4-24-2012

The recursive neural network

Chaouki T. Abdallah

Don Hush

Bill Horne

The Recursive Neural Network

Don Hush, Chaouki Abdallah, and Bill Horne
Department of Electrical Engineering and Computer Engineering
University of New Mexico
Albuquerque, NM, 87131 USA.

ABSTRACT

This paper describes a special type of dynamic neural network called the Recursive Neural Network (RNN). The RNN is a single-input single-output nonlinear dynamical system with three subnets, a *nonrecursive* subnet and two *recursive* subnets. The nonrecursive subnet feeds current and previous input samples through a multi-layer perceptron with second order input units (SOMLP) [9]. In a similar fashion the two recursive subnets feed back previous output signals through SOMLPs. The outputs of the three subnets are summed to form the overall network output. The purpose of this paper is to describe the architecture of the RNN, to derive a learning algorithm for the network based on a gradient search, and to provide some examples of its use.

The work in this paper is an extension of previous work on the RNN [10]. In previous work the RNN contained only two subnets, a nonrecursive subnet and a recursive subnet. Here we have added a second recursive subnet. In addition, both of the subnets in the previous RNN had linear input units. Here all three of the subnets have second order input units. In many cases this allows the RNN to solve problems more efficiently, that is with a smaller overall network. In addition, the use of the RNN for inverse modeling and control was never fully developed in the past. Here, for the first time, we derive the complete learning algorithm for the case where the RNN is used in the general *model following* configuration. This configuration includes the following as special cases: system modeling, nonlinear filtering, inverse modeling, nonlinear prediction and control.

KEYWORDS

Recursive neural network, dynamic neural network, recurrent neural network, dynamic backpropagation, multi-layer perceptron, nonlinear control, nonlinear filter.

List of Acronyms, Symbols, and Notation

RNN	the recursive neural network
MLP	multi-layer perceptron
SOMLP	MLP with second order input units
$x(k)$	input to the RNN
$u(k)$	output of the RNN and input to the plant
$y(k)$	output of the plant
$d(k)$	desired output of the plant
$e(k)$	error output (difference between the desired and actual output of the plant)
a,b,c	superscripts and subscripts used to indicate subnets a, b, and c respectively
M_s	number of layers in subnet s ($s \in \{a,b,c\}$)
$\mathbf{v}_l^s(k)$	output vector for the l^{th} layer in the s^{th} subnet
$v_{l,j}^s(k)$	output of the j^{th} node in the l^{th} layer of subnet s. Note, because there is only one node in the output layer of each subnet the second subscript is suppressed, e.g. $v_{M_s}^s(k) = v_{M_s,1}^s(k)$ is the output of the M_s layer (output layer) in subnet s.
$\mathbf{W}_l^s$	weight matrix connecting outputs of layer $l-1$ to nodes in layer l in subnet s.
$\mathbf{w}_{l,0}^s$	bias weights for nodes in layer l of subnet s
$f(\cdot)$	nonlinear activation function
N_l	number of nodes in layer l
$w_{l,i,j}^s$	weight connecting node i in layer $l-1$ to node j in layer l in subnet s
$\alpha_{l,i,j}^s$	partial derivative of RNN output, $u(k)$, with respect to $w_{l,i,j}^s$
$\beta_{l,i,j}^s$	partial derivative of plant output, $y(k)$, with respect to $w_{l,i,j}^s$

1. Introduction

This paper introduces a dynamical neural network structure and derives a learning algorithm based on a gradient search. The network structure is shown in Figure 1. It consists of three subnets, a nonrecursive (feedforward) subnet and two recursive (feedback) subnets. All of the subnets are multi-layer perceptrons with second order connections at the input layer (SOMLPs). The nonrecursive subnet (subnet a) processes current and delayed values of the input signal $x(k)$. The recursive subnets, subnets b and c, feed back delayed values from the net output $u(k)$ and the plant output $y(k)$ respectively. This structure is basically an extension of the Recursive Neural Network discussed in [10]. The output of the RNN is the sum of the outputs of the three subnets. The weights of the RNN are adjusted using a gradient search technique. The model M and the plant P in Figure 1 are nonlinear dynamical systems which are to be modeled or controlled by the RNN.

The remainder of this paper is organized as follows. Section 2 describes the operation of the RNN and introduces the notation that will be used. Section 3 derives a learning algorithm for the RNN based on a gradient search. Section 4 presents several examples. These examples illustrate the use of the RNN in a variety of scenarios including system identification, nonlinear digital filtering, inverse modeling, control, and nonlinear prediction. Section 5 presents a brief analysis of the RNN, and section 6 contains a summary.

2. The Recursive Neural Network Structure

An integral part of the recursive neural network structure shown in Figure 1 is the plant, P. In general we allow P to be a nonlinear dynamical system whose input/output behavior is described by

$$y(k) = g\left[u(k), u(k-1), \cdots, u(k-o_n), y(k-1), \cdots, y(k-o_r)\right] \tag{1}$$

where o_n and o_r are the *nonrecursive* and *recursive* orders of the plant respectively. The plant may be an unknown system. That is we may not know the exact functional form of $g(\cdot)$. Depending on how the RNN is used the learning algorithms in this paper may require knowledge of the Jacobian of the plant. In those scenarios where this is true we assume that it is available or can be estimated. Methods for estimating the Jacobian are not developed in this paper, although a simple method will be mentioned as part of the discussion in section 3. The examples in this paper use the exact form of the Jacobian obtained from the difference equation of the plant.

The output of the recursive neural network, RNN, at time k is the sum of the outputs of three subnets, a nonrecursive subnet a and two recursive subnets b and c.

$$u(k) = v^a_{M_a}(k) + v^b_{M_b}(k) + v^c_{M_c}(k) \tag{2}$$

Each of these networks is a multi-layer perceptron with second-order terms at the input layer (SOMLP) [9]. The superscripts a, b, and c are used to distinguish between parameters in the nonrecursive and the two recursive subnets respectively. M_a, M_b, and M_c represent the number of layers in subnets a, b, and c respectively.

If we let $\mathbf{v}_l^s(k)$ represent the output vector of the l^{th} layer for the s^{th} subnet ($s \in \{a,b,c\}$) then signals are propagated through the networks according to

$$\mathbf{v}_l^s(k) = \mathbf{f}\,(\,\mathbf{W}_l^s \mathbf{v}_{l-1}^s(k) + \mathbf{w}_{l,\,0}^s) \tag{3}$$

$\mathbf{W}_l^s$ represents the weight matrix of subnet s that connects the outputs of layer $l-1$ to the nodes in layer l,

$$\mathbf{W}_l = \begin{bmatrix} w_{l,\,1,1}^s & w_{l,\,1,2}^s & \cdot & w_{l,\,1,N_{l-1}}^s \\ w_{l,\,2,1}^s & w_{l,\,2,2}^s & \cdot & w_{l,\,2,N_{l-1}}^s \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ w_{l,N_l,1}^s & w_{l,N_l,2}^s & \cdot & w_{l,N_l,N_{l-1}}^s \end{bmatrix} \tag{4}$$

and $\mathbf{w}_{l,\,0}^s$ are the bias weights for layer l.

$$\mathbf{w}_{l,\,0}^s = \left[w_{l,\,0,1}^s,\, w_{l,\,0,2}^s,\, \cdots,\, w_{l,\,0,N_l}^s \right]^T \tag{5}$$

$\mathbf{f}(\cdot)$ is a function which produces a vector result from a vector argument by applying a nonlinear function, $f\,(\cdot)$ to each component, that is

$$\mathbf{f}(\mathbf{z}) = \begin{bmatrix} f\,(z_1) \\ f\,(z_2) \\ \cdot \\ \cdot \\ f\,(z_m) \end{bmatrix} \tag{6}$$

One of the most common choices for for the nonlinear function is $f\,(z_i) = (1+e^{-z_i})^{-1}$.

The input vector to subnet a, the nonrecursive subnet, consists of current and delayed values of the input signal, and all second order combinations of these values. Thus, $\mathbf{v}_0^a(k)$ takes on the form

$$\begin{aligned} \mathbf{v}_0^a(k) = [&x\,(k),\, x\,(k-1),\, \cdots,\, x\,(k-o_a), \\ &x^2(k),\, x\,(k)x\,(k-1),\, \cdots,\, x\,(k)x\,(k-o_a), \end{aligned}$$

$$
\begin{aligned}
& x^2(k-1),\ x(k-1)x(k-2),\ \cdots,\ x(k-1)x(k-o_a), \\
& \cdots, \\
& x^2(k-o_a)] \qquad (7)
\end{aligned}
$$

The input to subnet b consists of previous RNN outputs, and all second order combinations of these.

$$
\begin{aligned}
\mathbf{v}_0^b(k) \ = \ & [u(k-1),\ u(k-2),\ \cdots,\ u(k-o_b), \\
& u^2(k-1),\ u(k-1)u(k-2),\ \cdots,\ u(k-1)u(k-o_b), \\
& u^2(k-2),\ u(k-2)u(k-3),\ \cdots,\ u(k-2)u(k-o_b), \\
& \cdots, \\
& u^2(k-o_b)] \qquad (8)
\end{aligned}
$$

Similarly the input to subnet c consists of previous plant outputs, and all second order combinations of these.

$$
\begin{aligned}
\mathbf{v}_0^c(k) \ = \ & [y(k-1),\ y(k-2),\ \cdots,\ y(k-o_c), \\
& y^2(k-1),\ y(k-1)y(k-2),\ \cdots,\ y(k-1)y(k-o_c), \\
& y^2(k-2),\ y(k-2)y(k-3),\ \cdots,\ y(k-2)y(k-o_c), \\
& \cdots, \\
& y^2(k-o_c)] \qquad (9)
\end{aligned}
$$

In (7)-(9) o_a, o_b, and o_c are the orders of subnets a, b, and c respectively. Note, all subnets have only one output node so that $\mathbf{W}_{M_a}^a$, $\mathbf{W}_{M_b}^b$, and $\mathbf{W}_{M_c}^c$ are row vectors and $\mathbf{w}_{M_a,0}^a$, $\mathbf{w}_{M_b,0}^b$, and $\mathbf{w}_{M_c,0}^c$ are scalars.

3. Weight Update Equations for the Recurrent Neural Network

Our goal here is to derive a learning algorithm that can be used to find the set of network weights that minimize the total sum-of-squared error over the duration of the training signal,

$$
E \ = \ \frac{1}{2}\sum_{k=1}^{N} e^2(k) \ = \ \frac{1}{2}\sum_{k=1}^{N} (d(k)-y(k))^2 \qquad (10)
$$

where $d(k)$ is the output of the model M in Figure 1. The weights are sought using a gradient search of the form

$$w_{l,i,j}^{s}(m+1) = w_{l,i,j}^{s}(m) - \mu \frac{\partial E}{\partial w_{l,i,j}^{s}} \tag{11}$$

where m is the iteration index, and $w_{l,i,j}^{s}$ is the element of $\mathbf{W}_l^s$ that corresponds to the weight connecting node i in layer $l-1$ to node j in layer l. Applying the partial to E yields

$$w_{l,i,j}^{s}(m+1) = w_{l,i,j}^{s}(m) + \mu \sum_{k=1}^{N} e(k)\, \beta_{l,i,j}^{s}(k) \tag{12}$$

where

$$\beta_{l,i,j}^{s}(k) = \frac{\partial y(k)}{\partial w_{l,i,j}^{s}} \tag{13}$$

From (1) this term evaluates to

$$\beta_{l,i,j}^{s}(k) = \frac{\partial y(k)}{\partial w_{l,i,j}^{s}} = \mathbf{q}_1^T(k)\mathbf{a}_{l,i,j}^{s}(k) + \mathbf{q}_2^T(k-1)\mathbf{b}_{l,i,j}^{s}(k-1) \tag{14}$$

where

$$\mathbf{q}_1^T(k) = \left[\frac{\partial g(\cdot)}{\partial u(k)}, \frac{\partial g(\cdot)}{\partial u(k-1)}, \cdots, \frac{\partial g(\cdot)}{\partial u(k-o_n)}\right] \tag{15}$$

$$\mathbf{q}_2^T(k-1) = \left[\frac{\partial g(\cdot)}{\partial y(k-1)}, \frac{\partial g(\cdot)}{\partial y(k-2)}, \cdots, \frac{\partial g(\cdot)}{\partial y(k-o_r)}\right] \tag{16}$$

and

$$\begin{aligned}\mathbf{a}_{l,i,j}^{s}{}^{T}(k) &= \left[\frac{\partial u(k)}{\partial w_{l,i,j}^{s}}, \frac{\partial u(k-1)}{\partial w_{l,i,j}^{s}}, \cdots, \frac{\partial u(k-o_n)}{\partial w_{l,i,j}^{s}}\right] \\ &= \left[\alpha_{l,i,j}^{s}(k), \alpha_{l,i,j}^{s}(k-1), \cdots, \alpha_{l,i,j}^{s}(k-o_n)\right]\end{aligned} \tag{17}$$

$$\mathbf{b}_{l,i,j}^{s}{}^{T}(k-1) = \left[\frac{\partial y(k-1)}{\partial w_{l,i,j}^{s}}, \frac{\partial y(k-2)}{\partial w_{l,i,j}^{s}}, \cdots, \frac{\partial y(k-o_r)}{\partial w_{l,i,j}^{s}}\right]$$

$$= \left[\beta^s_{l,i,j}(k-1), \beta^s_{l,i,j}(k-2), \cdots, \beta^s_{l,i,j}(k-o_r)\right] \tag{18}$$

If $g(\cdot)$ is known then $\mathbf{q}_1$ and $\mathbf{q}_2$ can be evaluated directly. If $g(\cdot)$ is unknown then $\mathbf{q}_1$ and $\mathbf{q}_2$ must be estimated. One method for estimating these values is to form a model of the plant which has a known analytical form and then use the Jacobian of the model to determine $\mathbf{q}_1$ and $\mathbf{q}_2$. In fact, the RNN itself can be used to form the model since $\mathbf{q}_1$ and $\mathbf{q}_2$ are not needed when the RNN is used for system identification. An example of how the RNN can be used for system identification is shown in section 4.1. Other methods for estimating $\mathbf{q}_1$ and $\mathbf{q}_2$ are beyond the scope of this paper. For now we proceed under the assumption that a mechanism exists for computing $\mathbf{q}_1$ and $\mathbf{q}_2$ at each time k.

The other vectors in (14) are determined as follows. First we note that the current value of $\beta^s_{l,i,j}(k)$ is a recursive function of previous values through the components of the $\mathbf{b}^s_{l,i,j}$ vector. Thus, only the components of the $\mathbf{a}^s_{l,i,j}$ vector are yet to be determined. From (17) and the expression for $u(k)$ in (2) we have

$$\alpha^s_{l,i,j}(k) = \frac{\partial u(k)}{\partial w^s_{l,i,j}} = \frac{\partial v^a_{M_a}(k)}{\partial w^s_{l,i,j}} + \frac{\partial v^b_{M_b}(k)}{\partial w^s_{l,i,j}} + \frac{\partial v^c_{M_c}(k)}{\partial w^s_{l,i,j}} \tag{19}$$

Equations (12)-(19) are valid for weights in all three subnets. What remains is to derive an expression for $\alpha^s_{l,i,j}(k)$ for each subnet. This must be done differently for weights in the three subnets. We begin with subnet a.

3.1. Weight Updates for Subnet *a*

For weights in the nonrecursive subnet (19) is given by

$$\alpha^a_{l,i,j}(k) = \frac{\partial u(k)}{\partial w^a_{l,i,j}} = \frac{\partial v^a_{M_a}(k)}{\partial w^a_{l,i,j}} + \frac{\partial v^b_{M_b}(k)}{\partial w^a_{l,i,j}} + \frac{\partial v^c_{M_c}(k)}{\partial w^a_{l,i,j}} \tag{20}$$

It is easy to show that the first term in (20) gives rise to the standard backpropagation equations [2], that is

$$\frac{\partial v^a_{M_a}(k)}{\partial w^a_{l,i,j}} = BP^a_{l,i,j}(k) \tag{21}$$

where

$$BP^a_{l,i,j}(k) = \varepsilon^a_{l,j}(k)\, v^a_{l-1,i}(k) \tag{22}$$

and $\varepsilon^a_{l,j}(k)$ represents a signal which is backpropagated according to

$$\varepsilon^a_{l,j}(k) \;=\; h^a_{l,j}(k) \sum_{i=1}^{N_{l+1}} \varepsilon^a_{l+1,i}(k)\, w^a_{l+1,j,i} \tag{23}$$

where

$$h^a_{l,j}(k) \;=\; \frac{\partial f(z)}{\partial z}\Big|_{z^a_{l,j}(k)} \tag{24}$$

Note that for convenience we have defined $v_{l,\,0}$, the input to the bias weights, to be 1. Also, N_l represents the number of nodes in layer l. At the output layer (where we have only one node) $\varepsilon^a_{M_a,1}(k)$ is given by

$$\varepsilon^a_{M_a,1}(k) \;=\; h^a_{M_a,1}(k) \tag{25}$$

Note, $\varepsilon^a_{l,j}(k)$ differs from the "backpropagated error" signal defined in the standard back-propagation algorithm only in that it does not include the output error. In this paper the output error is incorporated separately in (12).

The second term in (20) represents the influence of weights in the nonrecursive sub-net on the output of subnet b. This influence is exhibited through previous net outputs $u(k-1), u(k-2)$, etc., which serve as the input to subnet b. From (23) we have

$$\frac{\partial v^b_{M_b}(k)}{\partial w^a_{l,i,j}} \;=\; \frac{\partial}{\partial w^a_{l,i,j}} \left[f(\,\mathbf{W}^b_{M_b} \mathbf{v}^b_{M_b-1}(k) + \mathbf{w}^b_{M_b,0}) \right] \tag{26}$$

Applying the partial derivative to the output layer yields

$$\begin{aligned} \frac{\partial v^b_{M_b}(k)}{\partial w^a_{l,i,j}} &= h^b_{M_b}(k) \frac{\partial}{\partial w^a_{l,i,j}} \left[\mathbf{W}^b_{M_b} \mathbf{v}^b_{M_b-1}(k) + \mathbf{w}^b_{M_b 0} \right] \\ &= h^b_{M_b}(k)\, \mathbf{W}^b_{M_b} \frac{\partial \mathbf{v}^b_{M_b-1}(k)}{\partial w^a_{l,i,j}} \end{aligned} \tag{27}$$

Repeating this operation for all layers of subnet b yields

$$\frac{\partial v^b_{M_b}(k)}{\partial w^a_{l,i,j}} \;=\; h^b_{M_b}(k)\, \mathbf{W}^b_{M_b}\; \mathbf{H}^b_{M_b-1}(k)\, \mathbf{W}^b_{M_b-1} \;\cdots\; \mathbf{H}^b_1(k)\, \mathbf{W}^b_1 \frac{\partial \mathbf{v}^b_0(k)}{\partial w^a_{l,i,j}} \tag{28}$$

where $\mathbf{H}_l^b(k)$ is a diagonal matrix of the form

$$\mathbf{H}_l^b(k) \; = \; diag\left[h_{l,n}^b(k)\right] \qquad n=1,2,\cdots,N_l \tag{29}$$

The last term in (28) represents the partial derivative of the input to subnet b with respect to $w_{l,i,j}^a$. Recall, this input contains all first and second order combinations of previous RNN outputs, so that

$$\begin{aligned}\frac{\partial \mathbf{v}_0^b(k)}{\partial w_{l,i,j}^a} \; = \; \frac{\partial}{\partial w_{l,i,j}^a}[&u(k-1),\, u(k-2),\, \cdots,\, u(k-o_b),\\ &u^2(k-1),\, u(k-1)u(k-2),\, \cdots,\, u(k-1)u(k-o_b),\\ &u^2(k-2),\, u(k-2)u(k-3),\, \cdots,\, u(k-2)u(k-o_b),\\ &\cdots,\, u^2(k-o_b)]^T\end{aligned} \tag{30}$$

which is given by

$$\frac{\partial \mathbf{v}_0^b(k)}{\partial w_{l,i,j}^a} \; = \; \mathbf{U}(k-1)\; \mathbf{a}_{l,i,j}^a(k-1) \tag{31}$$

where

$$\mathbf{U}(k-1) \; = \; \begin{bmatrix} \mathbf{I} \\ \mathbf{U}_1(k-1) \\ \mathbf{U}_2(k-1) \\ \cdot \\ \cdot \\ \mathbf{U}_{o_b}(k-1) \end{bmatrix} \tag{32}$$

and

$$\mathbf{U}_1(k-1) \; = \; \begin{bmatrix} 2u(k-1) & 0 & 0 & \cdot\;\cdot & 0 \\ u(k-2) & u(k-1) & 0 & \cdot\;\cdot & 0 \\ u(k-3) & 0 & u(k-1) & \cdot\;\cdot & 0 \\ \cdot & \cdot & \cdot & \cdot\;\cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot\;\cdot & \cdot \\ u(k-o_b) & 0 & 0 & \cdot\;\cdot & u(k-1) \end{bmatrix}_{o_b \; x \; o_b}$$

$$\mathbf{U}_2(k-1) = \begin{bmatrix} 0 & 2u(k-2) & 0 & \cdot\ \cdot & 0 \\ 0 & u(k-3) & u(k-2) & \cdot\ \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot\ \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot\ \cdot & \cdot \\ 0 & u(k-o_b) & 0 & \cdot\ \cdot & u(k-2) \end{bmatrix}_{(o_b-1)\ x\ o_b}$$

$$\vdots$$

$$\mathbf{U}_{o_b}(k-1) = \begin{bmatrix} 0 & 0 & 0 & \cdot\ \cdot & 2u(k-o_b) \end{bmatrix}_{1\ x\ o_b}$$

We note that here $\mathbf{a}^a_{l,i,j}$ is a vector with the same components as in (17), but with dimension o_b rather than o_n,

$$\mathbf{a}^a_{l,i,j}(k-1) = \left[\alpha^a_{l,i,j}(k-1), \alpha^a_{l,i,j}(k-2), \cdots, \alpha^a_{l,i,j}(k-o_b)\right]^T \tag{33}$$

With this (28) becomes

$$\frac{\partial v^b_{M_b}(k)}{\partial w^a_{l,i,j}} = FF^b(\ \mathbf{a}^a_{l,i,j}(k-1)\ ) \tag{34}$$

where $FF^b(\cdot)$ represents the operation

$$FF^b(\ \mathbf{a}) = h^b_{M_b}(k)\ \mathbf{W}^b_{M_b}\ \mathbf{H}^b_{M_b-1}(k)\ \mathbf{W}^b_{M_b-1}\ \cdots\ \mathbf{H}^b_1(k)\ \mathbf{W}^b_1\ \mathbf{U}(k-1)\ \mathbf{a} \tag{35}$$

Note, this equation depicts a feed-forward type of operation through subnet b.

The third term in (20) is evaluated in much the same way as the second. By definition

$$\frac{\partial v^c_{M_c}(k)}{\partial w^a_{l,i,j}} = \frac{\partial}{\partial w^a_{l,i,j}}\left[f(\ \mathbf{W}^c_{M_c}\mathbf{v}^c_{M_c-1}(k) + \mathbf{w}^c_{M_c0})\ \right] \tag{36}$$

Applying the partial derivative through all layers of subnet c (just as we did in (27)-(28) for subnet b) yields

$$\frac{\partial v^c_{M_c}(k)}{\partial w^a_{l,i,j}} = h^c_{M_c}(k)\ \mathbf{W}^c_{M_c}\ \mathbf{H}^c_{M_c-1}(k)\ \mathbf{W}^c_{M_c-1}\ \cdots\ \mathbf{H}^c_1(k)\ \mathbf{W}^c_1\ \frac{\partial \mathbf{v}^c_0(k)}{\partial w^a_{l,i,j}} \tag{37}$$

Using (9) the last term in (37) is given by

$$\begin{aligned}\frac{\partial \mathbf{v}_0^c(k)}{\partial w_{l,i,j}^a} = \frac{\partial}{\partial w_{l,i,j}^a}[&y(k-1),\, y(k-2),\, \cdots,\, y(k-o_b),\\ &y^2(k-1),\, y(k-1)y(k-2),\, \cdots,\, y(k-1)y(k-o_c),\\ &y^2(k-2),\, y(k-2)y(k-3),\, \cdots,\, y(k-2)y(k-o_c),\\ &\cdots, y^2(k-o_c)]^T \end{aligned} \tag{38}$$

In a manner similar to (31)-(32)) this term evaluates to

$$\frac{\partial \mathbf{v}_0^c(k)}{\partial w_{l,i,j}^a} = \mathbf{Y}(k-1)\mathbf{b}_{l,i,j}^a(k-1) \tag{39}$$

where

$$\mathbf{Y}(k-1) = \begin{bmatrix} \mathbf{I} \\ \mathbf{Y}_1(k-1) \\ \mathbf{Y}_2(k-1) \\ \cdot \\ \cdot \\ \mathbf{Y}_{o_c}(k-1) \end{bmatrix} \tag{40}$$

and

$$\mathbf{Y}_1(k-1) = \begin{bmatrix} 2y(k-1) & 0 & 0 & \cdot\,\cdot & 0 \\ y(k-2) & y(k-1) & 0 & \cdot\,\cdot & 0 \\ y(k-3) & 0 & y(k-1) & \cdot\,\cdot & 0 \\ \cdot & \cdot & \cdot & \cdot\,\cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot\,\cdot & \cdot \\ y(k-o_c) & 0 & 0 & \cdot\,\cdot & y(k-1) \end{bmatrix}_{o_c \; x \; o_c}$$

$$\mathbf{Y}_2(k-1) = \begin{bmatrix} 0 & 2y(k-2) & 0 & \cdot\,\cdot & 0 \\ 0 & y(k-3) & y(k-2) & \cdot\,\cdot & 0 \\ \cdot & \cdot & \cdot & \cdot\,\cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot\,\cdot & \cdot \\ 0 & y(k-o_c) & 0 & \cdot\,\cdot & y(k-2) \end{bmatrix}_{(o_c-1) \; x \; o_c}$$

$$\vdots$$

$$\mathbf{Y}_{o_c}(k-1) = \begin{bmatrix} 0 & 0 & 0 & \cdot & \cdot & 2y(k-o_c) \end{bmatrix}_{1\ x\ o_c}$$

With this (37) becomes

$$\frac{\partial v^c_{M_c}(k)}{\partial w^a_{l,i,j}} = FF^c(\, \mathbf{b}^a_{l,i,j}(k-1)\,) \tag{41}$$

where $FF^c(\cdot)$ represents the operation

$$FF^c(\mathbf{b}) = h^c_{M_c}(k)\, \mathbf{W}^c_{M_c}\ \mathbf{H}^c_{M_c-1}(k)\, \mathbf{W}^c_{M_c-1}\ \cdots\ \mathbf{H}^c_1(k)\, \mathbf{W}^c_1\ \mathbf{Y}(k-1)\, \mathbf{b} \tag{42}$$

Note in this case that $\mathbf{b}^a_{l,i,j}(k-1)$ represents a vector with the same components as in (18), but with dimension o_c rather than o_r. Again, this equation depicts a feed-forward type operation through subnet c.

In summary, for weights in subnet a the α term in (20) is given by

$$\alpha^a_{l,i,j}(k) = BP^a_{l,i,j}(k) + FF^b(\, \mathbf{a}^a_{l,i,j}(k-1)\,) + FF^c(\, \mathbf{b}^a_{l,i,j}(k-1)\,) \tag{43}$$

where $BP_{l,i,j}(k)$ is given in (22)-(25), $FF^b(\cdot)$ is given in (35), and $FF^c(\cdot)$ is given in (42). Note that $FF^b(\cdot)$ and $FF^c(\cdot)$ are the same type of feed-forward operation, one through subnet b and the other through subnet c.

3.2. Weight Updates for Subnet *b*

In this section we derive an expression for $\alpha^b_{l,i,j}(k)$ (see Eq. (19)). For weights in subnet b the first term in (19) is zero, that is

$$\frac{\partial v^a_{M_a}(k)}{\partial w^b_{l,i,j}} = 0 \tag{44}$$

thus,

$$\alpha^b_{l,i,j}(k) = \frac{\partial v^b_{M_b}(k)}{\partial w^b_{l,i,j}} + \frac{\partial v^c_{M_c}(k)}{\partial w^b_{l,i,j}} \tag{45}$$

The derivation of the first term in this expression will resemble that of the corresponding term in the previous section (Eq. (26)) except that now the partial is with respect to a weight in the same subnet. We begin by deriving this expression for the i^{th} weight in the output layer.

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{M_b,i,1}} = \frac{\partial}{\partial w^b_{M_b,i,1}}\left[f\left(\mathbf{W}^b_{M_b}\mathbf{v}^b_{M_b-1}(k) + \mathbf{w}^b_{M_b,0}\right)\right]$$

$$= h^b_{M_b}(k)\frac{\partial}{\partial w^b_{M_b,i,1}}\left[\mathbf{W}^b_{M_b}\mathbf{v}^b_{M_b-1}(k) + \mathbf{w}^b_{M_b,0}\right] \tag{46}$$

Now, since $w^b_{M_b,i,1}$ is a member of the $\mathbf{W}^b_{M_b}$ vector we have,

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{M_b,i,1}} = h^b_{M_b}(k)\left[v^b_{M_b-1,i}(k) + \mathbf{W}^b_{M_b}\frac{\partial \mathbf{v}^b_{M_b-1}(k)}{\partial w^b_{M_b,i,1}}\right] \tag{47}$$

or

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{M_b,i,1}} = h^b_{M_b}(k)\,v^b_{M_b-1,i}(k) + h^b_{M_b}(k)\,\mathbf{W}^b_{M_b}\frac{\partial \mathbf{v}^b_{M_b-1}(k)}{\partial w^b_{M_b,i,1}} \tag{48}$$

The first term in (48) represents the standard backpropagation equation for this weight. The second term is identical in form to the intermediate result obtained in Eq. (27) of the previous section. Thus, carrying out the partial as we did in (28) yields

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{M_b,i,1}} = h^b_{M_b}(k)\,v^b_{M_b-1,i}(k) +$$

$$h^b_{M_b}(k)\,\mathbf{W}^b_{M_b}\,\mathbf{H}^b_{M_b-1}(k)\,\mathbf{W}^b_{M_b-1}\;\cdots\;\mathbf{H}^b_1(k)\,\mathbf{W}^b_1\frac{\partial \mathbf{v}^b_0(k)}{\partial w^b_{M_b,i,1}} \tag{49}$$

Finally, using the notation developed in (22) and (35), (49) can be expressed

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{M_b,i,1}} = BP^b_{M_b,i,1}(k) + FF^b(\,\mathbf{a}^b_{M_b,i,1}(k-1)\,) \tag{50}$$

with $\varepsilon^b_{M_b}(k) = h^b_{M_b}(k)$ in the $BP^b_{M_b,i,1}(k)$ expression. In a similar manner one can show that for *any weight* in subnet *b*,

$$\frac{\partial v^b_{M_b}(k)}{\partial w^b_{l,i,j}} = BP^b_{l,i,j}(k) + FF(\,\mathbf{a}^b_{l,i,j}(k-1)\,) \tag{51}$$

The second term in (45) can be evaluated in much the same way as the third term in the previous section (Eq. (36)). Following the steps outlined in (36)-(42) we find that

$$\frac{\partial v^c_{M_c}}{\partial w^b_{l,i,j}} = FF^c(\mathbf{b}^b_{l,i,j}(k-1)) \tag{52}$$

where $FF^c(\cdot)$ is given in (42). Combining (51) and (52) gives the following results for weights in subnet *b*,

$$\alpha^b_{l,i,j}(k) = BP^b_{l,i,j}(k) + FF^b(\mathbf{a}^b_{l,i,j}(k-1)) + FF^c(\mathbf{b}^b_{l,i,j}(k-1)) \tag{53}$$

Note that the update for the weights in subnet *b* take exactly the same form as those in subnet *a* (see (43)).

3.3. Weight Updates for Subnet *c*

The derivation of $\alpha^c_{l,i,j}$ is very similar to the derivation for $\alpha^b_{l,i,j}$ in the previous section. For $\alpha^c_{l,i,j}$ the first term in (19) is zero,

$$\frac{\partial v^a_{M_a}(k)}{\partial w^c_{l,i,j}} = 0 \tag{54}$$

leaving,

$$\alpha^c_{l,i,j}(k) = \frac{\partial v^b_{M_b}(k)}{\partial w^c_{l,i,j}} + \frac{\partial v^c_{M_c}(k)}{\partial w^c_{l,i,j}} \tag{55}$$

Following the same procedures used in the previous section one can show that

$$\alpha^c_{l,i,j}(k) = BP^c_{l,i,j}(k) + FF^b(\mathbf{a}^c_{l,i,j}(k-1)) + FF^c(\mathbf{b}^c_{l,i,j}(k-1)) \tag{56}$$

In summary then, all weights in the RNN are updated according to (12) with $\beta_{l,i,j}(k)$ given in (14), $\alpha^a_{l,i,j}(k)$ in (43), $\alpha^b_{l,i,j}(k)$ in (53), and $\alpha^c_{l,i,j}(k)$ in (56). Note that the updates differ from regular backpropagation in the $FF(\cdot)$ terms. These are all recursive terms that give $\beta_{l,i,j}(k)$ and $\alpha_{l,i,j}(k)$ as a function of their previous values, and in all cases represent a feed forward operations through one of the *recursive* subnets.

Note that the RNN presented here will have its output bounded by the nonlinearity at the output node. If one's application requires a larger dynamic range, the output node of each subnet can be made linear, that is the nonlinear squashing function $f(\cdot)$ is removed. This means only that $h^s_{M_s,1}(k) = 1$ in the learning algorithm developed above. Also, learning is generally more efficient if the summation in (12) (which represents the negative gradient) is replaced by an instantaneous estimate of the gradient, that is the update takes on the form

$$w^s_{l,i,j}(k+1) = w^s_{l,i,j}(k) + \mu e(k)\, \beta^s_{l,i,j}(k) \tag{57}$$

This is the same type of estimate that is used in algorithms like backpropagation and LMS.

4. Examples

In this section we illustrate the use of the RNN in a variety of applications. These include

1. system identification,
2. nonlinear digital filtering,
3. inverse modeling,
4. control, and
5. nonlinear prediction.

In most of these applications only one or two of the subnets are needed. In fact, only the most general form of the control problem requires all three subnets. An attractive feature of the RNN is that with the proper parameter settings it can easily be reduced to well-known linear filtering structures. In this respect, the RNN can be viewed as a generalization of linear filtering structures to a class of nonlinear digital filters.

The learning algorithm derived in the previous section becomes simpler when one of the subnets is absent. It is also simpler when P=1, as it is in the examples in sections 4.1, 4.2, and 4.5 below. In this case $\mathbf{q}_2 = 0$, $\mathbf{q}_1 = 1$, and (14) simplifies to $\beta^s_{l,i,j}(k) = \alpha^s_{l,i,j}(k)$.

4.1. The RNN in System Identification

In this application subnet *c* is omitted, P=1, and M represents the unknown nonlinear system to be identified (refer to Figure 1). Here the RNN is trained to model the input/output behavior of M. The two subnets *a* and *b* allow the RNN to model both the recursive and nonrecursive behavior of M.

One can obtain a linear IIR filter from the RNN by reducing both subnets to a single linear node. If, in addition, subnet *b* is omitted one obtains a linear FIR filter. Procedures for adapting the weights of both FIR and IIR filters are well known. The gradient learning algorithm which is used to adapt the weights of the RNN reduces to the well known Recursive LMS algorithm when subnets *a* and *b* are linear [1]. This algorithm is known to exhibit instabilities, and correspondingly we have observed occasional instabilities in the learning algorithm for the RNN. Nevertheless this algorithm has served as a useful tool for training the RNN. For the most part we have found that the instabilities can be easily avoided, and have been successful at training the RNN to model several different nonlinear systems.

It is worth mentioning that the RNN is BIBO stable. That is, with the weights of the structure fixed (and bounded), the output is always bounded. This is true even when the output nodes of subnets *a* and *b* are made linear to allow the RNN to produce a larger range of output values. This is due to the contraction property of the nonlinearities in the hidden layers. In contrast, this property does not hold for the linear IIR filter.

As an example we have trained the RNN to model the input/output behavior of the nonlinear system described by the following difference equation

$$y(k) = x(k) + \frac{y(k-1)\, y(k-2) \left[y(k-1) + 2.5 \right]}{1 + y^2(k-1) + y^2(k-2)} \tag{58}$$

For proper generalization the network should be trained with white noise.

In our first simulation we used only linear terms in the input layers to the MLPs. The training signal consisted of uniformly distributed white noise over the range [-2,2]. After some trial and error it was found that the RNN was able to form an good model with a single hidden layer in both subnets. The nonrecursive subnet had 2 hidden layer nodes and the recursive subnet had 30. The inputs to the subnets were both of order 2, that is $o_a = o_b = 2$. This yielded a network with a total of 121 weights. Figure 2 compares the output of the RNN (after training) to the output of the actual system when presented with a sinusoidal input. The results are visually quite good.

In our second simulation we used second order terms at the input to the recursive subnet (i.e. a SOMLP is used for subnet *b*). The input signal to the nonlinear system and the RNN in this case was uniform white noise over the interval [-3,3]. Here we found that the RNN was capable of producing a model with the same accuracy as in the first example with only 1 layer (with 1 node) in the nonrecursive subnet, and 2 layers with 6 hidden layer nodes in the recursive subnet. This implementation contains a total of 46 weights, which is nearly 1/3 the number in the previous case. An example of the operation of this network (after training) is shown in Figure 3. Once again the results are visually quite good.

A closer examination of the manner in which the RNN actually forms the model in this second case suggests that the nonrecursive subnet simply passes the input $x(k)$ trough to the output sum with a gain of 1, and that the recursive subnet attempts to model the recursive part of the system in (58). That is, the recursive subnet attempts to produce a mapping of the form

$$q_3 = \frac{q_1 q_2 \left[q_1 + 2.5 \right]}{1 + q_1^2 + q_2^2} \tag{59}$$

Actually the recursive subnet is not asked to produce q_3 for all possible pairs $[q_1,q_2]$, only those that correspond to valid state trajectories in the nonlinear system. The mapping in (59) is shown in Figure 4 along with the mapping produced by the recursive subnet. We note that the two are remarkably similar.

4.2. The RNN in Nonlinear Filtering

When the RNN is used for nonlinear filtering the network configuration is the same as in the previous section. That is, subnet *c* is omitted and P=1. The difference in this case is that M represents a known nonlinear filtering operation that we wish to implement with the RNN. In the general case both subnets *a* and *b* will be needed. Many popular nonlinear filtering techniques are nonrecursive however, and can be implemented using only subnet *a*. As an example we trained the RNN (subnet *a* only with linear input terms) to act as a 1-D median filter with a window size of 3 samples (i.e. $o_a = 2$). The subnet had two layers with 12 nodes in the hidden layer. These results are illustrated in Figure 5 which shows the input to the filter, the median filter output, and the RNN output.

4.3. The RNN in Inverse Modeling

In this application the configuration in Figure 1 is such that M=1 and P represents the system that we wish to invert. As in the previous applications subnet *c* is omitted. Here the RNN is trained to form an inverse model of P so that the transformation from the input of the RNN to the output of P is 1 (possibly with a fixed delay, that is M may be equal to $z^{-\Delta}$). Obviously P must be invertible before the RNN can be applied successfully in this application.

For the configuration in Figure 1 we note that the RNN is modeling the *left inverse* of P. If the positions of the RNN and P are interchanged the RNN can be used to model the *right inverse* of P. Since P is nonlinear its left inverse and right inverse are not necessarily the same.

To carry out learning in this application requires knowledge of the Jacobian of the plant. If such knowledge is not available it can be estimated by using a separate RNN to form a direct model of the plant (as described previously), and extracting partial derivative estimates from this model.

As an example, the RNN was trained to form an inverse model of the plant described by the difference equation in (58). To insure proper generalization the network was trained with uniform white noise in the range [-3,3]. Both subnets in the RNN had two layers, the nonrecursive subnet had 6 hidden layer nodes, and the recursive subnet had 2. Second-order terms were computed at the input to both subnets and $o_a = o_b = 2$. This resulted in an overall network structure with 82 weights. The plots in Figure 6 show the output of the plant (after training) with the RNN subjected to a different white noise input.

4.4. The RNN in Control

In this application all three subnets may be used. M is a model of our choosing that we would like P to follow. That is, we want the combination of the RNN and P to behave like M. Obviously the plant must be controllable before the RNN can be applied successfully in this application. Also, as in the previous application, the Jacobian of the plant is required in order to carry out learning. Fortunately, in many control applications, the plant is required to follow the model for specific inputs only. This often makes the training task easier since generalization is not as much an issue.

As an example the RNN was trained to make a plant described by the difference equation in (58) follow a linear model described by

$$d(k) = 0.32d(k-1) + 0.64d(k-2) - 0.5d(k-3) + x(k-1)$$

The input in this example was a sinusoid of the form $x(k) = \sin(2\pi k/25)$. Subnet *b* was disabled so that only subnets *a* and *c* were used. Subnet *a* contained a single linear node with $o_a = 2$. Subnet *c* was a two layer network with $o_c = 2$, linear input terms, and 8 hidden layer nodes. Figure 5 compares the output of the model to the output of the plant after training.

The RNN structure here generalizes the Model-Reference-Adaptive-Controllers used in conjunction with linear systems. In fact, in the linear case subnet *a* attempts to match the gain of the plant to that of the model, subnet *b* is used to place the zeros of the plant at the zeros of the model, and subnet *c* moves the poles of the plant to the poles of the model. The RNN in the control of nonlinear systems is thus capable of making the plant follow a desired model. Unlike the linear case however, no stability proofs are available yet and in fact, we can not analytically determine whether an RNN controller exists to make a certain plant follow a given model. These and related questions are topics of further research.

4.5. The RNN in Nonlinear Prediction

When the RNN is used for prediction the structure in Figure 1 is such that P=1, M=1, and subnet *c* is omitted. This leaves only the RNN with subnets *a* and *b*. For Δ-step prediction the input to subnet *a* is delayed by Δ. Thus, the RNN is asked to use past inputs $x(k-\Delta)$, $x(k-\Delta-1)$, etc. to predict the current input, $x(k)$. As an example the RNN was trained to perform 2-step prediction of the chaotic sequence

$$x(k) = 4.0\, x(k-1)\, [1.0 - x(k-1)]$$

In this example only subnet a was used. It had three layers with $o_a = 2$, second order input terms, 8 nodes in the first hidden layer, and 4 nodes in the second hidden layer. The results are illustrated in Figure 8 which shows the actual and predicted sequences after training.

5. Analysis of the RNN

The RNN is a nonlinear, time-varying (during adaptation) dynamical system. As such, it is extremely difficult to analyze. This section will address some of the theoretical questions regarding the capabilities and the properties of the RNN. Many questions will be left unanswered as they are topics for further research.

We start out by stating our philosophy that the RNN is effectively a discrete-time dynamical system emulator. In other words, in all suggested applications (Modeling, Filtering, and Control), the RNN is attempting to approximate the desired input/output behavior of a discrete-time dynamical system. The first question then is whether the RNN structure is capable of such behavior, i.e. can the RNN form an arbitrarily close approximation to the behavior of the desired dynamical system? The theoretical foundation for answering these questions belong in the realm of functional analysis. Our examples and simulations suggest that the RNN is capable of approximating a large class of discrete-time systems, namely those systems whose recursive and nonrecursive parts are additive. Let us note that although the RNN is dynamical, its dynamics are separated from the nonlinearities, and as far as its approximation capabilities are concerned, classical results such as the Stone-Weierstrass theorem [11] apply. In fact, if we examine the RNN structure carefully, we note that if a sufficient number of delays are included at the input of each RNN subnet, the current (and even previous) states are reconstructed so that the static nonlinear part of the network is asked to approximate the characteristics of an arbitrary nonlinear function. The ability of the MLP to effect such approximations has been shown in [12]. The RNN structure presented here however, will fail to model nonlinear systems with cross-term nonlinearities such as $y(k) = y(k-1)*u(k-1) + 1$. This limitation may be easily remedied by including an additional static mapping that accepts the outputs of the three subnets as inputs and produces a nonlinear function of these as the new RNN output. The requirement remains however, that the desired dynamical system be well-behaved. For example, in the inverse modeling problem, the RNN will certainly fail when attempting to invert a noninvertible system. The RNN will also fail to follow the trajectory of a plant in the neighborhood of its unstable equilibrium points. Note that the discussion of this section concerns the discrete-time version of the RNN presented in this report. In the case of a continuous-time RNN, the reconstruction of a state-vector using delay elements is more involved as was shown by Takens [13].

Next, we shall concern ourselves with the issue of whether the RNN is needed to approximate a particular system and whether simpler structures (polynomials, orthogonal expansions, etc.) may not be sufficient. Our philosophy is that the RNN should only be used when other established techniques (with guaranteed convergence and stability properties) either fail or require a large number of terms. The RNN is then an alternative rather than a substitute to more established functional approximation techniques.

Some of the remaining questions concern the required number of delays and the size of the RNN for solving a particular problem. Assuming that the recursive and non-recursive orders of the nonlinear plant are exactly known, the number of delay elements are easily obtained. The size of the required network (the number of layers and the number of nodes) is determined by the nature of the problem. As described in [12], one hidden layer is sufficient to approximate a given static mapping if it contains enough nodes. For the examples, we have started with one node, and continued to increase the size of the network until a satisfactory response was obtained.

The next theoretical issue that needs to be addressed is that of the convergence of the learned weights to those required for satisfactory approximation. There are two different aspects to this particular issue. First, the convergence of the gradient descent method in a nonlinear setting which is dependent on the specific problem and the adaptation step size remains an open problem. Second, the fact that the algorithm actually implemented uses an approximation of the true gradient rather than its exact expression. One can of course use the true gradient at the expense of slowing down the adaptation process. Based on our experimentation, it seems that by using an adaptive step size, the algorithm may be made to asymptotically converge to a fixed set of weight values. The question then arises as to whether this set is the one that minimizes the performance objective. This question remains open and will be the topic of future research.

When the recursive subnet c is activated, as in control applications, the stability question becomes critical. The RNN is now acting as a nonlinear controller to a nonlinear system, with a feedback connection that can effectively change the dynamics of the open-loop plant. In fact, a stable nonlinear plant may be driven unstable by a stable RNN placed in its feedback path. The question runs even deeper because the stability of the closed-loop system is not uniquely defined: A system which may be shown to be Lyapunov stable (i.e. stable with some initial conditions but no external input) may become unstable when presented with a bounded input. In fact, the number and properties of the equilibrium points of the closed-loop system will vary with different external inputs. Currently, the best approach to guaranteeing the stability of nonlinear systems with external inputs is to use Lyapunov theory. The results however, will not hold when a different input is presented to the system. Recently, an update law was derived for recurrent networks using Lyapunov function techniques [14] similar to those investigated in the adaptive control literature [15]. The RNN does not possess any guaranteed stability properties since the update rule is an approximation to the gradient descent method as described above. Other update rules using a Lyapunov approach are currently being investigated.

Finally, we note that the usage of the RNN as a nonlinear controller assumes that the nonlinear plant is controllable, so that a given model may be followed. The theoretical difficulties behind such an assumption are related to the controllability question of nonlinear systems. Considerable progress has been made in this area for continuous-time nonlinear systems [16] but discrete-time results are still lacking.

As an aside, it can be shown that a constant nonlinear mapping can match a time-varying linear one. This observation makes it possible for the recursive neural network to potentially replace time-varying linear filters and to stop updating the weights after a certain degree of performance is achieved.

In summary, we recognize that the theoretical development for the RNN is far from complete. In fact, we feel that the RNN is similar in its maturity level to the MIT rule approach to adaptive control [15]. It will take a large concentrated effort for the RNN and similar techniques to mature into a new design method for the identification and control of nonlinear systems.

6. Summary

We have introduced a nonlinear dynamical system called the recursive neural network. We have also presented a learning algorithm for this network which is based on a gradient search. The network has been shown to be useful in a variety of applications including system modeling, nonlinear filtering, inverse modeling, nonlinear prediction, and control.

The recursive neural network was motivated by problems and concepts from nonlinear filtering and control. It closely resembles the architectures proposed in [3]. The work here represents the algorithmic equivalent of the work in [3] and can be viewed as a complement to that work.

The RNN is similar in capability to the recurrent neural networks discussed in [4-6], in that both networks are nonlinear dynamical structures that contain adjustable weights. The RNN differs from recurrent neural networks in that the dynamics (the delays) are external to the nodes, that is the dynamics are separated from the nonlinear part of the structure. Thus the state of the two networks are different. In the RNN the states are created by feeding the input and output of the net through an observer composed of the o_n and o_r delays. This is similar to the idea of passing the output of a dynamical system through an observer in order to reconstruct the state before implementing state-feedback in control systems [7]. In recurrent neural networks every node in the network has access to its own state and assumes the ability of directly measuring the state of a system.

It should be cautioned that this paper and the simulated examples are not meant to illustrate the universality or the ease of using the RNN. Indeed, we can point out that the behavior of the closed-loop system is sensitive to many design parameters and modeling assumptions. The fact remains however, that after extensive simulation runs, the RNN presents itself as a useful alternative when analytical methods are lacking. In addition, the general model-following problem investigated in this paper (when neither the plant nor the model is 1) assumes that the Jacobian matrix of the nonlinear plant is available. If the plant is actually unknown, a gradient estimator scheme is needed in order to carry out the update laws. Work on this and related issues is currently in progress.

7. References

[1] Widrow, B. and Stearns, S.D., *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1985.

[2] Rumelhart, D.E., McClelland, J.L., and Williams, R.J., Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

[3] Narendra, K.S., and Parthasarathy, K., "Identification and control of dynamical systems using neural networks," *IEEE Trans. on Neural Networks,* Vol. 1, No. 1, pp. 4-27, March, 1990.

[4] Pineda, F.J., "Generalization of backpropagation to recurrent neural networks," *Physical Review Letters,* Vol. 18, pp. 2229-2232, 1987.

[5] Pineda, F.J., "Recurrent backpropagation and the dynamical approach to adaptive neural computation," *Neural Computation,* Vol. 1, pp. 161-172, 1989.

[6] Williams, R.J., and Zipser, D., "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation,* Vol. 1, pp. 270-280, 1989.

[7] Kailath, T., *Linear Systems,* Prentice-Hall, Inc., Englewood Cliffs, N.J., pp. 259-313, 1980.

[8] Parks, P.C., "Lyapunov redesign of model reference adaptive control system," *IEEE Trans. Automatic Control,* Vol. AC-11, pp. 362-367, July 1966.

[9] L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics,* Vol. 26, No. 23, pp. 4972-4978, 1987.

[10] D. Hush, C. Abdallah, and B. Horne, "The Recursive Neural Network and its Applications in Control Theory," *to appear in Computers and Electrical Engineering.*

[11] W. Rudin, *Functional Analysis,* McGraw-Hill, New York, 1973.

[12] C. Cybenko, "Approximation by Superpositions of Sigmoidal Function," *Mathematics of Control, Signals, and Systems,* Vol. 2, pp. 303-314, 1989.

[13] F. Takens, "Detecting Strange Attractors in Fluid Turbulence," D. Rand & L.-S. Young, Editors, *Dynamical Systems and Turbulence,* Springer-Verlag, Berlin 1981.

[14] J. Barhen, N. Toomarian and S. Gulati, "Adjoint Operator Algorithms for Faster Learning in Neural Networks," *Adv. Neur. Inf. Proc. Sys.,* Vol. 2, pp. 498-508, 1990.

[15] K.S. Narendra, and A.M. Annaswamy, *Stable Adaptive Systems,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1989.

[16] A. Isidori, *Nonlinear Control Systems,* Second Ed., Springer-Verlag, Berlin, 1989.