Communicated by Michael Jordan

Refining PID Controllers Using Neural Networks

Gary M. Scott
Department of Chemical Engineering, University of Wisconsin, Madison, WI 53706 USA

Jude W. Shavlik
Department of Computer Sciences, University of Wisconsin, Madison, WI 53706 USA

W. Harmon Ray
Department of Chemical Engineering, University of Wisconsin, Madison, WI 53706 USA

The KBANN (Knowledge-Based Artificial Neural Networks) approach uses neural networks to refine knowledge that can be written in the form of simple propositional rules. We extend this idea further by presenting the MANNCON (Multivariable Artificial Neural Network Control) algorithm by which the mathematical equations governing a PID (Proportional-Integral-Derivative) controller determine the topology and initial weights of a network, which is further trained using backpropagation. We apply this method to the task of controlling the outflow and temperature of a water tank, producing statistically significant gains in accuracy over both a standard neural network approach and a nonlearning PID controller. Furthermore, using the PID knowledge to initialize the weights of the network produces statistically less variation in testset accuracy when compared to networks initialized with small random numbers.

1 Introduction

Research into the design of neural networks for process control has largely ignored existing knowledge about the task at hand. One form this knowledge (often called the "domain theory") can take is embodied in traditional controller paradigms. The recently developed KBANN approach (Towell *et al.* 1990) addresses this issue for tasks for which a domain theory (written using simple, nonrecursive propositional rules) is available. The basis of this approach is to use the existing knowledge to determine an appropriate network topology and initial weights, such that the network begins its learning process at a "good" starting point.

Neural Computation **4, 746–757** (1992)

This paper describes the MANNCON algorithm, a method of using a traditional controller paradigm to determine the topology and initial weights of a network. The use of a PID controller in this way eliminates network-design problems such as the choice of network topology (i.e., the number of hidden units) and reduces the sensitivity of the network to the initial values of the weights. Furthermore, the initial configuration of the network is closer to its final state than it would normally be in a randomly configured network. Thus, the MANNCON networks perform better and more consistently than the standard, randomly initialized three-layer approach.

The task we examine here is learning to control a nonlinear Multiple-Input, Multiple-Output (MIMO) system. There are a number of reasons to investigate this task using neural networks. First, many processes involve nonlinear input–output relationships, which matches the nonlinear nature of neural networks. Second, there have been a number of successful applications of neural networks to this task (Bhat and McAvoy 1990; Jordan and Jacobs 1990; Miller *et al.* 1990). Finally, there are a number of existing controller paradigms that can be used to determine the topology and the initial weights of the network.

The next sections introduce the MANNCON algorithm and describe an experiment that involves controlling the temperature and outflow of a water tank. The results of our experiment show that our network, designed using an existing controller paradigm, performs significantly better (and with significantly less variation) than a standard, three-layer network on the same task. The concluding sections describe some related work in the area of neural networks in control and some directions for future work on this algorithm.

In the course of this article, we use many symbols not only in defining the topology of the network, but also in describing the physical system and the PID controller. Table 1 defines these symbols and indicates the section of the paper in which each is defined. The table also describes the various subscripts to these symbols.

2 Controller Networks

The MANNCON algorithm uses a *Proportional–Integral–Derivative* (PID) controller (Stephanopoulos 1984), one of the simplest of the traditional feedback controller schemes, as the basis for the construction and initialization of a neural network controller. The basic idea of PID control is that the control action $\mathbf{u}$ (a vector) should be proportional to the error, the integral of the error over time, and the temporal derivative of the error. Several tuning parameters determine the contribution of these various components. Figure 1 depicts the resulting network topology based on the PID controller paradigm. The first layer of the network, that from $\mathbf{y}_{sp}$ (desired process output or setpoint) and $\mathbf{y}_{(n-1)}$ (actual process output

Table 1: Definitions of Symbols.

Symbol	Definition	Section Introduced
$\mathbf{d} = [F_d, T_d]$	Process disturbances	Section 2
$\mathbf{u} = [F_C, F_H]$	Process inputs	Section 2
$\mathbf{y} = [F(h), T]$	Process outputs	Section 2
$\mathbf{e}$	Simple error	Section 2
$\boldsymbol{\varepsilon}$	Precompensated error	Section 2
$\mathbf{G_I}$	Precompensator matrix	Section 2
F	Flow rate	Figure 1
T	Temperature	Figure 1
h	Height	Figure 2
K, τ_I, τ_D	PID tuning parameters	Section 2
ΔT	Time between control actions	Section 2
w	Network weights based on PID controller	Section 2
δ_{yj}	Error signal at plant output	Section 3
δ_{ui}	Error signal at plant input	Section 3
	Subscripts	
(n)	Value at current step	
$(n-1)$	Value at previous step	
sp	Setpoint	
d	Disturbance	
C	Cold water stream	
H	Hot water stream	

of the past time step), calculates the simple error ($\mathbf{e}$). A simple vector difference,

$$\mathbf{e} = \mathbf{y}_{sp} - \mathbf{y}$$

accomplishes this. The second layer, that between $\mathbf{e}$, $\varepsilon_{(n-1)}$, and ε, calculates the actual error to be passed to the PID mechanism. In effect, this layer acts as a steady-state precompensator (Ray 1981), where

$$\boldsymbol{\varepsilon} = \mathbf{G_I}\mathbf{e}$$

and produces the current error and the error signals at the past two time steps. This compensator is a constant matrix, $\mathbf{G_I}$, with values such that interactions at steady state between the various control loops are eliminated. The final layer, that between $\boldsymbol{\varepsilon}$ and $\mathbf{u}_{(n)}$ (controller output/plant input), calculates the controller action based on the velocity form of the discrete PID controller:

$$\begin{aligned} u_{C(n)} &= u_{C(n-1)} + K_C\left[1 + \frac{\Delta T}{\tau_{IC}} + \frac{\tau_{DC}}{\Delta T}\right]\varepsilon_{1(n)} \\ &\quad - K_C\left[1 + \frac{2\tau_{DC}}{\Delta T}\right]\varepsilon_{1(n-1)} + K_C\left[\frac{\tau_{DC}}{\Delta T}\right]\varepsilon_{1(n-2)} \end{aligned}$$

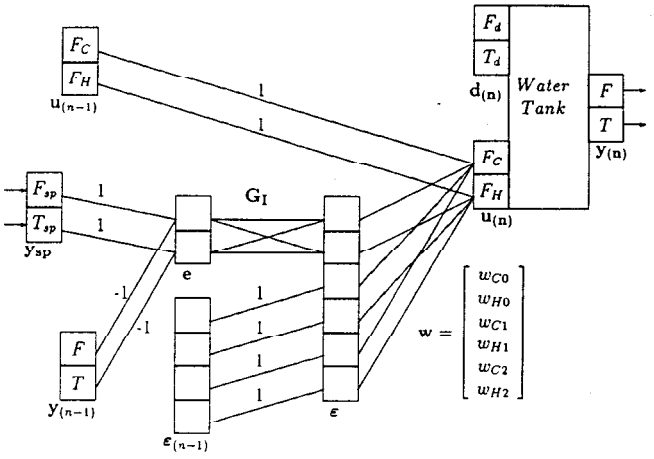

Figure 1: MANNCON network showing weights that are initialized using Ziegler–Nichols tuning parameters. See Table 1 for definitions of symbols.

where K_C, τ_{IC}, and τ_{DC} are the tuning parameters mentioned above, and ΔT is the discrete time interval between controller actions. This can be rewritten as

$$u_{C(n)} = u_{C(n-1)} + w_{C0}\varepsilon_{1(n)} + w_{C1}\varepsilon_{1(n-1)} + w_{C2}\varepsilon_{1(n-2)}$$

where w_{C0}, w_{C1}, and w_{C2} are constants determined by the tuning parameters of the controller for that loop. A similar set of equations and constants (w_{H0}, w_{H1}, w_{H2}) exist for the other controller loop.

Figure 2 shows a schematic of the water tank (Ray 1981) that the network controls. This figure also shows the variables that are the controller variables (F_C and F_H), the tank output variables [$F(h)$ and T], and the disturbance variables (F_d and T_d). The controller cannot measure the disturbances, which represent noise in the system.

MANNCON initializes the weights of the network in Figure 1 with values that mimic the behavior of a PID controller tuned with Ziegler–Nichols (Z–N) parameters (Stephanopoulos 1984) at a particular operating condition (the midpoint of the ranges of the operating conditions). Using the KBANN approach (see Appendix), it adds weights to the network such that all units in a layer are connected to all units in all subsequent layers, and initializes these weights to small random numbers several orders of magnitude smaller than the weights determined by the PID parameters. We scaled the inputs and the outputs of the network to be in the range $[0, 1]$.

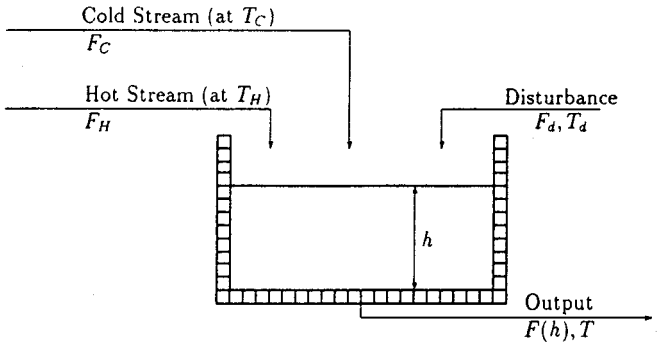

Figure 2: Stirred mixing tank requiring outflow and temperature control. See Table 1 for definitions of symbols.

Initializing the weights of the network in the manner given above assumes that the activation functions of the units in the network are linear, that is,

$$o_{j,\text{linear}} = \sum w_{ji} o_i$$

but the strength of neural networks lie in their having nonlinear (typically sigmoidal) activation functions. For this reason, the MANNCON system initially sets the weights (and the biases of the units) so that the linear response dictated by the PID initialization is *approximated* by a sigmoid over the output range of the unit. For units that have outputs in the range $[-1, 1]$, the activation function becomes

$$o_{j,\text{sigmoid}} = \frac{2}{1 + \exp(-2.31 \sum w_{ji} o_i)} - 1$$

which approximates the linear response quite well in the range $[-0.6, 0.6]$.

Once MANNCON configures and initializes the weights of the network, it uses a set of training examples and backpropagation to improve the accuracy of the network. The weights initialized with PID information, as well as those initialized with small random numbers, change during backpropagation training.

3 Experimental Details

We compared the performance of three networks that differed in their topology and/or their method of initialization. Table 2 summarizes the

Table 2: Topology and Initialization of Networks.

Network	Topology	Weight Initialization
1. Standard neural network	Three-layer (14 hidden units)	Random
2. MANNCON network I	PID topology	Random
3. MANNCON network II	PID topology	Z–N tuning

network topology and weight initialization method for each network. In this table, "PID topology" is the network structure shown in Figure 1. "Random" weight initialization sets all weights to small random numbers centered around zero. We trained the networks using backpropagation over a randomly determined schedule of setpoint $\mathbf{y}_{sp}$ and disturbance $\mathbf{d}$ changes that did not repeat. The setpoints, which represent the desired output values that the controller is to maintain, are the temperature and outflow of the tank. The disturbances, which represent noise, are the inflow rate and temperature of a disturbance stream. The magnitudes of the setpoints and the disturbances each formed gaussian distributions centered at 0.5. The number of training examples between changes in the setpoints and disturbances were exponentially distributed. For example, the original setpoints could be an output flow of 0.7 liters/sec at a temperature of 40°C. After 15 sec (which represents 15 training examples since time is discretized into one-second slices), the setpoints could change to new values, such as a flow of 0.3 liters/sec at 35°C. The flow rate and the temperature of the disturbance stream also varied in this manner.

We used the error at the output of the plant ($\mathbf{y}$ in Fig. 1) to determine the network error (at $\mathbf{u}$) by propagating the error backward through the plant (Jordan and Rumelhart 1990). In this method, the error signal at the input to the process is given by

$$\delta_{ui} = f'(\text{net}_{ui}) \sum_j \delta_{yj} \frac{\partial y_i}{\partial u_i}$$

where δ_{yj} represents the simple error at the output of the water tank and δ_{ui} is the error signal at the input of the tank. Since we used a *model* of the process and not a real tank, we can calculate the partial derivatives from the process model equations. We periodically interrupted training and tested the network over a different (but similarly determined) schedule. Results are averaged over 10 runs for each of the networks.

We also compare these networks to a (nonlearning) PID controller, that had its tuning parameters determined using a standard design methodology (Stephanopoulos 1984). Using the MIDENT program of the CONSYD package (W. Harmon Ray Research Group 1989), we fit a linear, first-order

model to the outputs of the system when stimulated by random inputs. We then determined an appropriate steady-state precompensator (Ray 1981) and Z–N tuning parameters for a PID controller (Stephanopoulos 1984) using this model. Further details and additional experimentation are reported in Scott (1991).

4 Results

Figure 3 compares the performance of the three networks. As can be seen, the MANNCON networks show an increase in correctness over the standard neural network approach. Statistical analysis of the errors using a t test show that they differ significantly ($p = 0.005$). Furthermore, while the difference in performance between MANNCON network I and MANNCON network II is not significant, the difference in the *variance* of the testing error over different runs is significant ($p = 0.005$). Finally, the MANNCON networks perform significantly better ($p = 0.0005$) than the nonlearning PID controller tuned at the operating midpoint. The performance of the standard neural network represents the best of several trials with a varying number of hidden units ranging from 2 to 20.

A second observation from Figure 3 is that the MANNCON networks learned much more quickly than the standard neural-network approach. The MANNCON networks required significantly fewer training instances

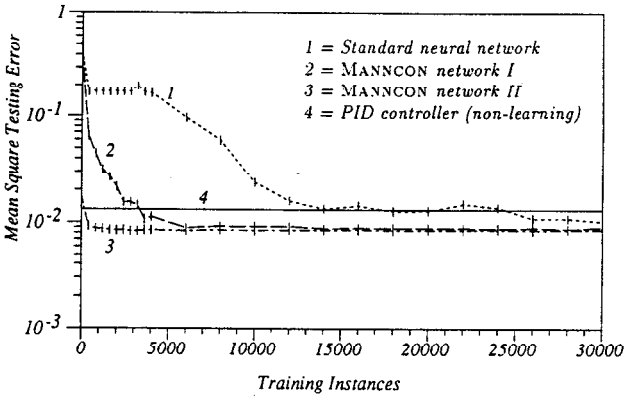

Figure 3: Mean square error of networks on the testset as a function of the number of training instances presented.

Table 3: Comparison of Network Performance.

Method	Mean square error	Training instances
1. Standard neural network	0.0103 ± 0.0004	$25,200 \pm 2,260$
2. MANNCON network I	0.0090 ± 0.0006	$5,000 \pm 3,340$
3. MANNCON network II	0.0086 ± 0.0001	640 ± 200
4. PID control (Z–N tuning)	0.0131	

to reach a performance level within 5% of its final error rate. Table 3 summarizes the final mean error for each of these three network paradigms, as well as the number of training instances required to achieve a performance within 5% of this value.

5 Related Research

A great deal of research in both recurrent networks and in using neural networks for control share similarities with the approach presented here. The idea of returning the output of the network (and of the system) from the previous training instance to become part of the input for the current training instance is quite common in works pertaining to control (Jordan and Jacobs 1990; Miller *et al.* 1990). However, this idea also appears in problems pertaining to natural language (Gori *et al.* 1989) and protein folding (Maclin and Shavlik 1991).

In the area of process control, there have been many approaches that use neural networks. Introductions to neural network control are given by Bavarian (1988), Franklin (1990), and Werbos (1990). Feedforward controllers, in which the neural network learns to mimic the *inverse* of the plant are discussed by Psaltis *et al.* (1988), Hosogi (1990), Li and Slotine (1989), and Guez and Bar-Kana (1990). Chen (1990) proposes an adaptive neural network controller where the controller uses a system of two neural networks to model the plant. Hernández and Arkun (1990) propose a Dynamic Matrix Control (DMC) scheme in which the linear model is replaced by a neural network model of the process. Narendra and Parthasarathy (1990) propose a method of indirect adaptive control using neural networks in which one network is used as a controller while the second is the identification model for the process. Bhat and McAvoy (1990) propose an Internal Model Controller (IMC) that utilizes a neural network model of the process and its inverse. Systems that use neural networks in a supervisory position to other controllers have been developed by Kumar and Guez (1990) and Swiniarski (1990). The book by Miller *et al.* (1990) gives an overview of many of the techniques mentioned here.

6 Future Work

In training the MANNCON initialized networks, we found the backpropagation algorithm to be sensitive to the value of the learning rate and momentum value. There was much less sensitivity in the case of the randomly initialized networks. Small changes in either of these values could cause the network to fail to learn completely. The use of a method involving adaptive training parameters (Battiti 1990), and especially methods in which each weight has its own adaptive learning rate (Jacobs 1988; Minai and Williams 1990) should prove useful. Since not all weights in the network are equal (that is, some are initialized with information while some are not), the latter methods would seem to be particularly applicable.

Another question is whether the introduction of extra hidden units into the network would improve the performance by giving the network "room" to learn concepts that are completely outside of the given domain theory. The addition of extra hidden units as well as the removal of unused or unneeded units is still an area with much ongoing research.

Some "ringing" occurred in some of the trained networks. A future enhancement of this approach would be to create a network architecture that prevented this ringing from occurring, perhaps by limiting the changes in the controller actions to some relatively small values.

Another important goal of this approach is the application of it to other real-world processes. The water tank in this project, while illustrative of the approach, was quite simple. Much more difficult problems (such as those containing significant time delays) exist and should be explored.

There are several other controller paradigms that could be used as a basis for network construction and initialization. There are several different digital controllers, such as Deadbeat or Dahlin's, that could be used in place of the digital PID controller used in this project. DMC and IMC are also candidates for consideration for this approach.

Finally, neural networks are generally considered to be "black boxes," in that their inner workings are completely uninterpretable. Since the neural networks in this approach are initialized with information, it may be possible to in some way interpret the weights of the network and extract useful information from the trained network.

7 Conclusions

We have shown that using the MANNCON algorithm to structure and initialize a neural-network controller significantly improves the performance of the trained network in the following ways:

- Improved mean testset accuracy

- Less variability between runs
- Faster rate of learning

The MANNCON algorithm also determines a relevant network topology without resorting to trial-and-error methods. In addition, the algorithm, through initialization of the weights with prior knowledge, gives the backpropagation algorithm an appropriate direction in which to continue learning. Finally, since the units and some of the weights initially have physical interpretations, it seems that the MANNCON networks would be easier to interpret after training than standard, three-layer networks applied to the same task.

Appendix: Overview of the KBANN Algorithm

The KBANN algorithm translates symbolic knowledge into neural networks by defining the topology and connection weights of the network (Towell *et al.* 1990). It uses knowledge in the form of PROLOG-like clauses, to define what is known about a topic. As an example of the KBANN method, consider the simple knowledge base in Figure 4a, which defines the membership in category A. Figure 4b represents the hierarchical structure of these rules where solid lines and dashed lines represent necessary and prohibitory dependencies, respectively. Figure 4c represents the neural network that results from a translation of this knowledge base. Each unit in the neural network corresponds to a consequent or an antecedent in the knowledge base. The solid and dashed lines represent heavily weighted links in the neural network. The dotted lines represent the lines added to the network to allow refinement of the knowledge base.

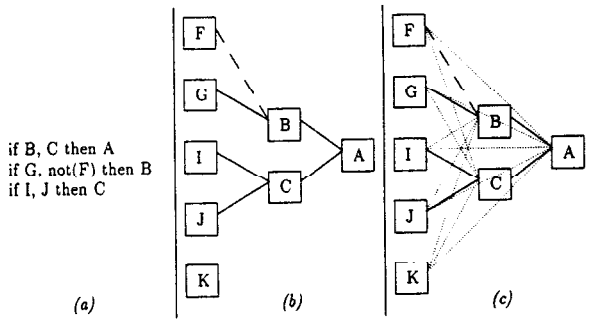

Figure 4: Translation of a knowledge base into a neural network using the KBANN algorithm.

Acknowledgments

G. M. S. was supported under a National Science Foundation Graduate Fellowship. J. W. S. was partially supported by Office of Naval Research Grant N00014-90-J-1941 and National Science Foundation Grant IRI-9002413. W. H. R. was partially supported by National Science Foundation Grant CPT-8715051.

References

Battiti, R. 1990. Optimization methods for back-propagation: Automatic parameter tuning and faster convergence. In *International Joint Conference on Neural Networks*, Vol. I, pp. 593–596, Washington, DC. Lawrence Erlbaum.

Bavarian, B. 1988. Introduction to neural networks for intelligent control. *IEEE Control Syst. Mag.* **8**, 3–7.

Bhat, N., and McAvoy, T. J. 1990. Use of neural nets for dynamic modeling and control of chemical process systems. *Comput. Chem. Eng.* **14**, 573–583.

Chen, F.-C. 1990. Back-propagation neural networks for nonlinear self-tuning adaptive control. *IEEE Control Syst. Mag.* **10**.

Franklin, J. A. 1990. Historical perspective and state of the art in connectionist learning control. In *28th Conference on Decision and Control*, Vol. 2, pp. 1730–1735, Tampa, FL. IEEE Control Systems Society.

Gori, M., Bengio, Y., and DeMori, R. 1989. Bps: A learning algorithm for capturing the dynamic nature of speech. In *International Joint Conference on Neural Networks*, Vol. II, pp. 417–423. San Diego, CA. IEEE.

Guez, A., and Bar-Kana, I. 1990. Two degree of freedom robot adaptive controller. In *American Control Conference*, Vol. 3, pp. 3001–3006. San Diego, CA. IEEE.

Hernández, E., and Arkun, Y. 1990. Neural network modeling and an extended DMC algorithm to control nonlinear systems. In *American Control Conference*, Vol. 3, pp. 2454–2459. San Diego, CA. IEEE.

Hosogi, S. 1990. Manipulator control using layered neural network model with self-organizing mechanism. In *International Joint Conference on Neural Networks*, Vol. 2, pp. 217–220. Washington, DC. Lawrence Erlbaum.

Jacobs, R. A. 1988. Increased rates of convergence through learning rate adaptation. *Neural Networks* **1**, 295–307.

Jordan, M. I., and Jacobs, R. A. 1990. Learning to control an unstable system with forward modeling. In *Advances in Neural Information Processing Systems*, Vol. 2, pp. 325 331. San Mateo, CA. Morgan Kaufmann.

Jordan, M. I., and Rumelhart, D. E. 1990. Forward models: Supervised learning with a distal teacher. Occasional Paper #40, Massachusetts Institute of Technology (to appear in *Cog. Sci.*).

Kumar, S. S., and Guez, A. 1990. Adaptive pole placement for neurocontrol. In *International Joint Conference on Neural Networks*, Vol. 2, pp. 397–400. Washington, DC. Lawrence Erlbaum.

Li, W., and Slotine, J.-J. E. 1989. Neural network control of unknown nonlinear systems. In *American Control Conference*, Vol. 2, pp. 1136–1141. San Diego, CA. IEEE.

Maclin, R., and Shavlik, J. W. 1991. Refining domain theories expressed as finite-state automata. In *Eighth International Workshop on Machine Learning*. Morgen Kaufmann, San Mateo, CA.

Miller, W. T., Sutton, R. S., and Werbos, P. J., eds. *Neural Networks for Control*. The MIT Press, Cambridge, MA.

Minai, A. A., and Williams, R. D. 1990. Acceleration of back-propagation through learning rate and momentum adaptation. In *International Joint Conference on Neural Networks*, Vol. I, pp. 676–679. Washington, DC. Lawrence Erlbaum.

Narendra, K. S., and Parthasarathy, D. 1990. Identification and control of dynamical systems using neural networks. *IEEE Transact. Neural Networks* 1(1), 4–27.

Psaltis, D., Sideris, A., and Yamamura, A. A. 1988. A multilayered neural network controller. *IEEE Control Syst. Mag.* 8, 17–21.

Ray, W. H. 1981. *Advanced Process Control*. McGraw-Hill, New York.

Scott, G. M. 1991. Refining PID controllers using neural networks. Master's project, University of Wisconsin, Department of Computer Sciences, May.

Stephanopoulos, G. 1984. *Chemical Process Control: An Introduction to Theory and Practice*. Prentice-Hall, Englewood Cliffs, NJ.

Swiniarski, R. W. 1990. Novel neural network based self-tuning PID controller which uses pattern recognition technique. In *American Control Conference*, Vol. 3, pp. 3023–3024. San Diego, CA. IEEE.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. 1990. Refinement of approximate domain theories by knowledge-base neural networks. In *Eighth National Conference on Artificial Intelligence*, pp. 861–866. AAAI Press, Menlo Park, CA.

W. Harmon Ray Research Group. 1989. Department of Chemical Engineering, University of Wisconsin, Madison. *CONSYD: Computer-Aided Control System Design*.

Werbos, P. J. 1990. Neural networks for control and system identification. In *28th Conference on Decision and Control*, Vol. 1, pp. 260–265. Tampa, FL. IEEE Control Systems Society.

Received 6 September 1991; accepted 28 February 1992.