Learning Rules for Neuro-Controller via Simultaneous Perturbation

Yutaka Maeda, *Member, IEEE,* and Rui J. P. De Figueiredo, *Fellow, IEEE*

***Abstract*—This paper describes learning rules using simultaneous perturbation for a neuro-controller that controls an unknown plant. When we apply a direct control scheme by a neural network, the neural network must learn an inverse system of the unknown plant. In the case, we must know the sensitivity function of the plant to use a kind of the gradient method as a learning rule of the neural network. On the other hand, the learning rules described here do not require information about the sensitivity function. Some numerical simulations of a two-link planar arm and a tracking problem for a nonlinear dynamic plant are shown.**

***Index Terms*— Dynamical systems, indirect inverse modeling, neural networks, neuro-controller, simultaneous perturbation, tracking problems.**

I. Introduction

RECENTLY, neural networks (NN's) have been well studied and widely used in many fields. Also, in the field of the control problem, NN's are used as a controller, an identifier, or an adjuster that gives some parameters in a conventional controller [1]–[13]. Usually, it is very difficult to treat a nonlinear objective function in control theory. However, NN's have an ability to map nonlinear functions on a compact set. Therefore, we anticipate that NN's will be able to handle a nonlinear plant well. Actually, many feasible realizations of neuro-controller have been reported [3], [6], [7], [9], [10], [12], [13].

When we apply NN's to a control problem, the direct control scheme constitutes a simple approach. In order to use a NN as a direct controller, the NN must be an inverse system of an objective plant, that is, the NN must learn an inverse system in so-called indirect inverse modeling. Then, the learning rule plays a practical role, because it will be related to an arrangement of overall system.

In the case of indirect inverse modeling, generally, we need a plant model or a sensitivity function of the plant to acquire the derivatives needed for learning like the backpropagation (BP) method [14], because the error function is usually defined not by an output of the NN but by that of the plant.

Manuscript received November 21,1995; revised August 12, 1996 and April 2, 1997. Y. Maeda was supported by the Research Institute of Industrial Technology of Kansai University.

Y. Maeda is with the Department of Electrical Engineering, Kansai University, Suita, Osaka, 564 Japan.

R. J. P. De Figueiredo is with the Department of Electrical and Computer Engineering, University of California at Irvine, Irvine, CA 92715 USA.

Publisher Item Identifier S 1045-9227(97)04905-9.

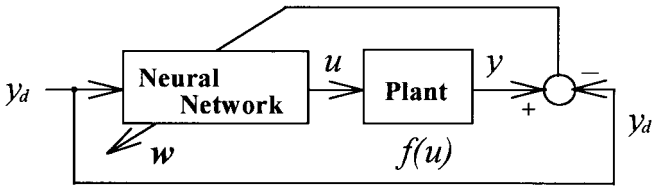


Fig. 1. Basic arrangements for indirect inverse modeling.

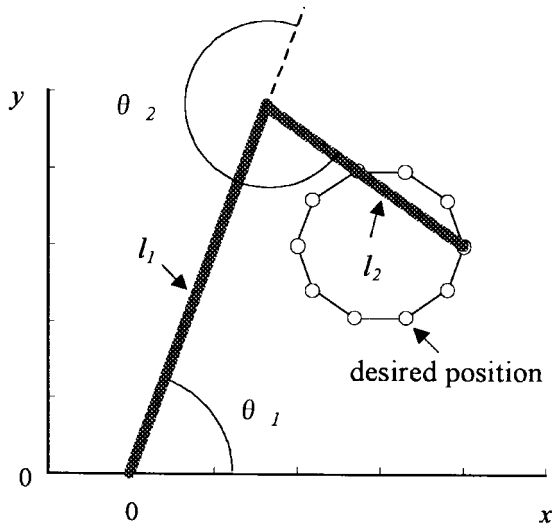


Fig. 2. Two-link planar arm.

These requirement can be bypassed by using an estimator of the derivative. The finite difference is a simple example. Jabri *et al.* pointed out the usefulness of such a learning rule [15]–[17]. However, that learning rule has a fault in that it does not take good advantage of parallel processing of NN's. In the present technique, we add a perturbation to all weights one by one and obtain corresponding values of an error function, in order to use a difference approximation. Therefore, the learning rule using the finite difference approximation requires n-times forward operation of the network to obtain modifying quantities for all weights, where n denotes the total number of weights of the network. Thus, if the NN is large, we cannot expect viability of the learning rule in the sense of the operating speed. Accordingly, we need a kind of simultaneous perturbation technique.

The basic idea of "simultaneous perturbation" was introduced by Spall [18]. He reported applications of simultaneous

```
do
for j:=1 to 10 do
        begin
        ●Input (x_dj  y_dj) to the NN, and the NN outputs.
        ●Convert the two outputs of the NN in interval [0  +1] into angles θ_1j and θ_2j in [0  2π].
        ●Input θ_1j and θ_2j to the system.
        ●Obtain (x_j  y_j).      (* position of the top of the arm *)
        ●Calculate the temporary squared error of the top of the arm.  (* (x_j − x_dj)^2 + (y_j − y_dj)^2 *)
        ●Accumulate the squared error.
        end;
●add the perturbation vector to the weights of the NN.
for j:=1 to 10 do
        begin
        ●Input (x_dj  y_dj) to the NN, and the NN outputs.
        ● Convert the two outputs of the NN in interval [0  +1] into angles θ_1j and θ_2j in [0  2π].
        ●Input θ_1j and θ_2j to the system.
        ●Obtain (x̂_j  ŷ_j).      (* position under perturbation *)
        ●Calculate the temporary squared error of the top of the arm.  (* (x̂_j − x_dj)^2 + (ŷ_j − y_dj)^2 *)
        ●Accumulate the squared error under perturbation.
        end;
●Calculate modifying quantities for all weights and thresholds by using (7).
●Update weights of the NN.
until{ the error is enough small }
```

Fig. 3. Implementation of the learning rule 1.

perturbation to the optimization problem, stochastic approximation, adaptive control, and neural-network controller and its applications [19]–[22]. He points out that simultaneous perturbation type of methods are superior to the conventional optimization techniques. Independently, the authors also have reported a comparison between the simultaneous perturbation type of learning rule of NN's, the simple finite difference type of learning rule and the ordinary BP method [23]. Simultaneous perturbation technique as a learning rule for neuro-controllers is promising in the control problem as well [21], [22], [24]. Alespector *et al.* and Cauwenberghs also proposed a parallel gradient descent method and stochastic error-descent algorithm, respectively, which are identical to ours [25], [26]. Moreover, Fujita proposed trial-and-error correlation learning rule of NN's. His learning rules include these types of learning rule in a broad sense [27].

In this paper, we describe two learning rules via the simultaneous perturbation for the direct control scheme. By using the learning rules provided here, the NN as neuro-controller can learn an inverse of a plant without any information about the sensitivity function of the objective plant.

II. Neuro-Controller and Learning Rule Using Simultaneous Perturbation

There are many schemes to utilize NN's in the field of control problem [1], [2]. If an NN can learn an inverse system of a plant, the NN can be used as a controller of the plant as a direct controller. Fig. 1 shows the basic arrangement for neuro-controller to learn the inverse system of the plant. The arrangement shown in Fig. 1 is called indirect inverse modeling. In this arrangement, it is possible for the NN to work as a controller even during so-called learning process.

In this paper, we consider the direct control scheme by an NN using the arrangement shown in Fig. 1. This is the simplest arrangement to control the unknown plant and simultaneously, to learn an inverse system of the plant. However, in the arrangement, the error are obtained through the unknown plant. Therefore, we need information about the plant. Next, we explain this point.

For convenience, in this discussion, we assume a case that an input of the plant u and an output of the plant y are both scalar and a characteristic of the plant $y = f(u)$ is static. However, similar discussion is applicable to multiinput–multioutput dynamic plant.

Now, $w = (w^1, \cdots, w^n)^T$ denotes a weight vector of the NN including thresholds. Superscript T is transpose of a vector. Ordinarily, an error function $J(w)$ is defined as the difference between an output of a plant y and a desired output y_d for the plant in the arrangement shown in Fig. 1. We must adjust the weights in the NN so that the NN must produce an input of the plant that decreases the error. When we use usual gradient method as a learning rule in this arrangement, in order to update the weights, we need a gradient of the error

do
for j:=1 **to** 10 **do**
begin
● Input $(x_{dj}\ y_{dj})$ to the NN, and the NN outputs.
● Convert the two outputs of the NN in interval [0 +1] into angles θ_{1j} and θ_{2j} in [0 2π].
● Input θ_{1j} and θ_{2j} to the system.
● Obtain $(x_j\ y_j)$. (* position of the top of the arm *)
● Calculate the error of the top of the arm. (* $(x_j - x_{dj})$ and $(y_j - y_{dj})$ *)
● add the perturbation vector to the weights of the NN.
● Input $(x_{dj}\ y_{dj})$ to the NN, and the NN outputs.
● Convert the two outputs of the NN in interval [0 +1] into angles θ_{1j} and θ_{2j} in [0 2π].
● Input θ_{1j} and θ_{2j} to the system.
● Obtain $(\hat{x}_j\ \hat{y}_j)$. (* position under perturbation *)
● Calculate a difference of the position. (* $(\hat{x}_j - x_j)$ and $(\hat{y}_j - y_j)$ *)
● Calculate modifying quantities for all weights and thresholds by using (14).
● Accumulate the modifying quantities for all weights.
end;
● Update the weights of the NN.
until{ the error is enough small }

Fig. 4. Implementation of the learning rule 2.

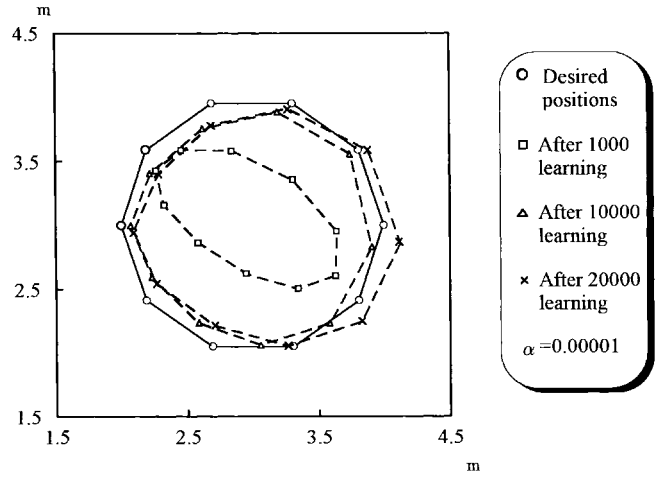


Fig. 5. Simulation results of two-link arm by the learning rule 1.

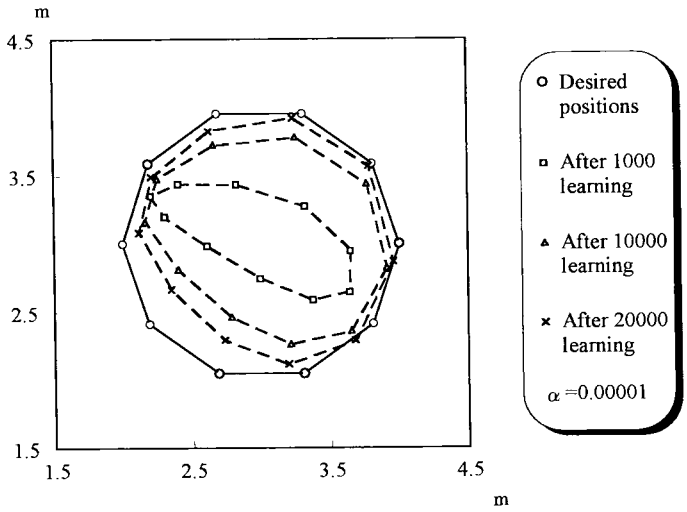


Fig. 6. Simulation result of two-link arm by the learning rule 2.

function. Namely, we must know the following quantity to utilize the gradient method:

$$\frac{\partial J(w)}{\partial w}.$$

Define an error function as follows:

$$J(w) = \frac{1}{2}(y - y_d)^2. \tag{1}$$

Thus, we have

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J(w)}{\partial y}\frac{\partial y}{\partial w} = e\frac{\partial f(u)}{\partial w} = e\frac{\partial f(u)}{\partial u}\frac{\partial u}{\partial w} \tag{2}$$

where $e = (y - y_d)$.

Then, we can measure e that is an error between the actual output of the plant and the corresponding desired output. Moreover, we can calculate $\partial u/\partial w$ as same as the error BP.

At the same time, we must know the sensitivity function $\partial f(u)/\partial u$ or, at least, the sign of the sensitivity function of the plant. However, if the plant is completely unknown, we can neither obtain the quantity $\partial f(u(w))/\partial u$ nor the sign of this quantity. Therefore, it is difficult to use gradient type of learning rule in this arrangement. This is one of difficulties that arises in the indirect inverse modeling scheme.

On the other hand, the difference approximation is a well-known approach to obtain a derivative of a function. We can utilize this kind of technique to our problem.

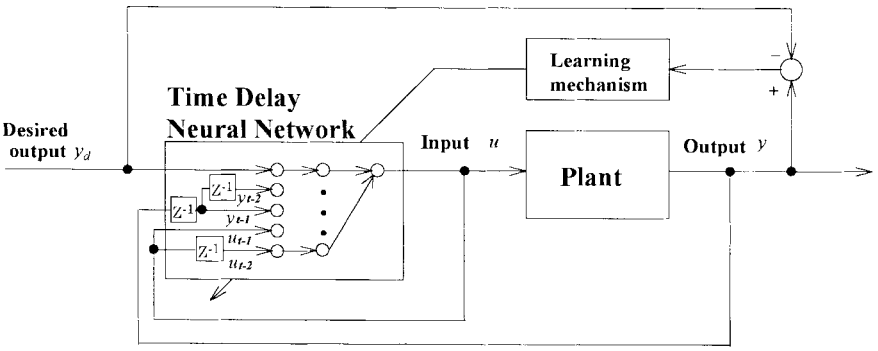


Fig. 7. Configuration. A time-delay NN is used to handle the dynamic plant.

```
do
for p:=1 to 40 do
        begin
        ●Input ydp to the NN, and the NN outputs.
        ●Convert the output of the NN in interval [0   +1] into [-2.5   +2.5].
        ●Input this to the plant.
        ●Obtain yp.      (* output of the plant *)
        ●Calculate the temporary squared error of the plant. (* (yp − ydp)^2 *)
        ●Accumulate the squared error.
        end;
●add the perturbation vector to the weights of the NN.
for p:=1 to 40 do
        begin
        ●Input ydp to the NN and the NN outputs.
        ●Convert the output of the NN in interval [0   +1] into [-2.5   +2.5].
        ●Input this to the system.
        ●Obtain ŷp.   (* output of the plant under perturbation *)
        ●Calculate the temporary squared error of the plant.(* (ŷp − ydp)^2 *)
        ●Accumulate the squared error under perturbation.
        end;
●Calculate modifying quantities for all weights and thresholds by using (7).
●Update weights of the NN.
until{ the error is enough small }
```

Fig. 8. Implementation of the learning rule 1.

We add a small perturbation c to the ith weight of the weight vector, that is, we define w^i as follows:

$$w^i = (w^1, \cdots, w^i + c, \cdots, w^n)^T. \tag{3}$$

First, we can employ the difference approximation to obtain $\partial J(w)/\partial w$ overall. That is, by using the following quantity, we can update the weights of the NN:

$$\frac{\partial J(w)}{\partial w^i} \approx \frac{J(w^i) - J(w)}{c}. \tag{4}$$

On the other hand, since u, which is the output of the NN, is a function of the weight vector, we obtain

$$\frac{\partial y}{\partial w^i} \approx \frac{f(u(w^i)) - f(u(w))}{c}. \tag{5}$$

Also by using (5), we can apply the gradient method to the indirect inverse modeling scheme.

As a result, introducing the idea of the difference approximation provides the possibility of using the indirect inverse modeling scheme more easily. There is no need to know the sensitivity function of the unknown plant. One of the authors investigated a feasibility of this type of learning rule in the control problem [17].

However, this simple idea described above needs much more forward operations of the NN and the plant. Namely, we must know $J(w^i)$ for all $i = 1, \cdots, n$ to obtain the modifying quantity for all weights. Therefore, we cannot expect parallel operation of modifying the weights. If the NN is large, this approach is not promising.

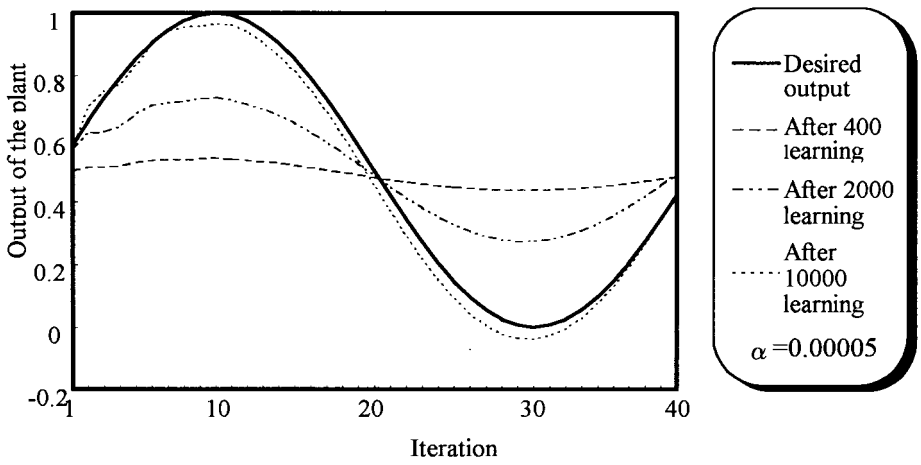


Fig. 9. Simulation results using a time-delay NN for sinusoidal desired output by the learning rule 1.

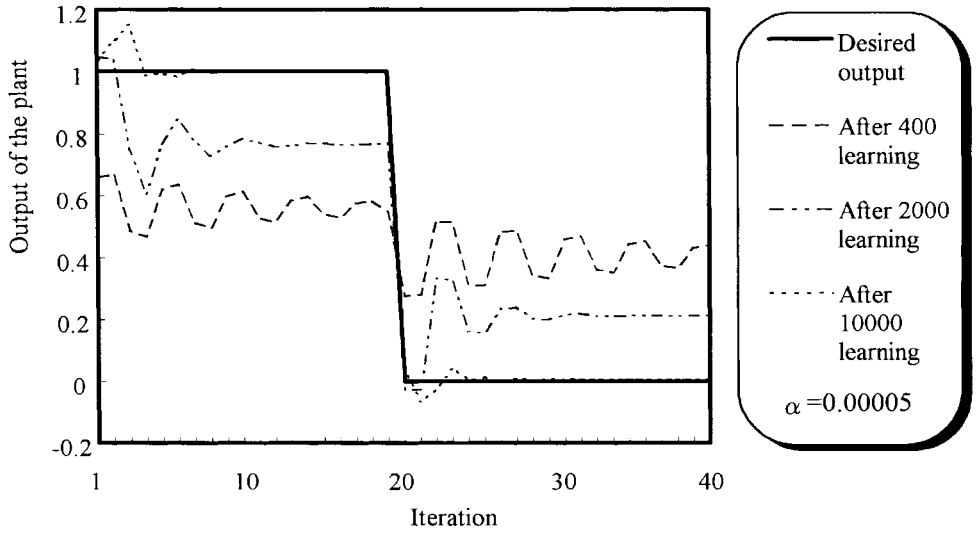


Fig. 10. Simulation results using a time-delay NN for square desired output by the learning rule 1.

In order to overcome this difficulty, we introduce an idea of a simultaneous perturbation.

First of all, we define the following perturbation vector $\mathbf{c}_t$ that gives small disturbances to all weights

$$c_t = \left(c_t^1, \cdots, c_t^n\right)^T \tag{6}$$

where the subscript t denotes an iteration.

The perturbation vector $\mathbf{c}_t$ has the following properties.

[A1] c_t^i is a uniformly random number in an interval $[-c_{\max}, c_{\max}]$ except an interval $[-c_{\min}, c_{\min}]$ and is independent with respect to time t for $i = 1, \cdots, n$.

[A2] $E(c_t) = 0$.

[A3]

$$E\left(c_t^i c_t^j\right) = \begin{cases} 0 & \text{if } i \neq j \\ \sigma^2 & \text{if } i = j. \end{cases}$$

$E(\cdot)$ denotes expectation. σ^2 is a variance of the perturbation c_t^i.

Learning Rule 1: First, we estimate $\partial J/\partial w$ overall like (4), then the ith component of the modifying vector of the weights $\Delta \mathbf{w}_t$ is defined as follows:

$$\Delta w_t^i = \frac{J\left(u(w_t + c_t)\right) - J\left(u(w_t)\right)}{c_t^i}. \tag{7}$$

Weights of the network are updated in the following manner:

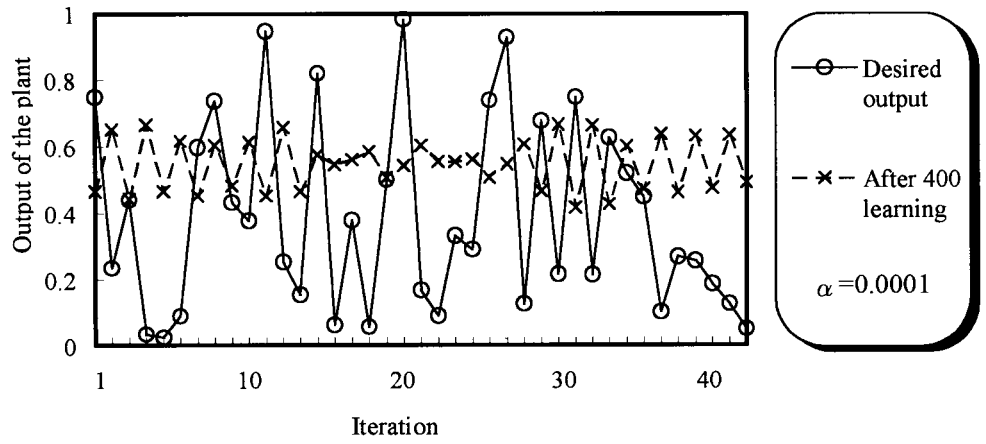
$$w_{t+1} = w_t - \alpha \Delta w_t$$

where α is a positive learning coefficient.

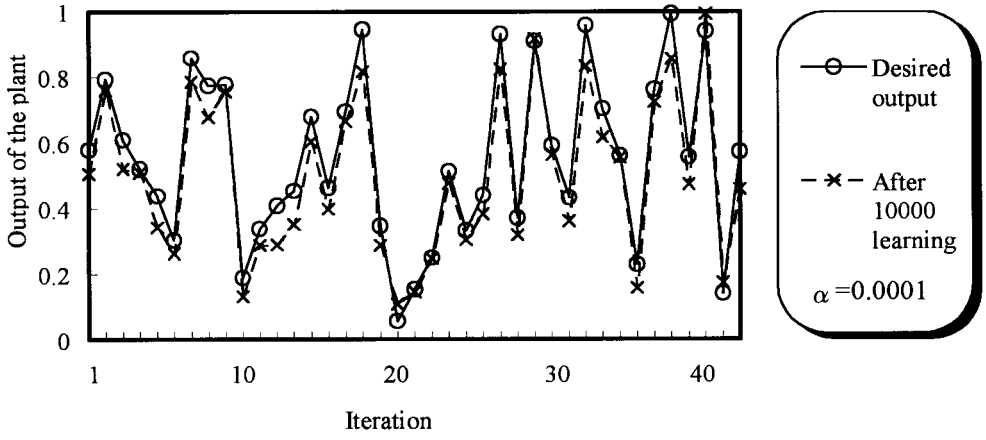
Learning Rule 2: Next, we estimate the first differential coefficient of the unknown plant with respect to the weights of the NN, then we can obtain the following modifying quantity:

$$\Delta w_t^i = e_t \frac{f(u(w_t + c_t)) - f(u(w_t))}{c_t^i} \tag{8}$$

where $e_t = (y_t - y_{dt})$. Since the error e_t can be easily measured, in this learning rule, only $\partial f/\partial w$ is estimated by means of the simultaneous perturbation.



(a)



(b)

Fig. 11. Simulation results using a time-delay NN for random desired output by the learning rule 1.

do
begin

- Input y_{dp} to the NN, and the NN outputs.
- Convert the output of the NN in interval [0 +1] into [-2.5 +2.5].
- Input this to the plant.
- Obtain y_p. (* output of the plant *)
- Calculate the error of the plant. (* $(y_p - y_{dp})$ *)

- add perturbation vector to weights of the NN.
  - Input y_{dp} to the NN, and the NN outputs.
  - Convert the output of the NN in interval [0 +1] into [-2.5 +2.5].
  - Input this to the plant.
  - Obtain $\hat{y}_p$. (* output of the plant under perturbation *)
  - Calculate a difference of the plant output. (* $(\hat{y}_p - y_{dp})$ *)
- Calculate modifying quantities for all weights and thresholds by using (8).
- Update weights of the NN.

end;
until{ the error is enough small }

Fig. 12. Implementation of the learning rule 2.

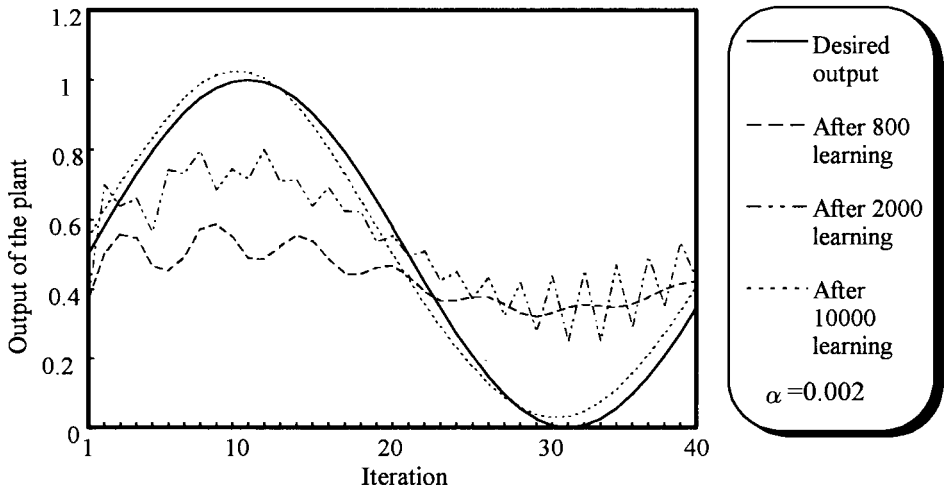


Fig. 13. Simulation results using a time-delay NN for sinusoidal desired output by the learning rule 2.

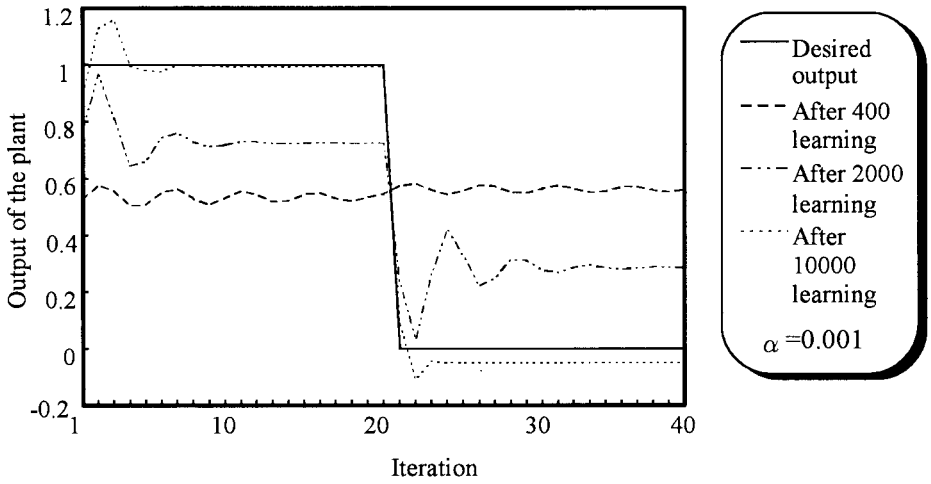


Fig. 14. Simulation results using a time-delay NN for square desired output by the learning rule 2.

Let us consider the quantity described in (7). Expanding $J(w_t + c_t)$ at w_t, there exists certain point w_{Sl} such that

$$\Delta w_t^i = \frac{J(w_t + c_t) - J(w_t)}{c_t^i} = \frac{c_t^T}{c_t^i}\frac{\partial J(w_t)}{\partial w} + \frac{1}{2c_t^i} c_t^T \frac{\partial^2 J(w_{\mathrm{Sl}})}{\partial w^2}\mathbf{c}_t. \tag{9}$$

The point $\mathbf{w}_{\mathrm{Sl}}$ exists in a hyper-cube with a diagonal line connecting w_t and $w_t + \mathbf{c}_t$. Taking expectation of the above equation, from the assumptions [A1]–[A3] of properties of the perturbation, we have the following relation:

$$E(\Delta w_t^i) = \frac{\partial J(w_t)}{\partial w^i} + E\left\{\frac{1}{2c_t^i} c_t^T \frac{\partial^2 J w_{\mathrm{Sl}}}{\partial w^2} c_t\right\}. \tag{10}$$

This means that if the perturbation $\mathbf{c}_t$ is sufficiently small, the second term of the right-hand side of (10) is small. Then the right side of (10) is nearly equal to the derivative of the error function $\partial J(\mathbf{w}_t)/\partial w^i$ in the sense of the expected value. Therefore, we can find the learning rule (7) a type of stochastic gradient methods. If we would like to prove the convergence of this type of algorithm, we need conditions for the perturbation, the error function, and so on. Moreover, we have to a reducing learning coefficient α into account, especially, under a stochastic environment. Detailed strict convergence conditions are described by Spall [20]. We can obtain a similar result for (8) as well.

Basically, these learning rules are concrete implementations of simultaneous perturbation technique by Maeda [17], Spall [20], and Alespector *et al.* [25] for neuro-controllers.

III. Simulation Results

In this chapter, we examine a viability of these learning rules by numerical simulations. NN's used in these simulations are three-layered feedforward networks with ten neurons in their hidden layers. The weights and the thresholds of the NN's are randomly initialized in the interval $[-1\ +1]$. The perturbations

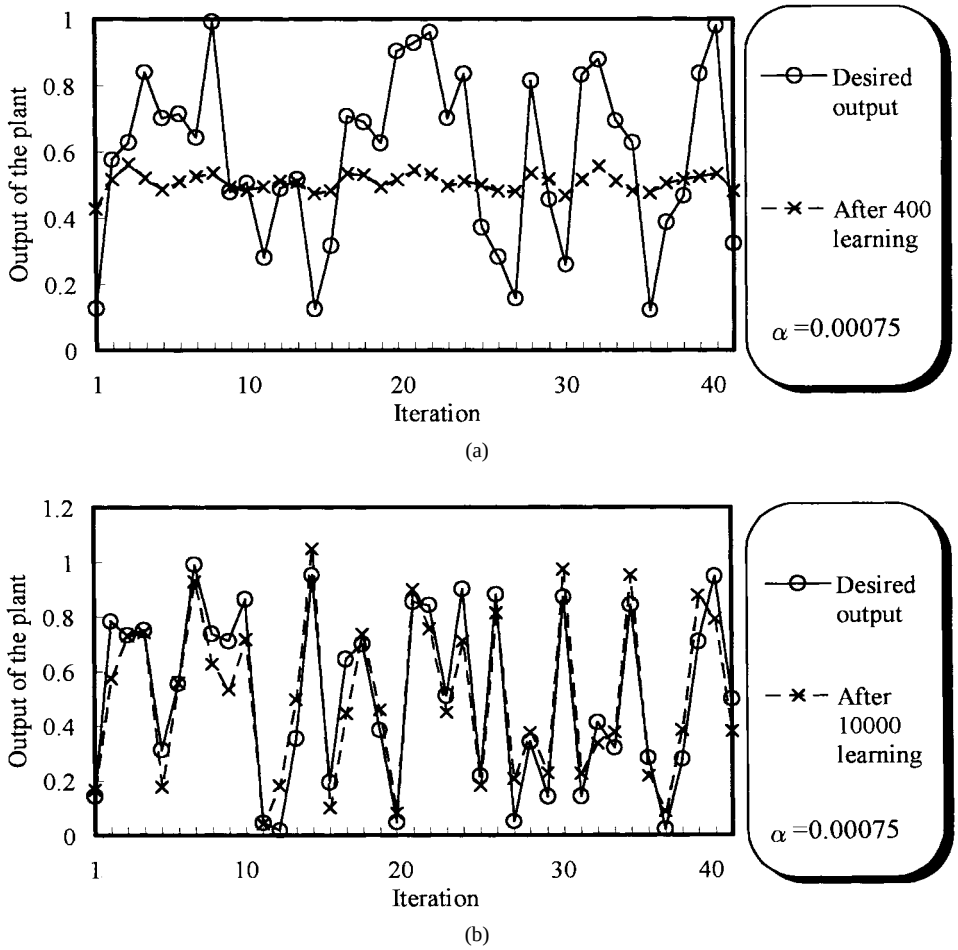


Fig. 15. Simulation results using a time-delay NN for random desired output by the learning rule 2.

used these simulations are randomly generated in the interval [−0.01 +0.01] except [−0.001 +0.001]. That is, $c_{\max} = 0.01$ and $c_{\min} = 0.001$. A characteristics of each neuron is the ordinary sigmoid function $1/(1+e^{-x})$ except the input layer. This layer is linear.

Simulation 1: First, we handle a simple static problem. We consider a two-link planar arm shown in Fig. 2. l_1 and l_2 denote length of the arms. θ_1 and θ_2 represent angles shown in the figure. Top of the arm is (xy). x and y are represented as follows using arm length l_1, l_2, and angles θ_1, θ_2

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2). \tag{11}$$

Then, we would like to find θ_1 and θ_2 so as to agree (x/y) with a desired position (x_d/y_d). A NN used here has two inputs and two outputs. Open circles in Fig. 2 show the desired positions.

Implementation: We prepare ten positions to be learned. We assume that we can measure a position of the top of the arm. Thus, we obtain an squared error for both x and y axes for the jth position

$$(x_j - x_{dj})^2 + (y_j - y_{dj})^2. \tag{12}$$

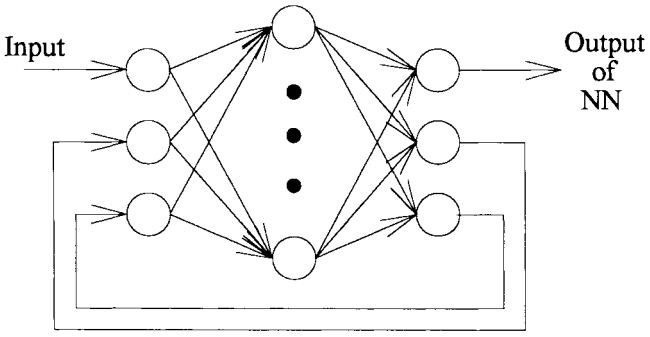


Fig. 16. Recurrent NN.

If we use the learning rule 1 and modify the weights every sets of the desired positions, we can define the total error function for the ten desired positions as follows, and can use the learning rule 1 of (7) directly

$$J = \sum_{j=1}^{10} \{(x_j - x_{dj})^2 + (y_j - y_{dj})^2\}. \tag{13}$$

That is, the error function is a sum of a squared error of both axes for ten desired positions.

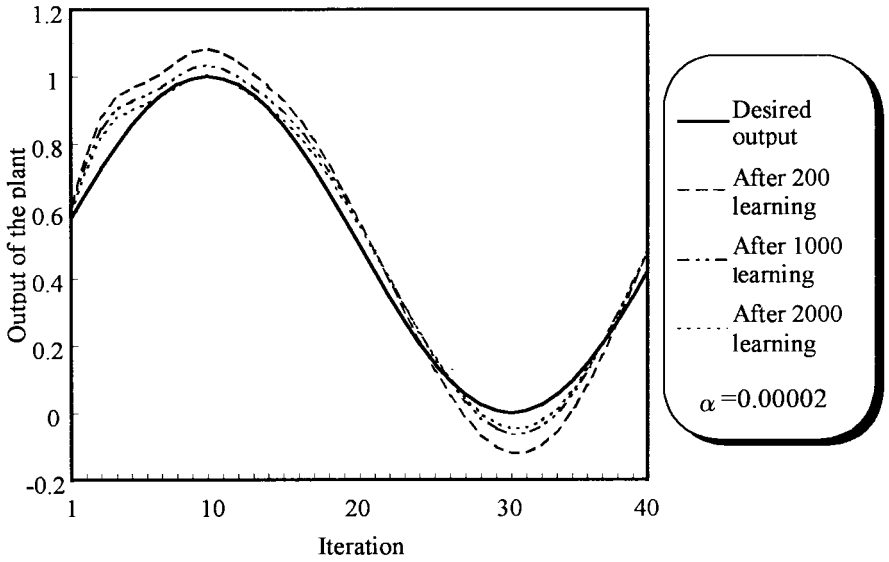


Fig. 17. Simulation results using a recurrent NN for sinusoidal desired output by the learning rule 1.

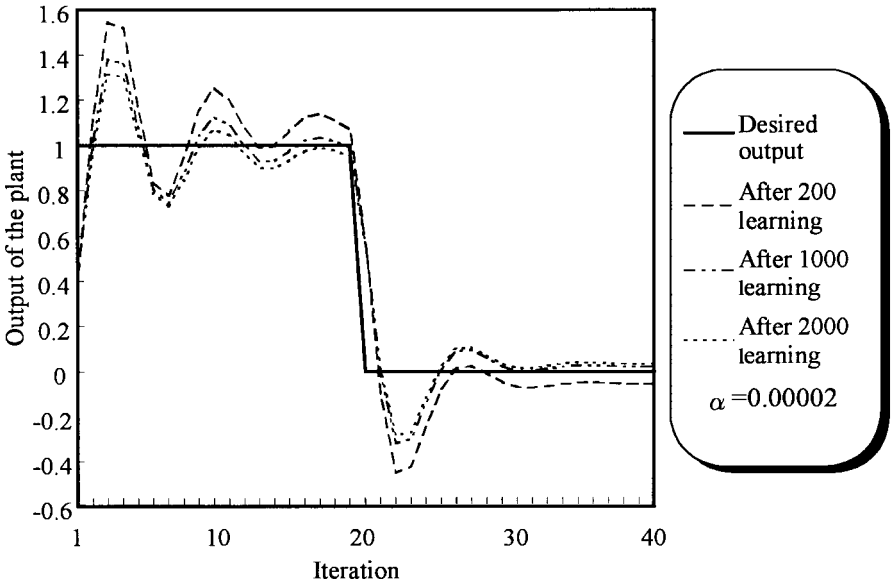


Fig. 18. Simulation results using a recurrent NN for square desired output by the learning rule 1.

The desired positions in x-y axes are applied to the network. The NN outputs two values in [0 1]. These values are converted to [0 2π], then applied to the system. We measure a position of the top of the arm and calculate the squared error. We repeat the same procedure for all desired positions and obtain the total error. Next, with perturbation, we reiterate the same procedure. Then, we obtain the total error similarly. After that, the weights of the NN were updated by the learning rule 1 of (7). The details are shown in Fig. 3.

In case of the learning rule 2 of (8), we do not use the total error function like (13). The objective system has two outputs of x and y axes. Therefore, by using errors $(x_j - x_{dj})$ and $(y_j - y_{dj})$, we can derive the following rule corresponding to the learning rule 2 of (8):

$$\Delta w_t^i = (x_j - x_{dj})\frac{\hat{x}_j - x_j}{c_t^i} + (y_j - y_{dj})\frac{\hat{y}_j - y_j}{c_t^i} \qquad (14)$$

where $\hat{x}$ and $\hat{y}$ denote positions when we added the perturbation vector to the weights.

When a desired position are applied to the NN, we can measure a position of the arm and calculate errors $(x_j - x_{dj})$ and $(y_j - y_{dj})$. Next, we add the perturbation to the NN. We reiterate the same procedure for the same desired position. Then, we obtain a position of the arm. Using (14), we calculate modifying quantities for the weights. We accumulate these for one cycle. Modification of the weights of the NN are carried out for every cycle. The details of this procedure are shown in Fig. 4.

Simulation results are shown in Figs. 5 and 6. Gradually, the top of the arm is approaching to the desired position as the learning proceeds. Then, the NN learns an inverse mapping $(x_{di}\ y_{di}) \mapsto (\theta_1\ \theta_2)$ at those ten positions.

Simulation 2: Next, we deal with a tracking problem. An objective plant has the following dynamics with a nonlinear

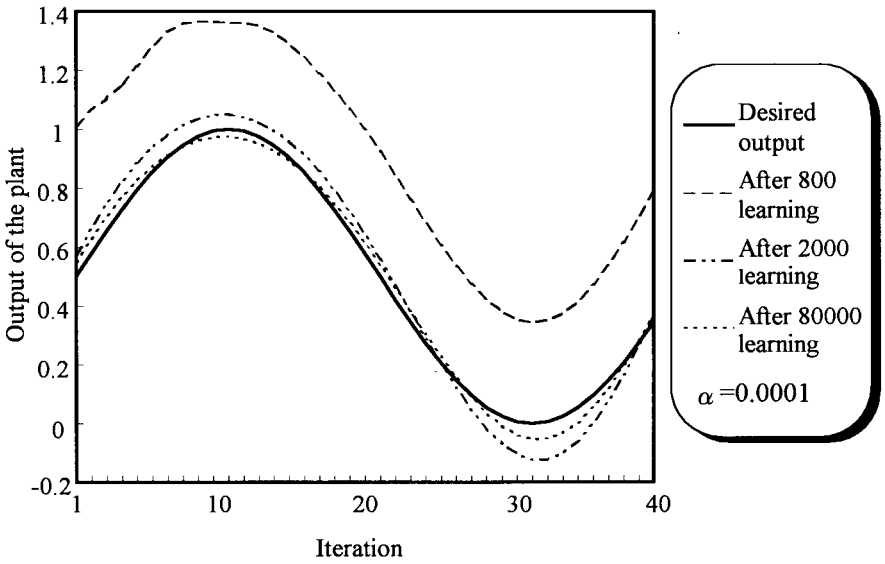


Fig. 19. Simulation results using a recurrent NN for sinusoidal desired output by the learning rule 2.

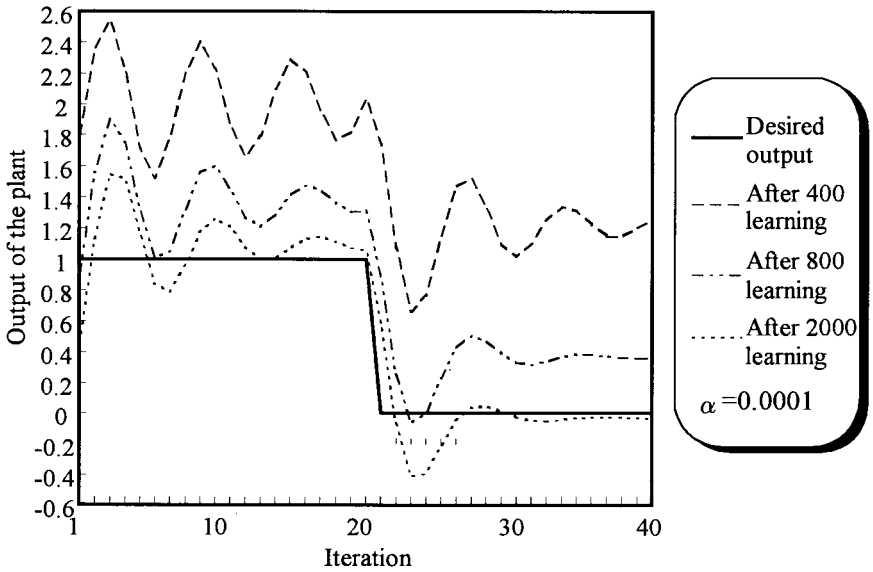


Fig. 20. Simulation results using a recurrent NN for square desired output by the learning rule 2.

term:

$$y_t = -a_1 y_{t-1} - a_2 y_{t-2} - a_3 y_{t-2}^2 + u_{t-1} + b u_{t-2} \quad (15)$$

where $a_1 = -1, a_2 = 0.5, a_3 = 0.1, b = 0.4$.

We would like to find an optimal input so that an output of the plant agrees with a desired output.

Implementation: Overall configuration are shown in Fig. 7. We use so-called time-delay feedforward NN with ten neurons in the hidden layer. The time-delay NN's consists of an ordinary feedforward NN's and unit time-delay elements. Feedforward NN's can handle static problems only, because they have no memory elements. However, by adding time-delay elements, they can deal with dynamic problems, virtually.

In this example, inputs and outputs of the plant are connected to the time-delay feedforward NN. Using unit time-delay elements, inputs to the network are $u_{t-1}, u_{t-2}, y_{t-1}$, and y_{t-2}. In addition, the desired output is also one of the inputs to the network. Therefore, the network converts these five inputs to manipulated variable.

The desired output of the plant are sinusoidal, square, or random waves, for example, shown in Fig. 9. One cycle consists of 40 iterations. Output [0 1] of the NN is linearly converted to [−2.5 2.5].

When we utilize the learning rule 1, we define a total error function as follows:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{40} (y_p - y_{dp})^2 \quad (16)$$

where p denotes the sampling time in each repetition and y_{dp} represents a desired output of the plant. That is, the total error function is the sum of the squared error between a practical output and a desired output of the plant for every one cycle.

First, we can measure a series of outputs of the plant for one cycle. Thus, we obtain a value of the total error function as a sum of the squared error at each sampling time. Next, we add the perturbation vector to the weight vector of the NN. Then, we repeat the same procedure. As a result, we obtain a value of the total error function under the perturbation. We apply the learning rule 1 shown in (7). Modification of the weights is carried out for every cycle. The details of the procedure are shown in Fig. 8.

Consider the learning rule 2. In this learning rule, we have to calculate the modifying quantities of the weights for every sampling time. Of course, we can sum up these quantities and update for every cycle as well. In this simulation, we update every sampling time.

A desired output applies to the NN, the NN outputs an input for the plant. Then, we can measure an output of the plant. We add the perturbation to the NN and repeat the same procedure. We obtain an output of the plant for the same desired output under an effect of the perturbation. By using the outputs of the plant, we utilize (8) and update the weights of the NN. The details are shown in Fig. 12.

The results of the simulations are shown in Figs. 13–15. Also in these examples, the more the learning proceeds, the more the accuracy improved.

The results of the simulations are shown in Figs. 9–11. The outputs of the plant are close to the desired outputs after enough learning.

We tried the same simulation for a different architecture of NN. Instead of the time-delay NN in Fig. 7, we apply a recurrent NN shown in Fig. 16. The setting of these simulations is as same as the previous examples. The simulation results by the learning rule 1 and the learning rule 2 are shown in Figs. 17–20. These results show that the learning rules using simultaneous perturbation is applicable to recurrent NN's.

IV. Conclusion

When we use an indirect inverse modeling by NN's, in order that we let the NN learn an inverse of an plant, we must know a sensitivity function of the plant, which may not be available. Using the difference approximation technique, we can apply a gradient-like learning rule to this scheme without information about the sensitivity function of the plant. This paper provides a difference approximation type learning rule using the simultaneous perturbation, which is a stochastic gradient-like learning rule, for the indirect inverse modeling by NN's. Two learning rules are described. These rules need only twice operations to obtain the modifying quantities of all weights.

In order to confirm a feasibility of the scheme, we examined two examples; a two-link planar arm and a tracking problem of a nonlinear dynamical system.

References

[1] W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., *Neural Network for Control*. Cambridge, MA: MIT Press, 1990.

[2] D. A. White and D. A. Sofge, Eds., *Handbook of Intelligent Control*. New York: Van Nostrand, 1992.

[3] Y. Ichikawa and T. Sawa, "Neural-network application for direct feedback controllers," *IEEE Trans. Neural Networks*, vol. 3, pp. 224–231, 1992.

[4] T. Fukuda and T. Shibata, "Theory and applications of neural networks for industrial control systems," *IEEE Trans. Ind. Electron.*, vol. 39, pp. 472–489, 1992.

[5] H. Tai, J. Wang, and K. Ashenayi, "Neural network-based tracking control system," *IEEE Trans. Ind. Electron.*, vol. 39, pp. 504–510, 1992.

[6] J. Tonomaru and S. Omatu, "Process control by on-line trained neural controllers," *IEEE Trans. Ind. Electron.*, vol. 39, pp. 511–521, 1992.

[7] A. Karakasoglu, S. I. Sudharsanam, and M. K. Sundareshan, "Identification and decentralized adaptive control using dynamical neural networks with application to robotic manipulators," *IEEE Trans. Neural Networks*, vol. 4, pp. 919–930, 1993.

[8] W. H. Schiffmann and H. W. Geffers, "Adaptive control of dynamic systems by backpropagation networks," *Neural Networks*, vol. 6, pp. 517–524, 1993.

[9] A. E. B. Ruano, P. J. Fleming, and D. I. Jones, "Connectionist approach to PID autotuning," in *Proc. Inst. Elect. Eng. pt. D*, vol. 139, 1992, pp. 279–285.

[10] T. Yamada and T. Yabuta, "Neural-network controller using autotuning method for nonlinear functions," *IEEE Trans. Neural Networks*, vol. 3, pp. 595–601, 1992.

[11] M. Khalid, S. Omatu, and R. Yusof, "MIMO furnace control with neural networks," *IEEE Trans. Contr. Syst. Technol.*, vol. 1, pp. 238–245, 1993.

[12] G. Lightbody and G. W. Irwin, "Direct neural model reference adaptive control," in *Proc. Inst. Elect. Eng. pt. D*, vol. 142, 1995, pp. 31–43.

[13] Y. S. Kung, C. M. Liaw, and M. S. Ouyang, "Adaptive speed control for induction motor drives using neural networks," *IEEE Trans. Neural Networks*, vol. 42, pp. 25–32, 1995.

[14] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986.

[15] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 154–157, 1992.

[16] Y. Maeda, H. Yamashita, and Y. Kanata, "Learning rules for multilayer neural networks using difference approximation," in *Proc. IJCNN*, Singapore, vol. 1, 1991, pp. 628–633.

[17] Y. Maeda, "Learning rule of neural networks for inverse systems," *Trans. Inst. Electron., Inform., Commun. Eng.*, vol. J75-A, pp. 1364–1369, 1992 (in Japanese); English version of this paper is in *Electron. Commun. Japan*, vol. 76, pp. 17–23, 1993.

[18] J. C. Spall, "A stochastic approximation technique for generating maximum likelihood parameter estimates," in *Proc. 1987 Amer. Contr. Conf.*, pp. 1161–1167.

[19] J. C. Spall, "A stochastic approximation algorithm for large-dimensional systems in the Kiefer–Wolfowitz setting," in *Proc. 27th IEEE Conf. Decision and Contr.*, 1988, pp. 1544–1548.

[20] J. C. Spall, "Multivariable stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 332–341, 1992.

[21] J. C. Spall and J. A. Cristion, "Nonlinear adaptive control using neural networks: Estimation with a smoothed form of simultaneous perturbation gradient approximation," *Statistica Sinica*, vol. 4, pp. 1–27, 1994.

[22] J. C. Spall and D. C. Chin, "A model-free approach to optimal signal light timing for system-wide traffic control," in *Proc. 1994 IEEE Conf. Decision and Contr.*, 1994, pp. 1868–1875.

[23] Y. Maeda, H. Hirano, and Y. Kanata, "A learning rule of neural networks via simultaneous perturbation and its hardware implementation," *Neural Networks*, vol. 8, pp. 251–259, 1995.

[24] Y. Maeda and Y. Kanata, "Learning rules for recurrent neural networks using perturbation and their application to neuro-control," *Trans. Inst. Electr. Eng. Japan*, vol. 113-C, pp. 402–408, 1993 (in Japanese).

[25] J. Alespector, R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe, "A parallel gradient descent method for learning in analog VLSI neural networks," in *Advances in Neural Information Processing Systems*, vol. 5, S. J. Hanson, J. D. Cowan, and C. Lee, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 836–844.

[26] G. Cauwenberghs, "A fast stochastic error-descent algorithm for supervised learning and optimization," in *Advances in Neural Information*

Processing Systems, vol. 5, S. J. Hanson, J. D. Cowan, and C. Lee, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 244–251.

[27] O. Fujita, "Trial-and-error correlation learning," *IEEE Trans. Neural Networks*, vol. 4, pp. 720–722, 1993.

Yutaka Maeda (M'92) received the B.E., M.E., and Ph.D. degrees in electronic engineering from the Osaka Prefecture University, Japan, in 1979, 1981, and 1990, respectively.

He joined Kansai University Faculty of Engineering in 1987, where he is an Associate Professor. In August to September 1993, he was a Visiting Researcher in Automatic Control Center of the Northeastern University, P.R. China. From 1995 to 1996, he was a Visiting Researcher in Electrical and Computer Engineering Department, University of California at Irvine. His research interests include control theory, artificial neural networks, and biomedical engineering.

Rui J. P. De Figueiredo (S'54–M'59–SM'74–F'76) received the B.S. and M.S. degrees in electrical engineering from Massachusetts Institute of Technology, Cambridge, and the Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA.

He is Professor of Electrical and Computer Engineering and Mathematics and Director of the Laboratory for Machine Intelligence and Neural and Soft Computing at the University of California, Irvine. He has authored or coauthored about 300 reviewed papers, book chapters, and books.

Dr. De Figueiredo served as Chair or Co-chair of eight international conferences including the 1980 IEEE International Symposium on Circuits and Systems (ISCAS'80), on the editorial boards of some journals of his specialty, and as Chair or Member of several national and international panels and committees. At the University of California, Irvine, he served as a Member of the Chancellors's Academic Planning Task Force in 1993–1994 and Chair of the Faculty of Engineering in 1994–1995. He has received a number of awards including the the 1988 NCR Faculty Award of Excellence, the 1990 Houston and Clear Lake Professional Societies (IEEE, ISA, AIAA) Council Technical Educator of the Year Award, and the 1994 IEEE CAS. At present, he is serving as President-Elect of the IEEE Circuits and Systems Society, and he will be the President of the Society in 1998.