)\\ \delta\mathbf{y}(k+1) &= \hat{\mathbf{y}}(k+1) - \bar{\mathbf{y}}(k+1)\\ &= \mathbf{H}(\mathbf{z}(k), \mathbf{u}(k)) + \nu(k) - \mathbf{H}(\bar{\mathbf{z}}(k), \mathbf{u}(k))\\ &= \mathbf{H}_0(k)\delta\mathbf{y}(k) + \nu(k) \qquad (28)\\ \mathbf{F}_0(k) &= \left.\frac{\partial\mathbf{F}}{\partial\mathbf{z}}\right|_{\bar{\mathbf{z}}(k),\mathbf{u}(k)} = \begin{bmatrix}\partial\mathbf{F}/\partial\mathbf{x} & \partial\mathbf{F}/\partial\theta\\ \partial\theta/\partial\mathbf{x} & \partial\theta/\partial\theta\end{bmatrix}\\ &= \begin{bmatrix}\partial\mathbf{F}/\partial\mathbf{x} & \partial\mathbf{F}/\partial\theta\\ 0 & \mathbf{I}\end{bmatrix} \qquad (29)\end{aligned}$$

where and

$$\mathbf{H}_0(k) = \left.\frac{\partial\mathbf{H}}{\partial\mathbf{z}}\right|_{\bar{\mathbf{z}}(k),\mathbf{u}(k)} = [\partial\mathbf{H}/\partial\mathbf{x}\quad \partial\mathbf{H}/\partial\theta]. \qquad (30)$$

If the *a priori* and *posteriori* estimates of the variable $\mathbf{z}$ at time k are denoted by $\mathbf{z}(k \mid k-1)$ and $\mathbf{z}(k \mid k)$, respectively, and the *a priori* and *posteriori* state covariance estimates denoted by $\mathbf{P}(k \mid k-1)$ and $\mathbf{P}(k \mid k)$, then the EKF algorithm at time k is composed of the following steps.

1) Compute the linearized dynamics $\mathbf{F}_0(k-1)$ and $\mathbf{H}_0(k-1)$ by linearizing $\mathbf{F}$ and $\mathbf{H}$ about the previous *posteriori* estimate $\mathbf{z}(k-1 \mid k-1)$.
2) Compute the *a priori* state estimate:

$$\mathbf{z}(k \mid k-1) = \mathbf{F}(\mathbf{z}(k-1 \mid k-1), \mathbf{u}(k-1)).$$

3) Compute the *a priori* state noise covariance estimate:

$$\mathbf{P}(k \mid k-1) = \mathbf{F}_0(k-1)\mathbf{P}(k-1 \mid k-1)\mathbf{F}_0^T(k-1) + \mathbf{Q}(k).$$

4) Compute the Kalman gain matrix $\mathbf{K}(k)$ as:

$$\mathbf{K}(k) = \mathbf{P}(k \mid k-1)\mathbf{H}_0^T(k-1)[\mathbf{H}_0(k-1) \times \mathbf{P}(k \mid k-1)\mathbf{H}_0^T(k-1) + \mathbf{R}]^{-1}.$$

5) Compute the *posteriori* state estimate by using the new observation $\mathbf{y}(k)$:

$$\mathbf{z}(k \mid k) = \mathbf{z}(k \mid k-1) + \mathbf{K}(k)(y(k) - \mathbf{H}(\mathbf{z}(k \mid k-1), \mathbf{u}(k))).$$

6) Update the state noise covariance:

$$\mathbf{P}(k \mid k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H}_0(k-1))\mathbf{P}(k \mid k-1).$$

In the above algorithm, the initial gain matrix $\mathbf{P}(0)$ reflects the level of confidence in the initial parameters. As such, it is set to a high value for highly nonlinear systems which are not closely approximated by the linearized model. As learning progresses, the gain matrix decreases in value, and the learning rate is controlled more by the level of noise reflected in $\mathbf{Q}$ and $\mathbf{R}$. A large $\mathbf{R}$ is indicative of little confidence in the measurement values, so learning is performed more slowly, whereas a large $\mathbf{Q}$ is indicative of smaller confidence in the parameter values, causing learning to be performed more aggressively. As such, the overall rate of convergence in EKF-based training is controlled by the relative values of $\mathbf{Q}$ and $\mathbf{R}$.

IV. Performance Evaluation

The effectiveness of MBRNN has been demonstrated for several examples. The first example is taken from the literature [10] so as to facilitate a comparison between MBRNN and other neural networks in terms of modeling effectiveness and training effort.

Example 1:

$$y(k+1) = f[y(k), y(k-1)] + u(k) \qquad (31)$$

where

$$f[y(k), y(k-1)] = \frac{y(k)y(k-1)[y(k)+2.5]}{1+y^2(k)+y^2(k-1)} \qquad (32)$$

This example represents a single-input single-output plant, with a nonlinear autoregressive part. The linearized state-space form of this model about the operating point $(x_1 = 0, x_2 = 0)$ was obtained as

$$\begin{Bmatrix}x_1(k+1)\\ x_2(k+1)\end{Bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{Bmatrix}x_1(k)\\ x_2(k)\end{Bmatrix} + \begin{Bmatrix}0\\ 1\end{Bmatrix}u \qquad (33)$$

$$y(k) = [0\quad 1]\begin{Bmatrix}x_1(k)\\ x_2(k)\end{Bmatrix}. \qquad (34)$$

The MBRNN was structured according to the above state-space model and then trained by EKF using random excitation with values between -0.6 and 0.6. The initial structure of MBRNN for this example was quite similar to that of Fig. 1, except that this MBRNN had one output and did not need to include nodes associated with d_1 and d_2. One epoch of training was used which consisted of 600 sample points. The trained MBRNN was then tested by data generated with random inputs from a different seed than that which was used during testing. The values of $\hat{y}$ and y for the test session before and after training are shown in Fig. 5. The results indicate that the output of MBRNN is much closer to the plant output after training, despite the relatively short training session used for MBRNN. These results are similar to those obtained by a feedforward network that was trained with 100 000 sample points [10]. The considerably more efficient training of MBRNN is due to its fixed structure based on the linearized model of the plant.

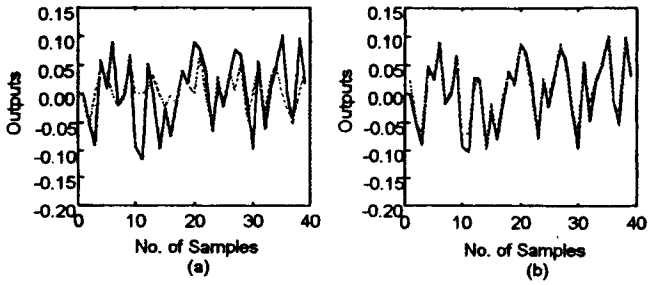

Fig. 5. Outputs of the plant and MBRNN (a) before training and (b) after training by EKF, for Example 1.

Example 2:

$$\begin{Bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{Bmatrix} = \begin{bmatrix} -0.3(x_1(k)+\sin^2(x_1(k))) & 0 & 0 \\ 2.9x_1(k) & -0.62x_2(k) & -2.3x_3(k) \\ 0 & 2.3x_2(k) & 0 \end{bmatrix} + \begin{Bmatrix} u \\ 0 \\ 0 \end{Bmatrix} \tag{35}$$

$$y(k) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -3 & 1 \end{bmatrix} \begin{Bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{Bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{36}$$

The above model represents a single-input multiple-output system, with nonlinear state equations. The linearized state-space model of this plant about the operating point ($x_1 = 0, x_2 = 0, x_3 = 0$) was obtained as

$$\begin{Bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{Bmatrix} = \begin{bmatrix} -0.3 & 0 & 0 \\ 2.9 & -0.62 & -2.3 \\ 0 & 2.3 & 0 \end{bmatrix} \begin{Bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{Bmatrix} + \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} u \tag{37}$$

$$y(k) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -3 & 1 \end{bmatrix} \begin{Bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{Bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{38}$$

The MBRNN was structured according to the above state-space model and then trained by dynamic BP without a regularization parameter. The training session consisted of 1500 sample points generated with random excitation inputs with values between -6.0 and 6.0. One epoch of training was used. The trained MBRNN was then tested by data generated with random inputs from a different seed than that which was used during testing. The values of $\hat{y}_1$ and y_1 for the test session before and after training are shown in Fig. 6. The results indicate a potential pitfall associated with dynamic BP as applied to MBRNN. The reason for the poor results in this application is entrapment of the system in a limit cycle at $x_1^* = 0.475$ $x_2^* = -0.003$ $x_3^* = -0.0751$,[1] which is caused by the shape of the activation function shown for one of the nodes in Fig. 7. Similar results to those in Fig. 6(b) can be obtained by introducing small excitation values for u to $\Phi_{\text{net}}(\mathbf{x}^*, u)$.

[1]These points which are the equilibrium points of $\mathbf{x}^* = \Phi_{\text{net}}(\mathbf{x}^* 0)$ represent the coordinates of a stable limit cycle. The stability of this limit cycle is verified by the eigenvalues of the Jacobian being inside the unit cycle.

The MBRNN was next trained by EKF using the same training data. The values of $\hat{y}_1$ and y_1 from this MBRNN are shown in Fig. 8, indicating a much better representation by MBRNN.

Example 3:

$$\begin{bmatrix} (m_1+m_2)a_1^2+m_2a_2^2+2m_2a_1a_2\cos\theta_2 & m_2a_2^2+m_2a_1a_2\cos\theta_2 \\ m_2a_2^2+m_2a_1a_2\cos\theta_2 & m_2a_2^2 \end{bmatrix} \times \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} + \begin{bmatrix} -m_2a_1a_2(2\dot{\theta}_1\dot{\theta}_2+\dot{\theta}_2^2)\sin\theta_2 \\ m_2a_1a_2\dot{\theta}_1^2\sin\theta_2 \end{bmatrix} + \begin{bmatrix} (m_1+m_2)ga_1\cos\theta_1+m_2ga_2\cos(\theta_1+\theta_2) \\ m_2ga_2\cos(\theta_1+\theta_2) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}. \tag{39}$$

The above model represents the dynamics of a two-link plannar elbow robot arm [24], where θ_1 and θ_2 represent the angular positions of the two arms, m_1 and m_2 denote their masses, and τ_1 and τ_2 represent the torques exerted on the arms. If we define the position vector $\mathbf{q}$ as $\mathbf{q} = [\theta_1 \ \theta_2]^T$ and the generalized force vector τ as $\tau = [\tau_1 \ \tau_2]^T$, then the above state-space equation can be written as

$$M(\mathbf{q})\ddot{\mathbf{q}} + V(\mathbf{q}, \dot{\mathbf{q}}) + G(\mathbf{q}) = \tau \tag{40}$$

where $M(\mathbf{q})$ denotes the inertia matrix, $V(\mathbf{q}, \dot{\mathbf{q}})$ represents the Coriolis/centripetal vector, and $G(\mathbf{q})$ denotes the gravity vector.

The linear state-space form of the above model about $\mathbf{q} = [0 \ 0]^T$, $\dot{\mathbf{q}} = [0 \ 0]^T$ was obtained as

$$\begin{Bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{Bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 0.01 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.01 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{Bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{Bmatrix} + \begin{bmatrix} 0.0001 & -0.0002 \\ -0.0002 & 0.0004 \\ 0.02 & -0.04 \\ -0.04 & 0.085 \end{bmatrix} \begin{Bmatrix} \tau_1 \\ \tau_2 \end{Bmatrix}$$

$$\begin{Bmatrix} y_1(k) \\ y_2(k) \\ y_3(k) \\ y_4(k) \end{Bmatrix} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{Bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{Bmatrix}$$

where $\mathbf{x}^T = [\mathbf{q} \ \dot{\mathbf{q}}]$. The MBRNN was structured according to the above state-space model and then trained by EKF. The training session consisted of 5500 sample points generated by random excitation inputs with values between -1.0 through 1.0 and -0.5 through 0.5 for two inputs, respectively. One epoch of training was used. The trained MBRNN was then tested by data generated with random inputs from a different seed than that which was used during testing. The values of $\hat{y}$ and y before and after training are shown in Figs. 9 and 10, respectively. The results indicate that the outputs of the trained MBRNN are quite close to the plant outputs, again despite the relatively short training session used by MBRNN.

V. Conclusion

A MBRNN is introduced which can be formulated according to the analytical knowledge of the plant. This network is defined to initially emulate a linearized state-space model of the plant. It can then be trained to accommodate the nonlinearities of the plant by modifying its activation functions, which are defined as contours of RBF's comprising each node. As such, the MBRNN has the structure of a modular network, where each module represents a node. Both

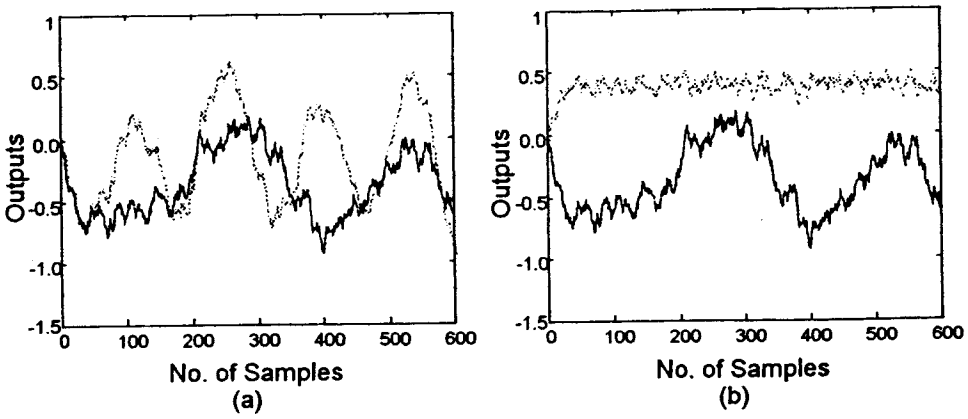

Fig. 6. Outputs of the plant and MBRNN (a) before training and (b) after training by dynamic BP without a regularization parameter, for Example 2.

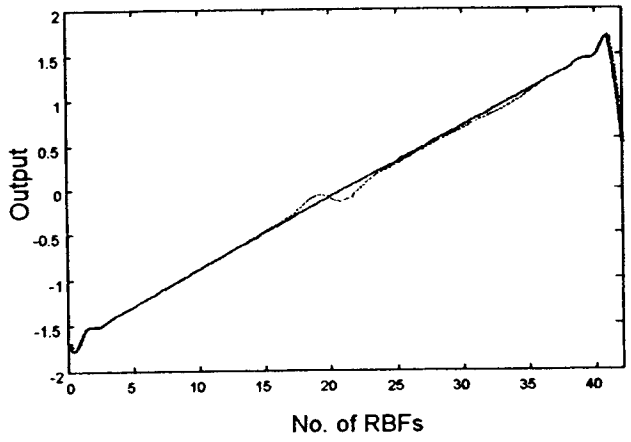

Fig. 7. Modified activation function of the a_{22} component causing the limit cycle problem.

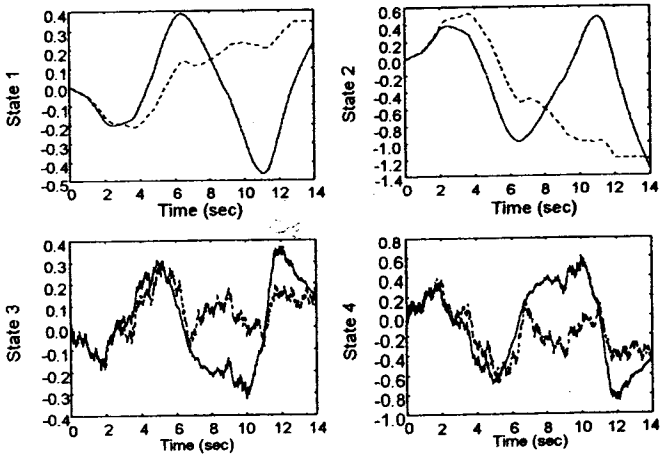

Fig. 9. Four states of the plant and MBRNN before training for Example 3.

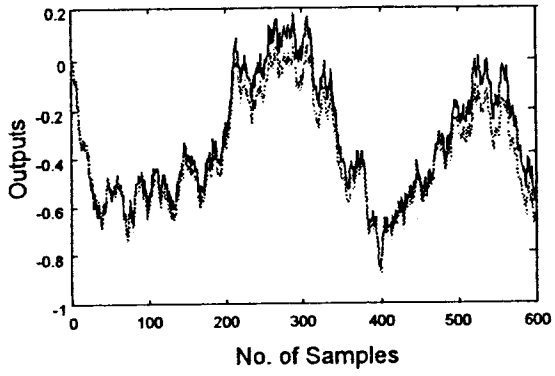

Fig. 8. Output 1 of the plant and MBRNN after training by the EKF for Example 2.

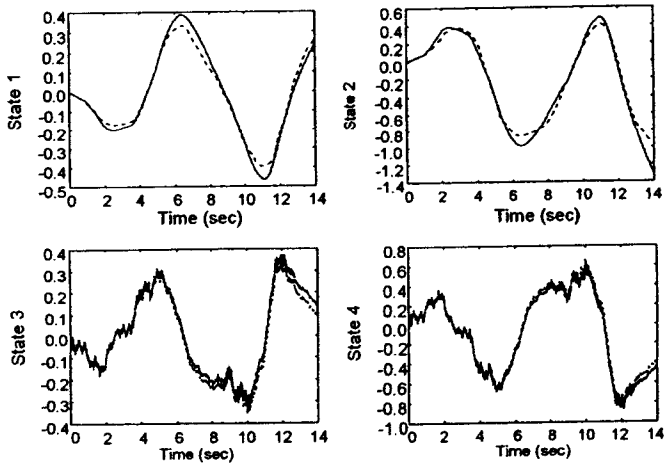

Fig. 10. Four states of the plant and MBRNN after training for Example 3.

dynamic backpropagation and the EKF can be used for training. Simulation results indicate that the EKF provides more satisfactory performance. The results also indicate that the MBRNN requires significantly less training than ordinary recurrent networks.

The MBRNN will have potential utility in control applications. A common drawback of most neuro-control schemes is the extensive training required to model the plant by the neural network. In these cases, training is needed for both establishing the topology of the network, as well as adjusting its connection weights. MBRNN not only provides a good initial structure for the plant model, but defines the connection weights of this neural network model according to the linearized model of the plant. As such, MBRNN provides a functional initial estimate for the plant model that can be used for control purposes from the beginning of operation and that can be subsequently improved over time with training.

REFERENCES

[1] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus, "Training recurrent neural networks: Why and how? An illustration in dynamical process modeling," *IEEE Trans. Neural Networks*, vol. 5, pp. 178–184, Feb. 1994.

[2] J. Albus, "A new approach to manipulaor control: The cerebellar model articulation controller," *ASME J. Dyn. Syst., Meas., and Control*, vol. 97, pp. 220–227, 1975.

[3] W. T. Miller, "Real time application of neural networks for sensor-based control of robots with vision," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 825–831, 1989.

[4] A. G. parlos, K. T. Chong, and A. F. Atiya, "Application of the recurrent multilayer perceptron in modeling complex process dynamics," *IEEE Trans. Neural Networks*, vol. 5, pp. 255–266, Feb. 1994.

[5] A. Srinivasan and C. Batur, "Back propagation through adjoints for the identification of nonlinear dynamic systems using recurrent neural models," *IEEE Trans. Neural Networks*, vol. 5, pp. 213–227, Feb. 1994.

[6] B. Srinivasan, U. R. Prasad, and N. J. Rao, "Back propagation through adjoints for the identification of nonlinear dynamic systems using recurrent neural models," *IEEE Trans. Neural Networks*, vol. 5, pp. 213–227, 1994.

[7] D. Obradovic, "On-line training of recurrent neural networks with continuous topology adaptation," *IEEE Trans. Neural Networks*, vol. 7, pp. 222–228, 1996.

[8] C. C. Ku and K. Y. Lee, "Diagonal recurrent neural networks for dynamic systems control," *IEEE Trans. Neural Networks*, vol. 6, pp. 144–156, Jan. 1995.

[9] T. Denoeux and R. Lengelle, "Initializing back propagation networks with prototypes," *IEEE Trans. Neural Networks*, vol. 6, pp. 351–363, 1993.

[10] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, Jan. 1990.

[11] J. S. Jang and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 4, pp. 156–159, 1993.

[12] K. J. Hunt, R. Haas, and R. Murray-Smith, "Extending the functional equivalence of radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 7, pp. 776–781, 1996.

[13] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artif. Intell.*, vol. 70, pp. 110–165, 1994.

[14] C. W. Omlin and C. L. Giles, "Rule revision with recurrent neural networks," *IEEE Trans. Knowl. Data Eng.*, vol. 8, pp. 183–197, Jan. 1996.

[15] M. M. Livstone, J. A. Farrell, and W. L. Baker, A computationally efficient algorithm for training recurrent connectionist networks, in ACC/WM2, pp. 555–561, 1992.

[16] R. R. Hocking, "The analysis and selection of variables in linear regression," *Biometrics*, vol. 32, pp. 1–49, 1976.

[17] R. R. Hocking, "Developments in linear regression methodology," *Technometrics*, vol. 25, pp. 219–249, 1983.

[18] M. J. L. Orr, *Introduction to Radial Basis Function Networks*. London, U.K.: Centre for Cognitive Science, Univ. Edinburgh, 1996.

[19] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560, Oct. 1990.

[20] K. S. Narendra and K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks," *IEEE Trans. Neural Networks*, vol. 2, pp. 252–262, Feb. 1991.

[21] A. Gelb, Ed., *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.

[22] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing: Explorations in Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986.

[23] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Boston, MA: PWS, 1996.

[24] F. L. Lewis, C. T. Abdallah, and D. M. Dawson, *Control of Robot Manipulators*. New York: Macmillan, 1993.

A Two-Level Hybrid Evolutionary Algorithm For Modeling One-Dimensional Dynamic Systems by Higher-Order ODE Models

Hong-Qing Cao, Li-Shan Kang, Tao Guo, Yu-Ping Chen, and Hugo de Garis

***Abstract*—This paper presents a new algorithm for modeling one-dimensional (1-D) dynamic systems by higher-order ordinary differential equation (HODE) models instead of the ARMA models as used in traditional time series analysis. A two-level hybrid evolutionary modeling algorithm (THEMA) is used to approach the modeling problem of HODE's for dynamic systems. The main idea of this modeling algorithm is to embed a genetic algorithm (GA) into genetic programming (GP), where GP is employed to optimize the structure of a model (the upper level), while a GA is employed to optimize the parameters of the model (the lower level). In the GA, we use a novel crossover operator based on a nonconvex linear combination of multiple parents which works efficiently and quickly in parameter optimization tasks. Two practical examples of time series are used to demonstrate the THEMA's effectiveness and advantages.**

***Index Terms*—Evolutionary modeling, genetic alogorithm, genetic programing, higher order ordinary differential equation.**

I. INTRODUCTION

Many complex systems (e.g., dynamic systems) and nonlinear phenomena which change with time exist in engineering, economic management, natural sciences, social sciences, and many other fields. Examples include price fluctuations, the change of temperature during a chemical process, weather changes, population growth, and so on. It is important to build dynamic mathematical models for these systems based upon the observed data (i.e., time series) so as to provide a basis for system analysis, design, and prediction. In traditional time series analysis, three kinds of models have been used in most cases. These are autoregressive (AR) models, moving average (MA) models, and ARMA models [1], [2]. Variations based on these three models have also been studied, such as autoregressive integrated moving average (ARIMA) models [3], [4], Bilinear models [5], [6], threshold autoregressive (TAR) models [7], and TARMA models [8]. The traditional method for solving the modeling problem of dynamic systems is to choose a model structure for the system initially, including the type and the order of the model, to determine the parameters contained in the model, and finally, to test the validity of the model [1], [2], [9], [10]. However, it is often difficult for people to choose a suitable model structure, without having sufficient domain knowledge and expertise. The determination of the parameters usually requires the modeler to have a rich mathematical knowledge and professional skills. Additionally, the traditional ARMA models are linear, which restricts the range of analysis of dynamic processes. In contrast to ARMA models, which use linear difference equations, we present a new idea for modeling one-dimensional (1-D) dynamic systems, which uses higher-order ordinary differential equation (HODE) models. To overcome the difficulties and drawbacks existing in traditional modeling methods, we pro-

Manuscript received September 18, 1998; revised January 5, 2000. This work was supported by State Key Laboratory of Parallel and Distributed Processing, National Natural Science Foundation of China (Grant 69635030), and National 863 High Technology Project of China. This paper was recommended by Associate Editor T. Sudkamp.

H.-Q. Cao, L.-S. Kang, T. Guo, and Y.-P. Chen are with State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China (e-mail: jxyu@whu.edu.cn).

H. De Garis is with Starlab, Brussels B-1040, Belgium.

Publisher Item Identifier S 1083-4419(00)02973-3.