ADAPTIVE CONTROL OF NONLINEAR SYSTEMS USING NEURAL NETWORKS

By

Fu-Chuang Chen

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1990

ABSTRACT

ADAPTIVE CONTROL OF NONLINEAR SYSTEMS USING NEURAL NETWORKS

By

Fu-Chuang Chen

Layered neural networks are used in the adaptive control of nonlinear discrete-time systems. The control algorithm is described and two convergence results are provided. The first result shows that the plant output converges to zero in the adaptive regulation system. The second result shows that the error between the plant output and the reference command converges to a bounded ball in the adaptive tracking system. Computer simulations verify the theoretical results at the end of this thesis.

To my parents,

Chan-Chwang Chen and Hu Li-Shueh Chen

and my wife

Hway-Ming Ker

ACKNOWLEDGEMENTS

I wish to thank Dr. Hassan K. Khalil for his Patience and Guidance; and my committee members, Dr. R. O. Barr, Dr. P. M. FitzSimons, Dr. C. R. MacCluer, Dr. F. M. A. Salam, and Dr. R. Schlueter, for their help and valuable suggestions.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

Linearization by feedback [15] is a promising approach to the control of nonlinear systems. The essence of the idea is to transform a state space model of the plant into new coordinates where nonlinearities can be canceled (fully or partially) by feedback. The major challenge in performing such cancellation is the need to know precise models of the nonlinearities. One approach to address this challenge is to use adaptive control where the controller learns the nonlinearities on line. This idea has been investigated for continuous-time systems [17,18] assuming that the nonlinearities can be parametrized linearly in some unknown parameters. In this thesis we investigate a similar scheme for discrete-time systems, but we do not assume that the nonlinearities depend linearly on unknown parameters. Instead, we explore the use of layered neural networks to model the nonlinearities. In the discrete-time self-tuning adaptive control scheme, the linearizing control is generated from the information provided by the neural network. Then, the observed error is used to train the neural network to improve its approximation of the unknown nonlinear plant. A review of neural network research is given in sections **1.1** and **1.2** in chapter **1**. Section **1.3** provides some background for feedback linearization.

In chapter **2** we derive output feedback linearizing control for a discrete-time nonlinear system represented by an input-output model. The relative degree of the system could be higher than one. In chapter **3**, a neural network architecture is suggested for modeling nonlinear systems. We will describe two different methods for applying layered neural networks to adaptive control problems and provide the associated learning rules. The theoretical results of this research are presented in chapters **4** and **5**. In these two chapters, we show local convergence properties based on different network models and different learning rules. Simulation results are provided in

chapter **6**. Chapter **7** is the conclusion.

1.1 Neural Computing Research

Theoretical brain research has been published in the contexts of theoretical biology, mathematical psychology, cybernetics, pattern recognition, the theories of adaptive systems, and others. Recently the terms "neural computing" and "neural networks" have been adopted to address more practical issues such as vision, sensory-motor control, associative memory, supervised learning, unsupervised learning, robotics, etc. Although these studies have rather diverse origins, they often have one common objective : *to implement new types of computers.*

Traditional digital computers do very well on tasks which we know how to proceed to solve. However, it is very difficult to program a digital computer to solve problems such as vision and speech recognition. The reasons are: first, we do not have enough information about how these tasks are actually done in the brain; second, even if sufficient knowledge is available about the function of the brain, it may be incomputable by digital computers. The result of decades of research in artificial intelligence may justify the arguments above. The most successful subfield in AI is expert system. Expert systems are programs which solve specific problems using information collected from domain experts. In contrast, the results are much more limited when applying AI techniques to vision and language understanding problems.

Artificial neural networks are networks of processing elements (i.e., "neurons") that are interconnected. Each neuron can have multiple input signals, but only one output signal. Different interconnection topologies and learning rules determine different neural network paradigms. Artificial neural networks are considered models of the brain, and they are intended to interact with the real world in the same way as the biological nervous systems do (at least for the original purpose). Most existing

neural network architectures are constructed to reproduce specific brain functions. Since our knowledge about how brain functions is still very limited, existing artificial neural networks may be too simple compared with their biological counterpart. However, as our knowledge and experience increase, new and more sophisticated neural network models will replace the old ones.

"Neural computing" became a very hot research area starting from the mid 80's. Not all research efforts in this field are biologically motivated. In particular, in engineering applications, artificial neural networks can be viewed as some new tools which seem able to attack traditionally difficult problems.

Historical reviews and current developments in neural computing can be found in [20,21,22].

1.2 Neural Networks in Control

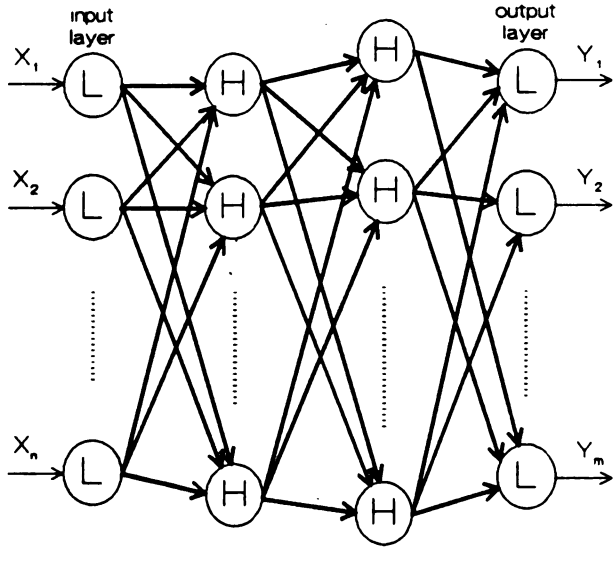


Figure 1.1 A layered neural network with two nonlinear hidden layers

In this section we concentrate on the discussion of layered neural networks, since they are the most prevailing network architecture studied for identification and control applications. A layered neural network, shown in Figure 1.1, consists of an input layer, an output layer, and at least one layer of nonlinear neurons. The nonlinear neurons sum incoming signals and generate output signals according to some predefined functions. The neurons are interconnected layer by layer. The output of one neuron multiplied by a weight becomes the input of adjacent neurons of next layer.

Layered neural networks have good potentials for control applications because they can approximate nonlinear functions. it was noted more than two decades ago by Minsky and Papert [23] that by inserting "nonlinear hidden neurons" between the input layer and the output layer, the XOR mapping (which is a nonlinear mapping) can be represented by the network. Recently, it is shown by Funahashi [24], Cybenko [25], Hornik et al. [26], and Hecht-Nielson [3], using different techniques, that layered neural networks can approximate any "well-behaved" nonlinear function to any desired accuracy. The theorem shown by Funahashi is quoted here.

Theorem

Let $\phi(x)$ be a nonconstant, bounded and monotonically increasing continuous function. Let K be a compact subset of R^n and $f(x_1,\ldots,x_n)$ be a real valued continuous function on K. Then for any $\epsilon > 0$, there exists an integer N and real constants c_i, $\theta_i (i = 1,\ldots,N)$, $w_{ij} (i = 1,\ldots,N, j = 1,\ldots,n)$ such that

$$\tilde{f}(x_1,\ldots,x_n) = \sum_{i=1}^{N} c_i \phi\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right) \tag{1}$$

satisfies $\max_{x \in K} \left| f(x_1,\ldots,x_n) - \tilde{f}(x_1,\ldots,x_n) \right| < \epsilon$.

In other words, given any function $f(x_1,\ldots,x_n)$ and any arbitrary $\epsilon > 0$, there exists a three-layer network $\tilde{f}(x_1,\ldots,x_n)$ with linear input and output layers and with a hidden layer whose output functions are $\phi(x)$, such that

$\max_{x \in K} \left| f(x_1, \ldots, x_n) - \tilde{f}(x_1, \ldots, x_n) \right| < \epsilon$. Similar result for neural networks with more than one hidden layer can be derived from the theorem above or be shown from scratch [24]. Notice that the theorem is an existence result. It does not give an estimate of the number of neurons needed to approximate a nonlinear function given a specified error bound, nor does it say how to choose the weights. In control applications some *ad hoc* procedures are used to determine a suitable size of the network.

The next crucial issue is to train the network to approximate a given function. The back propagation algorithm [1] is a widely accepted method to train a neural network to approximate a function. If there is difference between the function output and the network output for the same input, the difference can be used in the back propagation algorithm to adjust the weights in the neural network in order to reduce the error. The training is usually a time consuming process, and researchers are suggesting modifications to the original back propagation algorithm to increase the learning speed [32,33]. There is no theoretical result available yet about the convergence of the training. However, many applications reported in the literature have confirmed the value of applying layered neural networks to various problems, e.g. [1 - 14].

Some recent papers on the application of neural networks to control and identification problems are reviewed in the following examples.

Example 1.1 : [4,5,6]

If the input vector $U(k)$ of a nonlinear system can be uniquely determined by its output vector $Y(k)$ through a static mapping

$$U(k) = f(Y(k)),$$

then layered neural networks can be used to learn this mapping and generate controls.

For example, the dynamical equations of robotics can be rearranged into

$$U(k) = f(q(k), \dot{q}(k), \tilde{q}(k)),$$

where $U(k)$ is the vector of the joint torques and $q(k)$ is the vector of the joint angles. The neural network can be trained to approximate the inverted mapping as shown in Figure 1.2(a) and then be used as a feedforward compensator in Figure 1.2(b). On line learning can be carried out as illustrated in Figure 1.2(c), where two identical neural networks are used. The neural network in the feedback loop identifies the inverted plant on line and its updated weights are copied to the second neural network in the feedforward path. Simulation results have been presented in the referenced papers. □

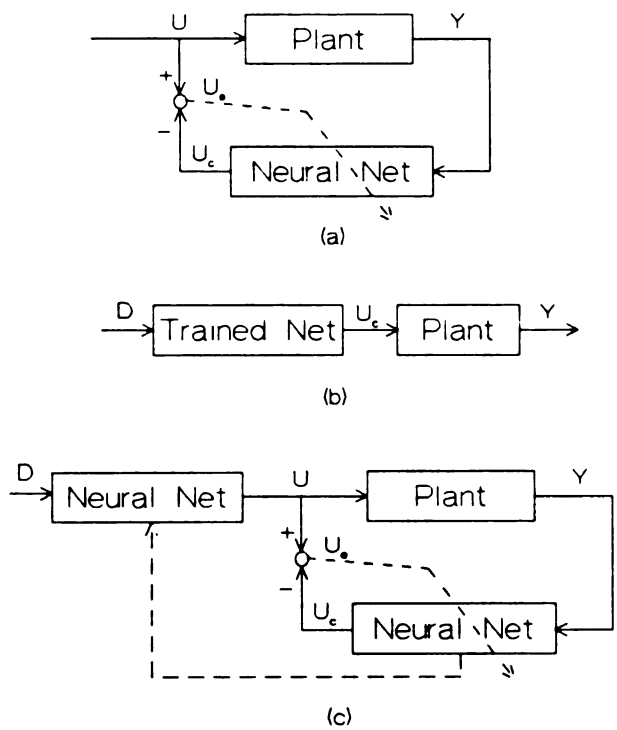


Figure 1.2 See Example 1.1 of Section 1.2 for description.

Example 1.2 : [7]

Consider a dynamical system

$$x(k+1) = f_1(x(k), u(k)).$$

Assume that the order of the plant (i.e., the number of the states) is known, say n, and that the states are physically measurable. The states at $k+2$ are

$$x(k+2) = f_1(x(k+1), u(k+1)) = f_1(f_1(x(k), u(k)), u(k+1)) = f_2(x(k), u(k), u(k+1)).$$

Repeating the process, one concludes that the states at time $k+n$ is determined by the state at time k and the controls from time k to $(k+n-1)$, i.e.,

$$x(k+n) = f_n(x(k), U), \tag{2}$$

where

$$U = [u(k), u(k+1), \ldots, u(k+n-1)]^T.$$

Assume that equation (2) is uniquely invertible for U. Then U can be solved as

$$U = g(x(k), x(k+n)). \tag{3}$$

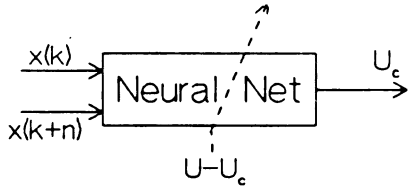


Figure 1.3 See Example 1.2 of Section 1.2 for description

A layered neural network, as shown in Figure 1.3, can be used to approximate (3).

$$U_c = \hat{g}(x(k), x(k+n), W)$$

At each time step k the training of the neural network can be described as follows:

- Input the states $x(k-n)$ and $x(n)$ to the neural network.

- The error $(U - U_c)$ is used to update the weights of the neural network.

It was suggested to implement this learning and control scheme on line. □

Example 1.3 : [8,9]

Dynamical Systems Identification. Layered neural networks can be used to identify a class of unknown nonlinear functions

$$y(k+1) = f(y(k), y(k-1), \ldots, y(k-n+1), u(k), u(k-1), \ldots, u(k-m+1)),$$

where $n \geq m$. As depicted in Figure 1.4(a), this is essentially a function approximation problem. At each time step k, the control $u(k)$ as well as all of the relevant past inputs and outputs are applied to the neural network input layer. The error between the output of the neural network and the plant output $y(k+1)$ is used to train the network.

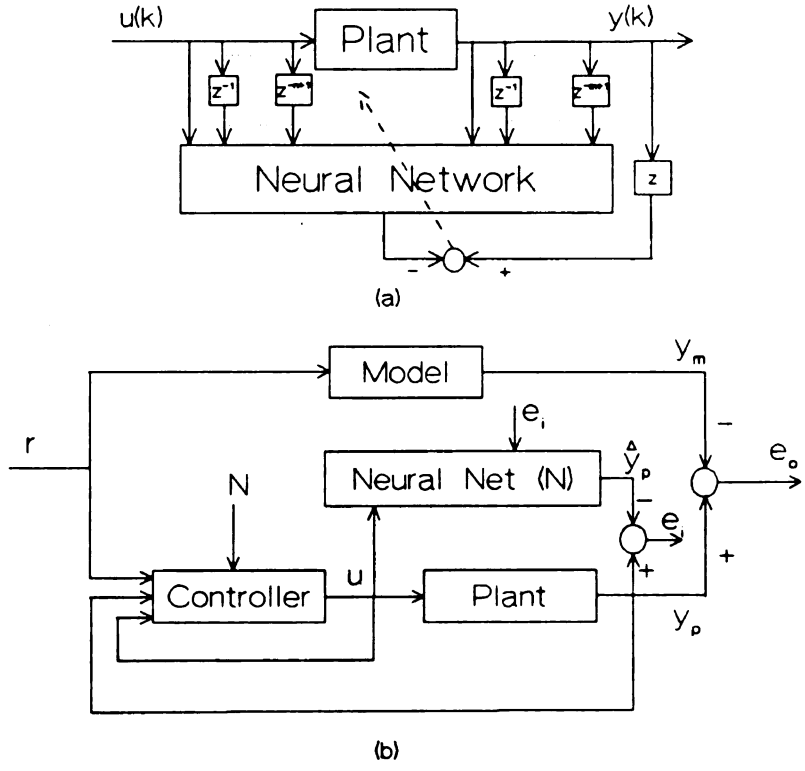


Figure 1.4 See Example 1.3 of Section 1.2 for description.

Adaptive Control. For a special class of nonlinear unknown systems such as

$$y(k+1) = f(y(k), y(k-1), \ldots, y(k-n+1)) + g(y(k), y(k-1), \ldots, y(k-n+1))u(k),$$

where the control $u(k)$ appears linearly, layered neural networks can be used in the Self-Tuning framework (Figure 1.4(b), with the Model block as 1) or in the Model Reference framework (Figure 1.4(b)) to adaptively control the system. The neural network is used to learn the characteristics of the plant on line and generate appropriate controls to be applied to the plant in order to cancel the plant (self-tuning) or control the closed-loop system to follow the output of a desired model (model reference). Details about neural-network-based self-tuning adaptive control will be provided in chapter **3.** □

There are other approaches suggested in recent papers. In [10] and [2], neural nets are used directly as controllers, but this approach bears a less direct connection to traditional control design methods. Other works include the application of the CMAC neural networks to robotics control problems [11] and the Reinforcement Learning Problems [12]. Favorable simulation results related to these techniques are available in the references listed.

The research in applying neural networks to control problems is still at the stage of proposing ways to incorporate neural networks into control systems. Few theoretical results are available to date, although there have some attempts to obtain theoretical results [13,14].

1.3 Feedback Linearization of Minimum-Phase Nonlinear Discrete Time Systems

The concept of zero dynamics and the minimum phase property for nonlinear continuous-time systems were introduced by Isidori and coworkers [15]. They were adapted to the discrete-time case by Monaco and Normand-Cyrot [16]. Consider a single-

input/single-output system of the form:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) \\ y(k) &= h(x(k)) \end{aligned} \tag{4}$$

with $x(k) \in R^n$, $u(k) \in R$, $y(k) \in R$, and f and h analytic functions on their domains. Denoting by f_0 the undriven state dynamics $f(\cdot, 0)$ and by f_0^j the j-times iterated composition of f_0. The system is said to be of *relative degree d* if

$$\frac{\partial h \circ f_0^k \circ f(x,u)}{\partial u} \equiv 0 \qquad 0 \leq k < d-1$$

and

$$\frac{\partial h \circ f_0^{d-1} \circ f(x,u)}{\partial u} \neq 0 \ \ a.e. \ \ in \ \ R^{n+1}$$

$y(d)$ is the first output affected by the input $u(0)$. Let $r \in R$ be an external control. A nonlinear static state feedback control is denoted by

$$u = \gamma(x, r) \tag{5}$$

If $\frac{\partial \gamma}{\partial r} \neq 0$, the feedback law (5) is said to be nonsingular.

Suppose the system (4) is of relative degree d. Solve the state equation of (4) recursively to express $y(k+d)$ in the form

$$y(k+d) = F(x(k), u(k))$$

An important assumption about the system (4) is that

$$0 \in Range(F(x, \bullet)) \quad \forall x \tag{6}$$

It also follows from the definition of relative degree d that

$$\frac{\partial F(x,u)}{\partial u} \neq 0$$

Therefore, the implicit function theorem can be used to show the following theorem.

Theorem [16]

Assuming (6) is satisfied and $d < \infty$, then there exists a nonsingular feedback control law of the form (5) such that the closed-loop system, after a suitable change of coordinates $z = T(x)$, is described by the equations

$$\begin{aligned} z_1(k+1) &= Az_1(k) + Br(k) \\ z_2(k+1) &= F(z_1(k), z_2(k), r(k)) \\ y(k) &= Cz_1(k) \end{aligned} \tag{7}$$

where dim $z_1 = d$ and (A, B, C) is a controllable-observable triple. □

A very informative proof is available in [16]. From (7) we see that if $d < n$, the z_2 component of the state will be strongly unobservable in the closed-loop system, because z_2 has no effect on the plant output; if $d = n$ the system is fully linearizable.

If system (7) starts from $z_1(0) = 0$ and $r \equiv 0$, then $z_1 \equiv 0$ and the plant output stays at zero. The motion of the system is determined by the dynamics of z_2, which gives rise to the notion of zero dynamics.

Definition

The *Zero Dynamics* of system (7) are defined to be

$$z_2(k+1) = F(0, z_2(k), 0). \quad \square \tag{8}$$

The system is said to be minimum phase if the zero dynamics have an asymptotically stable equilibrium at the origin.

2 Linearizing Feedback Control

Many interesting systems can be described by a nonlinear model in which the control appears linearly. We are interested in the single-input/single-output nonlinear discrete-time system:

$$\begin{aligned} y_{k+1} &= f_0(y_k, y_{k-1}, \cdots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \cdots, u_{k-d-m+1}) \\ &+ g_0(y_k, y_{k-1}, \cdots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \cdots, u_{k-d-m+1})u_{k-d+1}, \end{aligned} \tag{9}$$

where $m \leq n$, y is the output, u is the input, d is the relative degree of the system, and g_0 is bounded away from zero. The arguments of f_0 and g_0 are real variables. Compared with the *deterministic autoregressive moving average* (DARMA) model

$$\begin{aligned} y_{k+1} &= \sum_{i=0}^{n-1} a_i y_{k-i} + \sum_{i=0}^{m-1} b_i u_{k-d+1-i} \\ &= [\sum_{i=0}^{n-1} a_i y_{k-i} + \sum_{i=1}^{m-1} b_i u_{k-d+1-i}] + b_0 u_{k-d-1} \end{aligned} \tag{10}$$

$[\sum_{i=0}^{n-1} a_i y_{k-i} + \sum_{i=1}^{m-1} b_i u_{k-d+1-i}]$ is a special case of f_0 and b_0 is a special case of g_0.

The functions f_0 and g_0 are unknown. The objective is to design a self-tuning control system using neural networks so that the output of the plant will asymptotically track the command. Two obvious difficulties show up immediately. First, even if f_0 and g_0 were known, the control law cannot simply be

$$u_{k-d+1} = -\frac{f_0(\cdot)}{g_0(\cdot)} + \frac{r(k)}{g_0(\cdot)},$$

because this control is noncausal when $d > 1$. Second, since f_0 and g_0 depend on past inputs, the system may become internally unstable after the feedback control, if it exists, cancels the plant dynamics. These two issues are well known for linear discrete-time systems [19,27]. Especially it is shown in [27] that the system (10) can be converted into

$$y_{k+d} = \sum_{i=0}^{n-1} \alpha_i y_{k-i} + \sum_{i=0}^{m+d-1} \beta_i u_{k-i} \tag{11}$$

$$= [\sum_{i=0}^{n-1} \alpha_i y_{k-i} + \sum_{i=1}^{m+d-1} \beta_i u_{k-i}] + \beta_0 u_k$$

Then the control u_k in (11) can be determined in terms of past inputs and past outputs to cancel the plant dynamics, and the effect of the control u_k will show up at the plant output d steps later. The purpose of this section is to derive the nonlinear counterpart of (11) for the nonlinear system (9) and to define the zero dynamics associated with (9).

The work of Monaco and Normand-Cyrot [16] suggests that important properties of system (9) may be revealed if the system is put into state space form and some suitable coordinate transformation is performed on the model. We select the state variables as the current output and all past inputs and outputs up to the most delayed input or output on the right-hand side of (9), i.e.,

$$\begin{aligned}
x_1(k) &= y_{k-n+1} \\
&\vdots \\
x_{n-1}(k) &= y_{k-1} \\
x_n(k) &= y_k \\
x_{n+1}(k) &= u_{k-d-m+1} \\
&\vdots \\
x_{n+m+1}(k) &= u_{k-d+1} \\
&\vdots \\
x_{n+m+d-1}(k) &= u_{k-1}.
\end{aligned}$$

Let $\mathbf{x}(k)$ be the state vector. A state space model of (9) is constructed accordingly as

$$x_1(k+1) = x_2(k)$$

$$
\begin{aligned}
&\vdots \\
x_{n-1}(k+1) &= x_n(k) \qquad (12)\\
x_n(k+1) &= f_0(x_n(k), x_{n-1}(k), \ldots, x_1(k), x_{n+m}(k), x_{n+m-1}(k), \ldots, x_{n+1}(k)) \\
&\quad + g_0(x_n(k), \ldots, x_1(k), x_{n+m}(k), \ldots, x_{n+1}(k)) x_{n+m+1}(k) \\
&= f_0(x_1(k), \ldots, x_{n+m}(k)) + g_0(x_1(k), \ldots, x_{n+m}(k)) x_{n+m+1}(k) \\
x_{n+1}(k+1) &= x_{n+2}(k) \\
&\vdots \\
x_{n+m+1}(k+1) &= x_{n+m+2}(k) \\
&\vdots \\
x_{n+m+d-1}(k+1) &= u_k \\
y(k) &= x_n(k).
\end{aligned}
$$

There are $(n+m+d-1)$ states. The state space representation (12) is, in general, a nonminimal realization. However, no difficulty arises from working with this nonminimal realization since the redundant dynamics are stable (for linear systems the uncontrollable/unobservable eigenvalues are at the origin). In the following we derive a transformation that transforms system (12) into the form (7).

$$
\begin{aligned}
x_n(k+2) &= y(k+2) \\
&= f_0(x_1(k+1), \ldots, x_n(k+1), \ldots, x_{n+m}(k+1)) \qquad (13)\\
&\quad + g_0(x_1(k+1), \ldots, x_n(k+1), \ldots, x_{n+m}(k+1)) x_{n+m+1}(k+1).
\end{aligned}
$$

After substituting (12) into (13), we have

$$
\begin{aligned}
x_n(k+2) &= y(k+2) \\
&= f_1(x_1(k), \ldots, x_{n+m+1}(k)) + g_1(x_1(k), \ldots, x_{n+m+1}(k)) x_{n+m+2}(k).
\end{aligned}
$$

By applying the same technique recursively, one gets

$$\begin{aligned} x_n(k+3) &= f_2(x_1(k),\ldots,x_{n+m+2}(k)) + g_2(x_1(k),\ldots,x_{n+m+2}(k))x_{n+m+3}(k). \\ &\vdots \\ x_n(k+d-1) &= f_{d-2}(x_1(k),\ldots,x_{n+m+d-2}(k)) \\ &\quad + g_{d-2}(x_1(k),\ldots,x_{n+m+d-2}(k))x_{n+m+d-1}(k). \end{aligned}$$

Then the following state transformation is suggested,

$$\mathbf{z}(k) = \begin{bmatrix} z_{11}(k) \\ \vdots \\ z_{1n}(k) \\ z_{1,n+1}(k) \\ \vdots \\ z_{1,n+d-1}(k) \\ z_{21}(k) \\ \vdots \\ z_{2m}(k) \end{bmatrix} = \begin{bmatrix} x_1(k) \\ \vdots \\ x_n(k) \\ x_n(k+1) \\ \vdots \\ x_n(k+d-1) \\ x_{n+1}(k) \\ \vdots \\ x_{n+m}(k) \end{bmatrix} \tag{14}$$

$$= \begin{bmatrix} x_1(k) \\ \vdots \\ x_n(k) \\ f_0(\bullet) + g_0(\bullet)x_{n+m+1}(k) \\ \vdots \\ f_{d-2}(\bullet) + g_{d-2}(\bullet)x_{n+m+d-1}(k) \\ x_{n+1}(k) \\ \vdots \\ x_{n+m}(k) \end{bmatrix} = T(\mathbf{x}(k))$$

After this transformation, (12) becomes

$$\begin{aligned} z_{11}(k+1) &= z_{12}(k) \\ &\vdots \\ z_{1n}(k+1) &= z_{1,n+1}(k) \\ z_{1,n+1}(k+1) &= z_{1,n+2}(k) \\ &\vdots \end{aligned}$$

$$
\begin{aligned}
z_{1,n+d-1}(k+1) &= f_{d-1}(\mathbf{x}(k)) + g_{d-1}(\mathbf{x}(k))x_{n+m+d-1}(k+1) \\
&= f_{d-1}(T^{-1}(\mathbf{z}(k))) + g_{d-1}(T^{-1}(\mathbf{z}(k)))u_k \\
&= F(\mathbf{z}(k)) + G(\mathbf{z}(k))u_k \qquad (15) \\
z_{21}(k+1) &= z_{22}(k) \\
&\vdots \\
z_{2,m-1}(k+1) &= z_{2m}(k) \\
z_{2m}(k+1) &= u_{k-d+1} \qquad (16) \\
y(k) &= z_{1n}(k).
\end{aligned}
$$

Two interesting points about this transformed model can be discussed.

- If $F(\bullet)$ and $G(\bullet)$ in (15) were known, the control $u(k)$ could be defined as

$$u(k) = \frac{-F(\mathbf{z}(k)) + r(k)}{G(\mathbf{z}(k))} \qquad (17)$$

  and $r(k)$ will appear as the desired output d steps later.

- The past control $u(k-d+1)$ appears in (16). Notice that

$$
\begin{aligned}
u(k-d+1) &= \frac{-F(\mathbf{z}(k-d+1)) + r(k-d+1)}{G(\mathbf{z}(k-d+1))} \qquad (18) \\
&= \frac{-f_{d-1}(T^{-1}(\mathbf{z}(k-d+1))) + r(k-d+1)}{g_{d-1}(T^{-1}(\mathbf{z}(k-d+1)))} \qquad (19) \\
&= \frac{-f_{d-1}(\mathbf{x}(k-d+1)) + r(k-d+1)}{g_{d-1}(\mathbf{x}(k-d+1))} \qquad (20) \\
&= \frac{-f_0(\mathbf{x}(k)) + r(k-d+1)}{g_0(\mathbf{x}(k))} \qquad (21) \\
&= \frac{-f_0(T^{-1}(\mathbf{z}(k))) + r(k-d+1)}{g_0(T^{-1}(\mathbf{z}(k)))} \qquad (22)
\end{aligned}
$$

  Thus, (22) can be substituted into (16). This makes the right hand side of the transformed model a function of the state $\mathbf{z}(k)$ and the input $r(k)$. Similar to the definition given in Monaco and Normand-Cyrot [16], we say that the system

(9) is Minimum-Phase if the strongly unobservable part

$$\begin{aligned} z_{21}(k+1) &= z_{22}(k) \\ &\vdots \\ z_{2,m-1}(k+1) &= z_{2m}(k) \\ z_{2m}(k+1) &= \frac{-f_0(T^{-1}(\mathbf{z}(k))) + r(k-d+1)}{g_0(T^{-1}(\mathbf{z}(k)))} \end{aligned}$$

has an asymptotically stable equilibrium at the origin when $z_{11} = z_{12} = \cdots = z_{1n} = 0$ and $r = 0$, i.e., when the plant output and the reference command are restricted to be zero.

Example 2.1

This example is used to illustrate the transformation process. For $n = m = d = 2$, the system is

$$\begin{aligned} y_{k+1} &= f_0(y_k, y_{k-1}, u_{k-2}, u_{k-3}) \\ &+ g_0(y_k, y_{k-1}, u_{k-2}, u_{k-3})u_{k-1}. \end{aligned}$$

We select $n + m + d - 1 = 5$ states as

$$\begin{aligned} x_1(k) &= y_{k-1} \\ x_2(k) &= y_k \\ x_3(k) &= u_{k-3} \\ x_4(k) &= u_{k-2} \\ x_5(k) &= u_{k-1}, \end{aligned}$$

and the state space model is

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= f_0(x_2(k), x_1(k), x_4(k), x_3(k)) + \end{aligned}$$

$$
\begin{aligned}
& g_0(x_2(k), x_1(k), x_4(k), x_3(k))x_5(k) \\
= \; & f_0(x_1(k), \ldots, x_4(k)) + g_0(x_1(k), \ldots, x_4(k))x_5(k) \\
x_3(k+1) = \; & x_4(k) \\
x_4(k+1) = \; & x_5(k) \\
x_5(k+1) = \; & u_k \\
y(k) = \; & x_2(k).
\end{aligned}
$$

The purpose of the next step is to bring out the control $u(k)$ explicitly.

$$
\begin{aligned}
x_2(k+2) = \; & f_0(x_2(k+1), x_1(k+1), x_4(k+1), x_3(k+1)) + \\
& g_0(x_2(k+1), x_1(k+1), x_4(k+1), x_3(k+1))x_5(k+1) \\
= \; & f_0(f_0(x_1(k), \ldots, x_4(k)) + g_0(x_1(k), \ldots, x_4(k))x_5(k), \\
& x_2(k), x_5(k), x_4(k)) + \\
& g_0(f_0(x_1(k), \ldots, x_4(k)) + g_0(x_1(k), \ldots, x_4(k))x_5(k), \\
& x_2(k), x_5(k), x_4(k))u(k) \\
= \; & f_1(x_1(k), \ldots, x_5(k)) + g_1(x_1(k), \ldots, x_5(k))u(k).
\end{aligned}
$$

Then, after the state transformation

$$
\mathbf{z}(k) = \begin{bmatrix} \mathbf{z}_1(k) \\ \mathbf{z}_2(k) \end{bmatrix} = \begin{bmatrix} z_{11}(k) \\ z_{12}(k) \\ z_{13}(k) \\ z_{21}(k) \\ z_{22}(k) \end{bmatrix} = \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_2(k+1) \\ x_3(k) \\ x_4(k) \end{bmatrix} = \begin{bmatrix} T_1(\mathbf{x}(k)) \\ T_2(\mathbf{x}(k)) \end{bmatrix},
$$

we get

$$
\begin{aligned}
z_{11}(k+1) = \; & z_{12}(k) \\
z_{12}(k+1) = \; & z_{13}(k) \\
z_{13}(k+1) = \; & f_1(\mathbf{x}(k)) + g_1(\mathbf{x}(k))x_5(k+1) \\
= \; & F(\mathbf{z}(k)) + G(\mathbf{z}(k))u(k)
\end{aligned}
$$

$$
\begin{aligned}
z_{21}(k+1) &= z_{22}(k) \\
z_{22}(k+1) &= u(k-1) \\
y(k) &= z_{12}(k). \qquad \square
\end{aligned}
$$

Before concluding this chapter, it remains to show that the inverse of the transformation (14), i.e. $\mathbf{x}(k) = T^{-1}(\mathbf{z}(k))$, exists. It suffices to show that the partial derivative $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is nonsingular.

$$
\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix}
1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & \ddots & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & * & * & g_0 & 0 & \cdots & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_1 & 0 & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_2 & 0 & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & \ddots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_{d-2} \\
0 & 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & \cdots & 0 & \ddots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0
\end{bmatrix} \tag{23}
$$

where $g_i \neq 0, 0 \leq i \leq d-2$. The matrix in (23) is nonsingular, since, after some row interchange, it becomes

$$
\begin{bmatrix}
1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\vdots & & \ddots & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & \cdots & 0 & \ddots & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & * & * & g_0 & 0 & \cdots & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_1 & 0 & \cdots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_2 & 0 & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & \ddots & 0 \\
* & * & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & * & * & g_{d-2}
\end{bmatrix}.
$$

3 Adaptive Control using Neural Networks

The purpose of this chapter is to introduce the adaptive control system. Convergence results will be provided in chapters **4** and **5**. Two different methods for applying neural networks to adaptive control of unknown nonlinear systems will be described in sections **3.1** and **3.2**, respectively. In **3.1** the neural network is used to model the plant, and it needs to go through the same transformation described in chapter **2** in order to bring out the control. In **3.2** the neural network is used to model the transformed plant directly. A comparison of these two approaches will be given in section **3.3**.

Although neural networks can model any nonlinear function to any desired accuracy (see section **1.2**), there is no result about how many neurons should be used to achieve that accuracy. In practice, given a nonlinear plant, some identification process is needed to determine a suitable neural network size for modeling the plant. The size of the error between the plant and the network model may also be available from the identification process (see section **6.1**). In chapter **4**, we are going to assume that the nonlinear plant can be exactly modeled by a multi-layer neural network. However, in chapter **5**, some error between the plant and the model is allowed. Thus, the analysis in chapter **5** incorporates a robustness result.

3.1 Method 1

Rewrite the system

$$\begin{aligned} y_{k+1} &= f_0(y_k, y_{k-1}, \cdots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \cdots, u_{k-d-m+1}) \\ &\quad + g_0(y_k, y_{k-1}, \cdots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \cdots, u_{k-d-m+1})u_{k-d+1} \end{aligned}$$

as

$$y_{k+1} = f_0(x_1(k), \ldots, x_{n+m}(k)) + g_0(x_1(k), \ldots, x_{n+m}(k))u_{k-d+1} \qquad (24)$$

We propose to use a layered neural network

$$\hat{y}_{k+1} = \hat{f}_0(\mathbf{x}(k), \mathbf{w}) + \hat{g}_0(\mathbf{x}(k), \mathbf{v})u_{k-d+1} \tag{25}$$

to model the system (24), where $\mathbf{w}$ and $\mathbf{v}$ are vectors containing variable weights in the neural network. The neural network can have as many nonlinear hidden layers as desired.

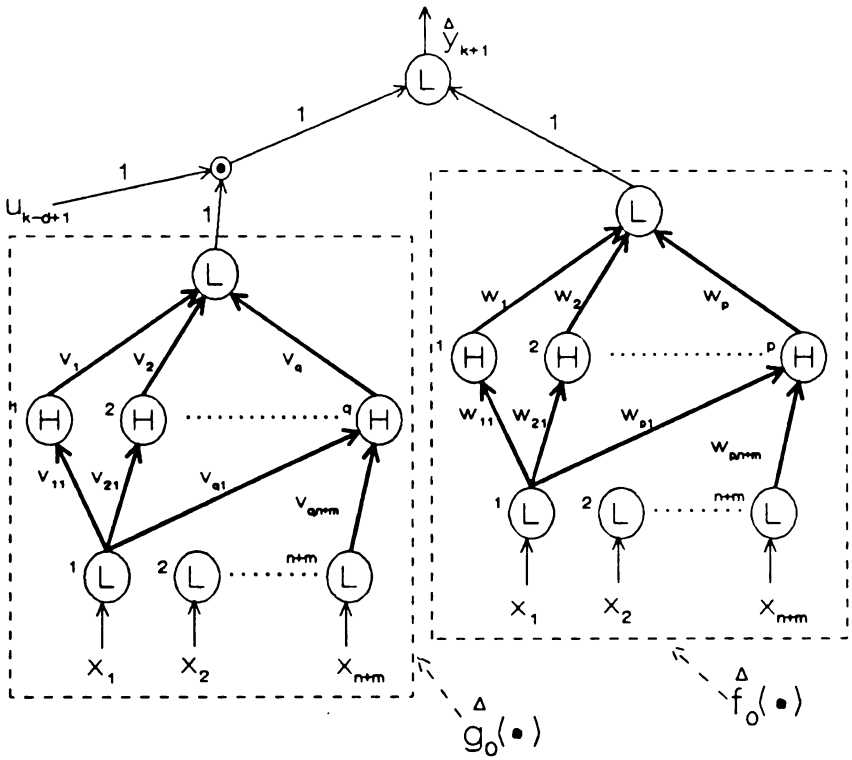


Figure 3.1 The neural network model

Figure 3.1 shows the architecture of a neural network model with one hidden layer in $\hat{f}_0$ and $\hat{g}_0$. The neurons labeled "L" are linear ones which can scale or shift the sum of incoming signals. The nonlinear neurons, which are labeled "H", employ the Hyperbolic Tangent Function h,

$$h(x) = (e^x - e^{-x})/(e^x + e^{-x}),$$

as their transfer function. The mathematical descriptions of $\hat{f}_0$ and $\hat{g}_0$ with one hidden layer are

$$\hat{f}_0(\mathbf{x}(k), \mathbf{w}) = \sum_{i=1}^{p} w_i \frac{e^{\sum_{j=1}^{n+m} w_{ij} x_j(k) + \hat{w}_i} - e^{-\sum_{j=1}^{n+m} w_{ij} x_j(k) + \hat{w}_i}}{e^{\sum_{j=1}^{n+m} w_{ij} x_j(k) + \hat{w}_i} + e^{-\sum_{j=1}^{n+m} w_{ij} x_j(k) + \hat{w}_i}} \tag{26}$$

and

$$\hat{g}_0(\mathbf{x}(k), \mathbf{v}) = \sum_{i=1}^{q} v_i \frac{e^{\sum_{j=1}^{n+m} v_{ij} x_j(k) + \hat{v}_i} - e^{-\sum_{j=1}^{n+m} v_{ij} x_j(k) + \hat{v}_i}}{e^{\sum_{j=1}^{n+m} v_{ij} x_j(k) + \hat{v}_j} + e^{-\sum_{j=1}^{n+m} v_{ij} x_j(k) + \hat{v}_i}} \tag{27}$$

The weights $\hat{w}_1, \ldots, \hat{w}_p$ and $\hat{v}_1, \ldots, \hat{v}_q$ in (26) and (27), which are not shown in Figure 3.1, are the bias weights, each attached to a corresponding nonlinear neuron. A neural network with any number of hidden layers can be described mathematically by iterative substitution from the output layer toward the input layer, although the final expression can be very complex.

Let $\theta = [\mathbf{w} \quad \mathbf{v}]$. At time step k, the neural network weights are denoted by $\theta(k)$ and the estimated output is

$$y^*_{k+1} = \hat{f}_0(\mathbf{x}(k), \mathbf{w}(k)) + \hat{g}_0(\mathbf{x}(k), \mathbf{v}(k)) u_{k-d+1} \tag{28}$$

The control algorithm is described as follows.

At each time step,

1. Calculate the control from the current states of the model (28), and apply it to the plant (24) and the model (28). Section **3.1.1** describes how to calculate the control.

2. Update the parameters $\theta(k)$ using the error between the plant output (24) and the model output (28). The updating rule is provided in section **3.1.2**.

If there are parameter errors, the output error may be observed. The output error is then used to reduce the parameter errors in order to produce better controls. This is a recursive process.

3.1.1 Control Law for Method 1

The general procedure for calculating the control is given in chapter **2**, where the transformation is performed on the plant. In the adaptive control system, the transformation is performed on the model, which is a neural network. Next example shows how to use the neural network model to generate controls.

Example 3.1

Let us revisit Example **2.1**. The unknown plant is

$$\begin{aligned} y_{k+1} &= f_0(x_1(k),\ldots,x_4(k)) \\ &\quad + g_0(x_1(k),\ldots,x_4(k))u_{k-1}. \end{aligned}$$

The neural network model of the plant is

$$\begin{aligned} y^*_{k+1} &= \hat{f}_0(x_1(k),\ldots,x_4(k),\mathbf{w}(k)) \\ &\quad + \hat{g}_0(x_1(k),\ldots,x_4(k),\mathbf{v}(k))u_{k-1}. \end{aligned}$$

which is shown in Figure 3.2.

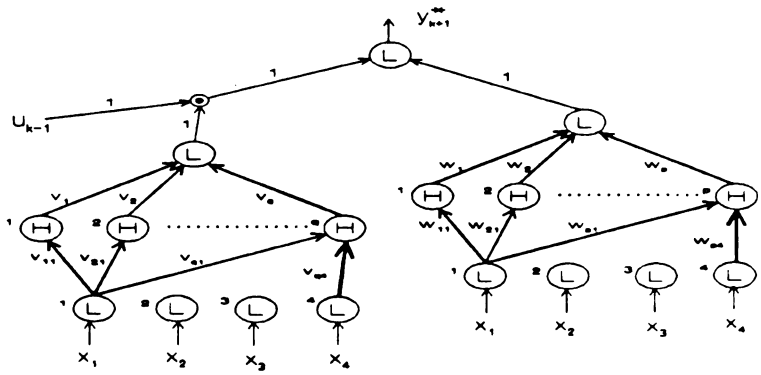


Figure 3.2 The neural network model (see example 3.1)

In order to bring out u_k, the following transformation is performed.

$$y^*_{k+2} = \hat{f}_0(\hat{x}_2(k+1), x_1(k+1), x_4(k+1), x_3(k+1), \mathbf{w}(k))$$

$$
\begin{aligned}
& + \hat{g}_0(\hat{x}_2(k+1), x_1(k+1), x_4(k+1), x_3(k+1), \mathbf{v}(k)) x_5(k+1) \\
= \; & \hat{f}_0(\hat{f}_0(x_1(k), \ldots, x_4(k), \mathbf{w}(k)) + \hat{g}_0(x_1(k), \ldots, x_4(k), \mathbf{v}(k)) x_5(k), \quad (29) \\
& x_2(k), x_5(k), x_4(k), \mathbf{w}(k)) + \\
& \hat{g}_0(\hat{f}_0(x_1(k), \ldots, x_4(k), \mathbf{w}(k)) + \hat{g}_0(x_1(k), \ldots, x_4(k), \mathbf{v}(k)) x_5(k), \\
& x_2(k), x_5(k), x_4(k), \mathbf{v}(k)) u_k \\
= \; & \hat{f}_1(x_1(k), \ldots, x_5(k), \theta(k)) + \hat{g}_1(x_1(k), \ldots, x_5(k), \theta(k)) u_k.
\end{aligned}
$$

where $\theta(k) = [\mathbf{w}(k) \;\; \mathbf{v}(k)]$ and $\hat{x}_2(k+1)$ is the estimation of $x_2(k+1)$.

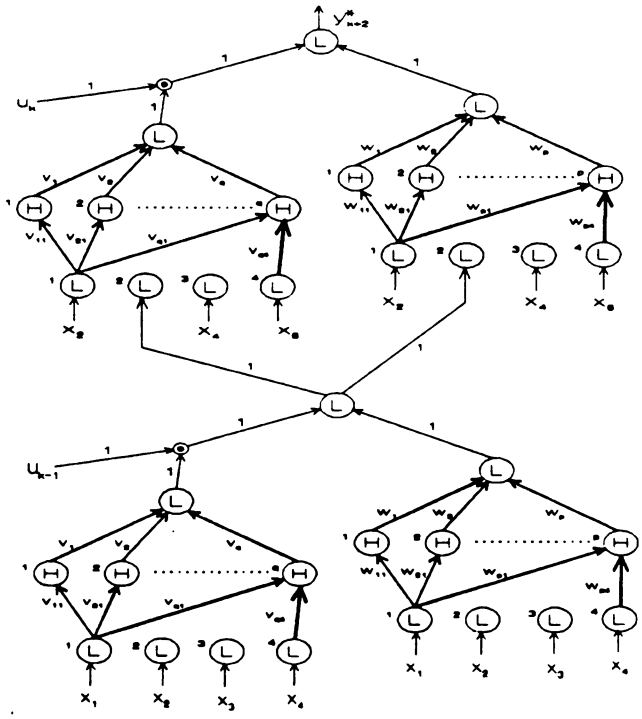


Figure 3.3 See example 3.1 for description

Notice that at the second equality of (29), the output of the network becomes one of its inputs. Therefore, the transformation process can be realized by duplicating and reconnecting the neural network model. Figure 3.3 shows the idea. The functions $\hat{f}_1$

and $\hat{g}_1$ are available from the network in Figure 3.3, and the control can be generated to be

$$u_k = \frac{-\hat{f}_1(\cdot) + r(k)}{\hat{g}_1(\cdot)} \qquad \square$$

The procedure used in Example **3.1** can be generalized to the general case. The calculation of the transformation can be a time consuming task for digital computers. It has been observed that the digital computer is a serious bottleneck when applying computed-torque techniques to robotics control [30]. On the other hand, neural networks, with its massive parallel computing capability, should be able to handle the computation efficiently, provided adequate hardware implementations are available.

3.1.2 Updating Rule for Method 1

Define the cost function to be

$$J_k = (y^*_{k+1} - y_{k+1})^2$$

The effect of adjusting weights on the cost function can be revealed by the following gradient:

$$\nabla_{\theta(k)} J_k = 2(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix}$$

The weights are updated as follows:

$$\begin{aligned} \theta(k+1) &= \theta(k) - \frac{\mu}{2r_k} \nabla_{\theta(k)} J_k \\ &= \theta(k) - \frac{\mu}{r_k}(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} \end{aligned} \tag{30}$$

where μ is a positive constant and

$$r_k = 1 + \left\| \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k+d-1} \end{bmatrix} \right\|^2 .$$

3.2 Method 2

In chapter **2** the plant

$$y_{k+1} = f_0(x_1(k), \ldots, x_{n+m}(k)) + g_0(x_1(k), \ldots, x_{n+m}(k))u_{k-d+1}(k)$$

is transformed into

$$y_{k+d} = f_{d-1}(x_1(k), \ldots, x_{n+m+d-1}(k)) + g_{d-1}(x_1(k), \ldots, x_{n+m+d-1}(k))u_k \quad (31)$$

Here the transformed plant is modeled by the neural network

$$\hat{y}_{k+d} = \hat{f}_{d-1}(x_1(k), \ldots, x_{n+m+d-1}(k), \mathbf{w}) + \hat{g}_{d-1}(x_1(k), \ldots, x_{n+m+d-1}(k), \mathbf{v})u_k \quad (32)$$

which is shown in Figure 3.4.

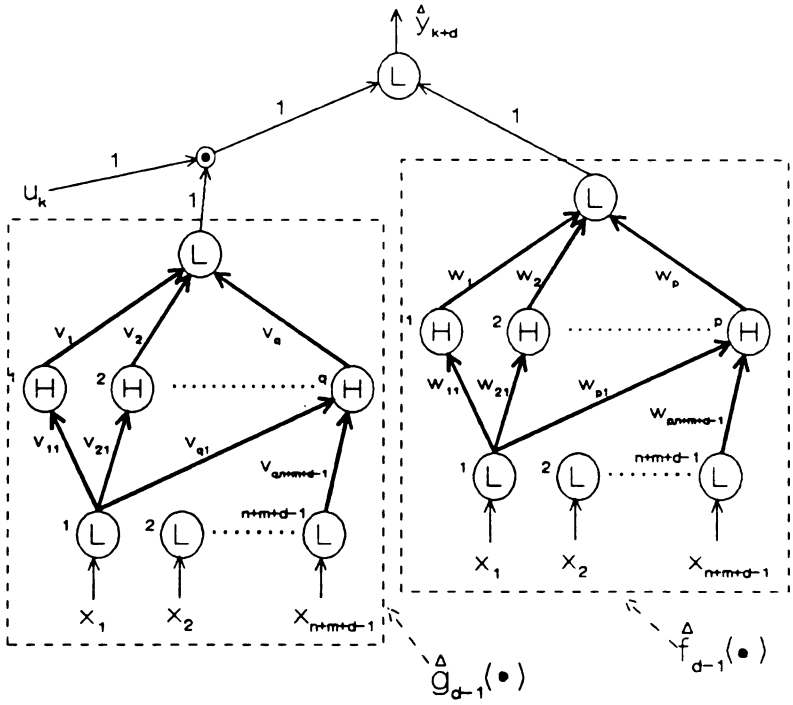


Figure 3.4 The neural network model for the transformed plant

Similar to the control algorithm in Method 1, at each time step a control is applied to the plant and the model. Then the network weights are updated according to the

observed error. However, the control law and the updating rule are slightly different as explained next.

3.2.1 Control Law for Method 2

At each time step the estimate of the transformed plant is

$$y^*_{k+d} = \hat{f}_{d-1}(x_1(k),\ldots,x_{n+m+d-1}(k),\mathbf{w}(k)) + \hat{g}_{d-1}(x_1(k),\ldots,x_{n+m+d-1}(k),\mathbf{v}(k))u_k \tag{33}$$

The control is defined straightforwardly from (33) as

$$u_k = \frac{-\hat{f}_{d-1}(\cdot) + r(k)}{\hat{g}_{d-1}(\cdot)} \tag{34}$$

3.2.2 Updating Rule for Method 2

Rewrite (31) and (32) as

$$\begin{aligned} y_{k+1} &= \hat{f}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1)) \\ &\quad + \hat{g}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1))u_{k-d+1} \end{aligned} \tag{35}$$

and

$$\begin{aligned} \hat{y}_{k+1} &= \hat{f}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1),\mathbf{w}) \\ &\quad + \hat{g}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1),\mathbf{v})u_{k-d+1} \end{aligned} \tag{36}$$

Calculate the estimated plant output using current network weights as

$$\begin{aligned} y^*_{k+1} &= \hat{f}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1),\mathbf{w}(k)) \\ &\quad + \hat{g}_{d-1}(x_1(k-d+1),\ldots,x_{n+m+d-1}(k-d+1),\mathbf{v}(k))u_{k-d+1} \end{aligned} \tag{37}$$

Define the cost function to be

$$J_k = (y^*_{k+1} - y_{k+1})^2$$

The effect of adjusting weights on the cost function can be revealed by the following gradient:

$$\nabla_{\theta(k)} J_k = 2(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix}$$

The weights are updated as follows:

$$\begin{aligned} \theta(k+1) &= \theta(k) - \frac{\mu}{2r_k} \nabla_{\theta(k)} J_k \\ &= \theta(k) - \frac{\mu}{r_k}(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} \end{aligned} \tag{38}$$

where μ is a positive constant and

$$r_k = 1 + \left\| \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} \right\|^2 .$$

3.3 Comparison between Method 1 and Method 2

Method 2 is simpler and more direct compared with Method 1. In Method 2, Only one neural network architecture is needed for generating controls and updating weights. In Method 1, two networks are needed: one for updating weights and the other for generating controls. For relative degree one system, Method 1 and Method 2 are the same.

The two methods introduced here have been standard algorithms for linear systems. The first method is described in [19] and the second method appears in [27].

4 Convergence Result : Part One

The adaptive control systems and the related algorithms have been introduced in chapter **3**. Two different convergence results based on different assumptions will be shown in this and next chapters. In this chapter we consider adaptive regulation problem for a single-input/single-output relative-degree-one system

$$\begin{aligned} y_{k+1} &= f_0(y_k, y_{k-1}, \ldots, y_{k-n+1}, u_{k-1}, u_{k-2}, \ldots, u_{k-m}) \\ &+ g_0(y_k, y_{k-1}, \ldots, y_{k-n+1}, u_{k-1}, u_{k-2}, \ldots, u_{k-m})u_k \end{aligned} \tag{39}$$

Important assumptions about the system are listed here.

Assumptions

1. f_0 vanishes at the origin, i.e., $f_0 = 0$ when the arguments of f_0 are all zeros.

2. g_0 is bounded away from zero.

3. This system is minimum phase. By that we mean the zero dynamics

$$\begin{aligned} z_{21}(k+1) &= z_{22}(k) \\ &\vdots \\ z_{2,m-1}(k+1) &= z_{2m}(k) \\ z_{2m}(k+1) &= -\frac{F(0, \mathbf{z}_2(k), \mathbf{w})}{G(0, \mathbf{z}_2(k), \mathbf{v})} \end{aligned}$$

has an globally exponentially stable equilibrium point at the origin, and there exists a Lyapunov function $V_2(\mathbf{z}_2(k))$ such that

$$c_1 |\mathbf{z}_2(k)|^2 \leq V_2(\mathbf{z}_2(k)) \leq c_2 |\mathbf{z}_2(k)|^2, \tag{40}$$

$$V_2(\mathbf{z}_2(k+1)) - V_2(\mathbf{z}_2(k)) \leq -\alpha |\mathbf{z}_2(k)|^2, \text{ and} \tag{41}$$

$$\left| \frac{\partial V_2(\mathbf{x})}{\partial \mathbf{x}} \right| \leq L |\mathbf{x}|. \tag{42}$$

4. The nonlinear functions f_0 can be exactly represented by a multi-layer neural network $\hat{f}_0$ which does not have bias weights. In the case of a three-layer neural network,

$$f_0(\mathbf{x}(k)) = \hat{f}_0(\mathbf{x}(k), \mathbf{w}) = \sum_{i=1}^{p} w_i H(\sum_{j=1}^{m+n} w_{ij} x_j) \tag{43}$$

where H is the *hyperbolic tangent* function.

5. The function g_0 can be exactly represented by a multi-layer neural network without bias weights connected to nonlinear neurons. However, there is a bias weight added to the linear neuron at the output layer. In the case of a three-layer neural network,

$$g_0(\mathbf{x}(k)) = \hat{g}_0(\mathbf{x}(k), \mathbf{v}) = v_0 + \sum_{i=1}^{q} v_i H(\sum_{j=1}^{m+n} v_{ij} x_j) \tag{44}$$

The $\mathbf{w}$ and $\mathbf{v}$ in (43) and (44) are vectors containing variable weights in the neural networks. Now the plant (39) can be written as

$$y_{k+1} = \hat{f}_0(\mathbf{x}(k), \mathbf{w}) + \hat{g}_0(\mathbf{x}(k), \mathbf{v}) u_k \tag{45}$$

The estimate of the plant is

$$y^*_{k+1} = \hat{f}_0(\mathbf{x}(k), \mathbf{w}(k)) + \hat{g}_0(\mathbf{x}(k), \mathbf{v}(k)) u_k \tag{46}$$

There is no theoretical evidence about how good three-layer neural networks without bias weights can approximate nonlinear functions. But it is for sure that they can deal with certain classes of systems. Some evidence from simulation will be provided in chapter **6**. The Method 1 and Method 2 described in chapter **3** are the same for relative-degree-one systems. After the transformation described in chapter **2**, the plant (39) becomes

$$
\begin{aligned}
z_{11}(k+1) &= z_{12}(k) \\
&\vdots \\
z_{1n}(k+1) &= \hat{f}_0(\mathbf{x}(k),\mathbf{w}) + \hat{g}_0(\mathbf{x}(k),\mathbf{v})x_{n+m}(k+1) \\
&= F(\mathbf{z}(k),\mathbf{w}) + G(\mathbf{z}(k),\mathbf{v})u(k) \qquad (47) \\
z_{21}(k+1) &= z_{22}(k) \\
&\vdots \\
z_{2,m-1}(k+1) &= z_{2m}(k) \\
z_{2m}(k+1) &= u(k) \qquad (48) \\
y(k) &= z_{1n}(k).
\end{aligned}
$$

It is convenient to recall that (see chapter **2**)

$$z_{1n}(k) = y(k), \ldots\ldots, \; z_{11}(k) = y(k-n+1),$$

$$z_{2m}(k) = u(k-1), \ldots\ldots, \; z_{21}(k) = u(k-m).$$

The purpose of the control is to regulate the plant output to zero asymptotically. At each time step, the control

$$u_k = -\frac{F(\mathbf{z}(k),\mathbf{w}(k))}{G(\mathbf{z}(k),\mathbf{v}(k))} \qquad (49)$$

is applied to the plant. Then the weights are updated.

$$\theta(k+1) = \theta(k) - \frac{\mu}{r_k}(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_k \end{bmatrix} \qquad (50)$$

Notice that y^*_{k+1} in (50) equals zero, because the control (49), which is calculated from the model, can exactly cancel the model dynamics. Therefore, the updating rule is rewritten as

$$\theta(k+1) = \theta(k) + \frac{\mu}{r_k} y_{k+1} \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_k \end{bmatrix} \qquad (51)$$

Next a local convergence result is provided.

Theorem 1 *Under the gradient-descent updating rule (51) and the assumptions 1 - 5, given any initial condition* $\mathbf{x}(0)$*, there exists a positive constant* R *such that if*

$$|\theta - \theta(0)| \leq R,$$

then

$$y_k \textit{ will asymptotically converge to zero.}$$

Proof:

step 1. The closed-loop control system.

Substituting u_k defined in (49) and u_{k-d+1} into (39) and (47), one gets

$$\begin{aligned}
z_{11}(k+1) &= z_{12}(k) \\
&\vdots \\
z_{1n}(k+1) &= \left[F(\mathbf{z}(k),\mathbf{w}) + G(\mathbf{z}(k),\mathbf{v})(-\frac{F(\mathbf{z}(k),\mathbf{w}(k))}{G(\mathbf{z}(k),\mathbf{v}(k))})\right]_1 \qquad (52) \\
z_{21}(k+1) &= z_{22}(k) \\
&\vdots \\
z_{2,m-1}(k+1) &= z_{2m}(k) \\
z_{2m}(k+1) &= -\frac{F(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{w}(k))}{G(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{v}(k))} \\
y(k) &= z_{1n}(k).
\end{aligned}$$

The functions F and G in (52) and their derivatives are continuously differentiable infinitely many times. The term $[\bullet]_1$ vanishes when $\theta(k) = \theta$ and the derivative of $\frac{F(\mathbf{z}(k),\mathbf{w})}{G(\mathbf{z}(k),\mathbf{v})}$ with respect to θ is zero when $\mathbf{z}(k) = 0$. Using these properties, we have

$$\begin{aligned}
[\bullet]_1 &= \left[F(\mathbf{z}(k),\mathbf{w}) + G(\mathbf{z}(k),\mathbf{v})(-\frac{F(\mathbf{z}(k),\mathbf{w}(k))}{G(\mathbf{z}(k),\mathbf{v}(k))})\right] \\
&= \left[F(\mathbf{z}(k),\mathbf{w}) + G(\mathbf{z}(k),\mathbf{v})(-\frac{F(\mathbf{z}(k),\mathbf{w}(k))}{G(\mathbf{z}(k),\mathbf{v}(k))})\right]
\end{aligned}$$

$$
\begin{aligned}
& - \left[F(\mathbf{z}(k), \mathbf{w}) + G(\mathbf{z}(k), \mathbf{v})(-\frac{F(\mathbf{z}(k), \mathbf{w})}{G(\mathbf{z}(k), \mathbf{v})}) \right] \\
= \; & G(\mathbf{z}(k), \theta) \left[\frac{F(\mathbf{z}(k), \mathbf{w})}{G(\mathbf{z}(k), \mathbf{v})} - \frac{F(\mathbf{z}(k), \mathbf{w}(k))}{G(\mathbf{z}(k), \mathbf{v}(k))} \right] \\
= \; & G(\mathbf{z}(k), \theta) \frac{\partial}{\partial \theta} \left[\frac{F(\mathbf{z}(k), \mathbf{w})}{G(\mathbf{z}(k), \mathbf{v})} \right] \Bigg|_{\theta + (1-\zeta)(\theta(k) - \theta)} (\theta(k) - \theta)
\end{aligned}
$$

where the last equality follows from the Mean Value Theorem [28].

Therefore

$$
|[\, \bullet \,]_1| \leq k_1 \left| \tilde{\theta}(k) \right| \cdot |\mathbf{z}(k)|, \quad \textit{where } \tilde{\theta}(k) = \theta(k) - \theta \tag{53}
$$

step 2. To choose a Lyapunov function associated with $\mathbf{z}_1$.

$$
\mathbf{z}_1(k+1) = A\mathbf{z}_1(k) + \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} [\, \bullet \,]_1, \quad \textit{where } A = \begin{bmatrix} 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}.
$$

A is a stable matrix (since all eigenvalues are at the origin). $\Longrightarrow$ Given any symmetric $Q > 0$, $\exists$ a symmetric $P > 0$ such that $A'PA - P = -Q$ [31]. Choose the Lyapunov function

$$
V_1(\mathbf{z}_1(k)) = \mathbf{z}_1'(k) P \mathbf{z}_1(k),
$$

Then, using (53),

$$
\begin{aligned}
V_1(\mathbf{z}_1(k+1)) - V_1(\mathbf{z}_1(k)) \; = \; & -\mathbf{z}_1'(k) Q \mathbf{z}_1(k) + 2\mathbf{z}_1(k)' A' P \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} [\, \bullet \,]_1 \\
& + [\, \bullet \,]_1^2 [0 \cdots 1] P \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} \qquad (54) \\
\leq \; & -\mathbf{z}_1'(k) Q \mathbf{z}_1(k) + k_3 \left| \tilde{\theta}(k) \right| |\mathbf{z}(k)|^2 + k_4 \left| \tilde{\theta}(k) \right|^2 |\mathbf{z}(k)|^2.
\end{aligned}
$$

step 3. To choose a Lyapunov function associated with $\mathbf{z}_2$.

The plant dynamics associated with $\mathbf{z}_2$ is

$$\begin{aligned}
z_{21}(k+1) &= z_{22}(k) \\
&\vdots \\
z_{2,m-1}(k+1) &= z_{2m}(k) \\
z_{2m}(k+1) &= \frac{-F(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{w}(k))}{G(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{v}(k))} \\
&= \frac{-F(0,\mathbf{z}_2(k),\mathbf{w}(k))}{G(0,\mathbf{z}_2(k),\mathbf{v}(k))} \\
&\quad + \left[\frac{-F(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{w}(k))}{G(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{v}(k))} - \frac{-F(0,\mathbf{z}_2(k),\mathbf{w}(k))}{G(0,\mathbf{z}_2(k),\mathbf{v}(k))}\right]_a \\
&= \frac{-F(0,\mathbf{z}_2(k),\mathbf{w})}{G(0,\mathbf{z}_2(k),\mathbf{v})} \\
&\quad + \left[\frac{-F(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{w}(k))}{G(\mathbf{z}_1(k),\mathbf{z}_2(k),\mathbf{v}(k))} - \frac{-F(0,\mathbf{z}_2(k),\mathbf{w}(k))}{G(0,\mathbf{z}_2(k),\mathbf{v}(k))}\right]_a \\
&\quad + \left[\frac{F(0,\mathbf{z}_2(k),\mathbf{w}(k))}{G(0,\mathbf{z}_2(k),\mathbf{v}(k))} - \frac{-F(0,\mathbf{z}_2(k),\mathbf{w})}{G(0,\mathbf{z}_2(k),\mathbf{v})}\right]_b
\end{aligned}$$

By using similar techniques in showing (53), we can arrive at

$$|[\,\bullet\,]_a| \le c_3\,|\mathbf{z}_1(k)| \tag{55}$$

and

$$|[\,\bullet\,]_b| \le c_4\left|\tilde{\theta}(k)\right| \cdot |\mathbf{z}_2(k)|\,. \tag{56}$$

Let

$$S_k = \left[z_{22}(k),\ldots,z_{2m}(k),-\frac{F(0,\mathbf{z}_2(k),\mathbf{w})}{G(0,\mathbf{z}_2(k),\mathbf{v})}\right]'$$

and

$$Q_k = [0,\ldots,0,[\bullet]_a + [\bullet]_b]'$$

Applying (55), (56), (41), and (42), we have

$$V_2(\mathbf{z}_2(k+1)) - V_2(\mathbf{z}_2(k)) \;=\; V_2(S_k + Q_k) - V_2(\mathbf{z}_2(k))$$

$$
\begin{aligned}
&= [V_2(S_k) - V_2(\mathbf{z}_2(k))] + V_2(S_k + Q_k) - V_2(S_k) \\
&\leq -k_5 |\mathbf{z}_2(k)|^2 + \hat{k} |\zeta S_k + (1-\zeta) Q_k| \cdot |Q_k| \\
&\leq -k_5 |\mathbf{z}_2(k)|^2 + \hat{k}_1 |S_k| \cdot |Q_k| + \hat{k}_2 |Q_k|^2 \\
&\leq -k_5 |\mathbf{z}_2(k)|^2 + k_6 |\mathbf{z}_1(k)| \cdot |\mathbf{z}_2(k)| + k_7 |\mathbf{z}_1(k)|^2 \\
&\quad + c_1' |\mathbf{z}_2(k)|^2 |\tilde{\theta}(k)| + c_2' |\mathbf{z}_1(k)| \cdot |\mathbf{z}_2(k)| \left|\tilde{\theta}(k)\right| \\
&\quad + c_3' |\mathbf{z}_2(k)|^2 |\tilde{\theta}(k)|^2 \\
&\leq -k_5 |\mathbf{z}_2(k)|^2 + k_6 |\mathbf{z}_1(k)| \cdot |\mathbf{z}_2(k)| + k_7 |\mathbf{z}_1(k)|^2 \\
&\quad + k_{11} |\mathbf{z}(k)|^2 |\tilde{\theta}(k)| + k_{12} |\mathbf{z}(k)|^2 |\tilde{\theta}(k)|^2 \qquad (57)
\end{aligned}
$$

step 4. To combine step 3 and step 4.

Let $V(\mathbf{z}(k)) = V_1(\mathbf{z}_1(k)) + \beta V_2(\mathbf{z}_2(k))$. Then, from (54) and (57),

$$
\begin{aligned}
V(\mathbf{z}(k+1)) - V(\mathbf{z}(k)) &\leq -\mathbf{z}_1'(k) Q \mathbf{z}_1(k) + k_3 \left|\tilde{\theta}(k)\right| |\mathbf{z}(k)|^2 + k_4 \left|\tilde{\theta}(k)\right|^2 |\mathbf{z}(k)|^2 \\
&\quad -\beta k_5 |\mathbf{z}_2(k)|^2 + \beta k_6 |\mathbf{z}_1(k)| |\mathbf{z}_2(k)| + \beta k_7 |\mathbf{z}_1(k)|^2 \\
&\quad +\beta k_{11} |\mathbf{z}(k)|^2 |\tilde{\theta}(k)| + \beta k_{12} |\mathbf{z}(k)|^2 |\tilde{\theta}(k)|^2 \\
&\leq [(-k_{10} + \beta k_7) |\mathbf{z}_1|^2 - \beta k_5 |\mathbf{z}_2(k)|^2 + \beta k_6 |\mathbf{z}_1(k)| |\mathbf{z}_2(k)|] \\
&\quad +(k_3 + \beta k_{11}) \left|\tilde{\theta}(k)\right| |\mathbf{z}(k)|^2 + (k_4 + \beta k_{12}) \left|\tilde{\theta}(k)\right|^2 |\mathbf{z}(k)|^2 \\
&\leq -k_1' |\mathbf{z}(k)|^2 + k_3' \left|\tilde{\theta}(k)\right| |\mathbf{z}(k)|^2 + k_4' \left|\tilde{\theta}(k)\right|^2 |\mathbf{z}(k)|^2 . \qquad (58)
\end{aligned}
$$

The last inequality is true if β is small enough.

step 5. A Lyapunov-type function related to weight convergence.

Rewrite the updating rule (51) as

$$
\theta(k+1) = \theta(k) + \frac{\mu}{r_k} \Delta_k
$$

where

$$
\Delta_k = y_{k+1} \begin{bmatrix} \left(\frac{\partial \hat{f}_0(\mathbf{x}(k), \mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_0(\mathbf{x}(k), \mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' \end{bmatrix} u_k
$$

Define $\tilde{\theta}(k) = \theta(k) - \theta$, then

$$\tilde{\theta}(k+1) = \tilde{\theta}(k) + \frac{\mu}{r_k}\triangle_k,$$

and the inner product of $\tilde{\theta}_{k+1}$ by itself is

$$\tilde{\theta}(k+1)'\tilde{\theta}(k+1) = \tilde{\theta}(k)'\tilde{\theta}(k) + \frac{2\mu}{r_k}\tilde{\theta}(k)'\triangle_k + \frac{\mu^2}{r_k^2}\triangle_k'\triangle_k. \tag{59}$$

Notice that

$$y_{k+1} = \hat{f}_0(\mathbf{x}(k), \mathbf{w}) + \hat{g}_0(\mathbf{x}(k), \mathbf{v})u_k \tag{60}$$

and

$$0 = \hat{f}_0(\mathbf{x}(k), \mathbf{w}(k)) + \hat{g}_0(\mathbf{x}(k), \mathbf{v}(k))u_k \tag{61}$$

After subtracting (61) from (60), y_{k+1} can be rewritten as

$$y_{k+1} = -\tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\Gamma(k)=\theta+(1-\zeta)(\theta(k)-\theta)} \tag{62}$$

Now let us investigate the term $\tilde{\theta}(k)'\triangle_k$ in (59). It can be quickly verified that

$$\tilde{\theta}(k)'\triangle_k = \tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\theta(k)} y_{k+1}$$

Then, making use of (62), we have

$$\begin{aligned}
\tilde{\theta}(k)'\triangle_k &= \tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\theta(k)} y_{k+1} \\
&= \tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\Gamma(k)} y_{k+1} + \left[\tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\theta(k)} y_{k+1} - \tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\Gamma(k)} y_{k+1}\right] \\
&= -y_{k+1}^2 + \left[\tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\theta(k)} - \tilde{\theta}(k)' \left.\frac{\partial y_{k+1}}{\partial \theta}\right|_{\Gamma(k)}\right] y_{k+1} \qquad (63) \\
&= -y_{k+1}^2 + \left[\sum_{i=1}^{p} \tilde{w}_i \left(\left.\frac{\partial \hat{f}_0}{\partial w_i}\right|_{\theta(k)} - \left.\frac{\partial \hat{f}_0}{\partial w_i}\right|_{\Gamma(k)}\right) + \sum_{i=1}^{p}\sum_{j=1}^{m+n} \tilde{w}_{ij} \left(\left.\frac{\partial \hat{f}_0}{\partial w_{ij}}\right|_{\theta(k)} - \left.\frac{\partial \hat{f}_0}{\partial w_{ij}}\right|_{\Gamma(k)}\right)\right. \\
&\quad \left. + \left\{\sum_{i=1}^{q} \tilde{v}_i \left(\left.\frac{\partial \hat{g}_0}{\partial v_i}\right|_{\theta(k)} - \left.\frac{\partial \hat{g}_0}{\partial v_i}\right|_{\Gamma(k)}\right) \sum_{i=1}^{q}\sum_{j=1}^{m+n} \tilde{v}_{ij} \left(\left.\frac{\partial \hat{g}_0}{\partial v_{ij}}\right|_{\theta(k)} - \left.\frac{\partial \hat{g}_0}{\partial v_{ij}}\right|_{\Gamma(k)}\right)\right\} u_k\right] y_{k+1}
\end{aligned}$$

The functions and parameters used at the last equality of (63) are defined in (43) and (44). By virtue of special property of the *hyperbolic tangent* function, each $\tilde{w}\left(\frac{\partial \hat{f}_0}{\partial w}\Big|_{\theta(k)} - \frac{\partial \hat{f}_0}{\partial w}\Big|_{\Gamma(k)}\right)$ or $\tilde{v}\left(\frac{\partial \hat{g}_0}{\partial v}\Big|_{\theta(k)} - \frac{\partial \hat{g}_0}{\partial v}\Big|_{\Gamma(k)}\right)$ term is of the order $\left|\tilde{\theta}(k)\right|^2 \cdot |\mathbf{z}(k)|$. Moreover, the control u_k, as defined in (49), is of the order $|\mathbf{z}(k)|$ and y_{k+1} is of the order $\left|\tilde{\theta}(k)\right| \cdot |\mathbf{z}(k)|$ (see (53)).

So,

$$\tilde{\theta}(k)'\triangle_k \leq -y_{k+1}^2 + k' \left|\tilde{\theta}(k)\right|^3 (|\mathbf{z}(k)|^2 + |\mathbf{z}(k)|^3). \tag{64}$$

Next check about $\frac{\mu^2}{r_k^2}\triangle_k'\triangle_k$ in (59).

$$\frac{\mu^2}{r_k^2}\triangle_k'\triangle_k = \frac{\mu^2}{r_k^2}\left\| \begin{bmatrix} \left(\frac{\partial \hat{F}(T(\mathbf{x}(k)),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{G}(T(\mathbf{x}(k)),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' \end{bmatrix} u_k \right\|^2 y_{k+1}^2.$$

Define

$$r_k = 1 + \left\| \begin{bmatrix} \left(\frac{\partial \hat{F}(T(\mathbf{x}(k)),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{G}(T(\mathbf{x}(k)),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' \end{bmatrix} u_k \right\|^2.$$

Then

$$\frac{\mu^2}{r_k^2}\triangle_k'\triangle_k < \frac{\mu^2}{r_k} y_{k+1}^2.$$

It can be shown that r_k is bounded. Then, setting $\mu = 1$, the equation (59) becomes

$$\tilde{\theta}(k+1)'\tilde{\theta}(k+1) - \tilde{\theta}(k)'\tilde{\theta}(k) \leq -k_8 y_{k+1}^2 + k_9 \left|\tilde{\theta}(k)\right|^3 (|\mathbf{z}(k)|^2 + |\mathbf{z}(k)|^3). \tag{65}$$

Final step. A Lyapunov function for the overall system.

Choose the Lyapunov function

$$\tilde{V}(k) = \tilde{\theta}(k)'\tilde{\theta}(k) + \gamma^2 V(\mathbf{z}(k)).$$

By (58) and (65),

$$\begin{aligned} \tilde{V}(k+1) - \tilde{V}(k) \leq\ & -k_8 y_{k+1}^2 + k_9 \left|\tilde{\theta}(k)\right|^3 (|\mathbf{z}(k)|^2 + |\mathbf{z}(k)|^3) \\ & \gamma^2 \left(-k_1' |\mathbf{z}(k)|^2 + k_3' \left|\tilde{\theta}(k)\right| |\mathbf{z}(k)|^2 + k_4' \left|\tilde{\theta}(k)\right|^2 |\mathbf{z}(k)|^2\right). \end{aligned} \tag{66}$$

Suppose $|\mathbf{z}(0)| \leq K$. Then, by (40) and the definition of $V(\mathbf{z}(k))$, there exists a constant d_1 such that

$$V(\mathbf{z}(0)) \leq d_1 K^2.$$

Consider the set

$$S = \left\{ \begin{pmatrix} \mathbf{z} \\ \tilde{\theta} \end{pmatrix} \mid \tilde{\theta}'\tilde{\theta} + \gamma^2 V \leq c^2 \right\},$$

It can be verified that if γ is chosen to be

$$\gamma = \frac{c}{K\sqrt{2d_1}}, \tag{67}$$

then

$$\left|\tilde{\theta}(0)\right| \leq \frac{c}{\sqrt{2}} \; and \; |\mathbf{z}(0)| \leq K \Longrightarrow \begin{pmatrix} \mathbf{z}(0) \\ \tilde{\theta}(0) \end{pmatrix} \in S.$$

Next, we show that if c is chosen small enough, then the set S is an invariant set. For any $\begin{pmatrix} \mathbf{z}(k) \\ \tilde{\theta}(k) \end{pmatrix} \in S$,

$$\begin{aligned} \left|\tilde{\theta}(k)\right| &\leq c, \quad and \\ \left|\sqrt{V(\mathbf{z}(k))}\right| &\leq K\sqrt{2d_1}. \end{aligned} \tag{68}$$

Again, by (40) and the definition of $V(\mathbf{z}(k))$, there exists a constant d_2 such that

$$|\mathbf{z}(k)| \leq d_2 \left|\sqrt{V(\mathbf{z}(k))}\right| \leq d_2 K\sqrt{2d_1} \tag{69}$$

Substituting (67), (68) and (69) into (66), one gets

$$\begin{aligned} \tilde{V}(k+1) - \tilde{V}(k) \;\leq\; & -k_8 y_{k+1}^2 - |\mathbf{z}(k)|^2 \frac{k_1' c^2}{K^2(2d_1)} \cdot \\ & [1 - \frac{c}{k_1'}(k_3' + k_4' c + 2k_9 K^2 d_1 + 2d_1 k_9 K^3 d_2\sqrt{2d_1})]. \end{aligned} \tag{70}$$

It is obvious that there exists a c_0 such that if $c \leq c_0$, then (68) can be rewritten as

$$\tilde{V}(k+1) - \tilde{V}(k) \leq -k_8 y_{k+1}^2 - \tilde{k}\,|\mathbf{z}(k)|^2, \tag{71}$$

and the set S is an invariant set. Since $\begin{pmatrix} \mathbf{z}(0) \\ \tilde{\theta}(0) \end{pmatrix} \in S$, $\begin{pmatrix} \mathbf{z}(k) \\ \tilde{\theta}(k) \end{pmatrix} \in S \quad \forall k \geq 0.$

Finally, (71) implies that

$$\tilde{V}(k) \longrightarrow \bar{V} \quad as\ k \longrightarrow \infty\ , \tag{72}$$

and therefore

$$y_k \longrightarrow 0 \quad as\ k \longrightarrow \infty\ . \qquad \square \tag{73}$$

5 Convergence Result: Part Two

In this section we consider adaptive tracking problem for a single-input/single-output relative-degree-d system

$$\begin{aligned} y_{k+1} &= f_0(y_k, y_{k-1}, \ldots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \ldots, u_{k-d-m+1}) \\ &+ g_0(y_k, y_{k-1}, \ldots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \ldots, u_{k-d-m+1})u_{k-d+1} \end{aligned} \tag{74}$$

After the transformation described in section **2**, the plant (74) becomes

$$\begin{aligned} z_{11}(k+1) &= z_{12}(k) \\ &\vdots \\ z_{1n}(k+1) &= z_{1,n+1}(k) \\ &\vdots \\ z_{1,n+d-1}(k) &= f_{d-1}(\mathbf{x}(k)) + g_{d-1}(\mathbf{x}(k))x_{n+m+d-1}(k+1) \\ &= f_{d-1}(T^{-1}(\mathbf{z}(k))) + g_{d-1}(T^{-1}(\mathbf{z}(k)))u_k \\ &= F(\mathbf{z}(k)) + G(\mathbf{z}(k))u_k \end{aligned} \tag{75}$$

$$\begin{aligned} z_{21}(k+1) &= z_{22}(k) \\ &\vdots \\ z_{2,m-1}(k+1) &= z_{2m}(k) \\ z_{2m}(k+1) &= u_{k-d+1} \\ &= \frac{-f_0(T^{-1}(\mathbf{z}(k))) + r(k-d+1)}{g_0(T^{-1}(\mathbf{z}(k)))} \\ y(k) &= z_{1n}(k). \end{aligned} \tag{76}$$

Some assumptions about the plant are made here:

Assumption 1.

$g_{d-1}(\mathbf{x}(k))$ is bounded away from zero over any compact set. More precisely,

$$|g_{d-1}(\mathbf{x}(k))| \geq B > 0, \quad \forall \mathbf{x}(k) \in \Xi \tag{77}$$

where B is a known constant and Ξ is a compact subset of $R^{m+n+d-1}$.

Assumption 2. *The system is minimum phase.*

Setting $\mathbf{z}_1(k)$ and $r(k-d+1)$ in (76) to be zero, the dynamics associated with $\mathbf{z}_2(k)$ become

$$\begin{aligned} z_{21}(k+1) &= z_{22}(k) \\ &\vdots \\ z_{2,m-1}(k+1) &= z_{2m}(k) \\ z_{2m}(k+1) &= \frac{-f_0(T^{-1}(0,\mathbf{z}_2(k)))}{g_0(T^{-1}(0,\mathbf{z}_2(k)))} \end{aligned} \tag{78}$$

have equilibrium at C, where $C = [c,\ldots,c]'$, then

$$c = \frac{-f_0(T^{-1}(0,C))}{g_0(T^{-1}(0,C))}$$

After the state shift

$$e_{2i} = z_{2i} - c,$$

(78) is transformed into

$$\begin{aligned} e_{21}(k+1) &= e_{22}(k) \\ &\vdots \\ e_{2,m-1}(k+1) &= e_{2m}(k) \\ e_{2m}(k+1) &= \frac{-f_0(T^{-1}(0,\mathbf{e}_2(k)+C))}{g_0(T^{-1}(0,\mathbf{e}_2(k)+C))} - c \end{aligned} \tag{79}$$

The dynamics (79) are called the zero dynamics. By assuming that the system is minimum phase, we mean the zero dynamics have an asymptotically stable equilibrium point at the origin and that there exists a Lyapunov function $V_2(\mathbf{e}_2(k))$ such that

$$c_1 |\mathbf{e}_2(k)|^2 \leq V_2(\mathbf{e}_2(k)) \leq c_2 |\mathbf{e}_2(k)|^2, \tag{80}$$

$$V_2(\mathbf{e}_2(k+1)) - V_2(\mathbf{e}_2(k)) \leq -\alpha |\mathbf{e}_2(k)|^2, \text{ and} \tag{81}$$

$$\left|\frac{\partial V_2(\mathbf{x})}{\partial \mathbf{x}}\right| \leq L|\mathbf{x}|. \tag{82}$$

Rewrite the plant (75) in an input-output form as

$$y_{k+d} = f_{d-1}(\mathbf{x}(k)) + g_{d-1}(\mathbf{x}(k))u_k \tag{83}$$

The plant (83) is modeled by the neural network

$$\hat{y}_{k+d} = \hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}) + \hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v})u_k \tag{84}$$

In the case that (84) is a three-layer neural network, then

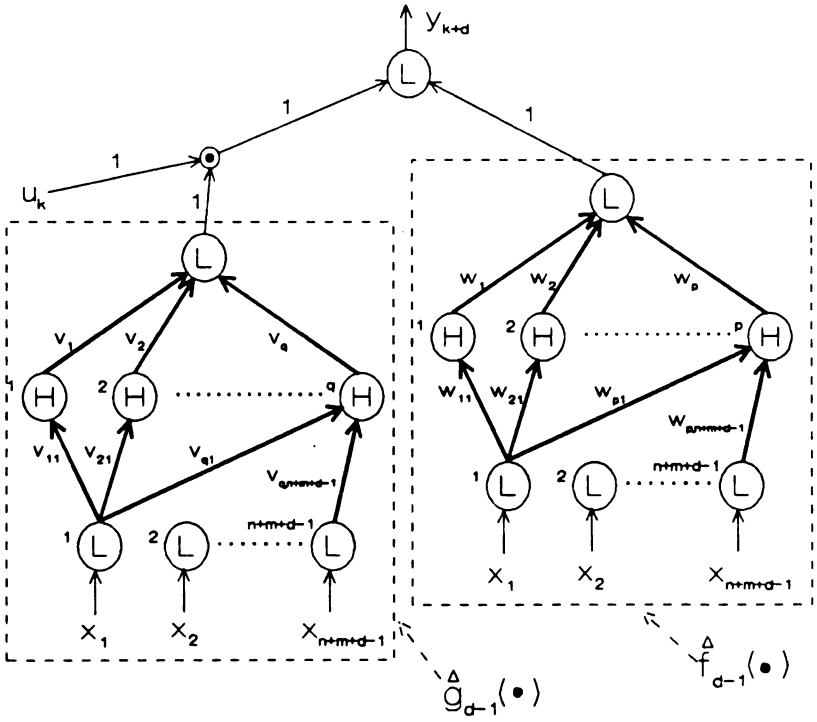


Figure 5.1 The neural network model for the transformed plant

$$\hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}) = \sum_{i=1}^{p} w_i H\Big(\sum_{j=1}^{m+n+d-1} w_{ij} x_j + \hat{w}_i\Big) \tag{85}$$

and

$$\hat{g}_{d-1}(\mathbf{x}(k),\mathbf{v}) = \sum_{i=1}^{q} v_i H\Big(\sum_{j=1}^{m+n+d-1} v_{ij}x_j + \hat{v}_i\Big) \tag{86}$$

The function H in (85) and (86) is the *hyperbolic tangent* function. The three-layer neural network model is shown in Figure 5.1.

Assumption 3.

Assume that there exist $\mathbf{w}$ *and* $\mathbf{v}$ *such that* $\hat{f}_{d-1}$ *and* $\hat{g}_{d-1}$ *can approximate* f_{d-1} *and* g_{d-1}, *which are continuous functions, to within* ϵ *accuracy over the compact set* Ξ, *i.e.,*

$$\exists\, \mathbf{w},\, \mathbf{v}\ s.t.\ \max\left|\hat{f}_{d-1}(\mathbf{x}(k),\mathbf{w}) - f_{d-1}(\mathbf{x}(k))\right| \ \leq\ \epsilon,\ \ \forall \mathbf{x}(k) \in \Xi \tag{87}$$

$$and\ \max\left|\hat{g}_{d-1}(\mathbf{x}(k),\mathbf{v}) - g_{d-1}(\mathbf{x}(k))\right| \ \leq\ \epsilon,\ \ \forall \mathbf{x}(k) \in \Xi \tag{88}$$

The weights $\mathbf{w}$ and $\mathbf{v}$ are unknown. $\mathbf{w}(k)$ and $\mathbf{v}(k)$ represent the estimates of $\mathbf{w}$ and $\mathbf{v}$. Let $\theta = [\mathbf{w}\ \mathbf{v}]$ and define the parameter error as

$$\tilde{\theta}(k) = \theta(k) - \theta \tag{89}$$

We are going to employ a dead-zone algorithm for updating the weights which has been adapted from [29]. At each time step, if the error between the plant output and the model output is larger than a certain threshold, the weights are updated. Otherwise, the weights are not changed. In order to better define the error, rewrite (83) and (84) as

$$y_{k+1} = f_{d-1}(\mathbf{x}(k-d+1)) + g_{d-1}(\mathbf{x}(k-d+1))u_{k-d+1} \tag{90}$$

and

$$\hat{y}_{k+1} = \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}) + \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v})u_{k-d+1} \tag{91}$$

The estimated plant output is

$$y^*_{k+1} = \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k)) + \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))u_{k-d+1} \tag{92}$$

The error e_{k+1} is defined as

$$e^*_{k+1} = y^*_{k+1} - y_{k+1} \tag{93}$$

This error is applied as input to a dead-zone function $D(e)$ which is depicted in Figure 5.2.

$$D(e) = \begin{cases} 0 & if\ |e| \leq d_0 \\ e - d_0 & if\ e > d_0 \\ e + d_0 & if\ e < -d_0 \end{cases} \tag{94}$$

The output of the dead-zone function is used in the updating rule.

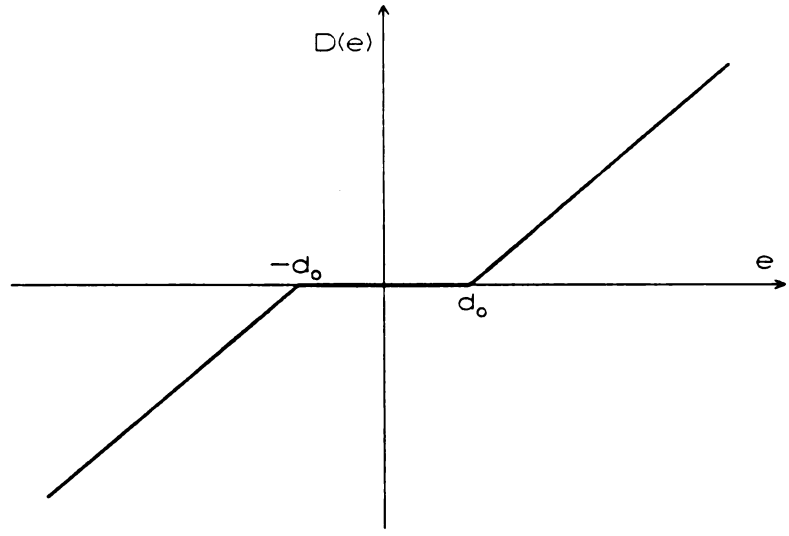


Figure 5.2 The deadzone function

Updating Rule

$$\begin{aligned} \theta(k+1) &= \theta(k) - \frac{1}{r_k} D(y^*_{k+1} - y_{k+1}) \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} \\ &= \theta(k) - \frac{1}{r_k} D(e_{k+1}) J_{k-d+1} \end{aligned} \tag{95}$$

where

$$r_k = 1 + \left\| \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} \right\|^2 = 1 + J'_{k-d+1} J_{k-d+1}$$

This updating rule is similar to that of Method 2 described in chapter **3**, except that e_{k+1} in (38) is replaced by $D(e_{k+1})$.

The control law is specified next.

Control Law

$$u_k = \frac{-\hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}(k)) + r(k)}{\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k))} \tag{96}$$

where $r(k)$ is the reference command satisfying $|r(k)| < d_1$, d_1 being a positive constant.

Now, based on the assumptions made above, a local convergence result is given.

Theorem 2

Given any initial condition $\mathbf{x}(0)$ and any small constant d_0, if the initial parameter error $\tilde{\theta}(0)$ (see (89)) and the modeling error ϵ (see (87)) are small enough (depending on d_0), then

1. *$\left|\tilde{\theta}(k)\right|$ will be monotonically nonincreasing, and $\theta(k)$ will converge to a constant vector.*

2. *The tracking error between the plant output and the reference command will converge to a bounded ball centered at the origin and has radius d_0.*

Proof :

step 1: State transformation.

The dynamics associated with $\mathbf{z}_1$ is

$$\begin{aligned} z_{11}(k+1) &= z_{12}(k) \\ &\vdots \\ z_{1n}(k+1) &= z_{1,n+1}(k) \\ &\vdots \end{aligned}$$

$$z_{1,n+d-1}(k) = F(\mathbf{z}(k)) + G(\mathbf{z}(k))u_k \tag{97}$$

The last equation can be rewritten as

$$\begin{aligned} z_{1,n+d-1}(k+1) &= F(\mathbf{z}(k)) + G(\mathbf{z}(k))u_k \\ &= \hat{F}(\mathbf{z}(k),\mathbf{w}) + \hat{G}(\mathbf{z}(k),\mathbf{v})u_k \\ &\quad + \{F(\mathbf{z}(k)) - \hat{F}(\mathbf{z}(k),\mathbf{w}) + [G(\mathbf{z}(k)) - \hat{G}(\mathbf{z}(k),\mathbf{v})]u_k\}_1 \\ &= \hat{F}(\mathbf{z}(k),\mathbf{w}) + \hat{G}(\mathbf{z}(k),\mathbf{v})u_k + \{\bullet\}_1 \end{aligned} \tag{98}$$

where $\hat{F}(\mathbf{z}(k),\mathbf{w}) = \hat{f}_{d-1}(T^{-1}(\mathbf{z}(k)),\mathbf{w})$ and $\hat{G}(\mathbf{z}(k),\mathbf{v}) = \hat{g}_{d-1}(T^{-1}(\mathbf{z}(k)),\mathbf{v})$. Plugging u_k into (98), we have

$$\begin{aligned} z_{1,n+d-1}(k) &= \hat{F}(\mathbf{z}(k),\mathbf{w}) + \hat{G}(\mathbf{z}(k),\mathbf{v})\frac{-\hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + r(k)}{\hat{G}(\mathbf{z}(k),\mathbf{v}(k))} + \{\bullet\}_1 \\ &= \hat{F}(\mathbf{z}(k),\mathbf{w}) + \hat{G}(\mathbf{z}(k),\mathbf{v})\left[\frac{-\hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + r(k)}{\hat{G}(\mathbf{z}(k),\mathbf{v}(k))}\right. \\ &\quad \left. + \left(\frac{-\hat{F}(\mathbf{z}(k),\mathbf{w}) + r(k)}{\hat{G}(\mathbf{z}(k),\mathbf{v})} - \frac{-\hat{F}(\mathbf{z}(k),\mathbf{w}) + r(k)}{\hat{G}(\mathbf{z}(k),\mathbf{v})}\right)\right] + \{\bullet\}_1 \\ &= r(k) + [\hat{F}(\mathbf{z}(k),\mathbf{w}) - r(k) + \hat{G}(\mathbf{z}(k),\mathbf{v})(\frac{-\hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + r(k)}{\hat{G}(\mathbf{z}(k),\mathbf{v}(k))})] \\ &\quad - \hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + \hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + \{\bullet\}_1 \\ &= r(k) + \{\bullet\}_1 \\ &\quad + \{\hat{F}(\mathbf{z}(k),\mathbf{w}) - \hat{F}(\mathbf{z}(k),\mathbf{w}(k)) + [\hat{G}(\mathbf{z}(k),\mathbf{v}) - \hat{G}(\mathbf{z}(k),\mathbf{v}(k))]u_k\}_2 \\ &= r(k) + \{\bullet\}_1 + \{\bullet\}_2 \end{aligned} \tag{99}$$

Define

$$\begin{aligned} e_1(k) &= z_{11}(k) - r(k-n-d+2) \\ &\vdots \\ e_{1,n+d-1}(k) &= z_{1,n+d-1}(k) - r(k) \end{aligned} \tag{100}$$

In other words,

$$\mathbf{e}_1(k) = \mathbf{z}_1(k) - \Pi(k)$$

Then (97) can be represented in new state variables $\mathbf{e}_1$ as

$$\begin{aligned} e_1(k+1) &= e_2(k) \\ &\vdots \\ e_n(k+1) &= e_{n+1}(k) \\ &\vdots \\ e_{n+d-1}(k) &= \{\bullet\}_1 + \{\bullet\}_2 \end{aligned} \tag{101}$$

With the transformation

$$e_{2i} = z_{2i} - c, \tag{102}$$

the dynamics associated with $\mathbf{z}_2$ is transformed into

$$\begin{aligned} e_{21}(k+1) &= e_{22}(k) \\ &\vdots \\ e_{2,m-1}(k+1) &= e_{2m}(k) \\ e_{2m}(k+1) &= u_{k-d+1} - c \end{aligned} \tag{103}$$

Thus, (101) and (103) together is the new state space representation of the closed-loop system.

Step 2: To show that $\left|\tilde{\theta}(k)\right|$ will decrease and converge, and $\theta(k)$ will converge if the states of the system stay in a compact set.

Consider the sets

$$I_{\mathbf{e}} = \left\{ \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix} \mid |\mathbf{e}_1| \leq \mu_1, |\mathbf{e}_2| \leq \mu_2 \right\} \tag{104}$$

and

$$I_\theta = \{\tilde{\theta} \mid |\tilde{\theta}| \leq \delta\} \tag{105}$$

In the forth coming analysis we will assume that $\mathbf{e}(k)$ stays in $I_{\mathbf{e}}$ for all $k \geq 0$ and investigate how $\tilde{\theta}(k)$ will behave in that situation. Later on, we will show that, under certain conditions, $\mathbf{e}(k)$ indeed stays in $I_{\mathbf{e}}$.

From the facts that

$$\mathbf{z}(k) = \mathbf{e}(k) + [\Pi(k) \quad C]'$$

and

$$\mathbf{x}(k) = T^{-1}(\mathbf{z}(k)),$$

it is clear that if $\mathbf{e}(k)$ stays in $I_{\mathbf{e}}$, then $\mathbf{x}(k)$ is bounded for all k.

Claim 1

If ϵ and δ are small enough, then, for all $(\mathbf{e}(k), \theta(k)) \in I_{\mathbf{e}} \times I_\theta$, $\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k))$ is bounded away from zero.

proof: (please refer to **Assumption 1**, **Assumption 3** and **Control Law** for some of the constant variables used here)

$$\begin{aligned} |\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k)) - g_{d-1}(\mathbf{x}(k))| &\leq |\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k)) - \hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v})| \\ &\quad + |\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}) - g_{d-1}(\mathbf{x}(k))| \\ &\leq c_1|\tilde{\mathbf{v}}(k)| + \epsilon \leq c_1\delta + \epsilon. \end{aligned} \tag{106}$$

If $c_1\delta + \epsilon < B/2$, then

$$|\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k))| > B/2 \quad \clubsuit. \tag{107}$$

Claim 1 ensures that the control u_k is uniformly bounded for all $(\mathbf{e}(k), \theta(k)) \in I_{\mathbf{e}} \times I_\theta$. The input-output form of the system is

$$y_{k+1} = f_{d-1}(\mathbf{x}(k-d+1)) + g_{d-1}(\mathbf{x}(k-d+1))u_{k-d+1}$$

As long as $(\mathbf{e}(k), \theta(k)) \in I_{\mathbf{e}} \times I_\theta$, all previous controls are bounded and, by the assumptions (87) and (88), we have

$$\begin{aligned} y_{k+1} &= f_{d-1}(\mathbf{x}(k-d+1)) + g_{d-1}(\mathbf{x}(k-d+1))u_{k-d+1} \\ &= \hat{f}_{d-1}(\mathbf{x}(k-d+1), \mathbf{w}) + \hat{g}_{d-1}(\mathbf{x}(k-d+1), \mathbf{v})u_{k-d+1} \\ &\quad + [f_{d-1}(\mathbf{x}(k-d+1)) - \hat{f}_{d-1}(\mathbf{x}(k-d+1), \mathbf{w})] \end{aligned}$$

$$+ [(g_{d-1}(\mathbf{x}(k-d+1)) - \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}))u_{k-d+1}]$$

$$= \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}) + \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v})u_{k-d+1} + O(\epsilon) \quad (108)$$

The weights $\mathbf{w}$ and $\mathbf{v}$ in (108) are unknown. The estimate of the plant output is

$$y^*_{k+1} = \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k)) + \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))u_{k-d+1} \quad (109)$$

The error between the neural network output and the plant output is

$$\begin{aligned} e^*_{k+1} &= y^*_{k+1} - y_{k+1} \\ &= \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k)) - \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}) \\ &\quad + [\hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k)) - \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v})]u_{k-d+1} + O(\epsilon) \\ &= \tilde{\theta}(k)' \begin{bmatrix} \left(\frac{\partial \hat{f}_{d-1}(\mathbf{x}(k-d+1),\mathbf{w}(k))}{\partial \mathbf{w}(k)}\right)' \\ \left(\frac{\partial \hat{g}_{d-1}(\mathbf{x}(k-d+1),\mathbf{v}(k))}{\partial \mathbf{v}(k)}\right)' u_{k-d+1} \end{bmatrix} + O(|\tilde{\theta}(k)|^2) + O(\epsilon) \\ &= \tilde{\theta}(k)' J_{k-d+1} + \eta(k) \end{aligned}$$

Since the states $\mathbf{x}(k)$ and all previous controls are bounded, there exist c_7 and c_8 (depending on μ_1 and μ_2) such that

$$|\eta(k)| \leq c_7|\tilde{\theta}(k)|^2 + c_8\epsilon$$

Assume that δ and ϵ are small enough such that

$$|\eta(k)| \leq M < d_0 \quad (110)$$

(d_0 defined in (94)). Next, some analysis related to the deadzone function is provided.

- If $|e^*_{k+1}| \leq d_0$, then $D(e^*_{k+1}) = 0$.
- If $e^*_{k+1} > d_0$, i.e., $\tilde{\theta}(k)'J_{k-d+1} + \eta(k) > d_0$, then $\tilde{\theta}(k)'J_{k-d+1} > 0$, since $|\eta(k)| < d_0$.

$$\begin{aligned} D(e^*_{k+1}) &= \tilde{\theta}(k)'J_{k-d+1} + \eta(k) - d_0 < \tilde{\theta}(k)'J_{k-d+1} + d_0 - d_0 \\ &\Longrightarrow D(e^*_{k+1}) < \tilde{\theta}(k)'J_{k-d+1} \end{aligned}$$

- If $e^*_{k+1} < -d_0$, i.e., $\tilde{\theta}(k)'J_{k-d+1} + \eta(k) < -d_0$, then $\tilde{\theta}(k)'J_{k-d+1} < 0$, since $|\eta(k)| < d_0$.

$$\begin{aligned} D(e^*_{k+1}) &= \tilde{\theta}(k)'J_{k-d+1} + \eta(k) + d_0 > \tilde{\theta}(k)'J_{k-d+1} - d_0 + d_0 \\ &\Longrightarrow \quad D(e^*_{k+1}) > \tilde{\theta}(k)'J_{k-d+1} \end{aligned}$$

Thus in all cases we can represent $D(e^*_{k+1})$ as

$$D(e^*_{k+1}) = \alpha(k)\tilde{\theta}(k)'J_{k-d+1} \tag{111}$$

where $0 \leq \alpha(k) \leq 1$. Plug (111) into the updating rule (95). we get

$$\theta(k+1) = \theta(k) - \alpha(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})J_{k-d+1}}{1 + J'_{k-d+1}J_{k-d+1}} \tag{112}$$

Subtracting θ from both sides of (112), it becomes

$$\tilde{\theta}(k+1) = \tilde{\theta}(k) - \alpha(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})J_{k-d+1}}{1 + J'_{k-d+1}J_{k-d+1}} \tag{113}$$

Then,

$$\begin{aligned} &\tilde{\theta}(k+1)'\tilde{\theta}(k+1) - \tilde{\theta}(k)'\tilde{\theta}(k) \\ &= -2\alpha(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} + \alpha^2(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2 J'_{k-d+1}J_{k-d+1}}{(1 + J'_{k-d+1}J_{k-d+1})^2} \\ &\leq -2\alpha(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} + \alpha^2(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} \qquad (114) \\ &\leq -\alpha^2(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} - (2\alpha(k) - 2\alpha^2(k))\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} \\ &\leq -\alpha^2(k)\frac{(\tilde{\theta}(k)'J_{k-d+1})^2}{1 + J'_{k-d+1}J_{k-d+1}} \qquad (115) \end{aligned}$$

$$\Longrightarrow \quad \tilde{\theta}(k)'\tilde{\theta}(k) \textit{ is monotonically decreasing and } \tilde{\theta}(k)'\tilde{\theta}(k) \longrightarrow C_1 \textit{ as } k \longrightarrow \infty \tag{116}$$

where C_1 is a constant. This shows that the norm of parameter errors will converge. Then, (116) implies that

$$\alpha(k)\frac{\tilde{\theta}(k)'J_{k-d+1}}{\sqrt{1 + J'_{k-d+1}J_{k-d+1}}} \longrightarrow 0 \textit{ as } k \longrightarrow \infty \tag{117}$$

$$\Longrightarrow \quad \tilde{\theta}(k+1) - \tilde{\theta}(k) \longrightarrow 0 \ \ as \ k \longrightarrow \infty \tag{118}$$

$$\Longrightarrow \quad \tilde{\theta}(k) \longrightarrow C_2 \ \ as \ k \longrightarrow \infty \tag{119}$$

$$\Longrightarrow \quad \theta(k) \longrightarrow C_2 + \theta \ \ as \ k \longrightarrow \infty \tag{120}$$

where C_2 is a constant vector. The result (118) can be verified by rewriting (113) as

$$\tilde{\theta}(k+1) = \tilde{\theta}(k) - \alpha(k) \frac{(\tilde{\theta}(k)' J_{k-d+1})}{\sqrt{1 + J'_{k-d+1} J_{k-d+1}}} \frac{J_{k-d+1}}{\sqrt{1 + J'_{k-d+1} J_{k-d+1}}} \tag{121}$$

It is shown in (120) that the weights in the neural network model will converge.

Step 3: To show that $|\mathbf{e}_1(k)|$ is uniformly bounded by μ_1 if ϵ and δ are small enough.

Rewrite the dynamics associated with $\mathbf{e}_1$ as

$$\mathbf{e}(k+1) = A\mathbf{e}(k) + \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} [\bullet]_1, \quad where \ A = \begin{bmatrix} 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix},$$

and

$$\begin{aligned} [\bullet]_1 &= \{\bullet\}_1 + \{\bullet\}_2 \\ &\leq c_3\delta + c_4\epsilon \end{aligned} \tag{122}$$

The constants c_3 and c_4 in (122) depend on μ_1 and μ_2.

A is a stable matrix (since all eigenvalues are at the origin). $\Longrightarrow$ Given any symmetric $Q > 0$, $\exists$ a symmetric $P > 0$ such that

$$A'PA - P = -Q.$$

Choose the Lyapunov function

$$V_1(\mathbf{e}(k)) = \mathbf{e}'(k)P\mathbf{e}(k),$$

Then,

$$V_1(\mathbf{e}(k+1)) - V_1(\mathbf{e}(k)) = -\mathbf{e}'(k)Q\mathbf{e}(k) + 2\mathbf{e}(k)'A'P \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} [\bullet]_1$$

$$
\begin{aligned}
& + [\bullet]_1^2 [0 \cdots 1] P \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} \\
\leq \quad & -\mathbf{e}'(k) Q \mathbf{e}(k) + (c_3 |\tilde{\theta}(k)| + c_4 \epsilon) |\mathbf{e}(k)| + (c_3 |\tilde{\theta}(k)| + c_4 \epsilon)^2 \\
\leq \quad & -\lambda_{min}(Q) |\mathbf{e}(k)|^2 + (c_3 \delta + c_4 \epsilon) |\mathbf{e}(k)| + (c_3 \delta + c_4 \epsilon)^2 \\
\leq \quad & -\lambda_{min}(Q) |\mathbf{e}(k)|^2 + c_{11} (c_3 \delta + c_4 \epsilon)^2 \\
\leq \quad & -\lambda V_1(\mathbf{e}(k)) + c_{11} (c_3 \delta + c_4 \epsilon)^2, \ \textit{where } \lambda = \frac{\lambda_{min}(Q)}{\lambda_{max}(P)} \\
\Longrightarrow \quad & \textit{The R.H.S. will be negative} \\
& \textit{whenever } V_1(\mathbf{e}(k)) > \frac{c_{11}}{\lambda} (c_3 \delta + c_4 \epsilon)^2 \qquad (123)
\end{aligned}
$$

The conclusion from (123) is that, given any $\mu_3 > 0$, if δ and ϵ are small enough such that

$$\frac{c_{11}}{\lambda} (c_3 \delta + c_4 \epsilon)^2 < \mu_3 \qquad (124)$$

then

$$\{V_1(\mathbf{e}) \leq \mu_3\} \textit{ is an invariant set, i.e.,}$$

$$V_1(\mathbf{e}(0)) \leq \mu_3 \Longrightarrow V_1(\mathbf{e}(k)) \leq \mu_3, \ \forall \, k \geq 0.$$

It follows that

$$|\mathbf{e}(0)| \leq \sqrt{\frac{\mu_3}{\lambda_{max}(P)}} \Longrightarrow |\mathbf{e}(k)| \leq \sqrt{\frac{\mu_3}{\lambda_{min}(P)}}, \ \forall \, k \geq 0.$$

Hence, μ_1 in (104) can be chosen to be $\sqrt{\frac{\mu_3}{\lambda_{min}(P)}}$, where μ_3 is chosen large enough so that $|\mathbf{e}(0)| \leq \sqrt{\frac{\mu_3}{\lambda_{max}(P)}}$. Notice also that $|\mathbf{e}(k)|$ will decrease toward a ball of radius $O(c_3 \delta + c_4 \epsilon)$.

step 4: To show that the dynamics associated with $\mathbf{e}_2$ are bounded.

The dynamics associated with $\mathbf{e}_2$ is

$$
\begin{aligned}
e_{21}(k+1) &= e_{22}(k) \\
&\vdots
\end{aligned}
$$

$$e_{2,m-1}(k+1) = e_{2m}(k)$$

$$e_{2m}(k+1) = u_{k-d+1} - c$$

where

$$\begin{aligned}
& u_{k-d+1} \\
&= \frac{-\hat{F}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{w}(k-d+1))+r(k-d+1)}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))} \\
&= \frac{-F(0,\mathbf{e}_2(k-d+1)+C)}{G(0,\mathbf{e}_2(k-d+1)+C)} \\
&\quad + \left(\frac{-\hat{F}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{w}(k-d+1))}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))}\right. \\
&\quad \left. - \frac{-F(0,\mathbf{e}_2(k-d+1)+C)}{G(0,\mathbf{e}_2(k-d+1)+C)}\right) \\
&\quad + \frac{r(k-d+1)}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))} \\
&= \frac{-f_0(T^{-1}(0,\mathbf{e}_2(k)+C))}{g_0(T^{-1}(0,\mathbf{e}_2(k)+C))} \\
&\quad + \left(\left[\frac{-\hat{F}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{w}(k-d+1))}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))}\right.\right. \\
&\quad \left.\left. - \frac{-\hat{F}(0,\mathbf{e}_2(k-d+1)+C,\mathbf{w}(k-d+1))}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))}\right]\right. \\
&\quad + \left[\frac{-\hat{F}(0,\mathbf{e}_2(k-d+1)+C,\mathbf{w}(k-d+1))}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v}(k-d+1))}\right. \\
&\quad \left. - \frac{-\hat{F}(0,\mathbf{e}_2(k-d+1)+C,\mathbf{w})}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v})}\right] \\
&\quad + \left[\frac{-\hat{F}(0,\mathbf{e}_2(k-d+1)+C,\mathbf{w})}{\hat{G}(\mathbf{e}_1(k-d+1)+\Pi(k-d+1),\mathbf{e}_2(k-d+1)+C,\mathbf{v})}\right. \\
&\quad \left.\left. - \frac{-F(0,\mathbf{e}_2(k-d+1)+C)}{G(0,\mathbf{e}_2(k-d+1)+C)}\right]\right) \\
&\quad + O(\Pi(k-d+1)) \\
&= \frac{-f_0(T^{-1}(0,\mathbf{e}_2(k)+C))}{g_0(T^{-1}(0,\mathbf{e}_2(k)+C))} \\
&\quad + [O(\tilde{\theta}(k-d+1)) + O(\mathbf{e}_1(k-d+1)) + O(\Pi(k-d+1)) + O(\epsilon)]_2
\end{aligned}$$

Since all the states are bounded, $[\bullet]_2$ is bounded by a constant.

Let

$$S_k = \left[e_{22}(k), \ldots, e_{2m}, \frac{-f_0(T^{-1}(0, \mathbf{e}_2(k) + C))}{g_0(T^{-1}(0, \mathbf{e}_2(k) + C))} - c \right]'$$

and

$$Q_k = [0, \ldots, 0, [\bullet]_2]'$$

Using the $V_2(\mathbf{e}_2(k))$ described in (80) as a Lyapunov function candidate, we obtain

$$\begin{aligned} V_2(\mathbf{e}_2(k+1)) - V_2(\mathbf{e}_2(k)) &= V_2(S_k + Q_k) - V_2(\mathbf{e}_2(k)) \\ &= [V_2(S_k) - V_2(\mathbf{e}_2(k))] + V_2(S_k + Q_k) - V_2(S_k) \\ &\leq -k_5 |\mathbf{e}_2(k)|^2 + \hat{k} |\zeta S_k + (1-\zeta) Q_k| \cdot |Q_k| \\ &\leq -k_5 |\mathbf{e}_2(k)|^2 + \hat{k}_1 |S_k| \cdot |Q_k| + \hat{k}_2 |Q_k|^2 \\ &\leq -k_5 |\mathbf{e}_2(k)|^2 + k_6 |\mathbf{e}_2(k)| + k_7. \end{aligned} \tag{125}$$

Thus, if μ_2 is chosen large enough, there is an invariant set $\{\mathbf{e}_2(k) \mid V_2(\mathbf{e}_2(k)) \leq c_{12}\}$ inside $\{\mathbf{e}_2 \mid |\mathbf{e}_2| \leq \mu_2\}$.

We summarize the results from Step 2, Step 3 and Step 4 here: Given any initial condition, if μ_1 and μ_2 are chosen large enough, and δ and ϵ are small enough, then

1. $(\mathbf{e}(k), \theta(k)) \in I_{\mathbf{e}} \times I_\theta, \ \forall\, k \geq 0.$
2. $\left|\tilde{\theta}(k)\right|$ will be monotonically nonincreasing, and $\theta(k)$ will converge to a constant vector.

Step 5: To show that the plant output will eventually track the reference command with an error less than d_0.

Since $\mathbf{x}(k)$ and u_k are bounded for all k, it can be verified that J_{k-d+1} is bounded. Hence, (117) implies that

$$\alpha(k)\tilde{\theta}(k) J_{k-d+1} \longrightarrow 0 \ \ as\ k \longrightarrow \infty \tag{126}$$

$$\Longrightarrow \qquad D(e_{k+1}) \longrightarrow 0 \ \ as\ k \longrightarrow \infty \tag{127}$$

$$\Longrightarrow \qquad |e_{k+1}| < d_0 \ \ as \ k \longrightarrow \infty \tag{128}$$

$$\Longrightarrow \qquad |y^*_{k+1} - y_{k+1}| < d_0 \ \ as \ k \longrightarrow \infty \tag{129}$$

Recall that

$$y^*_{k+d} = \hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}(k+d-1)) + \hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k+d-1))u_k \tag{130}$$

While the control u_k is generated from

$$\bar{y}_{k+d} = \hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}(k)) + \hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k))u_k \tag{131}$$

Then

$$|y^*_{k+d} - \bar{y}_{k+d}| \leq |\hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}(k+d-1)) - \hat{f}_{d-1}(\mathbf{x}(k), \mathbf{w}(k))| \tag{132}$$

$$+ |[\hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k+d-1)) - \hat{g}_{d-1}(\mathbf{x}(k), \mathbf{v}(k))]u_k| \tag{133}$$

$$\leq K|\theta(k-d+1) - \theta(k)| \longrightarrow 0, \ as \ k \longrightarrow \infty. \tag{134}$$

$$\Longrightarrow \qquad |\bar{y}_{k+d} - y_{k+d}| < d_0 \ \ as \ k \longrightarrow \infty \tag{135}$$

□

6 Simulation

This section is divided into several subsections, with each subsection dealing with a specific issue. Some issues are directly related to the results presented in the previous chapters, while others are not, as will be described at the beginning of each subsection. All the simulation programs are written in Microsoft C and run on a IBM PC compatible machine. The plants under consideration are of the general form

$$y_{k+1} = f_0(y_k, y_{k-1}, \cdots, y_{k-n+1}, u_{k-d}, u_{k-d-1}, \cdots, u_{k-d-m+1}) + g_0 u_{k-d+1} \qquad (136)$$

which is a relative degree d system, with g_0 being a constant. We will simulate on relative-degree-one and relative-degree-two systems only.

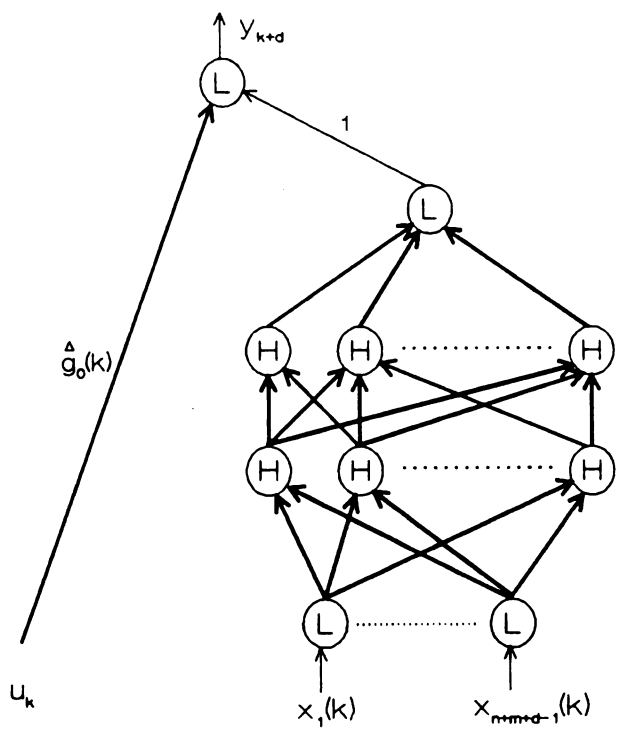


Figure 6.1 The neural network model

The neural network architecture which will be used for the rest of this chapter is

shown in Figure 6.1 and denoted by NN(m, n), where m and n are the number of the hidden neurons used in the first and the second hidden layers (the first hidden layer is the one close to the inputs).

We look at identification problem first in section **6.1**, and study various control issues in subsequent sections.

6.1 Identification

We start with identification problems, although identification is not directly related to the results presented in the previous chapters. There are several reasons to pursue identification first:

1. To demonstrate that layered neural networks can learn to approximate nonlinear functions.

2. To get some insight about how to choose a suitable neural network size to handle our current control problems.

3. Before the neural networks are applied to control problems, some "pretraining" may be needed.

It is worth while trying to clearly describe the training process here. Neural networks of the form in Figure 6.1 will be trained to approximate the system

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.7sin[0.5(y_k + y_{k-1})] \cdot cos[0.5(y_k + y_{k-1}))] + 1.2u_k \quad (137)$$

Several network sizes will be used to develop some feeling about how to choose a suitable neural network size. Each nonlinear neuron in the neural network has a bias weight attached to it, though the bias weight is not shown in Figure 6.1. In order to focus on the effect of the number of nonlinear hidden neurons on the identification process, the weight between u_k and the output neuron in Figure 6.1 (i.e., $\hat{g}_0$) is fixed to be 1.2 (the same as the plant).

At each time step, the control u_k is randomly selected from the interval $[-1.7, 1.7]$. The inputs u_k, y_k and y_{k-1} are applied to the plant and the neural network model. The error between the plant and the model is used to train the neural network. We use standard back-propagation algorithm [1] to update the weights. The algorithm is described here.

The plant is

$$y_{k+1} = f_0(y_k, y_{k-1}) + 1.2u_k$$

The model is

$$y^*_{k+1} = \hat{f}_0(y_k, y_{k-1}, W_k) + 1.2u_k$$

Define the error to be $J_k = (y^*_{k+1} - y_{k+1})^2$. The weights are updated as follows:

$$\begin{aligned} W_{k+1} &= W_k - \mu_k \nabla_{W_k} J_k \\ &= W_k - \mu_k (y^*_{k+1} - y_{k+1}) \left[\frac{\partial \hat{f}_0(y_k, y_{k-1}, W_k)}{\partial W_k} \right]' \end{aligned}$$

where $\frac{\partial \hat{f}_0(y_k, y_{k-1}, W_k)}{\partial W_k}$ is calculated by the back-propagation algorithm. The learning rate μ_k used in this simulation is

$$\mu_k = \begin{cases} 0.2, & k \leq 1000 \\ 0.45, & 1000 < k \leq 5000 \\ 0.25, & 5000 < k \leq 10000 \\ 0.15, & 10000 < k \leq 20000 \\ 0.1, & 20000 < k \leq 30000 \\ 0.5, & 30000 < k \end{cases}$$

Our experiences suggest that using smaller learning rate at the early stage of training can avoid large oscillation which may cause instability. The learning rate is increased and then gradually decreased for better convergence. The initial weights of the neural networks are selected randomly between -0.1 and 0.1. In the rest of this chapter, the initial weights of the neural networks are always selected this way. Small initial weights can reduce the chance of saturating the nonlinear neurons.

In all of the figures shown in this subsection, each data point represents the error between $f_0(y_k, y_{k-1})$ and $\hat{f}_0(y_k, y_{k-1}, W_k)$ averaged over 1000 time steps.

Figure 6.2 shows the training results for the first 30 thousand time steps. The network NN(2,2) does not contain enough hidden neurons, so it is incapable of bringing the error down. It is also apparent from Figure 6.2 that NN(4,4) learns to approximate $f(y_k, y_{k-1})$ faster than NN(6,6), and NN(6,6) faster than NN(12,12). For these three neural networks, errors are reduced to about 0.033 at the end of 30 thousand training. It seems that NN(4,4), NN(6,6) and NN(12,12) result in similar errors, but next figure will provide more insight.

Figure 6.3 shows the training of NN(4,4), NN(6,6) and NN(12,12) from 30 to 200 thousand time steps. It is obvious that NN(4,4) is leveling off, while NN(12,12) is reducing the error at a much faster pace. NN(12,12) pushes the error to 0.012 at the end of 200 thousand training.

Figure 6.4 shows the result of the same identification problem using NN(25,25). Compared with NN(12,12), NN(25,25) takes much longer time to reduce the error significantly.

We conclude that neural networks of sizes between NN(6,6) and NN(12,12) are adequate to be used in the adaptive control of the system (137), in terms of the speed they can learn to approximate the plant and the accuracy they can achieve.

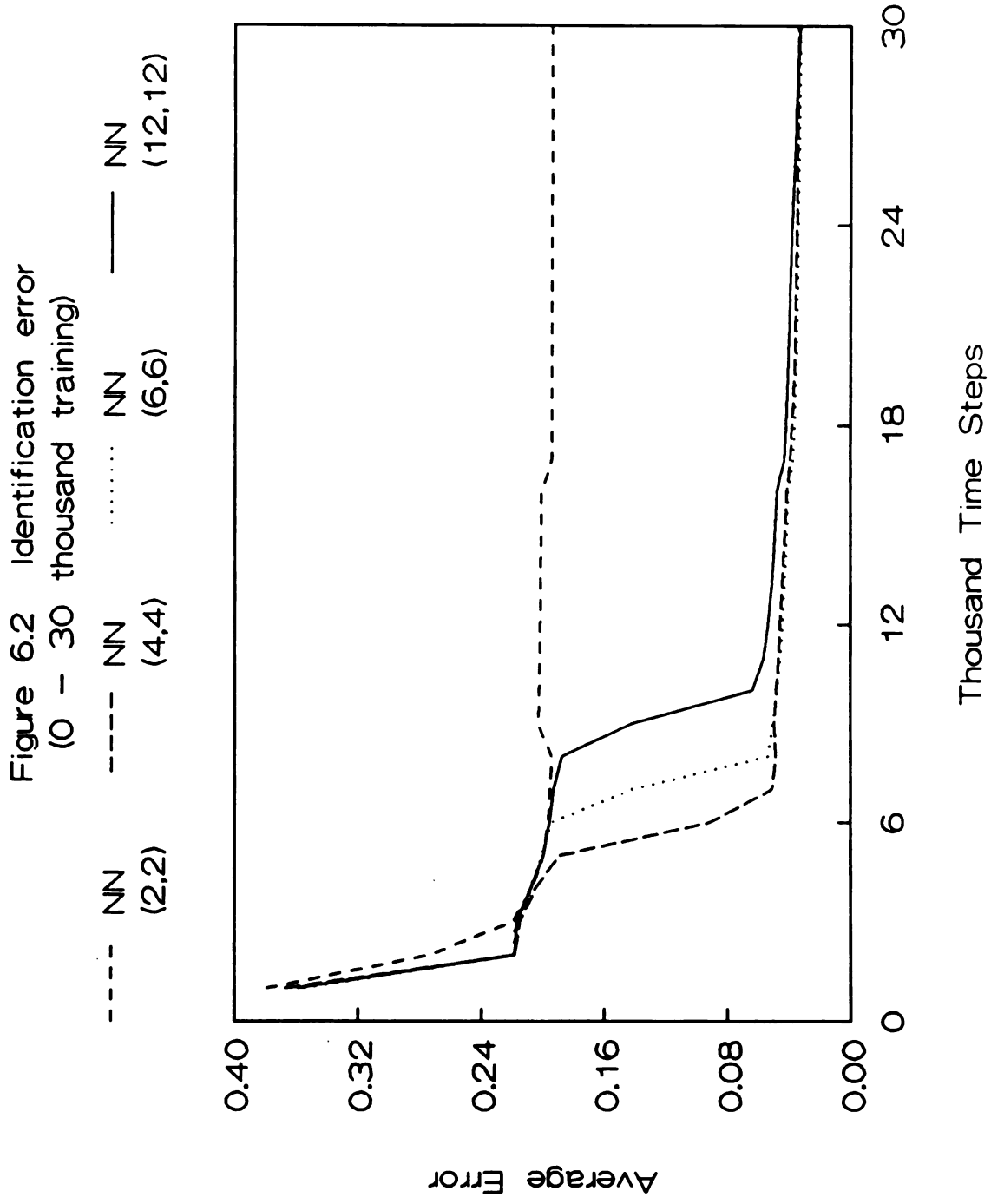


Figure 6.2 Identification error (0 – 30 thousand training)

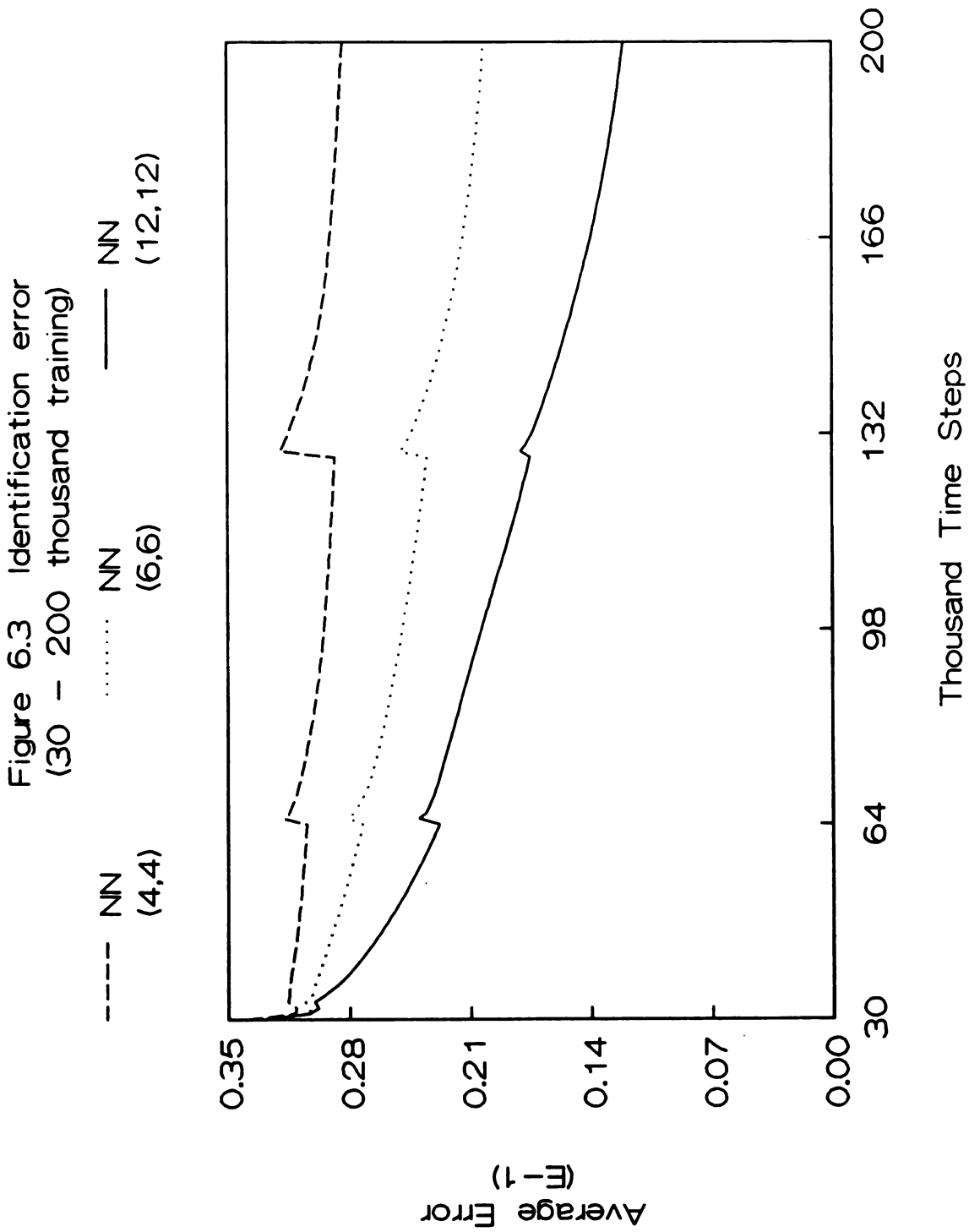


Figure 6.3 Identification error
(30 - 200 thousand training)
---- NN (4,4)
..... NN (6,6)
—— NN (12,12)
Average Error (E-1)
0.35
0.28
0.21
0.14
0.07
0.00
30
64
98
132
166
200
Thousand Time Steps

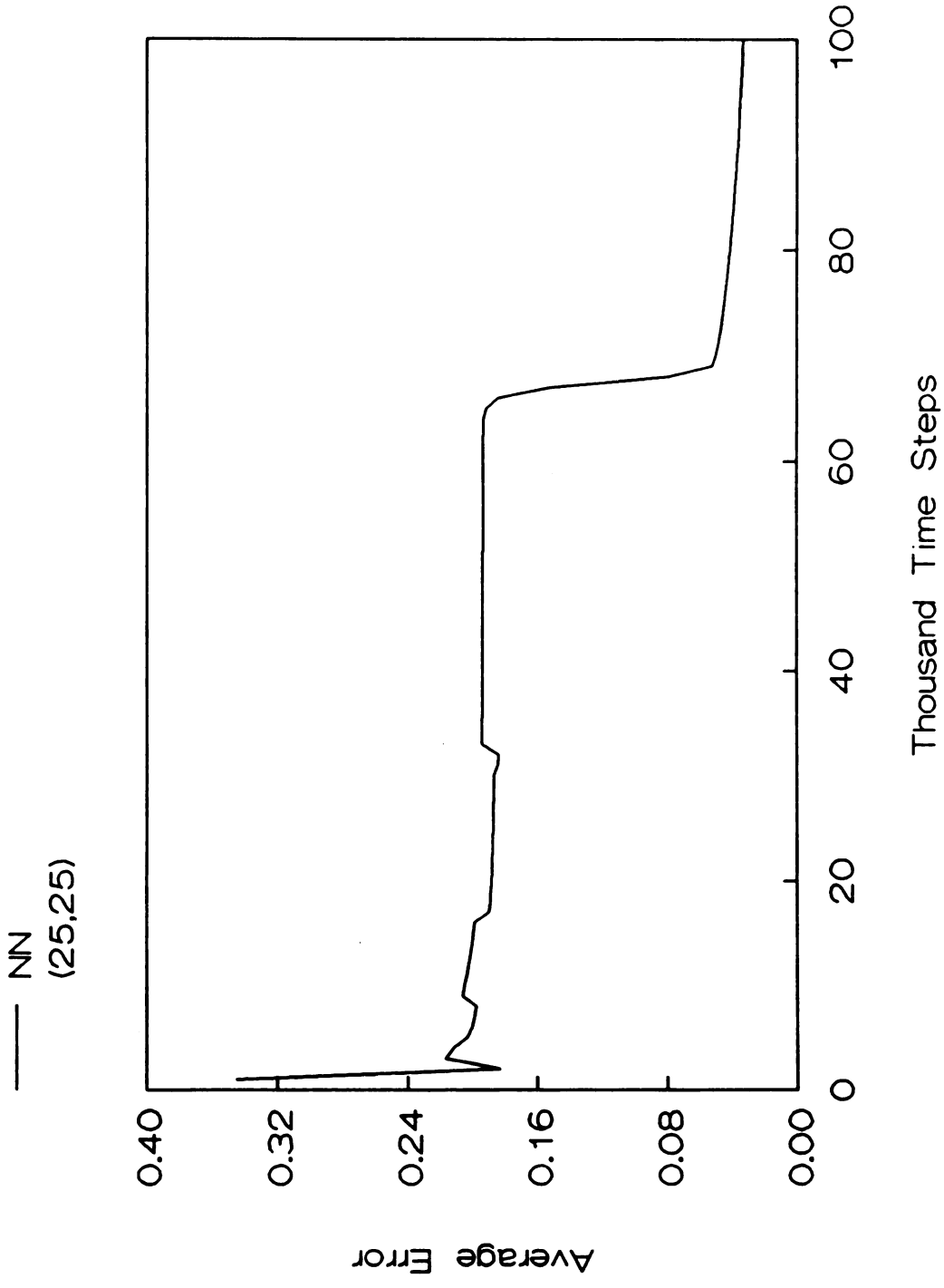


Figure 6.4 Identification error
(learning is slow using large NN)

6.2 Regulation using Neural Networks without Bias Weights

Part 1:

The simulation here corresponds to the result in Chapter **4**. The neural network NN(7, 7) is used in the control system to model and regulate the output of the plant (138) to zero.

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.7sin[0.5(y_k + y_{k-1})] \cdot cos[0.5(y_k + y_{k-1})] + 1.2u_k \quad (138)$$

The plant satisfies the assumption that $f_0(X(k)) = 0$ when $X(k) = 0$.

The neural network contains no bias weights as it has been assumed in chapter **4** so that $\hat{f}_0(X(k), W_k) = 0$ when X(k)=0. Before the neural network is used for control purpose, it may go through similar identification process as in **6.1** using standard back propagation algorithm. But this time the weight $\hat{g}_0$ is not fixed. When the neural network is used in feedback regulation, the updating rule is switched to (45) described in chapter **4**, where μ is set to be one. Simulation results are provided next to show how different amount of training on the neural network NN(7, 7) would affect the performance of closed-loop regulation. From the experience of **6.1**, it is expected that as a neural network receives more training cycles, it should have better approximation of the plant and should perform better in feedback regulation.

In this simulation, the weights of the neural networks after the training (or identification) process become the initial weights of the neural networks in the control system. The initial condition of the plant is

$$(y_0, y_{-1}) = (-1.5, -1.5)$$

Figure 6.5 compares the plant outputs resulted from three neural network controllers; one received no training, while the other two received 7000 and 14000 training cycles, respectively, before being used to regulate the plant output. It confirms the expectation that better trained neural networks should perform better in the control system.

However, an important message from Figure 6.5 is that the plant output is regulated to zero even when the neural network is not pretrained.

Part 2:

Figure 6.6 shows the result when the same neural network is used to regulate the plant

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.3cos[y_k + y_{k-1}] + 1.2u_k \tag{139}$$

Large oscillations appear again and again when the plant output is close to zero. The problem is that the neural network $\hat{f}_0(y_k, y_{k-1}, W_k)$ vanishes at the origin (since it contains no bias weights), but $f_0(y_k, y_{k-1})$ is nonzero when $y_k = y_{k-1} = 0$. Therefore, it is impossible for the neural network to model the plant well and generate satisfactory controls.

To solve the problem, we want to use the idea of dead-zone described in chapter 5. The cosine term in (139) can be treated as the error between the plant and the best approximation which the neural network can achieve. The maximum magnitude of $0.3cos[y_k + y_{k-1}]$ is 0.3, so it is intuitive to specify $[-0.3, 0.3]$ as the dead zone region (i.e., setting $d_0 = 0.3$) to cover the dynamics which can not be modeled by the neural network. The simulation with $d_0 = 0.3$ is shown in Figure 6.7, where the plant output converges to the dead zone (more precisely, converges to $[-0.1, 0.3]$). Figure 6.8 shows the result when d_0 is set to be 0.15. Figure 6.8 ($d_0 = 0.15$) exhibits worse transient behavior than that of Figure 6.7 ($d_0 = 0.3$), and the region the plant output converges to in Figure 6.8 is of the same size as that of Figure 6.7. These are evidences that $d_0 = 0.15$ is not a good choice. How about if $d_0 > 0.3$? Figure 6.9 gives the plant output when $d_0 = 0.4$. After a quick initial shift, the plant output converges to 0.4. This suggests that a good choice of d_0 may be somewhere between 0.3 and 0.4.

Although the plant output converges to a point within the specified dead zone

when $d_0 = 0.4$, it just happens to be the case. We did not provide theoretical results that guarantee the convergence of plant output to a fixed point within the dead zone.

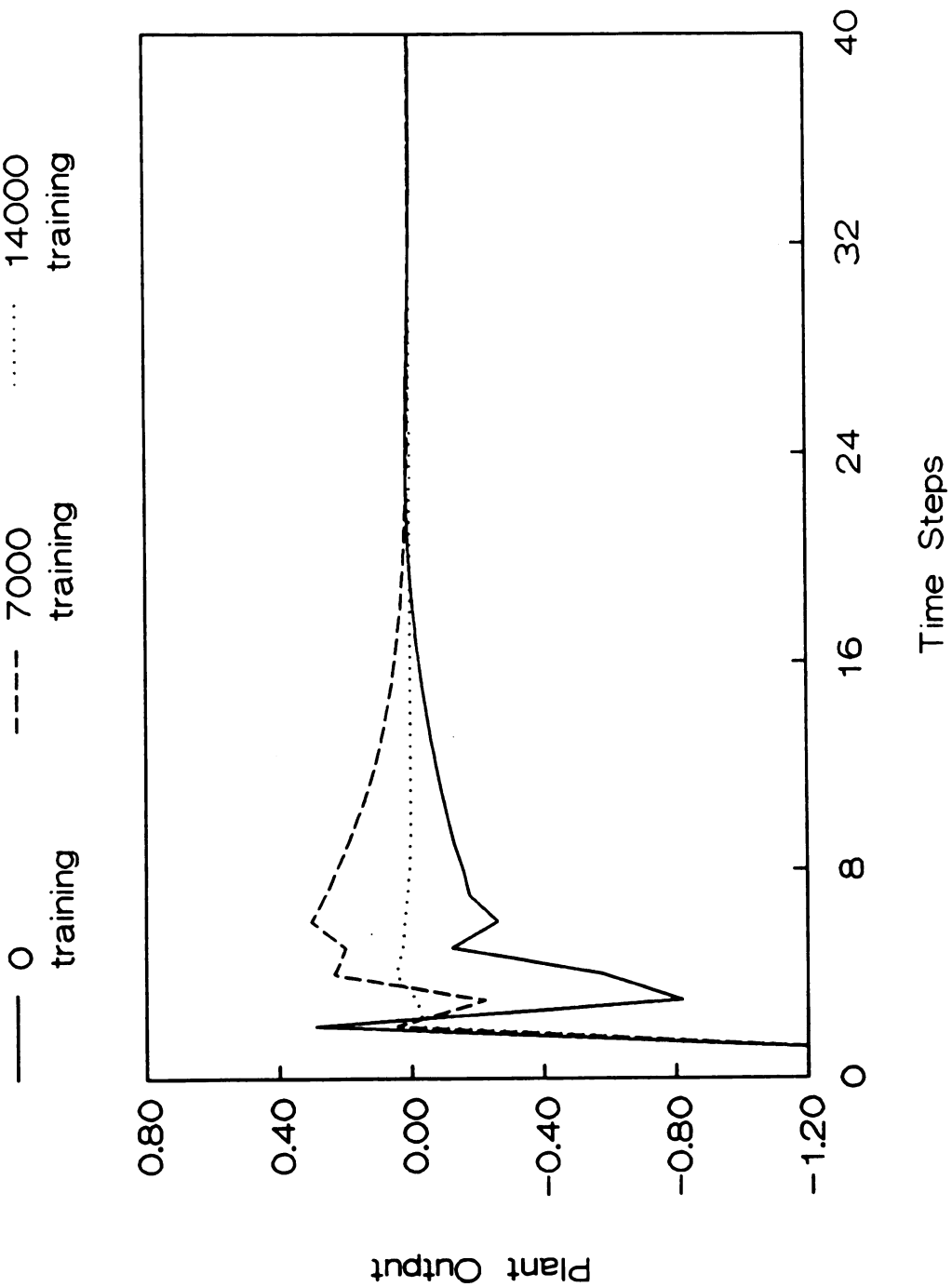


Figure 6.5 Regulation Error vs Training

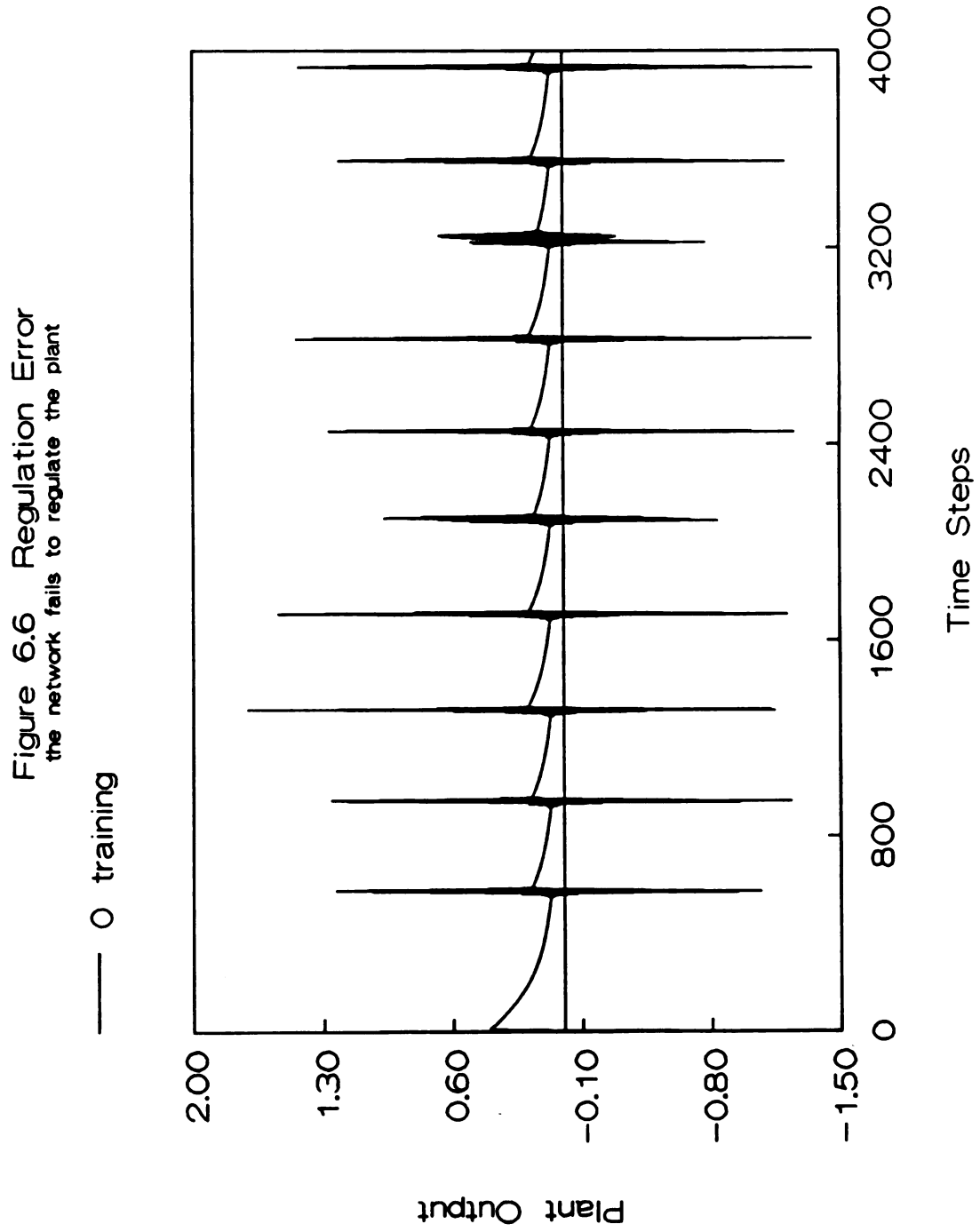


Figure 6.6 Regulation Error
the network fails to regulate the plant

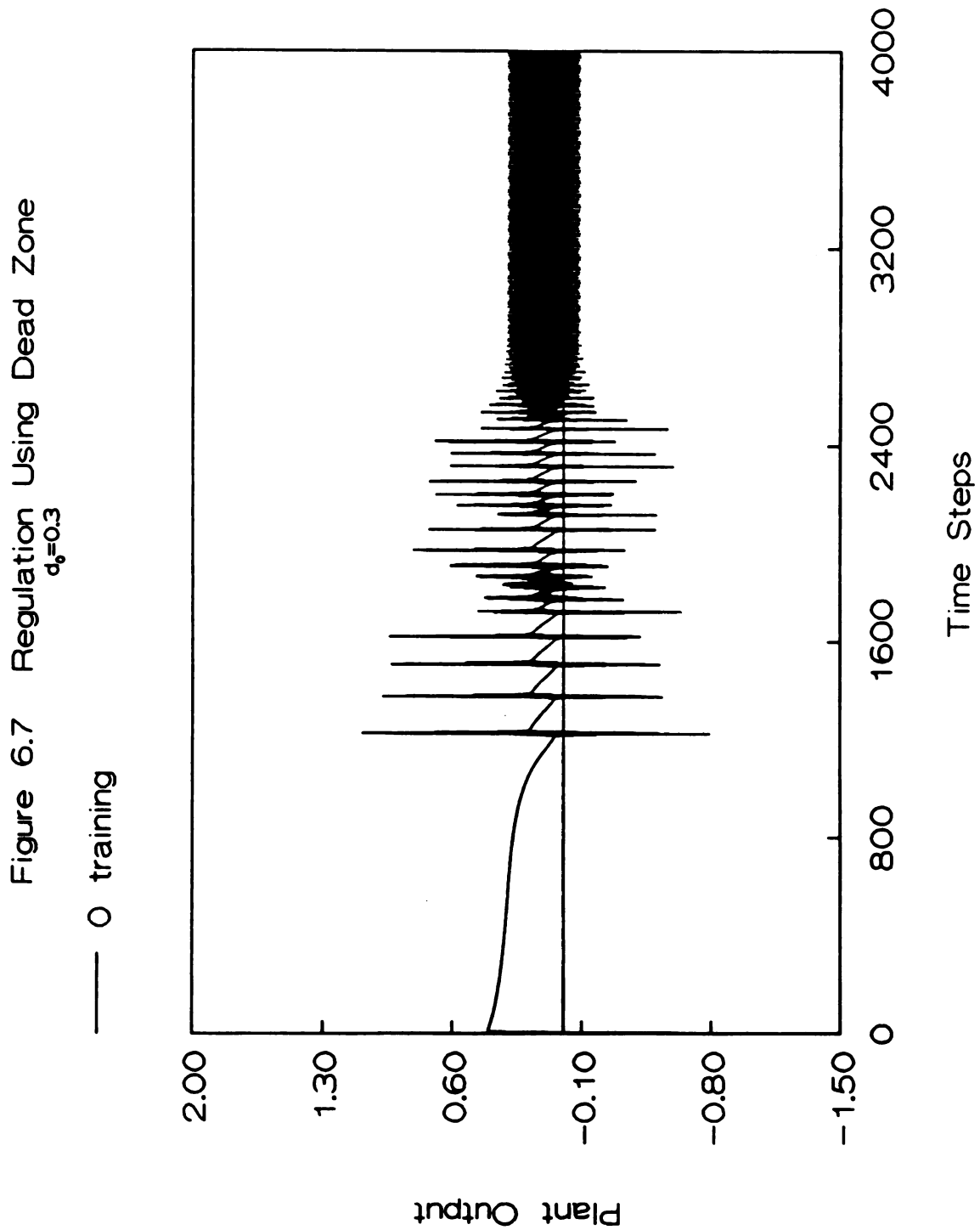

Regulation Using Dead Zone
d_0=0.3
— 0 training
2.00
1.30
0.60
−0.10
−0.80
−1.50
Plant Output
0
800
1600
2400
3200
4000
Time Steps

Figure 6.7

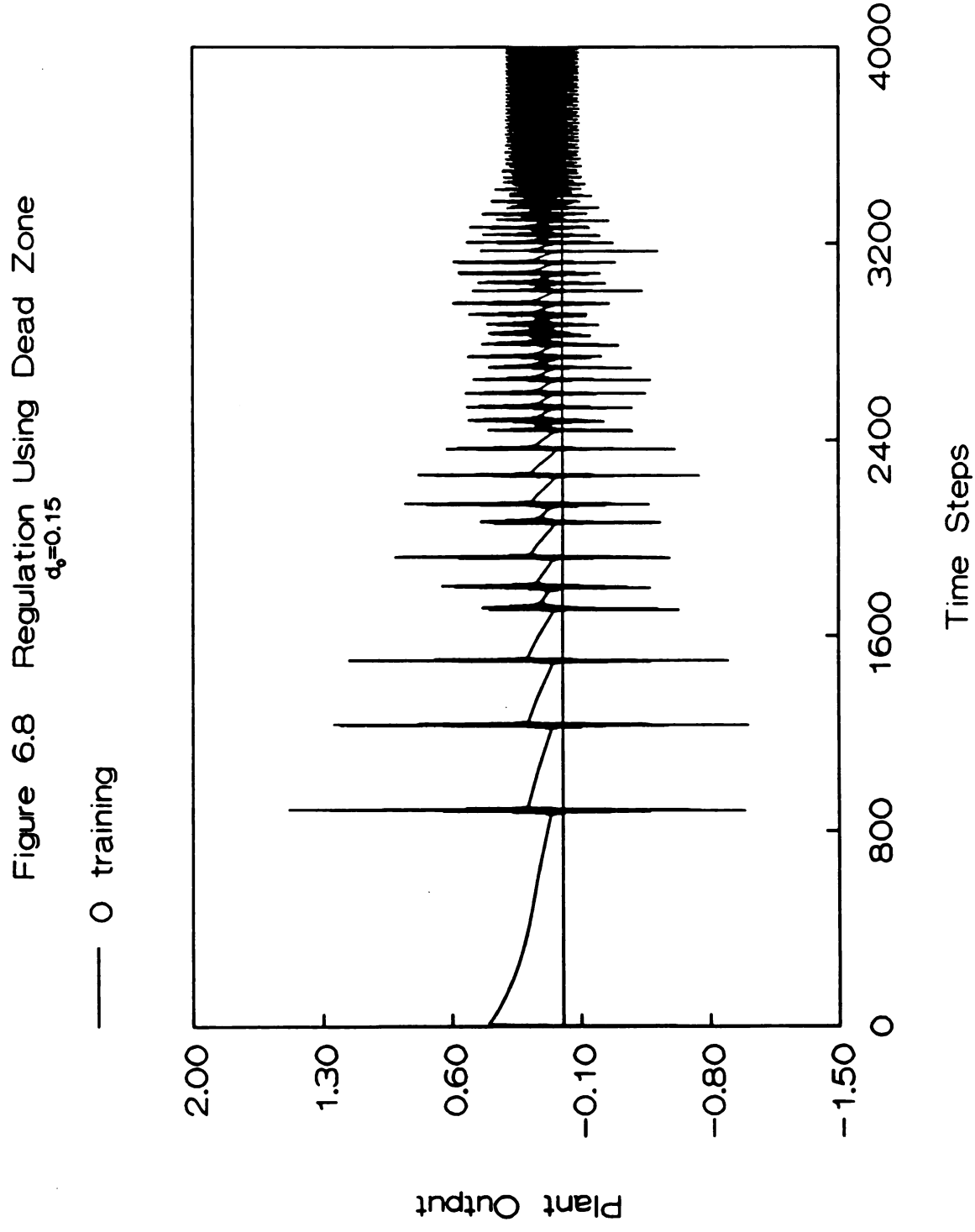

Regulation Using Dead Zone
d_0=0.15
0 training
Plant Output
2.00
1.30
0.60
-0.10
-0.80
-1.50
0
800
1600
2400
3200
4000
Time Steps

Figure 6.8

Figure 6.9 Regulation Using Dead Zone

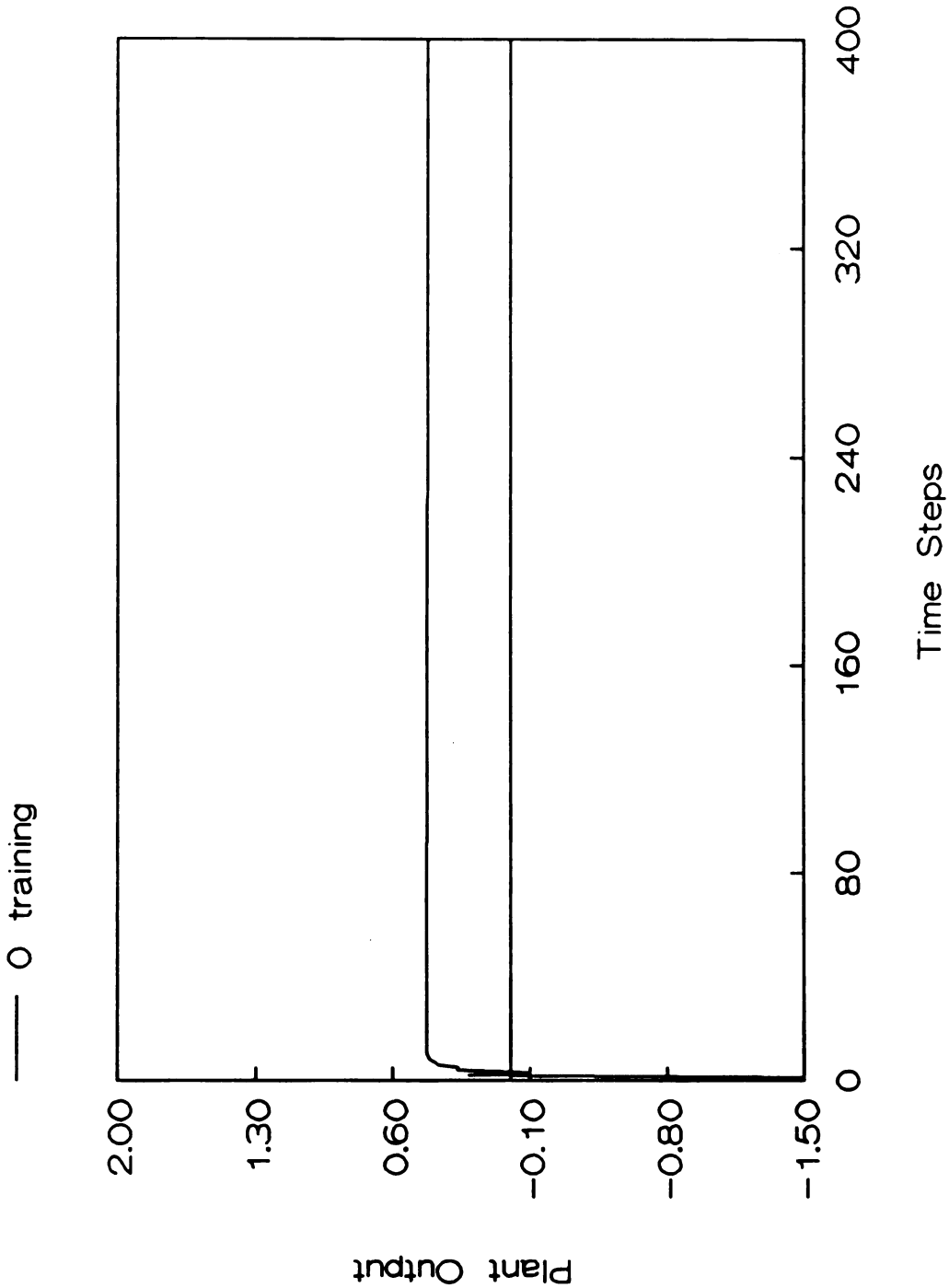


Figure 6.9 Regulation Using Dead Zone d_0=0.40

6.3 Regulation using Neural Networks with Bias Weights

Part 1:

In section **3.2** part 2, a layered neural network without bias weights is unable to model and regulate the system

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.3cos[y_k + y_{k-1}] + 1.2u_k \tag{140}$$

and the idea of dead zone is tried to confine the plant output to a bounded area. Here we apply the neural network NN(7, 7) with bias weights to model and regulate the system (140). Figure 6.10 shows the plant output when NN(7, 7), which receives no pretraining, is used in the control system. No dead zone is specified in the learning rule (92) ($d_0 = 0$), because the plant output can be regulated to zero perfectly.

Part 2:

In this thesis it is assumed that the nonlinearities of the plant are unknown but can be modeled by neural networks. There may be cases when the nonlinear functions of the plant are known, but the coefficients attached to these nonlinear functions are unknown. The second situation has been considered recently in the continuous-time setup by Sastry and Isidori [18]. We are going to refer to the second situation as the analytic approach. The purpose of this part of simulation is to compare the robustness properties of the neural network method and the analytic method.

Suppose the plant is

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 1.2u_k \tag{141}$$

In the neural network method, (141) is modeled by NN(7, 7). In the analytic method, (141) is modeled by

$$y_{k+1} = a(k)\frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + b(k)u_k \tag{142}$$

where $a(k)$ and $b(k)$ are unknown coefficients. Both the neural net model and the analytic model are trained to approximate (142) well before they are applied to regulation problems. However, the actual plant to be controlled is

$$y_{k+1} = \frac{1.5 y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + w \cdot cos[y_k + y_{k-1}] + 1.2 u_k \tag{143}$$

Figure 6.11 through Figure 6.14 compare the plant outputs resulted from the neural network controller and the analytic controller when w = 0.0, 0.1, 0.4, and 0.7, respectively. The neural network control can always bring the plant output to zero in all of these cases, while errors, which are roughly proportional to the magnitude of w, are observed when the analytic controller is used. The neural networks are universal approximators in the sense that they can accommodate variations in the part of the plant.

Figure 6.10 Regulation Error
Using NN with bias weights

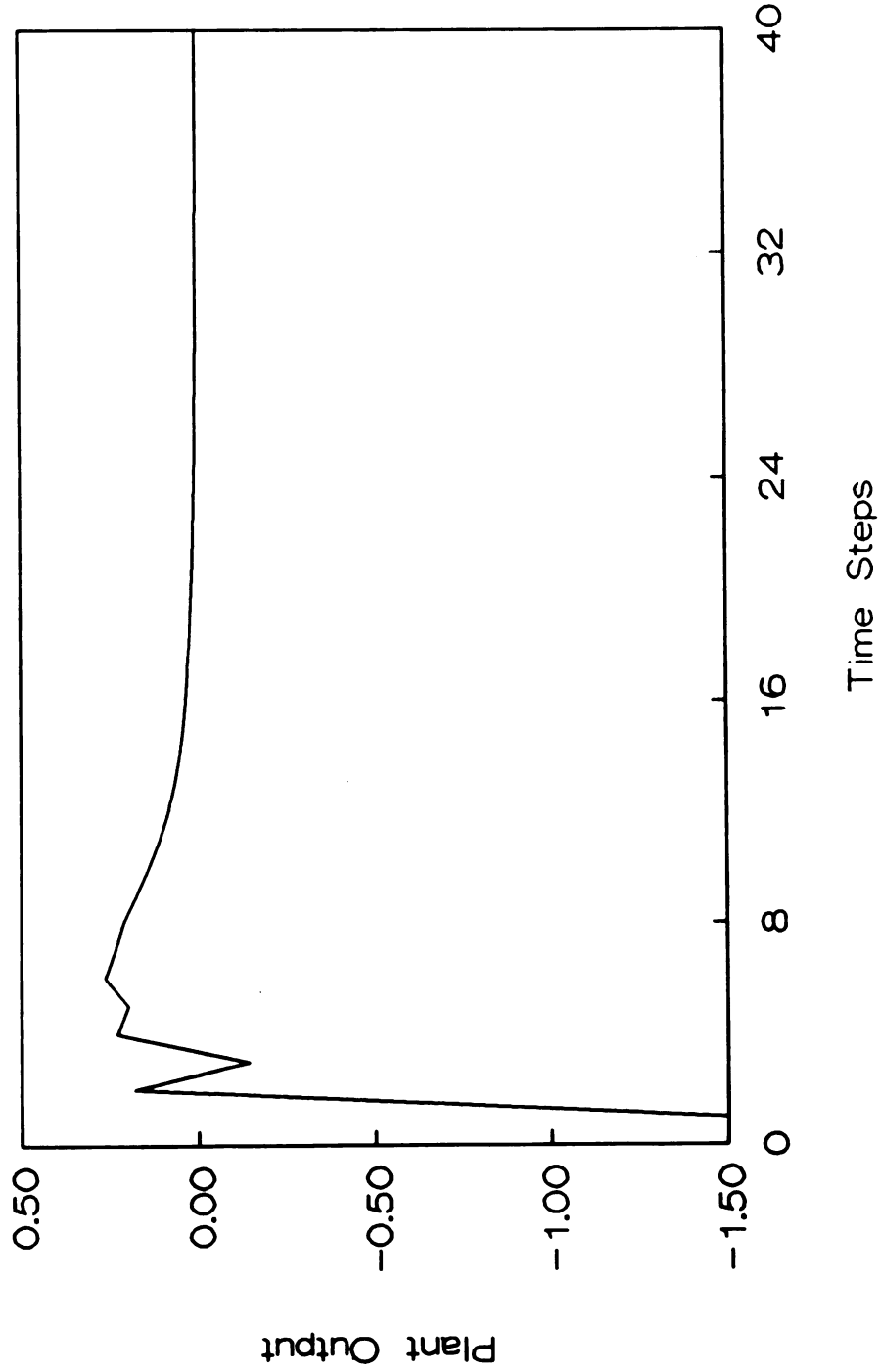


Figure 6.10 Regulation Error
Using NN with bias weights

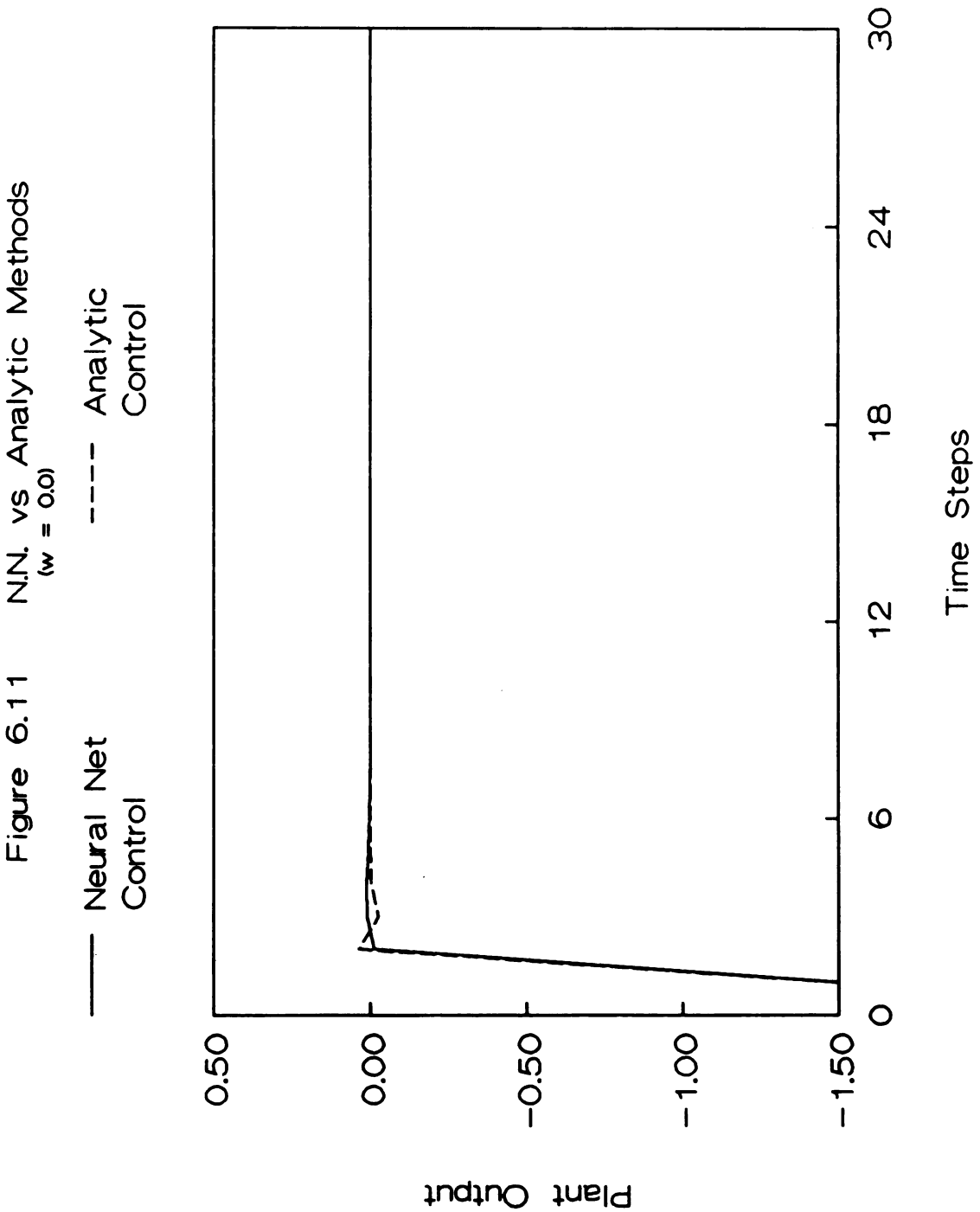

N.N. vs Analytic Methods
(w = 0.0)
Neural Net Control
Analytic Control
Plant Output
0.50
0.00
-0.50
-1.00
-1.50
0
6
12
18
24
30
Time Steps

Figure 6.11

Figure 6.12 N.N. vs Analytic Methods

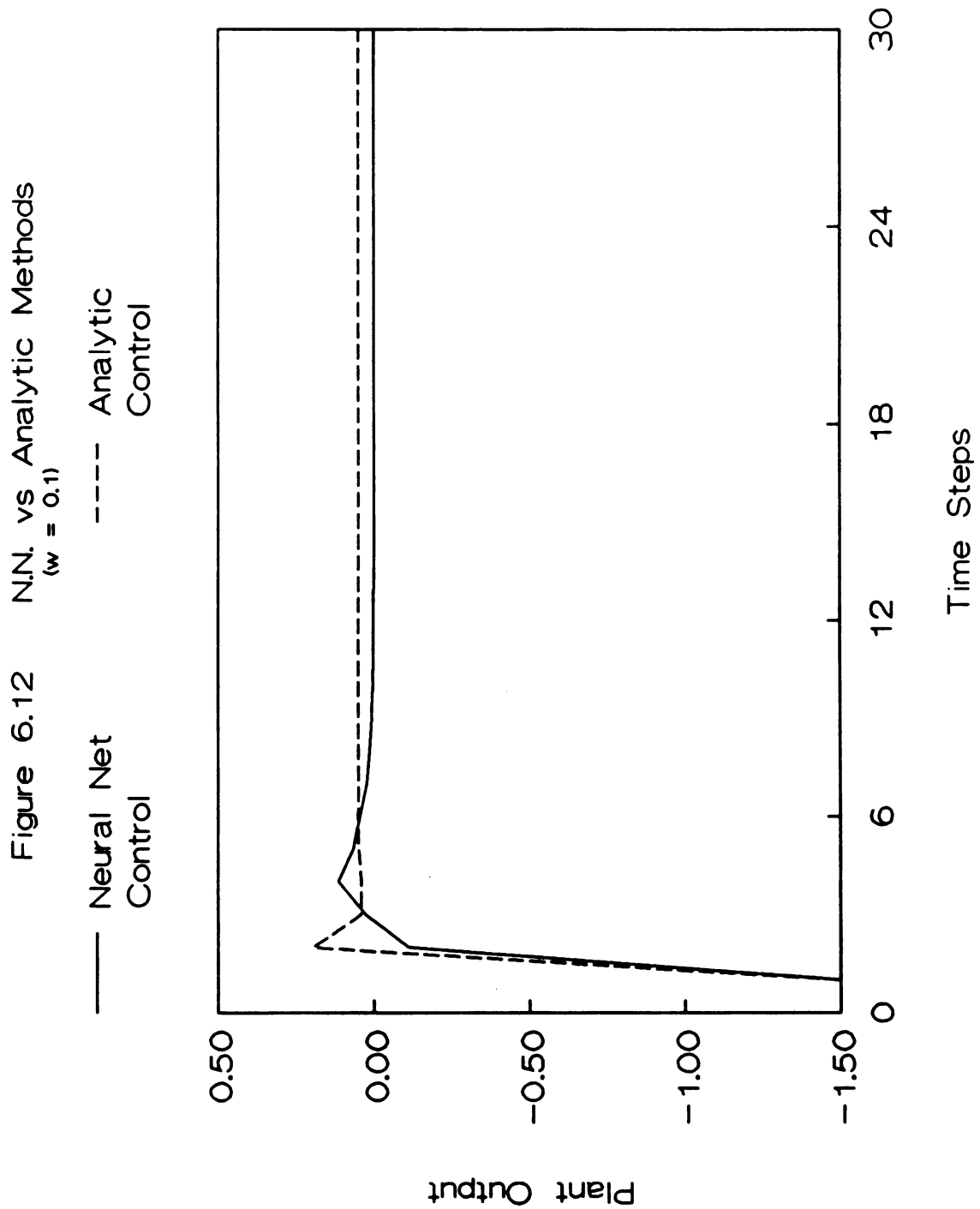

N.N. vs Analytic Methods
(w = 0.1)
Neural Net Control
Analytic Control
Plant Output
0.50
0.00
-0.50
-1.00
-1.50
0
6
12
18
24
30
Time Steps

Figure 6.12

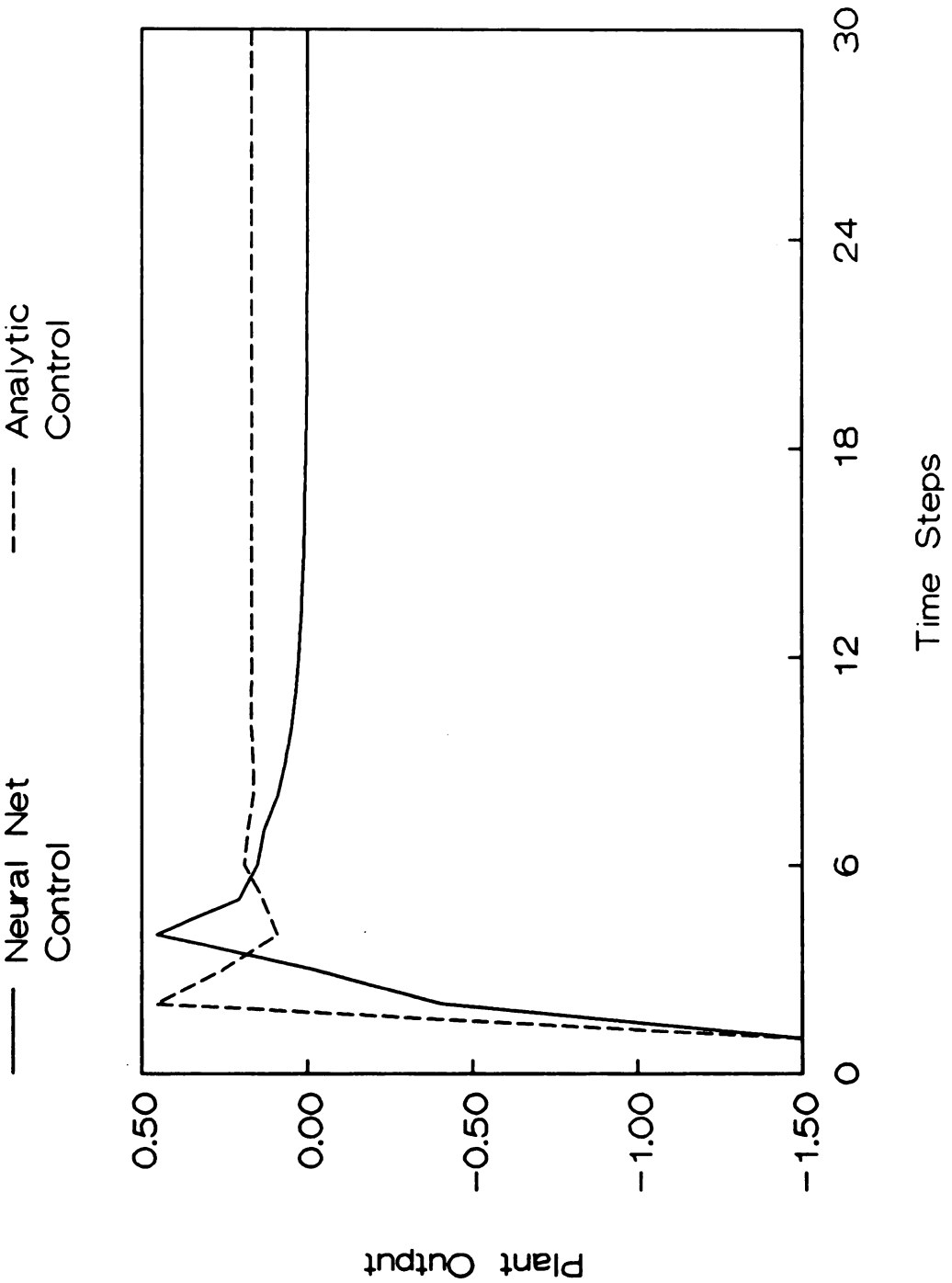


Figure 6.13 N.N. vs Analytic Methods
(w = 0.4)
Neural Net Control
Analytic Control
Plant Output
0.50
0.00
-0.50
-1.00
-1.50
0
6
12
18
24
30
Time Steps

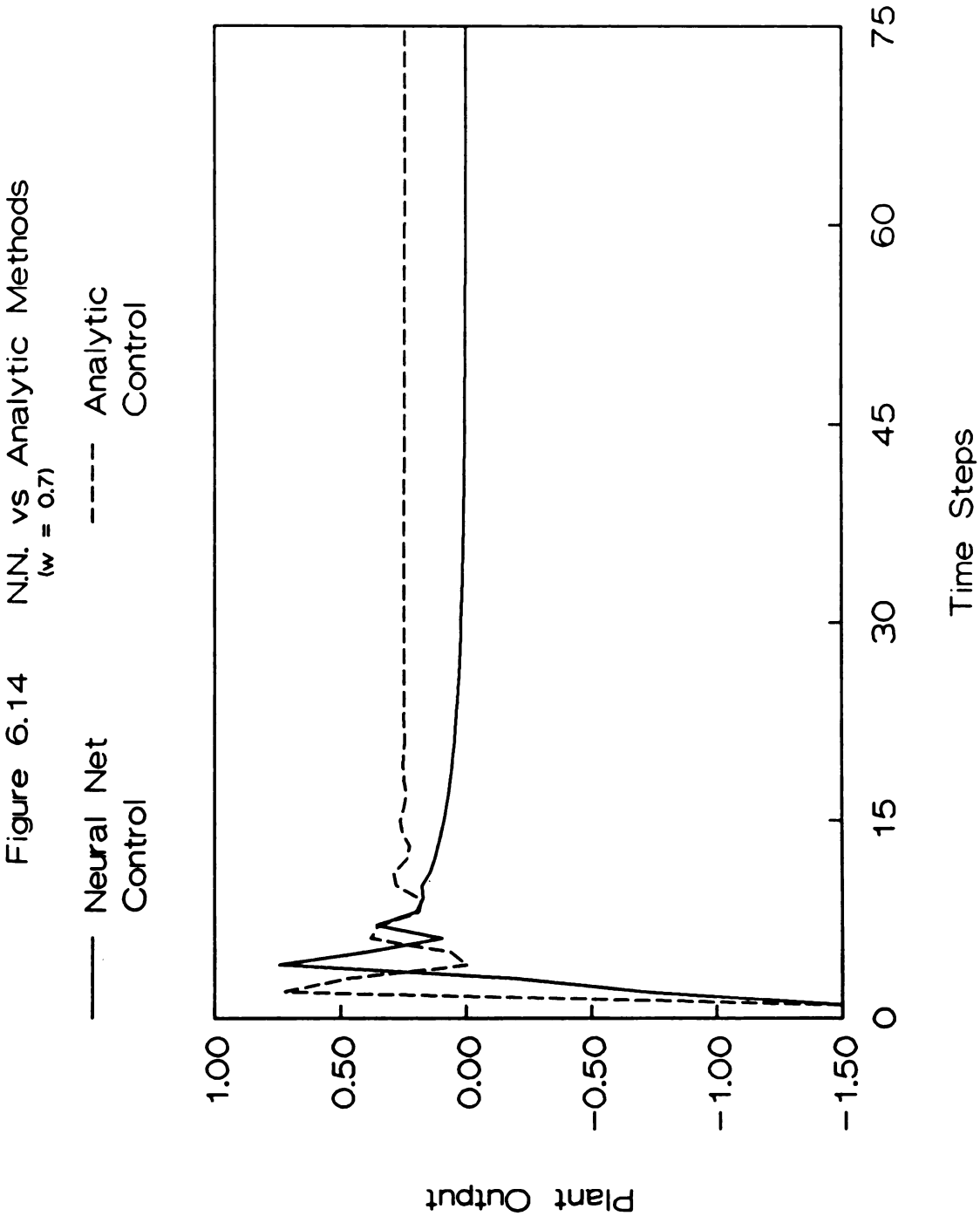


Figure 6.14 N.N. vs Analytic Methods ($w = 0.7$)

6.4 Tracking : 1. The Plant is Stable

The neural network NN(10, 10) is used in the adaptive tracking problem to control the plant

$$y_{k+1} = \frac{1.5y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.7sin[0.5(y_k + y_{k-1})] \cdot cos[0.5(y_k + y_{k-1}))] + 1.2u_k \quad (144)$$

to track a reference command. Since the nonlinear function $f_0(y_k, y_{k-1})$ in (144) is bounded uniformly over R^2, the plant is stable in the sense that given any sequence of bounded control $\{u_k\}$, the plant output is bounded. This means that the neural network will have plenty of time to learn to approximate the plant and generate adequate controls without the fear that the closed-loop system may blow out. Therefore, in this part of simulation, the neural network NN(10, 10) is directly applied to the control of the plant without any pretraining. From the data shown in Figure 6.15 to Figure 6.20, it is observed that the neural network can reduce the tracking error significantly during the early stage of the control process. But it takes much longer time to achieve "perfect" control. The same fact is also observed in section **6.1**, where the neural networks can reduce 85of error. Letting the neural networks go through some identification process will definitely help a lot when they are used in the control system. It is clear from Figure 6.15 to Figure 6.20 that certain part of reference signals cause persisting errors at the plant output. The neural networks may be pretrained intensively around these corners in order to achieve quick convergence in the control process.

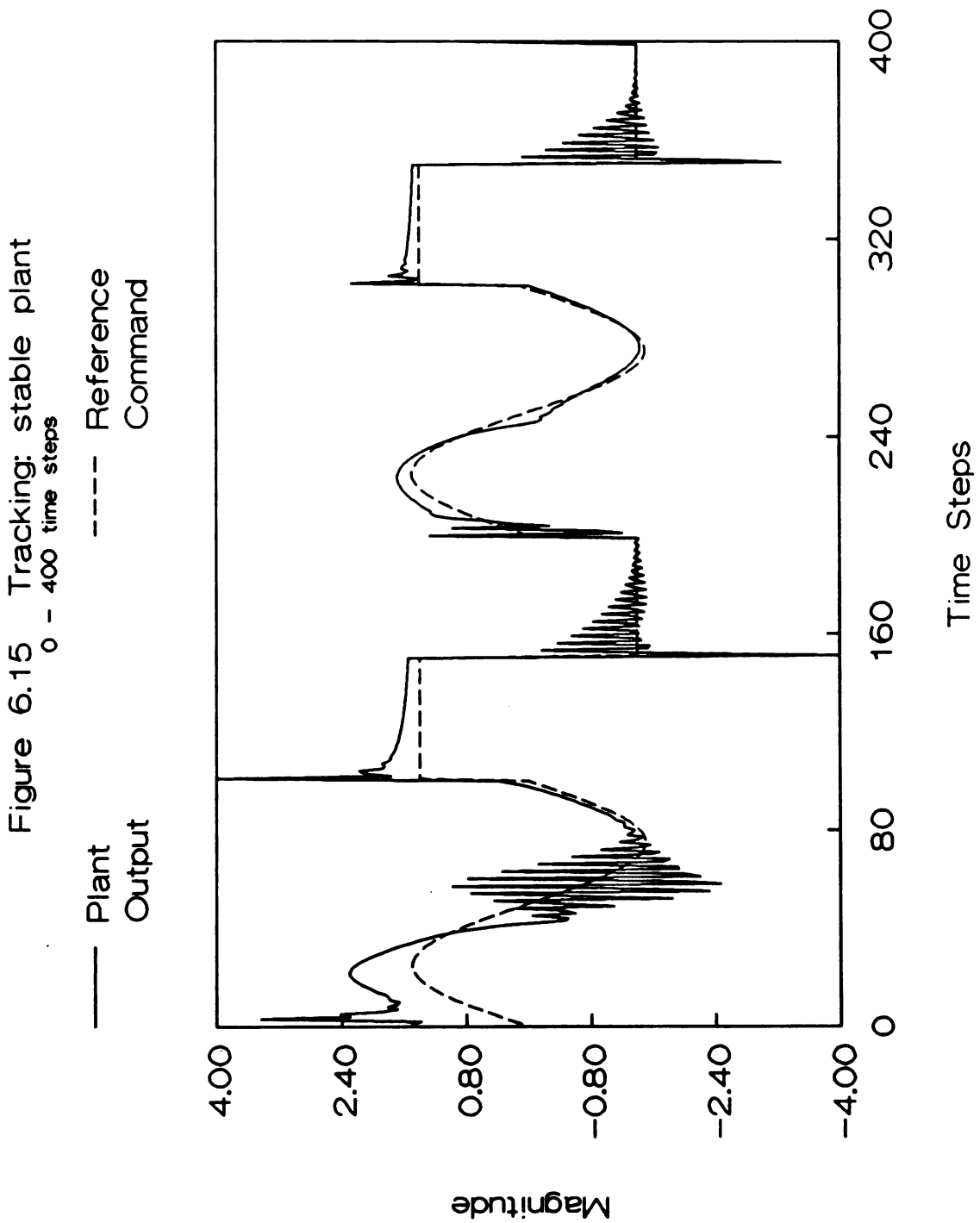


Figure 6.15 Tracking: stable plant
0 - 400 time steps

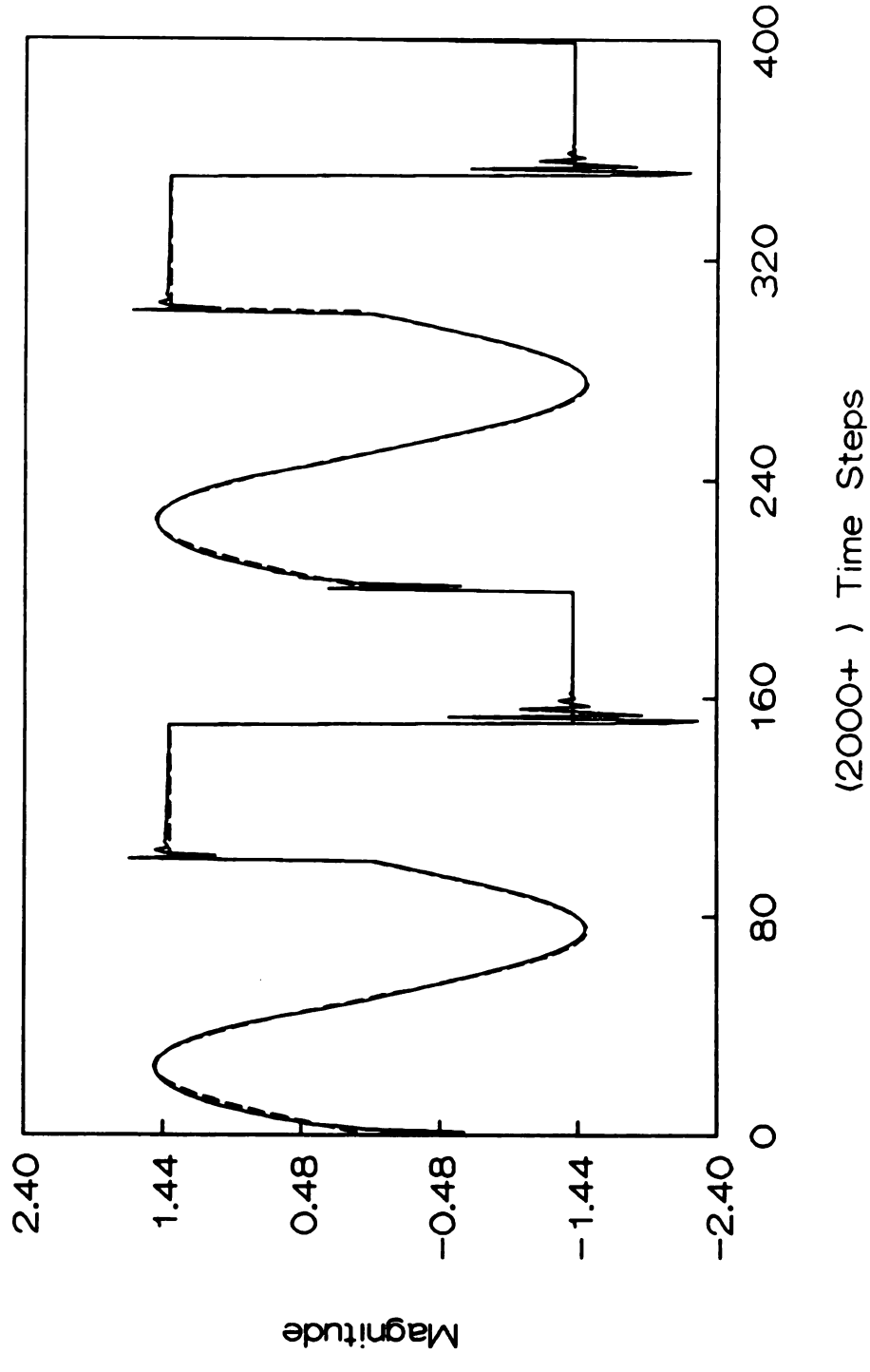


Figure 6.16 Tracking: stable plant
2000 - 2400 time steps

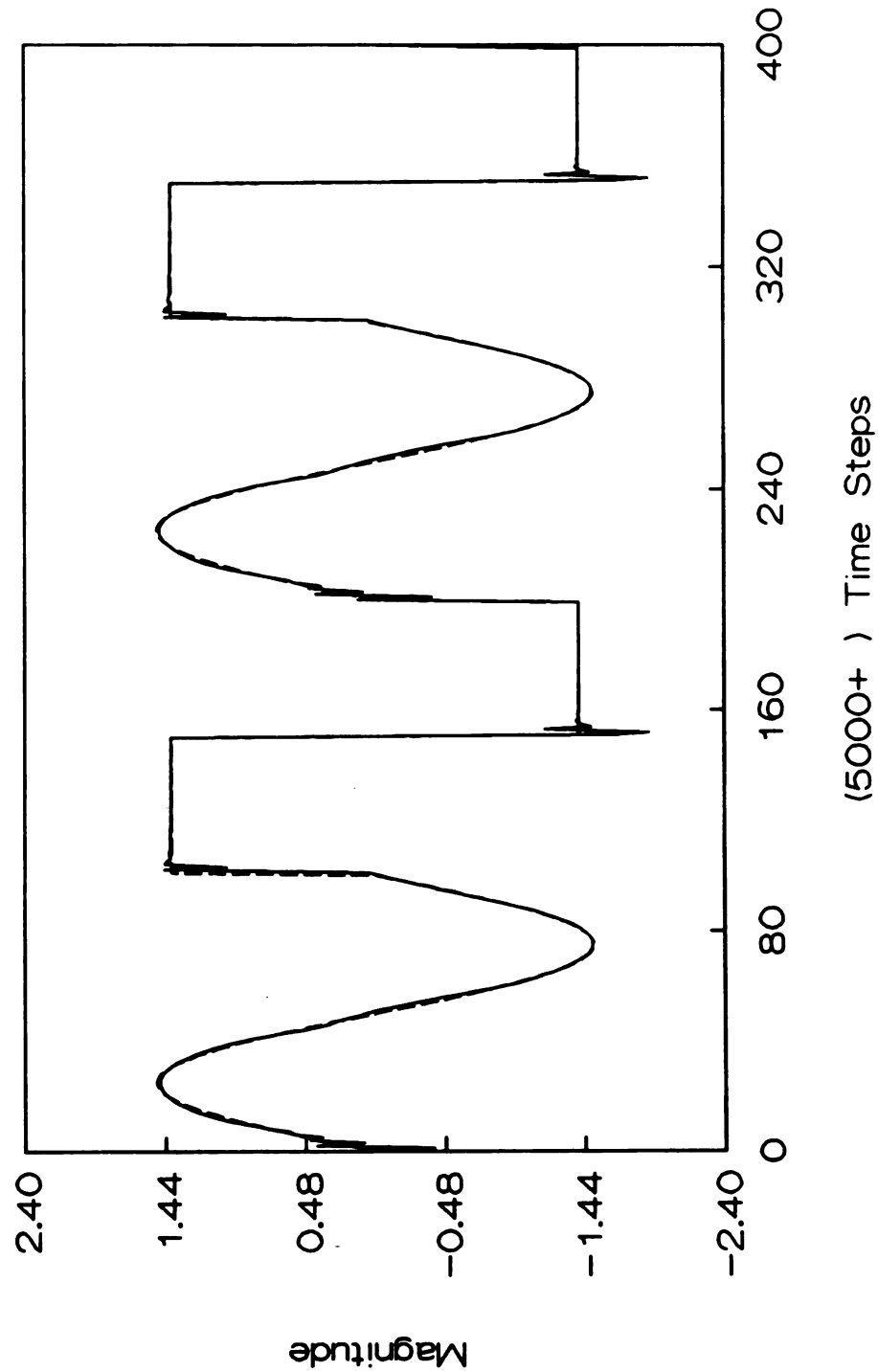


Figure 6.17 Tracking: stable plant
5000 - 5400 time steps

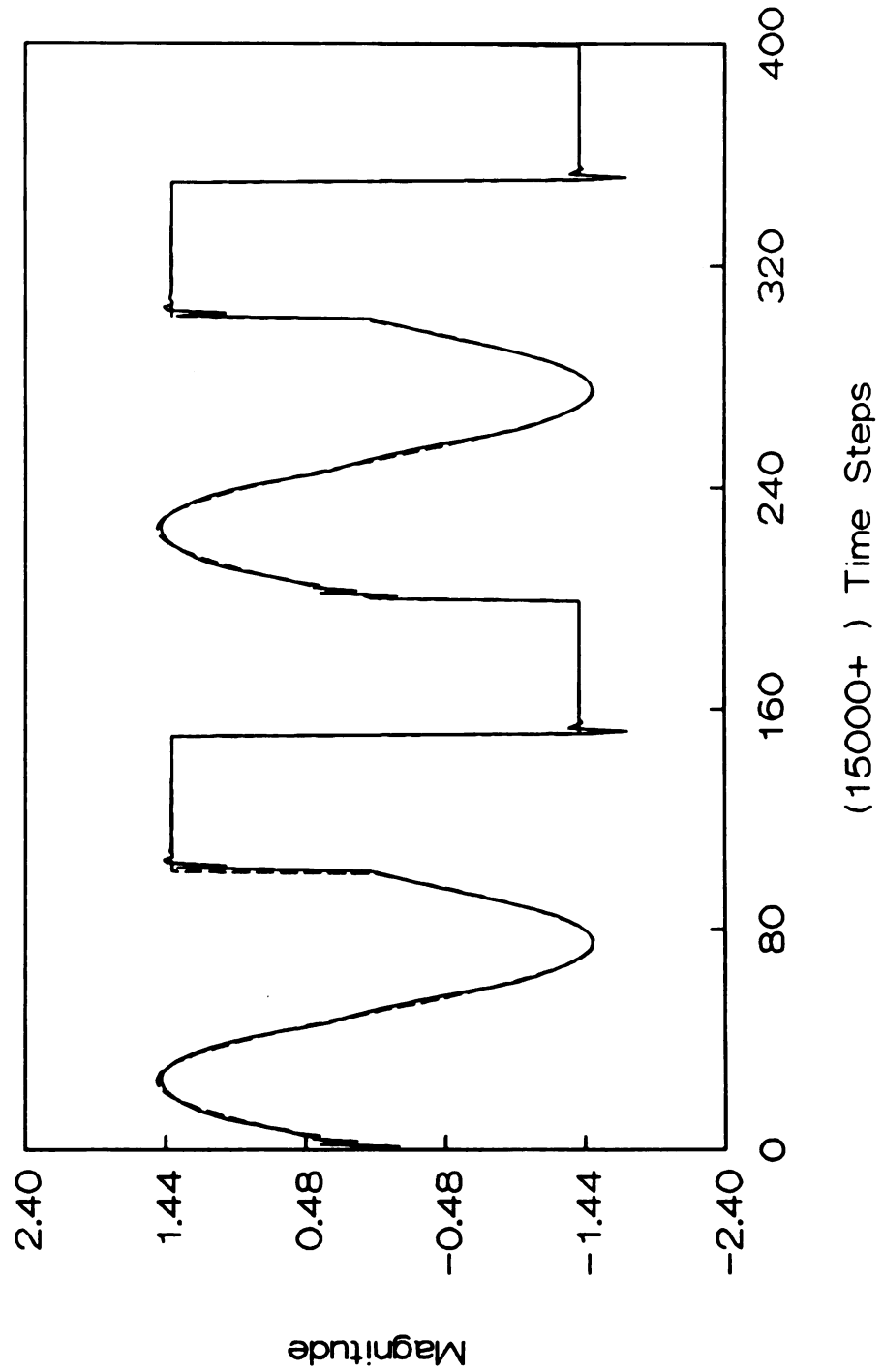


Figure 6.18 Tracking: stable plant
15000 - 15400 time steps

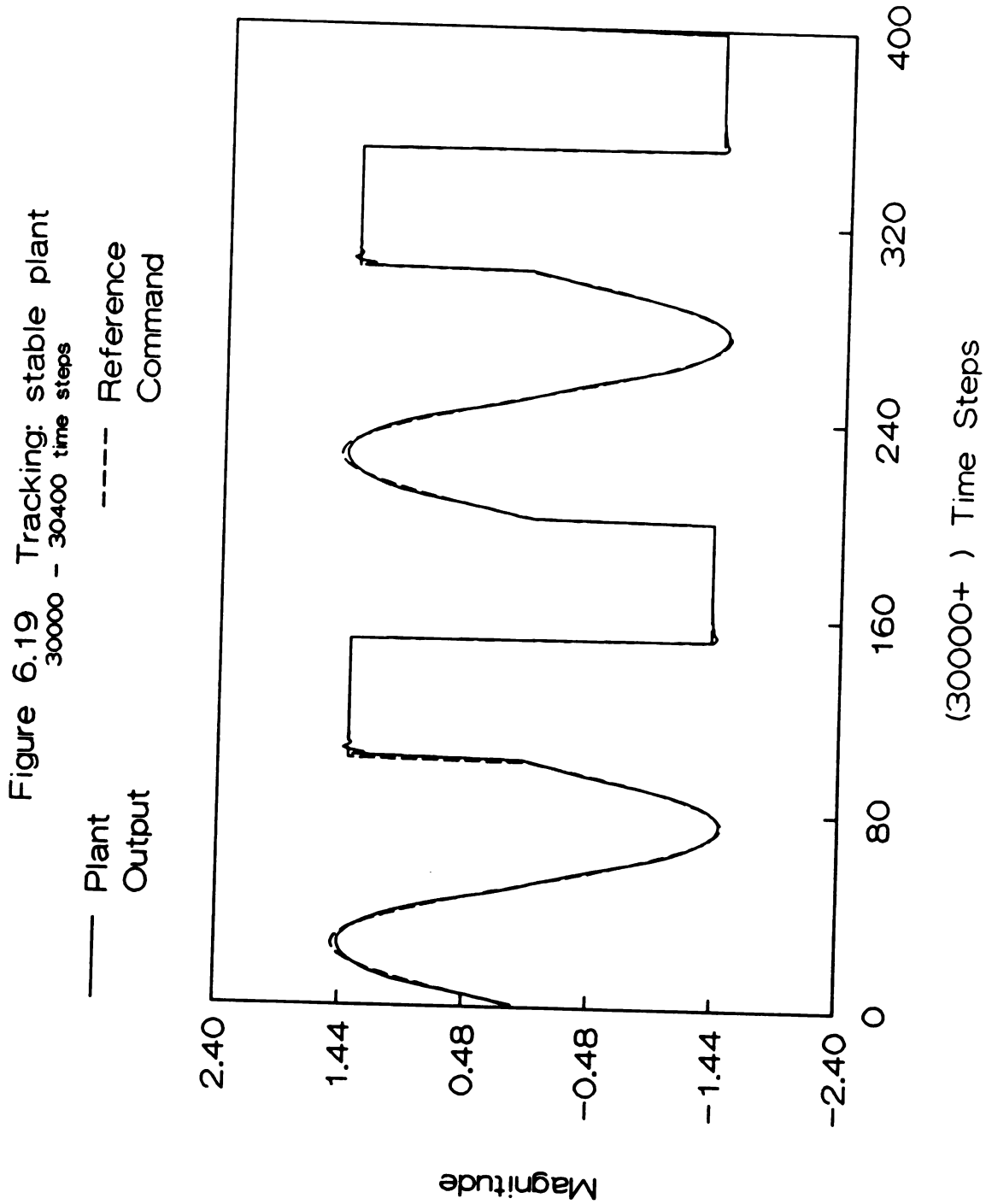


Figure 6.19 Tracking: stable plant
30000 - 30400 time steps
Plant Output
Reference Command
Magnitude
2.40
1.44
0.48
-0.48
-1.44
-2.40
0
80
160
240
320
400
(30000+) Time Steps

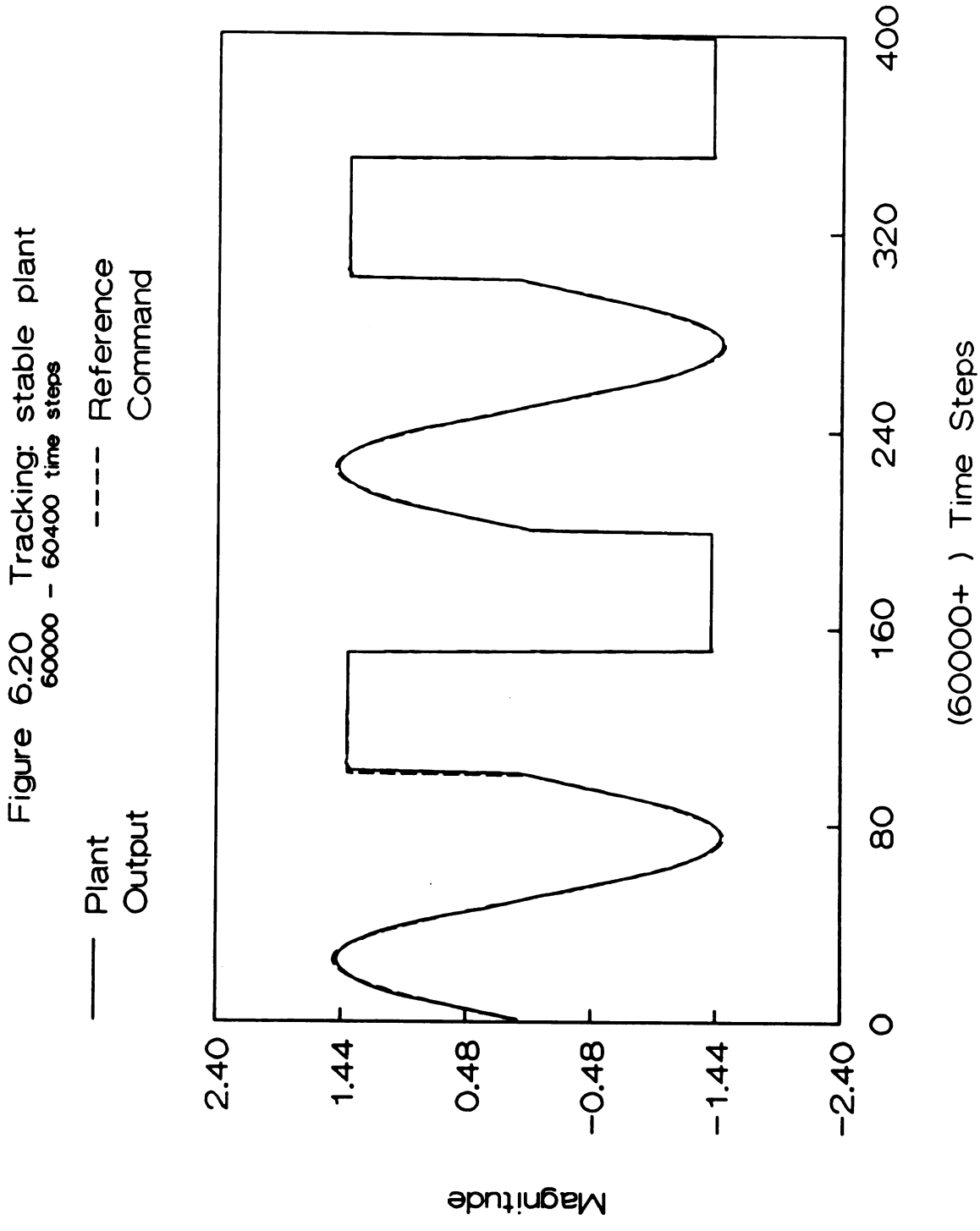


Figure 6.20 Tracking: stable plant
60000 - 60400 time steps

6.5 Tracking : 2. The Plant is Unstable

The plant used in this part of simulation is

$$y_{k+1} = 0.2y_k^2 + 0.2y_{k-1} + 0.4sin[0.5(y_k + y_{k-1})] \cdot cos[0.5(y_k + y_{k-1}))] + 1.2u_k \quad (145)$$

The plant (145) is unstable in the sense that given a sequence of uniformly bounded controls $\{u_k\}$, the plant output may diverge. An example is shown in Figure 6.21, where the plant output runs away after the step input $u_k = 0.9$, $\forall\, k \geq 0$ is applied to the plant.

In this section we are going to use the neural network NN(10, 10) to control the unstable plant to track a bounded reference command. Figure 6.22 shows the result when NN(10, 10) is applied to control (145) without being pretrained. The plant output runs away in the first 10 time steps. Since the initial weights of the neural network, including $\hat{g}_0(0)$ are chosen small (between -0.1 and 0.1) as mentioned in section **6.1**, the magnitude of the control u_k, which is defined as $\frac{\hat{f}_0(X(k))+v(k)}{\hat{g}_0(k)}$, can be very large initially, forcing the plant output to grow larger and larger. Hence, the system blows out before the neural network has a chance to learn to approximate the the plant.

As a remedy to the situation above, the neural network is trained for 2000 cycles in the hope that $\hat{g}_0$ may come close to 1.2 after the training. During the training process, u_k is selected randomly from the interval [-1.2, 1.2]. The results of applying the partially trained neural network to control the the plant are shown in Figures 6.23 to 6.25. It seems that good control is not achieved at the beginning of the step reference command. The plant response when controlled by a neural network which is pretrained for 10000 cycles is presented in Figure 6.26.

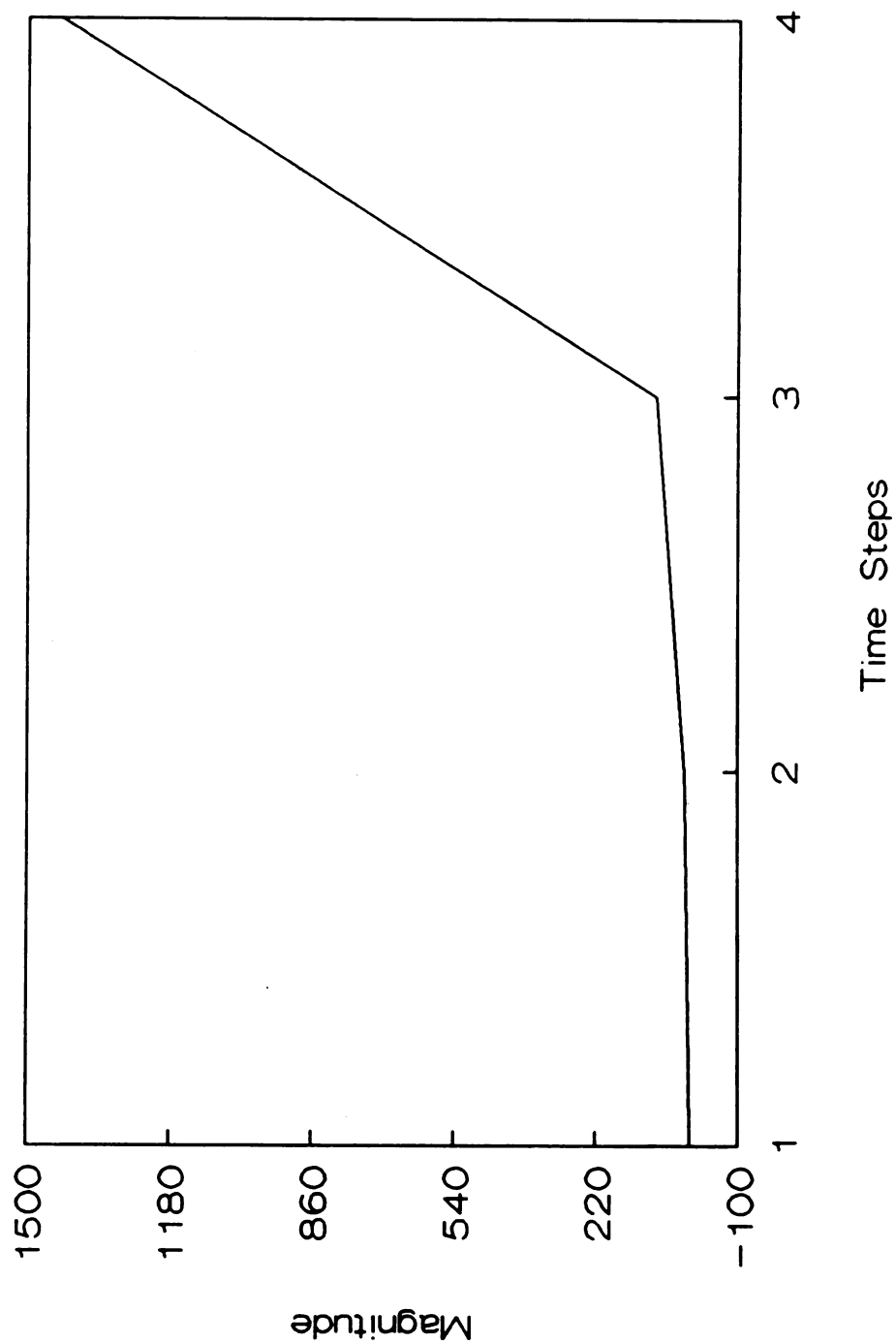


Figure 6.2.1 The unstable plant
the plant response of a step input

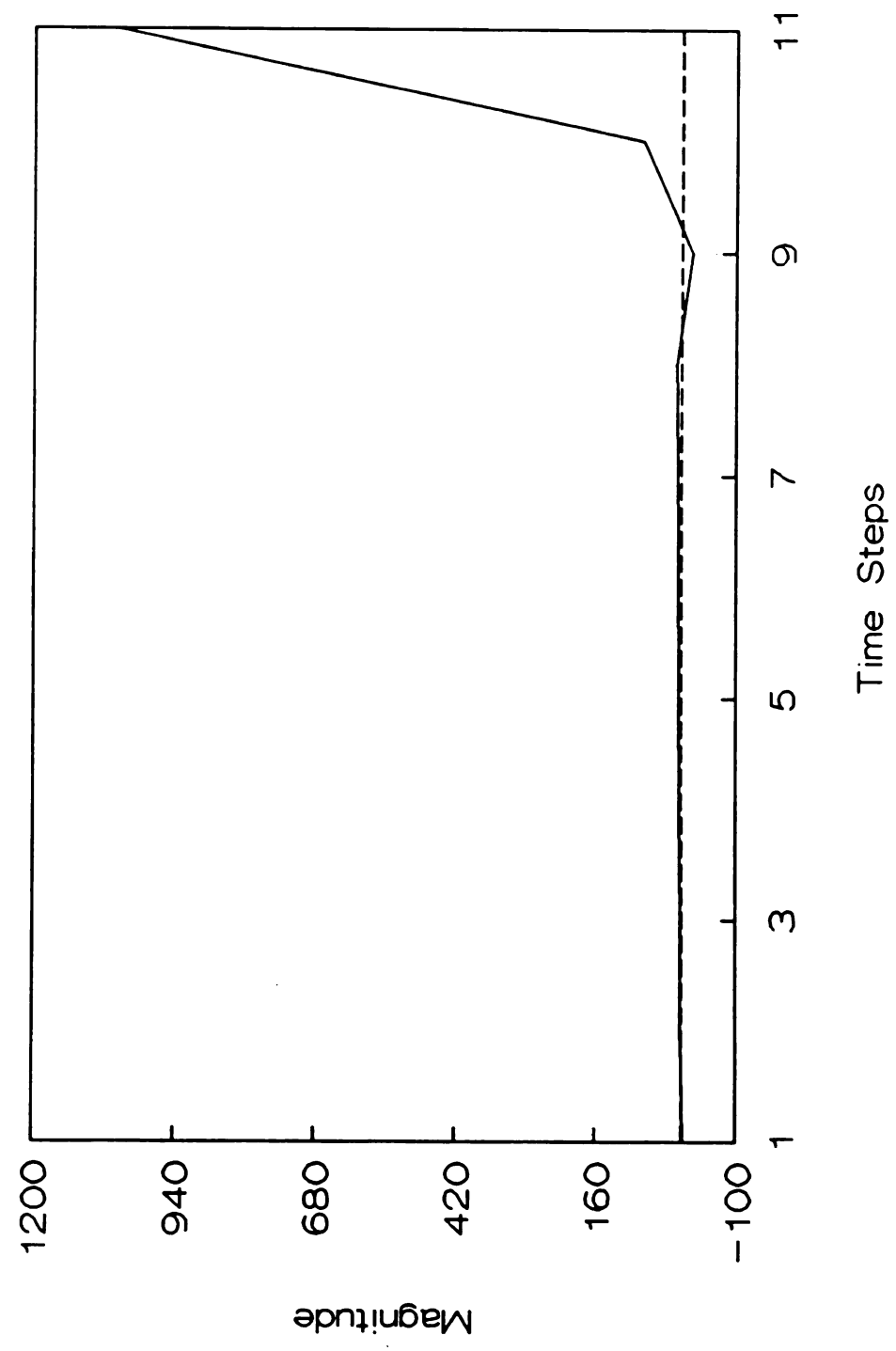


Figure 6.22 Tracking: unstable plant
The neural network is not pretrained

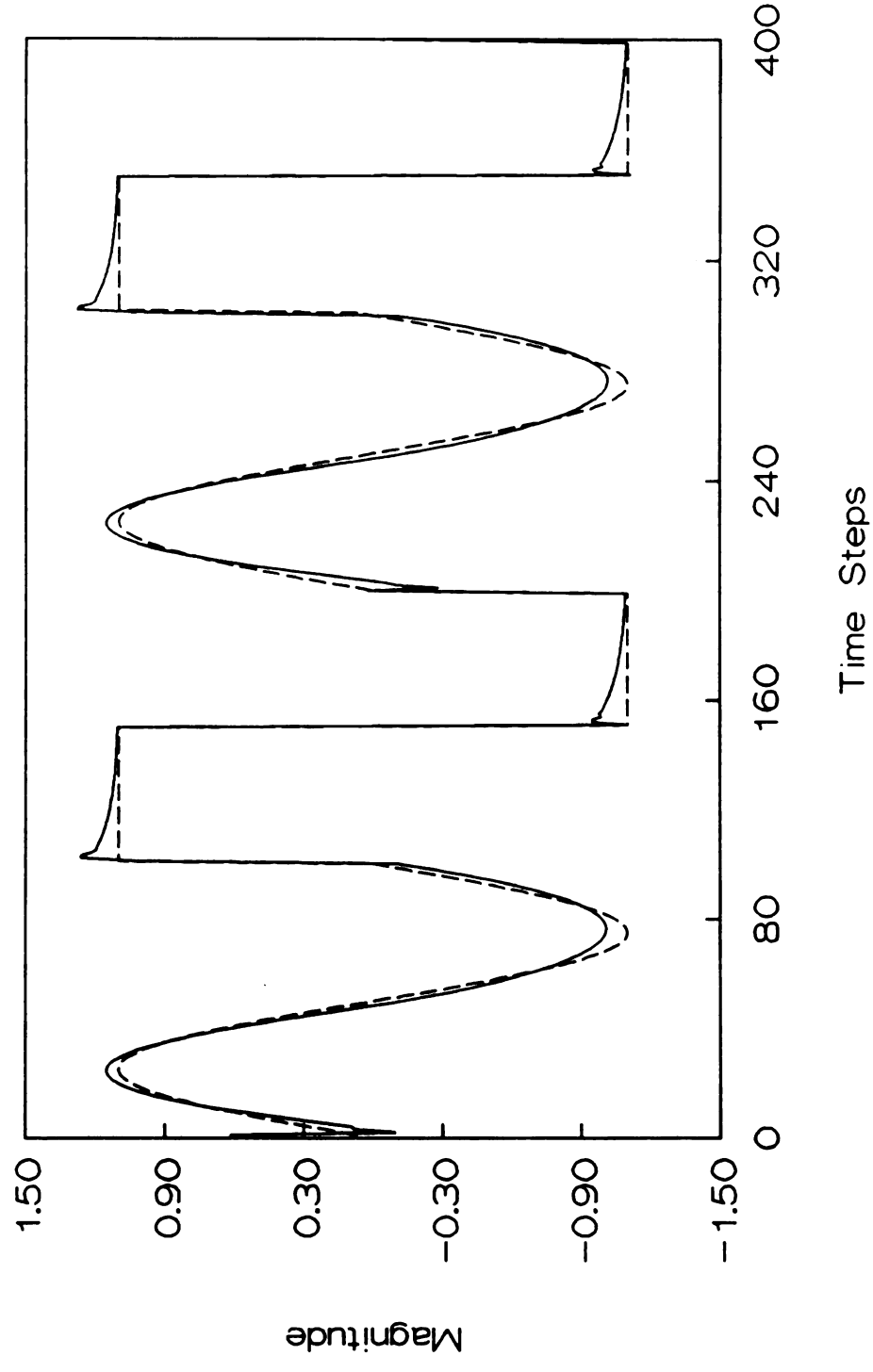


Figure 6.23 Tracking: unstable plant
The neural network is pretrained

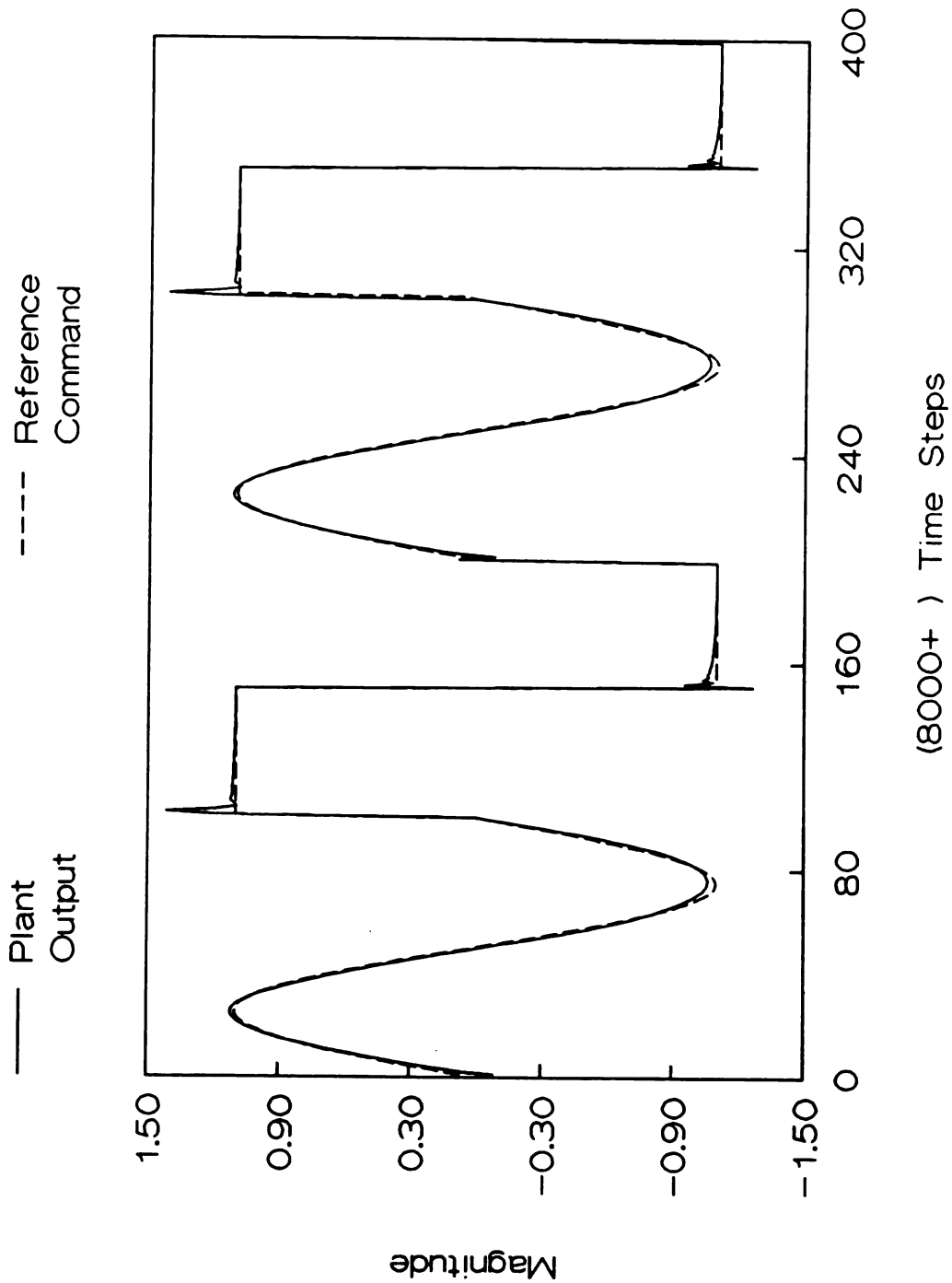


Figure 6.24 Tracking: unstable plant
The neural network is pretrained

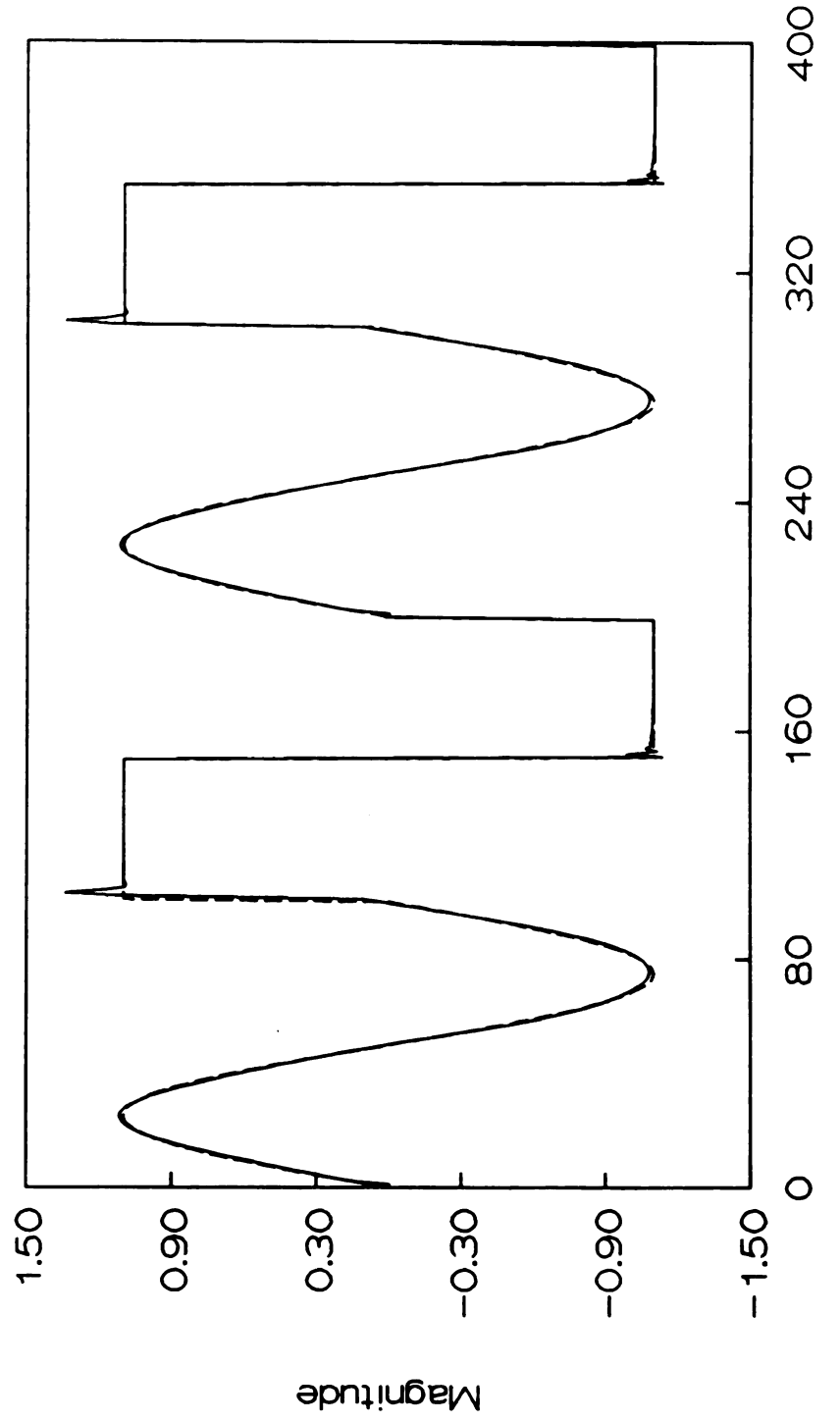


Figure 6.25 Tracking: unstable plant
The neural network is pretrained

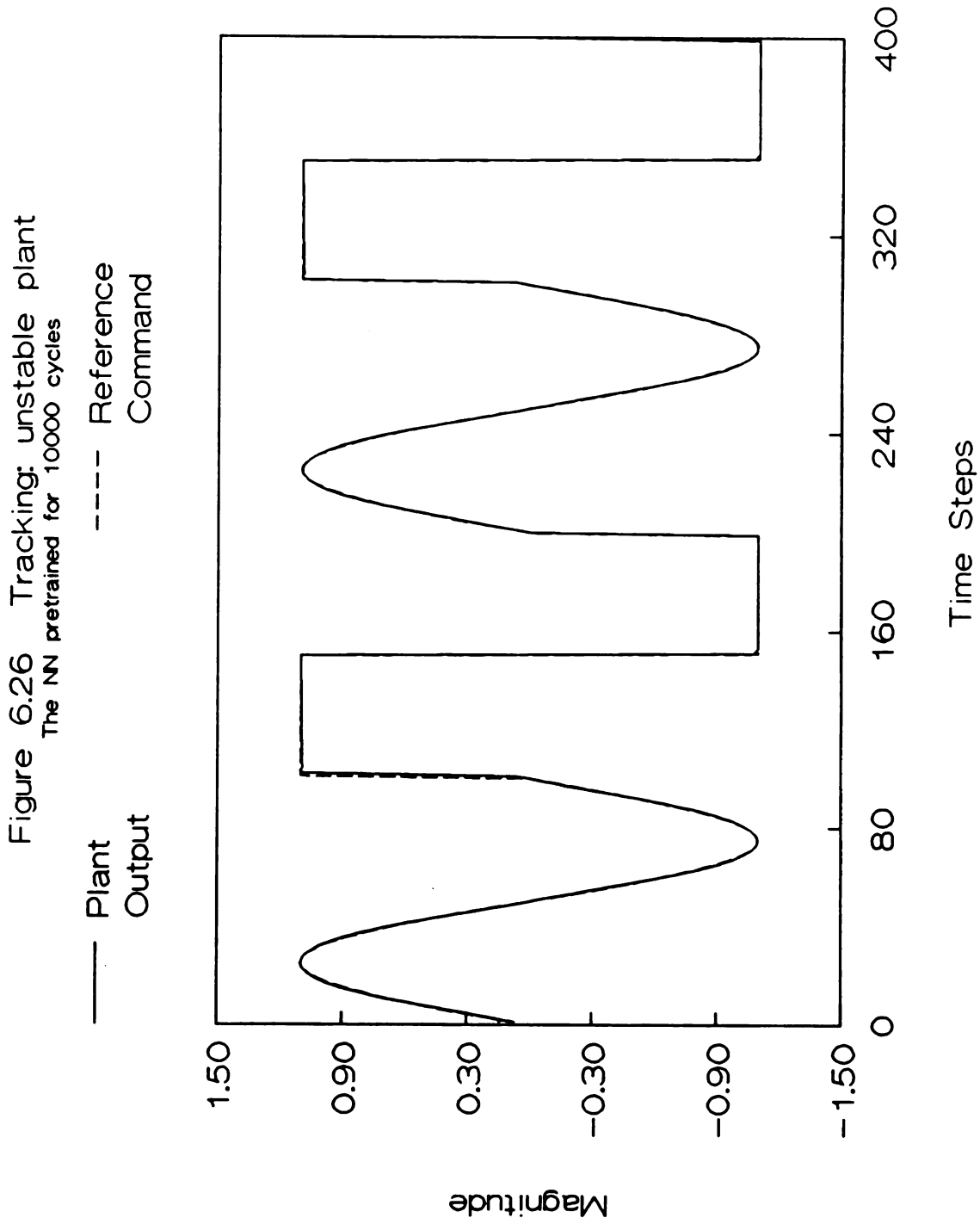
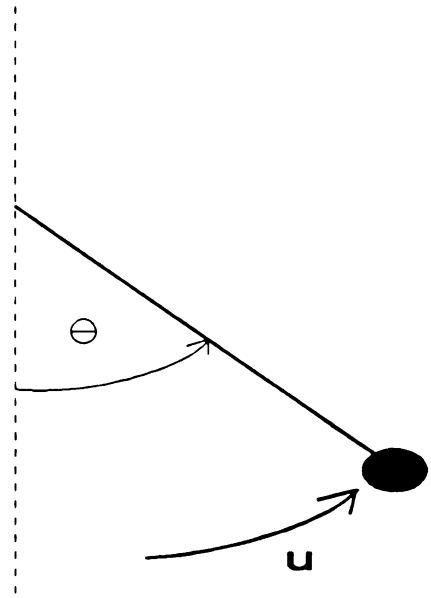


Figure 6.26 Tracking: unstable plant
The NN pretrained for 10000 cycles

6.6 Controlling a Relative-Degree-Two System: 1. the Pendulum

In this section we are going to apply the neural network to control a pendulum (Figure 6.27).



Suppose that the equation of motion of the pendulum is

$$\ddot{\theta}(t) + \dot{\theta}(t) + sin(\theta(t)) = u(t) \tag{146}$$

To discretize the system equation, let T be the sampling period. Then (146) is discretized (using *Euler's Rule*) as

$$\theta_{k+1} = (2-T)\theta_k + (T-1)\theta_{k-1} - T^2 sin(\theta_{k-1}) + T^2 u_{k-1} \tag{147}$$

It is verified through simulation that we can set $T = 0.3$ for the discretized model to be a good approximation of (146). In order to define the control, the same transformation as described in chapter **2** is performed.

$$\begin{aligned} \theta_{k+2} &= (2-T)\theta_{k+1} + (T-1)\theta_k - T^2 sin(\theta_k) + T^2 u_k \\ &= (2-T)[(2-T)\theta_k + (T-1)\theta_{k-1} - T^2 sin(\theta_{k-1}) \end{aligned}$$

$$
\begin{aligned}
&+ T^2 u_{k-1}] + (T-1)\theta_k - T^2 sin(\theta_k) + T^2 u_k \\
= \; & [(2-T)^2 + (T-1)]\theta_k + (2-T)(T-1)\theta_{k-1} - T^2 sin(\theta_k) \\
& - T^2(2-T)sin(\theta_{k-1}) + T^2(2-T)u_{k-1} + T^2 u_k \qquad (148)
\end{aligned}
$$

which is modeled by the neural network

$$\hat{\theta}_{k+2} = \hat{f}(\theta_k, \theta_{k-1}, u_{k-1}, W_k) + \hat{g}_k u_k \qquad (149)$$

Since the control u_k is multiplied by T^2, it is difficult to reduce T from 0.3 to 0.2 in the simulation. At each time step, the control u_k is generated from (149). The updating of the weights is based on the error between θ_{k+1} and

$$\hat{\theta}_{k+1} = \hat{f}(\theta_{k-1}, \theta_{k-2}, u_{k-2}, W_k) + \hat{g}_k u_{k-1} \qquad (150)$$

Figure 6.28 shows the result of using the neural network NN(10, 10) to control the pendulum to track a sinusoidal reference command. Since the pendulum is stable around the origin, the neural network is not trained before being used in the control system. The pendulum angle tracks the reference command quickly, although slight errors are still observed at the end of 1200 time steps. Figure 6.29 shows that even at 10000 time steps, the errors are still detectable.

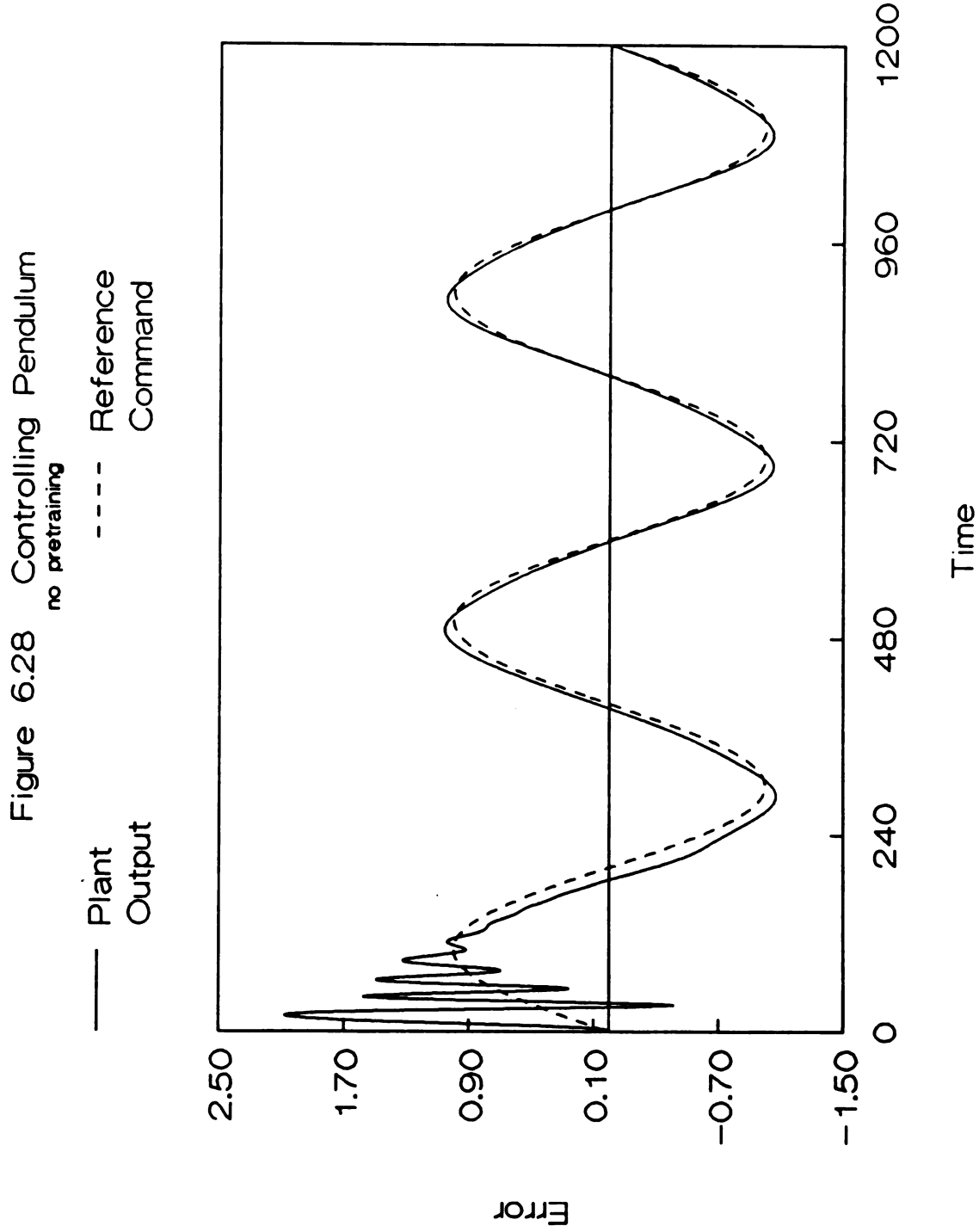

Controlling Pendulum
no pretraining
Plant Output
Reference Command
Error
2.50
1.70
0.90
0.10
−0.70
−1.50
0
240
480
720
960
1200
Time

Figure 6.28

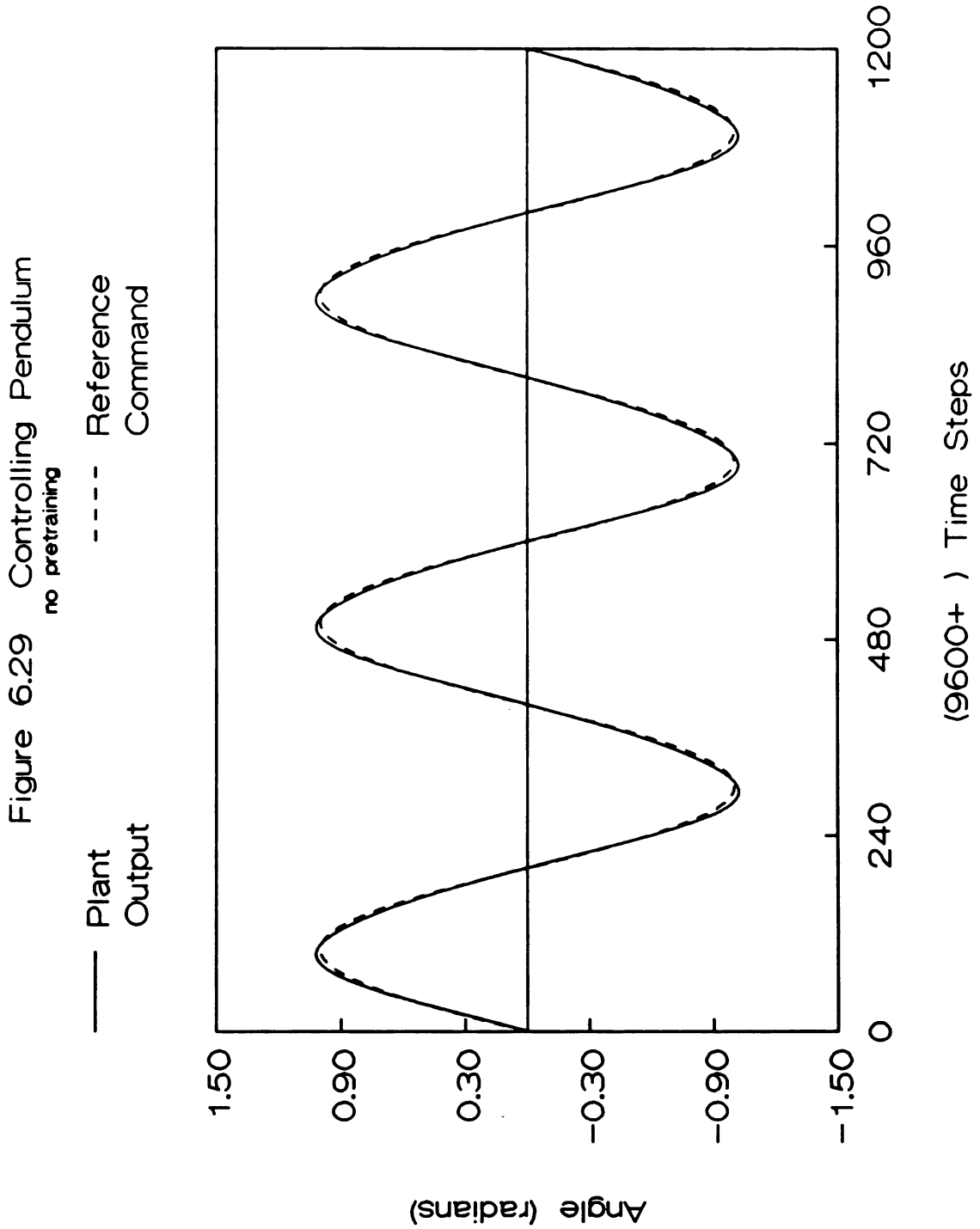


Figure 6.29 Controlling Pendulum
no pretraining

6.7 Controlling a Relative-Degree-Two System: 2. the Inverted Pendulum

The pendulum equation has been described in (146).

Let

$$y = \theta - \pi$$

to be the output of the system. The system equation is rewritten in terms of the new variable as

$$\ddot{y}(t) + \dot{y}(t) - sin(y(t)) = u(t) \tag{151}$$

which becomes

$$y_{k+1} = (2 - T)y_k + (T - 1)y_{k-1} + T^2 sin(y_{k-1}) + T^2 u_{k-1} \tag{152}$$

after being discretized.

The simulation setup is very similar to that of section **6.6**, except that the origin is unstable. The simulations presented in Figures 6.30 to 6.35 are conducted under the assumption that the coefficient associated with u_k, i.e. T^2, is known. Otherwise it is very difficult to produce any reasonable result. Any small error in the estimation of T^2 can produce large error in the control, driving the plant output from the origin which is already an unstable equilibrium point. It is obvious that controlling the inverted pendulum is much more difficult.

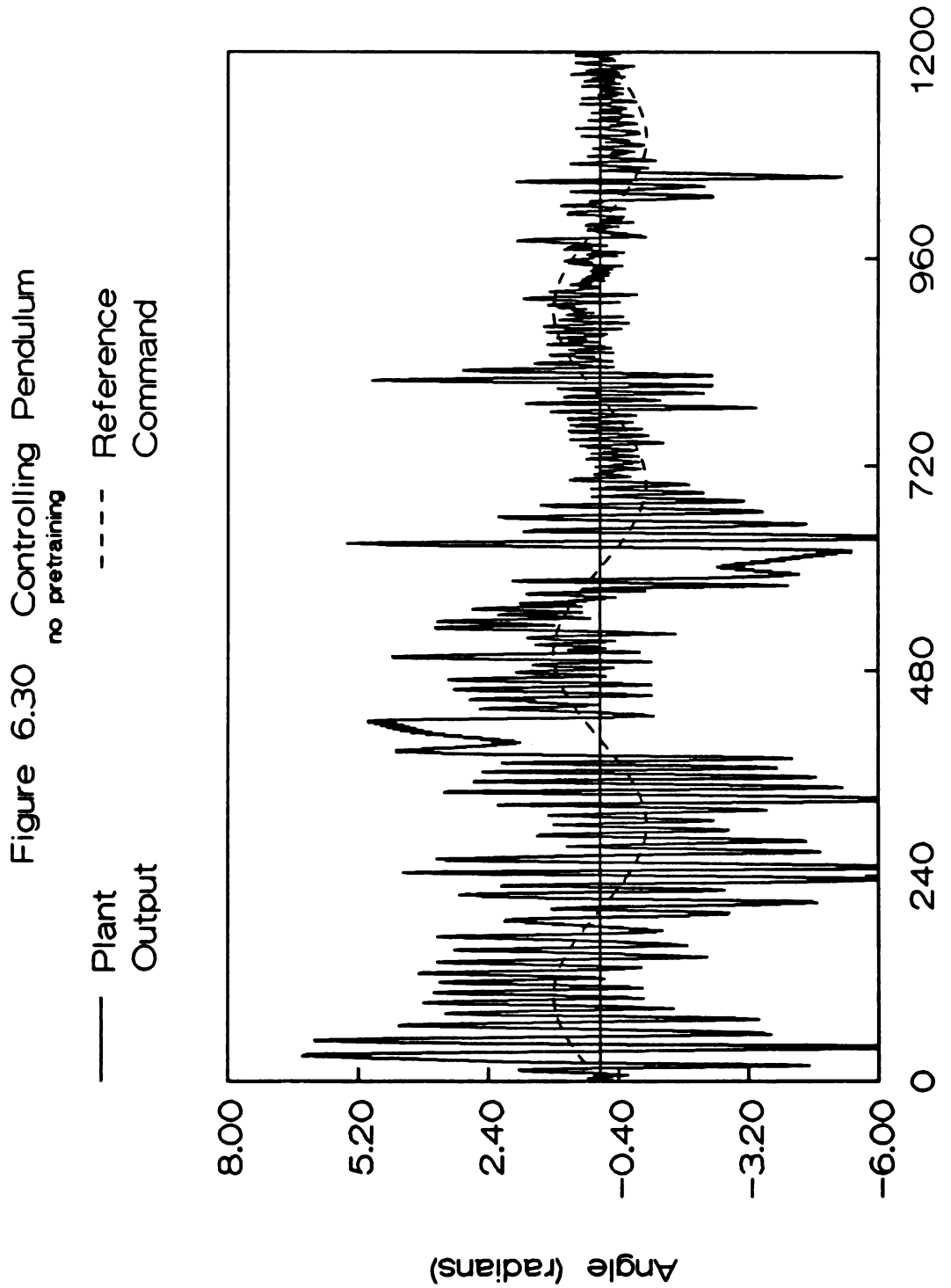


Figure 6.30 Controlling Pendulum
no pretraining

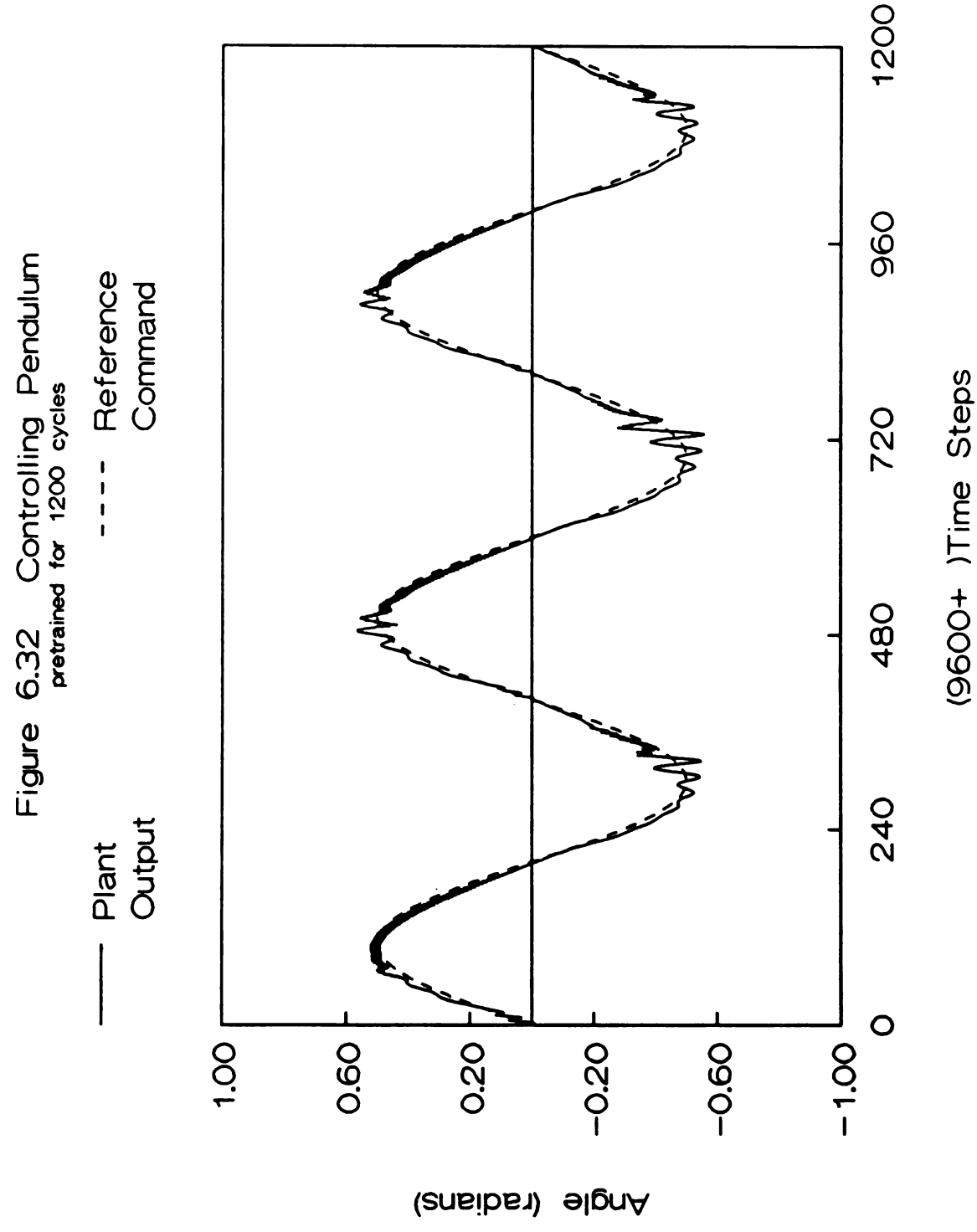


Figure 6.32 Controlling Pendulum
pretrained for 1200 cycles
Plant Output
Reference Command
Angle (radians)
1.00
0.60
0.20
−0.20
−0.60
−1.00
0
240
480
720
960
1200
(9600+)Time Steps

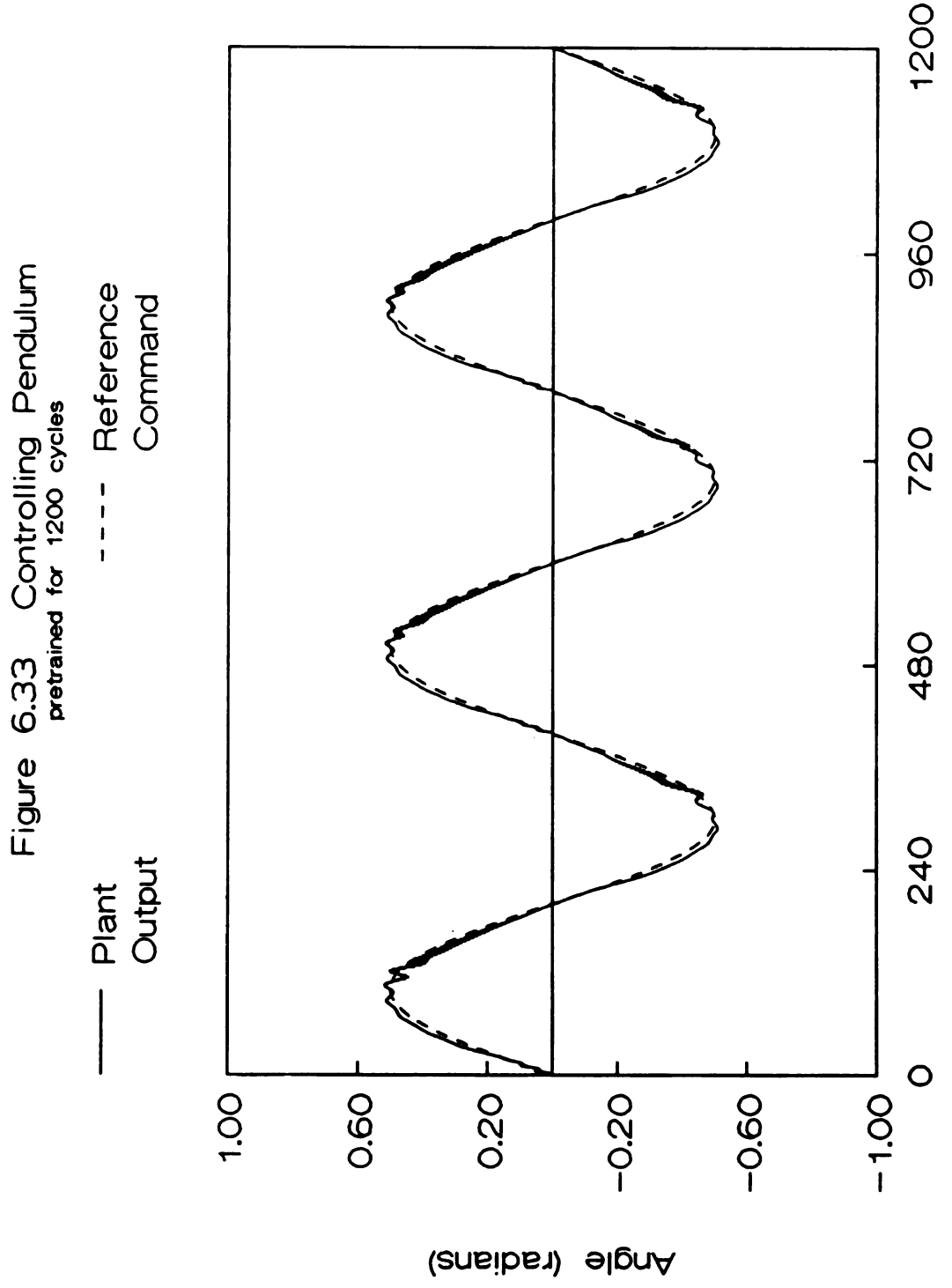


Figure 6.33 Controlling Pendulum
pretrained for 1200 cycles

7 Conclusion

In this research, layered neural networks are applied to the adaptive control of single-input/single-output nonlinear systems. Under different assumptions, two local convergence results are shown in chapter **4** and chapter **5**. The simulation results in chapter **6** suggest that neural networks can be powerful tools for practical identification and control problems.

The advantage of using neural networks in identification and adaptive control problems is that neural networks are universal approximators. The neural networks can learn to approximate various nonlinear functions as long as they contain enough nonlinear hidden neurons.

The disadvantage is that the learning of neural networks to approximate nonlinear functions is a very time-consuming process. In the case that the plant is stable and nicely behaved, the neural network controller usually can achieve satisfying control. In the case that the plant is unstable, the neural network may need to be trained before being applied to the control system. Otherwise, the performance of the closed-loop system can be very bad because the neural network may not learn quickly enough on line to produce suitable controls.

The speed of "learning " in the part of the neural network is the bottleneck of the control system. Slow learning of neural networks can limit the usefulness of their applications to real world control problems. How to improve the learning speed is an important issue to look at.

Another interesting issue is to generalize the current control scheme to multi-input / multi-output setup. Many important systems are highly coupled multi-input / multi-output systems, e.g. robotics. The control problem for MIMO systems is more involved. The neural network architecture needs also be reconstructed to ac-

commodate MIMO systems.

In my point of view, there are two research directions in applying neural networks to control problems. One is to combine neural networks with analytic control algorithms. This thesis is an example of this approach. In addition to layered neural networks, other neural network paradigms may directly or indirectly help to solve control problems. The other research direction is to use neural networks to reproduce biological control system, which is usually called sensory-motor control. This is a relatively new research area for control engineers. Since we can control our hands and fingers easily, it is very logical to try to understand how those excellent controls are accomplished.

References

[1] D. Rumelhart, G. E. Hinton, & R. J. Williams, " Learning Internal Representations by Error Propagation," In Rumelhart and McClelland (Ed), *Parallel Distributed Processing*, Vol. 1, MIT Press, 1986.

[2] B. Widrow, & R. Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *IEEE Computer*, March 1988.

[3] R. Hecht-Nielsen, "Theory of the Back-propagation Neural Network," *Proceedings Int'l Joint Conf. on Neural Networks*, pp. I-593–605, June 1989.

[4] D. Psaltis, A. Sideris, & A. A. Yamamura, "A Multilayered Neural Network Controller," *IEEE Control Systems Magazine*, April 1988.

[5] M.-S. Lan, "Adaptive Control of Unknown Dynamical Systems via Neural Network Approach," *Proceedings 1989 American Control Conference*, pp. 910–915.

[6] V. Zeman, R. V. Patel, & K. Khorasani, "A Neural Network Based Control Strategy for Flexible-Joint Manipulators," *Proceedings 1989 IEEE Conf. on Decision and Control*, pp. 1759–1764.

[7] W. Li & J. -J. E. Slotine, "Neural Network Control of Unknown Nonlinear Systems," *Proceedings 1989 American Control Conference*, pp. 1136–1141.

[8] K. S. Narendra & K. Parthasarathy, "Adaptive Identification and Control of Dynamical Systems using Neural Networks," *Proceedings 1989 IEEE Conf. on Decision and Control*, pp. 1737–1738.

[9] F.-C. Chen, "Back-propagation Neural Networks for Nonlinear Self-tuning Adaptive Control," *Proceedings 1989 IEEE int'l Symp. on Intelligent Control*, pp.

274–279. Also to appear in *IEEE Control System Magazine, Special Issue on Neural Networks for Control Systems*, April 1990.

[10] A. Guez & J. Selinsky, "A Neuromorphic Controller with a Human Teacher," *Proceedings IEEE 1988 Int'l Conf. on Neural Networks*, pp. II-595–602.

[11] W. T. Miller, F. H. Glanz & L. G. Kraft, "Application of a General Learning Algorithm to the Control of Robotics Manipulators," *Int'l J. of Robotics Research*, 6.2:84–98, 1987.

[12] C. W. Anderson, "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control Systems Magazine*, pp.31–37, April 1989.

[13] L. G. Kraft, E. An & D. P. Campagna, "Comparison of CMAC Controller Weight Update Laws," *Proceedings 1989 IEEE Conf. on Decision and Control*, pp. 1746–1747.

[14] F. Pourboghrat, "Neuromorphic Controllers," *Proceedings 1989 IEEE Conf. on Decision and Control*, pp. 1748–1749.

[15] A. Isidori, *Nonlinear Control Systems: An Introduction.* New York: Spring-Verlag, 1985.

[16] S. Monaco, D. Normand-Cyrot, "Minimum-Phase Nonlinear Discrete-Time Systems and Feedback Stabilization," *Proceedings 1987 IEEE Conf. on Decision and Control*, pp. 979–986.

[17] D. G. Taylor, P. V. Kokotovic, R. Marino & I. Kanellakopoulos, "Adaptive Regulation of Nonlinear Systems with Unmodeled Dynamics," *IEEE Transaction on Automatic Control*, Vol. 34, No. 4, pp. 405–412, April 1989.

[18] S. S. Sastry & A. Isidori, "Adaptive Control of Linearizable Systems," *IEEE Transaction on Automatic Control*, Vol. 34, No. 11, pp. 1123–1131, November 1989.

[19] K. J. Astrom & B. Wittenmark, *Adaptive Control*, Addison-Wesley, 1989.

[20] J. A. Anderson & E. Rosenfeld, *Neural Computing: Foundations and Research*, MIT Press, 1988.

[21] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, 1989.

[22] P. K. Simpson, *Artificial Neural Networks: Foundations, paradigms, applications, and implementations*, Elmsford, NY: Pergamon Press, 1990.

[23] M. L. Minsky & S. S. Papert, *Perceptron*, MIT Press, 1969.

[24] K. Funahashi, "On the Approximate Realization of Continuous Mapings by Neural Networks", *Neural Networks*, Vol. 2, pp. 183–192, 1989.

[25] G. Cybenko, "Approximation by Superpositions of a Sigmoidal function", Tech. Rep. No. 856, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1988.

[26] K. Hornik, M. Stinchcombe & H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vo. 2, pp. 359–366, 1989.

[27] G. C. Goodwin & K. S. Sin, *Adaptive Filtering , Prediction and Control*, Prentice-Hall, Inc., 1984.

[28] W. Rudin, *Principles of Mathematical Analysis*, 3rd Edition, McGraw-Hill, 1976.

[29] G. Kreisselmeier & B. Anderson, "Robust Model Reference Adaptive Control", *IEEE Transaction on Automatic Control*, Vol. AC-31, No. 2, February 1986, pp. 127–133.

[30] V. D. Tourasis & C. P. Neuman, "Robust Nonlinear Feedback Control for Robotic Manipulators", *IEE Proc.*, Vol. 132, pt. D, p.p. 134–143, 1985.

[31] R. E. Kalman & J. E. Bertram, "Control System Analysis and Design Via the "Second Method" of Lyapunov: Discrete-Time Systems", *ASME Journal of Basic Engineering*, p.p. 394–400, 1960.

[32] F. M. A. Salam, "Artificial Neural Nets: Basic Theory and Engineering Implementations", Department of Electrical Engineering, Michigan State University, October 1989.

[33] A. J. Owen and D. L. Filkin, "Efficient Training of the Back Propagation Network by Solving a System of Stiff Ordinary Differential Equations", *Int'l Joint Conf. on Neural Networks*, p.p. 381–386.