DESIGN OF AN INDUSTRIAL PROCESS CONTROLLER USING NEURAL NETWORKS

Xianzhong Cui and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

ABSTRACT

There are many problems in industrial process control systems due to a long system response delay, dead-zone and/or saturation of the actuator mechanisms, uncertainties in the system model and/or parameters, and process noise. To overcome these problems, an adaptive controller is designed using neural networks and tested extensively via simulations.

One of the key problems in designing such a controller is to develop an efficient training algorithm. Neural networks are usually trained using the output errors of the network, instead of using the output errors of the controlled plant. However, when a neural network is used to control a plant directly, the output errors of the network are unknown, since the desired control actions are unknown. In this paper, we propose a simple training algorithm for a class of nonlinear systems, which enables the neural network to be trained by the output errors of the controlled plant. The only *a priori* knowledge of the controlled plant is the direction of its output response. A detailed analysis of the algorithm is presented and the corresponding theorems are proved. Due to its simple structure and algorithm, and good performance, the proposed controller has high potential for handling difficult problems in industrial process control systems.

1. INTRODUCTION

The main issues to consider for the design of an industrial process control system are the negative effects such as a long system response delay, dead zone and/or saturation of actuator mechanisms, and nonlinear response of control valves. Process and measurement noises also degrade system performance. The dynamic property of a controlled plant may not be very complex, even though its detailed structure and parameters are usually unknown. However, when such a plant is put in operation, the control system is difficult to achieve high performance due mainly to the negative effects mentioned above. Contemporary industrial process control systems dominantly rely on PID-type controllers, though the hardware to implement control algorithms has been improved significantly in recent years. Despite the difficulty to achieve high control quality, the fine tuning of the controller's parameters is a tedious task, requiring experts with knowledge both in control theory and process dynamics. The reliability of such a system is also very important for operation security and efficiency. All of these call for the development of new controllers. The goal of this paper is to develop such a new controller using neural networks (NNs). Particularly, we shall focus on dealing with the nonlinearity of dead zone and saturation, and the negative effects of long response delays and process noises.

The potential of NNs for control applications lies in the following properties: (1) they could be used to approximate any continuous mapping, (2) they achieve this approximation through learning, (3) parallel processing and fault tolerance are easy to be accomplished with NNs.

One of the most popular NN architectures is a multilayer perceptron with the back propagation (BP) algorithm. It is proved that a four-layer (with two hidden layers) perceptron can be used to approximate any continuous function with the desired accuracy [3]. BP has been successfully used for pattern classification, though its original development placed more stress on control applications [13]. A controller is usually connected serially to the controlled plant under consideration. For a multilayer perceptron, the weights of the network need to be updated using the network's output error. For an NN-controller, the NN's output is the control command to the system. However, when the NN is serially connected to a controlled plant, the network's output error is unknown, since the desired control action is unknown. This implies that BP cannot be applied to control problems directly. Thus, one of the key problems in designing an NN-controller is to develop an efficient training algorithm.

Several related schemes have been proposed to solve this problem. One of them is training an NN to learn the system's inverse, and then the desired system output is achieved using the control input produced by the system's inverse. Certainly, this requires the system to be invertible. [2], [7], and [9] are such examples. In [2], the controlled plant was treated as an additional, unmodifiable layer, and the output error of the network was computed from the output error of the system. In [9], the system's output error was propagated back through the plant using its partial derivatives at an operating point. In [7], a set of actual system outputs are selected as training data and fed into the NN during its training period. Comparing the output of the NN with the desired system output, the network's output error is computed, which is then used to train the NN. After the NN becomes well-trained, the input of the NN is switched to the desired system output. Then, the NN acts as the inverse of the plant, and its output will drive the system to reach the desired value. However, in practice, even if the system is invertible, the inverse control scheme may be not acceptable. For example, if the system is in a non-minimum phase, then the resulting design is not internally stable. The invertibility of nonlinear systems was discussed in [4], and a sufficient-input criterion for designing an NN to learn a system's inverse was established. Other examples of NN-controllers are [1] [5], which used reference models to train the NN. Kraft and Miller designed controllers using a similar structure to CMAC (cerebellar model articulation controller) [6] [8]. Five dominant system architectures with NNs for control applications were summarized in [13] and [14], and the importance and applications of NNs to control and system identification were also addressed there.

However, most of the work mentioned above has the problems of complex training methods and system structures. Thus, we shall in this paper propose a simple algorithm based on the BP for a class of nonlinear systems typified by industrial process control applications. The proposed NN-controller is trained by using the system's output errors directly with little *a priori* knowledge of the controlled plant. In Section 2, the control problem using NNs is stated formally, and the basic structure of the proposed NN-controller is analyzed. The training algorithm is developed in Section 3, and the corresponding theorems are proved. Section 4 presents the procedures of designing the NN-controller, and addresses problems related to implementation. Section 5 summarizes the simulation results. A system model with a long response delay, nonlinearity of dead zone and saturation, and process noise is used to test the proposed NN-controller. The paper concludes with Section 6.

2. PROBLEM STATEMENT AND THE NN-CONTROLLER

Any controlled plant can be viewed as a mapping from control input to system output:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \qquad \mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t),$$

where $\mathbf{x} \in \mathbf{R}^{n_1}$, $\mathbf{y} \in \mathbf{R}^{n_2}$, and $\mathbf{u} \in \mathbf{R}^{N_2}$ are the state, system output and input, respectively. The controller is also a mapping from the system

The work reported in this paper was supported in part by the National Science Foundation under Grant No. DMC-8721492. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the NSF.

feedback and/or feedforward to control commands:

$$u = c(y, y_d, t), \tag{2-1}$$

where y_d is the desired system output. It is assumed that only the system output is measured.

We want to design an NN-controller to replace a conventional controller. In other words, the NN-controller is cascaded with the controlled plant as shown in Fig. 1, and trained to learn the mapping in Eq. (2-1). The control input $u_d(t)$ is required to produce the desired output $y_d(t)$. The *system-output error* and the *control-input error* are then defined, respectively, by

$$e_y(t) = y_d(t) - y(t), \tag{2-2}$$

$$e_u(t) = u_d(t) - u(t). \tag{2-3}$$

The control-input error $e_u(t)$ is also called the *network-output error*, since $u(t)$ is the output of the NN-controller. An NN is usually trained by minimizing the network-output error $e_u(t)$. However, if the NN controller is cascaded in series with the controlled plant, $e_u(t)$ is not known, since the desired control input $u_d(t)$ is unknown. So, the immediate problem in designing such an NN-controller is how to train the NN.

One of the well-developed NNs is a multilayer perceptron with BP [10] [12]. The basic structure of a three-layer perceptron is shown in Fig. 2. The BP algorithm is based on the gradient algorithm to minimize the network-output error, and derived from the special structure of the networks. Let θ_{1j} and θ_{2k} be the thresholds at the HIDDEN and the OUTPUT layer, respectively, where $1 \le j \le N_1$ and $1 \le k \le N_2$. Using the structure in Fig. 2, computation of the NN output and updating of the NN weights are summarized in the following five steps.

(1). **Compute the output of the HIDDEN layer:**

$$X_{1j}(t) = \frac{1}{1 + \exp\left[-O_{1j} - \theta_{1j}\right]}, \tag{2-4}$$

where $O_{1j} = \sum_{i=1}^{N} W_{ij} X_i(t), \quad 1 \le j \le N_1.$

(2). **Compute the output of the OUTPUT layer:**

$$X_{2k}(t) = \frac{1}{1 + \exp\left[-O_{2k} - \theta_{2k}\right]}, \tag{2-5}$$

where $O_{2k} = \sum_{j=1}^{N_1} W_{1jk} X_{1j}(t), \quad 1 \le k \le N_2.$

(3). **Update the weights from the HIDDEN to the OUTPUT layer:**

$$W_{1jk}(t+\delta t) = W_{1jk}(t) + \Delta W_{1jk}, \tag{2-6}$$

where $\Delta W_{1jk} = \eta_1 \, \delta_{1k} \, X_{1j}(t),$

$$\delta_{1k} = \left[X^d_{2k}(t) - X_{2k}(t)\right] X_{2k}(t) \left[1 - X_{2k}(t)\right]. \tag{2-7}$$

(4). **Update the weights from the INPUT to the HIDDEN layer:**

$$W_{ij}(t+\delta t) = W_{ij}(t) + \Delta W_{ij}, \tag{2-8}$$

where $\Delta W_{ij} = \eta \, \delta_j \, X_i(t),$

$$\delta_j = \left[\sum_{k=1}^{N_2} \delta_{1k} \, W_{1jk}\right] X_{1j}(t) \left[1 - X_{1j}(t)\right].$$

(5). **Update the thresholds: θ_{2k} and θ_{1j}.**

$$\theta_{2k}(t+\delta t) = \theta_{2k}(t) + \eta_{1\theta} \, \delta_{1k}, \tag{2-9}$$

$$\theta_{1j}(t+\delta t) = \theta_{1j}(t) + \eta_{\theta} \, \delta_j, \tag{2-10}$$

where η, η_1, η_θ, and $\eta_{1\theta} > 0$ are the gain factors.

In any control system design, it is desired to specify the system performance in terms of system-output errors, rather than the unknown network-output error $e_u(t)$. To design such a controller using NNs, we adopt the basic principle of BP, because of its ability of universal approximation and its convergent property based on the gradient algorithm [11]. The major obstacle to design such an NN-controller is to train the NN using system-output errors, rather than the network-output errors. The next section presents a solution to this problem.

3. TRAINING AN NN-CONTROLLER WITH SYSTEM-OUTPUT ERRORS

To derive the BP algorithm, the cost function of the network is defined as:

$$E_u(t) = \frac{1}{2} \sum_{k=1}^{N_2} \left[e_{uk}(t)\right]^2$$

where $e_{uk}(t) = u_{kd}(t) - u_k(t)$ is the network-output error at the k^{th} node of the OUTPUT layer. As mentioned earlier, $E_u(t)$ is not available since $u_{kd}(t)$ is unknown for all k. Let the l^{th} component of the system-output error be defined by

$$e_{yl}(t) = y_{ld}(t) - y_l(t), \qquad l = 1, \ldots\ldots, n. \tag{3-1}$$

Then, the cost function in terms of the system-output error is defined as:

$$\begin{aligned} E_y(t) &= \frac{1}{2} \sum_{l=1}^{n} \left[e_{yl}(t)\right]^2 = \frac{1}{2} \sum_{l=1}^{n} \left[y_{ld}(t) - y_l(t)\right]^2 \\ &= \frac{1}{2} \sum_{l=1}^{n} \left[G_l(u_d) - G_l(u)\right]^2, \end{aligned} \tag{3-2}$$

where $G_l(u)$ is the l^{th} component of the mapping $y = G(u)$ where $y \equiv \left[y_1, \ldots\ldots, y_n\right]^T$, and $u \equiv \left[u_1, \ldots\ldots, u_{N_2}\right]^T$. Eq. (3-2) is computable from the measurement of the system output. In other words, we know a function of the network-output error, though the detailed structure and parameters of the mapping G(.) may not be known. We want to train the NN by minimizing the cost function Eq. (3-2).

Using the gradient algorithm, the weights from the HIDDEN to the OUTPUT layer are modified by

$$W_{1jk}(t+\delta t) = W_{1jk}(t) + \Delta W_{1jk}, \tag{3-3}$$

and set $$\Delta W_{1jk} \approx -\frac{\partial E_y(t)}{\partial W_{1jk}(t)}. \tag{3-4}$$

Noting that $u_k(t) = X_{2k}(t)$ in the NN-controller [1], we get

$$\frac{\partial E_y(t)}{\partial W_{1jk}(t)} = -\sum_{l=1}^{n} \left[y_{ld}(t) - y_l(t)\right] \frac{\partial y_l(t)}{\partial u_k(t)} \frac{\partial X_{2k}(t)}{\partial O_{2k}} \frac{\partial O_{2k}}{\partial W_{1jk}(t)}. \tag{3-5}$$

Because $\frac{\partial X_{2k}(t)}{\partial O_{2k}} = X_{2k}(t) \left[1 - X_{2k}(t)\right]$, and $\frac{\partial O_{2k}}{\partial W_{1jk}(t)} = X_{1j}(t)$, Eq. (3-5) becomes $\frac{\partial E_y(t)}{\partial W_{1jk}(t)} =$

$$-\sum_{l=1}^{n} \left[y_{ld}(t) - y_l(t)\right] \frac{\partial y_l(t)}{\partial u_k(t)} X_{2k}(t) \left[1 - X_{2k}(t)\right] X_{1j}(t). \tag{3-6}$$

Substituting Eq. (3-6) into Eq. (3-4), one can get

$$\Delta W_{1jk}(t) = \eta_1' \, \delta_{1k}' \, X_{1j}(t), \tag{3-7}$$

where $$\delta_{1k}' = \sum_{l=1}^{n} \left[y_{ld}(t) - y_l(t)\right] \frac{\partial y_l(t)}{\partial u_k(t)} X_{2k}(t) \left[1 - X_{2k}(t)\right], \tag{3-8}$$

[1] In fact, $X_{2k}(t)$ is the scaled value of $u_k(t)$. At this stage, it is assumed that the scaling factor is one. The scaling problem will be discussed later.

$\eta_l^\gamma > 0$ is a constant. The only unknown in Eq. (3-8) is $\frac{\partial y_l(t)}{\partial u_k(t)}$, the $(l, k)^{th}$ component of the Jacobian matrix of the controlled plant.

Recall that the network-output error at the k^{th} node of the OUTPUT layer is defined by

$$e_{ak}(t) = u_{kd}(t) - u_k(t). \tag{3-9}$$

Referring to Eq. (3-8), the component of system-output error contributed by the k^{th} control input is defined by

$$e_{ak}(t) = \sum_{l=1}^{n} \left[y_{ld}(t) - y_l(t)\right] \frac{\partial y_l(t)}{\partial u_k(t)}. \tag{3-10}$$

To apply the gradient algorithm, we have the following theorem.

Theorem 1: Suppose the system response delay corresponding to the k^{th} control input is d. To train the NN using the system-output error and ensure the convergence of the training algorithm, the necessary and sufficient condition is

$$sign\left[e_{ak}(t)\right] = sign\left[e_{ak}(t-d)\right]. \tag{3-11}$$

Proof: In the gradient algorithm, the solution converges to a minimum of the cost function if and only if the search is made along the negative direction of the gradient of the cost function. BP is based on the gradient algorithm and listed in Eqs. (2-6) to (2-10). Because $u_{kd}(t) - u_k(t) = X_{2k}^d(t) - X_{2k}(t)$, Eq. (2-7) becomes

$$\delta_{1k} = e_{ak}(t)\, X_{2k}(t) \left[1 - X_{2k}(t)\right]. \tag{3-12}$$

Substituting Eq. (3-10) into Eq. (3-8), we get

$$\delta_{1k}^\gamma = e_{ak}(t)\, X_{2k}(t) \left[1 - X_{2k}(t)\right]. \tag{3-13}$$

Because both Eqs. (3-12) and (3-13) are derived by applying the gradient algorithm, to ensure the convergence of the training algorithm given in Eqs. (3-3) and (3-7), the necessary and sufficient condition is Eq. (3-11), when the system response delay is accounted for. □

The accurate value of $\left|\frac{\partial y_l(t)}{\partial u_k(t)}\right|$ is not important, because the step size can be adjusted by setting $\eta_k = \eta_l^\gamma \left|\frac{\partial y_l(t)}{\partial u_k(t)}\right|$. However, the sign of $\frac{\partial y_l(t)}{\partial u_k(t)}$ at each instant is not available and difficult to estimate on-line. So, a further simplification is necessary. In what follows, the case of SISO (single input, single output) systems is considered for simplicity.

Definition 1: If the system output monotonically increases (decreases) as the control input of a controlled plant increases, then the system is called *positive-responded* (*negative-responded*). Both positive-responded and negative-responded systems are called *monotone-responded*.

Definition 2: For an SISO system $y = G(u)$, if the system is positive-responded (negative-responded), then the *system direction* is defined by $direction(G) = 1$ ($direction(G) = -1$).

Definition 1 characterizes a class of systems. For example, a linear system in this class is cascaded with an element of pure response delay, dead zone and/or saturation. Fortunately, there are many industrial process control systems that possess the property of monotone-response. To train an NN-controller for such a class of systems, we have the following theorem.

Theorem 2 : For an SISO monotone-responded system, in order to train the NN-controller in Fig. 2 using system-output error, the weights on the arcs from the HIDDEN to the OUTPUT layer are updated by:

$$W_{1j1}(t+\delta t) = W_{1j1}(t) + \Delta W_{1j1}, \tag{3-14}$$

where $\Delta W_{1j1} = \eta_l^\gamma\, \delta_{11}^\gamma\, X_{1j}(t)$,

$$\delta_{11}^\gamma = \left[y_d(t) - y(t)\right] direction(G)\, X_{21}(t) \left[1 - X_{21}(t)\right].$$

Proof: For an SISO system, Eqs. (3-9) and (3-10) are simplified to $e_a(t) = u_d(t) - u(t)$ and $e_a(t) = \left[y_d(t) - y(t)\right] \frac{\partial y(t)}{\partial u(t)}$. From Eq. (3-11), we get the condition of convergence: $sign\left[e_a(t)\right] = sign\left[e_a(t-d)\right]$. If the system response delay is d, then for a positive-responded system

$$sign\left[u_d(t-d) - u(t-d)\right] = sign\left[y_d(t) - y(t)\right] \tag{3-15}$$

Similarly, for a negative-responded system, we have

$$sign\left[u_d(t-d) - u(t-d)\right] = -sign\left[y_d(t) - y(t)\right] \tag{3-16}$$

From Eqs. (3-15) and (3-16), we conclude that the condition for convergence is

$$sign\left[u_d(t-d) - u(t-d)\right] = sign\left[y_d(t) - y(t)\right] direction(G). \tag{3-17}$$

Eq. (3-17) then implies that the corresponding training algorithm be based on Eq. (3-14). □

4. DESIGN OF THE NN CONTROLLER

Figs. 1 and 2 show the basic structures of the system and the NN-controller, respectively. For an SISO system, there is one node at the OUTPUT layer, i.e., $N_2 = 1$. The inputs of the NN-controller are usually the system's desired and actual outputs, and tracking errors: $y_d(t)$, $y_d(t-\delta t)$, ..., $y_d(t-m_1\delta t)$, $y(t)$, $y(t-\delta t)$, ..., $y(t-m_2\delta t)$, $e_y(t)$, $e_y(t-\delta t)$, ..., $e_y(t-m_3\delta t)$, where m_1, m_2 and $m_3 > 0$ are integer constants, and $e_y(t) \equiv y_d(t) - y(t)$. The number of the HIDDEN nodes depends on the controlled plant under consideration. However, selection of a suitable number may require extensive experiments.

Based on Theorem 2, the formulas for updating the weights from the INPUT to the HIDDEN layer and the thresholds are derived using the same procedure given in Section 3. The computation of the NN-controller for an SISO system is then summarized as follows.

A. Compute the output of the HIDDEN layer: $X_{1j}(t)$.

$$X_{1j}(t) = \frac{1}{1 + \exp\left[-O_{1j} - \theta_{1j}\right]},$$

where $O_{1j} = \sum_{i=1}^{N} W_{ij}\, X_i(t)$, $j = 1, 2, ..., N_1$.

B. Compute the output of OUTPUT layer: $X_{21}(t)$.

$$X_{21}(t) = \frac{1}{1 + \exp\left[-O_{21} - \theta_{21}\right]},$$

where $O_2 = \sum_{j=1}^{N_1} W_{1j}\, X_{1j}(t)$.

C. Update the weights from HIDDEN to OUTPUT layer: $W_{1j1}(t)$

$$W_{1j1}(t+\delta t) = W_{1j1}(t) + \Delta W_{1j1},$$

where $\Delta W_{1j1} = \eta_l^\gamma\, \delta_{11}^\gamma\, X_{1j}(t)$,

$$\delta_{11}^\gamma = \left[y_d(t) - y(t)\right] direction(G)\, X_{21}(t) \left[1 - X_{21}(t)\right].$$

D. Update the weights from INPUT to HIDDEN layer: $W_{ij}(t)$

$$W_{ij}(t+\delta t) = W_{ij}(t) + \Delta W_{ij},$$

where $\Delta W_{ij} = \eta^y\ \delta_j^y\ X_i(t), \qquad \delta_j^y = \delta_{11}^y\ W_{1j1}\ X_{1j}(t)\left[1 - X_{1j}(t)\right]$

where $\eta^y > 0$ is the gain factor.

E. Update the thresholds: θ_{21} and θ_{1j}.

$$\theta_{21}(t+\delta t) = \theta_{21}(t) + \eta_{\theta}^y\ \delta_{11}^y, \qquad \theta_{1j}(t+\delta t) = \theta_{1j}(t) + \eta_{\theta}^y\ \delta_j^y,$$

where η_{θ}^y and $\eta_{\theta}^y > 0$ are the gain factors of the thresholds at the OUTPUT and the HIDDEN layer, respectively.

It is not difficult to extend this algorithm to the case of SIMO (single input, multiple outputs) systems. However, for the case of coupled MIMO (multiple inputs, multiple outputs) systems, we have not get a set of simple formulas, due mainly to the coupling effects.

Another problem in designing such an NN-controller is the choice of scaling factors. The sigmoid function in NN computation forces the NN outputs to be within the range of (0, 1), although the control inputs are limited by the range of actuators, (U_{min}, U_{max}). Therefore, the NN outputs should coincide with, or little narrower than, the range of the actuator's limits. The output of the NN-controller is then computed by $u(t) = X_{21}(t)\ (U_{max} - U_{min}) + U_{min}$.

Generally, an NN works in the mode of *train-first-then-operate*. In other words, an NN is put into operation only after it is "well-trained." By "well-trained," we mean that the weights of the NN need not be modified any more. However, for a time-varying system, it is meaningless to say that an NN is "well-trained", since the system always changes with time. Thus, not updating the weights for a time-varying system may result in the system going out of control. It is therefore necessary to always update the weights of the NN-controller, though the updating may not be done during every sampling interval.

5. SIMULATION RESULTS AND DISCUSSION

Many industrial process control systems are characterized by a linear system cascaded with a nonlinear element as a result of dead zone and actuator limits, and/or a pure time delay caused by transportation delay and system response delays. To test the capability of the proposed NN-controller, we conducted simulations while emphasizing the ability to overcome the negative effects of dead zone, saturation, long response delay, and process noise. The simulated system is a simplified temperature control system of an once-through boiler in a thermal power plant. The input is the variation of feedwater flow rate. The output is the variation of the temperature at the meddle point where water becomes steam. The system is represented by an ARMAX model:

$$A(z^{-1})\ y(k) = B(z^{-1})\ u(k-d) + C(z^{-1})\ \xi(k) \qquad (5\text{-}1)$$

where $A(z^{-1}) = 1 - 0.45181\ z^{-1} - 0.47546\ z^{-2}$,

$B(z^{-1}) = -0.04560\ z^{-1} - 0.00404\ z^{-2}$,

$C(z^{-1}) = 1 - 0.35740\ z^{-1} - 0.03392\ z^{-2}$,

$d = 18$ sampling intervals.

Here the sampling interval is chosen to be 8 seconds, $y(k)$ and $u(k)$ are the system output and control input at a discrete time k, respectively, and $\xi(k)$ is an uncorrelated random sequence with zero mean and variance R that represents the process noise.

There are six inputs at the INPUT layer of the NN-controller ($N = 6$), which are the desired system outputs and the output errors: $y_d(k)$, $y_d(k-1)$, $y_d(k-2)$, $y_d(k) - y(k)$, $y_d(k-1) - y(k-1)$, $y_d(k-2) - y(k-2)$. The number of the HIDDEN nodes is selected to be three, i.e., $N_1 = 3$. A nonlinear element of dead zone and saturation is cascaded with the system Eq. (5-1) to model an actuator, which is described by

$$u_a(t) = \begin{cases} 0 & \text{if } |u(t)| < dead\_zone \\ u(t) & \text{if } dead\_zone \le |u(t)| < U_{max} \\ U_{max} & \text{if } |u(t)| \ge U_{max}. \end{cases} \qquad (5\text{-}2)$$

The overall system structure is sketched in Fig. 3. The main simulation results are summarized below.

(1) When $dead\_zone = 5.0$, $U_{max} = 10.0$, and no process noise ($R = 0.0$), the result is plotted in Fig. 4. The initial weights of the NN are selected randomly, and the NN weights converge within 150 sampling intervals.

(2) When $dead\_zone = 7.0$, $U_{max} = 10.0$, and $R = 0.0$, Figs. 5 and 6 present the system response and the corresponding control input, respectively. Obviously, a large dead zone affects the system performance seriously, but the NN-controller still works well.

(3) When $dead\_zone = 5.0$, $U_{max} = 10.0$, and $R = 0.5$ to test the ability of noise rejection, the desired and actual system output responses are plotted in Fig. 7. The corresponding control input and the process noise are shown in Fig. 8, where $n(k) = \xi(k) - 0.35740\ \xi(k-1) - 0.03392\ \xi(k-2)$.

From the above simulation results, we conclude that the proposed NN-controller performs well for this class of nonlinear systems. In the NN-controller, the system-output error is computed from the measurements. As *a priori* knowledge, the system direction is easily obtained either from a step response experiment or the physical property of a controlled plant. To test the need of Eq. (3-14), *–direction(G)* is used in the training algorithm, which instantly results in the NN's divergence.

The remaining problems include:

- Choice of the number of the HIDDEN nodes: There is no systematic way to choose the number of the nodes at the HIDDEN layer(s) to approximate a given mapping. Fig. 9 shows the result using $N_1 = 6$, $dead\_zone = 5.0$, $U_{max} = 10.0$ and $R = 0.0$. Comparing with Fig. 4, one can see that adding more HIDDEN nodes may not improve the system performance. But intuitively, adding more nodes may improve the system's reliability.
- Training the NN-controller for an open-loop unstable system: Even for an open-loop stable system, large oscillation in the system output may not be acceptable for a controlled plant. This problem is important, especially at the beginning of training.
- Applying the NN-controller for other kinds of nonlinear systems, such as robotic manipulators.

6. CONCLUSION

Focusing on industrial process control systems, we designed and tested a new NN-controller. The negative effects of a long system response delay, nonlinear elements with dead zone and/or saturation, and process noises are the main obstacles in designing a controller and fine tuning its parameters. The proposed NN-controller can replace a conventional controller, and has overcome all of the problems mentioned above.

A training algorithm is derived based on BP, enabling the NN to be trained with system-output errors, rather than the network-output errors. In the BP algorithm, it is required to modify the weights by network-output error which is not known when a multilayer perceptron is applied directly to the controlled plant. Therefore, the proposed algorithm enhances the NN's ability to handle control applications. A detailed analysis of the algorithm is presented and the corresponding theorems are proved. The only *a priori* knowledge about the controlled plant is the direction of its response, which is usually easy to determine. Extensive simulations have been carried out and the results are quite promising. Good performance, a simple structure and algorithm, and the potential for fault tolerance make the proposed NN-controller attractive to industrial process control applications.

REFERENCES

[1] I. Bar-Kana and A. Guez, "Neuromorphic Adaptive Control", *Proc. of 1989 IEEE Int'l. Conf. on Decision and Control*, vol. 2, pp. 1739-1743.

[2] V. C. Chen and Y. H. Pao, "Learning Control with Neural Networks", *Proc. of 1989 IEEE Conf. on Robotics & Automation*, pp. 1448-1453.

[3] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, 1989, pp. 303-314.

[4] Y. L. Gu, "On Nonlinear system Invertibility and Learning approaches by Neural Networks", *Proc. of 1990 American Control Conference*, vol. 3, pp. 3013-3017.

[5] A. Guez and J. Selinsky, "A Neuromorphic Controller with a Human Teacher", *Proc. of 1988 Int'l. Conf. on Neural Networks*, vol. 2, pp. II595-II602.

[6] L. G. Kraft and D. P. Campagna, "A Comparison of CMAC Neural Network and Traditional Adaptive Control", *Proc. of 1989 American Control Conference*, vol. 1, pp. 884-889.

[7] E. Levin, R. Gewirtzman, and G. F. Inbar, "Neural Network Architecture for Adaptive System Modeling and Control", *Proc. of 1989 Int'l. Joint Conf. on Neural Networks*, vol. 2, pp. 311-316.

[8] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft, "Real-Time Dynamic Control of an Industrial Manipulator using a Neural-Network-Based Learning Controller", *IEEE Trans. on Robotics and Automation*, vol. 6, no. 1, pp. 1-9, Feb. 1990.

[9] D. Psaltis, A. Sideris, and A. A. Yamamura, "A Multilayered Neural network Controller", *IEEE Control System Magazine*, April 1988, pp. 17-21.

[10] D. E. Rumelhart, and J. L. McCelland, "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, MIT Press, 1986.

[11] J. Wang and B. Malakooti, "On Training of Artificial Neural Networks", *Proc. of 1989 Int'l. Joint Conf. on Neural Networks*, vol. 2, pp. 387-393

[12] P. J. Werbos, "Back propagation: Past and Future", *Proc. of 1988 Int'l. Conf. on Neural Networks*, vol. 1, pp. I343-I353.

[13] P. J. Werbos, "Back propagation and Neurocontrol: A Review and Prospectus", *Proc. of 1989 Int'l. Joint Conf. on Neural Networks*, vol. 1, pp. I209-I216.

[14] P. J. Werbos, "Neural Networks for Control and System Identification", *Proc. of 1989 IEEE Int'l. Conf. on Decision and Control*, vol. 1, pp. 260-265.

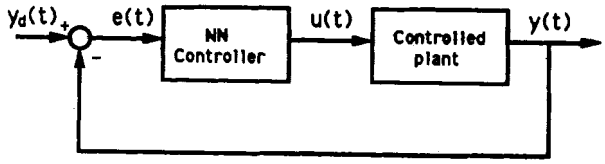

Fig. 1. A control system with an NN controller.

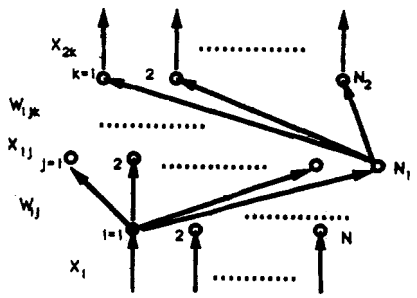

Fig. 2. Basic structure of a multilayer perceptron.

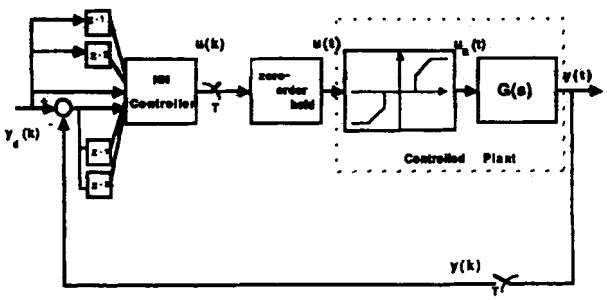

Fig. 3. Structure of the NN-based control system.

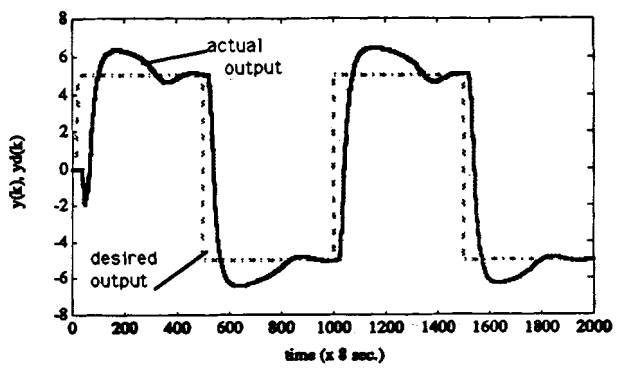

Fig. 4. System output response with dead_zone = 5.0.

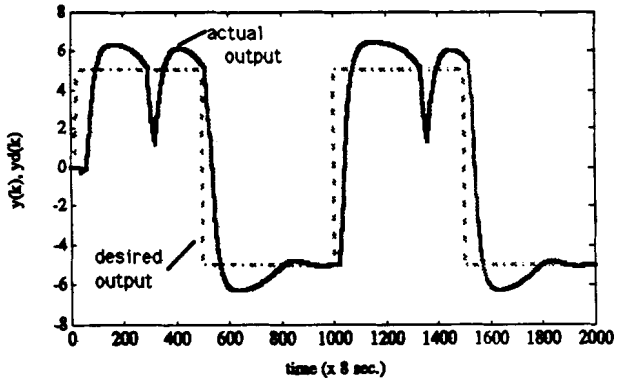

Fig. 5. System output response with dead_zone = 7.0.

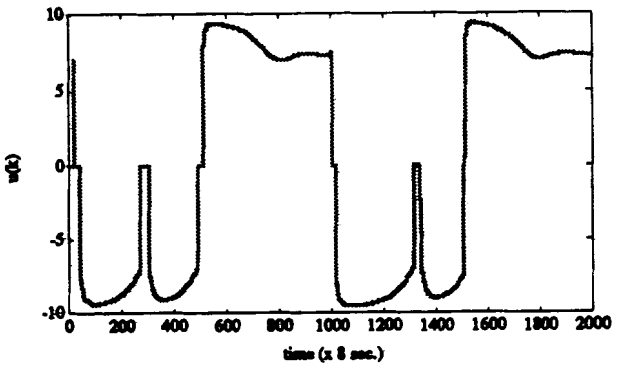

Fig. 6. System control input with dead_zone = 7.0.

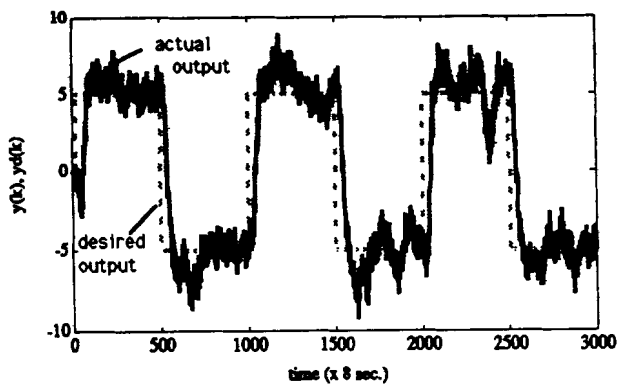

Fig. 7. System output response with process noise R = 0.5.

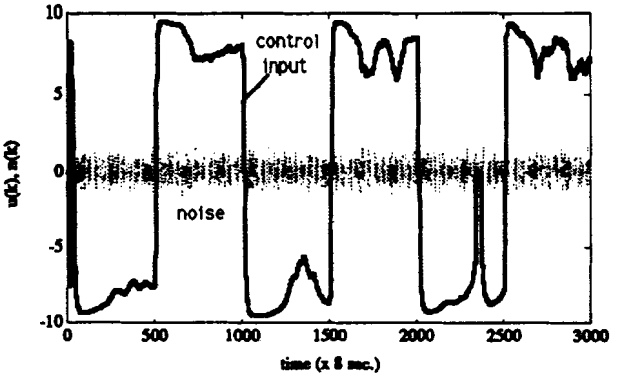

Fig. 8. System control input and process noise with R = 0.5.

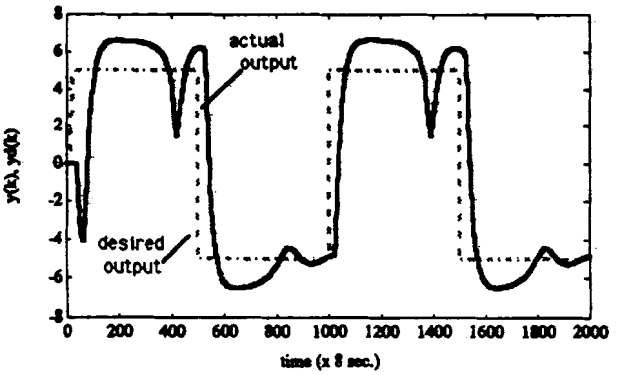

Fig. 9. System output response with six HIDDEN nodes.