Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks

Kumpati S. Narendra, *Fellow, IEEE*, and Kannan Parthasarathy, *Student Member, IEEE*

***Abstract*—Multilayer neural networks have been used successfully in pattern recognition problems, and numerous applications have been suggested in the literature. Back propagation is one of the standard methods used in these cases to adjust the weights (parameters) of the neural networks. In a recent paper the authors suggested the use of multilayer neural networks for the identification and control of nonlinear dynamical systems and proposed an extension of the back-propagation method, termed dynamic back propagation, to such problems. Further, the authors feel that multilayer neural networks will find increasing use as subsystems in a variety of dynamical contexts in the coming years. The aim of the paper is to discuss in detail the dynamic back-propagation method, so that it can be applied in a straightforward manner for the optimization of the parameters of these multilayer neural networks. Emphasis is placed on the diagrammatic representation of the system which generates the gradient of the performance function, to facilitate the practical implementation of the proposed method.**

I. Introduction

The use of multilayer neural networks for pattern recognition is currently well known and several applications have been reported in the literature. Given a training set of input-output pairs which define a static function, such a network is considered to "learn" that function. The learning process involves the determination of the weights (denoted by the vector θ), so that the approximation of the given function is achieved in some optimal fashion.

Recently, interest has also been increasing in the use of multilayer neural networks as elements in dynamical systems. This has been motivated by two considerations. Since all physical systems involve dynamics, modeling a physical system such as a (natural) neural network should realistically include dynamical elements. Further, from a control point of view, if the powerful concept of feedback is to be introduced in neural networks in the form of feedback networks, dynamical elements have to be included to have well-posed problems. The models introduced by the authors for the identification and control of nonlinear dynamical systems [1] belong to this class.

The method commonly used to evaluate the gradient of a performance function J with respect to a weight vector θ of multilayer neural networks in static problems is called back propagation. In this paper we discuss the extensions of this method to nonlinear dynamical systems composed of linear time-invariant dynamical systems and multilayer neural networks. While this method, termed dynamic back propagation in this paper, was introduced in [1], it has also been suggested in the past by other authors. In [2] Williams *et al.* considered a learning algorithm for a continually running fully recurrent neural network, while in [3] Werbos proposed a method for using "back-propagation through time." Since neural networks and linear dynamical systems can be interconnected in arbitrary configurations in complex systems, dynamic back propagation was explained in [1] in terms of specific simpler configurations. However, because of space limitations, the treatment had to be kept brief. In this paper a detailed study of dynamic back propagation is presented to provide a clear understanding of the principal ideas that contributed to the evolution of the concept and the details concerning its practical implementation, with attention also given to its advantages and limitations.

Gradient methods for the optimization of parameters in static and dynamic systems are based on the notion of a partial derivative. In Section II, the well-established method using such partial derivatives to optimize a performance index is reviewed. The determination of the partial derivatives of a performance criterion with respect to the parameters of linear dynamical systems was studied extensively in the 1960's and several theoretically important and practically useful results were derived [4]-[7]. To provide historical continuity, as well as to provide the basis for the method discussed in the following sections, Section III deals briefly with such systems. It is then shown that the same concepts can be readily extended to suitably parameterized nonlinear dynamical systems, e.g. systems composed of multilayer neural networks and linear dynamical systems. The four different representations introduced in [1] are considered in Section IV, and in each case the details for generating the gradient of a performance function along with simulation studies are presented. In Section V, the control of a nonlinear plant using dynamic back propagation is illustrated.

II. Gradient Methods for the Optimization of Static Systems

A. The Gradient of a Function

Let $f(\theta) = f(\theta_1, \cdots, \theta_n)$ be a scalar function of n variables $\theta_1, \theta_2, \cdots, \theta_n$, where θ_i are the elements of the vector θ. The gradient of $f(\theta)$ with respect to the vector θ is defined as the row vector

$$f_\theta \triangleq \left[\frac{\partial f}{\partial \theta_1}, \cdots, \frac{\partial f}{\partial \theta_n}\right].$$

The value of the gradient depends upon the point $\theta_{\text{nom}} \in \mathbb{R}^n$ (denoting the nominal value of θ) at which f_θ is evaluated. If the operating point is changed from θ_{nom} to $\theta_{\text{nom}} + \Delta\theta$, where $\Delta\theta = \eta f_\theta^T$, $\eta \ll 1$ a positive constant, it follows that $f(\theta_{\text{nom}} + \Delta\theta) \geq f(\theta_{\text{nom}})$ (since $f(\theta_{\text{nom}} + \Delta\theta) \approx f(\theta_{\text{nom}}) + \eta f_\theta f_\theta^T$). It is this concept that is used in all gradient methods for the

Manuscript received August 4, 1990; revised November 16, 1990. This work was supported by the National Science Foundation under Grant ECS-8912397 and by Sandia National Laboratories under Contract 86-0253.

The authors are with the Center for Systems Science, Department of Electrical Engineering, Yale University, New Haven, CT 06520-1968.

IEEE Log Number 9041956.

optimization of performance indices in static and dynamical systems. If $f : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}$, the gradient of $f(\theta, t)$ with respect to θ is a time-varying function and it is with such functions that we will be concerned in this paper.

B. Optimization Using the Gradient Method

Let $J[\theta]$, a performance index which has to be optimized with respect to a parameter vector θ, be defined as the functional

$$J(\theta) = \frac{1}{T} \int_0^T L[e(\theta, \tau)] d\tau \tag{1}$$

in the continuous case and as

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L[e(\theta, i)] \tag{2}$$

in the discrete case. The error e in (1) and (2) is assumed to be a function of time as well as the parameter θ. If the gradient method is used to minimize the performance index, $\partial J/\partial\theta$ is determined, by the chain rule, as

$$\frac{\partial J}{\partial \theta} = \frac{1}{T} \int_0^T \left[\frac{\partial L[e(\theta, \tau)]}{\partial e(\theta, \tau)}\right] \left[\frac{\partial e(\theta, \tau)}{\partial \theta}\right] d\tau$$

or, alternatively, $1/N \sum_{i=1}^{N} [\partial L[e(\theta, i)]/\partial e(\theta, i)] [\partial e(\theta, i)/\partial\theta]$ in the discrete case. In both cases θ is adjusted as $\theta = \theta_{\text{nom}} - \eta\, \partial J/\partial\theta$ and the process is repeated.

In practice, however, the parameter θ in a dynamical system is adjusted on-line and the system cannot be reinitialized. The performance index is then defined over a finite interval $[t - T, t]$ for continuous-time systems or a finite interval $[k - T + 1, k]$ for discrete-time systems. Further, a commonly used function $L[e(\theta, \tau)]$ has the form $\|e(\theta, \tau)\|^2$, where $e(\theta, \tau)$ is an output error vector. For the discrete-time case, $J(\theta)$ has the form

$$J(\theta) = \frac{1}{T} \sum_{i=k-T+1}^{k} e^T e \tag{3}$$

and the gradient is computed over the interval $[k - T + 1, k]$ as

$$\frac{\partial J}{\partial \theta} = \frac{2}{T} \sum_{i=k-T+1}^{k} e^T \left[\frac{\partial e}{\partial \theta}\right].$$

The parameter θ is then adjusted as

$$\theta(k+1) = \theta(k - T + 1) - \eta \frac{2}{T} \sum_{i=k-T+1}^{k} \left[\frac{\partial e}{\partial \theta}\right]^T e \tag{4}$$

where $\eta \ll 1$, the step size, is a suitably chosen constant. In (3) and (4), the explicit dependence of e on θ and i is omitted for simplicity of notation. Equation (4) implies that the gradient evaluated over the interval $[k - T + 1, k]$ is applied to the interval $[k + 1, k + T]$. Further, strictly speaking, θ is no longer a constant parameter but a function of time so that the concept of a partial derivative described earlier has to be replaced by the concept of a functional derivative. However, if η is sufficiently small, we shall assume that the concept of a partial derivative (or gradient in parameter space) can still be applied. A similar expression for the gradient can also be derived for the continuous-time case.

From (4) it is clear that the gradient method described above can be implemented if $\partial e(i)/\partial\theta$ can be computed on-line for all values of i in the interval $[k - T + 1, k]$. In multilayer neural networks this partial derivative is computed using back propagation. The importance of the choice of the parameter T in practical problems will become evident from the simulation studies included in Section IV.

C. Multilayer Neural Networks and Back Propagation

A typical three-layer network whose input is the vector u and output is the vector y is shown in Fig. 1. The nonlinear transformation $\gamma(x)$ in each layer is usually chosen to be a sigmoidal function of the form $\gamma(x) = (1 - e^{-x})/(1 + e^{-x})$. The input-output mapping of the multilayer neural network is parametrized as[1]

$$y = N[u] = \Gamma\left[W_3\Gamma\left[W_2\Gamma\left[W_1u + b_1\right] + b_2\right] + b_3\right]$$

where Γ is a diagonal nonlinear operator with identical elements γ, and W_1, W_2, W_3 and b_1, b_2, b_3 are the weight matrices and bias vectors respectively. If such a multilayer neural network is used in a pattern recognition problem, the weight matrices and bias vectors are adjusted in such a manner that the mapping function $N[u]$, realized by the network, maps input vectors into the corresponding output classes. If, for a given input u, the desired output is y_d, the output error vector is defined as $e = y - y_d$. If the patterns are assumed to be presented at every instant of time, both inputs and outputs can be viewed as time sequences and the performance criterion given in (3) can be used.

The back-propagation method for the adjustment of the parameters of a multilayer neural network along the gradient of a performance criterion is well established in the literature. In Fig. 1, a sensitivity network is shown for computing the gradient. While this network is merely a realization of the back-propagation method and is equivalent to it from an information theory point of view, it is found to have substantial advantages in dynamic situations where the algorithmic representation becomes cumbersome. This will become evident in Sections IV and V, where such sensitivity networks are used in all the discussions.

III. Gradient Methods for Dynamical Systems

A. Linear Systems

As mentioned earlier, in Section I, gradient methods were extensively used in the 1960's for the optimization of linear dynamical systems. The interesting concepts generated and the results derived during this period are relevant in the context of problems currently being investigated using neural networks and hence form a convenient starting point for our discussion of gradient methods in dynamical systems.

Example 1: Consider a system described by the scalar differential equation

$$\ddot{y} + \alpha\dot{y} + y = u \tag{5}$$

where the parameter α is to be adjusted so that the performance function

$$J(\alpha) = \frac{1}{T} \int_0^T [y(\tau) - y_d(\tau)]^2 d\tau$$

[1]In [1] for ease of exposition the bias terms were not included in figures 2 and 7. However, bias terms were used in all the simulations that were reported. The expression given here, as well as Fig. 1, includes bias terms explicitly.

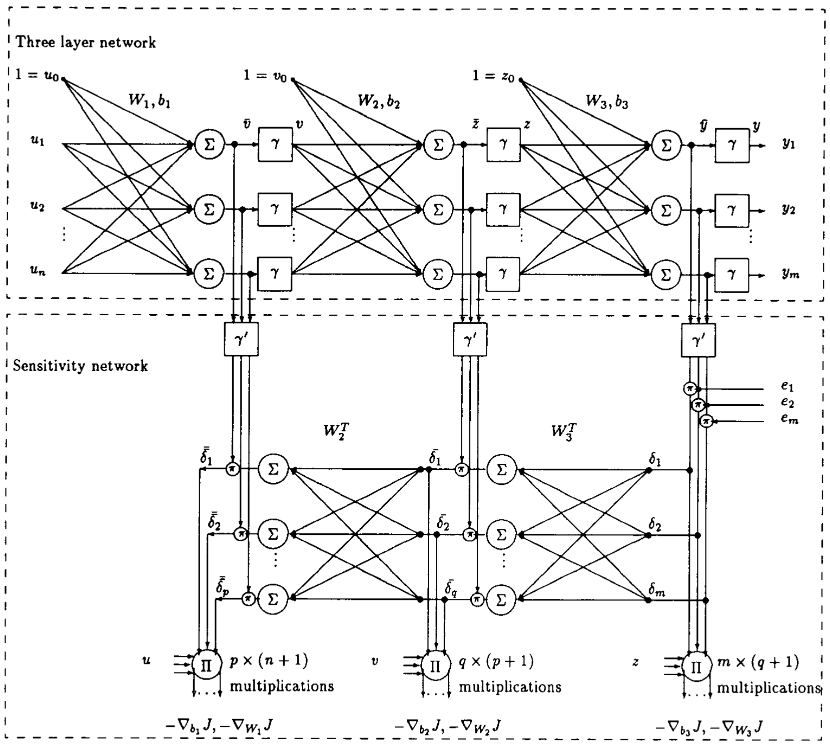

Three layer network
$1 = u_0$
W_1, b_1
u_1
u_2
u_n
$1 = v_0$
W_2, b_2
$1 = z_0$
W_3, b_3
y_1
y_2
y_m
Sensitivity network
W_2^T
W_3^T
e_1
e_2
e_m
$p \times (n+1)$ multiplications
$q \times (p+1)$ multiplications
$m \times (q+1)$ multiplications
$-\nabla_{b_1} J, -\nabla_{W_1} J$
$-\nabla_{b_2} J, -\nabla_{W_2} J$
$-\nabla_{b_3} J, -\nabla_{W_3} J$

Fig. 1. Architecture for back-propagation.

is to be minimized, where $y_d(t)$ is the desired response defined over the interval $[0, T]$. Since J_α can be expressed as

$$J_\alpha(\alpha) = \frac{1}{T} \int_0^T 2[y(\tau) - y_d(\tau)] \frac{\partial y(\tau)}{\partial \alpha} d\tau \tag{6}$$

the computation of the desired gradient involves the determination of the partial derivative $\partial y(t)/\partial \alpha$ as a function of time. Assuming that the latter can be generated, J_α can be computed using (6), and an algorithm of the form

$$\alpha(T) = \alpha(0) - \eta J_\alpha(\alpha)$$

can be used to adjust the parameter α. After each adjustment, the system must be reinitiated and J_α determined. In practice, if $u(\cdot)$ is assumed to be a stationary random process, α can be adjusted on-line by determining the gradient of $1/T \int_{t-T}^{t} [y(\tau) - y_d(\tau)]^2 \, d\tau$ and using it to update α at discrete intervals of time as

$$\alpha(t) = \alpha(t - T) - \eta \frac{2}{T} \int_{t-T}^{t} [y(\tau) - y_d(\tau)] \frac{\partial y(\tau)}{\partial \alpha} d\tau.$$

If $\alpha(t)$ is adjusted continuously, the updating equation has the form $\dot{\alpha}(t) = -\eta[y(t) - y_d(t)] \partial y(t)/\partial \alpha$.

Generation of $\partial y(t)/\partial \alpha$: Taking the partial derivative of the two sides of (5) and assuming that d/dt and $\partial/\partial \alpha$ commute, we obtain

$$\frac{\partial \ddot{y}(t)}{\partial \alpha} + \alpha \frac{\partial \dot{y}(t)}{\partial \alpha} + \frac{\partial y(t)}{\partial \alpha} = -\dot{y}(t) \qquad \frac{\partial y(t_0)}{\partial \alpha} = \frac{\partial \dot{y}(t_0)}{\partial \alpha} = 0$$

where $\partial \ddot{y}/\partial \alpha = d^2/dt^2[\partial y(t)/\partial \alpha]$. Hence, $\partial y(t)/\partial \alpha$ is obtained as the output of a dynamical system described by (5) but with input $u(t)$ replaced by $-\dot{y}(t)$. In view of this, if $\dot{y}$ is known at every instant, the method outlined earlier can be implemented for the optimization of the parameter α with respect to a given performance index $J(\alpha)$.

In any complex linear time-invariant system, a general result based on the above procedure can be stated for the adjustment of a single parameter along the gradient [8]. This important result is shown diagrammatically in Fig. 2. Let S be a linear time-invariant system with $u(\cdot)$ as its input, $e(\cdot)$ as its output, and α as a parameter. Let S_M be an exact model of the system with input identically zero. If the input to the element α in S is made the input to the model S_M at the output of α (in S_M) as shown in Fig. 2, the output of S_M is the desired partial derivative $\partial e(t)/\partial \alpha$. We shall refer to the system within the dotted lines which generates this partial derivative as the sensitivity model.

The generation of the partial derivative of $e(t)$ w.r.t. the parameter α in Fig. 2 assumes that α is a constant. If the generated time signal is in turn used to adjust the parameter α, the above assumption no longer holds and the parameter α becomes a state variable, making the overall system nonlinear. In such a case, the stability of the overall system cannot be ensured. It is therefore important to assume a sufficiently small step size η to ensure that the conditions for determining the gradient are approximated.

It is clear from the above discussion that one model of the system will be needed for the generation of each partial derivative $\partial e(t)/\partial \alpha$. Alternatively, if multiple parameters are to be adjusted, multiple models will have to be used, i.e., one for

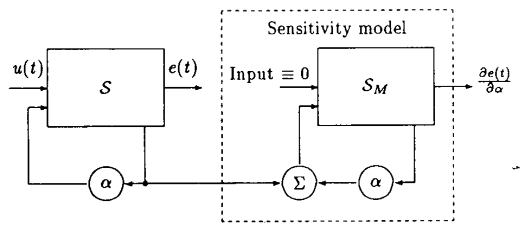


Fig. 2. Generation of the partial derivative using a sensitivity model.

each parameter. However, it was shown independently by Narendra and McBride [4], [6] and Kokotovic [5] that in many cases the important result discussed earlier can be modified so that the entire gradient in parameter space can be generated, for linear time-invariant systems, using a single model. This was further extended to multivariable systems by Cruz *et al.* [9]. All these results have been collected in a volume on sensitivity theory edited by Cruz [7].

B. Nonlinear Systems

The same approach used in subsection III-A for determining the gradient of a time-varying function $y(t)$ with respect to a parameter α using a sensitivity model can be directly extended to nonlinear dynamical systems as well. However, the sensitivity models, unlike those used for linear systems, cannot be directly obtained from the equations describing the plant but involve the Jacobian matrices of the corresponding nonlinear functions, as shown in this section. This is illustrated in Examples 2 and 3.

Example 2: Consider the nonlinear dynamical system described by the second-order scalar differential equation

$$\ddot{y} + F(\alpha, \dot{y}) + y = u \qquad y(0) = y_{10}, \dot{y}(0) = y_{20} \quad (7)$$

where α is a scalar parameter to be adjusted. As in subsection III-A, the problem is to optimize a performance index of the form $1/T \int_0^T [y(\tau) - y_d(\tau)]^2 \, d\tau$, where $y_d(t)$ is a specified function of time, by the choice of α. The implementation of the gradient procedure once again involves the determination of the function $\partial y(t)/\partial\alpha$ over the interval $[0, T]$. However, in contrast to the linear case, it is seen that $\partial y/\partial \alpha$ satisfies the nonlinear differential equation

$$\frac{\partial \ddot{y}}{\partial \alpha} + \left[\frac{\partial F}{\partial \dot{y}}\frac{\partial \dot{y}}{\partial \alpha} + \frac{\partial F}{\partial \alpha}\right] + \frac{\partial y}{\partial \alpha} = 0$$

$$\frac{\partial y(0)}{\partial \alpha} = \frac{d}{dt}\left[\frac{\partial y(0)}{\partial \alpha}\right] = 0$$

or

$$\ddot{z} + \frac{\partial F}{\partial \dot{y}}\dot{z} + z = -\frac{\partial F}{\partial \alpha} \qquad z(0) = \dot{z}(0) = 0 \quad (8)$$

where $z = \partial y/\partial\alpha$. Hence, to determine $\partial y(t)/\partial\alpha$ as a function of time, $\partial F/\partial \dot{y}$ and $\partial F/\partial \alpha$ have to be computed on-line and used in the linear time-varying model described by (8). If, in (7), y as well as F are vector functions, the sensitivity model described by (8) will involve Jacobian matrices.

Example 3: Consider the nonlinear vector differential equation

$$\dot{x}(t) = f[x(t), \alpha, u(t)] \qquad x(t_0) = x_0$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, and α is a scalar parameter. For a given input $u(t)$ over the interval $[t_0, T]$, let α_{nom} and $x_{\text{nom}}(t)$ represent the nominal values of the parameter α and the state $x(t)$. The gradient of $x(t)$ with respect to α around the nominal value α_{nom} is given by the linearized equation

$$\frac{\partial \dot{x}(t)}{\partial \alpha} = f_x(t)\frac{\partial x(t)}{\partial \alpha} + f_\alpha(t) \qquad \frac{\partial x(t_0)}{\partial \alpha} = 0$$

where $f_x(t) \in \mathbb{R}^{n \times n}$ and $f_\alpha(t) \in \mathbb{R}^n$ are Jacobian matrices evaluated around the nominal values $(x_{\text{nom}}(t), u(t), \alpha_{\text{nom}})$.

In both Examples 2 and 3, it is clear that the relevant nonlinear functions must be known explicitly so that the required Jacobian matrices can be computed. In many of the problems of interest to us, such explicit information may not be available. However, if the nonlinear maps are represented by multilayer neural networks with the parametrization described in subsection II-C, the Jacobian matrices can be readily computed using the network shown in Fig. 1. It is this fact that makes neural networks particularly attractive in the optimization of nonlinear systems.

IV. Dynamic Back Propagation in Generalized Neural Networks

Generalized neural networks, as defined in [1], contain multilayer neural networks and linear dynamics as subsystems. Since, from Sections II and III, it is evident that gradient methods can be applied to both classes of subsystems, it follows that back propagation can, in theory, be extended to generalized neural networks. This is referred to as dynamic back propagation. Since a large class of complex generalized neural networks can be generated using four simpler prototypes introduced in [1], we shall indicate how the approach can be used for these simple cases. Anyone familiar with the application of the back-propagation approach to these configurations should also be able to use the method in more complex situations.

With regard to multilayer neural networks, in the discussion of the sensitivity models included in this section, the effects of changes in parameters as well as changes in inputs of these networks on the outputs have to be determined frequently. The method outlined in subsection II-C can be directly used for the former. A simple modification of the method also enables the method to be applied to the latter case. For example, let $y(t) = \theta_1 f[\theta_2 u(t)]$. Then, the variation $\delta y(t)$ in the output is given by $\{\theta_1 f'[\theta_2 u]\theta_2\}\delta u(t)$. Hence, the functional derivative of y w.r.t. u is $\theta_1 f'[\theta_2 u]\theta_2$. Similarly for the more general case of the three-layer neural network considered in subsection II-C, the derivative of one of the outputs y_i w.r.t. u can be expressed as $\bar{\delta}^T W_1$, where u is the vector input to the network. We shall use this frequently in this section for the determination of partial derivatives in the four different representations. If the multilayer neural network has more than one output, a separate gradient generating network of the type shown in Fig. 1 has to be used to compute the partial derivative of each of the outputs with respect to the input vector. In all cases, we shall assume that the reader is familiar with back propagation for static networks as well as the network discussed in subsection II-C for generating the desired partial derivatives.

Representation 1: $y = W(z)N[u]$

A dynamical system in which a neural network (denoted by N) is connected in cascade with a multivariable linear time-

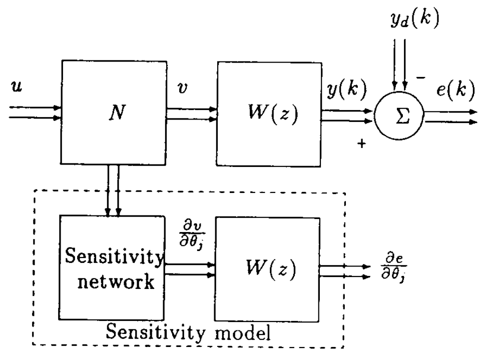


Fig. 3. Generation of the gradient in Representation 1.

invariant system (denoted by a discrete transfer matrix $W(z)$) is shown in Fig. 3. While $N: \mathbb{R}^n \to \mathbb{R}^L$ represents a static map, $W(z)$ is an $(M \times L)$ matrix of rational transfer functions so that the dynamical system has M outputs. The application of a fixed input at time $k = 0$ results in a vector output function of time $y(k)$ defined for all $k \in \mathfrak{N}$ (the set of all nonnegative integers). The desired output $y_d(k)$ as well as the error $e(k)$ are also functions of time and the latter depends upon a parameter vector θ (corresponding to the weights of the neural network). If the performance criterion J can be expressed as a functional of e, our interest is in determining $\partial e(k)/\partial \theta$ as a function of time. The gradient of the performance criterion J w.r.t. θ can then be computed from a knowledge of $\partial e(k)/\partial \theta$. In the following, we indicate how the desired partial derivatives with respect to a typical parameter θ_j of the neural network can be computed.

Since $y = W(z)N[u]$ and $e(k) = y(k) - y_d(k)$, it follows that

$$\frac{\partial e(k)}{\partial \theta_j} = \frac{\partial y(k)}{\partial \theta_j} = W(z)\frac{\partial v}{\partial \theta_j}$$

where $W(z)$ and $v(k)$ are as shown in Fig. 3. Since $\partial v/\partial \theta_j$ can be computed using static back propagation, the linear dynamic operator $W(z)$ operating on $\partial v/\partial \theta_j$ yields the desired partial derivatives. It is also clear that the output of the linear operator at any time $k > 0$ represents the partial derivative of e at time k with respect to the parameter, where the change in parameter is assumed to be made at time zero. Assuming that the network shown in Fig. 1 for computing $\partial v/\partial \theta_j$ is available, the desired partial derivatives $\partial e/\partial \theta_j$ can be generated as functions of time by using $W(z)$ in cascade as shown in Fig. 3. Hence, this can be considered as the sensitivity model for the system $W(z)N[u]$.

Simulation 1: Simulation studies carried out using Representation 1 included in this section indicate how the scheme described earlier can be implemented in practice. Consider the dynamical system which has the form

$$y_d(k) = W(z)f[u(k)]$$

where $f[u] = 4u/1 + 4u^2$ is assumed to be unknown and $W(z)$ is a specified scalar pulse transfer function. Let the model used to estimate the function f have the form

$$y(k) = W(z)N[u(k)]$$

where $N[\cdot] \in \mathfrak{N}^3_{1,20,10,1}$ is a multilayer neural network which is assumed to be sufficiently large to approximate $f[\cdot]$ arbitrarily closely. If $e(k) \triangleq y(k) - y_d(k)$ is the error at time k, the partial derivative $\partial e(k)/\partial \theta_j$ with respect to a typical parameter θ_j of N is needed to implement the method described earlier. We consider three cases of increasing generality in the following.

Case 1 $W(z) = z^{-d}$: The simplest situation arises when the linear dynamical transfer function is a pure delay of d units. The procedure outlined earlier involves merely the determination of the partial derivative $\partial v/\partial \theta_j$ (using static back propagation) and delaying it by d units of time.

Case 2 $W(z) = \Sigma_{i=1}^{d} \alpha_i z^{-i}$: In this case $W(z)$ is assumed to have a finite pulse response of duration d units. The desired partial derivative cannot be obtained merely by using static back propagation but is determined by taking a linear combination of its delayed values.

Case 3 *$W(z)$: A Stable Rational Transfer Function:* This represents the most general case of interest to us. $W(z)$ has a pulse response which lasts for all $k > 0$. In this case the procedure described earlier and depicted in Fig. 3 has to be used.

Discussion: When $W(z) = z^{-d}$ or $\Sigma_{i=1}^{d} \alpha_i z^{-i}$, the parameter must be kept constant over intervals of length d units. In practice, however, it is more common to adjust the parameter at every instant. The assumption in this case is that the step size η is sufficiently small so that the parameter does not vary significantly over the interval. If a large value of the step size is used, violating the above assumption, instability may result. Theoretically, it is also possible to adjust the parameter at time k along the negative gradient of a performance function of the form $J = 1/T \Sigma_{i=k-T+1}^{k} e^2(i)$, which depends upon the error at the previous T instants. The period T is chosen to be larger than the interval over which the pulse response of the system is significant. All the above comments carry over to the general case where $W(z)$ is a rational transfer function. In all cases, the choice of T and the step size η are based on a compromise between speed and accuracy.

The simulation results for the system described by $W(z)f[u]$, where $f[u] = 4u/1 + 4u^2$, are shown in Figs. 4-6. In all cases the input $u(k) = \sin \pi k/25$ was used to train the network. The parameters η and T were chosen and the numbers of steps needed to obtain the best approximations were determined.

In Fig. 4(a), $W(z) = z^{-5}$ and the parameters of N are adjusted every instant of time with $\eta = 0.1$. In Fig. 4(b), the parameters are adjusted every five steps with $\eta = 0.2$ using the partial derivative of $\Sigma_{i=k-4}^{k} e^2(i)$ and parameter adjustments were carried out for 50 000 steps. The improvement in performance in Fig. 4(b) is evident.

The simulation results when $W(z) = 0.1z^{-1} + 1.0z^{-2} + 0.5z^{-3}$ are shown in Fig. 5(a) and (b). Training was carried out over 50 000 steps using a step size of $\eta = 0.01$. While from a theoretical standpoint the parameters should be adjusted over every three units of time, the results obtained by adjusting the parameters at every instant were equally satisfactory. This may be attributed to the fact that the pulse response has a duration of only three instants and assumes only positive values.

The simulation results for the case when $W(z) = z + 0.8/z^2 - 0.15z - 0.595$ are shown in Fig. 6. The pulse response, shown in Fig. 6(a), is positive and lasts for approximately 30 instants. The adjustment of the parameters at every instant with a step size of 0.005 did not yield satisfactory response, as shown in Fig. 6(b); $y_d(k)$ (shown in solid line) and $y(k)$ (shown in

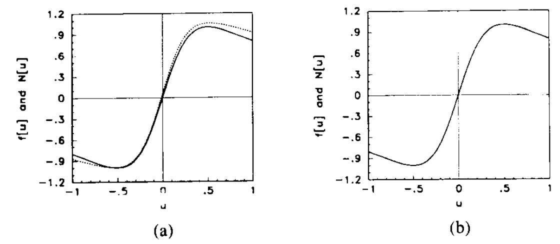


Fig. 4. Representation 1 with $W(z) = z^{-5}$. Estimate of f when parameters are adjusted (a) every instant and (b) every five instants.

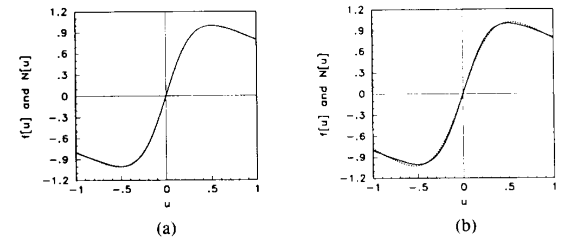


Fig. 5. Plots of $f[u]$ and $N[u]$ when $W(z) = 0.1z^{-1} + 1.0z^{-2} + 0.5z^{-3}$. In (a) parameters are adjusted at every instant while in (b) they are adjusted every three steps.

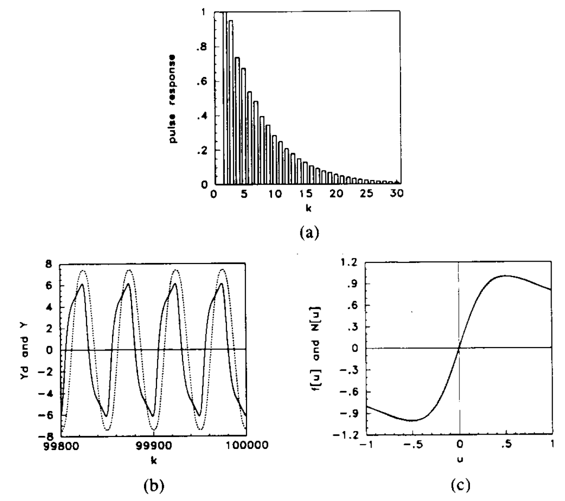


Fig. 6. (a) Pulse response of $W(z) = z + 0.8/z^2 - 0.15z - 0.595$. (b) Plots of y_d and y when parameters are adjusted at every instant. (c) f and its estimate realized by the neural network when parameter adjustments are carried out every five steps.

dotted line) are seen to be quite different. However, when the same step size along with an interval of length $T = 5$ was used, the output of the model matched y_d exactly. The corresponding function $N[u]$ also approximated $f[u]$ arbitrarily closely over the interval $[-1, 1]$, as shown in Fig. 6(c). Equally accurate results could be achieved by adjusting the parameters at every step by reducing the step size to 0.0005. This indicates that increasing the step size has to be accompanied by a larger window size to achieve the desired accuracy.

Comment 1: In each of the above cases the desired partial derivatives $\partial v/\partial\theta_j$ are obtained using static back propagation with a single sensitivity network as shown in Fig. 3, and each signal is processed through a dynamical system with a transfer function $W(z)$. Hence, as many models of $W(z)$ are needed as there are parameters.

It is evident that the extent to which the output of $W(z)$ differs from its input is the extent to which the dynamic gradient will differ from that obtained by static back propagation. In general this will be significant when $W(z)$ has either a large time delay (or relative degree) or an oscillatory pulse response. If the output of $W(z)$ is approximately the same as its input, static back propagation may result in performance almost as good as that obtained by dynamic back propagation. This also applies to all the representations which follow.

Representation 2: $y = N^1[W(z)N^2[u]]$

In this case two neural networks, N^1 and N^2, are connected in cascade as shown in Fig. 7 with a transfer function $W(z)$ between them. The inputs and outputs of the three subsystems are as shown in Fig. 7. The partial derivative of any one of the outputs $y_i (i = 1, 2, \cdots)$ with respect to the weights in the neural network N^1 can be computed directly as in Fig. 1. Hence our main interest is in determining the effect of $W(z)$ in computing the gradient w.r.t. the parameter vector of N^2.

To indicate the role played by $W(z)$, we consider a typical output $y_i(k)$ and determine by dynamic back-propagation the partial derivative $\partial y_i(k)/\partial\theta_j$ where θ_j is a parameter of N^2. Since in Fig. 7,

$$\frac{\partial y_i}{\partial\theta_j} = \sum_l \frac{\partial y_i}{\partial v_l}\frac{\partial v_l}{\partial\theta_j} \tag{9}$$

the desired partial derivative is obtained as the sum of products of functions which can be generated using methods described thus far. In particular, while $\partial y_i/\partial v_l$ can be determined using back-propagation, the term $\partial v_l/\partial\theta_j$ is computed exactly along the lines described in Representation 1. This is indicated in Fig. 7.

From (9), it is seen that the computation of $\partial y_i/\partial\theta_j$ for specified values of i and j involves the functions $\partial y_i/\partial v_l$ and $\partial v_l/\partial\theta_j$ where v_l is a typical component of the output vector v of dimension L. A single sensitivity network $\mathcal{S}_1$ is adequate to determine $\partial y_i/\partial v_l$ for all values of l. However, $\partial v_l/\partial\theta_j$ requires a single sensitivity network of the type given in Fig. 1 for each value of l. Hence the sensitivity network $\mathcal{S}_2$ includes L such networks. If the output vector y is of dimension M, M such combinations of networks will be needed to determine all the required partial derivatives.

Simulation 2: The second set of simulations were carried out to identify a nonlinear dynamical system using the representation described in this section. The unknown plant is assumed to have the representation

$$y_d = f[W(z)g(u)]$$

where $W(z)$ is a known scalar transfer function and $f: \mathbb{R} \to \mathbb{R}$ and $g: \mathbb{R} \to \mathbb{R}$ are unknown continuous nonlinear functions defined on a compact set. The identification model has N^2, $W(z)$, and N^1 in cascade as shown in Fig. 7 with $N^2 \in \mathfrak{N}^3_{1,20,10,1}$ and $N^1 \in \mathfrak{N}^3_{1,30,20,1}$. The neural networks chosen were adequate to approximate the functions f and g respectively.

Even when $W(z) = 1$, the neural networks can only approximate the composition of the two maps f and g are not f and g

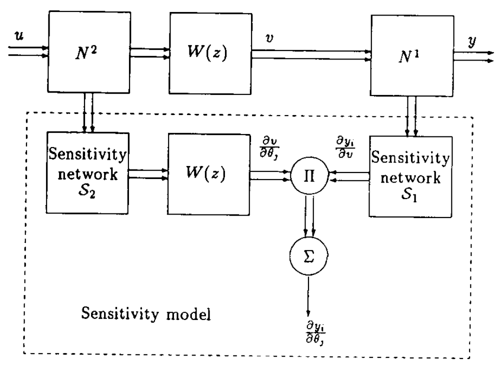

u
N^2
$W(z)$
v
N^1
y
Sensitivity network S_2
$W(z)$
$\frac{\partial v}{\partial \theta_j}$
$\frac{\partial y_i}{\partial v}$
Π
Sensitivity network S_1
Σ
Sensitivity model
$\frac{\partial y_i}{\partial \theta_j}$

Fig. 7. Generation of the gradient in Representation 2.

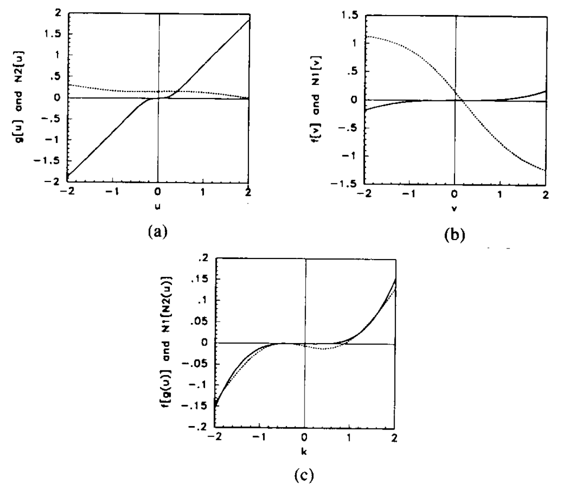


Fig. 8. (a) $g[u]$ and its estimate $N^2[u]$. (b) $f[v]$ and its estimate $N^1[v]$. (c) Plots of $f[g[u]]$ and $N^1[N^2[u]]$.

individually. For example, if $f(v) = [v/3.5]^3$, $g(u) = 4u^3/1 + 4u^2$ and $W(z) = 1$, Fig. 8 shows the plots of $N^2[u]$, $N^1[v]$, and $N^1[N^2(u)]$, where the neural networks are trained with a random input distributed uniformly over the interval $[-2, 2]$. As expected, while $N^1[N^2(u)]$ approximates $f[g(u)]$ quite accurately over the interval $[-2, 2]$, $N^1[v]$ and $N^2[u]$ are substantially different from $f[v]$ and $g[u]$ respectively.

Simulation studies carried out by adjusting the parameters of N^2, while assuming that f is known, resulted in a small output error and accurate identification of g. If, however, f was chosen to be nonmonotonic (e.g. $f(v) = (v - 0.8)v(v + 0.5)$), the map $N^2[u]$ converged to a local optimum and the output error was nonzero.

Simulation studies were next carried out with the same functions $f(v) = [v/3.5]^3$ and $g(u) = 4u^3/1 + 4u^2$ but with $W(z) = z + 0.3/z^2 - 0.8z + 0.15$. When the network was trained with an input $u(k) = \sin \pi k/25$, the output error was small for that input but the estimated model was not satisfactory for other inputs. The results obtained when the networks were trained for 100 000 steps using a random input in the interval $[-2, 2]$ are shown in Fig. 9. While N^1 and N^2 do not approximate f and g respectively, the output error is seen to be small. During the testing phase, the input was chosen to be a sinusoid ($u(k) = \sin \pi k/25$) in the interval $0 \le k < 100$, a constant $+1$ in the interval $100 \le k < 150$, a constant -1 in the interval $150 \le k < 200$, and a sum of two sinusoids ($u(k) = 0.2 \sin \pi k/25 + 0.8 \sin \pi k/10$) in the interval $200 \le k < 300$. It is seen that the output error is small in all cases, indicating that the model is, as expected, satisfactory for general inputs.

Comment 2: The presence of the transfer function $W(z)$ makes dynamic back-propagation essential to adjust the parameters of the network N^2. Simulation studies attempting to simplify the learning process using static back-propagation on a single neural network resulted in very poor identification.

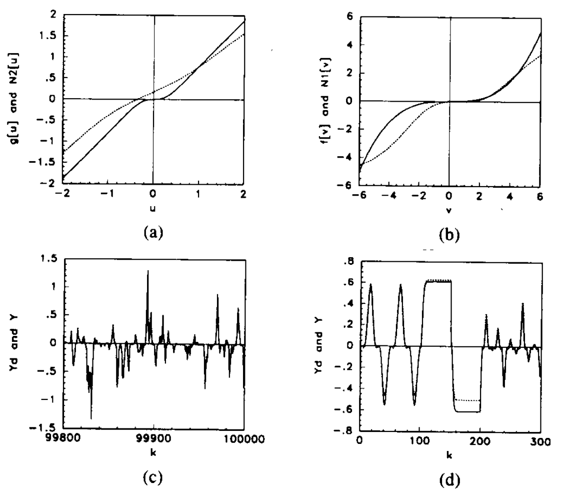


Fig. 9. (a) $g[u]$ and its estimate $N^2[u]$. (b) $f[v]$ and its estimate $N^1[v]$. (c) Outputs $y_d(k)$ and $y(k)$ during training. (d) Outputs $y_d(k)$ and $y(k)$ for the test input.

C. Representation 3: $y = N[u + W(z)y]$

Notation: Before considering Representation 3, we describe briefly a notational difficulty which arises in it which was not encountered in the earlier representations or in static systems. This concerns situations where the performance index involves time functions (e.g. $f[x(k, \theta), \theta]$) which are dependent on parameters $\theta_j (j = 1, 2, \cdots)$ both implicitly in terms of $x(k, \theta)$ and explicitly. In such cases, our interest is in the total derivative of f with respect to θ_j, which we shall denote by $\bar{\partial} f/\bar{\partial}\theta_j$. Hence,

$$\frac{\bar{\partial} f}{\bar{\partial}\theta_j} = \frac{\partial f}{\partial x}\frac{\bar{\partial} x}{\bar{\partial}\theta_j} + \frac{\partial f}{\partial\theta_j}$$

where $\partial f/\partial\theta_j$ is the partial derivative of f w.r.t. θ_j. The notation $\bar{\partial}/\bar{\partial}\theta_j$ is used to denote the total derivative in place of the usual notation $d/d\theta_j$ to emphasize the vector nature of the parameter θ.

Fig. 10 shows a neural network N connected in feedback with a transfer matrix $W(z)$. The input to the nonlinear feedback system is a vector $u(k)$. If θ_j is a typical parameter of the neural network, the aim is to determine the derivatives of $y_i (i = 1, 2, \cdots, M)$ with respect to $\theta_j (j = 1, 2, \cdots)$ for all $k \ge 0$. We observe here for the first time a situation not encountered ear-

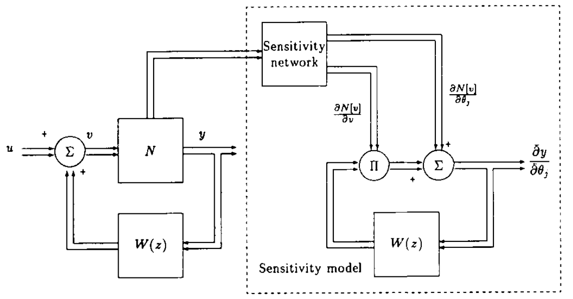

Sensitivity network
$\frac{\partial N[v]}{\partial v}$
$\frac{\partial N[v]}{\partial \theta_j}$
u
v
N
y
W(z)
Π
Σ
$\frac{\partial y}{\partial \theta_j}$
Sensitivity model

Fig. 10. Generation of the gradient in Representation 3.

lier, namely that $\overline{\partial y}_i(k)/\overline{\partial \theta}_j$ is the solution of a difference equation. The equation is computed along the same lines as for a linear system described in [4] and [5]. Since

$$y = N[v] = N[u + W(z)y],$$

$$\frac{\overline{\partial y}}{\overline{\partial \theta}_j} = \frac{\partial N[v]}{\partial v}\frac{\partial v}{\partial \theta_j} + \frac{\partial N[v]}{\partial \theta_j}$$

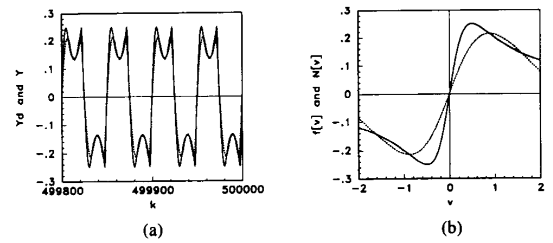
$$= \frac{\partial N[v]}{\partial v} W(z) \frac{\overline{\partial y}}{\overline{\partial \theta}_j} + \frac{\partial N[v]}{\partial \theta_j}. \qquad (10)$$

In (10), $\overline{\partial y}/\overline{\partial \theta}_j$ is a vector and $\partial N[v]/\partial v$ and $\partial N[v]/\partial \theta_j$ are the Jacobian matrix and vector respectively which are evaluated around the nominal trajectory. Hence it represents a linearized difference equation with known time-varying elements in the variables $\overline{\partial y}_i/\overline{\partial \theta}_j$. The dynamical process by which the gradient is generated is shown in Fig. 10. The Jacobian matrix $\partial N[v]/\partial v$ and the vector $\partial N[v]/\partial \theta_j$ can be computed at every instant by using the sensitivity network described earlier.

Comment 3: The amount of computational effort involved in determining the partial derivatives of the output vector y with respect to a typical parameter θ_j is worth emphasizing. The sensitivity network shown in Fig. 10 contains as many static back-propagation networks (Fig. 1) as the dimension of y. Each network yields the partial derivative $\partial y_i/\partial v$ as well as $\partial y_i/\partial \theta_j$ ($i = 1, 2, \cdots; j = 1, 2, \cdots$) simultaneously. Hence this part is common to all the parameters θ_j ($j = 1, 2, \cdots$) for the derivation of the signal $\overline{\partial y}/\overline{\partial \theta}_j$. However, a separate feedback circuit containing $W(z)$, shown in Fig. 10, has to be used for each parameter θ_j. The multiplication denoted by Π represents the multiplication of the output of $W(z)$ by the matrix $\partial N[v]/\partial v$.

Comment 4: If the parameters of the network N are to be adjusted using the dynamic back-propagation method, the computational effort needed is described in the previous comment. Since this is substantial, the designer may wish to use approximate methods which may require considerably less effort. For example, if the feedback structure (Representation 3) is to be used for identification purposes and the output y_p of the unknown plant is accessible, static back-propagation may be used by assuming a series-parallel structure [1]. Alternately, if the term $\partial N[v]/\partial \theta_j$ in (10) is dominant, static back-propagation may perform quite satisfactorily. However, it must be stressed that the precise gradient can be obtained only by using the methods described in this section.

Simulation 3: In this case we have a feedback system with $f[v] = v/1 + 4v^2$ in the forward loop and $W(z) = -1.1z + 0.33/z^2 - 0.8z + 0.15$ in the feedback path. The input to the system is $u(k) = 2 \sin(\pi k/25)$. The aim is to determine a feedback system, with a neural network $N \in \mathfrak{N}^3_{1,20,10,1}$ in the forward path and $W(z)$ in the feedback path, whose output $y(k)$ approximates the output of the unknown system $y_d(k)$ asymptotically.



(a) (b)

Fig. 11. (a) Outputs $y_d(k)$ and $y(k)$. (b) Plots of $f[v]$ and $N[v]$.

If a series-parallel model is used so that

$$y(k) = N[W(z)y_d(k) + u(k)]$$

and static back-propagation is used with a step size of $\eta = 0.1$, the map $N(v)$ is found to approximate f quite accurately. However, when the parameters of N are adjusted within the feedback system using the method described in this section with a step size of $\eta = 0.01$ and the adjustments carried out every ten steps for 500 000 time steps, the results are shown in Fig. 11. While N does not approximate f satisfactorily over the entire interval $[-2, 2]$, the output error $y(k) - y_d(k)$ shown in Fig. 11(a) is found to be small.

D. Representation 4: $y = N^1[N^2[u] + W(z)y]$

The feedback system described in the previous representation is preceded in this case by a neural network N^2. The presence of N^2 does not affect the computation of the partial derivatives of the outputs with respect to the parameters of N^1; hence the same procedure as in Representation 3 can be adopted. However, to compute the partial derivative with respect to the parameters of the network N^2, the following procedure is used. Since

$$y = N^1[v] = N^1[N^2[u] + W(z)y]$$

we have

$$\frac{\partial y}{\partial \theta_j} = \frac{\partial N^1[v]}{\partial v}\frac{\partial v}{\partial \theta_j}$$

$$= \frac{\partial N^1[v]}{\partial v}\left[\frac{\partial N^2[u]}{\partial \theta_j} + W(z)\frac{\partial y}{\partial \theta_j}\right].$$

As shown in Fig. 12, this can be expressed as the input $\partial N^2[u]/\partial \theta_j$ to a nonlinear system obtained using Representation 3. Once again, the sensitivity network is used to compute the Jacobian matrix $\partial N^1[v]/\partial v$ and the vector $\partial N^2[u]/\partial \theta_j$ at every instant.

Comment 5: Determination of partial derivatives involves linearization around the nominal trajectory (i.e., around the operating point). When dynamical systems are involved, this merely involves the computation of the Jacobian matrix and the input vector around the nominal trajectory, and the four representations indicate how this can be realized for the four structures chosen.

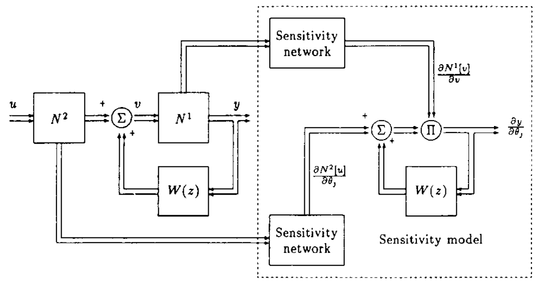

Sensitivity network
$\frac{\partial N^1[v]}{\partial v}$
u
N^2
v
N^1
y
$W(z)$
$\frac{\partial N^2[u]}{\partial \theta_j}$
$\frac{\partial y}{\partial \theta_j}$
Sensitivity network
Sensitivity model

Fig. 12. Generation of the gradient in Representation 4.

Simulation 4: Consider a plant described by the difference equation

$$f[W(z)y_d(k) + g[u(k)]] = y_d(k) \qquad (11)$$

where $f(v) = 1 - e^{-3v}/1 + e^{-3v}$, $g[u] = 4u^3$, and $W(z) = -0.4z + 0.2/z^2 - 0.8z + 0.15$. The functions f and g are assumed to be unknown while $W(z)$ is known. A structure identical to that of (11) is set up to approximate the input-output mapping of the plant with f and g replaced by neural networks $N^1 \in \mathfrak{N}^3_{1,20,10,1}$ and $N^2 \in \mathfrak{N}^3_{1,20,10,1}$ respectively. The results of adjusting the parameters of N^1 and N^2 for 500 000 steps using dynamic back-propagation are shown in Fig. 13. As in Representation 2, while the maps N^1 and N^2 bear no resemblance to f and g respectively, the output error is found to be very small (Fig. 13(a)). The networks are trained using an input $u(k) = \sin(\pi k/25)$ and the parameters are adjusted at every instant using a step size of $\eta = 0.01$. N^1 and N^2 as well as the functions f and g are also shown in Fig. 13(b) and (c) respectively.

Practical Considerations: Some relevant information concerning the simulations presented in this section is included below.

(i) In all the simulations carried out, the size of the network (i.e., number of layers, number of nodes in each layer) was chosen by trial and error to approximate the nonlinear functions that were present in the difference equations. Networks belonging to the class $\mathfrak{N}^3_{1,20,10,1}$ were found to be satisfactory in most cases. The size of the network was increased in Simulation 2 ($N^1 \in \mathfrak{N}^3_{1,30,20,1}$) when the approximation resulting from the smaller network was not satisfactory.

(ii) The inputs used to train the networks were those which were found to be satisfactory to identify the static nonlinear functions over the range of interest. For instance, a single sinusoid (e.g. $u(k) = \sin(\pi k/25)$) was adequate in most cases to approximate the functions (though the parameters do not converge to the same values for different initial conditions). In those cases in which this was not adequate, a random input with a uniform distribution over the same range was used.

(iii) The difficulties encountered in identifying the nonlinear maps f and g in Representations 2 and 4 are inherent in the problem and do not depend upon the inputs used. Hence persistently exciting inputs do not yield better results in such cases.

(iv) The results presented were not obtained by fine-tuning the response; rather they are typical of those obtained using the methods suggested in the paper. Interested readers should therefore be able to duplicate the simulation results presented here without much difficulty.

(v) The step size η and the window size T are the two important parameters in the simulations. Generally T was chosen to be 1 with a corresponding small value of η (typically 0.01.) In some experiments, increasing the step size to improve the rate of convergence resulted in an unstable control system. As a rule of thumb, an increase in step size was usually accompanied by an increase in T.

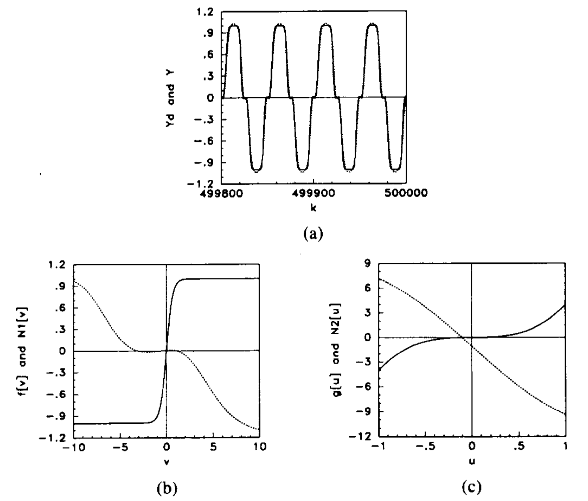


Fig. 13. (a) Outputs $y_d(k)$. (b) Plots of $f[v]$ and $N^1[v]$. (c) Plots of $g[u]$ and $N^2[u]$.

V. Control of Simple Nonlinear Systems Using Dynamic Back-Propagation

In this section we apply the methods developed in the previous sections to two control problems. The plants to be controlled are nonlinear and are described by first-order difference equations of the form $y_p(k+1) = f[y_p(k)] + g[u(k)]$, where f and g are continuous functions. In spite of the seeming simplicity of such systems, the decisions that have to be made are typical of those encountered in more complex cases. The first problem is the same as that considered in [1, example 11]. However, while the controller in [1] was based on obtaining an estimate of the inverse of g, dynamic back-propagation is used in the present case. In the second problem, g does not have an inverse and dynamic back-propagation is used to obtain the optimal parameters of the controller. In both problems, we shall assume that the designer has adequate information about the functions f and g so that an identification model of the form $\hat{y}_p(k+1) = N_f[\hat{y}_p(k)] + N_g[u(k)]$ can be set up to approximate the plant sufficiently accurately.

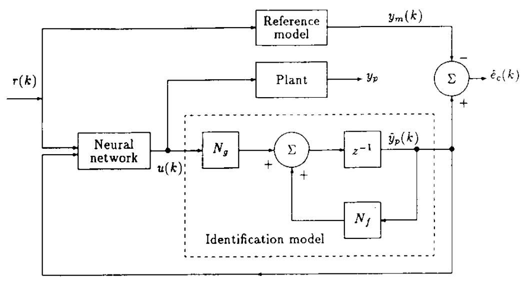
Problem 1

In this case the plant is described by the difference equation

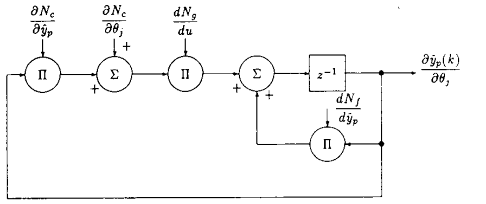
$$y_p(k+1) = \frac{y_p(k)}{1 + y_p^2(k)} + u^3(k)$$

and a reference model is described by the difference equation

$$y_m(k+1) = 0.6y_m(k) + r(k)$$



(a)


$\frac{\partial N_c}{\partial \hat{y}_p}$
$\frac{\partial N_c}{\partial \theta_j}$
$\frac{dN_g}{du}$
$\frac{\partial \hat{y}_p(k)}{\partial \theta_j}$
$\frac{dN_f}{d\hat{y}_p}$
z^{-1}

(b)

Fig. 14. (a) Structure of the controller. (b) The gradient generating circuit.

where $r(k) = \sin(2\pi k/25) + \sin(2\pi k/10)$. The objective is to determine an input $u(k)$ to the plant such that $\lim_{k\to\infty} |y_p(k) - y_m(k)| < \epsilon$, where ϵ is a suitably chosen constant. In [1], f, g, and g^{-1} were approximated by multilayer neural networks and the control input was determined as $u(k) = \widehat{g^{-1}}[-\hat{f}[y_p(k)] + 0.6y_p(k) + r(k)]$. In contrast to that, we assume in the present case that a multilayer neural network N_c is used as the controller to generate the control input. The parameters of N_c are then adjusted to minimize the error between the output of the reference model and that of the identification model. Since the identification model (which is a dynamical system) lies between N_c and the measured output error $\hat{e}_c(k) = \hat{y}_p(k) - y_m(k)$, dynamic back-propagation has to be used. Further, since the plant is of first order, the control input is generated as $u(k) = N_c[\hat{y}_p(k), r(k)]$, with $\hat{y}_p(k)$ and $r(k)$ as the inputs to the network N_c. The structure of the overall feedback loop as well as the gradient generating circuit is shown in parts (a) and (b) of Fig. 14 respectively.

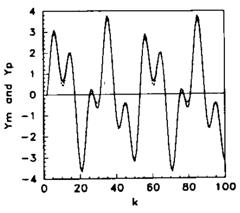
In the simulation studies $N_c \in \mathfrak{N}^3_{2,20,10,1}$ and contained 281 parameters. Adjustments were made at every step using a step size $\eta = 0.001$ and the gradient procedure was continued for 100 000 time steps. The outputs of the reference model and the plant obtained by using the method in [1] are shown in Fig. 15(a), while the same signals obtained by dynamic back-propagation are shown in Fig. 15(b). The two outputs are seen to approximate the output of the reference model quite accurately. Hence in the simple case where $g(\cdot)$ has an inverse, the two methods are seen to be equally effective but the dynamic back-propagation method bypasses the need to determine an inverse operator.

Problem 2

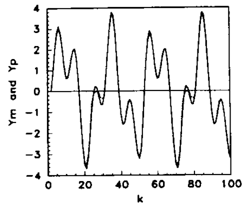
In this case the plant is described by the difference equation

$$y_p(k+1) = \frac{y_p(k)}{1 + y_p^2(k)} + (u(k) + 1.0)u(k)(u(k) - 1.0)$$

and the reference model is the same as in Problem 1. Since $g[u] = (u + 1.0)u(u - 1.0)$ does not have an inverse, the method outlined in [1] cannot be used. However, using the same structure for the controller as shown in Fig. 14(a), the dynamic back-propagation method can be used to determine the control input which minimizes the output error in some sense. The output of the plant (dotted lines) as well as that of the reference model (solid lines) are shown in parts (a), (b), and (c) of Fig. 16 for a reference input $r(k) = \alpha[\sin(2\pi k/25) + \sin(2\pi k/10)]$ with $\alpha = 1$, 0.5, and 0.25 respectively. The output of the plant is seen to follow the output of the reference model with increasing accuracy as the value of α is decreased.

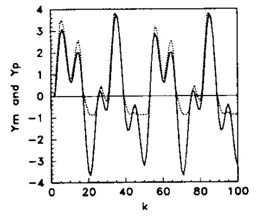


(a)

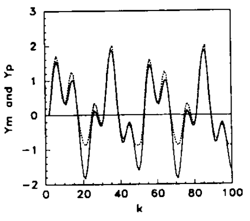


(b)

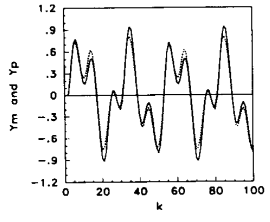
Fig. 15. (a) Outputs of the reference model (solid lines) and the plant (dotted lines) when the controller is implemented using the inverse of $\hat{g}$. (b) y_m and y_p when the controller is implemented using dynamic back-propagation.



(a)



(b)



(c)

Fig. 16. Outputs of the reference model (solid lines) and plant (dotted lines): (a) $\alpha = 1.0$, (b) $\alpha = 0.5$, (c) $\alpha = 0.25$.

VI. Conclusion

This paper deals with a prescriptive method for the optimal adjustment of the parameters of a neural network which is part of a complex dynamical system. The method can be regarded as an extension of the static back-propagation method to dynamical systems and hence is termed dynamic back-propagation. It is based on the fact that gradient methods used in linear dynamical systems in the 1960's can be combined with back-propagation methods for neural networks to obtain the gradient of a performance index of nonlinear dynamical systems. The method can be applied to any complex system which can be expressed as the interconnection of linear dynamical systems and multilayer neural networks.

The paper has been written in a tutorial style to enable the reader to follow all the details of the proposed method. Four simple representations introduced in [1], which can be regarded as building blocks for complex dynamical systems, are considered and in each case the method of determining the gradient is discussed in detail. Simulations are provided at the end of each representation and the method is also applied to two simple control problems in the last section.

From the discussions in Sections IV and V it is readily apparent that the use of the dynamic back-propagation method, even for relatively low order dynamical systems, involves considerable computational effort. The complexity of the method has naturally prompted the search for alternative schemes which are easier to implement. In some cases (e.g. system identification), this has led to models which require only static back-propagation. In other cases, the terms due to static back-propagation which occur in the difference equations describing the desired partial derivatives may be dominant so that static back-propagation may suffice (see Comment 4). In all cases, the step size of the gradient method has to be kept small to ensure the stability of the closed loop. The latter in turn has motivated efforts, following conventional adaptive control theory, to develop simple control algorithms based on stability theory. The robustness of biological systems, coupled with the fact that it may be unrealistic to expect such systems to use back-propagation, makes stability methods particularly appealing. However, in view of the nonlinear nature of the overall system, generating stable adaptive algorithms has proved to be a formidable task. Until such schemes are developed, dynamic back-propagation must be considered one of the general methods available to the designer for the adjustment of parameters of neural networks in complex dynamical systems.

Acknowledgment

The authors would like to thank the reviewers and the associate editor for their careful reading of the paper and their helpful comments.

References

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

[2] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270-280, 1989.

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

[4] K. S. Narendra and L. E. McBride, Jr., "Multiparameter self-optimizing system using correlation techniques," *IEEE Trans. Automat. Contr.*, vol. AC-9, pp. 31-38, 1964.

[5] P. V. Kokotovic, "Method of sensitivity points in the investigation and optimization of linear control systems," *Automat. Remote Contr.*, vol. 25, pp. 1512-1518, 1964.

[6] L. E. McBride, Jr., and K. S. Narendra, "Optimization of time-varying systems," *IEEE Trans.* Automat. Contr., vol. AC-10, no. 3, pp. 289-294, 1965.

[7] J. B. Cruz, Jr., Ed. *System Sensitivity Analysis* (Benchmark papers in Electrical Engineering and Computer Science). Stroudsburg, PA: Dowden, Hutchinson and Ross, 1973.

[8] K. S. Narendra, M. A. L. Thathachar, and T. Baker, "Adaptive control using correlation techniques," in *Proc. Allerton Conf.* (Urbana-Champaign, IL), 1965.

[9] J. B. Cruz, Jr., and W. R. Perkins, "A new approach to the sensitivity problem in multivariable feedback system design," *IEEE Trans. Automat. Contr.*, vol. AC-9, pp. 216-223, 1964.

Kumpati S. Narendra (S'55-M'60-SM'63-F'79) received the Ph.D. degree from Harvard University in 1959. He is currently a Professor in the Electrical Engineering Department at Yale University, New Haven, CT, and the Director of the Center for Systems Science there.

He is the author of numerous technical publications in the area of systems theory and the author of the books *Frequency Domain Criteria for Absolute Stability* (coauthor J. H. Taylor) published by Academic Press, *Stable Adaptive Systems* (coauthor A. M. Annaswamy) and *Learning Automata—An Introduction* (coauthor M. A. L. Thathachar) published by Prentice Hall. He is also the editor of three books and currently he is editing a reprint volume entitled *Recent Advances in Adaptive Control* (coeditors R. Ortega and P. Dorato), which will be published by the IEEE Press. His research interests are in the areas of stability theory, adaptive control, learning automata, and the control of complex systems using neural networks.

Dr. Narendra is a member of Sigma Xi and the American Mathematical Society, and a Fellow of the American Association for the Advancement of Science and the IEE (U.K.). He was the recipient of the 1972 Franklin V. Taylor Award of the IEEE Systems, Man and Cybernetics Society, the George S. Axelby best paper award of the IEEE Control Systems Society in 1988, and the Education Award of the American Automatic Control Council in 1990.

Kannan Parthasarathy (S'87) was born in India on August 31, 1964. He received the B. Tech. degree in electrical engineering from the Indian Institute of Technology, Madras, in 1986 and the M.S. and M.Phil. degrees in electrical engineering from Yale University, New Haven, CT, in 1987 and 1988 respectively. At present he is a doctoral candidate at Yale. His research interests include adaptive and learning systems, neural networks and their application to adaptive control problems.

Mr. Parthasarathy is a student member of the American Mathematical Society. He is the recipient of the Connecticut High Technology Graduate Scholarship for the period 1989-1991.