Dynamic Process Modeling with Recurrent Neural Networks

Yong You and Michael Nikolaou
Dept. of Chemical Engineering, Texas A&M University, College Station, TX 77843

A method of nonlinear static and dynamic process modeling via recurrent neural networks (RNNs) is studied. An RNN model is a set of coupled nonlinear ordinary differential equations in continuous time domain with nonlinear dynamic node characteristics as well as both feedforward and feedback connections. For such networks, each physical input to a system corresponds to exactly one input to the network. The system's dynamics are captured by the internal structure of the network. The structure of RNN models may be more natural and attractive than that of feedforward neural network models, but computation time for training is longer. Our simulation results show that RNNs can learn both steady-state relationships and process dynamics of continuous and batch, single-input/single-output and multi-input/multioutput systems in a simple and direct manner. Training of RNNs shows only small degradation in the presence of noise in the training data. Thus, RNNs constitute a feasible alternative to layered feedforward backpropagation neural networks in steady-state and dynamic process modeling and model-based control.

Introduction

Mathematical models play an important role in control system synthesis. However, due to the inherent nonlinearity, complexity and uncertainty of chemical processes, it is usually difficult to obtain an accurate model for a chemical engineering system.

In recent years, artificial neural networks (ANNs) have been used, among others, for process fault diagnosis, modeling, and control (Hoskins and Himmelblau, 1988; Ungar et al., 1990; Bhat and McAvoy, 1990). ANNs are computational paradigms based on the neurophysiology of biological neural systems such as the human brain. They can be used to capture certain input-output relationships. Major advantages of ANNs include their learning ability, parallel processing ability, and noise resistance ability.

A variety of ANN paradigms have been proposed and used in recent years (Rumelhart et al., 1986b; Simpson, 1989). However, most of the currently used ANNs for process modeling are layered feedforward neural networks (FNNs), also called multilayer perceptrons with backpropagation learning algorithms (Rumelhart et al., 1986a).

In an FNN, basic processing elements (also called nodes, neurons) are grouped into several layers: input layer, output layer, and hidden layer(s). Nodes in the input layer are merely used to scale inputs into the range of zero and one and then pass the scaled values to the hidden layer. Each node scales one input. Nodes in a hidden or an output layer receive signals from all elements in the previous layer. The output layer gives the final output values calculated through the network. Figure 1 shows the topology of a typical three-layer FNN.

The mathematical description for a node in a hidden or output layer is characterized by:

$$y_i^{l+1} = f\left(\sum_{j=1}^{N_l} w_{ij}^{l+1} y_j^l + \theta_i^{l+1}\right) \tag{1}$$

where y_i^{l+1} is the activity or state of the ith neuron in layer $(l+1)$; N_l is the number of nodes in layer l; w_{ij}^{l+1} is the connection strength or weight from the jth neuron in layer l; θ_i^{l+1} is a threshold value for the node; f is a nonlinear mapping function (also called transfer function or squashing function), typically a sigmoidal function of the form $f(x) = (1+e^{-x})^{-1}$.

Use of feedforward neural networks for dynamic modeling has basically relied on a nonlinear autoregressive moving average (ARMA) approach. This method requires as inputs to the network a number of past values for each physical input and output of a system, which greatly increase dimensionality

Correspondence concerning this article should be addressed to M. Nikolaou.

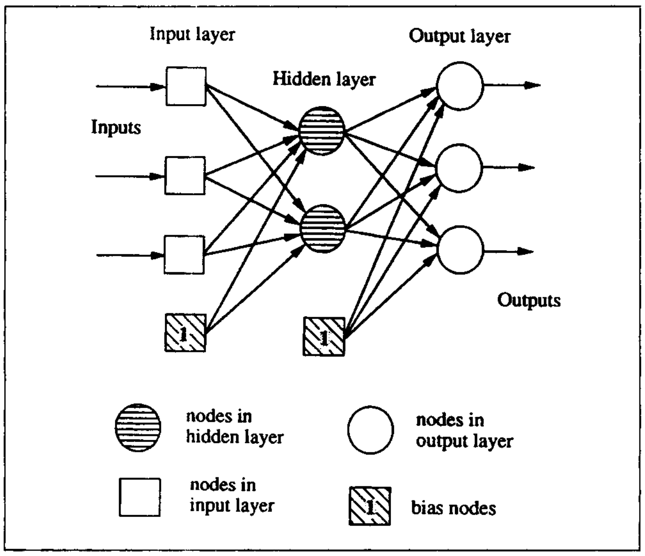


Figure 1. Topology of a three-layer FNN.

when multivariable systems are involved. The number of past input and output values needed for an optimal FNN model has to be determined by trial-and-error. FNN models are in the discrete time domain with a fixed sampling period.

In this article, a method of static and dynamic process modeling via recurrent neural networks (RNNs) is discussed. In a recurrent artificial neural network, basic processing units are connected arbitrarily so that there are feedforward and feedback paths. Nodes in an RNN are generally classified into three categories (instead of layers): input, output, and hidden nodes. Input nodes receive external input signals, and output nodes send off output signals calculated through the network. Hidden nodes neither receive external input signals nor send output signals, but rather exchange internal signals with other nodes. Processing units in an RNN are usually fully connected: they receive output signals from all nodes including themselves.

RNNs have two salient features that distinguish them from feedforward networks. One is their node characteristics, which involve nonlinear dynamic functions (ordinary differential equations, ODEs). A general form of the node characteristics used in this work, which will be discussed in more detail in later sections, is given by:

$$T_i \frac{dy_i}{dt} = -y_i + f_i(x_i) + I_i \tag{2}$$

where y_i is the activity or state of the ith neuron;

$$x_i = \sum_j w_{ij} y_j \tag{3}$$

is the weighted input of the node; w_{ij} is the connection strength or weight from the jth neuron; I_i is an external input. Figure 2 gives the structure of a neuron with characteristic Eq. 2. Note that the node mapping function of a neuron used in an FNN retains only the first two blocks in that figure with fewer incoming signals.

The other major distinction of RNNs is topology (Figure 3). In RNNs there are both feedforward and feedback connections, while in FNNs only feedforward paths exist. In this aspect, RNNs are more similar to biological neural systems.

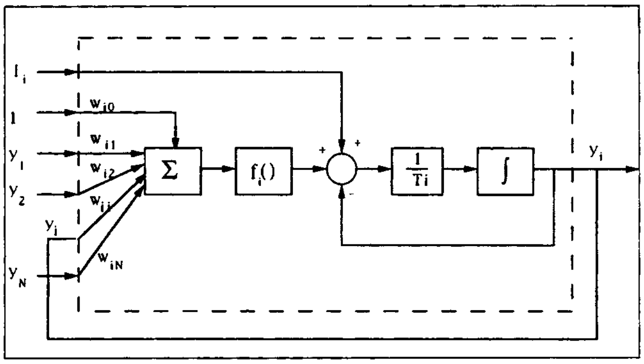


Figure 2. Typical processing element in a RNN.

Compared with FNN models, RNNs are characterized by sets of coupled nonlinear ODEs. RNNs have internal feedback paths that capture a system's dynamics, while there is a one-one correspondence between a system's physical inputs and RNN inputs. No past input or output values are needed. RNN models are in continuous time domain and can be discretized easily with different sampling periods. The model structure of RNNs is more attractive than that of FNNs. Comparisons of RNNs and FNNs in this work show that they have comparable modeling capabilities, but training of RNNs takes more computation time.

Simulation results show that RNNs can learn steady-state relationships as well as process dynamics for both continuous and batch, single-input/single-output (SISO) and multiinput/

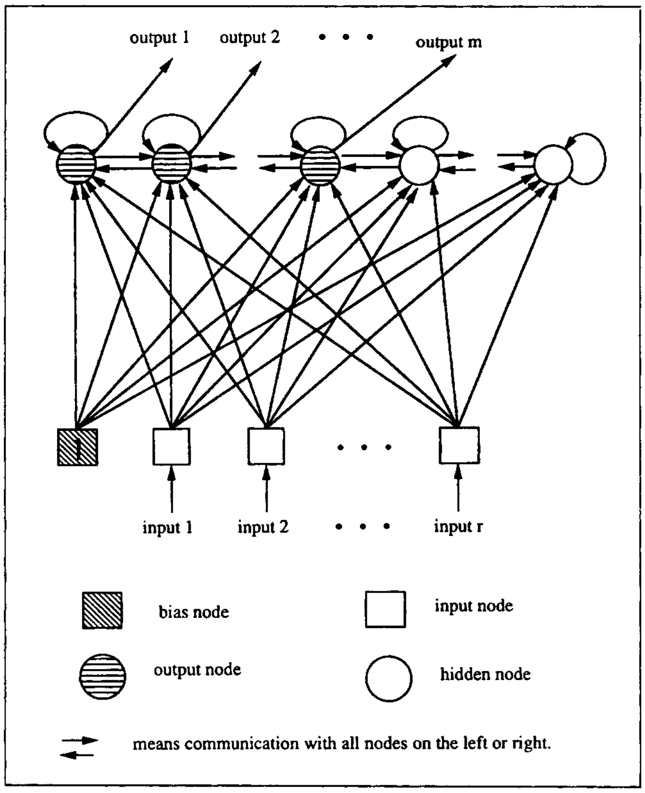


Figure 3. RNN topology used in simulations.

multioutput (MIMO) processes. Simulation results also show that the training of RNNs shows only small degradation in the presence of noise in training data. Thus, RNNs constitute an attractive alternative to layered feedforward backpropagation neural networks in dynamic process modeling and model-based control.

In the following sections, more details of RNNs will be introduced first, followed by simulation results of static and dynamic learning of RNNs on SISO or MIMO, continuous or batch systems. The effect of noise in training data will then be examined. Comparisons of RNNs with FNNs and locally linearized system models are also given. In the last section, discussion and conclusions are presented.

Recurrent Neural Networks

In this section, more details of recurrent neural networks will be given, including the general description of RNNs, the learning algorithms for RNNs, and stability of RNNs. The topology, node characteristics, and learning laws will also be specified for the RNNs used in the simulations in this work.

Network description

A recurrent neural network is characterized by a set of coupled nonlinear ordinary differential equations (ODEs). For a recurrent neural network with N nodes, the dynamics of the ith neuron can be described by the following general equation:

$$\frac{dy_i}{dt}=g_i(y_i,\ x_i,\ I_i) \tag{4}$$

where y_i is the activity or state of the ith neuron (node 1 is used as a bias node, $y_1 = 1$);

$$x_i=\sum_{j=1}^{N} w_{ij}y_j \tag{5}$$

is the weighted input of the node; w_{ij} is the connection strength or weight from the jth neuron (w_{i1} is a bias term); I_i is an external input ($I_i = 0$ when the ith node is not an input node); and g_i is an arbitrary nonlinear mapping function.

A particular form of Eq. 4, used in the 1960s (Grossberg, 1988) and studied by Cohen and Grossberg (1983) and Hopfield (1984), is:

$$T_i\frac{dy_i}{dt}=-y_i+\sum_j w_{ij}f_j(y_j)+I_i \tag{6}$$

where T_i is the time constant of the node; f_i is a differentiable mapping function, such as a sigmoidal function. An alternative specific description of the node characteristics of RNNs, which may be traced back to McCulloch and Pitts (1943) and was studied by Amari (1972) and Hopfield (1982), is:

$$T_i\frac{dy_i}{dt}=-y_i+f_i(x_i)+I_i \tag{7}$$

Equation 7 can be obtained from Eq. 6 by a simple linear transformation when the time constants in both equations are unit (see Appendix A). Figure 2 gives the structure of a neuron with characteristic Eq. 7. Note that the node mapping function of a typical neuron used in a feedforward backpropagation network retains only the first two blocks in this figure.

Learning algorithms for RNNs

Learning or training of an RNN is the process to determine the numerical values of the connection weights and time constants such that the network can have desired input-output relationship. Every recurrent network can be represented by an equivalent feedforward network in discrete time form. Each layer in the feedforward network corresponds to a time step in the RNN. Thus, the backpropagation learning algorithm for feedforward networks can be applied to the training of RNNs using the corresponding feedforward description (Rumelhart et al., 1986a; Nowlan, 1988). This method, however, needs to store the history of all output states and maintain all corresponding sets of weights between every two layers as identical. Thus, this method is difficult to use with complex RNNs trained over a long time period. To overcome that obstacle, several researchers have proposed specific learning algorithms for recurrent neural networks (Almeida, 1987, 1989; Piñeda, 1987; Pearlmutter, 1989; Williams and Zipser, 1989; Gherrity, 1989; Sato, 1990a,b).

Piñeda (1987) used the class of RNNs described by Eq. 7 and the error objective function:

$$E=\frac{1}{2}\sum_{i=1}^{N} J_i^2 \tag{8}$$

where $J_i=\tau_i-y_i^s$ when the ith node is an output node (otherwise, $J_i=0$); τ_i is the target output value; y_i^s is the steady-state output (fixed point) of the node, supposed to exist. The learning algorithm teaches to the RNN the equilibrium points of Eq. 7. It is an extension of the standard backpropagation algorithm and is derived by the gradient descent method according to the rule:

$$\frac{dw_{ij}}{dt}=-\eta_w\frac{\partial E}{\partial w_{ij}} \tag{9}$$

where η_w is the learning rate for weights and it is a positive number. Direct substitution of the derivatives in Eq. 9, based on Eqs. 7 and 8, cannot result in a local learning method (that is, using only information from the node); therefore, the associated dynamic system,

$$\frac{du_i}{dt}=-u_i+f_i'(x_i^s)\left(\sum_j w_{ji}u_j\right)-J_i,\quad i=1\ \ldots\ N \tag{10}$$

is introduced, where $x_i^s=\Sigma_j w_{ij}y_j^s$. This, together with the obtained learning equation,

$$\frac{dw_{ij}}{dt}=\eta_w u_i^s y_j^s \tag{11}$$

constitutes the algorithm that learns the steady states of Eq. 7, where u_i^s is the steady-state point of Eq. 10. Note that the node output values (y_i's) used in this equation are also steady-state values (of Eq. 7). Also note that the time constants T_i are not evaluated, since the lefthand side of Eq. 7 is assumed to be zero during the learning process.

Pearlmutter (1989), using the concept of ordered derivative (Werbos, 1988), developed a learning algorithm for the class of RNNs defined by Eq. 7 based on the minimization of:

$$E=\frac{1}{2}\sum_{i=1}^{N}\int_{t_0}^{t_f} J_i^2(t)dt \tag{12}$$

where $J_i(t)=\tau_i(t)-y_i(t)$ when the ith node belongs to the output space (zero otherwise); $J_i(t)$, $\tau_i(t)$, and $y_i(t)$ are continuous functions of time; t_0 and t_f are the integration time limits, that is, the time limits of the training data. Obviously, t_f-t_0 should be large enough to capture the dynamics of a system. Using Eq. 9 along with learning laws:

$$\Delta T_i \propto -\frac{\partial E}{\partial T_i},\quad \Delta w_{ij} \propto -\frac{\partial E}{\partial w_{i,j}} \tag{13}$$

for the weights and time constants, an RNN can be trained to simulate a desired trajectory.

Derivation of Eq. 13 based on Eq. 12 results in the following learning laws, at epoch p:

$$\Delta w_{ij}(p) = -\eta_w \frac{1}{T_i}\int_{t_0}^{t_f} f_i'(x_i) z_i y_j dt \tag{14}$$

$$\Delta T_i(p) = \eta_T \frac{1}{T_i}\int_{t_0}^{t_f} z_i \frac{dy_i}{dt} dt \tag{15}$$

where η_T is the learning rate for time constants; z_i is defined by the equation:

$$\frac{dz_i}{dt}=\frac{1}{T_i}z_i-\sum_j \frac{1}{T_j} f_j'(x_j) w_{ji} z_j + J_i,\quad i=1\ldots N \tag{16}$$

with boundary conditions $z_i(t_f) = 0$, for $i = 1 \ldots N$.

Almeida's learning algorithm (1987, 1989) is developed for static training. Gherrity (1989) derived a learning algorithm for the RNNs of the general form defined by Eq. 4. Sato's learning algorithms (1990a,b) are derived based on the Lagrange multiplier for the network described by Eq. 6. Williams and Zipser (1989) developed an on-line learning algorithm.

Stability

Since an RNN is described by a set of nonlinear ODEs, stability of a given RNN with certain weights and time constants may, in principle, arise. We denote this stability as RNN stability. The stability of the modified backpropagation algorithm, which produces the values of RNN parameters, is referred to as BP stability. BP stability problems can usually be circumvented, as for the feedforward networks, by experimenting with appropriate values of learning parameters, that is, the learning rates and momentum factors for weights and time constants, η_w, η_T, α_w, α_T. For example, smaller learning rates will facilitate a BP stability problem.

With regard to RNN stability, when $W=\{w_{ij}\}$ and is lower triangular, the recurrent networks represented by Eq. 6 or 7 are equivalent to feedforward networks; thus, they are stable. If W is symmetric, the networks are also stable when monotonically nondecreasing f_i, such as a sigmoidal function, are used (Cohen and Grossberg, 1983; Hopfield, 1984). In other cases, stability of an RNN is not guaranteed, although local RNN stability can be examined by eigenvalue analysis (Almeida, 1989; Simard et al., 1989), that is, locally linearize the nonlinear RNN model around a set of steady-state values to check the stability of the model at this steady state (using linear systems theory). RNN stability was not a problem in our simulations. Similar experience has also been reported by other investigators (Piñeda, 1987; Almeida, 1989; Simard et al., 1989; Hopfield, 1991).

RNN topology used in the simulations

The topology of the RNNs used in the following simulations is determined by the number of input nodes, hidden nodes, and output nodes. The number of input nodes depends on that of inputs of the system to be modeled. Input nodes only receive scaled input signals. All hidden nodes are fully connected. The output nodes, their number being equal to that of outputs of the system, are also fully connected. Additionally, a bias node with constant unit output is used to provide a bias term for each of the other nodes except for the input nodes. A typical RNN with r inputs and m outputs is shown in Figure 3.

The node characteristics used in this work are described by Eq. 7. The differentiable transfer function f_i is the sigmoidal function, $f_i(x_i)=(1+e^{-x_i})^{-1}$.

The training algorithm used for static training is described by Eqs. 7, 10 and 11. For dynamic training Eqs. 7, 14, 15 and 16 are used. To implement the learning algorithms, we use first-order difference equations to approximate the corresponding differential equations. This approximation results in a set of difference equations which need to be solved during learning. For example, Eq. 7 can be written as:

$$y_i(t+\Delta t)=\left(1-\frac{\Delta t}{T_i}\right)y_i(t)+\frac{\Delta t}{T_i}f_i[x_i(t)]+\frac{\Delta t}{T_i}I_i(t) \tag{17}$$

where Δt is the time step. More details will be given in the following sections.

Static Learning of RNNs

Fixed point learning of RNNs is examined using difference approximation of the learning algorithm defined by Eqs. 7, 10 and 11, with $\Delta t=1$. (All dynamic learning algorithms can be used for static learning. For simplicity, we use the algorithm developed specifically for steady-state learning.) To accelerate the convergence of the learning process and smooth the changes of weights and time constants, we added the momentum term $\alpha_w \Delta w_{ij}(p-1)$ to the difference form of Eq. 11, where $0<\alpha_w<1$ is the momentum factor for weights. Note that the y_i's and u_i's used in Eq. 11 are all steady-state values.

During the learning process, the fixed points of the network are calculated first by Eq. 7. Then, the steady-state values of u_i are determined by Eq. 10 using the fixed points of the nodes. The difference form of Eq. 11 is then used to adjust the weights of the network. Initial values of W are set as random numbers with uniform distribution in [− 1, 1]. The initial neuron states, y_i's, are selected as the initial external input values plus $f_i(0)$ $(=0.5)$.

The selection of the number of hidden nodes in our simulations was conducted by trial-and-error. In general, more hidden nodes result in better fitting of the training data. This, however, requires more computational effort, and overfitting of training data may deteriorate a model's predictive ability. Thus, tradeoffs are needed among the goodness of data fitting, predictive ability, and computational effort.

In our simulations, the number of hidden nodes were determined as follows:

1. Start with zero hidden node.
2. Train the network, that is, determine values for connection weights and time constants to minimize the objective function (adjustment of learning parameters may be necessary).
3. Check the "optimal" fitting of the RNN outputs (obtained in step 2) according to the coefficient of determination. Test the RNN model with novel data (not included in the training), by using the coefficient of determination. If the result is not acceptable, adjust the number of hidden nodes and go to step 2, that is, increase the number of hidden nodes until little improvement can be achieved.
4. Stop.

To evade instability problems during training in step 2 of the above procedure, we preset a maximum number of epochs (presentations of training examples to the network) and use different values for the learning parameters to train the network. During training (step 2), the best values for connection weights and time constants are determined by comparing the sum of square errors (SSE) between the desired outputs and the RNN outputs at each epoch. Best values are obtained either at the last epoch (when training converges) or at a previous epoch (when training diverges after going through a minimum).

All simulation programs were written in FORTRAN and ran on a SPARC station 1.

SISO system at steady state

The process employed to study the static modeling ability of RNNs is the pH CSTR system (Figure 4) used by Bhat and McAvoy (1990) (see Appendix B).

The training data used here are the steady-state relationships of F_2 and pH, which can be obtained by solving the steady-state form of Eqs. B1 and B2 in Appendix B. A set of 20 input-output pairs are used in the training database.

An RNN with two fully connected hidden nodes, one bias node, one input node, and one fully connected output node is used in learning the steady-state relationship described above. The weights of the network were obtained within 2,000 epochs with $\eta_w=0.1$ and $\alpha_w=0.8$. The RNN output for this static relationship, along with the desired output, is given in Figure 5.

To measure the goodness of the fitting, the coefficient of determination, denoted by R^2, is used, which, for the ith output, is defined by (Milton and Arnold, 1990):

$$R_i^2=(1-SSE_i/S_{TT_i})\times 100\% \tag{18}$$

where

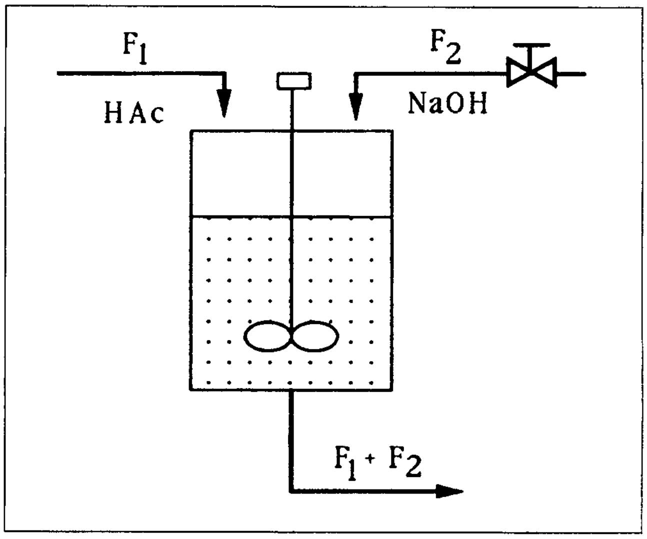


Figure 4. pH CSTR system.

$$SSE_i=\sum_{k=1}^{N_d}[\tau_i(k)-y_i(k)]^2 \tag{19}$$

and

$$S_{TT_i}=\sum_{k=1}^{N_d}[\tau_i(k)-\bar{\tau}_i(k)]^2 \tag{20}$$

$\bar{\tau}_i$ is the mean values of $\tau_i(k)$; N_d is the number of data.

The coefficient of determination for this simulation is 99.65.

Dynamic Learning of RNNs

In this section, dynamic learning of RNNs is discussed. After an introduction of the general learning procedure, simulation results on a SISO pH CSTR system, a MIMO nonisothermal CSTR system, as well as initial condition modeling of a MIMO batch biochemical system, will be given.

Dynamic training of RNNs was conducted with the difference forms of the learning algorithms defined by Eqs. 7, 14,

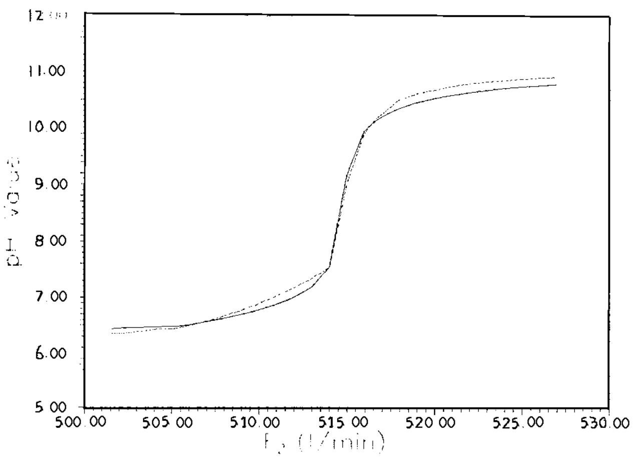


Figure 5. Desired (—) and RNN (- - -) outputs for steady-state relationship of pH CSTR.

15 and 16. To accelerate the convergence of the learning process and smooth the changes of weights and time constants, we added the momentum terms $\alpha_w \Delta w_{ij}(p-1)$ and $\alpha_T \Delta T_i(p-1)$ to Eqs. 14 and 15, respectively, where $0<\alpha_T$, $\alpha_w<1$.

The initial values of W were assigned to uniformly distributed random numbers in $[-1, 1]$. The initial values of the time constants were set equal to one. The initial neuron states, y_i's, depend on the initial values of training data.

Direct application of the learning law, as used in the simulations in this work, involves forward integration of y_i's from t_0 to t_f and backward integration of z_i's from t_f to t_0 with the initial condition $z_i(t_f)=0$, while corresponding changes on the weights and time constants are made according to Eqs. 14 and 15.

The training input data used for the systems under consideration are generated by adding periodic pseudorandom number sequence (PRNS) signals to the steady-state input values of the systems. The maximal magnitudes of the PRNS signals were selected to be large enough to force outputs to vary in value ranges of interest.

Note that the direct method requires storage of the forward integration results of y_i and $f_i'(x_i)$. Also the learning cannot be conducted in real-time fashion. However, the backward integration of z_i can be avoided. Pearlmutter (1989) used guessing of the initial value of $z_i(t_0)$, so that both Eqs. 7 and 16 can be integrated forward from t_0 to t_f. The point behind this approximation is that the changing of weights is slow during training. Thus, if we divide the time period $[t_0, t_f]$ into some small intervals and use a small learning rate and momentum factor, the error of the initial guess will have little effect on the learning. A similar argument also applies to backpropagation for feedforward networks and to other learning algorithms for RNNs (Williams and Zipser, 1989; Sato, 1990b). Thus, with an appropriate initial guess of z_i, the dynamic learning of RNNs can be conducted in a real-time manner.

SISO continuous systems

The pH CSTR of Figure 4 is used as an example of dynamic learning for SISO continuous systems. The input and output data for dynamic training of this system were generated by adding 2% PRNS signals to the steady-state input. Each period of the data contains 1,000 data pairs; the sampling period is 0.4 min. The RNN used for this simulation contains three fully connected hidden nodes and one fully connected output node. The weights in the RNN model were obtained within 1,000 epochs with the learning parameters $\eta_w=0.02$, $\eta_T=0.0001$, $\alpha_w=\alpha_T=0.7$. A lower bound of time constants is used in this simulation.

After training, the RNN output corresponding to the training input and the desired output are shown in Figure 6. The coefficient of determination for the RNN output is $R^2=94.80$. A different PRNS input signal (from the training PRNS input signal) was used to test the predictive ability of the RNN model. The results are given in Figure 7. The coefficient of determination for the prediction is $R^2=95.58$.

MIMO continuous systems

The extension of use of RNNs in modeling continuous MIMO systems is straightforward. As an example, a nonisothermal CSTR (Figure 8) with a first-order reaction, $A \rightarrow B$, is considered (see Appendix C). The variables u, F_i, C_{Ai}, and T_i are considered as the inputs of the system, while L, C_A, and T are the outputs.

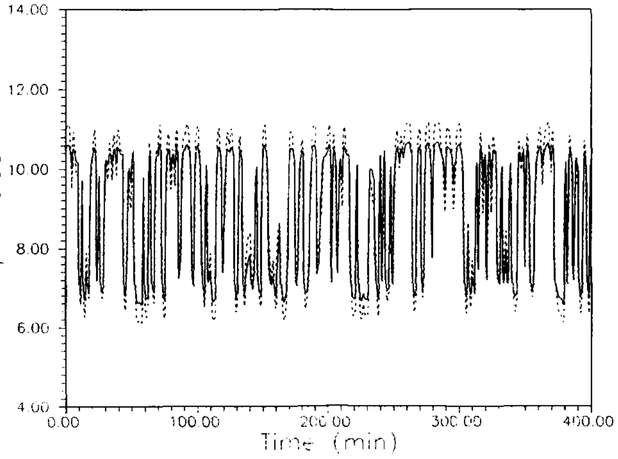


Figure 6. Desired (—) and RNN (· · ·) outputs after completion of dynamic training for the pH CSTR system.

The training data are generated by adding 3% PRNS signals to the steady-state input values (0.3 for u). The sampling period used here is 0.2 h. The training database has 1,000 input and output data pairs.

An RNN with 12 nodes is used for the modeling of this CSTR. There are four input nodes, three output nodes, one bias node, and eight hidden nodes. RNN parameters were obtained within 3,000 epochs. The learning parameters are $\eta_w=0.01$, $\eta_T=0.001$, and $\alpha_w=\alpha_T=0.7$. A lower bound of time constants is also used in this simulation. After training, the RNN output data of L, C_A and T based on the corresponding training inputs are given in Figure 9. The coefficients of determination for the three outputs are 98.90, 97.54, and 98.61, respectively.

A series of two alternating pulses for the input u (Figure 10a) of magnitude of 0.25 is used to test the predictive ability of the RNN model (other inputs are held at their corresponding steady-state values). The RNN prediction on the responses of L, C_A and T is given in Figure 10, along with the desired

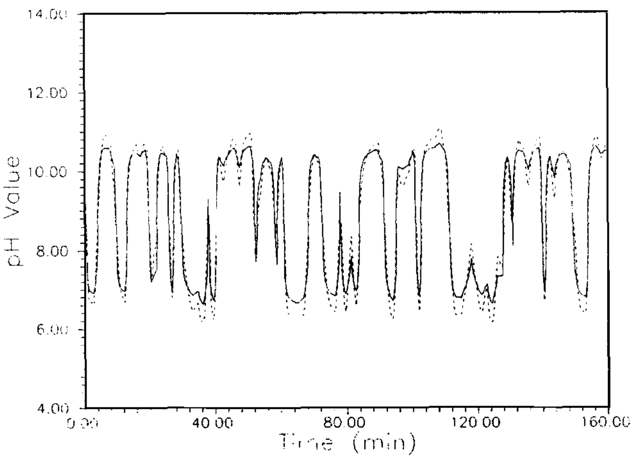


Figure 7. Desired output (—) and RNN prediction (· · ·) corresponding to test input for pH CSTR.

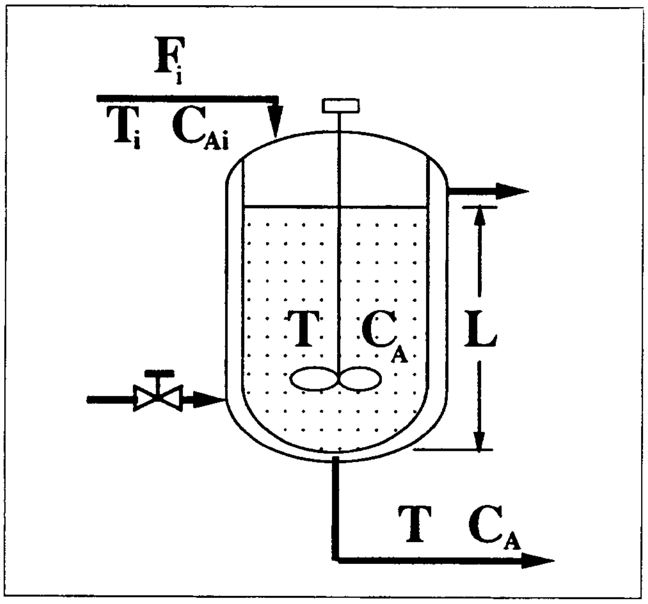


Figure 8. Nonisothermal CSTR.

responses. The coefficients of determination for the three outputs are 99.85, 98.39, and 98.26, respectively. Here, the coefficient of determination for level is calculated by adding 1% normally distributed white noise to the steady-state value.

Other tests, such as simultaneous pulse changes for different combinations among inputs, also showed good agreement. Thus, we conclude that the RNN model can successfully characterize the dynamics of this CSTR within the range of considered inputs.

To provide an idea about the impact of the number of hidden nodes, some simulation results using a different number of hidden nodes are given in Table 1, in terms of the value of the coefficients of determination for the fitting of training outputs. The corresponding values of the coefficients of determination for the test data (see Figure 10a) are given in Table 2.

Table 1 also gives a list of computation times when different numbers of hidden nodes are used.

MIMO batch systems

The output trajectories of a batch process are determined solely by their initial values. To model such a system, we need several training trajectories starting from different initial conditions. The objective is, then, to minimize:

$$E=\frac{1}{2}\sum_{m=1}^{M}\sum_{i=1}^{N}\int_{t_0}^{t_f}J_{i,m}^2(t)dt \tag{21}$$

where M is the total number of training trajectories; $J_{i,m}(t)=\tau_{i,m}(t)-y_{i,m}(t)$ when ith node is an output.

The difference between the training for batch processes and training for continuous ones is that the inputs are used only to initialize the RNN model, that is, solve the steady-state form of Eq. 7 for y_i^s. After initialization, there are no more external inputs to the RNN.

The batch process under consideration is a biochemical model developed by Bree et al. (1988) for hybridoma cell growth and the production of immunoglobulins (*IgG*) for a murine hybridoma cell line (see Appendix D).

Seven inputs of this system are the initial values of the variables, *Vc*(0), *G*(0), *Glu*(0), *Am*(0), *La*(0), *IgG*(0), and

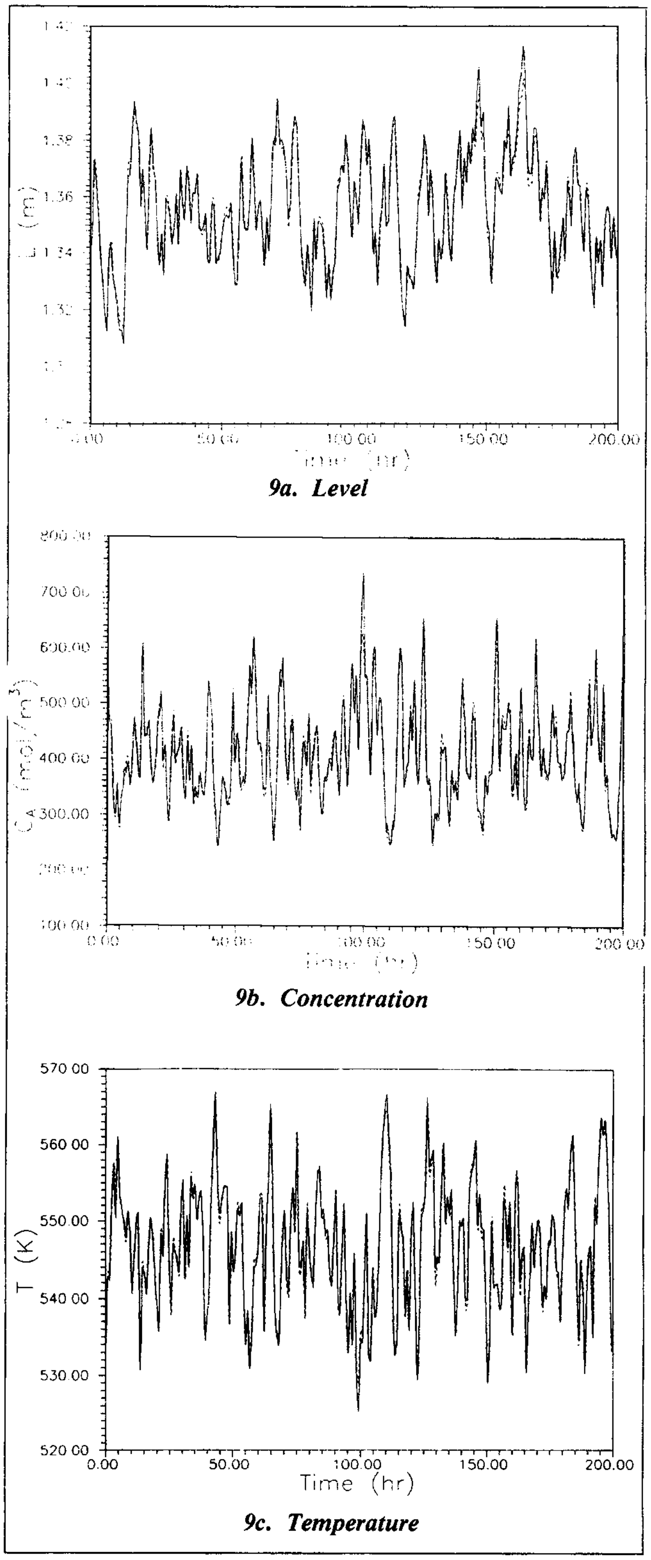


Figure 9. Desired (—) and RNN (· · ·) outputs after completion of dynamic training for nonisothermal CSTR.

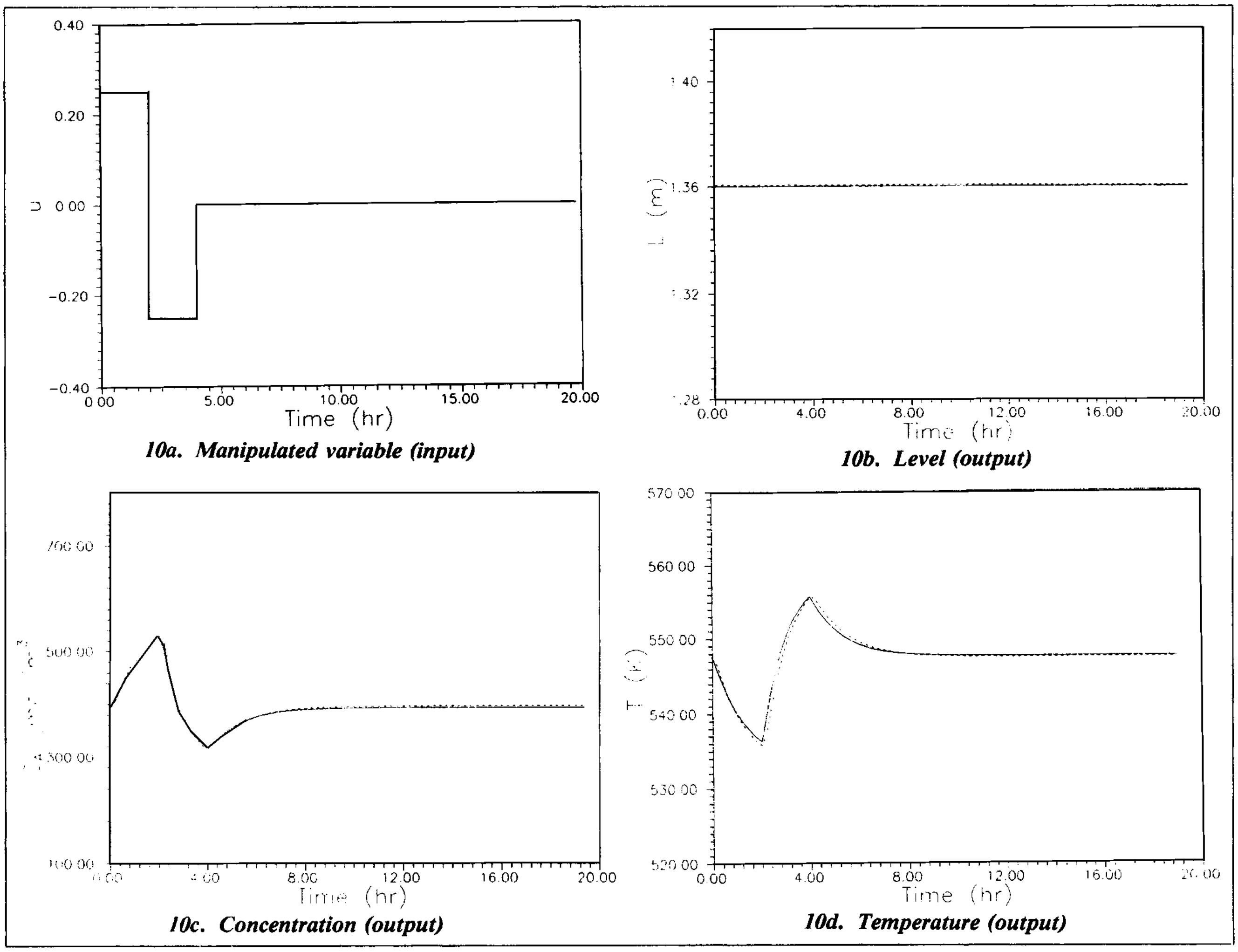


Figure 10. Desired output (—) and RNN prediction (· · ·) corresponding to test input for nonisothermal CSTR.

$EIgG(0)$. Seven outputs are the trajectories for X, G, Glu, Am, La, $EIgG$, and percent viability ($= Vc/X$) (see Eqs. D1 to D8).

The initial values used for generating training data for the RNN model are obtained by adding 2% PRNS signals to each input's normal value, with the initial values of $EIgG$ and IgG kept at zero. Ten different sets of output trajectories (with different initial conditions) are generated as the training data. The sampling period is 0.2 d.

An RNN with seven input nodes, seven output nodes, and one bias node is used for this simulation. No hidden node is used in this simulation. The results are obtained within 2,500 epochs with $\eta_w = 0.02$, $\eta_T = 0.0001$, and $\alpha_w = \alpha_T = 0.7$. The val-

Table 2. Coefficients of Determination (R^2) for Testing MIMO CSTR Model*

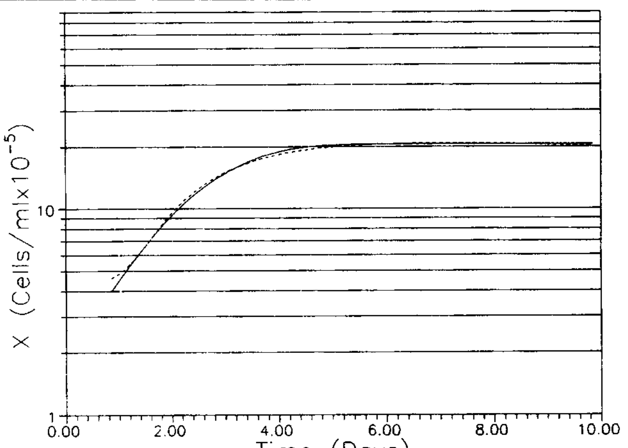
No. of Hidden Nodes*	Run 1			Run 2			Run 3		
	L**	T	C_A	L**	T	C_A	L**	T	C_A
0 (8)	96.07	94.18	87.28	98.04	92.23	90.32	95.12	93.81	91.58
1 (9)	95.28	93.52	94.18	97.92	96.81	97.00	94.10	95.89	96.20
2 (10)	99.11	96.89	98.31	97.52	97.00	99.36	97.58	96.40	98.85
3 (11)	92.06	92.89	95.62	98.91	95.04	98.09	99.16	96.00	98.74
4 (12)	94.55	96.92	98.02	98.14	96.22	98.02	99.34	94.82	98.93

*Total number of nodes given in parentheses.
**Calculated by adding 1% white noise to the steady-state value.

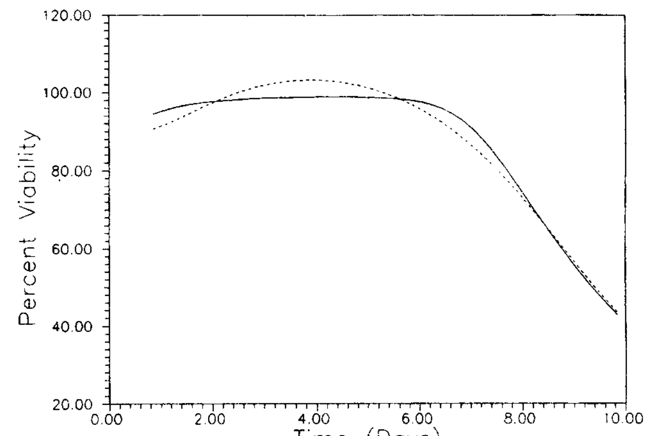
Table 1. Coefficients of Determination (R^2) for Training of MIMO CSTR*

No. of Hidden Nodes**	No. of Wt.	No. of Time Consts.	Compt. Time (h)	Run 1				Run 2				Run 3			
				$NE^{\dagger}$	L	T	C_A	NE	L	T	C_A	NE	L	T	C_A
0 (8)	24	7	1.05	1,440	88.66	92.22	93.22	1,555	88.38	93.83	94.05	1,621	89.12	94.89	93.75
1 (9)	36	8	1.49	1,898	91.82	94.41	95.65	3,997	93.31	95.41	97.34	2,952	93.35	94.67	96.83
2 (10)	50	9	2.03	4,000	92.21	95.66	96.65	4,000	95.92	95.51	96.56	4,000	95.16	94.47	97.79
3 (11)	66	10	2.41	2,075	95.55	94.03	96.18	3,998	97.46	95.13	97.35	3,557	96.30	95.79	97.05
4 (12)	84	11	2.90	3,984	94.99	95.85	97.26	3,776	95.83	95.05	96.50	3,984	94.92	95.44	96.81

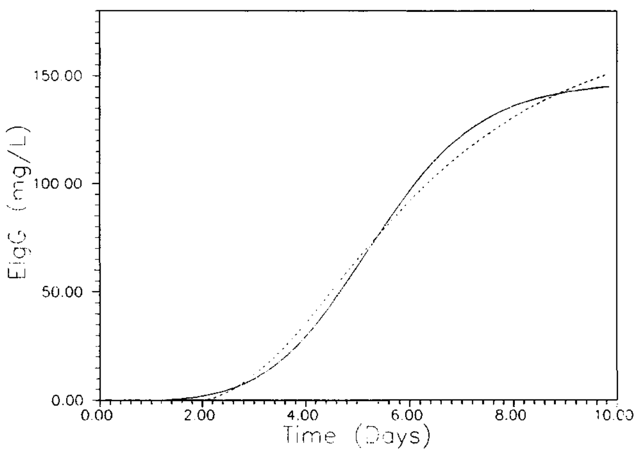
*$\alpha_w = 0.03$, $\eta_w = 0.7$; $\alpha_T = 0.0001$, $\eta_T = 0.7$; maximum epoch: 4,000
**Total number of nodes given in parentheses.
†Number of epoch at which the results attained.



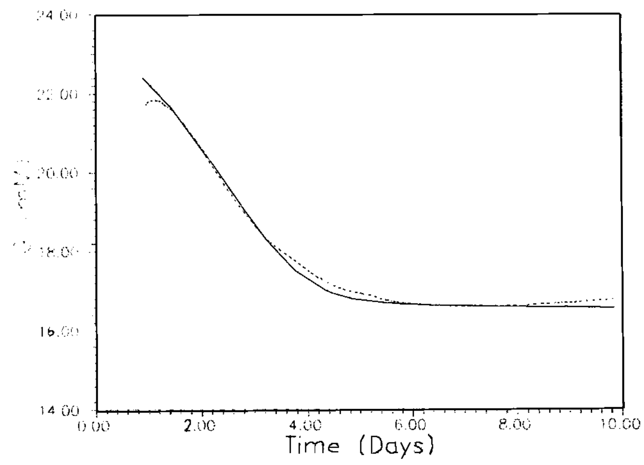
11a. Cell density



11b. Percent viability



11c. Concentration of extracellular IgG



11d. Concentration of glucose

ues of the coefficient of determination for the seven outputs are 99.95, 99.87, 99.62, 99.97, 99.45, 99.96, and 99.98.

The RNN prediction on a new set of initial conditions (other than those in the training database) is given in Figure 11. The values of the coefficient of determination for the seven predictions are 99.57, 98.43, 99.18, 99.07, 99.27, 99.49, and 98.92. The test initial conditions are also PRNS signals of the same order as in the training data. These figures show that the RNN model can predict the output trajectories very well, even when the initial conditions are not in the training database (but within the range of the training data).

Learning with Noisy Data

Training RNNs with noisy data is examined on the nonisothermal CSTR system. The noisy training data are generated by adding independent, normally distributed noise to the pure training data used in the previous section. The standard deviations for the inputs and outputs are 5% of the corresponding variable's range of interest, which approximately equal the range of the vertical axes in the figures for corresponding variables.

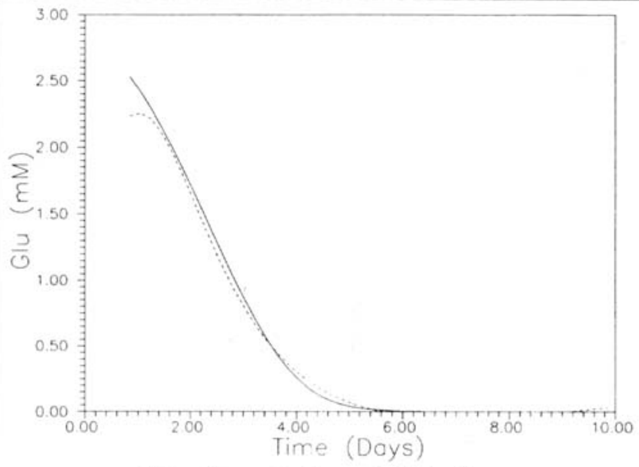
An RNN with four input nodes, three output nodes, and four hidden nodes was used. After 3,000 of epochs, converged RNN's output trajectories are shown in Figure 12. The coefficients of determination for the outputs L, C_A, and T, are 97.49, 96.32, and 97.64, respectively (compared with the pure outputs while noisy inputs are used). These figures show that, in spite of the presence of noise in the training data, the RNN predictions match the desired values well, albeit with higher error. Responses of the RNN model to pulse changes on u (Figure 10a) are given in Figure 13. The coefficients of determination for the predictions of L, C_A, and T, are 98.03, 97.31, and 98.38. Agreement has deteriorated, but may still be deemed acceptable.

From this simulation, we see that noise in training data has an aggravating, but not catastrophic, effect on RNN's predictive ability. Note also that no prefiltering of the noisy data was used here. Data smoothing could be used for improved results.

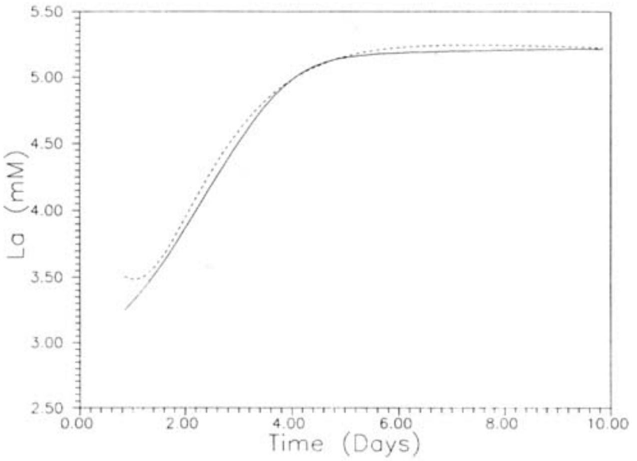
The computational effort for training with noisy data is the same as for training with pure data.

Comparison with Other Methods

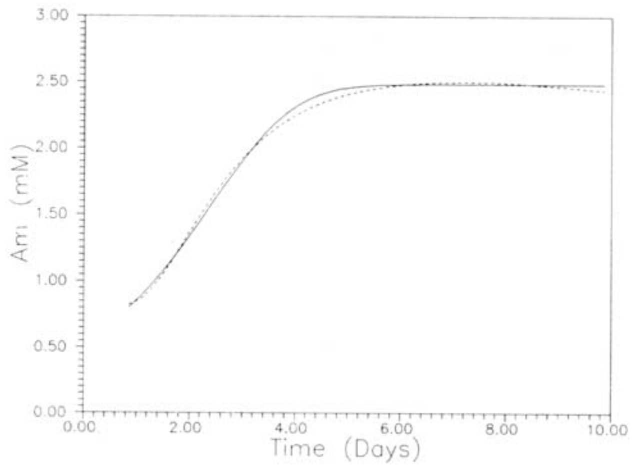
In this section, RNNs are compared with locally linearized models and backpropagation feedforward network models. The system to be modeled is the same MIMO CSTR used in previous sections. The same training data and testing data are used here for comparison of RNNs and backpropagation networks.



11e. Concentration of glutamine



11g. Concentration of lactate



11f. Concentration of ammonia

Figure 11. Desired output (—) and RNN prediction (···) corresponding to test initial conditions for biochemical batch system.

In the backpropagation network (Rumelhart et al., 1986), one past and the current input and output values are used, which, in addition to a bias node, give 15 input nodes. Six hidden nodes were used (including one as a bias node), thus resulting in a total of 93 connection weights. The RNN model has four hidden nodes (12 nodes in total).

The results for training and testing with pure data for backpropagation network models and RNN models are given in Table 3. Table 4 gives the corresponding results for the two methods when noisy data are used for training.

The locally linearized model was derived from Eqs. C1-C3. Step responses to $u = 0.25$ for the real CSTR, the RNN model, and the linearized model are given in Figure 14.

Discussion and Conclusions

Recurrent neural networks have two salient features that distinguish them from feedforward neural networks. One is their node characteristics, which involve nonlinear *dynamic* functions (ordinary differential equations), while FNNs have only *static* nonlinear node characteristics. The other major distinction is topology. In RNNs there are both feedforward and feedback connections, while in FNNs there are only feedforward paths.

Compared to FNN models, RNN models require knowledge only of the current value $v(k)$ for a given input v, as opposed to a number of past values $v(k-1), \ldots, v(k-m)$ that feedforward networks require. This drastically decreases the dimensionality of the input space, especially for multivariable cases. It also eliminates the trial-and-error process for determining the "optimal" number of past input or output values to be used. Moreover, an RNN model is a set of coupled ODEs in continuous time domain and can be discretized with different

Table 3. Comparison of R^2 for FNN and RNN Models

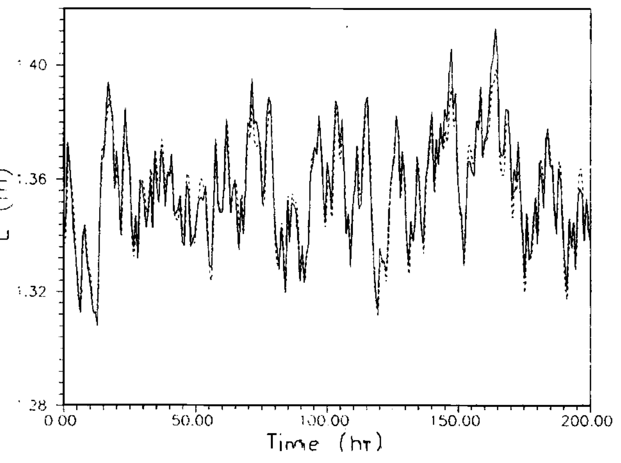
Method	No. of params	Compt Time (h)	Training L	Training T	Training C_A	Testing L^*	Testing T	Testing C_A
FNN	98	0.49	99.19	98.47	98.48	99.29	98.73	96.62
RNN	84+11	2.10	98.90	97.54	98.61	99.85	98.39	98.26

*Calculated by adding 1% white noise to the steady state.

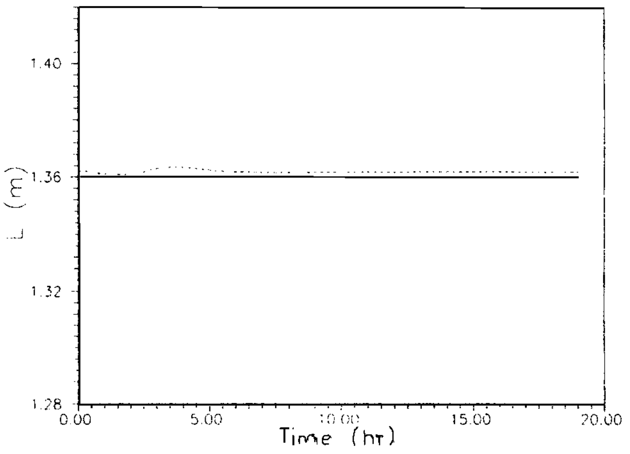
Table 4. Comparison of R^2 Using Noisy Data

Method	No. of params	Compt. time (h)	Training L	Training T	Training C_A	Testing L^*	Testing T	Testing C_A
FNN	93	0.49	97.02	97.53	97.97	99.19	99.39	97.71
RNN	84+11	2.10	97.49	96.32	97.64	98.03	97.31	98.38

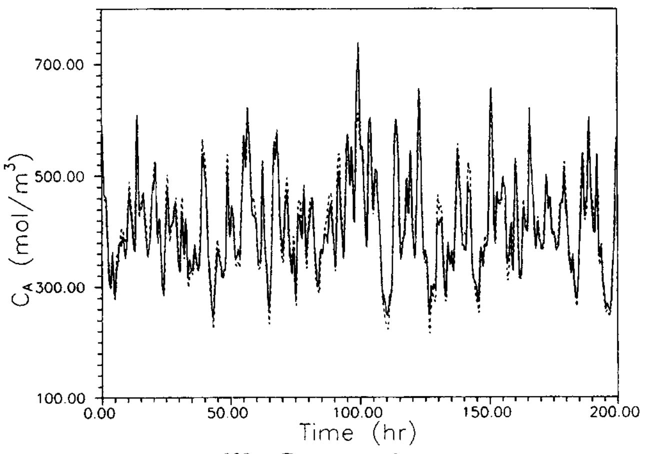
*Calculated by adding 1% white noise to the steady state.



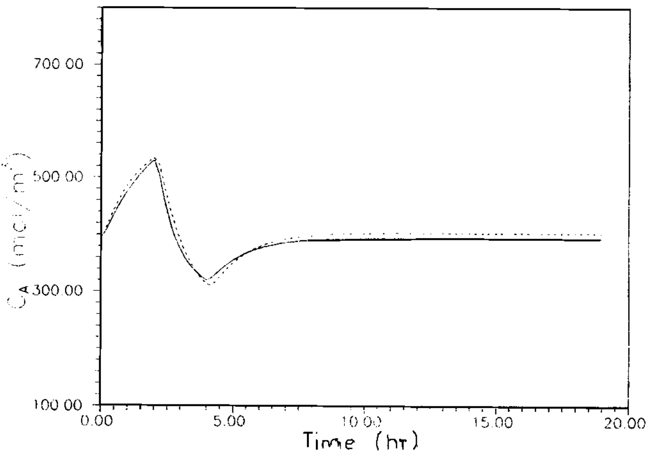
12a. Level



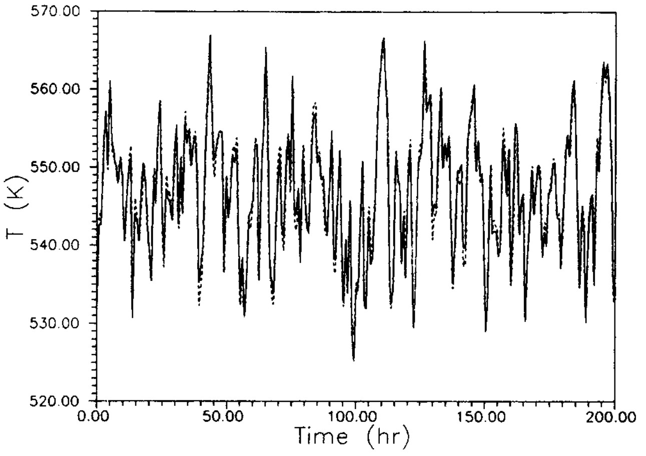
13a. Level



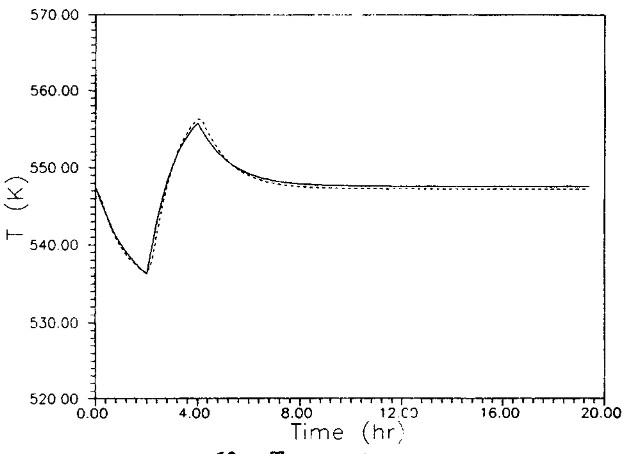
12b. Concentration



13b. Concentration



12c. Temperature



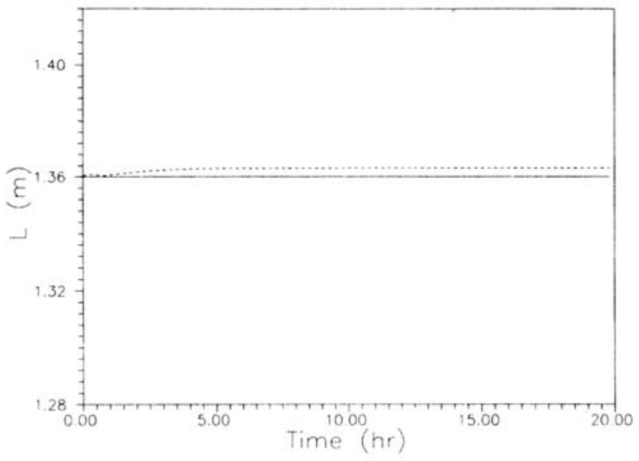
13c. Temperature

Figure 12. Desired (—) and RNN (- - -) outputs after completion of dynamic training with noisy data.

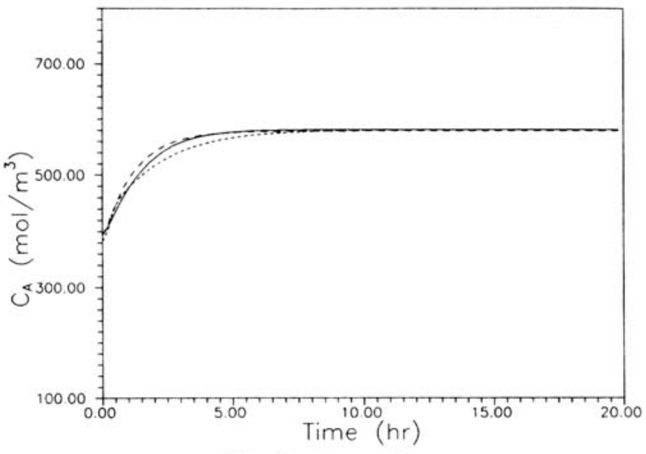
Figure 13. Desired output (—) and RNN prediction (---) corresponding to test input for nonisothermal CSTR.

sampling period. FNN models are discrete version with fixed sampling period. Thus, the structure of RNN models is more natural and attractive than that of FNNs. Comparisons of RNNs, locally linearized linear models, and feedforward neural networks have also been conducted. The modeling capabilities of RNNs and FNNs are comparable, but the training of RNNs takes longer time.

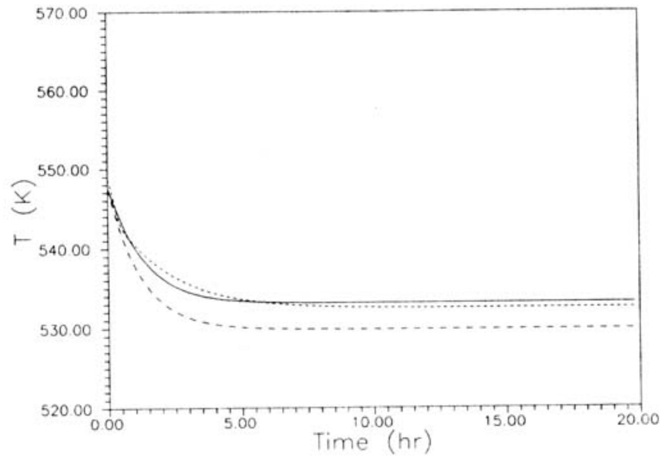
Simulation results in this work show that RNNs can learn both steady-state relationships and process dynamics of continuous and batch, SISO and MIMO systems in a simple and direct manner. The dynamic modeling capability of RNNs (including learning from noisy data) can be used for dynamic process simulation and control.



14a. Level



14b. Concentration



14c. Temperature

Figure 14. Step responses of the locally linearized model (— — —), RNN model (· · · ·), and exact model (———) of MIMO CSTR.

Notation

A	=	cross area of reactor, reactant
$[AC^-]$	=	concentration of acetate ion
Am	=	concentration of ammonia
B	=	product
c_p	=	specific heat
C_1	=	concentration of acetic acid
C_2	=	concentration of NaOH
C_A	=	concentration of A in reactor
C_{Ai}	=	concentration of A in inlet stream
E	=	activation energy
E	=	objective function
$EIgG$	=	extracellular concentration of immunoglobulins
f	=	transfer function of a neuron
f_i	=	transfer function of a neuron
F_1	=	flow rate of acetic acid
F_2	=	flow rate of NaOH
F_i	=	inlet flow rate of CSTR
G	=	concentration of glucose
Glu	=	concentration of glutamine
$[H^+]$	=	concentration of hydrogen ion
$[HAC]$	=	concentration of acetic acid
ΔH_R	=	reaction heat
IgG	=	intracellular concentration of immunoglobulins
I_i	=	external inputs of RNNs
J_i	=	error in output nodes
k	=	sampling point, discrete time
k_0	=	reaction rate constant
K	=	constant (see Table 6)
K_a	=	acetic acid equilibrium constant (1.8×10^{-5})
K_w	=	water equilibrium constant (10^{-14})
L	=	level
La	=	concentration of lactate
M	=	number of trajectories
N	=	total number of nodes in a RNN
N_d	=	number of data
N_l	=	number of nodes in layer l
$[\mathrm{Na}^+]$	=	concentration of sodium ion
$[\mathrm{OH}^-]$	=	concentration of hydroxyl
p	=	epoch number
Q	=	heat removed from the CSTR
R	=	gas constant
R^2	=	coefficient of determination
S_{TT}	=	see Eq. 20
t	=	time
t_0	=	lower integration limit
t_f	=	upper integration limit
Δt	=	difference time step
T	=	reactor temperature
T_i	=	temperature of inlet stream, time constant of a neuron
$\Delta T_p(p)$	=	change of time constant
u	=	$(Q-Q_s)/Q_s$, manipulated variable
u_i	=	state of an associated system for static learning
v	=	input variable
V	=	effective reactor volume
Vc	=	viable cell count
w_{ij}	=	weights
Δw_{ij}	=	change of weights
W	=	$\{w_{ij}\}$
x	=	weighted input of a neuron
X	=	total cell count
y	=	output of a neuron
z	=	state of an associated system for dynamic learning

Greek letters

α_T	=	momentum factor for time constants
α_w	=	momentum factor for weights
ζ	=	$[\mathrm{Na}^+]$
η_T	=	learning rate for time constants
η_w	=	learning rate for weights
θ_i	=	threshold value for the ith node
μ_1	=	specific growth rate of the viable cell
ξ	=	$[\mathrm{HAC}]+[\mathrm{AC}^-]$
ρ	=	density
τ_i	=	target, desired value
$\bar{\tau}_i$	=	mean value of τ_i
Ω_1	=	specific death rate of viable cells

Superscripts

$'$ = derivative
s = steady state
l = layer number in FNNs

Literature Cited

Almeida, L. B., "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment," *Proc. IEEE Int. Conf. on Neural Networks*, Vol. 2, San Diego, CA (June, 1987).
Almeida, L. B., "Back-propagation in Non-feedforward Networks," *Neural Computing Architectures*, I. Aleksander, ed., MIT Press, Cambridge, MA (1989).
Amari, S., "Characteristics of Random Nets of Analog Neurons," *IEEE Trans. Syst. Man Cybern.*, **SMC-2**(5), 643 (1972).
Bhat, N., and T. J. McAvoy, "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems," *Computers Chem. Eng.*, **14**(4/5), 573 (1990).
Bree, M. A., P. Dhurjati, R. F. Groghegan, and B. Robnett, "Kinetic Modeling of Hybridoma Cell Growth and Immunoglobulin Production in a Large-Scale Suspension Culture," *Biotechnol. and Bioeng.*, **32**(10), 1067 (1988).
Cohen, M. A., and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Trans. Syst. Man Cybern.*, **SMC-13**, 815 (1983).
Gherrity, M., "A Learning Algorithm for Analog, Fully Recurrent Neural Networks," *Proc. IJCNN*, Vol. 1, Washington, DC (1989).
Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Computational Abilities," *Proc. Nat. Acad. Sci. USA*, **79**, 2554 (1982).
Hopfield, J. J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Nat. Acad. Sci. USA*, **81**, 3088 (1984).
Hopfield, J. J., personal communication (1991).
Hoskins, J. C., and D. M. Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering," *Computers Chem. Eng.*, **12**(9/10), 881 (1988).
Luyben, W. L., *Process Modeling, Simulation, and Control for Chemical Engineers*, 2nd ed., McGraw-Hill, New York (1990).
McCulloch, W. S., and W. Pitts, "A Logical Calculus of the Ideas Imminent in Nervous Activity," *Bull. of Math. Biophys.*, **5**, 115 (1943).
Milton, J. S., and J. C. Arnold, *Introduction to Probability and Statistics*, McGraw Hill, New York (1990).
Nowlan, S. J., "Gain Variation in Recurrent Error Propagation Networks," *Complex Systems*, **2**, 305 (1988).
Pearlmutter, B. A., "Learning State Space Trajectories in Recurrent Networks," *Proc. IJCNN*, Vol. 2, Washington, DC (1989).
Piñeda, F. J., "Generalization of Back-Propagation to Recurrent Neural Networks," *Phys. Rev. Lett.*, **59**(19), 2229 (1987).
Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, **323**, 533 (1986a).
Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, MIT Press, Cambridge, MA (1986b).
Sato, M., "A Real Time Learning Algorithm for Recurrent Analog Neural Networks," *Biol. Cybern.*, **62**, 237 (1990a).
Sato, M., "A Learning Algorithm to Teach Spatiotemporal Patterns to Recurrent Neural Networks," *Biol. Cybern.*, **62**, 259 (1990b).
Simard, P. Y., M. B. Ottaway, and D. H. Ballard, "Analysis of Recurrent Backpropagation," *Proc. Connectionist Models Summer School*, p. 103, Morgan Kaufmann, San Mateo, CA (1989).
Stephanopoulos, G., *Chemical Process Control: An Introduction to Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ (1984).
Simpson, P. K., *Artificial Neural Systems*, Pergamon, New York (1990).
Ungar, L. H., B. A. Powell, and S. N. Kamens, "Adaptive Network for Fault Diagnosis and Process Control," *Computers Chem. Eng.*, **14**(4/5), 561 (1990).
Werbos, P. J., "Generalization of Backpropagation with Application to a Recurrent Gas Market Model," *Neural Networks*, **1**, 339 (1988).
Williams, R. J., and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, **1**, 270 (1989).

Appendix A: Equivalence of Eqs. 6 and 7

When time constants, T_i, are equal to 1, Eq. 6 can be obtained by a linear transformation from Eq. 7. Define the linear transformation

$$x_i = \sum_{j=1}^{N} w_{ij} y_j \tag{A1}$$

in Eq. 7. Then, take derivative of Eq. A1 to obtain:

$$\frac{dx_i}{dt} = \sum_{j=1}^{N} w_{ij} \frac{dy_j}{dt} \tag{A2}$$

Next, substitute Eq. 7 (note that $T_i = 1$) into Eq. A2 to get:

$$\begin{aligned}\frac{dx_i}{dt} &= \sum_{j=1}^{N} w_{ij}\left[-y_j + f_j\left(\sum_{i=1}^{N} w_{ji} y_i\right) + I_j\right] \\ &= -\sum_{j=1}^{N} w_{ij} y_j + \sum_{j=1}^{N} w_{ij} f_j\left(\sum_{i=1}^{N} w_{ji} y_i\right) + \sum_{j=1}^{N} w_{ij} I_j \\ &= -x_i + \sum_{j=1}^{N} w_{ij} f_j(x_j) + \sum_{j=1}^{N} w_{ji} I_j \end{aligned} \tag{A3}$$

After defining a new input I_i as $I_i = \Sigma_{j=1}^{N} w_{ij} I_j$ and changing the variable x_i to y_i in Eq. A3, we immediately obtain Eq. 6.

Appendix B: pH CSTR System

The pH CSTR system (Figure 4) used by Bhat and McAvoy (1990) has two inlet flows: one containing acetic acid (F_1), the other containing sodium hydroxide (F_2). The dynamics of the system is determined by the balances on $[HAC] + [AC^-](\xi)$ and $[Na^+](\zeta)$ (Bhat and McAvoy, 1990):

$$V\frac{d\xi}{dt} = F_1 C_1 - (F_1 + F_2)\xi \tag{B1}$$

$$V\frac{\zeta}{dt} = F_2 C_2 - (F_1 + F_2)\zeta \tag{B2}$$

and the auxiliary equations:

$$K_a = \frac{[AC^-] + [H^+]}{[HAC]}$$

$$K_w = [H^+][OH^-]$$

$$\zeta + [H^+] = [OH^-] + [AC^-]$$

where K_a and K_w are equilibrium constants. The values of the parameters and steady-state input and output values are shown in Table 5 (Bhat and McAvoy, 1990).

Table 5. Parameter and Steady-State Values for the *pH* CSTR System

V (l)	C_1 (mol/l)	C_2 (mol/l)	F_1 (l/min)	F_2 (l/min)	pH
1,000	0.3178	0.05	81	515	9

Table 6. Parameter Values for the CSTR System

A (m^2)	K (m$^{5/2}$/h)	k_0 (1/h)	E/R (K)
1.000	0.9715	7.080×10^7	8,375
ΔH_R (J/mol)	c_p (J/kg·K)	ρ (kg/m^3)	Q (J/h)
$-69{,}775$	3,140	800.8	1.055×10^8

Table 7. Steady-State Input and Output Values for the CSTR System

u	F_i (m^3/h)	C_{Ai} (mol/m^3)	T_i (K)
0.000	1.133	8,008	373.3
L (m)	C_A (mol/m^3)	T (K)	
1.360	393.3	546.7	

Appendix C: Nonisothermal CSTR System

The nonisothermal CSTR system used in this work is characterized by the following dynamic equations (see, for example, Stephanopoulos, 1984; Luyben, 1990):

$$A\frac{dL}{dt}=F_i-K\sqrt{L} \tag{C1}$$

$$\frac{dC_A}{dt}=\frac{F_i}{V}(C_{Ai}-C_A)-k_oe^{-E/RT}C_A \tag{C2}$$

$$\frac{dT}{dt}=\frac{F_i}{V}(T_i-T)+\frac{-\Delta H_R}{c_p\rho}k_oe^{-E/RT}C_A-\frac{Q}{c_p\rho V}(1+u) \tag{C3}$$

where $V=AL$ is the volume of the CSTR. The values of the parameters used in the model are shown in Table 6. The steady-state input values and the resulting steady-state output values are given in Table 7.

Appendix D: Biochemical Batch System

The dynamics of the biochemical batch process is described by the following equations:

$$\frac{dVc}{dt}=(\mu_1-\Omega_1)Vc \tag{D1}$$

$$\frac{dX}{dt}=\mu_1 Vc \tag{D2}$$

Table 8. Parameter Values for the Biochemical System

Parameter	Value	Unit
$\mu_{\max}$	3.0	day^{-1}
m	1.25×10^{-8}	mM glucose/day/viable cells mL
K_{Glu}	0.8	mM glutamine
K_{AI}	1.05	mM ammonia
K_{LI}	8.0	mM lactate
K_{GD}	5.0×10^{-4}	mM glutamine
K_{AD}	1.44	mM ammonia
K_{LD}	15.0	mM lactate
K_{XI}	1.0×10^7	total number of cells/mL
K_{GP}	0.01	mM glutamine
k_d	2.0	day^{-1}
k_p	1.5×10^{-7}	mg *IgG*/viable cells/day
k_s	0.8	day^{-1}
Y_G	2.9×10^5	total cells/mL/mM glucose
Y_{Glu}	9.434×10^5	total cells/mL/mM glutamine
Y_p	1,000	mg *IgG*/mL/mM glutamine
Y_{AG}	2.1937	mM ammonia/mM glutamine
Y_A	1.5	mM glutamine/mM ammonia
Y_L	3.00	mM glucose/mM lactate

$$\frac{dG}{dt}=-\frac{1}{Y_G}\mu_1 Vc-mVc \tag{D3}$$

$$\frac{dGlu}{dt}=-\frac{1}{Y_{Glu}}\mu_1 Vc-\frac{1}{Y_P}\frac{dIgG}{dt}-\frac{1}{Y_{AG}}\frac{dAm}{dt} \tag{D4}$$

$$\frac{dAm}{dt}=-\frac{1}{Y_A}\frac{dGlu}{dt} \tag{D5}$$

$$\frac{dLa}{dt}=-\frac{1}{Y_L}\frac{dG}{dt} \tag{D6}$$

$$\frac{dIgG}{dt}=k_PVc\left(\frac{Glu}{K_{GP}+Glu}\right)\left(\frac{X}{K_{XI}+X}\right) \tag{D7}$$

$$\frac{dEIgG}{dt}=k_s(IgG-EIgG) \tag{D8}$$

where

$$\mu_1=\mu_{\max}\left(\frac{Glu}{K_{Glu}+Glu}\right)\left(\frac{K_{AI}}{K_{AI}+Am}\right)\left(\frac{K_{LI}}{K_{LI}+La}\right)$$

is the specific growth rate of the viable cells and

$$\Omega_1=k_d\left(\frac{Am}{K_{AD}+Am}\right)\left(\frac{La}{K_{LD}+La}\right)\left(\frac{K_{GD}}{K_{GD}+Glu}\right)$$

is the specific death rate of the viable cells. The parameter values used in these equations are shown in Table 8 (Bree et al., 1988).

Manuscript received Nov. 25, 1992, and revision received Mar. 3, 1993.