Neural Network Architecture for Process Control Based on the RTRL Algorithm

Tibor Chovan, Thierry Catfolis, and Kürt Meert
Expert Systems Applications Development Group, Dept. of Chemical Engineering
Katholieke Universiteit Leuven, de Croylaan 46, B-3001 Heverlee, Belgium

Neural-network-based control schemes are generally designed by replacing standard elements of the classic control schemes by feedforward neural networks. The introduction of discrete time recurrent networks, which are inherently dynamic systems, into those schemes can simplify the design of neural controllers. The concept of applying recurrent networks in indirect adaptive control schemes is described. A combined network cluster consisting of the control network and the model network is constructed to allow the use of the real-time recurrent learning algorithm. To demonstrate the feasibility of the method two simulation examples are presented.

Introduction

Since the 1980s, artificial neural networks have been studied intensively with special regard to engineering applications. Techniques based on neural networks have been developed in several fields, ranging from banking and speech recognition to processor scheduling.

Advantages of neural networks, such as parallel computation, nonlinear mapping, and learning capabilities make them an attractive solution in many chemical engineering problems. Chemical engineering systems are typically nonlinear with complex dynamics, and our knowledge—and models—of them are often defective or uncertain. Typical chemical engineering applications based on the excellent classification properties of some neural networks include diagnosis of chemical plants (Watanabe et al., 1989; Venkatasubramanian et al., 1990; Catfolis, 1993b) and controller adaptation (Cooper et al., 1992). Based on the nonlinear mapping capabilities, problems like the identification, simulation, and control of chemical processes have been studied (Bhat and McAvoy, 1990; Willis et al., 1992; Meert and Catfolis, 1994). Neural controllers have been used in robotics (Sanderson, 1990), and applications in chemical industry have also been investigated (Hangos, 1992; Chen and Weigand, 1994; Meert, 1994).

Our basic goal is to build a neurocontroller based on the real-time recurrent learning (RTRL) algorithm. This controller omits some of the disadvantages of the more classic neurocontroller architectures, for example, time windowing to control dynamic processes (Bhat and McAvoy, 1990). To allow an easy calculation of the control network we combine model and controller into one large network with a new architecture based on network clusters. In order to test these controllers, they have been applied to two dynamic processes, a simple level control system and the control of a highly nonlinear bioreactor. In spite of the controller offset, which can be attributed primarily to the error of the model network, the results are very promising. In future studies, we want to try and solve this offset problem by adapting the model network. We want to emphasize that we did not optimize the neurocontroller performance by fine-tuning the neural networks, but rather made a generic architecture to demonstrate the advantages when solving highly nonlinear control problems.

In the first section we review the basic artificial neural networks architectures, with an emphasis on the RTRL algorithm. In the second section we describe some classic control schemes with neural networks and we introduce the use of the clustered RTRL algorithm in these control schemes. In the last two sections we demonstrate the feasibility of this method with two examples.

Neural Networks and Dynamic Systems

Numerous types of artificial neural networks exist, but each type consists of the same basic features: nodes, layers, and connections. The smallest element of a network is the node. Every node receives signals from connections, or links, that it

Correspondence concerning this article should be addressed to T. Catfolis.
Current address of T. Chovan: Dept. of Chemical Engineering Cybernetics, University of Veszprem, Egyetem utca 10, P.O. Box 158, H-8201 Veszprem, Hungary.

processes in a combination function and then converts to an output signal with a transfer function. These nodes are connected to each other by weighted links. These weights can be adapted by one or the other learning rule and represent the *long-term memory* of the neural net.

Learning or training of the neural network means finding a set of parameters (weights) that produce the desired behavior. Learning methods are divided into three main classes (Karna and Breen, 1989). *Supervised learning systems* are given training samples, that is, the input values and the corresponding desired output values. Networks applying the backpropagation or RTRL algorithm are typical examples of this group. *Reinforcement learning systems* do not receive the desired output value, but rather some performance measure of their operation. The concept of simulated annealing is based on this method. *Unsupervised learning systems* develop their own rules by extracting information from examples. No direct information about their operation is given to them. This method is often used in classification networks like adaptive resonance theory (ART) networks (Carpenter and Grossberg, 1987). In this article we will only discuss supervised learning systems.

Networks can be divided into two main classes by their topologies: feedforward networks, which do not contain any directed loop in their representing graph, and recurrent networks, which do.

Feedforward networks

Feedforward neural networks have been the more frequently used models. Their most typical form is the *multilayer network*. They basically give a static mapping of their inputs on their outputs. It can be proved (Hornik et al., 1989) that a three-layer network (input-hidden-output layer) can approximate any nonlinear function with arbitrary precision, given an infinite amount of nodes in the hidden layer.

The identification of the connection weights for real-world applications (the learning) is usually performed using the *backpropagation* (BP) algorithm, which is basically a form of the gradient descent method. The BP algorithm provides an efficient way for calculating the partial derivatives of the squared output error with respect to the weights by propagating the error backward to the previous layer (for details see Freeman and Skapura, 1991; Rumelhart et al., 1986).

When applying multilayer feedforward neural networks for modeling the system dynamics, a discrete time history of system inputs and outputs can be used as network input and target values. The network acts as a nonlinear mapping corresponding to a nonlinear form of the input–output model, which is widely used in control engineering and signal processing. The mapping is performed by the learning process, that is, applying to the neural net a number of training patterns consisting of input vectors and the corresponding target values.

A biologically inspired neural model for describing dynamic behavior is suggested by Wan (1993). His *discrete time neural network* extends the classic multilayer structure by using finite linear filters as tapped delay lines between neurons. The learning method is the temporal backpropagation algorithm that is a vector generalized form of the original backpropagation scheme. The main advantage of this model is that it maintains the local character of the computation of the network. The network itself operates by opening a wider time window to the original input history. Besides the usual problem of determining the appropriate number of neurons, discrete time neural networks have a specific problem—how to determine the number of delays in the filter lines.

Recurrent networks

A neural network becomes a dynamic system when feedback connections are introduced. In the case of continuous time networks the network operation can be described by differential equations of the nodes, whereas in the case of discrete time networks difference equations can be used (Narendra and Parthasarathy, 1990). Several models and learning methods of recurrent networks are known. *Discrete time recurrent networks* are built on a synchronous computing scheme, that is, activities of all the nodes are evaluated using the current inputs of the network and the current outputs of the neurons. Then all new outputs are calculated and take effect. A few of the recurrent networks, which are possible applicants for use in control, are discussed in the following paragraphs.

Pham and Liu (1993) use a modification of the *Elman network* for identification of dynamic systems. The Elman-type network is one of the simplest recurrent networks that can be trained using the backpropagation algorithm. The Elman network contains an extra layer called a *context layer* added to a one-hidden-layer feedforward network. The context neurons receive their inputs from the hidden layer with a fixed weight of 1.0 and feed back their outputs to the hidden layer through adaptable weights. The modification is the introduction of a self-feedback connection with fixed weight on each context neuron. Some improvements in modeling nonlinear dynamic systems are demonstrated in this article using this modified network. The simple structure, the relatively low number of adaptable weights, and the use of the standard backpropagation algorithm is promising for control applications. This architecture was also used by Ungar (1990).

Schmidhuber's (1989) concept of *neural bucket brigade* allows any arbitrary recurrent network topology with neurons organized into small partitions called *competitive subsets*. In the subsets a kind of winner-take-all mechanism is applied and learning takes place via the distribution by the winner of some of its "weight substance" among the other active competitors. The main advantage of this system is that the learning algorithm is local in space and time, that is, only information in a fixed recent time window from the node itself and its neighbors is required for the computation. The feasibility of the method is demonstrated on sequence generation and sequence recognition, among others.

Fully connected recurrent network

Fully connected recurrent neural networks probably have the most general topology. They can represent any feedforward or other, more simple recurrent structure. However, a discrete time delay (the minimum delay depends on the actual implementation of the algorithm and is one for this implementation) emerges, due to the structure of the network. Processes with an unknown time delay can easily be modeled

by a fully connected recurrent network if the process time delay is larger than the minimal network time delay and if the number of neurons is large enough.

Real-time recurrent learning algorithm

For fully connected recurrent networks Williams and Zipser (1989) derived a learning algorithm based on the gradient descent method. Since the weights are updated at each time step rather than at the end of a trajectory, the method, RTRL, is well-suited for on-line training. The concise form of the algorithm is presented here for later reference. The current output of a neuron is expressed as

$$z_k(t+1) = f_k\left(\sum_{l \in U \cup I} w_{kl} z_l(t)\right), \qquad k \in U, \tag{1}$$

where U is the set of the nodes in the recurrent layer, I is the set of input nodes, $z_k(t)$ is the current value of the corresponding input or output node, w_{kl} denotes the weight of the connection from node $l \in U \cup I$ to node $k \in U$, and $f_k(\cdot)$ stands for the transfer function of neuron k. The error of the target nodes is defined as

$$e_k(t) = \begin{cases} d_k(t) - z_k(t) & \text{if } k \in T(t) \\ 0 & \text{otherwise,} \end{cases} \qquad T(t) \subset U, \tag{2}$$

where $d_k(t)$ is the target value and $T(t)$ is the set of target neurons. The algorithm minimizes the sum of squared error and takes the following form

$$\Delta w_{ij}(t) = \alpha \sum_{k \in U} e_k(t) p_{ij}^k(t), \qquad i \in U \quad \text{and} \quad j \in U \cup I, \tag{3}$$

where α stands for the learning rate and

$$p_{ij}^k(t) = \frac{\partial z_k(t)}{\partial w_{ij}}$$

for the impact that can be calculated by the following recursive formula when a sigmoid transfer function is used

$$p_{ij}^k(t+1) = z_k(t+1)[1 - z_k(t+1)]\left[\sum_{l \in U} w_{kl} p_{ij}^l(t) + \delta_{ik} z_j(t)\right]$$

$$\text{and} \quad p_{ij}^k(t_0) = 0, \tag{4}$$

where δ_{ik} is the Kronecker delta (1 if $i = k$, 0 otherwise).

Although the RTRL algorithm gives a simple computation scheme, it has several computational disadvantages. The main problem is that the calculation is not local in the sense that it requires the use of all weights and activities in the network for the evaluation of each one of the nodes. Another problem is that the RTRL algorithm shows very slow convergence in many cases. A method for speeding up learning is demonstrated by Catfolis (1993a). It is based on resetting the learning algorithm (clearing the impacts) with a frequency that can be related to the time constants of the process.

Neural-Network-Based Control Architectures

Applications of neural networks for control are expected to make the achievement of the following capabilities feasible (Werbos, 1989):

- Implementation in parallel hardware.
- Real-time adaptation without instability.
- Handling severe nonlinearity and noise.
- "Planning" or optimizing over time.
- Controlling a large number of actuators in parallel.
- Coupling slower, comprehensive controllers to faster subordinate systems.

Neural-network-based control schemes are generally constructed by replacing linear models by neural networks in the standard control engineering structures. Possible architectures for control are discussed in several papers (Tanomaru and Omatu, 1992; Levin et al., 1991; Barto, 1990; Narendra and Parthasarathy, 1990; Werbos, 1989). Since the neural model has a nonlinear character, the analysis of the stability and robustness poses several problems and a great deal of current research effort is dedicated to finding a solution to these problems.

In most cases the neural networks used in control schemes are multilayer feedforward networks. The input of the network is formed by using a process input and process output history. The assignment of inputs and outputs is determined according to the requirements of the actual problem. The schemes can easily be adapted to multi-input–multi-output cases by applying the time window to all relevant process inputs and outputs.

The most frequently used schemes are based on *supervised learning*. In *supervised control*, neural networks are trained to map the input sensor signals onto the desired actions, which are given by a human expert. This kind of solution can be used to train controllers for tasks that can be successfully solved by human operators.

Direct inverse control is based on the process input and output signals. The training scheme is shown in Figure 1. This method can be used for both on- and off-line training. Since it relies heavily on the generalization ability of the neural network, it is not recommended that it be used as the only training scheme.

Direct adaptive control adjusts the controller parameters (weights) according to the error on the process output (see Figure 2). This method requires calculation of the sensitivity

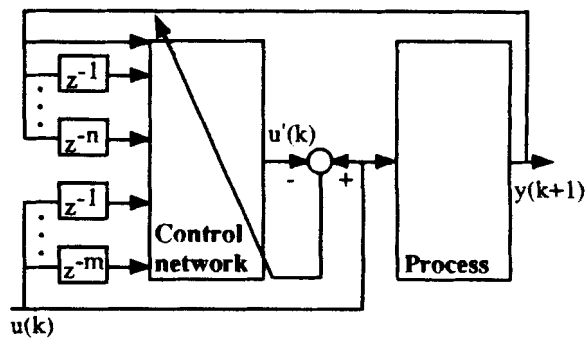

Figure 1. Direct inverse control using feedforward networks.

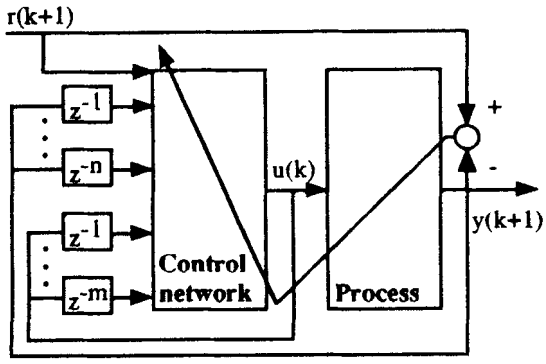

Figure 2. Direct adaptive control using feedforward networks.

of the error measure with respect to the process inputs, which assumes the knowledge of the Jacobian of the process.

Indirect adaptive control employs a process model, in this case a model network. The output error of the model is used in a standard backpropagation algorithm to pass the error back to the controller output. The adaptation mechanism is twofold since the neural process model is adapted to the true process output as target value. The indirect adaptive control scheme is given in Figure 3. Learning in both the direct and the indirect schemes are essentially performed on-line.

Two methods based on *reinforcement learning* are backpropagation through time and adaptive critics (Werbos, 1990; Sofge and White, 1990). In the *backpropagation through time* method one calculates the derivative of future utility or performance measure with respect to present actions using an explicit model of the environment. The *adaptive critics* method adapts a special "critic" network that estimates the future utility arising from present actions. It is essentially an approximation of dynamic programming methods.

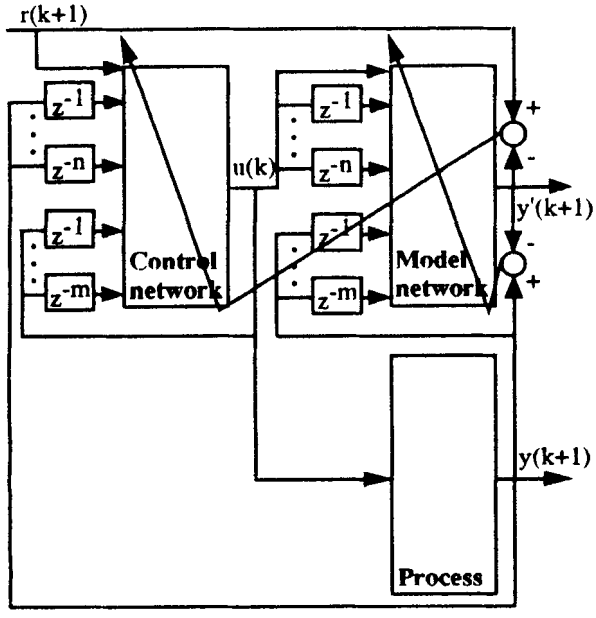

Figure 3. Indirect adaptive control using feedforward networks.

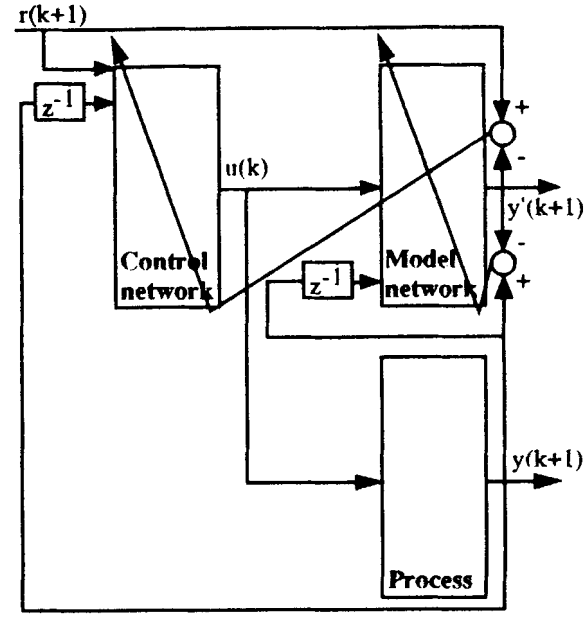

Figure 4. Indirect adaptive control using recurrent networks.

Application of Recurrent Networks in Control

When using feedforward networks to model process dynamics, a careful design of the history window of process inputs and outputs is necessary. The form of the model (basically an input–output model) requires the application of external feedback loops on the network.

Recurrent networks offer a solution to cope with these difficulties since they can develop an internal representation of time history through learning due to their feedback connections. Neural control schemes become significantly more intelligible when recurrent networks are used, since only the current process inputs and outputs are required instead of their time history. This scheme also reduces the number of input connections and results in a smaller computation cost in the controller operation. A major drawback is the higher computation time needed during the training phase. Figure 4 shows the application of recurrent networks in the indirect adaptive control scheme.

Using real-time recurrent learning in clustered networks

The training of fully connected recurrent control networks requires the knowledge of the controller output error, which is generally not available. In the indirect scheme the controller error is calculated by propagating the output error of the model network back to the model input. Although this can be easily done in feedforward networks using the standard backpropagation algorithm, it cannot be solved in a simple way in the case of recurrent networks. A direct error backpropagation algorithm cannot be derived for the RTRL algorithm. The method we suggest uses a combined network composed of the controller and the model clusters. The output node of the controller is identical to the corresponding input node of the model network. Such a clustered network is demonstrated in Figure 5 for the level control system pre-

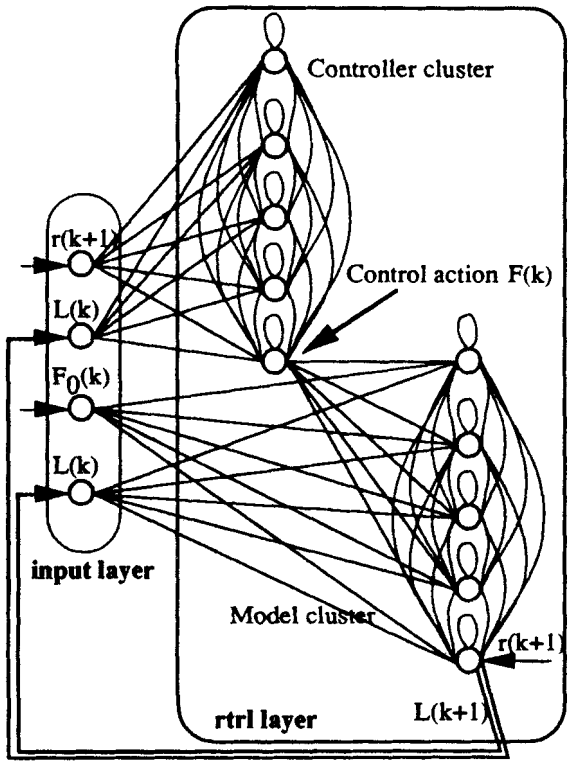

Figure 5. Construction of the clustered network for the level control system.

sented further on in two examples. Both the control and the model clusters are five-node, fully connected recurrent networks.

The simplest way to apply the RTRL algorithm is by giving zero weights to the nonexisting connections and by assigning a nonadaptable status to them, as to the weights in the model network. The algorithm can also be modified specifically for training the controller part of the cluster using only the existing connections.

For a system where one output of the control network is connected to one of the inputs of the model network (as shown in Figure 5), a simple form of the *modified RTRL algorithm* for training the controller part can be derived. Let U_1, I_1 denote the set of recurrent and input nodes of the controller network respectively, U_2, I_2 do the same for the model network, while $c \in U_1 \cap I_2$ stands for common node, that is, output node of the controller and input node of the model. Then Eq. 3 is reduced to the control network, since only the weights of the controller are adapted in the cluster network

$$\Delta w_{ij}(t) = \alpha \sum_{k \in U_1 \cup U_2} e_k(t) p_{ij}^k(t), \qquad i \in U_1 \quad \text{and} \quad j \in U_1 \cup I_1. \quad (5)$$

Equation 4 can be given separately for the two parts of the cluster

$$p_{ij}^k(t+1) = z_k(t+1)[1 - z_k(t+1)] \times \left[\sum_{l \in U_1} w_{kl} p_{ij}^l(t) + \delta_{ik} z_j(t) \right], \qquad k \in U_1 \quad (6)$$

$$p_{ij}^k(t+1) = z_k(t+1)[1 - z_k(t+1)] \times \left[\sum_{l \in U_2} w_{kl} p_{ij}^l(t) + w_{kc} p_{ij}^c(t) \right], \qquad k \in U_2, \quad (7)$$

where $i \in U_1$ and $j \in U_1 \cup I_1$ in both cases. Although this form requires less computation time than the standard algorithm, its structure becomes much more complicated for cluster networks, allowing more subnetworks and more connections.

The RTRL algorithm can now be applied to the cluster network without any difficulties. Of course the weights of the previously trained model subnetwork cannot be adapted during the controller training. Model adaptation (Catfolis, 1994) can take place in another cycle, even one with a different frequency.

Based on the application of the network cluster, the indirect adaptive control method using recurrent networks consists of the following steps:

1. Training of the recurrent model network.
2. Construction of the untrained, or "blank"—control network.
3. Composition of the network cluster—combining model and control network.
4. Adaptation of controller based on the model error with respect to the reference signal (setpoint)—training of the control network.
5. Adaptation of the model based on the model error with respect to the process output, if required.

The application of recurrent neural networks in the indirect scheme was studied by simulating two processes. The first example was a simple linear level control problem and was mainly used as a feasibility test. The second problem was a highly nonlinear bioreactor control problem. In this example we compared the results of the neurocontroller with a classic PID controller and analyzed the influence of dead time and noise. The cluster network method described earlier was applied using the standard RTRL algorithm. The model networks were trained to give a one-step-ahead prediction of the process dynamics. Then the on-line training of the control network in the inverse adaptive scheme was investigated. Step 5 of the control procedure (the adaptation of the model) was not used in these experiments.

Example 1: Level Control System

The first problem we studied was the level control in a tank in the presence of an external disturbance (input flow rate). The control variable is the output flow rate. The scheme of the system is shown in Figure 6. This simple linear example was selected to demonstrate the use of the inverse adaptive scheme with recurrent networks. The mathematical model of the tank is

$$A \frac{dL(t)}{dt} = F_0(t) - F(t), \quad (8)$$

where $L(t)$ is the level in the tank, $F_0(t)$ is the input flow rate, $F(t)$ is the output flow rate, and A is the cross section of the tank. A small simulation program applying the Euler method for the integration of the model was used to train the

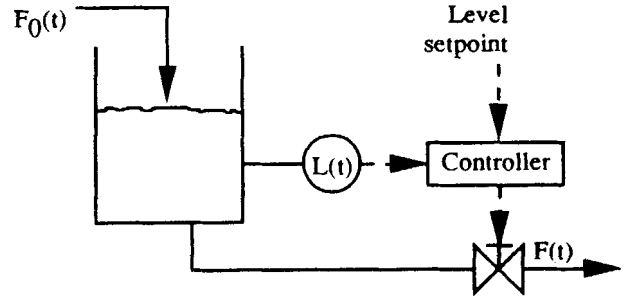

Figure 6. Level control system.

model network and then to train the controller network through the inverse adaptive scheme. An average error (0.01) of the output of the neural model was achieved by a 5-node network and giving random sequences on inputs $F_0(k)$, $F(k)$, and the setpoint $r(t)$. The actual configuration of the indirect adaptive control scheme is presented in Figure 7. Relatively slow convergence and sensitivity on the learning rate were observed during the controller training. A significant improvement in convergence was achieved by using the "clear impacts" method. Control networks of 4, 5, 6 and 8 neurons were trained, and even the smallest one provided a reasonable control performance.

The behavior of the controller with four neurons is shown in Figure 8. Responses on setpoint changes and on external disturbance $F_0(t)$ are demonstrated. The control offset is primarily due to the error of the model network.

Example 2: Control of a Bioreactor

The second problem, a bioreactor model, was suggested by Ungar (1990), to be used as a benchmark problem for neural controllers. It is a relatively simple problem having only a few variables. However it exhibits a difficult control problem due to its strongly nonlinear character. The system is a continu-

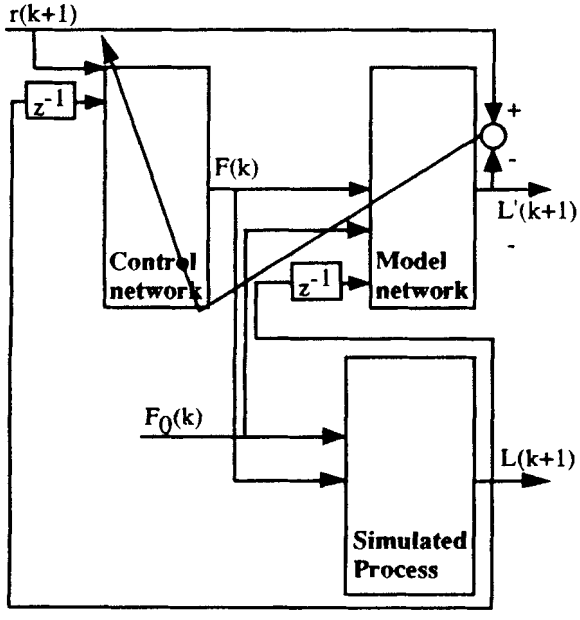

Figure 7. Training scheme for the level control system.

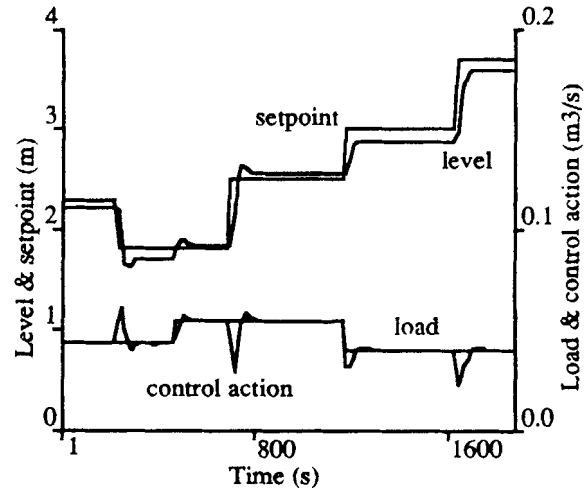

Figure 8. Operation of the controller for the level control system.

ous-flow stirred-tank reactor (CFSTR), with nutrient being fed into it. The control target is the cell mass yield. The flow rate through the tank is assumed to be constant and the volume of the tank is assumed to be unity. The scheme of the bioreactor is given in Figure 9.

The mathematical model (Agrawal et al., 1982) of the system gives an account of the concentration of both the cell mass and the substrate

$$\frac{dC_1(t)}{dt} = -C_1(t)w(t) + C_1(t)[1 - C_2(t)]e^{C_2(t)/\gamma} \quad (9)$$

$$\frac{dC_2(t)}{dt} = -C_2(t)w(t) + C_1(t)[1 - C_2(t)]e^{C_2(t)/\gamma} \times \frac{1+\beta}{1+\beta - C_2(t)}, \quad (10)$$

where $C_1(t)$ and $C_2(t)$ are, respectively, the dimensionless cell mass and substrate conversions, $w(t)$ is the substrate feed flow rate, β and γ are the rate coefficients.

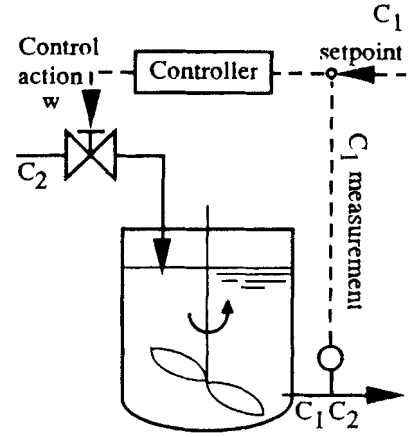

Figure 9. Bioreactor control system.

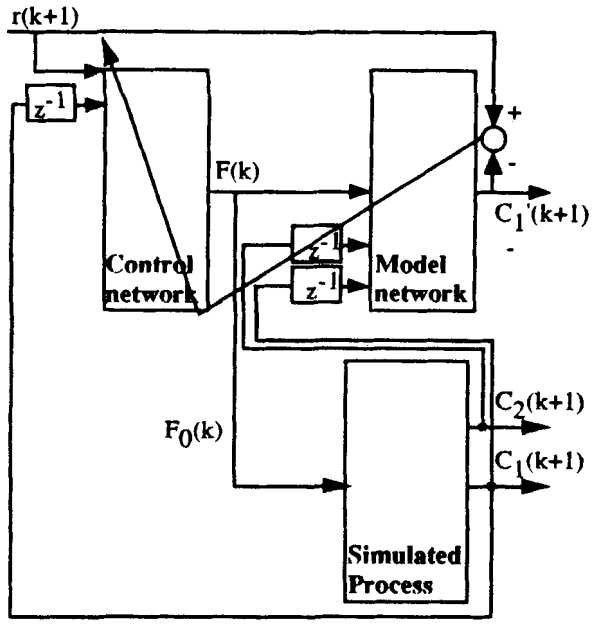

Figure 10. Training scheme for the bioreactor control system.

As Agrawal demonstrated, the necessary and sufficient conditions for the existence of limit cycles in the reactor behavior are

$$0.3 < \gamma < 0.5 \quad \text{and} \quad \beta < 1/(4 - 1/\gamma) - 0.5. \qquad (11)$$

In most of our experiments we used γ between 0.48 and 0.42 and β equal to 0.02, which ensures the existence of these limit cycles.

This process is extremely difficult to control with classic linear control systems. First, the equations are highly nonlinear and, due to the choice of the rate coefficients, exhibit limit cycles. Another problem is that the optimal behavior occurs in or near unstable regions. And finally, two different control actions (nutrient flow rate) can lead to the same cell mass output. This is called *multiplicity of the control action.*

Process modeling

The process model was used in the same way as in Example 1. However, only one of the system outputs, the cell mass conversion, $C_1(t)$, was represented by the model network, since the substrate conversion was not required for the controller training scheme (Figure 10).

The model was trained by changing the initial C_1 (random between 0.04 and 0.18) and C_2 (random between 0.79 and 0.97) every 2000 seconds and using a random control action (white noise between 0.0 and 2.0). Fully connected recurrent networks of 6, 8, 10 and 15 nodes were trained as one-step-ahead prediction models. When applying different learning rates, the best solution (8 neurons) gives an average error of 0.001 (after 300,000 training cycles) on the output with only a slight dependence on the number of neurons in the range studied. In Figure 11 the system output and the model output (C_1) are shown for different values of the control action w. We are able to incorporate the different behaviors of the system into one model (e.g., oscillations for $w < 1.2245$ or damped if $w > 1.2245$ for the given values of β and γ).

Classic PID vs. neurocontroller

The controller was trained by changing the initial C_1 and C_2 every 2000 seconds and the setpoint every 200 seconds. Several training experiments and manual adjustment of the learning rate were required to get a good control network. To test the performance of the neurocontroller we compared it with a classic proportional integral derivative (PID) controller. This controller was tuned by the Ziegler–Nichols technique.

As a test for the comparison between the PID and the neurocontroller we used a modified problem described by Ungar (Ungar, 1990). The initial concentrations in the reactor are $C_1 = 0.12$ and $C_2 = 0.88$. The initial setpoint is $C_1 = 0.06$ and the following setpoints are 0.08 and $t = 100$, 0.10 at $t = 200$, etc., until $C_1 = 0.16$ for t between 500 and 599. The results are shown in Table 1. We also tested how critical tuning was for both controllers. We tested the same problem with different values of γ. The importance of this rate coefficient is shown by Ungar, who explained that a change of 20% in γ

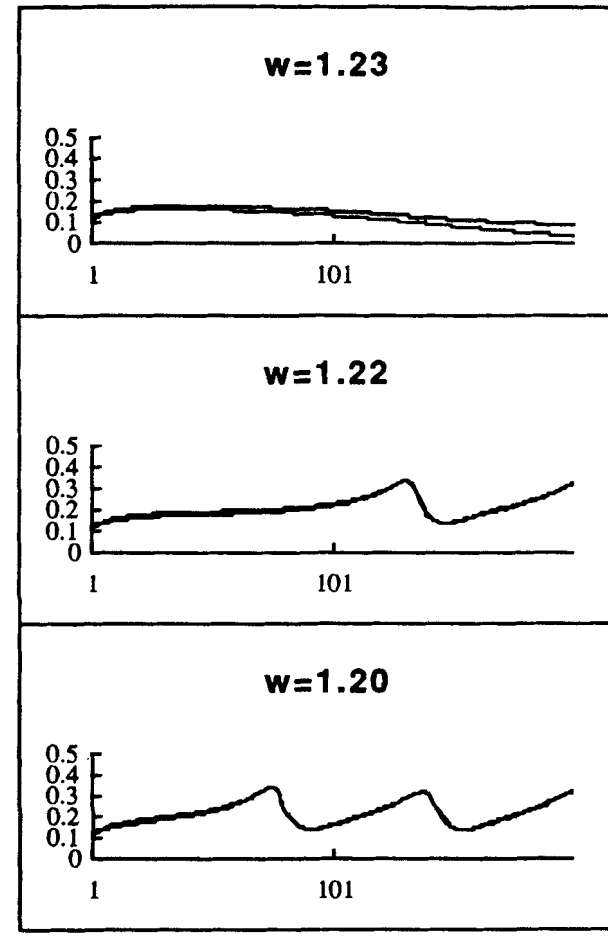

Figure 11. Highly nonlinear behavior of the bioreactor system for different values of the control action *w*.

Both system output and model are shown.

Table 1. Classic PID vs. Neurocontroller for the Bioreactor Example*

	PID Controller		Neurocontroller	
γ	Noise-Free	Noise	Noise-Free	Noise
0.48	0.015467	0.015863	0.011822	0.012258
0.46	0.015831	0.016458	0.012368	0.012654
0.44	0.020710	0.021162	0.013645	0.014005
0.42	0.023397	0.023466	0.016282	0.016882

*We tested both controllers for different values of γ and also with autocorrelated noise applied to the control action.

corresponds with a 50% error on the target cell mass. In all cases the PID controller was oscillatory for the last setpoint change. When γ equaled 0.44 instead of the original γ (0.48), the PID controller became unreliable due to the large offset. The behavior of the neurocontroller was stable even for a lower γ. In Figures 12 and 13 the last three setpoint changes and the corresponding control actions are shown for the PID controller. In Figures 14 and 15 the same data are shown for the neurocontroller.

The control offset of the neurocontroller can be attributed to the error of the neural process model, as in the level control problem. In both cases the application of step 5 of the present indirect adaptive control procedure—adaptation of the model network—can help solve the problem.

Neurocontroller and autocorrelated noise

Neural networks are known to be robust to noise (Rumelhart et al., 1986). To test if this is also the case for this neurocontrol architecture we applied autocorrelated noise—see Table 1—to the control action w. The differences between the system with noise and without noise for both the PID controller and the neurocontroller are shown in Table 2. The

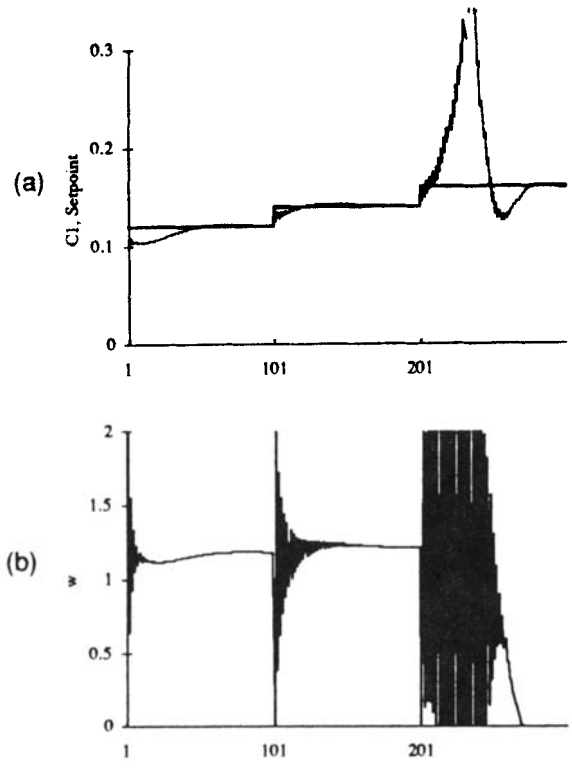

Figure 12. PID controller–rate coefficient γ is 0.48
(a) Setpoint and cell mass C_1. (b) control action w.

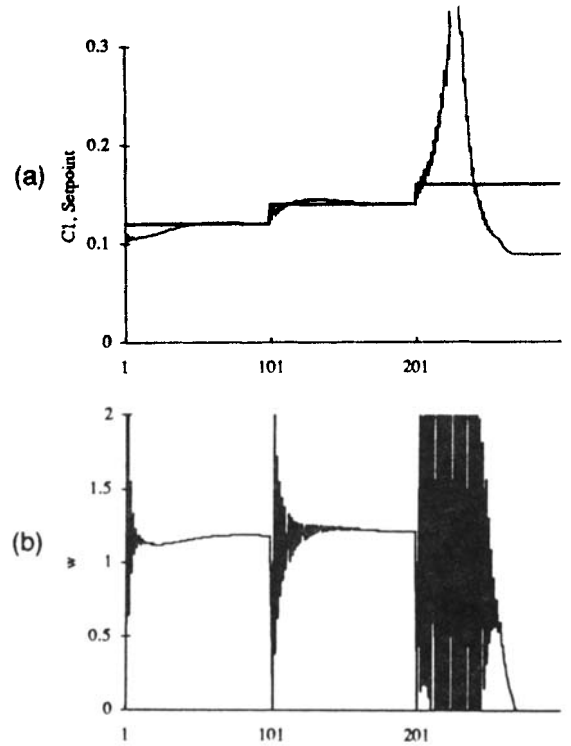

Figure 13. PID controller–rate coefficient γ is 0.44.
(a) Setpoint and cell mass C_1. (b) control action w.

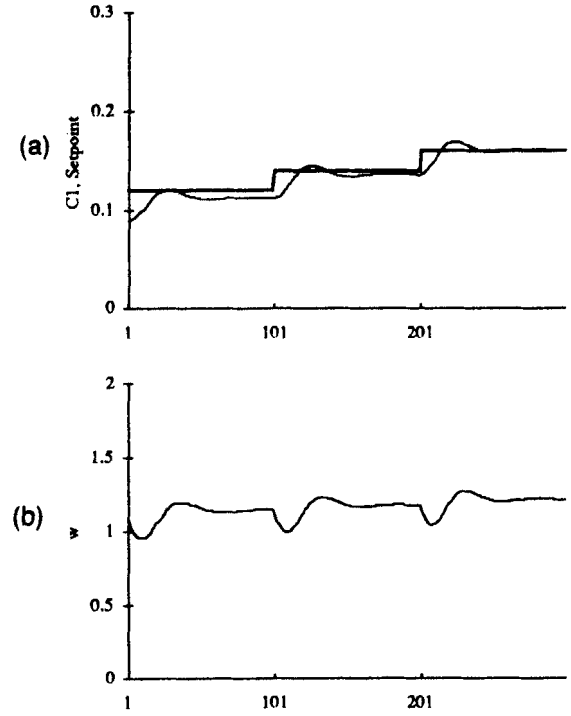

Figure 14. Neurocontroller–rate coefficient γ is 0.48
(a) Setpoint and cell mass C_1. (b) control action w.

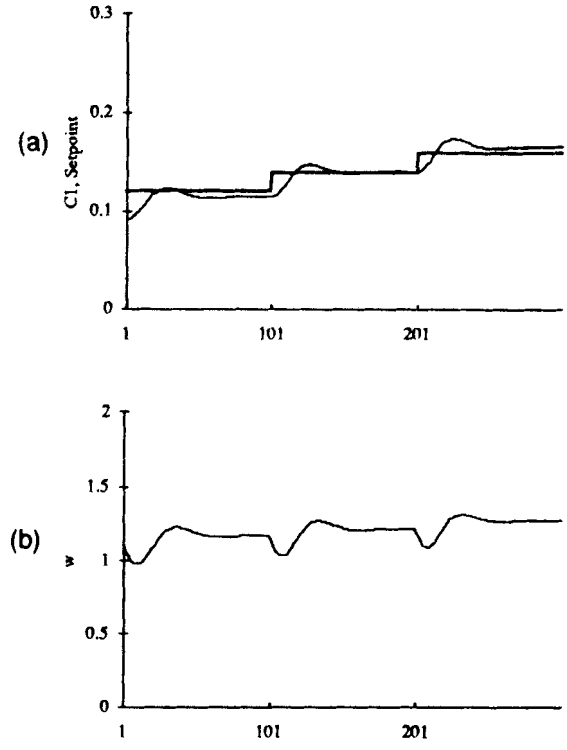

Figure 15. Neurocontroller–rate coefficient γ is 0.44.
(a) Setpoint and cell mass C_1. (b) control action w.

increase of the controller error is lower for the neurocontroller than for the PID controller. From this experiment we can conclude that the neurocontroller is as noise resistant as the PID controller.

Neurocontroller and dead time

One of the advantages of using recurrent neural networks is the possibility of mapping an unknown temporal relation between input and output. When using feedforward neural networks we need to know this relationship. Dead time introduces a phase lag between the system output and the input signal. When we look at a Nyquist plot, the diagram of the system without dead time becomes distorted into a spiral. This complicates control, since (additional) unstable regions are introduced. We tested our neurocontrol architecture on the same bioreactor system, but we introduced a—to the controller—unknown dead time of 3 seconds. The error of the neurocontroller with dead time (Table 3) is still of the same magnitude as the error of the PID controller without dead time.

Further Prospective Application of Cluster Networks

The concept of using network clusters can be generalized, that is, any number of already trained and "blank" subnetworks can be arbitrarily connected. Then the "blank" networks can be trained by applying the inputs and targets to the cluster network. This way, neural modeling can rely on system *decomposition into known and unknown parts* and also allows *neural-based hierarchical model building*. In practice, an amount of mainly structural information on the process is often available beforehand. The method outlined here allows the use of such knowledge by representing the known structural relations in relation to the appropriate subnetworks.

In control applications the standard and advanced control configurations can be transformed into neural controllers by building a cluster of recurrent subnetworks corresponding to the building blocks used in those schemes. The subnetworks themselves are not necessarily fully connected recurrent networks. Process delays, although easily mapped to a fully connected network, can also be represented by a simpler discrete time network containing only feedforward connections. A combination of recurrent and multilayer feedforward topologies may help to solve the problem of inherent delays in discrete time networks.

The preceding ideas require further study in order to find an efficient way to train the network cluster, to estimate dynamic properties arising from those of connected subnetworks, and to reveal other possible pitfalls.

Table 2. Autocorrelated Noise Applied to the Control Action of Controllers

Average of noise	−0.0028
Average of absolute value of noise	0.0283
Standard deviation of noise	0.0337
Correlation between noise (t) with noise ($t+1$)	0.0525

Conclusions

The method described in this article is based on a real-time recurrent learning algorithm and uses a network cluster constructed from the control network and the model network in the inverse adaptive control scheme.

We have demonstrated the advantages of using recurrent neural networks for the control of nonlinear systems. First, there is the advantage of using dynamic recurrent networks instead of static feedforward networks, the simpler control schemes. Second, there is the advantage of using a nonlinear controller instead of a linear PID controller.

Due to these advantages we were able to build a more robust controller. The performance of the neurocontroller is better than that of a PID controller. The presence of noise and dead time do not alter the controller performance in a significant way.

The control offset observed in both examples is primarily due to the error of the neural process model and is expected to be lowered by using an adaptive model of the system. A prediction of more than one timestep ahead by the system model, or the use of this controller together with a classic controller, could also improve the control offset.

Table 3. Error of the Neurocontroller with an Unknown Dead Time for Different Values of the Rate Coefficient γ

γ	Neurocontroller with Dead Time
0.48	0.015977
0.46	0.016142
0.44	0.016940
0.42	0.019014

Acknowledgments

This research was done at the Expert Systems Applications Development Group, headed by Prof. M. Rijckaert. This work was partially supported by the E.C. project "Cooperation in Science and Technology with Central and Eastern Europe" and by the Hungarian OTKA 2547 project.

Literature Cited

Agrawal, P., C. Lee, H. C. Lim, and D. Ramkrishna, "Theoretical Investigations of the Dynamic Behavior of Isothermal Continuous Stirred Tank Biological Reactors," *Chem. Eng. Sci.*, **37**, 453 (1982).

Barto, A. G., "Connectionist Learning for Control," *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, eds., MIT Press, Cambridge, MA (1990).

Bhat, N., and T. J. McAvoy, "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems," *Comput. Chem. Eng.*, **14**(4/5), 573 (1990).

Carpenter, G. A., and S. Grossberg, "ART2: Self-Organisation of Stable Category Recognition Codes for Analog Input Patterns," *Appl. Opt.*, **26**, 4919 (1987).

Catfolis, T., "A Method for Improving the Real-Time Recurrent Learning Algorithm," *Neural Networks*, **6**, 807 (1993a).

Catfolis, T., "Monitoring a Control System with a Hybrid Neural Network Architecture," *Proc. Int. Conf. Artificial Neural Networks*, Amsterdam, The Netherlands, p. 854 (1993b).

Catfolis, T., "Generating Adaptive Models of Dynamic Systems with Recurrent Neural Networks," *Proc. IEEE Int. Conf. Neural Networks*, Orlando, FL, **5**, p. 3238 (1994).

Chen, Q., and W. A. Weigand, "Dynamic Optimisation of Nonlinear Process by Combining Neural Net Model with UDMC," *AIChE J.*, **40**, 1488 (1994).

Cooper, D. J., L. Megan, and R. F. Hinde, "Disturbance Pattern Classification and Neuro-Adaptive Control," *IEEE Control Syst.*, 42 (Apr. 1992).

Freeman, J. A., and D. M. Skapura, *Neural Networks, Algorithms, Applications and Programming Techniques*, Addison-Wesley, Reading, MA (1991).

Hangos, K., "Application of Neural Networks in Chemical Process Control and Diagnostics," (in Hungarian) *Rep.* SCL-001-1992, MTA SZTAKI, Budapest, p. 42 (1992).

Hornik, K., M. Stinchcombe, and H. White, "Multi-layer Feedforward Networks are Universal Approximators," *Neural Networks*, **2**, 359 (1989).

Karna, K. N., and D. M. Breen, "An Artificial Neural Networks Tutorial: 1," *Neural Networks*, **1**(1), 1 (1989).

Levin, E., R. Gewirtzman, and G. F. Inbar, "Neural Network Architecture for Adaptive System Modeling and Control," *Neural Networks*, **4**, 185 (1991).

Meert, K., and T. Catfolis, "How Useful are Recurrent Neural Networks for Real-Time Calculation of the Average Chain Length of Polymethylmethacrylate?" *Process Control Qual.*, **6**, 195 (1994).

Meert, K., "Estimation of the Average Chain Length of Polymers with Neural Classifiers," *Proc. IEEE Int. Conf. Neural Networks* 1994, Orlando, FL, **6**, p. 3821 (1994).

Narendra, K. S., and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. Neural Networks*, **37**(1), 4 (1990).

Pham, D. T., and X. Liu, "Identification of Linear and Nonlinear Dynamic Systems Using Recurrent Neural Networks," *Artif. Intell. Eng.*, **8**, 67 (1993).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representation by Error Propagation," *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1: *Foundations*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, eds., MIT Press, Cambridge, MA (1986).

Sanderson, A. C., "Applications of Neural Networks in Robotics and Automation for Manufacturing," in *Neural Networks for Control* W. T. Miller, R. S. Sutton, and P. J. Werbos, eds. (1990).

Schmidhuber, J., "A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks," *Connect. Sci.*, **1**, 403 (1989).

Sofge, D. A., and D. A. White, "Neural Network Based Process Optimization and Control," *Proc. Conf. Decision and Control*, Honolulu, HI, p. 3270 (1990).

Tanomaru, J., and S. Omatu, "Process Control by On-Line Trained Neural Controllers," *IEEE Trans. Ind. Electron.*, **39**(6), 511 (1992).

Ungar, L. H., "A Bioreactor Benchmark for Adaptive Network-based Process Control," *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, eds., MIT Press, Cambridge, MA (1990).

Venkatasubramanian, V., R. Vaidyanathan, and Y. Yamatomo, "Process Fault Detection and Diagnosis Using Neural Networks: I. Steady-State Processes," *Comput. Chem. Eng.*, **14**(7), 699 (1990).

Wan, E. A., "Discrete Time Neural Networks," *J. Appl. Intell.*, **3**, 91 (1993).

Watanabe, K., I. Matsuura, M. Abe, M. Kubota, and D. M. Himmelblau, "Incipient Fault Diagnosis of Chemical Processes via Artificial Neural Networks," *AIChE J.*, **35**(11), 1803 (1989).

Werbos, P. J., "Neural Networks for Control and System Identification," *Proc. Conf. Decision and Control*, Tampa, FL, p. 260 (1989).

Werbos, P. J. "A Menu of Designs for Reinforcement Learning Over Time," *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, eds., MIT Press, Cambridge, MA (1990).

Williams, R. J., and D. Zipser, "Experimental Analysis of the Real-Time Recurrent Learning Algorithm," *Connec. Sci.*, **1**, 87 (1989).

Willis, M. J., G. A. Montague, and A. J. Morris, "Modelling of Industrial Processes Using Artificial Neural Networks," *Comput. Control Eng. J.*, **3**(3), 113 (1992).

Manuscript received Nov. 28, 1994, and revision received Apr. 17, 1995.