DEVELOPMENT OF DYNAMIC NEURAL STRUCTURES WITH CONTROL APPLICATIONS

A Thesis

Submitted to the College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

in the

Department of Electrical Engineering

and

Intelligent Systems Research Laboratory

University of Saskatchewan

by

Hulikunta Rao Dandina

(D.H. Rao)

Saskatoon, Saskatchewan, Canada

March, 1994

Copyright

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to Dr. M.M. Gupta for his valuable guidance throughout the course of this work. His encouragement and positive criticism have been mainly responsible for the success of this project. He has spent an enormous amount of time having many enlightening discussions with me on the project as well as guiding me in the preparation of this thesis and other materials for publication. His advice and help are very much appreciated.

I would like to extend my sincere appreciation and thanks to Dr. P.N. Nikiforuk and Dr. H.C. Wood for their advice and assistance in the preparation of this thesis.

Financial assistance provided by Natural Sciences and Engineering Research Council (NSERC) and a doctoral scholarship from the University of Saskatchewan is gratefully acknowledged. I would like to express my appreciation to Mr. D. Bitner, Mr. I.J. MacPhedran, Mr. K.D. Jeffery and Mr. J. E. Moore for their help throughout my work. I would also like to express my sincere thanks to the advisory committee members and the library staff for their assistance.

I would like to take this opportunity to express my thanks to my wife Alaka Rao for her patience, encouragement and moral support throughout this endeavor. Thanks are also due to the Principal and the executive members of Gogte Institute of Technology, Belgaum, India for granting me the study leave to pursue my higher studies at University of Saskatchewan.

Hulikunta Rao Dandina

UNIVERSITY OF SASKATCHEWAN

Electrical Engineering Abstract 93A388

Development of Dynamic Neural Structures with Control Applications

Student: Hulikunta Rao Dandina

Supervisors: Dr. M.M. Gupta
Dr. H.C. Wood
Dr. P.N. Nikiforuk

Ph.D. Thesis Submitted to the

College of Graduate Studies and Research

March, 1994

ABSTRACT

Dynamic neural networks, because they offer computational advantages over purely static neural networks, have many potential applications in a number of fields. The objective of the research described in this thesis was to develop dynamic neural structures for control applications. A dynamic model of the biological neuron called the *dynamic neural unit* (DNU) was developed for this purpose. The structure of the DNU is inspired by the topology of a reverberating circuit in a neuronal pool of the central nervous system. The DNU consists of internal feedforward and feedback synaptic weights followed by a nonlinear activation operator. It is thus different from the conventionally assumed structure of an artificial neuron. It is demonstrated in this thesis that a DNU can control unknown linear and simple nonlinear systems to track adaptively desired trajectories.

The efficacy of artificial neural networks comes more from the number of neurons connected in the network and from the topology rather than from the computational ability of an isolated neuron. Considering the DNU as the basic functional element, a multi-stage dynamic neural network has been developed. One of the most important characteristics of neural networks is their ability to approximate arbitrary nonlinear functions. While most of the

research work in this area has concentrated on static neural networks, a theoretical foundation of functional approximation using a dynamic neural network has been developed. Computer simulation studies are provided to substantiate the theoretical developments. Following this development, the dynamic neural network has been used in a direct adaptive control mode to cause unknown nonlinear systems to follow desired reference signals. In conventional static neural structures, the optimum slope of a nonlinear activation function is usually determined by trial and error. An improper selection of the slope may lead to instability. The importance of using an adaptive activation operator in neural networks has been demonstrated through computer simulations. In this context, the concept of *somatic adaptation* for dynamic neural structures has been introduced. The significance of this concept as applied to the control of unknown nonlinear dynamic systems has been extensively studied through computer simulations.

A new dynamic neural structure called the *dynamic neural processor* (DNP) that emphasizes the aggregate dynamic properties of a neural population has also been proposed and reported in this thesis. This structure is based upon the hypothesis that the neuro-physiological activities of any complexity are dependent upon the interaction of antagonistic (excitatory and inhibitory) neural subpopulations. The DNP consists of two DNUs which are configured to function as excitatory and inhibitory neurons. A mathematical model and an algorithm to modify the parameters of the DNP have been developed. Four applications of the DNP, the functional approximation of nonlinear functions, computation of inverse kinematic transformations of a two-link robot, control of unknown single-input-single-output nonlinear systems, and coordination and control of multiple-input-multiple-output systems, are presented. A brief comparative study of the performance of this neural model with that of conventionally used recurrent neural networks has also been presented. A generalized dynamic neural model based on the concept of neural subpopulations has been proposed in this thesis. It is shown that many existing neural structures can be derived from this generalized neural model.

TABLE OF CONTENTS

LIST OF FIGURES

(Abbreviated captions)

LIST OF TABLES

Chapter 6

LIST OF SYMBOLS

Symbol	Meaning
Chapter 1	
k	Discrete time index
$e(k)$	Error signal
$\mathbf{W}(k)$	Vector of synaptic weights
$\mathbf{X}(k)$	Vector of input signals
x_0	Initial state
$\mathbf{Y}(k)$	Vector of output signals
$y_d(k)$	Desired output signal
z^{-1}	Unit delay operator
$\Psi[.]$	Nonlinear activation function
θ	Neural threshold
$\Re$	Set of real numbers
Chapter 2	
$\mathbf{a}_{ff}$	Vector of feedforward weights of the DNU
$\mathbf{a}_{ff_0}$	Initial values of $\mathbf{a}_{ff}$
$\mathbf{b}_{fb}$	Vector of feedback weights of the DNU
$\mathbf{b}_{fb_0}$	Initial values of $\mathbf{b}_{fb}$
E	Expectation operator
exp	Exponential
g_s	Gain of the nonlinear (sigmoidal) function
$J(.)$	Performance index
m	Number of inflection points
$s(k)$	Input signal
$u(k)$	Output signal of the DNU
$\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$	Vector of adaptable weights of the DNU
$\mathbf{P}_{\Phi}(k)$	Parameter-state signals of Φ
$\mathbf{P}_{ff}(k)$	Parameter-state signals of the feedforward weights
$\mathbf{P}_{fb}(k)$	Parameter-state signals of the feedback weights
$\Gamma(k, v, s)$	Vector of feedforward and feedback signals of the DNU
$\sigma[.]$	Distribution of neural thresholds
$\nabla_{\Phi} J(.)$	Gradient of the performance index with respect to Φ

$\Psi'[.]$	Derivative of the function $\Psi[.]$
μ	Adaptive gain (learning factor)
dia[μ]	Diagonal matrix of adaptive gains

Chapter 3

C[e(k), Δe(k)]	Linear function of the error and the change of error
f[.]	Nonlinear function
$f^{-1}[.]$	Inverse of the function f[.]
$G_p[.]$	Plant dynamics
$G_p^{-1}[.]$	Inverse dynamics of $G_p[.]$
$\hat{G}_p(.)$	Estimated plant transfer function
k_p	Proportional gain
k_d	Differential gain
q(k)	Vector of state variables
$u_c(k)$	Output of the PD controller
$u_{nn}(k)$	Output of the DNU controller
y(k)	Plant output
β_{ff}	Feedforward parameters of the plant
α_{fb}	Feedback parameters of the plant
ε_{tol}	Tolerance limit

Chapter 4

D	Compact set
d(Ψ, f)	Distance between the functions $\Psi[.]$ and f[.]
$\hat{f}[.]$	Approximation of the function f[.]
$\hat{g}[.]$	Approximation of the function g[.]
H	Number of stages in the dynamic neural network
sup	Supremum

Chapter 5

g_{s_0}	Initial value of the gain of the sigmoidal function
μ_{g_s}	Adaptive gain of the gain of the sigmoidal function

Chapter 6

C2	cos (θ_2)
I	Vector of interconnecting strengths between systems

I_{12}	Interconnecting strength from system 1 to system 2
I_{21}	Interconnecting strength from system 2 to system 1
L	Length of the truck
L_1	Length of link 1
L_2	Length of link 2
n_E	Number of excitatory neurons
n_I	Number of inhibitory neurons
$s_E(k)$	Input to an excitatory unit
$s_I(k)$	Input to an inhibitory unit
S2	$\sin(\theta_2)$
$u_E(k)$	Output of an excitatory unit
$u_I(k)$	Output of an inhibitory unit
x_d	Desired x coordinate of the end-effector
y_d	Desired y coordinate of the end-effector
λ	Excitatory E, or inhibitory I, state of the DNP
$s_\lambda(k)$	Input to a neural subpopulation
$s_{t\lambda}(k)$	Total inputs to the neural units
$u_\lambda(k)$	Output of a neural subpopulation
$w_{\lambda\lambda}$	Strength of self-synaptic connections
w_{EE}	Self-feedback weight of the excitatory neuron
w_{EI}	Feedback weight from the excitatory to the inhibitory neuron
$w_{\lambda\lambda'}$	Strength of the cross synaptic or inter-subpopulations
w_{II}	Self-feedback weight of the inhibitory neuron
w_{IE}	Feedback weight from the inhibitory to the excitatory neuron
$\Psi_E[.]$	Nonlinear function of the excitatory neuron
θ_E	Threshold of the excitatory neuron
θ	Steering angle
ϕ	Angle of the truck with the ground
$\Psi_I[.]$	Nonlinear function of the inhibitory neuron
θ_I	Threshold of the inhibitory neuron
(x_i, ϕ_i)	Initial position of the truck
(x_f, ϕ_f)	Final position of the truck
A, B	Matrices of feedback weights in the generalized model
C	Matrix of self- and inter- neuron feedback weights
F, D	Scaling matrices in the generalized model

G, H, P	Matrices of feedforward weights in the generalized model

Appendix II

w^A, w^B	Weights of the elements of the matrices A and B
μ_A, μ_B	Adaptive gains of the elements of the matrices A and B
w^C	Weights of the elements of the matrix C
μ_C	Adaptive gain of the elements of the matrix C
w^G, w^H, w^P	Weights of the elements of the matrices G, H and P
μ_G, μ_H, μ_P	Adaptive gains of the elements of the matrices G, H and P

1. Introduction

1.1 Neuro-Biological Control: A Motivation

Biological control mechanisms are quite successful in dealing with uncertainty and complexity. They can smoothly coordinate many degrees of freedom during the execution of manipulative tasks within unstructured environments. Biological systems are usually very complex and defy exact mathematical formulations of their operation [1]. They carry out complex tasks without having to develop conventional mathematical models of the task or the environment. In executing a particular control task, for example *'pick up a glass of water'*, the plan to execute the task is carried out at the conscious level. To pick up a glass of water, it is necessary to determine the position of the hand relative to the glass and to find a way to move the hand toward the glass. The biological system executes this high level task at the conscious level. Most of the low level actions, such as determination of joint angles and muscle coordination, are performed at the subconscious level. The biological control system can learn to perform a new task, and can adapt to the changing environment with ease.

On the other hand, to make a robot arm perform the same task, *'pick up a glass of water'*, requires a large number of computations and *a priori* knowledge of the environment and the system. These computations, required to coordinate different robot joints to produce a desired trajectory, are performed by solving kinematic and dynamic relationships between the different structural members of the robot itself. The control methodology developed for this task may fail should the desired task or the environment change.

If the fundamental principles of neuro-biological control systems are understood, it may be possible to develop an entirely new generation of control methodologies. These control systems would be more robust and intelligent, far beyond the capabilities of the traditional control techniques based upon mathematical modeling. If system engineers could learn the structural, functional and behavioral aspects of biological control mechanisms, it might be possible to design an intelligent controller which can emulate at least some of the biological control capabilities. Although many biologists and psychologists share the view that the brain has a modular architecture, there is no general agreement on the number of modules, or the manner in which the modules develop or are interconnected [2]. One reason for this diversity of opinion is that the modular nature of the brain involves the difficulty of reasoning about a system with a large number of interacting components. Even systems of interacting components with a small fraction of the brain's complexity present formidable

computational and analytical difficulties. It is scientifically challenging to understand the control functions of biological neural systems and to use this knowledge to emulate some of these functions for the purpose of solving scientific and engineering problems. Based on the understanding of neuro-biological control aspects and the desire to develop simple models of neuronal structures for engineering applications, the field of *artificial neural networks* has evolved into a very promising area of research [3 - 6]. Artificial neural networks have been used in a variety of applications such as pattern recognition, system identification and control.

1.2 Neural Networks in Control Systems

The conventional design of an automatic control system often involves the construction of a mathematical model describing the dynamic behavior of the system to be controlled and the application of analytical techniques to this model to derive a control law. Usually, such a mathematical model consists of a set of linear or nonlinear differential or difference equations most of which are derived using some form of approximation and simplification. The traditional model-based control techniques break down, however, when a representative model is difficult to obtain due to uncertainty or sheer complexity, or when the model produced violates the underlying assumptions of the control law synthesis techniques [1, 7]. Modeling of a physical system for feedback control also involves a trade-off between the simplicity of the model and its accuracy in matching the behavior of the physical system. On the other hand, human operators do not always handle control problems with detailed mathematical models. Instead, they use imprecise and qualitative understanding of the controlled processes.

In conventional methods, two approaches are usually described in the literature [8] to achieve satisfactory performance from a dynamic plant that is only partially known. One approach uses robust stabilizers, or robust controllers [9, 10] and the other approach uses adaptive controllers. Using the first approach, if the actual physical system is contained in a class of systems which are close to the nominal plant, a robust controller by definition is guaranteed to stabilize it. Since one fixed controller is expected to stabilize a large class of control systems, the controller thus designed is highly complex compared to the complexity required to stabilize any single plant. Using the second approach, the parameters of the adaptive controller are made to adapt in accordance with some algorithm in order to keep the system performance at a desired level. In general, the adaptive approach is applicable to a wider range of uncertainties, but robust controllers are simpler to implement, and it may not

be necessary to tune the controller parameters of robust controllers to match the plant variations [11]. Detailed descriptions of robust and adaptive control techniques may be found in [10 - 13] and [14 - 16] respectively.

While adaptive control has shown potential for controlling complex systems and offers good disturbance rejection, the region of operation of such a control system is restricted. This is because the adjustable parameters of the adaptive controller are modified based on the convergence and stability conditions, and this may place severe limitations on the performance of the compensated system. These limitations can be seen as restrictions on the acceptable operating region of the controller [17, 18]. Perhaps the most unrealistic among the conditions are the assumptions that there are no disturbances and that the order of the plant is not higher than that of the model. Violation of even a few of these assumptions can cause the adaptive control algorithms to become unstable [8]. In many situations, it may be desirable to design control schemes that can exhibit both learning and adaptive capabilities.

To cope with uncertainties regarding plant dynamics, a controller needs to estimate unknown information during operation. If this estimated information gradually approaches the true information as time proceeds, then the controller can approach an optimal controller, and the controller may be viewed as a *learning controller* [19]. The controller learns the unknown information during operation, and this information is used as experience for future decisions and control, thereby possibly improving the system performance. The use of neural networks in control systems can be viewed as a natural step in the evolution of control methodologies. Neural networks with their massive parallelism and their ability to learn offer good possibilities for improved techniques in control systems.

Computational Neural Networks (CNNs), Artificial Neural Networks (ANNs) or simply neural networks, are described as connectionist models, parallel distributed processing units, or neuromorphic systems [20]. All of these representations are constructed with many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets. They have been shown to perform, on a small scale, such higher cognitive functions as learning, memory and recall, and pattern recognition. Indeed, several working examples in the fields of speech recognition [21] and visual pattern recognition have been described [22].

Although the field of artificial neural networks is not new, it has only recently become an active area of research. Some of the pioneering work in this field is due to

McCulloch and Pitts [23] who in 1943 published a simple abstract model of a neuron. This neuron had a finite number of inputs and a single output. The inputs were characterized by excitatory (+1) and inhibitory (-1) states, the neuron had an internal threshold and the nonlinear function was binary. It was thought that by connecting many of these simple devices it would be possible to model the human brain. Although this model proved inadequate to achieve human-like abilities, it did influence others to pursue research in the field of neural networks.

The next major development occurred in 1949 when Hebb [24] conjectured a learning mechanism in the brain. He postulated that as the brain learns, it changes its connectivity patterns. This idea of a learning mechanism was first incorporated in an artificial neural network by Rosenblatt [25] in 1959. He combined the simple McCulloch and Pitts neuron, with the adjustable synaptic weights based on the Hebbian learning scheme, to form the first artificial neural network with the capability to learn.

By introducing the least mean squares (LMS) learning algorithm, Widrow and Hoff [26] developed in 1960 a model of a neuron that learned quickly and accurately. This model was called ADALINE for ADAptive LInear NEuron. This learning algorithm first introduced the concept of supervised learning using a 'teacher' which guides the learning process. It is the recent generalization of this learning rule into the back propagation algorithm that has led to the resurgence in biologically-based neural network research today.

In 1969 research in the field of neural networks suffered a serious setback. Minsky and Papert [27] published a book entitled *Perceptrons*, in which they proved that single layer neural networks were limited in their abilities to process data and argued that the study of multi layer neural networks would be unproductive. As a result of this influential book, little progress was made in this area until the early 1980s.

Many of the early applications of neural networks have been in computationally intensive areas of signal processing, adaptive pattern recognition, real-time speech recognition, and image interpretation. There are also computationally intensive applications in control systems, such as real-time system identification and control of nonlinear systems. With specific reference to control system design, neural networks have shown great potential in the realm of nonlinear control problems. A neuro-controller (neural network-based control system), in general, performs a specific form of adaptive control, with the controller taking the form of a multi layered network and the adaptable parameters being defined as the adjustable weights. In general, neural networks represent parallel distributed processing

structures, which make them prime candidates for use in multi-variable control systems. The neural network approach defines the problem of control as the mapping of measured signals for 'change' into calculated control signals for 'actions'. The most significant characteristic of neural networks is their ability to approximate arbitrary nonlinear functions to any degree of accuracy [28]. This ability of neural networks has made them useful in the modeling of nonlinear systems which is of primary importance in the synthesis of nonlinear controllers. Furthermore, because neural networks exhibit learning features, it is not necessary to know *a priori* the dynamics of the plant under control. The general features of neural networks can be summarized as follows [28, 29]:

(i) Neural network models have many neurons (the computational units) linked via adaptive weights arranged in massive parallel structures;

(ii) Because of high parallelism, the failure of a few neurons does not necessarily significantly affect the overall system performance. This characteristic is also called fault-tolerance;

(iii) The main strength of the neural network structures lies in their learning and adaptive abilities. The ability to adapt and learn from the environment means that neural network models can deal with imprecise data and ill-defined situations; and

(iv) Neural network models can be used for the identification and control of non-linear dynamic systems.

The computational process that implements neural networks starts with the development of an 'artificial' neuron based on an understanding of biological neuronal structures, followed by the definition of structures and learning mechanisms for a given set of applications. This leads to the following three steps in a neural computational process:

(i) Development of neural models based on the understanding of biological neurons,

(ii) Development of models of synaptic connections and structures (that is, network topology), and

(iii) Specification of learning rules (that is, the method of adjusting the weights or inter-nodal connection strengths).

In this context, a simple model of a biological neuron has been proposed in the literature [25, 26]. This structure of an artificial (computational) neuron receives its inputs

either from other neurons or from sensors. A weighted sum of these inputs constitutes the argument of a 'fixed' nonlinear activation function as shown in Fig. 1.1. The weights correspond to the strength of the synapses while the activation function is associated with the electrical conduction mechanism in a biological neuron. The resulting value of the activation function is the axonal or neural output [20]. This neural output is transmitted to several other neurons.

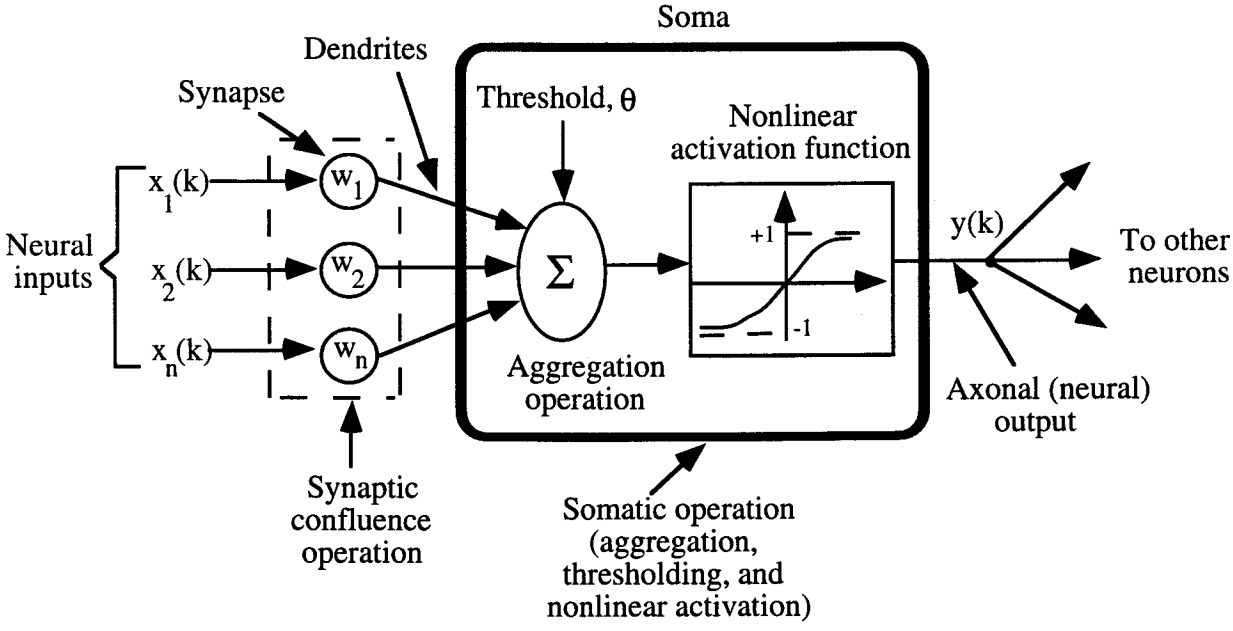


Figure 1.1: A static neural model of a biological neuron.

In Fig. 1.1, the vector $[x_1(k),..., x_n(k)]$ represents the neural inputs, $[w_1,..., w_n]$ represents the synaptic weights, k denotes the discrete time index, θ represents the threshold and y(k) the axonal (neural) output. The detailed descriptions of synaptic confluence and somatic operations may be found in [6]. Using the static neural model shown in Fig. 1.1, a number of neural structures, usually referred to as feedforward neural networks, have been reported in the literature [20, 28 - 32]. These feedforward networks respond instantaneously to inputs because they possess no dynamic elements in their structure. Therefore, feedforward neural network structures are also called static neural networks. A static neural network, in general, consists of a number of neural layers (stages) where the output of one neuron forms an input to other neurons in the next layer. This neural structure is often referred to as a multi layered neural network (MNN). A typical MNN consists of an input layer, hidden layers and an output layer. A three layered static neural network is shown in Fig. 1.2. In this figure, each shaded circle represents the static neuron shown in Fig. 1.1.

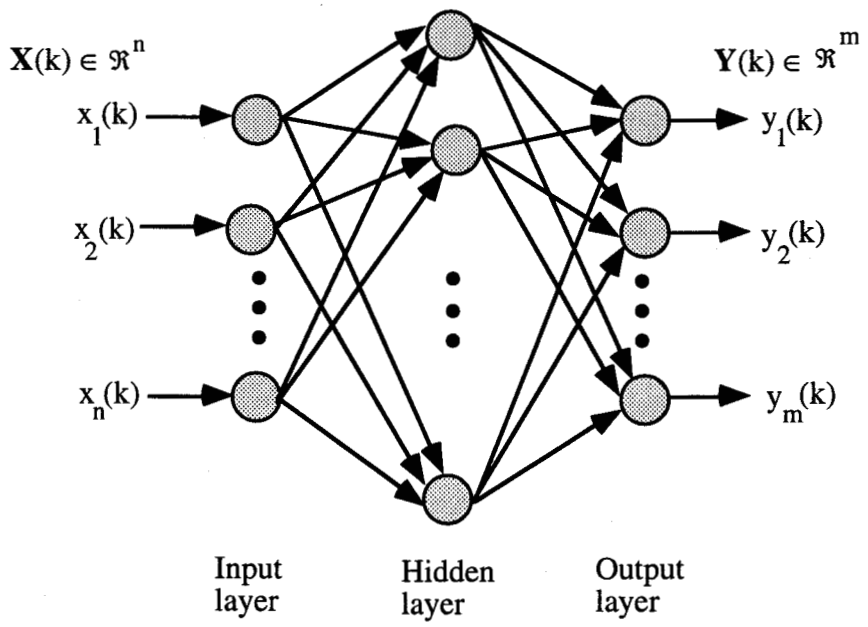


Figure 1.2: A three layered static neural network.

As an extension of static neural networks, dynamic (feedback) neural networks using static neurons with feedback have been proposed in the literature [22, 33, 34]. A general topology of a dynamic neural structure, commonly employed for system identification and control applications, with a static neuron as the basic functional unit is shown in Fig. 1.3. The feedforward inputs may arise from a source outside the neural layer or from other neurons, whereas the feedback signals are a result of dense lateral, self-excitatory (+) and self-inhibitory (-) connections in the layer as shown arbitrarily in Fig. 1.3.

Recurrent [33] and time-delay neural networks [34] fall in the category of dynamic neural architectures. The recurrent, or feedback, neural networks were first introduced by Hopfield [33] as a dynamic model of the biological neural structure. The recurrent structure consists of a single layer static network in a feedback configuration with a time delay as shown in Fig. 1.4.

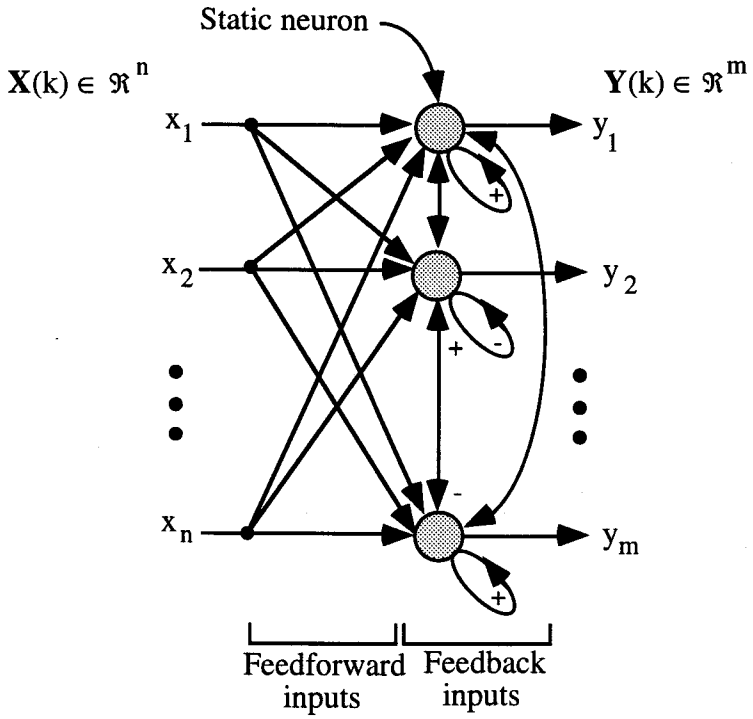


Figure 1.3: A generalized topology of a dynamic neural network with extensive feedforward and feedback inputs.

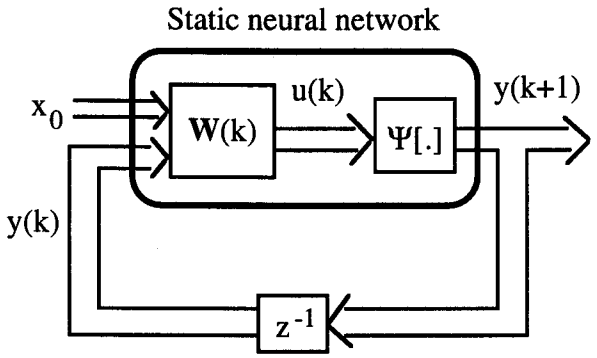


Figure 1.4: The state-space model of the Hopfield neural structure, also known as a recurrent neural network.

In Fig. 1.4, y(k) and y(k+1) represent the states of the neural network at instants k and k+1, x_0 represents the initial value, **W**(k) denotes the vector of the neural weights, Ψ[.] is the nonlinear activation function, and z^{-1} represents the unit delay operator. Given an initial value x_0, the dynamic system evolves to an equilibrium state if Ψ[.] is suitably chosen. The set of initial conditions in the neighborhood of x_0 which converge to the same equilibrium

state are then identified with that state. The term "associative memory" is often used to describe such systems. These feedback networks with or without constant inputs are merely nonlinear dynamic systems and the asymptotic behavior of such systems depends upon the initial conditions, the specific inputs and the particular nonlinear function [34]. A detailed explanation of recurrent neural networks is given in Chapter 6.

Another dynamic structure, called the time-delay (tapped-delay) neural network (TDNN), is basically a feedforward network with delay elements. This is equivalent to a finite impulse response (FIR) filter whose output forms an argument to a nonlinear activation function as shown in Fig. 1.5. The TDNN can function as an adaptive filter by computing the scalar product of the input vector $\mathbf{X}(k)$ and the synaptic weight vector $\mathbf{W}(k)$, and modifying the elements of the synaptic weight vector $\mathbf{W}(k)$ with a technique such as least-mean square (LMS) learning algorithm [26, 20, 28, 31]. This neural unit is adapted (trained) using noise-contaminated samples for which the correct uncontaminated signal values, $y_d(k)$, are known. In other words, the desired neural output is known for each input sample (supervised learning).

Although the recurrent neural network incorporates feedback with delay elements, and the time-delay neural network (TDNN) employs dynamic elements in the forward path, the basic architecture of the computing neuron is a static model. These neural structures have been used in many applications, such as system identification and the control of nonlinear dynamic systems [34], and for text-to-speech conversion [35] respectively.

1.3 Thesis Objectives

The static neural model of the neuron described in the preceding sections is a very simplified, but useful first approximation, of the biological neuron [30]. This model ignores many of the characteristics of its biological counterpart. For example, it does not take into account time delays that affect the system dynamics; that is, the inputs produce an instantaneous output with no memory involved [31]. Biological neurons continually integrate, on the average, up to 10,000 synaptic inputs, which do not add up in a simple linear manner. Each neuron is a sophisticated computing element, and it performs much more complex operations than simple summation [36]. The conventional neural network models have abstracted a few properties of biological neurons, such as weighted aggregation, nonlinear activation and parallelism [37]. However, it is essential to gain more insight into how a single biological neuron functions, how masses of neurons are structured, and how

they coordinate themselves to perform complex tasks. It is then necessary to incorporate the essential functions and features of biological neurons into neural models.

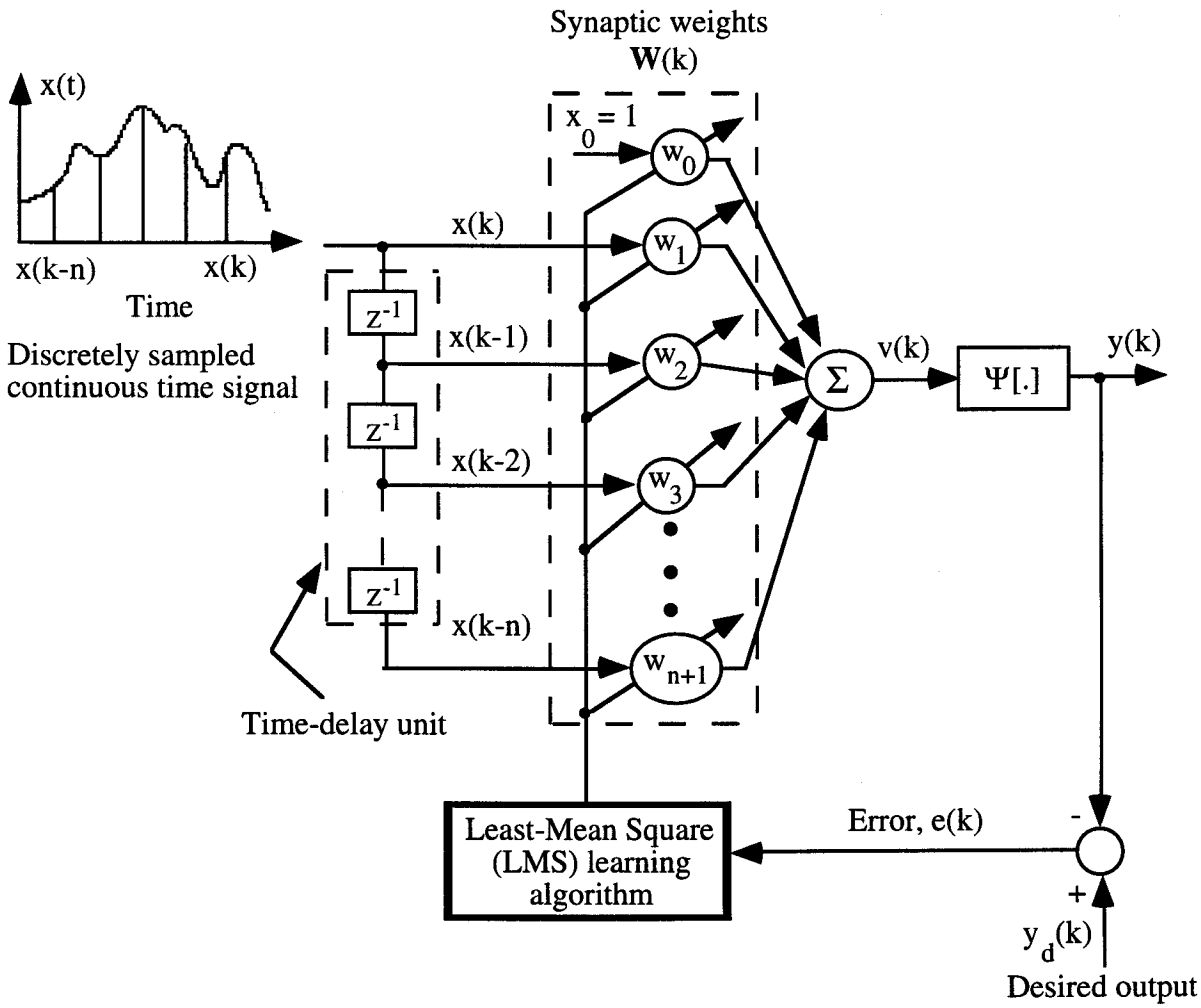


Figure 1.5: A time-delay neural unit based on a static neural architecture.

In this thesis, some of the concerns mentioned above have been addressed. At least a few limitations of the traditional neural networks based on the static neural model may be alleviated by restructuring the architecture of an artificial neuron, and deducing the necessary learning algorithms. As a first step towards this goal, the general objective of the work reported in this thesis has been to develop a model of a neuron which can more faithfully reflect the dynamics of a biological neuron, and to develop neural network structures and learning schemes for robotics and control applications.

Specifically, the objectives of the research that is described in this thesis were:

(i) To develop a dynamic model of a neuron, and dynamic neural network structures using this neural model. This development will follow the observed features of a biological neural structure.

(ii) To develop a dynamic artificial neural network structure based on suggested dynamic properties of a neural population or neural mass.

(iii) To study the effectiveness of the proposed dynamic neural structures, through computer simulation studies, for functional approximation, for control of linear and nonlinear dynamic systems, and for computation of inverse kinematic transformations of a two-link robot. A brief comparison of dynamic neural network-based control schemes with proportional and derivative (PD) controller and model-reference adaptive controller (MRAC) will be studied. The performance of dynamic neural structures developed in this thesis will also be compared with recurrent neural networks.

1.4 Organization of the Thesis

In the following chapters, both the mathematical foundation of the proposed neural structures and their potential for learning and control applications are presented. The structure of the proposed neuron, called the *dynamic neural unit* (DNU), is developed in Chapter 2. The mathematical modeling and the implementation scheme of the DNU are also detailed in this chapter.

The effectiveness of the DNU, as applied to the control of unknown linear systems, is demonstrated through computer simulation studies in Chapter 3. In this chapter, a control technique called the *inverse dynamic adaptive control* (IDAC) using the DNU is described. The IDAC technique is based on the concept of adaptive inverse control in which the controller structure is made to be an approximate inverse-model of the plant under control.

A multi-stage dynamic neural network is developed in Chapter 4 considering the DNU as the basic computing element, and the network is implemented to control nonlinear dynamic systems. The theoretical development and computer simulation studies of the functional approximation of the proposed dynamic neural network are also presented in this chapter.

The modified structure of the DNU, which accounts for both synaptic and somatic adaptations, is developed in Chapter 5. Accordingly, the modifications in the learning algorithm and its implementation are also discussed in this chapter. Using the modified DNU, a three-stage dynamic neural network is developed and used to make unknown nonlinear dynamic systems adaptively track desired trajectories. A comparative study of neural networks with and without somatic adaptation is also briefly discussed.

Based on the physiological evidence that neural activities of any complexity are dependent upon the interactions of antagonistic neural subpopulations, namely excitatory and inhibitory neurons, another neural structure proposed in this thesis, called the *dynamic neural processor* (DNP), is discussed in Chapter 6. The mathematical development and the algorithm to modify the self- and inter-subpopulation synaptic connections are discussed. Several applications of the DNP, namely the functional approximation, computation of the inverse kinematic transformations of a two-link robot, control of unknown single-input-single-output nonlinear systems, and coordination and control of multiple systems, are detailed in this chapter. A brief comparative study of recurrent neural networks and the DNP is also discussed. As an extension of the DNP model, a generalized dynamic neural model is proposed in this chapter.

Finally, the concluding remarks, the major contributions of the thesis, and suggested directions for future research are presented in Chapter 7. The significant contributions of the thesis are as follows: (i) development of a dynamic model called dynamic neural unit (DNU) and its associated dynamic neural structures, (ii) development of the theory of functional approximation for dynamic neural networks, and (iii) development of a dynamic neural processor based on the concept of excitatory and inhibitory neural subpopulations. It is demonstrated, through computer simulations, that the neural structures developed in this thesis performed better compared to the conventional control techniques and recurrent neural networks for several control problems. The parameter-state signals for the feedforward and the feedback weights of the modified DNU structure, proposed in Chapter 5, are derived in Appendix I. The learning algorithm for the generalized dynamic neural model, proposed in Chapter 6, is derived in Appendix II.

2. Dynamic Neural Unit

2.1 Introduction

In its simplest form, a computational neuron can be considered as a processing element that sums the weighted inputs and produces an output only if this sum exceeds an internal threshold. This neuronal model has no feedback connections; that is, there are no connections through the weights extending from the outputs of a layer to the inputs of the same or to the previous layers. Furthermore, this model has no memory. The neural output is solely determined by the current inputs and values of the synaptic weights. Neural network structures based on this model describe the synaptic connections by a single weight parameter vector. In a feedforward structure, this results in a static neural network. Biological neural systems are, generally, understood to be composed of structures with dynamic connections which are manifested in the temporal properties of the synapse along with such processes as impulse transmission and membrane excitation [22, 30, 33]. In order to emulate some of the dynamic functions, such as learning, adaptation, memory and recall, and to better reflect the dynamics of the biological neuron, it is useful to model the biological neuron using feedback networks. In this thesis, one such model called the *dynamic neural unit* (DNU) [38, 39] is proposed. The DNU consists of internal feedforward and feedback weights and a nonlinear activation function, and is thus different from the conventionally assumed structure of an artificial neuron.

This chapter is organized as follows. The architectural details of the DNU are presented in Section 2.2. An algorithm to modify the adjustable parameters of the DNU is then derived in Section 2.3. The implementation scheme for the developed algorithm is also presented in this section. Finally, the concluding remarks of this chapter are mentioned in the last section.

2.2 Architectural Details of Dynamic Neural Unit

The central nervous system (CNS) is divided into many different anatomic parts, in each of which are located accumulations of neurons called neuronal pools [40]. One of the most important circuits in the neuronal pool is the reverberating circuit, which functions as follows: an incoming signal stimulates the first neuron, which then stimulates the second, the third and so forth. However, branches return to the first neuron providing feedback and re-stimulate it as depicted in Fig. 2.1. The reverberating circuit is the basis of innumerable CNS activities, for it allows a single input signal to elicit a response lasting a few seconds,

minutes, or hours. Almost all rhythmic muscular activities, including the rhythmic movements of walking, are mainly controlled by the reverberating circuits [40].

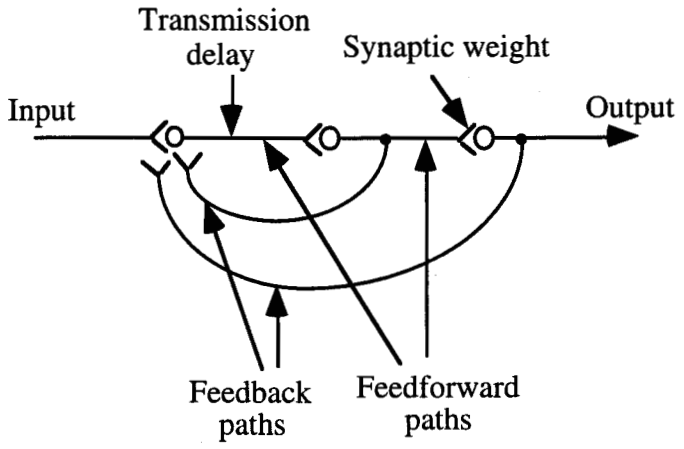


Figure 2.1: A reverberating circuit in a neuronal pool of the central nervous system (CNS).

Based on the topology of the reverberating circuit, a new architecture of the neuron called the *dynamic neural unit* (DNU), shown in Fig. 2.2, is proposed in this thesis. The dynamic structure of the DNU is assumed to be of second-order and is analogous to a reverberating circuit. The output of this dynamic structure becomes an argument to a nonlinear activation function. The DNU does not represent any specific anatomical region within the biological system. The delay elements in the DNU account for the synaptic delay in a biological neural structure. The occurrence of synaptic delay may be explained as follows [40]. In the transmission of an action potential from a neuron, a certain period of time is consumed in the processes of (a) discharge of the transmitter substance by the pre-synaptic neuron, (b) diffusion of the transmitter to the neuronal membrane, (c) action of the transmitter on the membrane, and (d) inward diffusion of sodium ions to raise the potential to a high enough value to elicit an action potential. The minimum time required for all these events to take place is approximately 0.5 millisecond. This important characteristic of the biological neuron has been ignored in the conventional structure of an artificial neuron.

The neural dynamics of the DNU can be expressed in the form of a second-order transfer relation

$$w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) = \frac{v(k)}{s(k)} = \frac{[a_0 + a_1 z^{-1} + a_2 z^{-2}]}{[1 + b_1 z^{-1} + b_2 z^{-2}]} \qquad (2.1)$$

where $s(k) = \left[\sum_{i=1}^{n} w_i s_i - \theta \right]$ is the neural input to the DNU, $s_i \in \Re^n$ are the inputs from other neurons or from sensors, $w_i \in \Re^n$ are the corresponding input weights, θ is an internal threshold, $v(k) \in \Re^1$ is the output of the dynamic structure (neural dynamics), $u(k) \in \Re^1$ is the neural output, $\mathbf{a}_{ff} = [a_0, a_1, a_2]^T$ and $\mathbf{b}_{fb} = [b_1, b_2]^T$ are the vectors of adaptable feedforward and feedback weights respectively, z^{-1} is the unit delay operator, and k is the discrete-time index.

Alternatively, Eqn. (2.1) may be described by the following difference equation

$$v(k) = - b_1 v(k-1) - b_2 v(k-2) + a_0 s(k) + a_1 s(k-1) + a_2 s(k-2) . \qquad (2.2)$$

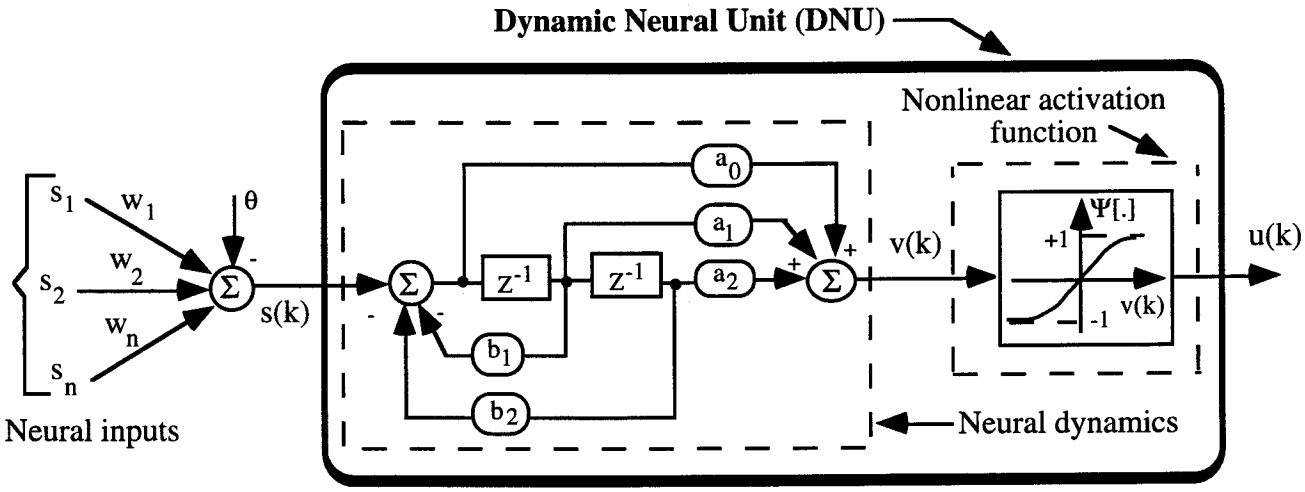


Figure 2.2: The basic structure of the DNU consisting of second-order dynamics followed by a nonlinear activation function.

The vectors of signals and adaptable weights of the DNU are defined as

$$\Gamma(k, v, s) = [v(k-1) \;\; v(k-2) \;\; s(k) \;\; s(k-1) \;\; s(k-2)]^T , \text{ and} \qquad (2.3)$$

$$\Phi_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})} = [-b_1 \;\; -b_2 \;\; a_0 \;\; a_1 \;\; a_2]^T \qquad (2.4)$$

where the superscript T denotes transpose. Using (2.3) and (2.4), Eqn. (2.2) can be rewritten as

$$v(k) = \Phi^{T}_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})}\ \Gamma(k, v, s) = \begin{bmatrix} -b_1 & -b_2 & a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} v(k-1) \\ v(k-2) \\ s(k) \\ s(k-1) \\ s(k-2) \end{bmatrix} . \qquad (2.5)$$

The nonlinear mapping operation on v(k) yields a neural output u(k) given by

$$u(k) = \Psi[\, v(k)] \qquad (2.6)$$

where $\Psi[.]$ is a nonlinear activation function. Many different forms of mathematical functions can be used as a nonlinear activation function [6]. However, the selection of this nonlinear function depends upon the following assumption.

Assumption : The neural system is comprised of only one 'type' of neuron.

This assumption enables the probability distribution of the neural thresholds about an aggregate value θ to be defined as a unimodal function, as shown in Fig. 2.3a. It follows then that the nonlinear transformation is sigmoidal as shown in Fig. 2.3b. However, if the neural unit is assumed to be comprised of 'm' different types of neurons, then the distribution of the thresholds can be redefined in m-modal functions that produce a nonlinear input transformation with m-inflection points [41, 42].

The proportion of neurons in a neural network that receive inputs greater than the threshold value can be modeled by a nonlinear transformation function, $\Psi[v(k)]$, which is related to the distribution of neural thresholds, $\sigma[v(k)]$, within the neural unit [41]. If the probability distribution of these neural thresholds about an aggregate value θ is given by an unimodal distribution function, then the nonlinear input transformation (activation operator) may be represented by a sigmoidal function. Thus, the proportion of neurons in a neural network receiving inputs greater than the intrinsic threshold may be modeled by the expression

$$u(k) = \Psi[v(k), g_s, \theta] = \int_{-\infty}^{v(k)} \sigma[v(k)]\, dv(k) \qquad (2.7)$$

where the pair $[g_s, \theta]$ determines the transformational properties of the function $\Psi[.]$ [41, 42]. The parameter g_s is defined as the maximum slope of the sigmoidal relationship at the point of inflection given by the aggregate value θ. In other words, for a particular distribution of

neural thresholds, it is possible to determine the proportion of neurons receiving inputs exceeding the threshold by integrating the neural threshold distribution over the total applied inputs, Eqn. (2.7).

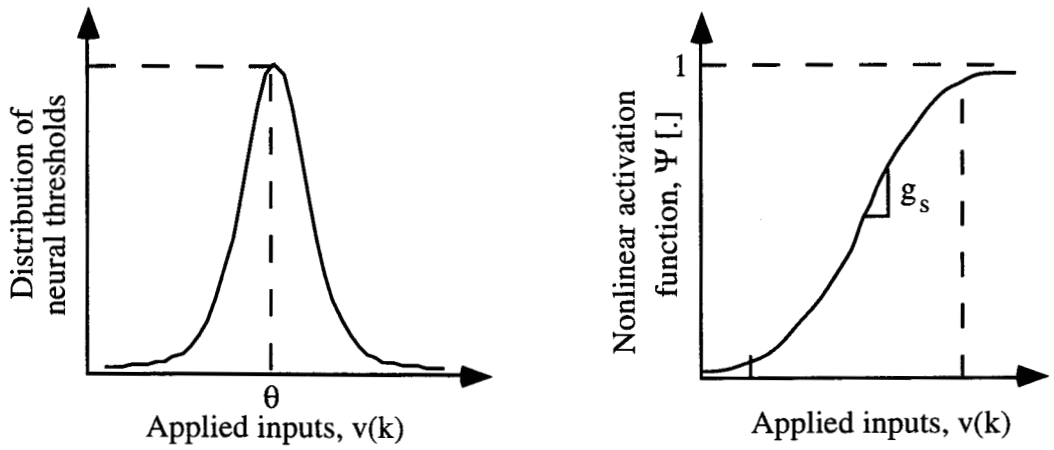


Figure 2.3: Unimodal threshold distribution and the corresponding sigmoidal input transformation function.
(a) An unimodal probability distribution function of neural thresholds θ,
(b) The nonlinear sigmoidal function, with slope g_s, arising from the neural threshold distribution given in (a).

An important assumption in deriving this sigmoidal function is that each neural unit in a densely connected neural network is comprised of only one 'type' of neuron. This enables the distribution of neural thresholds to be defined as a unimodal function. However, if the network is assumed to be comprised of m different types of neurons then the distribution of thresholds must be redefined as an m-modal function that produces a nonlinear activation function with m inflection points as depicted in Fig. 2.4 for m = 3.

In general, a m-modal probability distribution for the neural thresholds is expressed as

$$\sigma[(v(k)] = \frac{1}{2m} \sum_{i=1}^{m} g_{s_i} \operatorname{sech}^2\left[g_{s_i}\left(v(k) - \theta_i\right)\right] \tag{2.8}$$

and the corresponding monotonically increasing input transformation function is given by

$$\Psi[(v(k)] = \frac{1}{2}\left[1 + \frac{1}{m} \sum_{i=1}^{m} \tanh\left[g_{s_i}\left(v(k) - \theta_i\right)\right]\right] \tag{2.9}$$

where for each mode there is a slope parameter g_{s_i} and a corresponding inflection point θ_i.

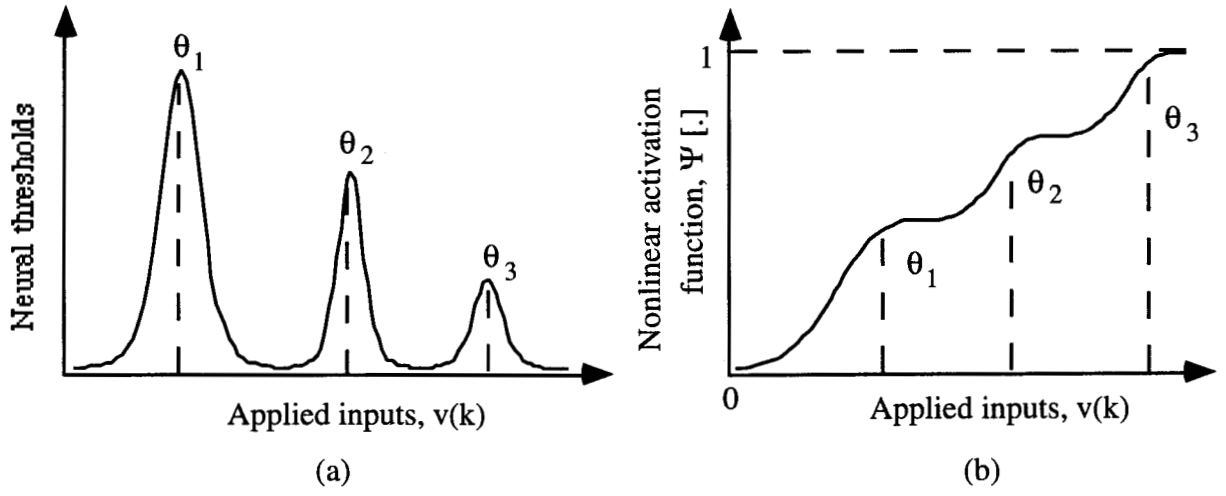


Figure 2.4: Multimodal threshold distribution and the corresponding sigmoidal input transformation function.
(a) Multimodal distribution of neural thresholds for m = 3,
(b) Nonlinear mapping operator with multiple inflection points that correspond to (a).

Physiologically, a multimodal distribution would be expected to correspond to the presence of a number of distinct cell types within the population of neurons [41, 43, 44, 45]. In further discussions, it is assumed that an artificial neural network is comprised of one type of neuron, and the corresponding distribution of thresholds is defined to be a unimodal function. Any function $\Psi[.]$ is said to belong to the class of sigmoidal functions, if

(a) $\Psi[v(k)]$ is a monotonically increasing function of v(k) in the interval $(-\infty, \infty)$. It follows that $\Psi[v(k)]$ is strictly increasing; that is if $v_1 < v_2$ for each v_1 and $v_2 \in \Re$, then it is true that

$$\Psi[v_1] < \Psi[v_2] \text{ and} \tag{2.10a}$$

$$\left|\Psi[v_1] - \Psi[v_2]\right| \leq C\,|v_1 - v_2|, \quad \forall\, v_1, v_2 \in \Re \tag{2.10b}$$

where C is a constant. Then $\Psi[v(k)]$ is said to satisfy a Lipschitz condition,

(b) $\Psi[v(k)]$ approaches or attains asymptotic values, say -1 and 1, as v(k) approaches $-\infty$ and ∞ respectively, and

(c) $\Psi[v(k)]$ has one and only one inflection point [41].

To extend the neural activity for both the excitatory (positive) and inhibitory (negative) inputs, the sigmoidal function can be redefined as a bipolar hyperbolic tangent function; that is,

$$\Psi[v(k)] = \frac{\exp\left(g_s\, v(k)\right) - \exp\left(-g_s\, v(k)\right)}{\exp\left(g_s\, v(k)\right) + \exp\left(-g_s\, v(k)\right)} = \tanh\left[g_s\, v(k)\right] \qquad (2.11)$$

where g_s is the gain which controls the slope of the activation function and is assumed to be constant in the following discussions. The effect of varying this parameter on the system performance is discussed in Chapter 5. In the limit, as $g_s \rightarrow \infty$, the sigmoidal function tends to become the sign (binary) function with an infinite slope at $v(k) = 0$, and a zero slope for $v(k) \neq 0$ as shown in Fig. 2.5.

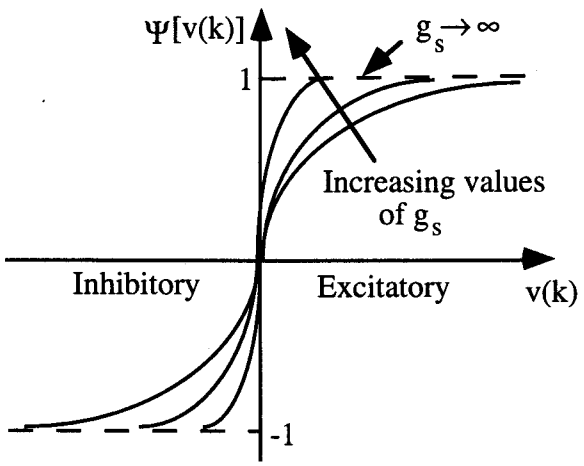


Figure 2.5: Sigmoid function $\Psi[v(k)] = \tanh\left[g_s\, v(k)\right]$ for excitatory and inhibitory signals.

In summary, the neural mathematical mapping of the DNU can be defined in a generalized form as depicted in Fig. 2.6. As shown in this figure, the first computation provides a dynamic linear mapping from $s(k) \in \Re^1$ to $v(k) \in \Re^1$ through the weighting vector $\Phi_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})} = \left[-b_1 \;\; -b_2 \;\; a_0 \;\; a_1 \;\; a_2\right]^T$. The second operation provides a nonlinear mapping from $v(k) \in \Re^1$ to $u(k) \in \Re^1$ through a nonlinear function $\Psi[.]$, which in this case is a hyperbolic tangent function.

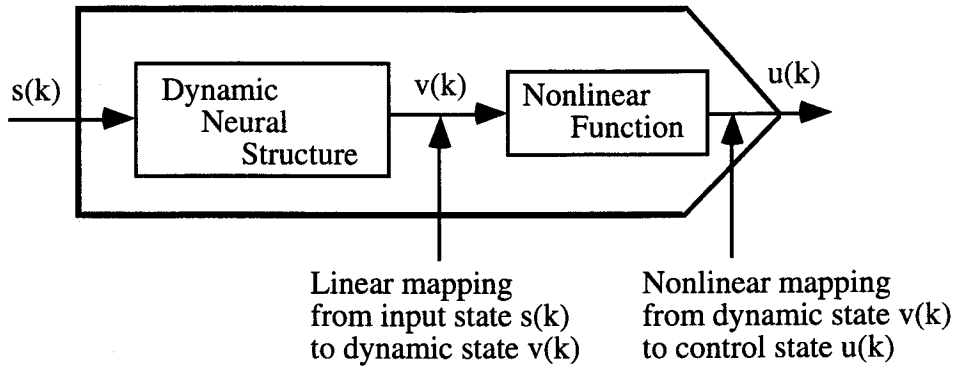


Figure 2.6: Neural mathematical operations of the DNU in a generalized form.

2.3 Learning and Adaptive Algorithm

Learning and adaptation are the terms used to describe the behavior modification in natural organisms as well as in machines. Biologists and mathematical psychologists have been primarily concerned with the modeling questions associated with these phenomena while the system theorists have addressed the problem of synthesizing machines which exhibit these properties [46, 47].

The function of the learning and adaptive algorithm involves the determination of feedforward and feedback synaptic weights which minimize the error function in some optimal fashion. The equivalency of the input and output is a convenient condition to test for the learning process [19, 47]. In an iterative learning scheme, the control sequence is modified in each learning iteration to cause the neural output u(k) to approach the desired state $y_d(k)$. If the error, e(k), can be reduced to an infinitesimally small value as the number of learning iterations increases, the learning scheme is said to be convergent [47]; that is,

$$u(k) \rightarrow y_d(k) \text{ as } k \rightarrow \infty \quad \text{or,}$$

$$\lim_{k \rightarrow \infty} \left[y_d(k) - u(k) = e(k) \right] \rightarrow 0 \tag{2.12}$$

for arbitrary initial conditions of the components of the weighting vector $\Phi_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})}$.

If neural networks are used in both pattern recognition (static identification) and system identification and control, the objective of the algorithm is to adjust the parameters of the network based on a given set of input-output pairs. If the parameters of the DNU, namely

the feedforward and feedback synaptic weights, are considered as the elements of a parameter vector $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$, the learning process involves the determination of optimal parameter vector represented as $\Phi^{*}_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$ that minimizes a performance index $J(\Phi)$ based on the output error. The components of the weighting vector $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$ and error e(k) vary with every learning trial k. To obtain $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k+1)$ requires only the information set $\{$ e(k-m), e(k), $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k)\}$, where m = 1,2, ... which determines the size of the window. As the number of learning trials increases, the information set reduces to only $\{e^{*}(k), \Phi^{*}_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k)\}$ which indicates that the error and neural weights have converged to the optimal values which satisfy the input-output equivalency condition. However, this may not guarantee the global optimization [34].

In this iterative process, the control sequence is modified in each learning iteration to cause the neural output u(k) to approach the desired state $y_d(k)$. To achieve this, a performance index which has to be optimized with respect to the weighting vector is defined as

$$J(\Phi) = E\left\{ F\left[e(k; \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})})\right] \right\} \tag{2.13}$$

where E is the expectation operator. A commonly used form of $F[e(k; \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})})]$ in Eqn. (2.13) is a squared function of the error; that is,

$$J(\Phi) = \frac{1}{2} E\left\{ e^{2}(k; \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}) \right\}. \tag{2.14}$$

Each component of the weighting vector $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$ is adapted in such a way so as to minimize $J(\Phi)$ based on the steepest-descent algorithm based on the following equation:

$$\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k+1) = \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k) - \text{dia}[\mu]\, \nabla_{\Phi} J(\Phi) \tag{2.15}$$

where dia[μ] is a diagonal matrix of the adaptive gains, $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k+1)$ and $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k)$ are the values of the parameter vector at the (k+1)-th and k-th instants respectively, and $\nabla_{\Phi} J(\Phi)$ is the gradient of the performance function J evaluated at $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k)$ and is written as $\dfrac{\partial J(\Phi)}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}$. In Eqn. (2.15), dia[$\mu$], the matrix of adaptive gains, is given by

$$\text{dia}[\mu] = \begin{bmatrix} \mu_{a_i} & 0 \\ 0 & \mu_{b_j} \end{bmatrix} \tag{2.16}$$

where μ_{a_i}, i = 0,1,2, and μ_{b_j}, j = 1,2, are the individual gains of the adaptable parameters of the DNU. The values of μ_{a_i} and μ_{b_j} are the measures of the strength of the adaptation of the DNU parameters which determine the stability and the speed of convergence. These issues are discussed more in detail in the next chapter.

From the definitions of the performance index, $J(\Phi)$, and error signal e(k), the gradient of the performance index with respect to the weighting vector is obtained as follows:

$$\frac{\partial J(\Phi)}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}} = \frac{1}{2} E\left[\frac{\partial [y_d(k) - u(k)]^2}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right]$$

$$= E\left\{ e(k)\left[- \frac{\partial u(k)}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right]\right\} = E\left\{e(k)\left[- \frac{\partial \Psi[v]}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right]\right\}$$

$$= E\left\{-e(k)\left[\frac{\partial \Psi[v]}{\partial v}\frac{\partial v}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right]\right\} = E\left\{-e(k)\left[\Psi'[v]\frac{\partial v}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right]\right\}$$

$$= E\left\{-e(k)\left[\frac{4}{[\exp(v(k)) + \exp(-v(k))]^2}\right]\frac{\partial v}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\right\}$$

$$= E\left\{ -e(k)\left[\text{sech}^2[v(k)]\ \mathbf{P}_{\Phi}(k)\right]\right\} \tag{2.17}$$

where $\mathbf{P}_{\Phi}(k) = \dfrac{\partial v(k)}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}$ is defined as a vector of parameter-state (or sensitivity) signals. These signals represent the direct impact of the parameter vector through the system equation on the DNU response. From Eqn. (2.2), the parameter-state vector is written as

$$\mathbf{P}_{\Phi}(k) = \frac{\partial}{\partial \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}}\left[- b_1\, v(k-1) - b_2\, v(k-2) + a_0\, s(k) + a_1\, s(k-1) + a_2\, s(k-2)\right]. \tag{2.18}$$

Equation (2.18) can be rewritten for feedforward and feedback weights of the DNU as

$$\mathbf{P}_{ff_i}(k) = \frac{\partial}{\partial a_{ff_i}(k)} \left[- b_1 v(k-1) - b_2 v(k-2) + a_0 s(k) + a_1 s(k-1) + a_2 s(k-2) \right], \quad (2.19a)$$

$$i = 0, 1, 2, \text{ and}$$

$$\mathbf{P}_{fb_j}(k) = \frac{\partial}{\partial b_{fb_j}(k)} \left[- b_1 v(k-1) - b_2 v(k-2) + a_0 s(k) + a_1 s(k-1) + a_2 s(k-2) \right], \quad (2.19b)$$

$$j = 1, 2.$$

The partial derivatives on the right-hand side of Eqns. (2.19a) and (2.19b) arise because the DNU structure has feedback connections, whereby previous output samples depend on previous parameter values which, in turn, are related to the present parameter values via successive updates of the algorithm in Eqn. (2.15). However, if the values of μ_{a_i} and μ_{b_j} are chosen sufficiently small, then the approximation in Eqn. (2.3), that is $\Gamma(k, v, s) \approx \Gamma((k-1), (v-1), (s-1)) \approx \Gamma((k-2), (v-2), (s-2))$, is valid [48]. This is a reasonable assumption for many applications, and the performance degradation due to this assumption is insignificant in practice. Based on this assumption, the parameter-state signals for the components of the weighting vector, namely the feedforward and the feedback weights, are obtained as follows:

(i) For feedforward weights, $\mathbf{a}_{ff_i}$, i = 0, 1, 2:

From Eqn. (2.19a) the parameter-state (or sensitivity) signals can be written as

$$\mathbf{P}_{ff_i}(k) = \frac{\partial}{\partial a_{ff_i}(k)} \left[a_0 s(k) + a_1 s(k-1) + a_2 s(k-2) \right], \; i = 0, 1, 2. \quad (2.20a)$$

The individual parameter-state signals for the feedforward weights may be written as

For i = 0, $\mathbf{P}_{a_0}(k) = [s(k)]$,

For i = 1, $\mathbf{P}_{a_1}(k) = [s(k-1)]$, and

For i = 2, $\mathbf{P}_{a_2}(k) = [s(k-2)]$.

Therefore, the parameter-state signals for the feedforward weights are

$$\mathbf{P}_{ff_i}(k) = [s(k-i)], \quad i = 0, 1, 2. \quad (2.20b)$$

(ii) For feedback weights, $\mathbf{b}_{fb_j}$, j = 1, 2:

Similarly, from Eqn. (2.19b) the parameter-state signals for feedback weights can be written as

$$\mathbf{P}_{fb_j}(k) = \frac{\partial}{\partial b_{fb_j}(k)} \left[- b_1 v(k-1) - b_2 v(k-2)\right], j = 1, 2. \qquad (2.21a)$$

The individual parameter-state signals for the feedback weights are

For j = 1, $\mathbf{P}_{b_1}(k) = - [v(k - 1)]$, and

For j = 2, $\mathbf{P}_{b_2}(k) = - [v(k - 2)]$.

Therefore, the parameter state signals for the feedback weights are

$$\mathbf{P}_{fb_j}(k) = - [v(k - j)], \quad j = 1,2. \qquad (2.21b)$$

As seen from Eqns. (2.20) and (2.21), the parameter-state signals for the feedforward weights may be obtained by tapping the node signals from the controller structure, while the generation of the parameter-state signals for the feedback weights manifests itself as an additional structure with only the feedback weights as shown in Fig. 2.7.

From Eqns. (2.15) and (2.17) the parameter vector $\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}$ is updated based on the following algorithm:

$$\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k+1) = \Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}(k) + \text{dia}[\mu]\, E\left\{ e(k)\, \text{sech}^2[v(k)]\, \mathbf{P}_{\Phi}(k) \right\}. \qquad (2.22)$$

From Eqns. (2.20) , (2.21) and (2.22) the following equations to modify the feedforward and feedback weights may be written as

$$a_{ff_i}(k+1) = a_{ff_i}(k) + \mu_{a_i} E\left\{ e(k)\, \text{sech}^2[v(k)]\, \mathbf{P}_{ff_i}(k) \right\}, \; i = 0,1,2, \qquad (2.23a)$$

and

$$b_{fb_j}(k+1) = b_{fb_j}(k) + \mu_{b_j} E\left\{ e(k)\, \text{sech}^2[v(k)]\, \mathbf{P}_{fb_j}(k) \right\}, \; j = 1,2. \qquad (2.23b)$$

The implementation scheme of Eqns. (2.23a) and (2.23b) is shown in Fig. 2.8, and the symbolic representation of the DNU in Fig. 2.9. In Fig. 2.8, the terms δa_{ff_i} and δb_{fb_j} are respectively the adaptive components of the feedforward and feedback weights of the DNU. Neural dynamics represent the second-order structure represented by Eqn. (2.1). An

additional structure with only feedback weights is necessary to compute the parameter-state signals for the weights $\mathbf{b}_{fb}$.

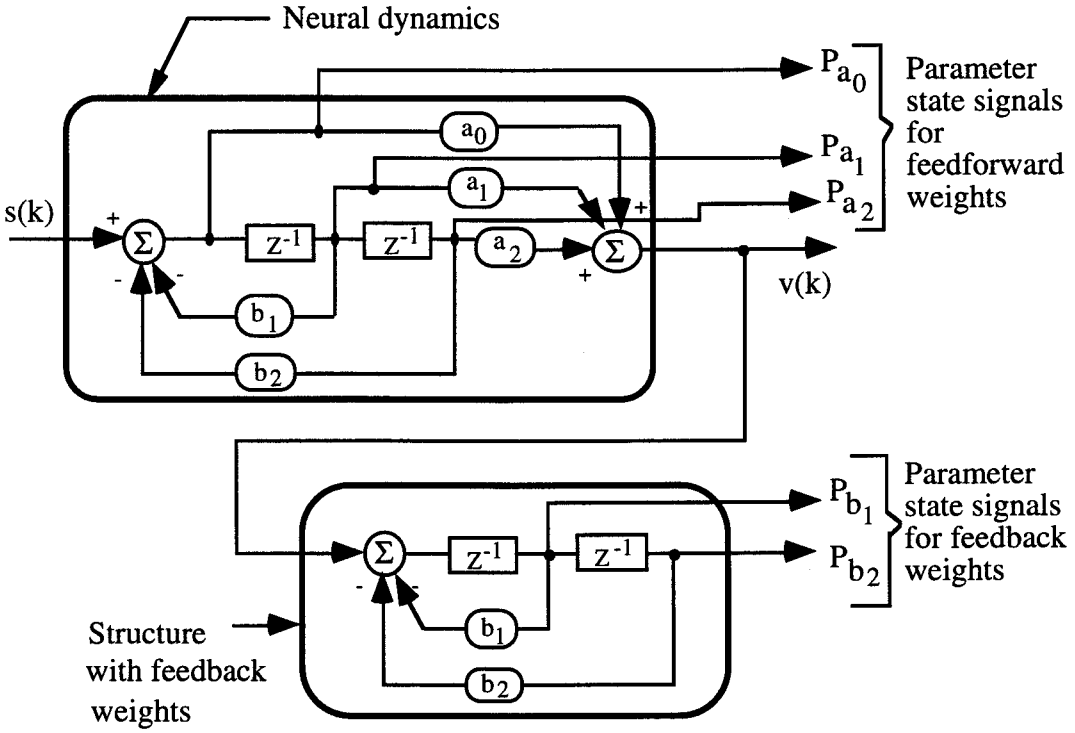


Figure 2.7: Obtaining the parameter-state signals for the dynamic neural unit.

In Fig. 2.9, w(k, $\mathbf{a}_{ff}$, $\mathbf{b}_{fb}$) represents neural dynamics of the DNU, $\mathbf{a}_{ff}$ and $\mathbf{b}_{fb}$ are the adaptable feedforward and feedback weights with the corresponding parameter-state signals $\mathbf{P}_{ff_i}(k)$ and $\mathbf{P}_{fb_j}(k)$ respectively, $\mathbf{a}_{ff_0}$ and $\mathbf{b}_{fb_0}$ represent the initial values of the adjustable weights, and $\Psi[.]$ represents the nonlinear activation function.

The following observations are made with reference to the algorithm derived above for the parameters of the DNU:

(i) The desired model M_D, in Fig. 2.8, is an entity representing a physical reality (in the case of system identification, for example), or model, in the designer's mind (in the case of pattern classification problems, for example).

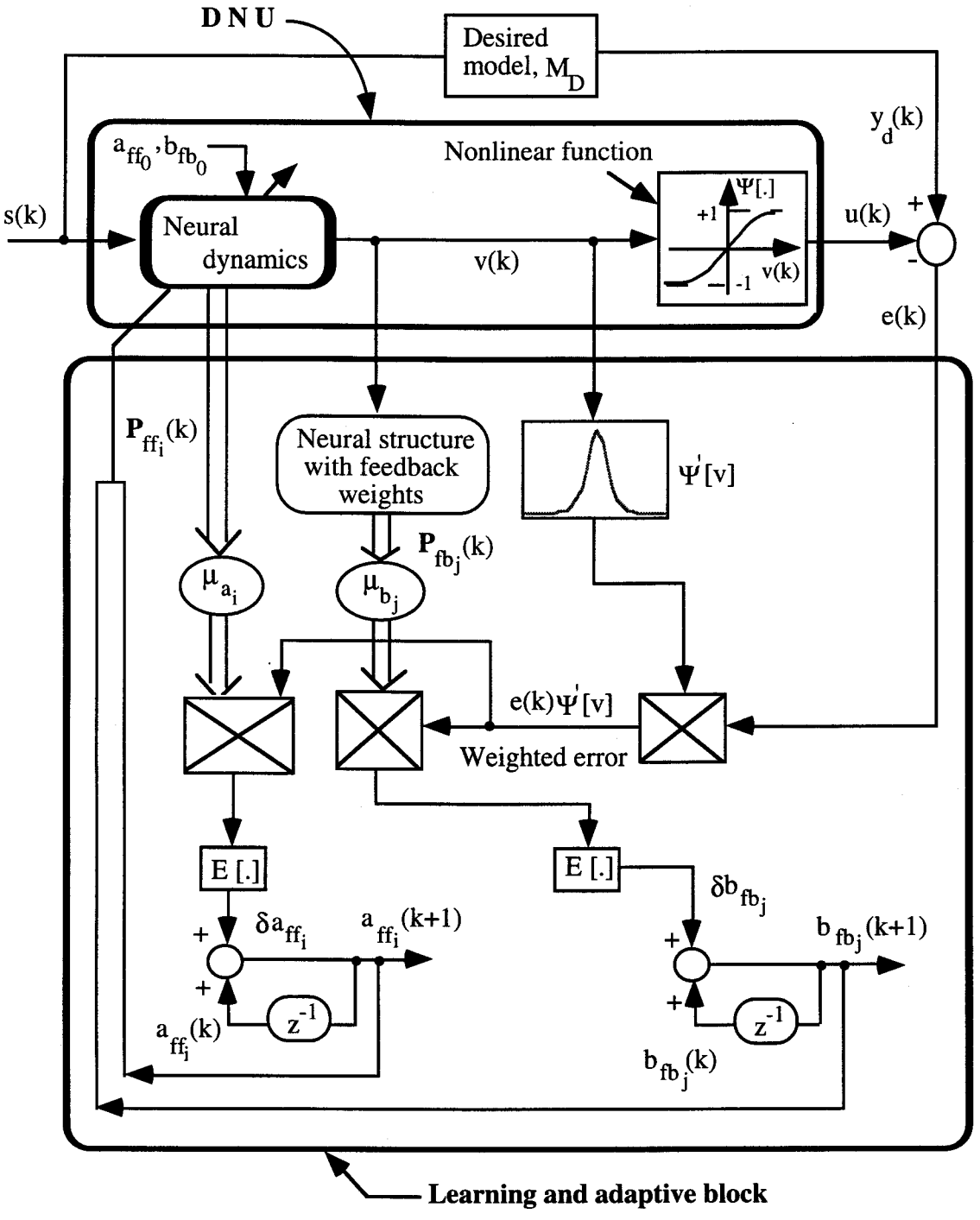


Figure 2.8: The implementation scheme of the learning and adaptive algorithm.

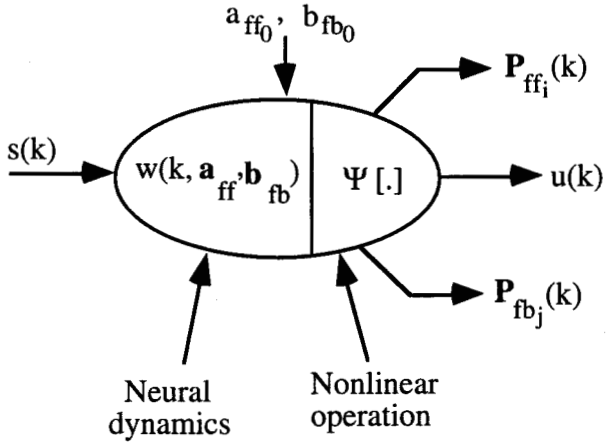


Figure 2.9: Symbolic representation of the dynamic neural unit (DNU).

(ii) The expectation of a random process x with the probability density function p(x) is defined as $E[x] = \int_{-\infty}^{\infty} x\, p(x)\, dx$. Thus, E[x] is an averaging process and can be approximated by the temporal (time) average $E[x(t)] = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} x(t)\, dt$. For a discrete case, $E[x(k)] = \frac{1}{K} \sum_{i=0}^{T} x(i)$. Due to the computational difficulty of an ensemble average, the gradient of a single time sample of the squared error $e^2(k)$ can be used to obtain an estimate of the gradient of the error function $J(\Phi)$ in Eqn. (2.14) [48]. The expectation operator in Eqns. (2.23a) and (2.23b) is then replaced by instantaneous values of the partial derivatives.

(iii) Equations (2.20) and (2.21) ignore interdependence of the parameter set; as such, they represent a greatly simplified version of the true gradients that would result in the presence of parameter dependence.

(iv) Equations (2.23a, and 2.23b) make use of the inner product of the error signal, e(k), and the derivative of the nonlinear function, $\Psi'[v]$. This term is significant in the sense that during the learning process, if |v| is small in the neighborhood of zero, then it provides a large weight to the error, thus making a large change in the weighting vector $\Phi_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})}$. On the other hand, if |v| is large, $\Psi'[v]$ is small, thus providing very little weight to the error.

2.4 Summary

A new architecture of a computational neuron called the dynamic neural unit (DNU) has been presented. The DNU is comprised of feedforward and feedback internal weights, delay and a nonlinear activation operator. The dynamic structure of the DNU was only analogous to that of a reverberating circuit in a neuronal pool of the central nervous system. The output of this neural dynamics formed an argument to a nonlinear activation function. An algorithm for updating the feedforward and feedback synaptic weights of the DNU and an implementation scheme for the proposed algorithm have also been presented. Due to its dynamic nature, the DNU can be trained to learn and control unknown dynamic systems. The application of DNU to linear control problems is discussed in the next chapter.

3. Inverse Dynamic Adaptive Control of Linear Systems Using Dynamic Neural Unit

3.1 Introduction

It has been demonstrated by many researchers [49 - 58] that an unknown plant (system) will track, within physical limitations, an input command signal if the plant is preceded by a controller which approximates the inverse of the plant's transfer function. Precascading a plant with its inverse model provides an unity mapping between the input and output signal space within the limitations of gain, power, etc. This concept of inverse modeling has been referred to as adaptive inverse control [51]. Adaptive inverse control using a finite impulse response (FIR) structure has been used to control a non-minimum phase system [51]. The concept of inverse-modeling has been utilized in reducing the intersymbol interference in digital communication systems [53]. In this application, the inverse-modeling of channel dynamics makes the received signal match the transmitted signal. Filtering the received signal through an approximation of the inverse of the channel model has been suggested as the principal remedy [53, 59]. Practical problems that may cause time-varying channel dynamics dictate the use of adaptive algorithms for tuning the channel equalizers. Equalization in data modems combats this distortion by filtering incoming signals. A modem's adaptive filter, by adjusting itself to become a channel inverse, can compensate for the irregularities in the channel magnitude and phase response [49].

In this chapter, a control technique called the **I**nverse **D**ynamic **A**daptive **C**ontrol (IDAC) using the DNU is described. The IDAC technique is based on the concept of adaptive inverse control in which the controller structure is made to be an approximate inverse-model of the plant under control. A brief introduction to IDAC scheme is given in the next section. The principle of IDAC and the rephrasing of the global performance are also described in this section. Computer simulation studies of the IDAC scheme are presented in Section 3.3. A feedback-error learning scheme using the DNU is discussed in Section 3.4, followed by a summary in the last section.

3.2 Inverse Dynamic Adaptive Control

Adaptive inverse control is based on the idea of inverse modeling. In this scheme, the inverse model of a plant is estimated and cascaded with the plant, making the overall transfer function of the plant and the inverse model unity. The adaptive inverse modeling and inverse control scheme are shown in Figs. 3.1a and 3.1b respectively.

If the inverse estimation is good, the error between the targeted and the observed outputs will be very small since the overall transfer function is almost unity. The major concern in the adaptive inverse control technique is to accurately obtain the inverse model of an unknown plant. The inverse model is, therefore, not expected to be the exact inverse of the plant but is intended to be a best fit of the reciprocal of the plant transfer function.

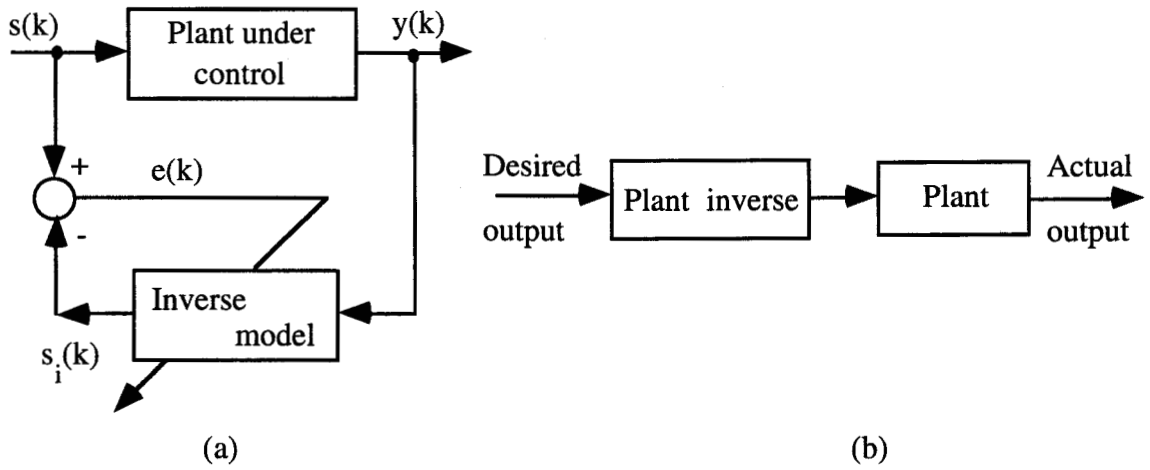


Figure 3.1: Adaptive inverse control scheme.
(a) Obtaining the inverse model (inverse-identification) of a plant, s(k), y(k) and e(k) represent input, output and error signals respectively, $s_i(k)$ represents output of the inverse model, and
(b) The control strategy following inverse-identification.

Traditionally, adaptive or self-tuning filters have been developed based on the FIR structures. The FIR filters have the advantage of a very well developed theory with regard to stability and convergence analysis. They have been generally used as they are unconditionally stable and because of the well understood adaptive FIR algorithms. However, the FIR realizations suffer from the problem of indeterminate order when it is necessary to model transfer function poles [28]. In particular, when the poles of the transfer

function are close to the unit circle in the z-plane, a high-order FIR filter may be required to meet a particular performance objective [59]. The adaptive inverse control technique proposed by Widrow [51, 52] provides only zeros to the controller. He suggested that for proper application, the plant must be stable and the plant zeros should not be extremely close to the $j\omega$–axis in the s-plane or the unit circle in the z-plane. Information about the upper bounds and the transport delay of the unknown plant is also required for the implementation of the adaptive inverse control. Furthermore, Widrow did not discuss the performance of the inverse-dynamic controller under structural perturbations. The adaptive inverse control scheme proposed by Widrow involves two modes of operation: (i) the learning phase that estimates an inverse model of the unknown plant, and (ii) the control phase that involves implementation of the inverse model to make the plant follow the desired trajectory. In other words, this scheme employs a 'learn-then-control' strategy. This strategy was also employed by Hunt and Sbarbaro in their internal model control scheme [54] .

The primary advantage of using an infinite impulse response (IIR) filter is that it can perform significantly better than an adaptive FIR filter for the same number of coefficients [48]. This is a consequence of the output feedback which generates an infinite impulse response with only a finite number of parameters. A desired response can be better approximated by a filter that has both poles and zeros (IIR filter) compared to one that has only zeros (FIR filter) [48]. Filters with feedback are particularly appropriate for system modeling (identification), control, and filtering applications.

Despite these advantages, the major obstacle to the use of IIR filters is the lack of well established and well understood adaptive algorithms. This is mainly due to the multimodal nature of their performance. The other major concern is to maintain stability during adaptation so that the poles of the filter do not accidentally move outside the unit circle causing instability. In general, the properties of an adaptive IIR filter are considerably more complex than those of an FIR filter, and it is more difficult to predict the behavior of the adaptive IIR algorithms [59, 60].

Most of the adaptive algorithms for the FIR and IIR filters reported in the literature are dominated by linear systems theory. There are many problems [29] which require nonlinear dynamics. Computational neural networks, which are inherently nonlinear, may be considered as a plausible alternative to the existing linear filters to overcome some of the limitations of the latter. The flexibility and learning capabilities of neural networks have made them applicable to a diverse set of nonlinear problems. The most significant characteristic of these networks, which is of primary importance from the view point of

control and communication systems, is their ability to approximate arbitrary nonlinear continuous functions. The application of inverse-modeling to robotic trajectory control using a feedforward neural network is discussed in [58].

In this thesis, a control scheme named the inverse dynamic adaptive control (IDAC) using the DNU is developed. The DNU, as described in the preceding section, is basically an IIR filter followed by a nonlinear activation function. The principle of the IDAC scheme and computer simulation studies are discussed in the following paragraphs.

3.2.1 The Principle

Consider a single-input-single-output (SISO) dynamic plant that has the following input-output relation

$$\begin{aligned} y(k+1) &= f(y(k),\ldots., y(k-n), u(k),\ldots., u(k-m)) \\ &= f(q(k), u(k)) \end{aligned} \tag{3.1}$$

where $q(k) = [y(k),\ldots., y(k-n), u(k-1),\ldots., u(k-m)]^T$ is a state vector, and f(.) is an unknown nonlinear function and satisfies $\partial f(q,u)/\partial u \neq 0$. In the IDAC scheme, shown in Fig. 3.2, the input-output equation of the DNU that produces the control signal to the plant is expressed as

$$u(k) = \Psi(w(.), q(k), s(k)) \tag{3.2}$$

where s(k) is the reference (desired) signal, and w(.) represents the dynamics of the DNU. Using the learning and adaptive algorithm derived in Section 2.3, the nonlinear mapping $\Psi[.]$ can be adapted to approximate the inverse function of the nonlinear system, that is

$$\Psi(w, q, s) \rightarrow f_u^{-1}(q, s) \tag{3.3}$$

where $f_u^{-1}(q, s)$ satisfies

$$y(k+1) = f(q(k), u(k)) = f(q(k), f_u^{-1}(q(k), s(k)) = s(k). \tag{3.4}$$

From Eqn. (3.4) it can be observed that an unknown dynamic plant can be made to track the desired trajectory by making the DNU mapping an inverse of that of the plant. This is the intended behavior of the scheme illustrated in Fig. 3.2, that is, the output y(k) follows the reference input s(k). Equation (3.3) places a constraint on the nonlinear function $\Psi[.]$ in that it's inverse should exist. Thus, the IDAC scheme uses an iterative constrained inverse technique to find the control inputs to the plant. That is, rather than training a controller

network and placing this network directly in the feedback or feedforward paths, the forward (inverse) model of the plant is learned, and iterative inversion is performed on line to generate control commands. This approach allows the controller to respond on line to changes in the plant dynamics, and avoids placing the highly nonlinear networks directly in the feedback control path [55].

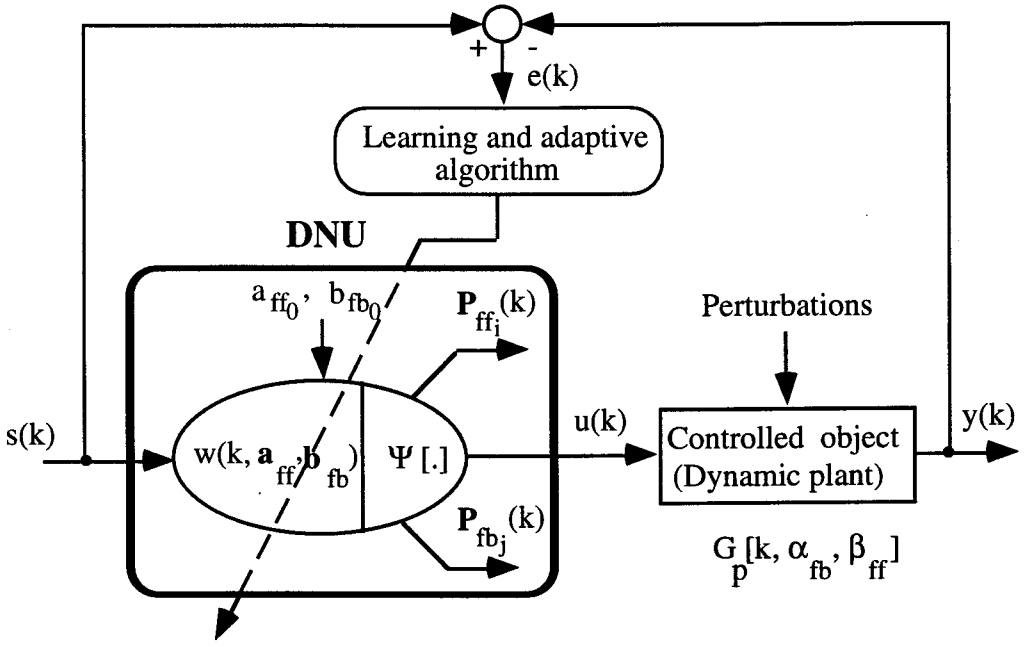


Figure 3.2: The inverse dynamic adaptive control (IDAC) scheme.

In this chapter, the dynamic plant represented by Eqn. (3.1) is assumed to have a linear relation between the input and output signal space. It then follows that an exact inverse model of the plant may be obtained by placing the controller poles on the plant zeros and the controller zeros on the plant poles. However, in practice it may be sufficient to match the numerator and denominator polynomials of the controller to those of the plant [51].

Let the plant dynamics and the controller (DNU) be described by transfer functions $G_p[k, \alpha_{fb}, \beta_{ff}]$ and $w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})$ respectively. The controller parameters are represented by $\mathbf{a}_{ff}$, $\mathbf{b}_{fb}$ and the plant parameters by vectors α_{fb}, β_{ff}. These may be written as:

$$\mathbf{a}_{ff} = [\, a_0, a_1, a_2]^T \quad , \quad \mathbf{b}_{fb} = [b_0, b_1, b_2]^T \quad : \text{for the controller, with } b_0 = 1$$

and

$$\beta_{ff} = [\beta_0, \beta_1, \beta_2]^T \, , \quad \alpha_{fb} = [\, \alpha_0, \alpha_1, \alpha_2]^T : \text{for the plant.}$$

The error signal is defined as the difference between the desired response s(k) and the actual response y(k). Mathematically, the error signal can be represented as

$$e(k) = s(k) - y(k) = s(k) - s(k)\, w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\, G_p[k, \alpha_{fb}, \beta_{ff}]$$

$$= s(k) \left[(1 - w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\, G_p[k, \alpha_{fb}, \beta_{ff}]) \right]. \qquad (3.5)$$

If, by using the learning and adaptive algorithm derived in the preceding chapter, the transfer relation of the controller is adapted to be an inverse of the plant under study so that

$$w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) = G_p^{-1}[k, \alpha_{fb}, \beta_{ff}] \qquad (3.6)$$

where $\mathbf{a}_{ff}$ is made equal to α_{fb} and $\mathbf{b}_{fb}$ equal to β_{ff}, Eqn. (3.5) then becomes

$$e(k) = s(k) \left[1 - G_p^{-1}[k, \alpha_{fb}, \beta_{ff}]\, G_p[k, \alpha_{fb}, \beta_{ff}] \right] = 0 . \qquad (3.7)$$

Equation (3.7) is valid for the linear operating region of the DNU. However, the DNU is a nonlinear computing element due to the presence of the nonlinear activation function $\Psi[.]$. The DNU can be operated in the linear region by changing the slope of the sigmoidal activation function. In other words, the saturated region of the sigmoidal function is controlled by its slope [61]. The sigmoidal function with a small slope makes the DNU operate with a linear mapping where Eqn. (3.7) is valid.

The IDAC scheme keeps track of the varying dynamics of the plant and adjusts the feedforward and feedback weights of the DNU in order to reduce errors between the input and output signals. The equivalency of input and output is a convenient condition to test to assure that the controller is the inverse model of the plant [53, 57]. The approximate dynamics of the plant under control may be obtained from the optimal feedforward and feedback weights. As the convergence of DNU weights depends on their initial settings and adaptive gains used in the algorithm, the optimal weights may not exactly represent the plant parameters even though the input-output equivalency condition is satisfied. This problem is discussed briefly in the next subsection.

3.2.2 Rephrasing the Global Performance

The control scheme shown in Fig. 3.2 monitors the error signal e(k) and adapts the weights of the controller in such a way that the performance index $J(\Phi)$ is kept minimum. At each trial, the changes in DNU weights are proportional to the error. This leads to a system

that settles to a stable weight configuration as the error becomes minimum. However, the changes become zero only for the zero gradient of the error in the weight space. This zero can represent either a true global minimum or only a local one, but from the practical point of view, this gradient descent algorithm generally results in useful, if not optimal, solutions [48, 59, 60]. The reason for the possible occurrence of local minima is explained below.

The error at the (k+1)th time moment, e(k+1), in terms of the error at the kth time moment, e(k), can be expressed as follows:

$$e(k+1) = e(k) + \sum_{i=0}^{2} \frac{\partial e(k)}{\partial a_{ff_i}(k)} \Delta a_{ff_i} + \sum_{j=1}^{2} \frac{\partial e(k)}{\partial b_{fb_j}(k)} \Delta b_{fb_j} \tag{3.8}$$

where Δa_{ff_i} and Δb_{fb_j} are the gradient terms of the weighting vector $\Phi_{(\mathbf{a}_{ff}, \mathbf{b}_{fb})}$ with respect to feedforward and feedback weights respectively. These gradient terms of the DNU are given by (derived in Chapter 2)

$$\Delta a_{ff_i} = a_{ff_i}(k+1) - a_{ff_i}(k), \; i = 0,1,2,$$

$$= \mu_{a_i} E \left\{ - e(k) \, \mathrm{sech}^2[v(k)] \, \mathbf{P}_{ff_i}(k) \right\}, \; i = 0,1,2, \text{ and} \tag{3.9a}$$

$$\Delta b_{fb_j} = b_{fb_j}(k+1) - b_{fb_j}(k), \; j = 1,2.$$

$$= \mu_{b_j} E \left\{ - e(k) \, \mathrm{sech}^2[v(k)] \, \mathbf{P}_{fb_j}(k) \right\}, \; j = 1,2. \tag{3.9b}$$

Substituting Eqns. (3.9a) and (3.9b) into Eqn. (3.8), neglecting the activation function components, the expectation operator, and squaring the result gives

$$e^2(k+1) = e^2(k) \left\{ 1 - \sum_{i=0}^{2} \mu_{a_i} \mathbf{P}_{ff_i}(k) \left[\frac{\partial e(k)}{\partial a_{ff_i}(k)} \right]^2 - \sum_{j=1}^{2} \mu_{b_j} \mathbf{P}_{fb_j}(k) \left[\frac{\partial e(k)}{\partial b_{fb_j}(k)} \right]^2 \right\}^2 . \tag{3.10}$$

The learning and adaptive algorithm, derived in the previous chapter, searches for the minimum value of $e^2(k+1)$. This may be satisfied for more than one value of the adaptive gains $\mu_{a_i}(k)$, i = 0, 1,2 and $\mu_{b_j}(k)$, j = 1,2. This can be shown by taking partial derivatives of $e^2(k+1)$ with respect to μ_{a_i} and μ_{b_j} and setting them to zero

$$\frac{\partial e^2(k+1)}{\partial \mu_{a_i}(k)} = 0, \quad \text{and} \quad \frac{\partial e^2(k+1)}{\partial \mu_{b_j}(k)} = 0. \tag{3.11}$$

Substituting Eqn. (3.10) into (3.11) gives

$$2e^2(k)\left\{1 - 2\sum_{i=0}^{2} \mu_{a_i}(k)\, \mathbf{P}_{ff_i}(k)\left[\frac{\partial e(k)}{\partial a_{ff_i}(k)}\right]^2 - 2\sum_{j=1}^{2} \mu_{b_j}(k)\, \mathbf{P}_{fb_j}(k)\left[\frac{\partial e(k)}{\partial b_{fb_j}(k)}\right]^2\right\}\left[\frac{\partial e(k)}{\partial a_{ff_i}(k)}\right]^2 = 0, \tag{3.12a}$$

and

$$2e^2(k)\left\{1 - 2\sum_{i=0}^{2} \mu_{a_i}(k)\, \mathbf{P}_{ff_i}(k)\left[\frac{\partial e(k)}{\partial a_{ff_i}(k)}\right]^2 - 2\sum_{j=1}^{2} \mu_{b_j}(k)\, \mathbf{P}_{fb_j}(k)\left[\frac{\partial e(k)}{\partial b_{fb_j}(k)}\right]^2\right\}\left[\frac{\partial e(k)}{\partial b_{fb_j}(k)}\right]^2 = 0, \tag{3.12b}$$

Assuming $e^2(k+1)$, $\frac{\partial e(k)}{\partial a_{ff_i}(k)}$ and $\frac{\partial e(k)}{\partial b_{fb_j}(k)} \neq 0$ (otherwise, there is no necessity for adaptation!); then both expressions in Eqn. (3.12) yield

$$\sum_{i=0}^{2} \mu_{a_i}(k)\, \mathbf{P}_{ff_i}(k)\left[\frac{\partial e(k)}{\partial a_{ff_i}(k)}\right]^2 + \sum_{j=1}^{2} \mu_{b_j}(k)\, \mathbf{P}_{fb_j}(k)\left[\frac{\partial e(k)}{\partial b_{fb_j}(k)}\right]^2 = 0.5. \tag{3.13}$$

Equation (3.13) represents a constraint on the optimum values of the adaptive gains $\mu_{a_i}(k)$ and $\mu_{b_j}(k)$. This constraint can be satisfied by several possible (may be infinite) sets of $\mu_{a_i}(k)$ and $\mu_{b_j}(k)$. However, the occurrence of multiple solutions can be avoided by imposing stability triangle criterion [48] on the feedback weights of the DNU, namely b_1 and b_2. This still may lead to several possible convergence values for the controller parameters while minimizing $e^2(k+1)$. One is free to choose any values of $\mu_{a_i}(k)$ and $\mu_{b_j}(k)$ as long as Eqn. (3.13) is satisfied. The partial derivatives of $e^2(k+1)$ with respect to adaptive gains, $\frac{\partial e^2(k+1)}{\partial \mu_{a_i}(k)}$, i = 0,1,2 and $\frac{\partial e^2(k+1)}{\partial \mu_{b_j}(k)}$, j = 1,2, may serve as a measure of the expected reduction of $e^2(k+1)$. Further, the values of $\mu_{a_i}(k)$ and $\mu_{b_j}(k)$ are the measures of the strength of the adaptation: larger values of $\mu_{a_i}(k)$ and $\mu_{b_j}(k)$ imply stronger adaptation, and vice versa. Thus, the idea of choosing individual values for adaptive gains, $\mu_{a_i}(k)$; i = 0,1,2 and $\mu_{b_j}(k)$; j = 1,2, is to adapt the parameters having larger gradients more than the parameters having

smaller gradients. It is very desirable to choose proper values for the adaptive gains μ_{a_i} and μ_{b_j} so that a global optimum performance may be obtained, and this needs further investigation.

Due to the problem of local minima, the individual controller parameters may not exactly and inversely match the plant parameters. However, the overall transfer function of the controller is the inverse of that of the plant, which therefore makes the transfer function from output-to-input unity. Certainly, this would be a useful attribute in the control of unknown systems. More often than not, in many control applications, satisfying the input-output equivalency is more crucial and important than the system identification in an exact sense. The convergence of error between the desired and the actual signals is viewed very significant. It is, of course, a desirable function of the algorithm to give a global optimal solution. Indeed, for a controller scheme that does not display such global performance, the controller can be trapped in a false minimum [48]. Therefore, it may be necessary to have a scheme-specific warning regarding the appropriate controller parameter initializations or avoidance of certain source (input) and controller combinations [53]. Due to these constraints, the definition of the global performance has to be rephrased.

The question of global "performance" convergence is rephrased, therefore, as whether or not a particular combination of source (input signal), and initialization of the controller parameters will only admit locally attractive adapted controller parameterizations that yield the optimum available performance in terms of input-output equivalency [53, 57, 61]. In terms of the parameter-space trajectories of the adapted parameters, all of the attractive basins have sinks that make the controller an inverse-dynamic model of the plant under control. Therefore, an admissible source (input), controller weight initializations and plant combination need not always result in an optimum fit of the controller to the plant inverse. All that is required for allowable control is that each locally stable stationary point of the average behavior of the adaptive controller algorithm results in a controller output that can be passed on to the plant which can result in an input-output equivalency.

With this description in context, computer simulations have been carried out in this work to study the IDAC scheme using DNU for linear systems. The details of the simulation studies are given in the next section.

3.3 Computer Simulation Studies for Linear Systems

In this thesis, discrete-time systems which can be represented by difference equations of the form

$$q(k+1) = f[q(k), u(k)],$$
$$y(k) = g[q(k)], \quad k = 0, 1, 2.... \tag{3.14}$$

are considered. In the above equation u(.) and y(.) represent the input and output of the plant, q(.) denotes the state of the plant, and f and g are static nonlinear mapping functions. If the system described by Eqn. (3.14) is linear and time-invariant, the equation governing its behavior can be expressed as

$$y(k+1) = \sum_{i=0}^{n-1} \alpha_i \, y(k-i) + \sum_{j=0}^{m-1} \beta_j \, u(k-j). \tag{3.15}$$

In the computer simulation studies, a dynamic plant with unknown parameters was cascaded with the dynamic neural unit discussed in the earlier section. If the error is defined as e(k) = s(k) - y(k), where s(k) is the reference input and y(k) is the actual output of the plant under control, the objective is to determine a bounded control input u(k) which results in

$$\lim_{k \to \infty} [s(k) - y(k) = e(k)] = 0 \tag{3.16}$$

such that the output follows the input as closely as possible.

The simulation program for the IDAC scheme was developed in the VAX/VMS environment. This program allowed the tolerance value of the error, ε_{tol}, to be set at the start of the program. In the simulation results presented in this section, ε_{tol} was set to 0.05. Four simulation examples are discussed in this chapter. In Example 1, a general linear plant described by Eqn. (3.15) and excited by a unit step input was considered. Following the error convergence to the preset tolerance value, a different input signal was applied to the system in order to validate the inverse model that was obtained. The objective of Example 2 was to demonstrate the adaptive capability of the control scheme. Here, the control scheme was made to respond to the variations in the plant parameters and input signal variations. The control of a plant under structural perturbations and an unstable plant were demonstrated in Examples 3 and 4 respectively.

Example 1: Learning and control of an unknown plant

Consider a linear plant described by the following difference equation

$$y(k+1) = \sum_{i=0}^{2} \alpha_i \, y(k-i) + \sum_{j=0}^{2} \beta_j \, u(k-j). \qquad (3.17)$$

with $\alpha_{fb} = [1.0, 0.7, 0.7]^T$ and $\beta_{ff} = [1.2, 1, 0.8]^T$. This is a second-order plant with two poles and two zeros located at $(-0.35 \pm j\, 0.76)$ and $(-0.42 \pm j\, 0.7)$ respectively. The initial values of the DNU weights were arbitrarily set to: $\mathbf{a}_{ff_0} = [0.4, 0.2, 0.2]^T$ and $\mathbf{b}_{fb_0} = [1.0, 0.2, 0.2]^T$. These initial values correspond to zeros and the poles located respectively at $(-0.25 \pm j\, 0.66)$ and $(-0.1 \pm j\, 0.44)$ The system was excited by a unit step input. The objective of this example was to show that the learning and control actions were performed simultaneously. In addition, the continuous adaptation capability of the IDAC scheme was demonstrated by changing the input signal to a sinusoidal signal, $s(k) = \sin(2\pi k/250)$ in the interval [-1,1] after obtaining the approximate inverse-model of the plant. The simulation results obtained for this example are shown in Fig. 3.3.

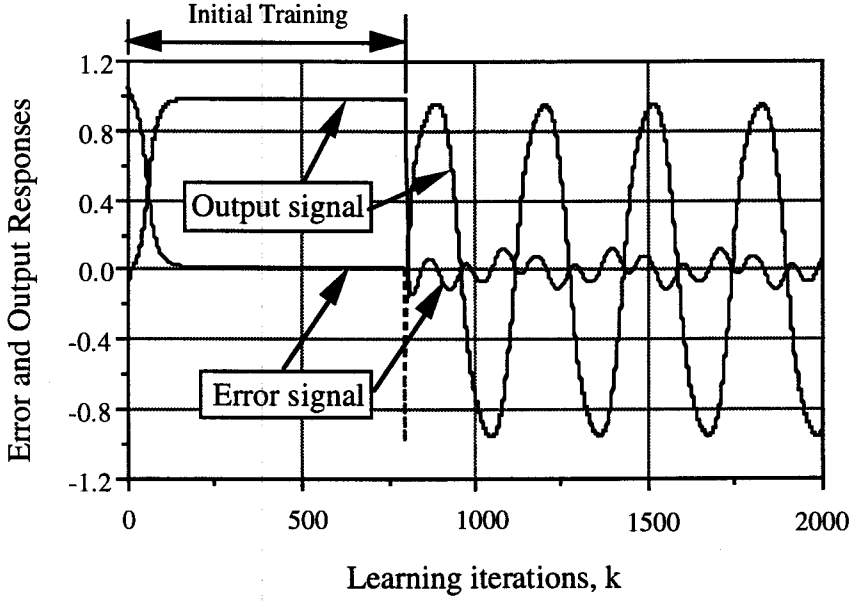


Figure 3.3: The error and output responses before and after obtaining the inverse-model of the plant under control, Example 1.

From the optimal (converged) weight values of the DNU, the zeros and the poles of the dynamic structure of the DNU were found to be respectively at (-0.342 ± j0.756) and (-0.408± j 0.714). The transfer function of the dynamic structure may be written as

$$w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) = \frac{u(k)}{s(k)} = \frac{(z + 0.342 - j\,0.756)\,(z + 0.342 + j\,0.756)}{(z + 0.408 - j\,0.714)\,(z + 0.408 + j\,0.714)}. \qquad (3.18)$$

Since the DNU represents an approximate inverse-model of the plant, the estimated plant transfer function may be written as

$$\hat{G}_p(k, \alpha_{fb}, \beta_{ff}) = \frac{y(k)}{u(k)} = \frac{(z + 0.408 - j\,0.714)\,(z + 0.408 + j\,0.714)}{(z + 0.342 - j\,0.756)\,(z + 0.342 + j\,0.756)} \qquad (3.19)$$

which makes the transfer relation from the output to the input nearly unity. This implies that the plant can follow changes in the command signal with very little error between the desired command signal and the actual response. The existence of the error after this initial learning phase indicates a mismatch between the ideal and the obtained inverse-model. To demonstrate this, the input signal was changed from a step to a sinusoidal signal at k = 800. The effect on the error and output responses of changing the command signal is also shown in Fig. 3.3. As can be observed from this figure, there is an error in the limits ± 0.2 due to a slight mismatch between the plant, Eqn. (3.17), and its inverse model, Eqn. (3.18). However, the error that may exist after training between the plant and its inverse model could be reduced by using a persistently exciting signal during training. This is because a persistently exciting signal can excite all the modes of the plant under control [15, 16] which may result in the true inverse model. There are, however, certain problems associated with persistently exciting signals being used in adaptive control systems as discussed in [8, 15].

Example 2: Plant with varying dynamics and input signal

The purpose of this example was to show the adaptive capability of the DNU to perturbations in the plant parameters, changes in the plant configuration, and variations in the input signal. The plant considered in this case was governed by Eqn. (3.17) with $\beta_{ff} = [1, 1.2, 0]^T$ and $\alpha_{fb} = [1.3, 0.8, 0]^T$. This was a first-order plant with a zero at (-1.2) and a pole at (-0.62). As the zero was outside the unit circle, the plant was of non-minimum phase type. At time k = 600, the plant configuration was changed to $\beta_{ff} = [1, 1.2, 0.7]^T$ and $\alpha_{fb} = [1.3, 0.8, 0.6]^T$. This change in configuration made the plant a second-order one with the zeros and poles located respectively at (-0.6 ± j 0.58) and (-0.31 ± j 0.61). At k = 1000, another perturbation was injected into the plant by making the parameter $\alpha_2 = 0$, which

changed the location of the plant zeros to (0 ± j 0.68). The input signal was also varied in the interval [-1, 1] as follows:

$$s(k) = 0.6, \text{ for } 1300 \leq k < 1425, \quad s(k) = 0.2, \text{ for } 1425 \leq k < 1575$$

$$s(k) = -0.2, \text{ for } 1575 \leq k < 1650, \quad s(k) = -1.0, \text{ for } 1650 \leq k < 2000. \qquad (3.20)$$

The simulation results obtained for this example are shown in Fig. 3.4. As can be observed from this figure, the DNU could make the plant follow the desired trajectories inspite of perturbations.

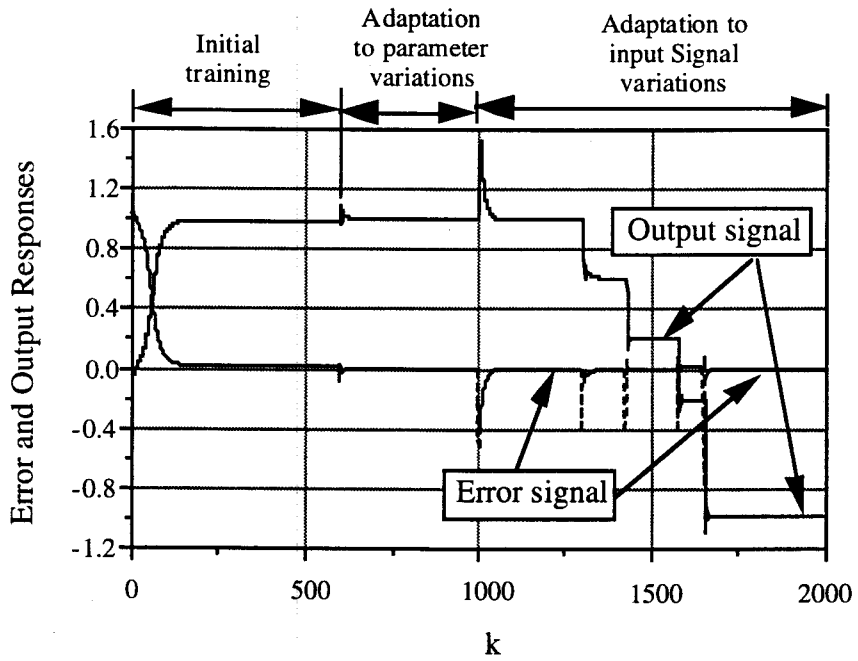


Figure 3.4: The error and output responses before and after obtaining the inverse-model of the plant under control, Example 2.

Example 3: Plant under structural perturbations

In this example, a linear plant with the following parameter values $\beta_{ff} = [1, 1.2, 1.4]^T$ and $\alpha_{fb} = [1.1, 1, 0.8]^T$ was considered. The poles and zeros of this plant were (-0.45 ± j 0.72) and (-0.6 ± j 1.02) respectively. At k = 500, the plant configuration was changed from second to first-order resulting in structural perturbations; that is, the parameters β_2 and α_2 were set to zero. From the simulation results shown in Fig. 3.5, it can be observed that the effect of the change in the dynamics was not significant on the response of the plant. Again, the plant dynamics were changed at k = 1200 by making $\beta_2 = 0.6$ and $\alpha_1 = 0$ which located

the zeros at (-0.6 ± j 0.49) and the poles at (0 ± j 0.85). The latter change in the plant dynamics made the error signal increase rapidly, as can be seen in Fig. 3.6, but the controller was able to reduce the error to the tolerance value very quickly.

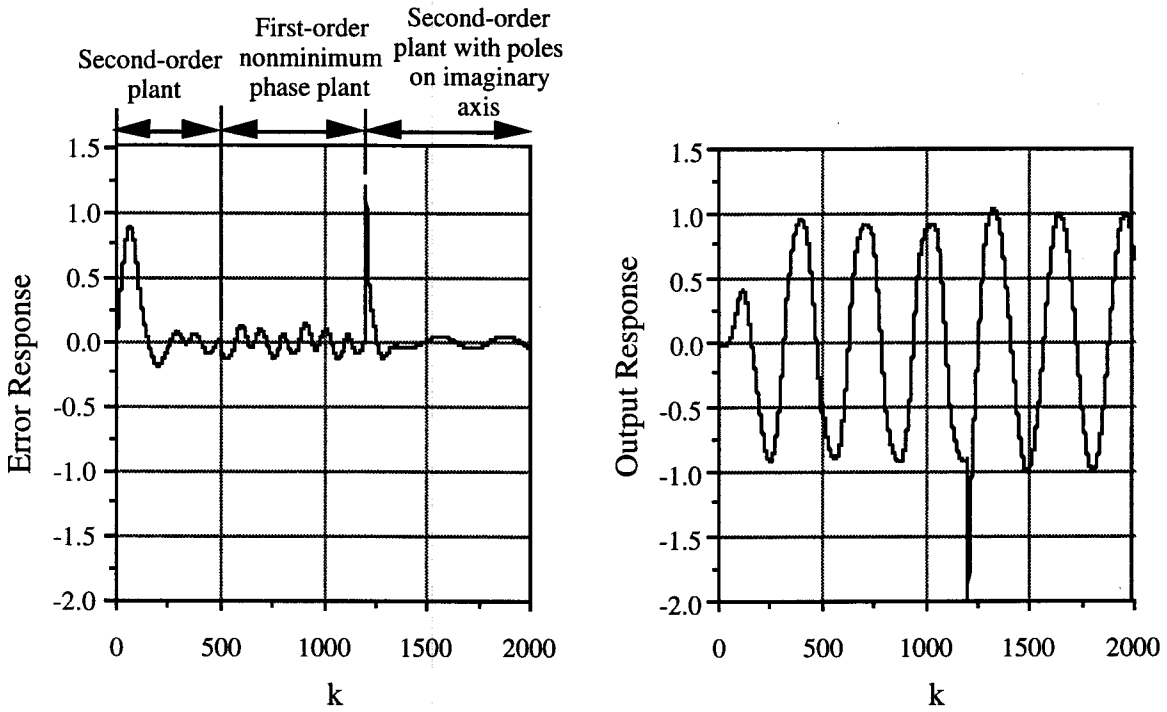


Figure 3.5: The error and output responses of a plant under parameter perturbations, Example 3.

Example 4: Control of an unstable plant

An unknown plant described by the following relation was considered in this example

$$0.8\ y(k) = -\ 0.9\ y(k-1) + 1.2\ u(k) - 1\ u(k-1). \qquad (3.21)$$

From this difference equation it is observed that the plant was of first-order, with a pole at (1.13) and a zero at (0.83) in the z-plane, and was unstable. This plant was precascaded with the same controller structure and had the same initial settings as in Example 1. The simulation results obtained for this situation are shown in Fig. 3.6. Although this plant was unstable, the output error was bounded and remained within the tolerance limits, and the feedback control input to the plant was bounded. It was also observed in this simulation study that the effect of the initial values of the DNU parameters, in particular the feedback weights, on the transient response was significant. The effect of the initial values of the

feedback weight b_1 is shown in Figs. 3.6a and 3.6b. Some initial settings of the DNU parameters led to system instability.

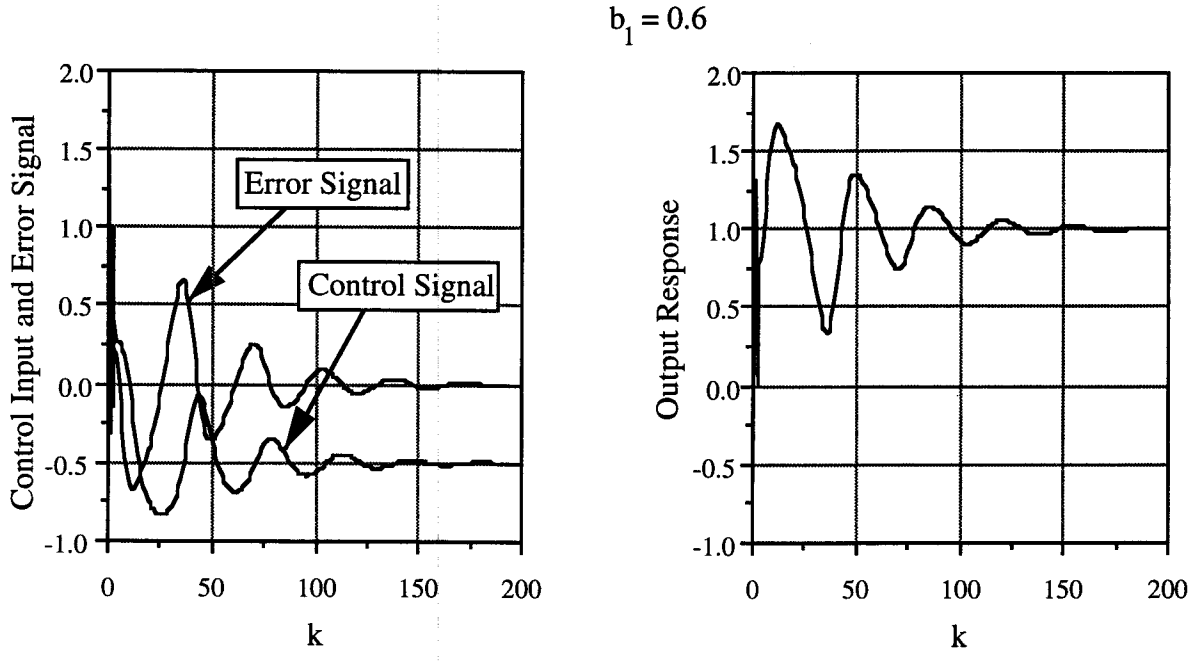


Figure 3.6: The error, control and output responses of an unstable plant for different initial settings of the feedback weight b_1, Example 4.

3.4 Feedback-Error Learning Scheme

3.4.1 The Principle

The IDAC scheme discussed in the preceding section was used as a feedforward inverse controller that made unknown linear systems follow desired trajectories. However, the IDAC scheme was an open-loop learning system. The stability of the IDAC scheme depends on the initial values of the parameters of the DNU. In this section, a feedback-error learning scheme [58] that consists of a linear feedback controller and the DNU as a feedforward inverse controller, as shown in Fig. 3.7, is discussed. This scheme consists of a fixed gain proportional-plus-derivative (PD) linear feedback controller that makes the overall system stable, and a feedforward controller which updates its internal weights to generate the control signal $u_{nn}(k)$ in the process of becoming an inverse model of the plant. During the initial training period, the control signal $u_{nn}(k)$ was very insignificant. The control signal from the feedback controller, $u_c(k)$, was significant because of the large initial error. Hence, in the early stage of learning, the component $u_c(k)$ was dominant over the $u_{nn}(k)$. However, as the learning trials increased $u_{nn}(k)$ became dominant over $u_c(k)$. The feedback controller guarantees the stability of the overall system [58, 62, 63]. In general, the feedback-error learning scheme has the following advantages [58]: (i) a teaching signal is not required to train the neural network, instead, the error signal is used as the training signal, (ii) the learning and control are performed simultaneously in sharp contrast to the conventional 'learn-then-control' approach, and (iii) back-propagation of the error signal through the controlled object or through the model of the controlled object is not necessary.

In Fig. 3.7, $G_p[k, \alpha_{fb}, \beta_{ff}]$ represents a dynamic plant, $C[e(k), \Delta e(k)]$ is a linear function of the error and the change of error representing a PD control law. The dynamics of the overall system shown in Fig. 3.7 are described by the following equations:

$$e(k) = s(k) - y(k) \tag{3.22a}$$

$$\Delta e(k) = e(k) - e(k-1) \tag{3.22b}$$

$$C[e(k), \Delta e(k)] \equiv k_p\, e(k) + k_d\, \Delta e(k) = u_c(k) \tag{3.22c}$$

$$u_{nn}(k) = \Psi\left[\left(w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\right)s(k)\right] \tag{3.22d}$$

$$u(k) = u_{nn}(k) + u_c(k) \tag{3.22e}$$

$$y(k) = G_p[k, \alpha_{fb}, \beta_{ff}]\, u(k). \tag{3.22f}$$

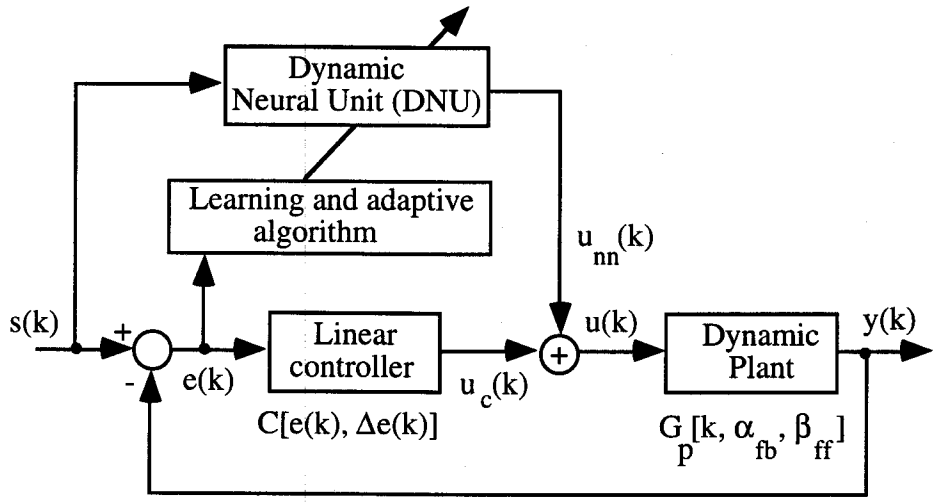


Figure 3.7: The feedback-error learning scheme consisting of a linear feedback controller and a feedforward neuro-controller.

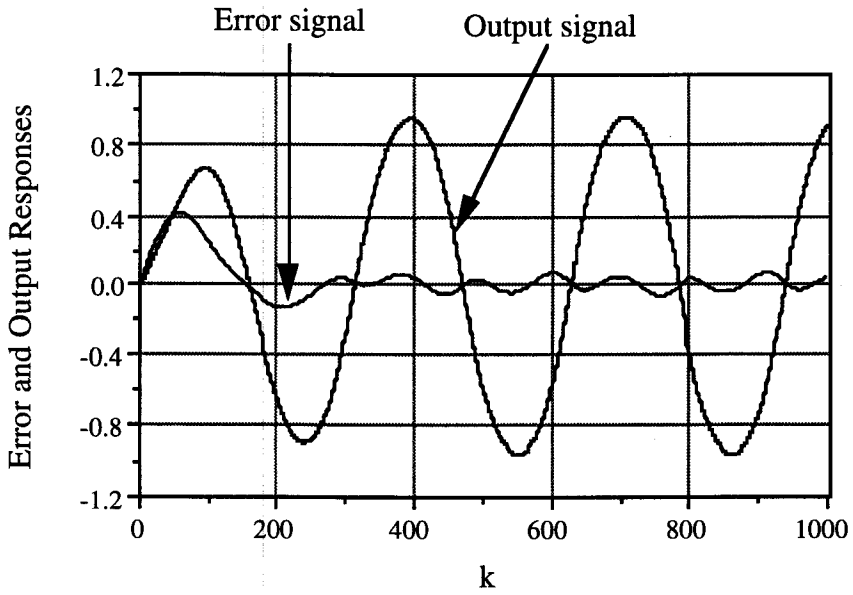
In Eqn. (3.22c), k_p and k_d denote the proportional and differential gains respectively. The DNU, once trained, will represent the inverse dynamics model of the dynamic plant. The fixed gain PD controller ensures adequate performance prior to the convergence of the DNU parameters, and reduces the steady-state output errors due to the disturbance inputs [62]. In essence, the output of the PD controller is an indication of the mis-match between the dynamics of the plant and the inverse dynamics model obtained by the DNU. This is because if the true inverse dynamics model has been learned, the DNU alone will provide the necessary control signal to achieve the desired trajectory. With zero trajectory error, the PD controller produces no output and, hence, indicates that learning has been completed [58, 62]. In the computer simulation studies discussed in the following subsection, the DNU parameters, $\mathbf{a}_{ff}$ and $\mathbf{b}_{fb}$, were adjusted based on the algorithm derived in Section 2.3.

3.4.2 Computer Simulation Studies

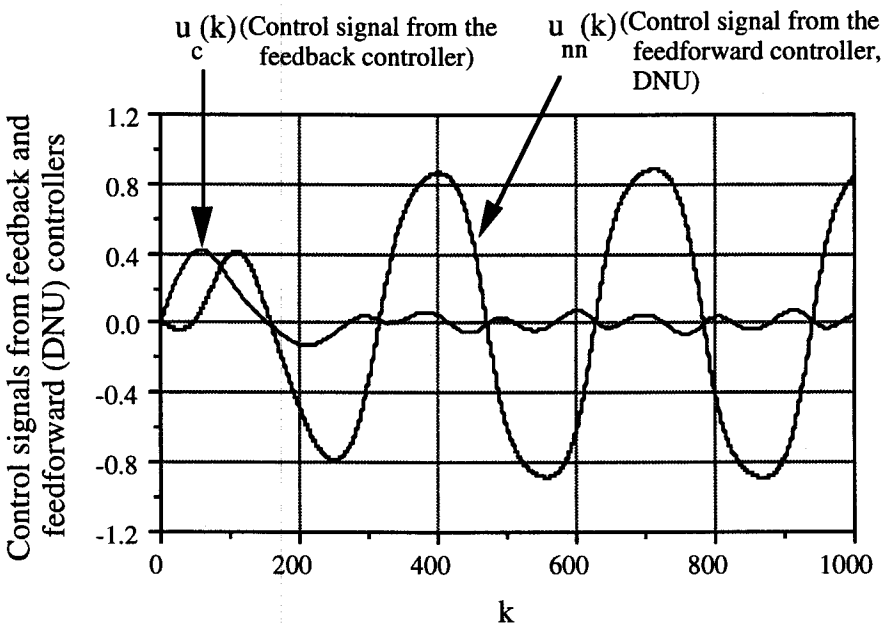
Example 1: Learning and control of an unknown plant

A plant described by Eqn. (3.15) with the following parameter values $\alpha_{fb} = [1.3, 0.9, 0.7]^T$ and $\beta_{ff} = [1.2, 1, 0.8]^T$ was considered. The input signal to the system was a sinusoidal signal $s(k) = \sin (2\pi k / 250)$ in the interval [-1,1]. The simulation results obtained for this example are shown in the following figures. Figure 3.8a shows the error and plant output responses. Figure 3.8b shows the control signal components generated by the feedback and the feedforward controllers. As seen from this figure, the control signal from the feedback

controller, $u_c(k)$, was predominant during the initial period of training, and was reduced as the feedforward controller took over which resulted in the dominance of the control signal from the later; that is $u_{nn}(k)$.



(a)



(b)

Figure 3.8: Simulation results, Example 1.

(a) The error and output responses,

(b) The control signals generated by the feedback and the feedforward controllers.

Example 2: Plant with varying dynamics

In this simulation example, the parameter adaptation capability of the feedback learning scheme was demonstrated by introducing perturbations in the plant parameters. The plant considered here was the same as in Example 1. At k = 500, the following parameter changes were made:

$$\left.\begin{aligned}\beta_2 &= 0\\ \alpha_2 &= 0\end{aligned}\right\}, \text{ for } 500 \le k \le 1000$$

thereby making the plant a first-order system with parameters: $\alpha_{fb} = [1.3,\ 0.9,\ 0]^T$ and $\beta_{ff} = [1.2,\ 1,\ 0]^T$. The error and plant responses for this case are shown in Fig. 3.9. From this simulation example it can be observed that the effect of changing the plant configuration on the system response was very insignificant.

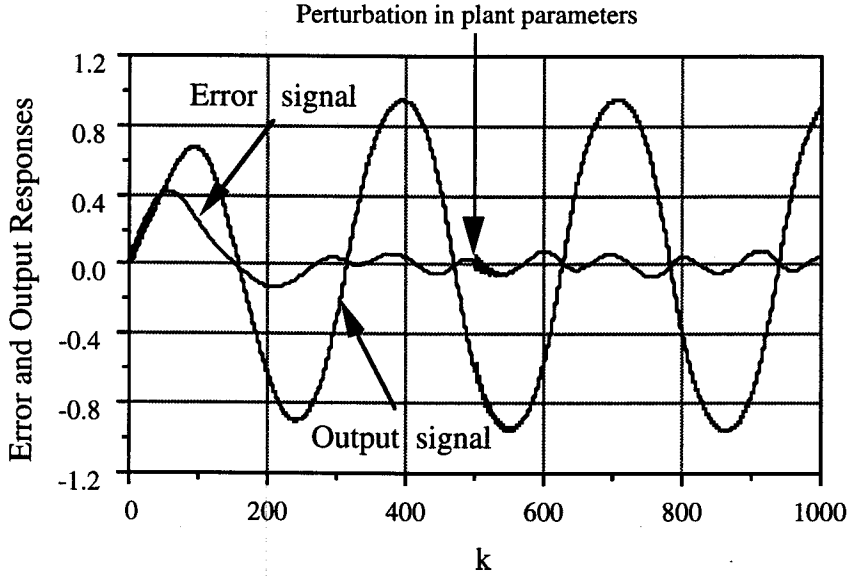


Figure 3.9: Error and output responses of the plant under parameter perturbations, Example 2.

Example 3: Space vehicle control problem

In this example, a model of the simplified space vehicle control system shown in Fig. 3.10 was considered. The purpose of this control system is to control the attitude of the space vehicle in one dimension. Assuming a rigid structure, the vehicle is represented by a pure inertia J_v, so that the transfer function between the applied torque and the output position is

$\frac{c(s)}{J(s)} = \frac{1}{J_v s^2}$. The position c(t) and velocity v(t), are fed back by the position and rate sensors, respectively, to form the closed loop control, where the transfer function is

$$G(s) = \frac{C(s)}{R(s)} = \frac{k_p}{J_v s^2 + k_r s + k_p}. \tag{3.23}$$

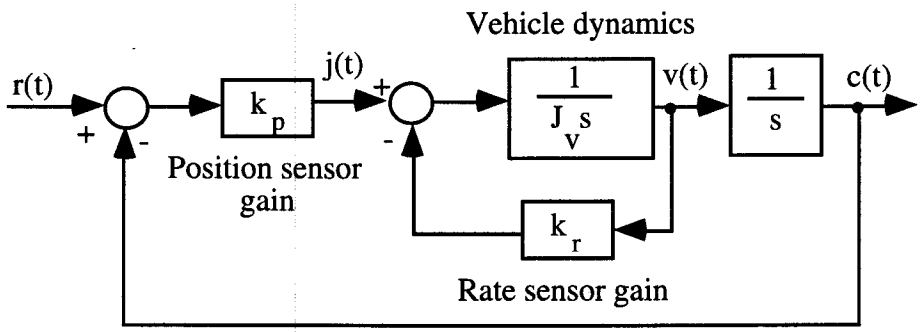


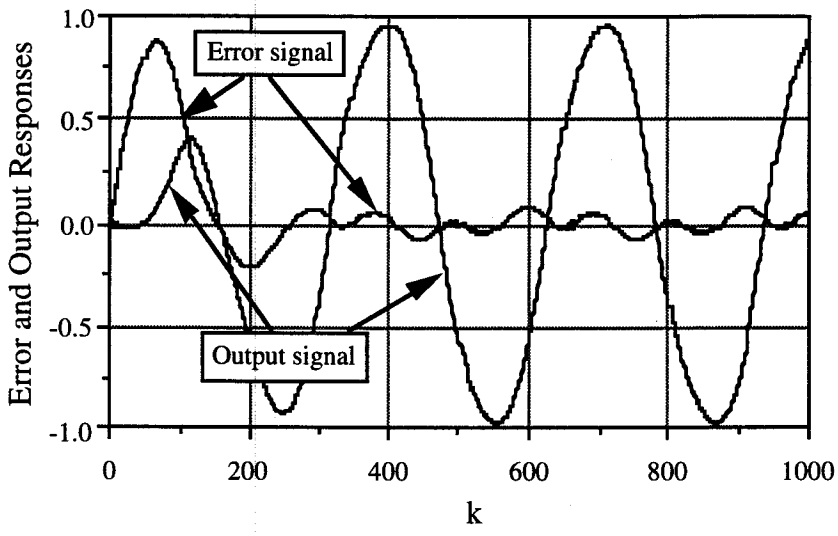
Figure 3.10: Block diagram of a simplified space vehicle with rigid dynamics.

The closed loop transfer function of the equivalent discrete system, using z-transforms, is [64]

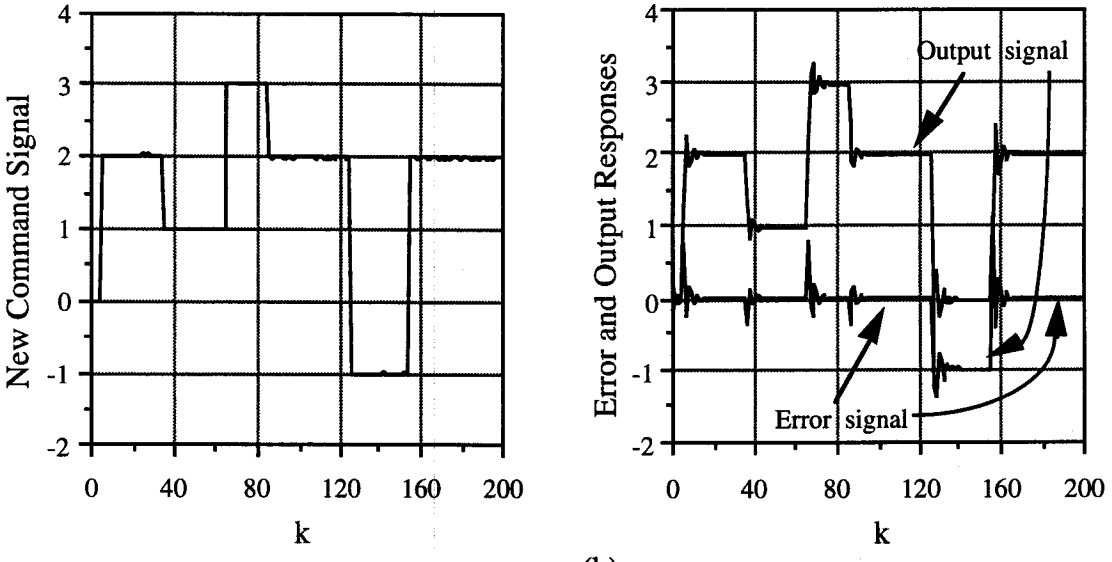
$$G(z) = \frac{T^2 k_{pg}(z+1)}{2J_v z^2 + (2 k_{rg} T - 4J_v + T^2 k_{pg})\, z + (2J_v - 2 k_{rg} T + T^2 k_{pg})}. \tag{3.24}$$

The parameters used in the simulation study were: k_{pg} = position sensor gain = 1.65 X 10^6, k_{rg} = rate sensor gain = 3.17 X 10^5, J_v = moment of inertia of vehicle = 41822. For T = 0.225 secs, the closed loop poles and zeros were: (-0.352 ± j 0.4114) and -1 respectively.

The above block diagram of the space vehicle system was used as the dynamic plant to be controlled in Fig. 3.7, and the desired trajectory was a sinusoidal signal in the interval [-1,1]. Following the convergence of the error and system responses to the desired values, the converged DNU parameter values corresponded to the following pole and zero locations respectively: (-1, 0.05) and (- 0.421 ± j 0.387). For practical purposes, this is a close inverse approximation of -1 and (-0.352 ± j 0.4114) respectively. The error and system responses obtained are shown in Fig. 3.11a. After this initial learning and control, the system was excited by an arbitrary signal. Except for small transients at the beginning, the system was able to follow the new command input as shown in Fig. 3.11b thereby demonstrating that the DNU was an approximate inverse model of the space vehicle system. The output of the PD controller was an indication of the mis-match between the obtained inverse model and that of the dynamic plant.



(a)



(b)

Figure 3.11: Simulation results of a space vehicle control system, Example 3.

(a) The error and output responses of a space vehicle during the initial learning and control phase,

(b) New command input and the corresponding error and output responses.

Example 4: Control of nonlinear plants

The purpose of this simulation example was to demonstrate that the feedback error learning control scheme shown in Fig. 3.7 could be used to drive simple nonlinear systems to follow the desired trajectories.

Case (i): In this case, a linear plant preceded with an actuator with deadzone, shown in Fig. 3.12, was considered. In this figure, the characteristics of the actuator with deadzone are described by the function

$$\begin{aligned} D[u] &= u(k) - d && \text{if } u(k) > d, \\ &= 0 && \text{if } -d \le u(k) \le d, \\ &= u(k) + d && \text{if } u(k) < -d \end{aligned} \qquad (3.26)$$

where the parameter 2d represents the width of the deadzone. The control signal to the plant is then given by

$$v(k) = D[u(k)]. \qquad (3.27)$$

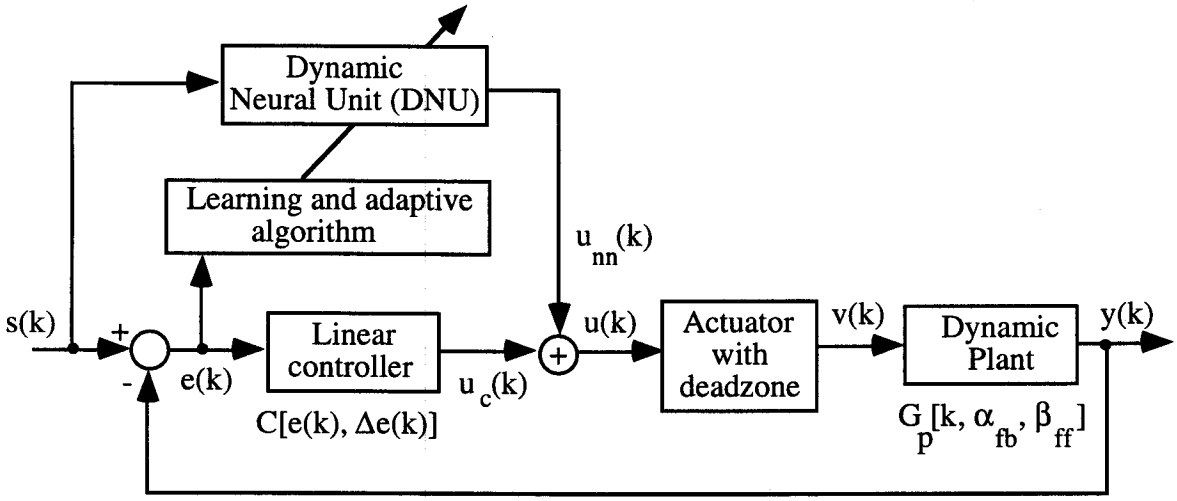


Figure 3.12: The feedback-error learning scheme with deadzone.

In this simulation study, a linear plant with the same parameters as in Example 1 was considered. The values of the proportional and derivative gains and the width of the deadzone were respectively 0.7, 10, 0.2. The system was excited by an unit step input. The simulation results for this case are shown in Fig. 3.13. The output responses of the plant with and without DNU are shown in Fig. 3.13a. From this figure it is clear that the PD controller could not drive the system to follow the step input. The same behavior was observed for

different values of the proportional and derivative gains. With both the feedback (PD) and the feedforward (DNU) controllers connected as shown in Fig. 3.12, the control signals obtained from these controllers are shown in Fig. 3.13b. From this figure it is observed that the control signal from the feedback controller was significant during the initial learning period, while the DNU output was more predominant as the learning and control actions continued.

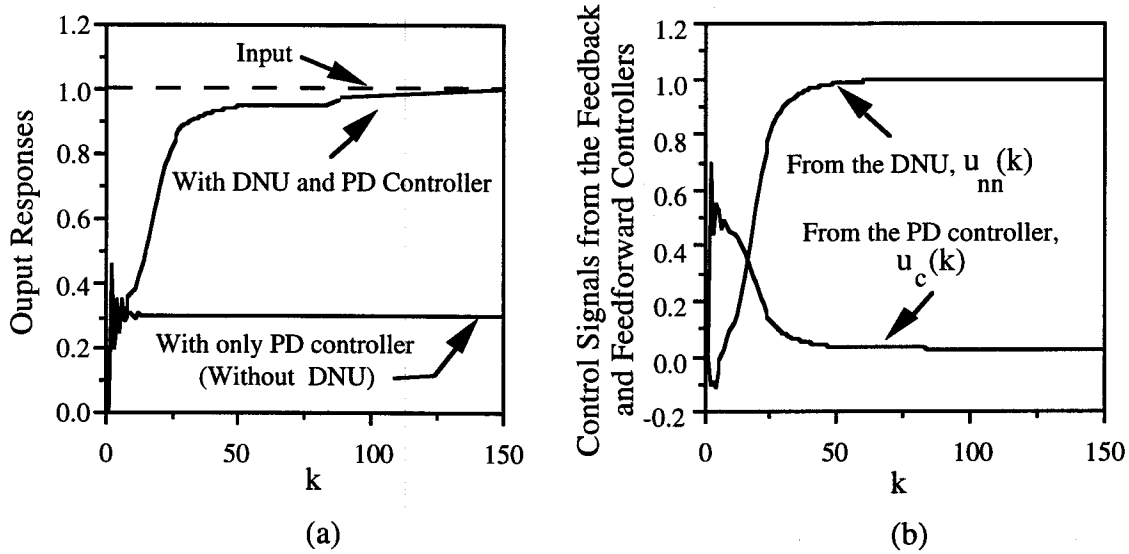


Figure 3.13: Simulation results of a plant with deadzone, Example 4, Case (i).
(a) The output responses of the plant with and without DNU,
(b) The control signals from the feedback (PD) and the feedforward (DNU) controllers.

Case (ii): A nonlinear plant described by the following difference equation

$$y(k+1) = u(k) + 0.5\, y^3(k) \tag{3.28}$$

was considered in this simulation example. The system was excited by a square input in the interval [-0.8, 08]. The simulation results were observed for 800 iterations. The input and output signals of the nonlinear plant are shown in Fig. 3.14a and the control signals of the feedforward and the feedback controllers are shown in Fig. 3.14b. From the output response it can be seen that the DNU could drive the nonlinear plant to follow the desired signal.

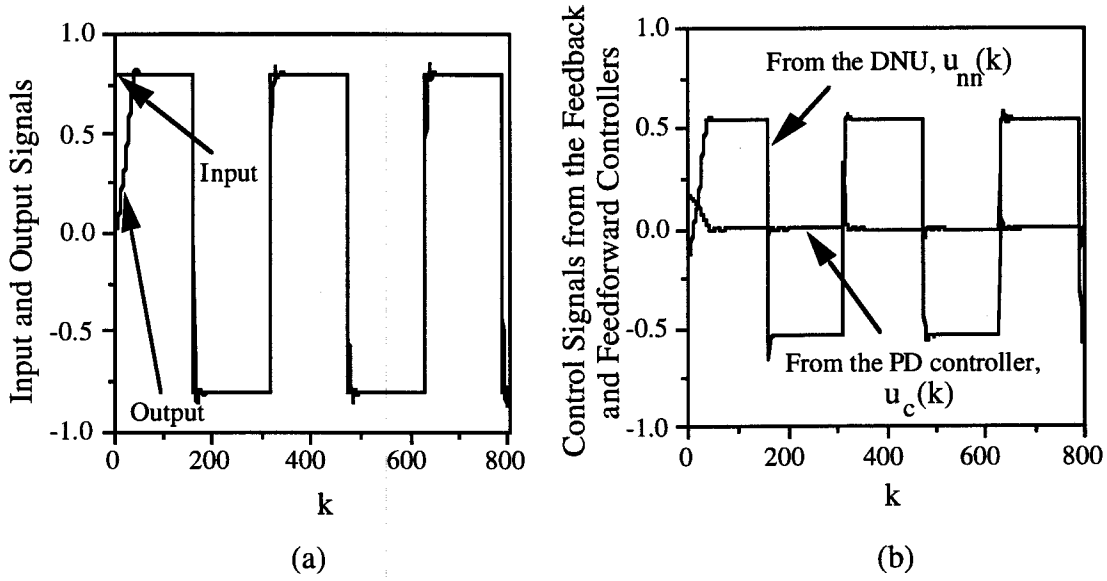


Figure 3.14: Simulation results of a plant with deadzone, Example 4, Case (ii).
(a) The input and output signals of the nonlinear plant,
(b) The control signals from the feedback (PD) and the feedforward (DNU) controllers.

3.5 Summary

An inverse dynamic adaptive control (IDAC) technique using the DNU has been discussed in this chapter. The effectiveness of the DNU as an inverse dynamic controller has been demonstrated through computer simulation studies. In this control scheme, the dynamic structure of DNU is made to be an approximate inverse model of the plant under control thereby achieving almost unity mapping between the input and output signal space. Extensive simulation studies have been conducted using the DNU and reported in [56, 57]. The application of DNU to equalization problems in communication channels is discussed in [65]. In the simulation examples considered in this chapter, the order of the plant did not exceed the order of the DNU structure and, therefore, it may be expected that it could adjust its weights to satisfy the input-output equivalency. In a situation where the order of the plant exceeds that of the controller, the scheme may not function satisfactorily. To circumvent this problem, it was proposed [66] that a single DNU be replaced with a set of DNU modules in the controller structure. The learning and adaptive algorithm decides the number of controller modules to be activated to match the order of the dynamic system under control. This scheme has been tested up to the fourth-order linear plants [66].

A feedback-error learning scheme with DNU as a feedforward controller has been presented. As can be seen from the simulation results, this learning scheme was able to adaptively control unknown linear and simple nonlinear plants in the presence of parameter perturbations and input signal variations. However, the performance of the feedback-error learning scheme was poor and sometimes unstable, when complex nonlinear characteristics were introduced into the plant under control. This implies that the functional approximation capabilities of a single DNU are limited.

Although a DNU can control linear and simple nonlinear systems, the power of neural computation comes from the network of such neural units. Furthermore, as many practical control problems are nonlinear in nature, it is desirable to develop a neural network structure with the DNU as the basic computing node. This dynamic neural structure can approximate nonlinear functions, and can be utilized in developing controllers for nonlinear systems. These aspects are discussed in the next chapter.

4. Dynamic Neural Structure for Nonlinear Systems

4.1 Introduction

Although a single neuron can be used to control linear and simple nonlinear systems, as was demonstrated in the previous chapter, and can perform certain simple pattern detection functions [5], it has been demonstrated in the literature that the real power of neural computation comes from the neurons connected in a network structure. Larger networks generally offer greater computational capabilities. Arranging neurons in layers or stages is believed to mimic the layered structure of a certain portion of the brain. These multi layer networks have been proven to have capabilities beyond those of a single layer [28, 29, 67].

It is well established that feedforward neural networks with at least one hidden layer can approximate nonlinear functions to a desired degree of accuracy [67 - 78]. As a result of this attribute of feedforward neural networks, many researchers now use them to model dynamic systems. However, dynamic neural networks offer computational advantages over purely static neural networks. For example, it is well known that an infinite order FIR filter, which is only a feedforward network, is required to emulate a single pole IIR filter [29]. The architectures of dynamic neural networks, by and large, are developed based on the understanding of the cerebellum and its associated circuitry [22, 36, 37].

Unlike a static neural network, a dynamic neural network employs extensive feedback connections between the neurons. The node equations in dynamic networks are described by differential or difference equations. The response of such networks is dynamic or recursive; that is, after applying a new input, the output is calculated and fedback to modify the input. The output is then recalculated, and the process is repeated. For a stable network, successive iterations produce smaller and smaller output changes until eventually the outputs become constant [28, 29]. In some situations, the process may never converge, and such networks are said to be unstable. Unstable networks have interesting properties and one example of such networks is *chaotic systems* [30].

Neural architectures with feedback are particularly appropriate for system modeling (identification), control and filtering applications. These networks are important because most physical systems are nonlinear and dynamic. In this chapter, one such dynamic neural structure is developed using the DNU as the computing node. The mathematical development and the implementation scheme of the proposed neural structure are presented in the next section. It is demonstrated in Section 4.3 that a dynamic neural network with

DNUs as the functional elements can approximate arbitrary nonlinear functions. Computer simulations are provided in this section to illustrate the functional approximation capability of this dynamic neural network. This property is used to synthesize a controller for nonlinear dynamic systems and is discussed in Section 4.4. It is followed by a summary in the last section.

4.2 Multi-Stage Dynamic Neural Structure

Multi layer networks may be formed by cascading a group of single layers, where the output of one layer provides the input to the subsequent layer. These networks may not provide much functional capabilities over a single layer network unless there is a nonlinear activation function between the layers [31]. Calculating the output of a layer in a neural network consists of multiplying the first weight matrix, (if there is no activation function) by the second weight matrix. This can be expressed as $\mathbf{S}(w^1)w^2$, where $\mathbf{S}$ is the input vector, and w^1, w^2 are the weight matrices of the two layers. Since matrix multiplication is associative, the terms may be regrouped as $\mathbf{S}(w^1w^2)$. This shows that a two-layer neural network without nonlinear activation functions is exactly equivalent to a single layer having a weight matrix equal to the product of the two weight matrices. Hence, any multi layer linear network can be replaced by an equivalent one-layer network. Linear networks are severely limited in their computational capability. Hence, nonlinear activation functions are vital to the expansion of the capabilities of neural networks. In general, multi-stage (multi layer) neural networks can be considered as versatile nonlinear maps with the elements of the weight matrices as parameters [34]. In this section, a multi-stage dynamic neural structure with a DNU as the basic computing node is developed.

4.2.1 Mathematical Development

Let the output of a DNU be written as

$$\omega(x) = \Psi\left[\left(w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\right)s(k)\right] \tag{4.1}$$

where $\Psi[.]$ is a sigmoid nonlinear activation function.

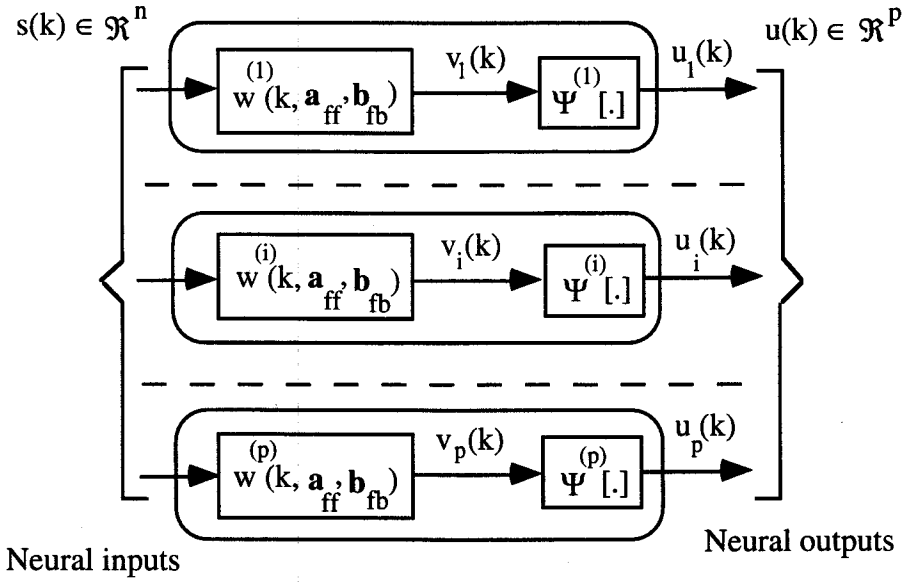
The input-output mapping of a single-stage of DNUs in sigma (parallel) configuration, shown in Fig. 4.1a, can be expressed as

$$u(k) = \Psi^{(l)}\left[\left(w^{(l)}(k, \mathbf{a}^1_{ff}, \mathbf{b}^1_{fb})\, s^{(l)}(k)\right)\right] + ... + \Psi^{(i)}\left[\left(w^{(i)}(k, \mathbf{a}^1_{ff}, \mathbf{b}^1_{fb})\, s^{(i)}(k)\right)\right] + ...$$

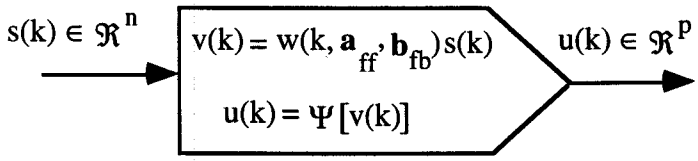
$$+ \Psi^{(p)}\left[\left(w^{(p)}(k, \mathbf{a}_{ff}^{p}, \mathbf{b}_{fb}^{p})\, s^{(p)}(k)\right)\right]$$

$$= \omega_1(s) + \ldots + \omega_i(s) + \ldots + \omega_p(s) = \sum_{n=1}^{p} \omega_n(s), \quad n = 1,..,i,..,p. \tag{4.2}$$

An equivalent representation of a one-stage dynamic neural network with 'p' DNUs in parallel (in sigma mode) is shown in Fig. 4.1b.



(a)



(b)

Figure 4.1: Dynamic neural structure with a DNU as the basic computing node.

(a) The sigma connection of 'p' DNUs to form a single-stage dynamic neural network, and

(b) The equivalent representation of a single stage dynamic neural network with 'p' DNUs in parallel (in sigma mode).

Similarly, the input-output mapping of a single-stage of DNUs in pi (series or cascaded) configuration can be expressed as

$$u(k) = \left\{ \Psi^{(l)}\left[\left(w^{(l)}(k, \mathbf{a}^{1}_{ff}, \mathbf{b}^{1}_{fb})\, x^{(l)}(k)\right)\right]\right\} .. \left\{ \Psi^{(i)}\left[\left(w^{(i)}(k, \mathbf{a}^{1}_{ff}, \mathbf{b}^{1}_{fb})\, x^{(i)}(k)\right)\right]\right\} \ ...$$

$$\left\{ \Psi^{(p)}\left[\left(w^{(p)}(k, \mathbf{a}^{p}_{ff}, \mathbf{b}^{p}_{fb})\, x^{(p)}(k)\right)\right]\right\}$$

$$= \omega_1(x) \ ... \ \omega_i(x) \ ... \ \omega_p(x) = \prod_{n=1}^{p} \omega_n(x), \quad n = 1,..,i,..,p. \tag{4.3}$$

The dynamic neural system described by Eqns. (4.2) and (4.3) maps an n-dimensional input vector $s(k) \in \Re^n$ into a p-dimensional neural output vector $u(k) \in \Re^P$ where the mapping operations of an i-th neuron is given by

(i) The linear mapping operation:

$$v_i(k) = w^{(i)}(k, \mathbf{a}^{i}_{ff}, \mathbf{b}^{i}_{fb})\, s^{(i)}(k), \quad \text{and} \tag{4.4a}$$

(ii) The nonlinear mapping operation:

$$u_i(k) = \Psi^{(i)}\left[\left(v_i(k)\right)\right], \ i = 1, 2,.., p. \tag{4.4b}$$

A multi-stage dynamic neural network can be formed by cascading a group of single stage DNUs where the output of one stage provides the input to the subsequent stages. A three-stage dynamic neural network that is configured to have an input-stage, an intermediate-stage and an output-stage is shown in Fig. 4.2. Shaded circles in this figure denote a DNU. w_1, w_2 and w_3 are the input scaling factors. All the inter connections of the network are not shown. From this figure, the output u(k) of the dynamic neural network can be written as

$$u(k) = u_{31}(k) + u_{32}(k) + u_{33}(k) \tag{4.5}$$

where $u_{31}(k)$, $u_{32}(k)$ and $u_{33}(k)$ are the outputs of the dynamic neural units $w^{(31)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})$, $w^{(32)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})$ and $w^{(33)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})$ respectively, and are given by

$$u_{31}(k) = \Psi\left[w^{(31)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\left(u_{21}(k) + u_{22}(k) + u_{23}(k)\right)\right], \tag{4.6a}$$

$$u_{32}(k) = \Psi\left[w^{(32)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\left(u_{22}(k) + u_{21}(k) + u_{23}(k)\right)\right], \quad \text{and} \tag{4.6b}$$

$$u_{33}(k) = \Psi\left[w^{(33)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\left(u_{23}(k) + u_{22}(k) + u_{21}(k)\right)\right]. \qquad (4.6c)$$

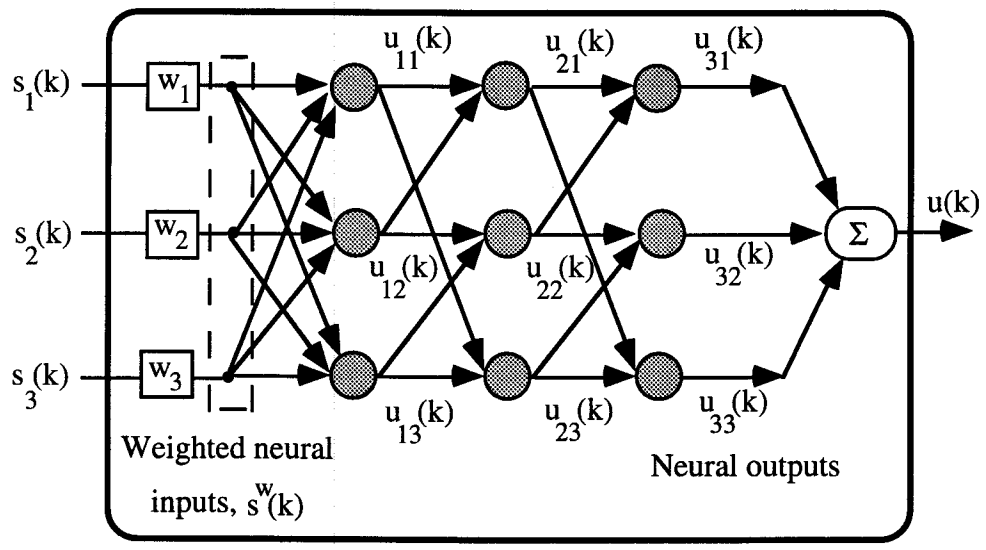


Figure 4.2: The structure of a three-stage dynamic neural network.

In Eqn. (4.6)

$$u_{21}(k) = \Psi\left[w^{(21)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\left(u_{11}(k) + u_{12}(k) + u_{13}(k)\right)\right]$$

$$= \Psi\left[w^{(21)}(.)\left[\Psi\left[w^{(11)}(.)\right] + \Psi\left[w^{(12)}(.)\right] + \Psi\left[w^{(13)}(.)\right]\right]\right] s^w(k) \qquad (4.7)$$

where $s^w(k)$ is the weighted input vector and is given by

$$s^w(k) = \left\{\begin{bmatrix} (s_1) \\ (s_2) \\ (s_3) \end{bmatrix}\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}\right\} = \mathbf{S}^T\mathbf{w}\,. \qquad (4.8)$$

In Eqn. (4.8), **S** and **w** are the vectors of the input signals and the scaling factors respectively.

Similarly, the output of each DNU may be calculated going from the output stage to the input stage. Combining these equations and substituting into (4.5) yields

$$u(k) = \Psi\left[w^{(31)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(32)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(33)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\right]$$

$$\Psi\left[w^{(21)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(22)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(23)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\right]$$

$$\Psi\left[w^{(11)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(12)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) + w^{(13)}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\right] s^{w}(k). \quad (4.9)$$

In Eqn. (4.9), the first term represents the output stage, the middle term the intermediate stage and the last term the input stage. The simplified diagram of the dynamic neural network is shown in Fig. 4.3.

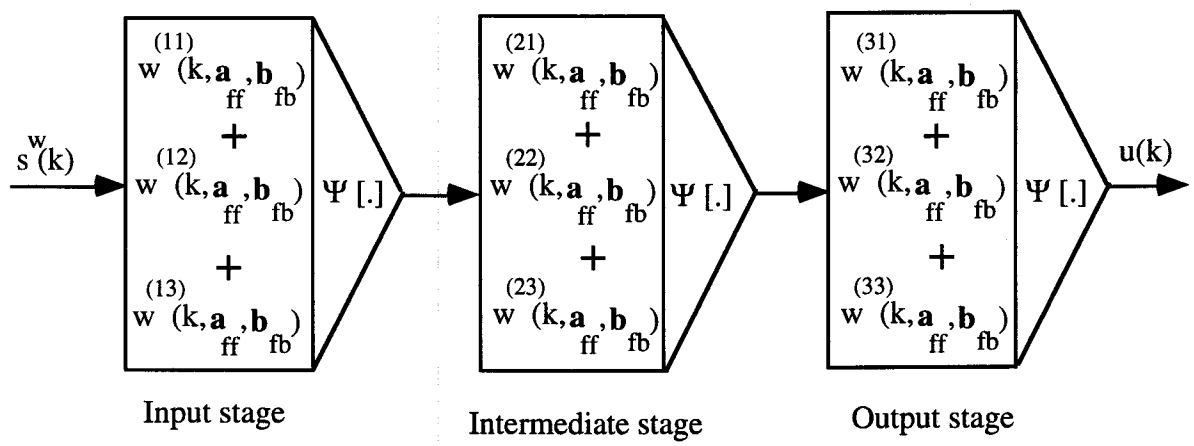


Figure 4.3: A modular representation of the dynamic neural network.

The output of the dynamic neural network with a fully connected neural structure with H stages, and N_h DNUs in each layer, where the output of the p DNU in layer h is connected to the input of the neuron r in the next layer, is

$$u_r^{h+1}(k) = \sum_{p=1}^{N_{h-1}} u_{(p,r)}^{h}(k) = \sum_{p=1}^{N_{h-1}} \left[\Psi^{h}\left[w_{(p,r)}^{h}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb})\, s_p^{h}(k)\right]\right] \quad (4.10)$$

where $1 \le p \le N_h$, $1 \le r \le N_{h+1}$, and $1 \le h \le H$.

4.2.2 Implementation

Figure 4.4 shows the implementation of a three-stage neural network and the corresponding parameter-state (sensitivity) model for the implementation of the learning and adaptive algorithm derived in Chapter 2.

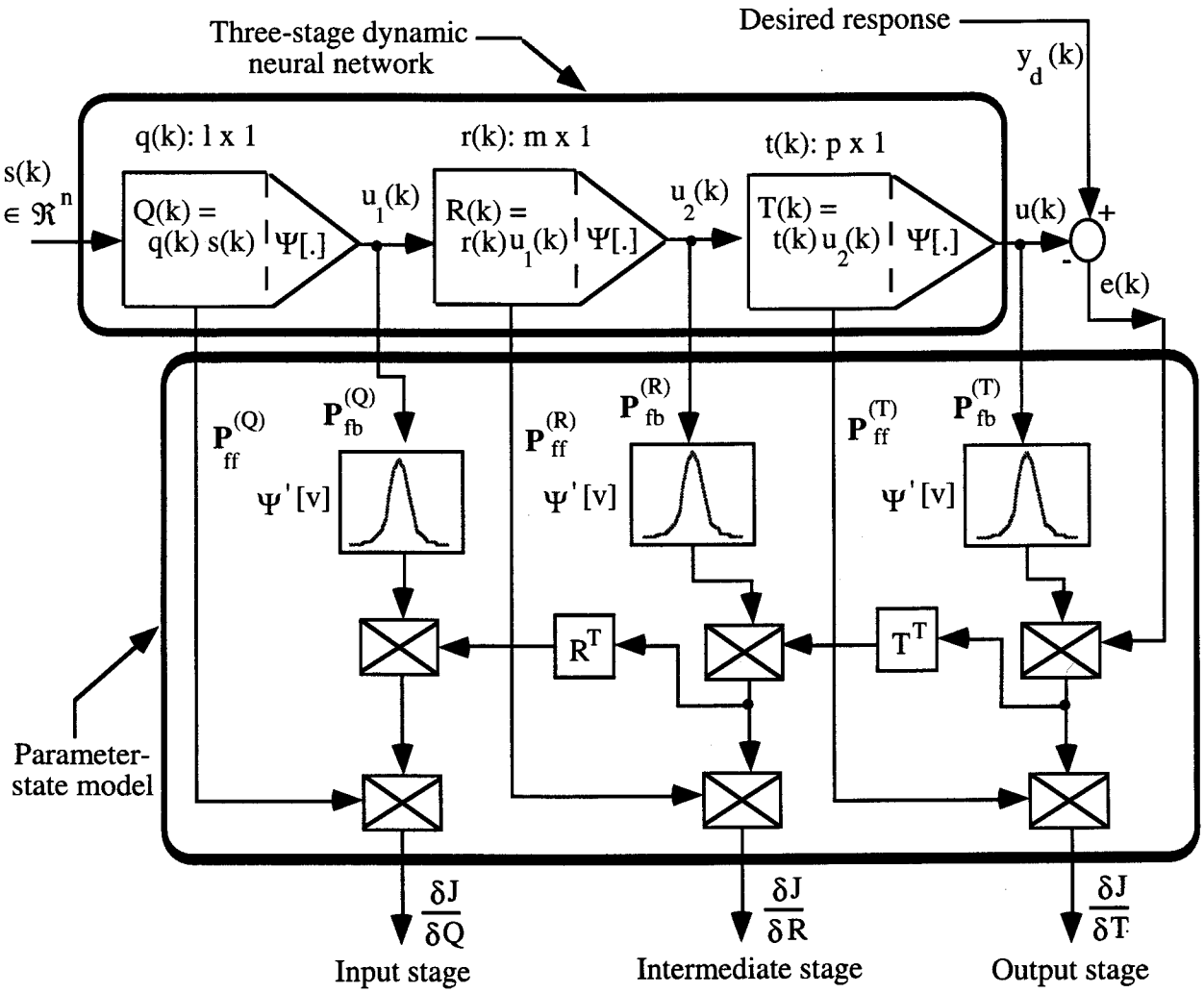


Figure 4.4: The implementation of a three-stage dynamic neural network and the parameter-state model.

As shown in Fig. 4.4, the neural network consists of the input (Q)-, intermediate (R)-, and output (T)-stages. $\mathbf{P}_{ff}^{(Q)}$, $\mathbf{P}_{ff}^{(R)}$, and $\mathbf{P}_{ff}^{(T)}$ represent the feedforward parameter-state signals of the Q, R, and S dynamic neural stages (layers) respectively. Similarly, $\mathbf{P}_{fb}^{(Q)}$, $\mathbf{P}_{fb}^{(R)}$, and $\mathbf{P}_{fb}^{(T)}$ represent the feedback parameter-state signals of the Q, R, and T dynamic neural stages

respectively. This multi-stage dynamic neural network is used to approximate arbitrary nonlinear functions and to synthesize a controller for nonlinear systems.

4.3 Neural Functional Approximation

One of the most significant characteristics of neural networks is their ability to approximate arbitrary nonlinear functions. This ability of neural networks has made them useful for modeling nonlinear systems which is of primary importance in the synthesis of nonlinear controllers [28]. Neural networks potentially offer a general framework for the modeling and control of nonlinear systems. The problem of learning a mapping between an input and an output space using neural networks is equivalent to the problem of estimating the system that transforms inputs and outputs given a set of examples of input - output pairs [67]. Training a neural network using the input-output data from a nonlinear dynamic system can be considered as a nonlinear functional approximation problem [28].

Recently, a number of researchers have shown that multi layer static (feedforward) neural networks can approximate arbitrary continuous functions to a desired degree of accuracy. Either Weierstrass's theorem or Kolmogorov's theorem has been employed for the theoretical development of functional approximation capabilities of neural networks. For example, it has been shown by Cybenko [68], Funahashi [69], Hornik et al [70], Cotter [71], and Blum and Li [72], based on Weierstrass's theorem, that a continuous function can be well approximated by a static neural network with one hidden layer, where each neuron in the hidden layer has a continuous sigmoidal nonlinearity. Gallant and White [73] showed that a static neural network with a single hidden layer using the monotone 'cosine squasher' is capable of embedding a Fourier network which yields a Fourier series approximation to a given function. Such networks thus possess all the approximation properties of a Fourier series representation. In particular, these networks are capable of approximation, to any degree of accuracy, of any square integrable function on a compact set using an infinite number of hidden units [70]. Cardaliaguet and Euvard [74] developed a noise-resistant approximation formula for a function and its derivative. They also addressed the limitations of neural network architecture on the accuracy of function approximation. Hecht-Nielsen [75], Cotter and Guillerman [76], and Kurkova [77] employed Kolmogorov's theorem to demonstrate the function approximation capabilities of static networks. However, it has been recently pointed out by Hornik et al [70], and Girosi and Poggio [78] that Kolmogorov's theorem requires a different nonlinear processing function for each unit in the network, and that functions in the second hidden layer depend upon the function being approximated. The problems associated with function approximation using static neural networks are addressed

in [28]. The use of dynamic networks to represent dynamic systems is of more significance and practical importance than using static neural networks.

The theory of functional approximation using static neural networks has been extensively studied as was briefly indicated above. As a result, static networks have been used to represent dynamic systems. A neural functional approximation theory for a dynamic neural structure described in the preceding section is developed in this thesis. It is shown in this section, using linear and trigonometric polynomials, that the proposed neural structure of DNUs can approximate arbitrary nonlinear functions [79]. Computer simulations are presented to demonstrate the functional approximation capabilities of this dynamic neural structure.

4.3.1 Theoretical Development

It is known that analytic functions can be approximated by means of a power series

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \tag{4.11}$$

which converges uniformly to the function f(x) in some interval [-a, a], a > 0. This means that if

$$S_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \ , \quad S(0) = a_0 \ , \tag{4.12}$$

then there exists a number N(ε) for ε > 0 such that the inequality n > N(ε) would imply the inequality $|f(x) - S_n(x)| < \varepsilon$, $-a \le x \le a$; that is, the polynomial $S_n(x)$ differs very little from the function f(x) if the degree n of the polynomial is sufficiently high. It is also known that Eqn. (4.12) implies (unlimited) differentiability of the function f(x) in the interval (- a < x < a), but any continuous function does not normally possess this property [80]. However, Weierstrass [80, 81, 82] has shown that any continuous function can be approximated by polynomials. The mathematical development presented in the following paragraphs is based on the functional analysis described in [80 - 83]. It is shown in this section, using linear and trigonometric polynomials, that the dynamic neural network with DNUs as the basic functional elements can approximate arbitrary functions. The neural models, both conventional (static) [61] and the DNU [79], can be operated in the linear range by appropriately choosing the sigmoidal slope. Therefore, the theory of linear operators can be extended to these neural structures.

Definition 1: The function $\Psi[.]$ is said to be a linear operator if it is both additive and homogeneous on **D** [83]. $\Psi[.]$ is said to be additive if $\Psi[x_1 + x_2] = \Psi[x_1] + \Psi[x_2]$, $(x_1, x_2 \in \mathbf{D})$, and homogeneous if

$$\Psi[\lambda x] = \lambda \Psi[x] \ , \ (x \in \mathbf{D} \ , \ \lambda \in C) \ .$$

A necessary condition for a linear function $\Psi[.]$ to be continuous on a space **D** is that Ψ be bounded on every bounded set. This condition is also sufficient if **D** satisfies the first axiom of countability [82].

Lemma 1: If $\Psi_H[1] \rightarrow 1$, $\Psi_H[x] \rightarrow p$, $\Psi_H[x^2] \rightarrow p^2$, then $\Psi_H[g] = 0$, where $g(x) = (x-p)^2$ where H is the number of stages (layers) in the dynamic neural structure.

Proof: $g(x) = (x-p)^2 = x^2 - 2\,p\,x + p^2$, it follows that

$$\Psi_H(g) = \Psi_H(x^2) - 2\,p\,\Psi_H(x) + p^2\,\Psi_H(1) \rightarrow p^2 - 2\,p\,p + p^2 = 0.$$

Theorem 1: If the two conditions: $\Psi_H[1] \rightarrow 1$, $\Psi_H[g] \rightarrow 0$ for $H \rightarrow \infty$

where $g(x) = (x-p)^2$, are satisfied for the sequence of linear positive functionals $\Psi_n[f]$, then

$$\lim_{H \rightarrow \infty} \Psi_H[f] = f(p)$$

for any function f(x) continuous at the point x = p and bounded on the real axis.

Proof: In view of the boundedness of the function f(x), $-M < f(x) < M$, where

$M(f) = \sup_x |f(x)|$, therefore

$$-2M < f(x) - f(p) < 2M, \ \forall x \in \text{on } \mathbf{D}. \tag{4.13}$$

In view of the continuity of this function at the point x = p

$$-\varepsilon < f(x) - f(p) < \varepsilon \ \text{ for } |x-p| < \delta. \tag{4.14}$$

Equations (4.13) and (4.14) imply the inequality

$$-\varepsilon - \left(\frac{2M}{\delta^2}\right) g(x) \ < f(x) - f(p) < \ \varepsilon + \left(\frac{2M}{\delta^2}\right) g(x) \ , \ \forall x \ . \tag{4.15}$$

In fact, if $|x-p| < \delta$, then (4.14) implies (4.15) since $g(x) = (x-p)^2 \geq 0$, and if $|x-p| \geq \delta$, then

$$\left(\frac{2M}{\delta^2}\right) g(x) \geq - \left(\frac{2M}{\delta^2}\right) \delta^2 = 2M,$$

and (4.15) follows from (4.13) since $\varepsilon > 0$. Using the inequality (4.15) and the fact that the linear positive functionals are monotonic gives

$$- \varepsilon \Psi_H[1] - \left(\frac{2M}{\delta^2}\right) \Psi_H[g] \leq \Psi_H[f] - f(p)\Psi_H[1] \leq \varepsilon \Psi_H[1] + \left(\frac{2M}{\delta^2}\right) \Psi_H[g] . \quad (4.16)$$

According to the conditions mentioned in the theorem, the right hand side of Eqn. (4.16) converges to ε, and the left hand side to $- \varepsilon$. Thus, there exists a number $N(\varepsilon)$ such that the inequality

$$- 2\varepsilon < \Psi_H[f] - f(p)\Psi_H[1] < 2\varepsilon \quad (4.17)$$

will be true $\forall\ H > N(\varepsilon)$. Since $\varepsilon > 0$ is arbitrary

$$\Psi_H[f] - f(p)\Psi_H[1] = \gamma_H \to 0. \quad (4.18)$$

Finally, since $\Psi_H[1] \to 1$

$$\Psi_H[f] = f(p)\Psi_H[1] + \gamma_H \to f(p).$$ Hence, the theorem is proved.

Lemma 2: If $\Psi_H[1] \to 1$, $\Psi_H(\cos x) \to \cos p$, $\Psi_H(\sin x) \to \sin p$, then

$$\Psi_H[g] = 0, \quad \text{where } g(x) = \sin^2\left\{\frac{(x-p)}{2}\right\}.$$

Proof : $g(x) = \sin^2\left\{\frac{(x-p)}{2}\right\} = \frac{1 - \cos(x-p)}{2} = \frac{1}{2} [\, 1 - \cos p \cos x - \sin p \sin x]$

$$\Psi_H(g) = \frac{1}{2} \left\{ \Psi_H(1) - \cos p\, \Psi_H(\cos x) - \sin p\, \Psi_H(\sin x) \right\}$$

$$\to \frac{1}{2} \left\{ 1 - \cos^2(p) - \sin^2(p) \right\} = 0.$$

Theorem 2: If the two conditions $\Psi_H[1] \to 1$, $\Psi_H(g) \to 0$ for $H \to \infty$

where $g(x) = \sin^2 \left\{\frac{(x-p)}{2}\right\}$ are satisfied for the sequence of linear positive functionals $\Psi_H[f]$, then $\lim_{H \to \infty} \Psi_H(f) = f(p)$ for any function f(x) with period 2π, continuous at the point x = p and bounded.

Proof: In so far as the function f(x) satisfies the conditions of Theorem 1, the inequalities

$$-2M < f(x) - f(p) < 2M, \ \forall\, x \in \text{on } \mathbf{D}, \text{ and} \tag{4.19}$$

$$-\varepsilon < f(x) - f(p) < \varepsilon \text{ for } |x-p| < \delta \tag{4.20}$$

are valid. Now consider a subinterval $(p - \delta) < x \leq (2\pi + p - \delta)$ of length 2π. The inequality

$$-\varepsilon - \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right) g(x) < f(x) - f(p) < \varepsilon + \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right) g(x), \ \forall\, x, \tag{4.21}$$

is valid in this subinterval. In fact, if $|x-p| < \delta$, then the inequality (4.21) follows from (4.20), since $g(x) = \sin^2 \left\{\frac{(x-p)}{2}\right\} \geq 0$.

If $\delta \leq x - p \leq 2\pi - \delta$, then $\frac{\delta}{2} \leq \frac{(x-p)}{2} \leq \pi - \frac{\delta}{2}$, and thus

$$\sin\frac{(x-p)}{2} \geq \sin\frac{\delta}{2}, \quad g(x) = \sin^2\left\{\frac{(x-p)}{2}\right\} \geq \sin^2\frac{\delta}{2}, \quad \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right) g(x) \geq 2M, \text{ and}$$

the inequality (4.21) follows from (4.19) since $\varepsilon > 0$. In order to prove the validity of inequality (4.21) $\forall$ x, the function $g(x) = \sin^2\left\{\frac{(x-p)}{2}\right\} = \frac{1 - \cos(x-p)}{2}$ has the period 2π and according to the conditions of the theorem the function f(x) also has this period; that is,

$$-\varepsilon - \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right) g(x+2k\pi) < f(x\ 2k\pi) - f(p) < \varepsilon + \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right) g(x+2k\pi), \ \forall\, x. \tag{4.22}$$

If x varies in the subinterval $(p - \delta, 2\pi + p - \delta)$, then $(x+ 2\pi)$ will vary in the subinterval $(2\pi + p - \delta, 4\pi + p - \delta)$, $(x+ 4\pi)$ in the subinterval $(4\pi + p - \delta, 6\pi + p - \delta)$, and in general $(x+ 2k\pi)$ in the subinterval $(2k\pi + p - \delta, 2k\pi+2\pi + p - \delta)$, $k = 0, \pm 1, \pm 2, \ldots$ The totality of

these subintervals covers without any gap the whole real axis, and thus the inequality (4.21), whose validity on every subinterval follows from (4.22), is proved for $\forall$ x.

The inequality (4.21) and the monotonic nature of the functions $\Psi_H(f)$ gives

$$-\varepsilon\,\Psi_H[1] - \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right)\Psi_H(g) < \Psi_H[f] - f(p)\,\Psi_H[1] < \varepsilon\,\Psi_H(1) + \left(\frac{2M}{\sin^2\frac{\delta}{2}}\right)\Psi_H[g]. \quad (4.23)$$

In view of the conditions, the right hand side of Eqn. (4.23) converges to ε, and the left hand side to $-\varepsilon$. Thus, there exists a number $N(\varepsilon)$ such that the inequality

$$-2\varepsilon < \Psi_H[f] - f(p)\,\Psi_H[1] < 2\varepsilon \quad (4.24)$$

will be true $\forall$ $H > N(\varepsilon)$. Since $\varepsilon > 0$ is arbitrary

$$\Psi_H[f] - f(p)\,\Psi_H[1] = \gamma_H \to 0,$$

$$\Psi_H[f] = f(p)\,\Psi_H[1] + \gamma_H \to f(p).$$ Hence, the theorem is proved.

Definition 3: Let f(x) and $\Psi[x]$ be two functions continuous in the interval [a,b] and let

$$d(f, \Psi) = \max_{a \le x \le b} \left| f(x) - \Psi[x] \right|.$$

The number $d(f, \Psi)$ is called the distance between the functions f(x) and $\Psi(x)$ [66] (deviation of the function f(x) from $\Psi(x)$).

Theorem 3: If the function f(x), continuous in the interval $[-\pi, \pi]$, is even, there exists an even trigonometric polynomial $\Psi_H(f, x)$ which deviates the least from the function f(x); that is, $d(\Psi, f) = \| f - \Psi_H(f, x) \|$.

Proof: If $\Psi_H(f, x)$ is any polynomial which deviates the least from the function f(x),

$$d(\Psi, f) = \| f - \Psi_H(f, x) \|. \quad (4.25)$$

Then, replacing x by -x and noting that the function f(x) is even gives

$$d(\Psi, f) = \max_{-\pi \le x \le \pi} \left| f(-x) - \Psi_H(f, -x) \right|$$

$$= \max_{-\pi \le x \le \pi} \left| f(x) - \Psi_H(f, -x) \right| = \| f - \Psi_H(f, -x) \|. \quad (4.26)$$

Let $Q(x) = \dfrac{\left[\Psi_H(f, x) + \Psi_H(f, -x)\right]}{2}$. The trigonometric polynomial Q(x) is even and has a degree not greater than n. Thus,

$$\| f - Q \| = \| \left(f - \frac{\left[\Psi_H(f, x) + \Psi_H(f, -x)\right]}{2} \right) \| = \left(\frac{1}{2}\right) \| f - \Psi_H(f, x) + f - \Psi_H[f, -x] \|$$

$$\leq \left(\frac{1}{2}\right) \| f - \Psi_H(f, x) \| + \left(\frac{1}{2}\right) \| f - \Psi_H(f, -x) \|$$

$$= \left(\frac{1}{2}\right) d(\Psi, f) + \left(\frac{1}{2}\right) d(\Psi, f) = d(\Psi, f). \qquad (4.27)$$

Hence, it follows that $d(\Psi, f) = \| f - Q \|$, the even polynomial Q(x) deviates the least from the function f(x).

4.3.2 Computer Simulation Studies

It is demonstrated in this section, through computer simulations, that the dynamic neural structure comprising of three stages, each stage having two DNUs as the basic computing nodes, can approximate arbitrary nonlinear functions. The learning scheme employed for this task is shown in Fig. 4.5.

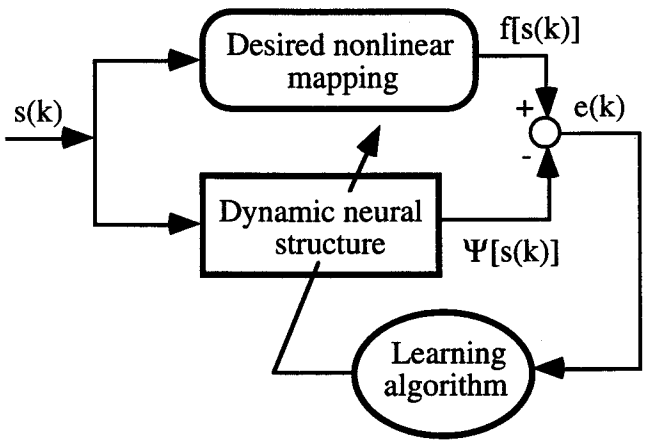


Figure 4.5: The learning scheme for functional approximation task using dynamic neural structure.

Four simulation examples are discussed in this section. Examples 1 and 2 demonstrate the neural network's ability to approximate arbitrary nonlinear functions as

shown in Figs. 4.6a and 4.6b. The nonlinear functions used in these examples were as follows:

Example 1: $f[s(k)] = s(k)$, and

Example 2: $f[s(k)] = 0.5\, s(k) + 0.1 \cos((2\pi k/1000)$.

In Example 3, the desired nonlinear function was changed during the learning process to study the adaptiveness of the neural network. The nonlinear functions used in this example were as follows:

$$f[s(k)] = \sin(2\pi k/250) \quad \text{for } 0 \le k < 500 \text{ and}$$

$$= \frac{0.5\, s(k)}{\sqrt{|\{1 + s^2(k)\}|}} \quad \text{for } 500 \le k < 1000.$$

The simulation results for this example are shown in Fig. 4.6c. In Example 4, another arbitrary nonlinear function of the form

$$f[s(k)] = s^3(k) + 0.3 \sin(2\pi s(k)) + 0.1 \sin(5\pi s(k)), \text{ where } s(k) = \sin(2\pi k/250)$$

was considered. As can be observed from Fig. 4.6d, the neural network could not approximate this function very well compared to the first three examples. The mean-square-error (MSE) of the function approximations shown in Figs. 4.6a. through 4.6d are respectively 0.163, 0.302, 0.266 and 1.348.

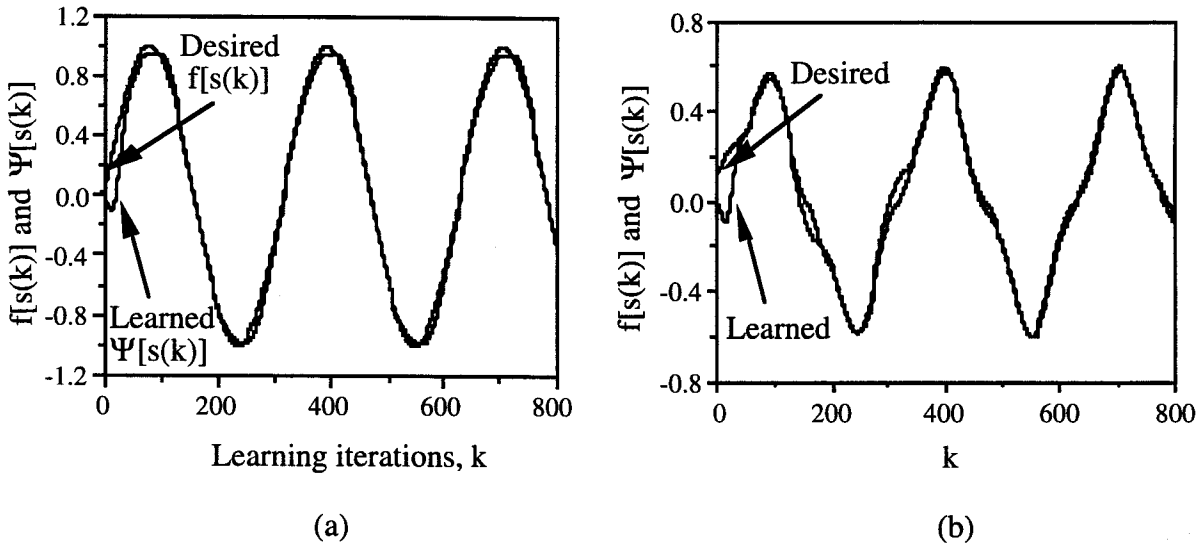


Figure 4.6: (Continued)

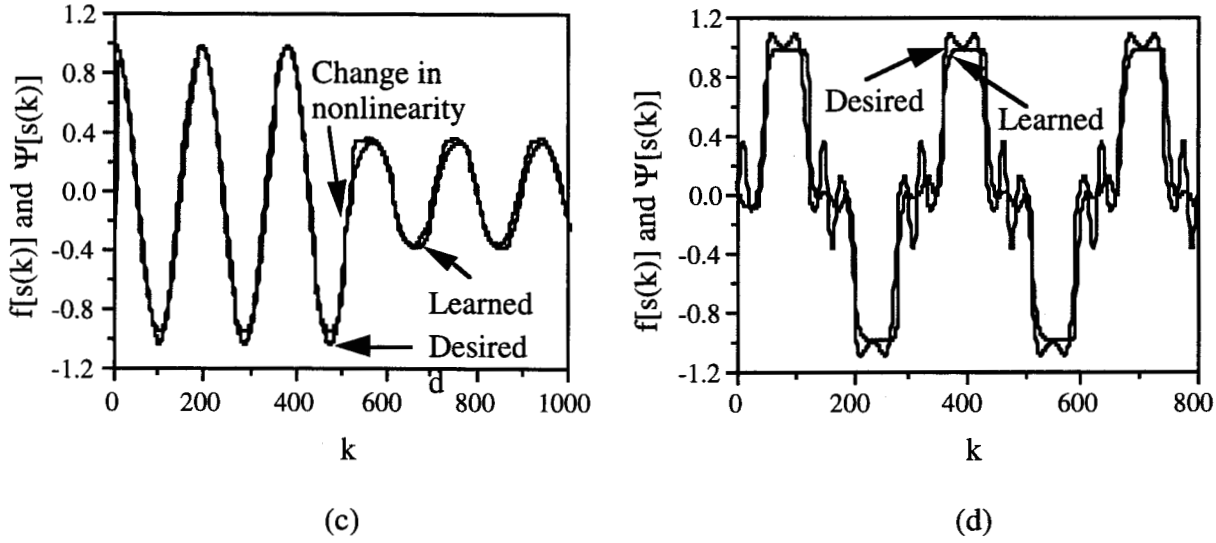


Figure 4.6: Arbitrary nonlinear functions and their approximations using dynamic neural network.

The above results do indicate that the proposed dynamic neural structure with DNU as the basic computing node can approximate arbitrary nonlinear functions. It was observed during the simulation studies that the neural network adapted to the changes in nonlinear functions, should they occur, during the approximation process. This approximation feature of the neural network is used in the on-line control of unknown nonlinear dynamic systems. This feature is discussed in the following section.

4.4 Control of Unknown Nonlinear Systems

One of the most significant features of neural networks is their ability to approximate arbitrary nonlinear functions. This ability of neural networks has made them useful for modelling nonlinear systems; this is of primary importance in the synthesis of nonlinear controllers. Static neural networks have been widely used for nonlinear system identification and control [34, 84 - 87]. It was demonstrated in the preceding section that the dynamic neural structure with the DNU as the functional element could approximate arbitrary nonlinear functions. In this section, it is demonstrated that this dynamic neural structure can be used for the control of unknown nonlinear dynamic systems.

Assume that a single-input-single-output (SISO) nonlinear discrete system is given in the form:

$q(k+1) = f[q(k), u(k)]$: State equation

$$y(k) = g[q(k)] \quad : \text{Output equation} \tag{4.28}$$

where $q \in \Re^n$ are the state variables, $u(k) \in \Re^1$ is the control input, f[.] and g[.] are the nonlinear maps on $\Re^n$, f[.] is bounded away from zero, and $y(k) \in \Re^1$ is the plant output.

The problem to be addressed in control systems is to find a control signal u(k) that will force the output y(k) to track asymptotically the desired output $y_d(k)$; that is,

$$\lim_{k \to \infty} \left[y_d(k) - y(k) \right] = 0 . \tag{4.29}$$

In order to achieve the above objective, the following assumptions about the nonlinear plant are required [34, 85, 88]:

Assumption 1: The plant is of relative degree one (that is, the input at k affects the output at k+1)

Assumption 2: For any $k \in [0, \infty]$ the desired output $y_d(k)$ and its n-derivatives $y_d^{(1)}(k)$, $y_d^{(2)}(k)$,..., $y_d^{(n)}(k)$, are uniformly bounded; that is,

$$\left| y_d^{(i)}(k) \right| \le m_i \ , \ i = 0, 1, 2, ..., n. \tag{4.30a}$$

Assumption 3: There exist coefficients $\mathbf{a}_{ff}$ and $\mathbf{b}_{fb}$ such that $\hat{f}[.]$ and $\hat{g}[.]$ are the approximations of the nonlinear functions f[.] and g[.] respectively with an accuracy ε on **D**, a compact subset of $\Re^n$; that is,

$$\max \left| f[.] - \hat{f}[.] \right| \le \varepsilon, \tag{4.30b}$$

$$\max \left| g[.] - \hat{g}[.] \right| \le \varepsilon, \quad \forall \, q \in \text{on } \mathbf{D}. \tag{4.30c}$$

4.4.1 Nonlinear Model Description

A nonlinear system may be represented by one of the four discrete-time models as suggested in [34]. These models can be described by the following difference equations:

Model I:

$$y(k+1) = \sum_{i=0}^{n-1} \alpha_i \ y(k-i) + g[u(k), u(k-1),..., u(k-m+1)], \tag{4.31a}$$

Model II:

$$y(k+1) = f[y(k), y(k-1),..., y(k-n+1)] + \sum_{j=0}^{m-1} \beta_j \; u(k-j), \tag{4.31b}$$

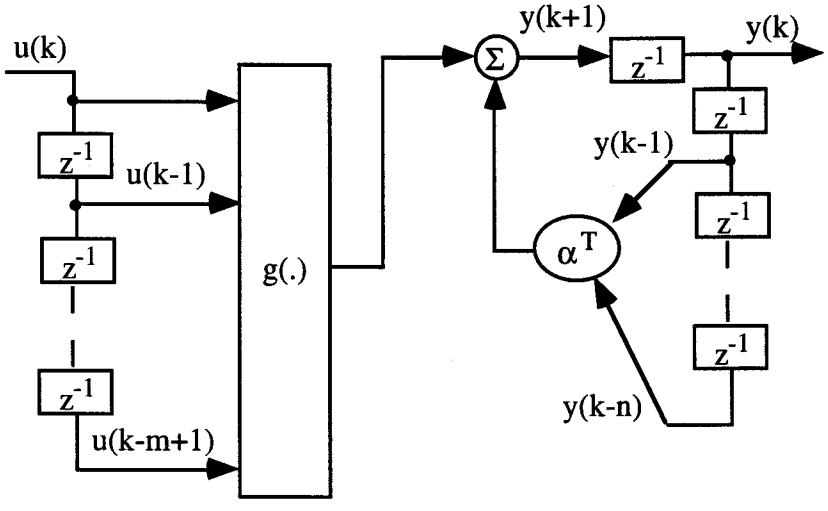
Model III:

$$y(k+1) = f[y(k), y(k-1),..., y(k-n+1)] + g[u(k), u(k-1),..., u(k-m+1)], \tag{4.31c}$$

Model IV:

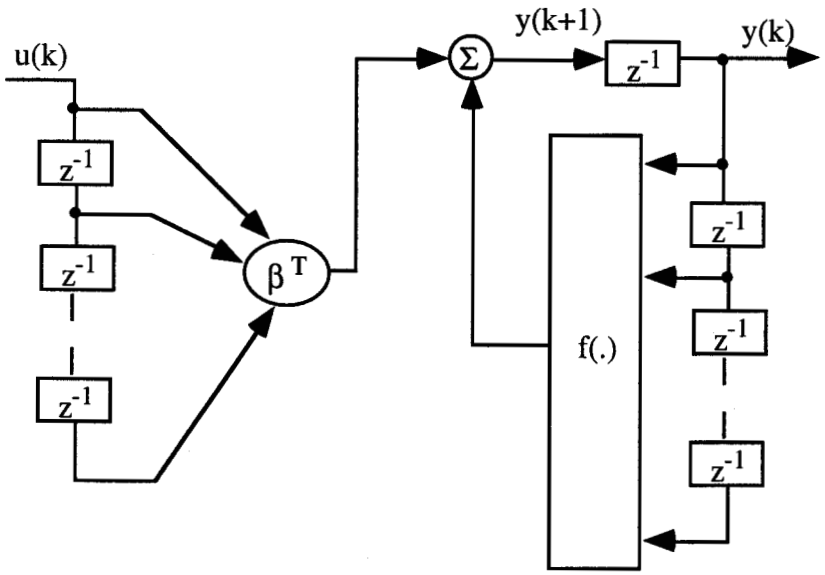
$$y(k+1) = f[y(k), y(k-1),..., y(k-n+1);\; u(k), u(k-1),..., u(k-m+1)] \tag{4.31d}$$

where [u(k), y(k)] represents the input - output pair of a SISO plant at time k, and $m \leq n$. In all four models, the output of the plant at time (k+1) depends both on its past n values of the output as well as the past m values of the input (output of the neural network). The functions f: $\Re^n \rightarrow \Re$ in models II and III, and f: $\Re^{n+m} \rightarrow \Re$ in Model IV and g: $\Re^m \rightarrow \Re$ in Models I and IV are assumed to be differentiable functions of their arguments. In general, f[.] and g[.] are nonlinear functions which may take different forms. In Model I, the plant output y(k+1) is a linear function of the past values y(k-i), while in Model II the relation between y(k+1) and the past values of the control input u(k-j) is assumed to be linear. In Model III, the nonlinear relation of y(k+1) with y(k-i) and u(k-j) is assumed to be separable, and Model IV in which y(k+1) is a nonlinear function of y(k-i) and u(k-j) subsumes Models I to III, and is analytically the least tractable. The block diagram representations of the four models are shown in Fig. 4.7.

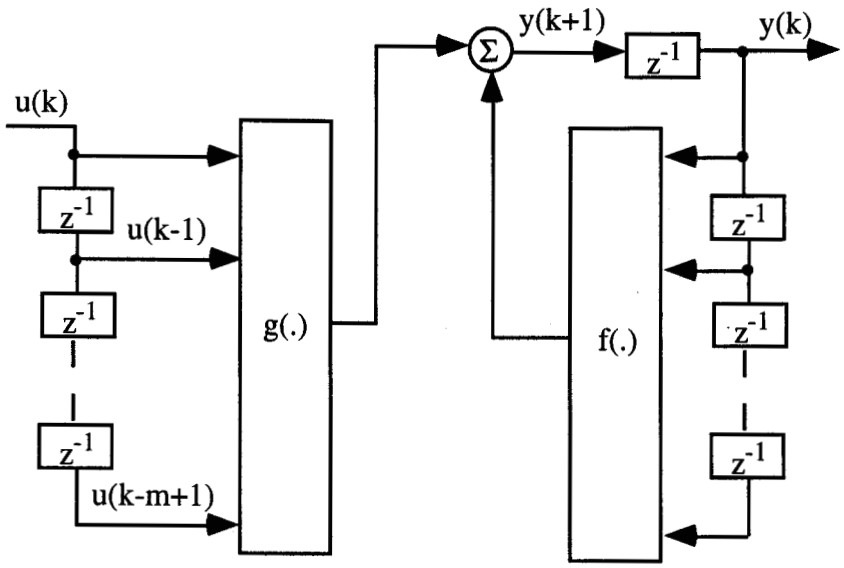


(a): Model I

Figure 4.7: (Continued)

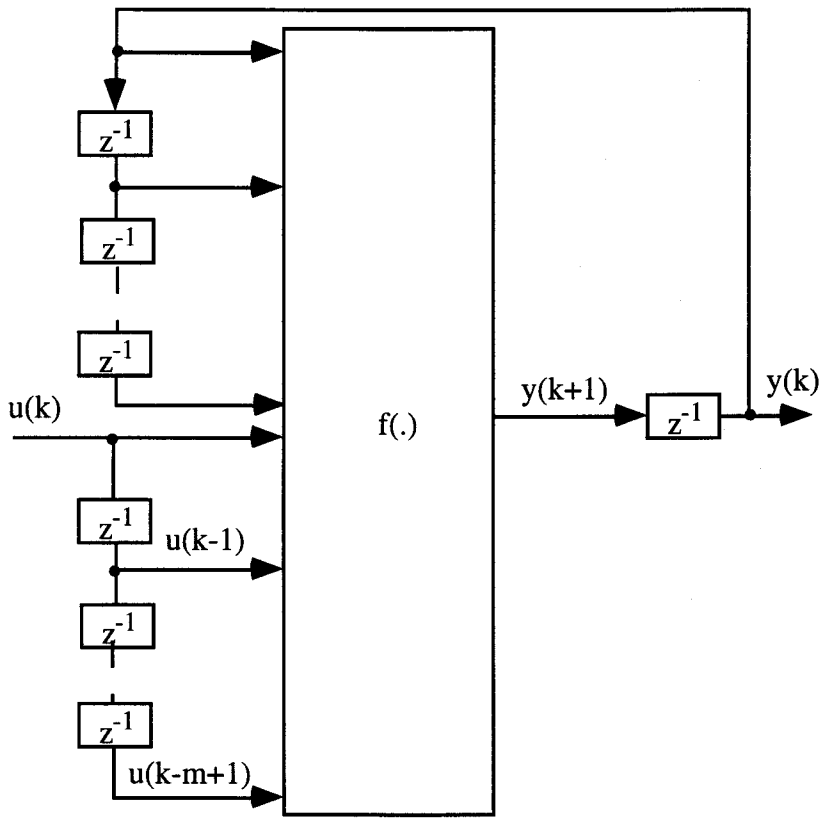


(b): Model II



(c): Model III

Figure 4.7: (Continued)



(d): Model IV

Figure 4.7: Representations of the SISO nonlinear plants: (a) Model I, (b) Model II, (c) Model III, (d) Model IV.

4.4.2 Computer Simulation Studies

In the computer simulation studies discussed in this section, a nonlinear dynamic system represented by one of the models shown in Fig. 4.7 was cascaded with the dynamic neural network presented in the earlier section. This control scheme is shown in Fig. 4.8. As depicted in this figure, the reference input s(k) is considered to be the target (desired) output $y_d(k)$ for the nonlinear system to track.

If the error is defined as $e(k) = y_d(k) - y(k)$, where $y_d(k)$ is the desired output and y(k) is the actual output of the plant under control, the aim of the control is to determine a bounded control input u(k) which results in the expression

$$\lim_{k \to \infty} \left[y_d(k) - y(k) = e(k) \right] = 0. \qquad (4.32)$$

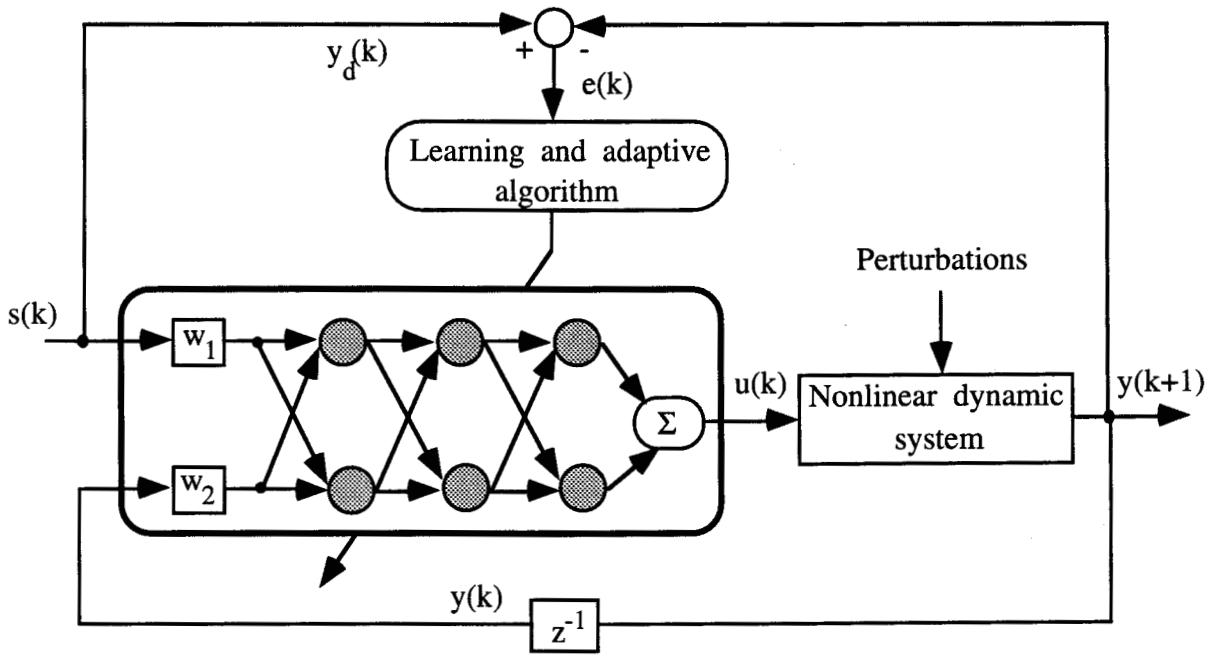


Figure 4.8: The control scheme for nonlinear dynamic systems using a three-stage dynamic neural structure.

The neural network used for the computer simulation studies consisted of three-stages with two DNUs in each stage. The input scaling factors w_1 and w_2 were set to 1 and -1 respectively. Six simulation examples are presented in this section each demonstrating a particular aspect of the control problem. In Example 1, a nonlinear dynamic plant governed by the difference equation (4.31a) was considered. The parameters of the plant, β_{ff} and α_{fb}, and the nonlinear function f[.] were assumed unknown. As the neural network weights were adjusted, the plant response followed the desired signal very closely. The adaptive capability of the control scheme for variations in the targeted output during the learning and control process was also demonstrated in this example. In Example 2, a study of the performance of the dynamic neural network was carried out for nonlinear systems represented by Model II. The ability of the neural network to adapt to the changing nonlinear characteristics in the system was also discussed in this example. The behavior of the dynamic neural network for nonlinear systems with the Model III configuration was investigated in Example 3. In this example it was shown that the neural control scheme would adapt to the changing input signal patterns and perturbations in the plant parameters. The control of a nonlinear plant represented by the Model IV configuration was studied in Example 4, including an important property of the neural network, called fault-tolerance. The performance of this neural network-based controller was compared with that of a PD controller. In conventional control

design based on feedforward neural networks, an optimal control law is often developed based on the model of the nonlinear plant. It was shown in Example 5 that this assumption was not necessary if the control scheme was designed based on the dynamic neural network approach. Finally, the effect of changing the slope of the nonlinear activation function was studied in Example 6. In these simulation studies, the error tolerance limits were set to $\pm$ 0.1.

Example 1: Model I, Equation (4.31a)

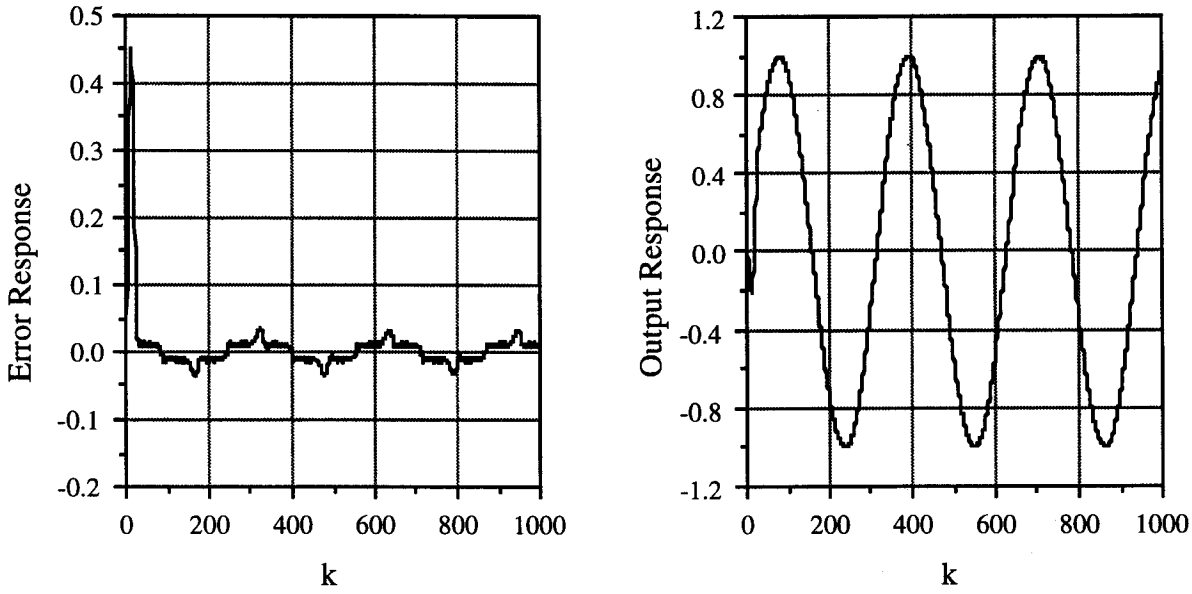
The plant to be controlled was governed by the difference equation

$$y(k+1) = \sum_{i=0}^{2} \alpha_i \, y(k-i) + g\left[\sum_{j=0}^{2} \beta_j \, u(k-j) \right] \tag{4.33}$$

where the unknown function was

$$g[.] = \sin(\pi u(k)) + 0.3 \sin(2\pi u(k-1)) + 0.1 \sin(5\pi u(k-2)),$$

and the plant parameters were $\beta_{ff} = [1.2, 1, 0.8]^T$ and $\alpha_{fb} = [1, 0.9, 0.7]^T$. The input to the system was $s(k) = \sin(2\pi k / 250)$ in the interval [-1, 1]. The error and output responses are shown in Fig. 4.9a. From the error response it can be seen that the error was initially large, but decreased very quickly to the tolerance limits. Also, the error was within the tolerance limits after about 1600 iterations even when the input was changed to be the sum of two sinusoids $s(k) = 0.8 \sin(2\pi k / 250) + 0.2 \sin(2\pi k / 25)$ at k = 500. The error and plant output responses to the change in input signal for 1000 learning iterations are shown in Fig. 4.9b.



(a)

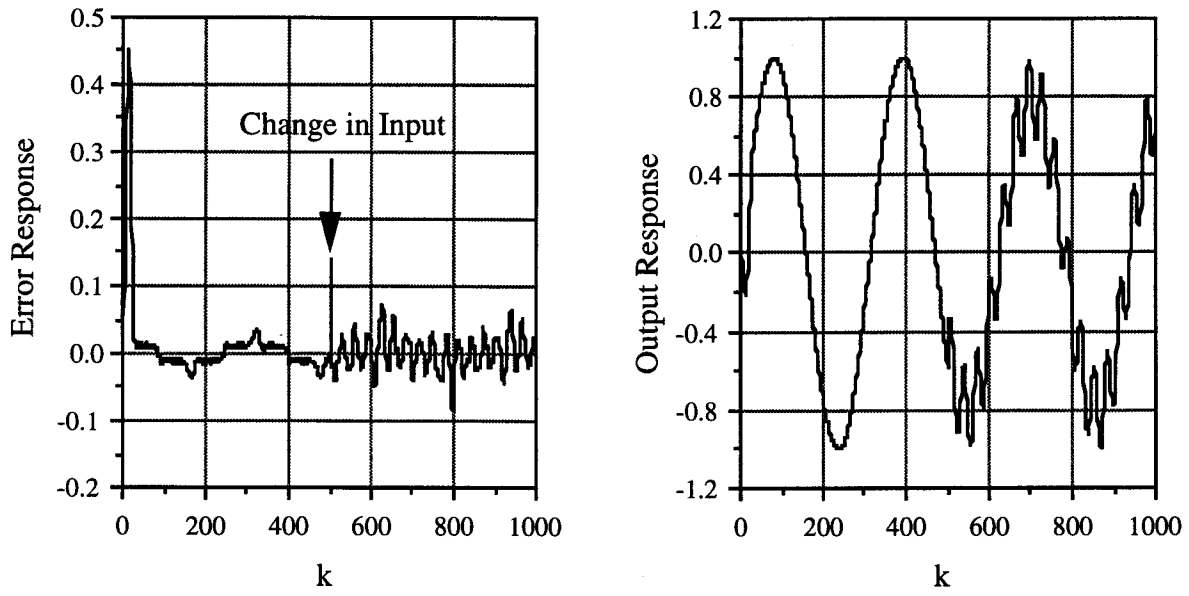


Figure 4.9: Simulation results, Example 1.

(a): The error and output responses of a nonlinear plant configured in Model I.

(b): The error and output responses to a change in input signal at k = 500.

Example 2: Model II, Equation (4.31b)

In this example a nonlinear plant represented by Eqn. (4.31b) was considered where the relation between y(k+1) and the past values of the control input u(k-j) was assumed to be linear, while y(k+1) was a nonlinear function of its past values, y(k-i). This plant model can be described by the following equation

$$y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i \, y(k-i)\right] + \sum_{j=0}^{2} \beta_j \, u(k-j) \, .$$

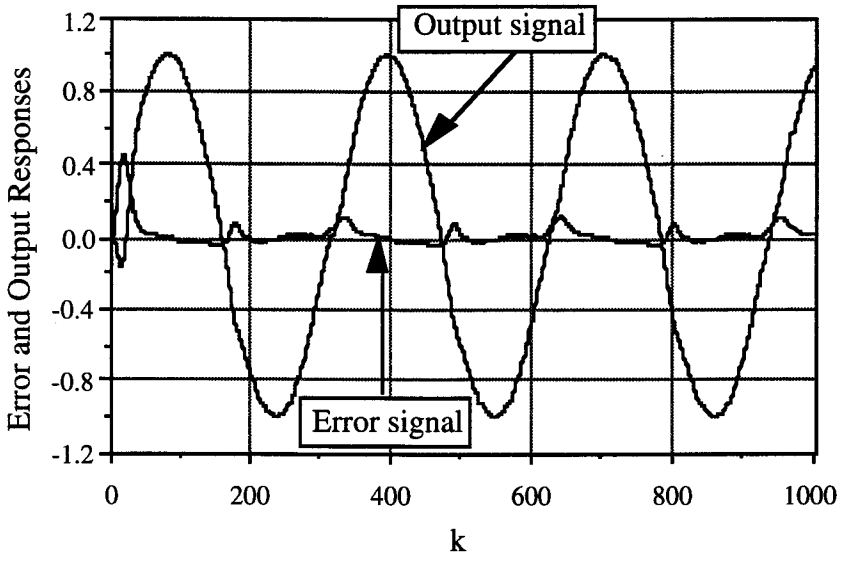
The plant parameters and the input signal were the same as in Example 1. The nonlinear function used in this simulation example was

$$f[.] = \frac{[y^2(k-1) + y^2(k-2)]}{(1+y^2(k))} \, . \qquad (4.34a)$$

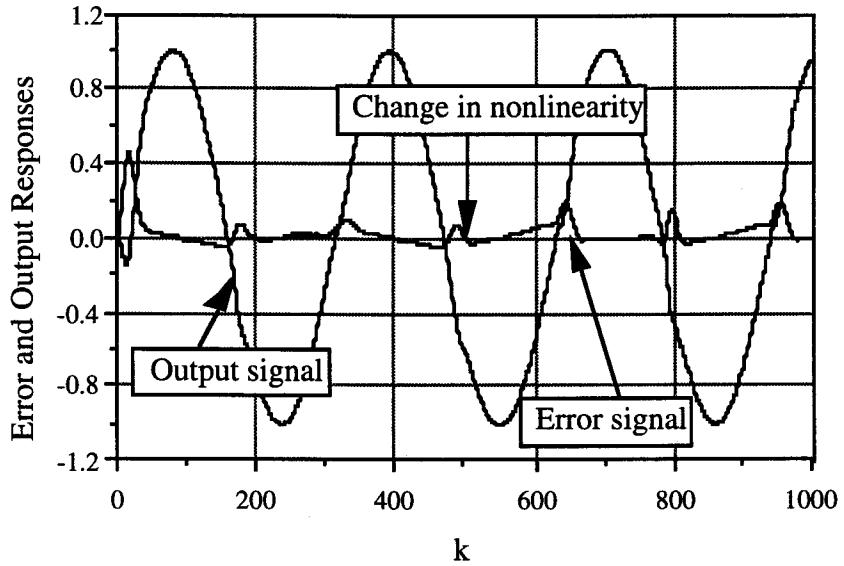
The error and output responses obtained for this simulation are shown in Fig. 4.10a. At k = 500, this nonlinear function, Eqn. 4.34a, was changed to

$$f[.] = \frac{0.2 \sin \pi \sqrt{|y^2(k)|}}{\left(1+y^2(k-1)\right)} \, . \qquad (4.34b)$$

The corresponding error and output responses are shown in Fig. 4.10b. As can be seen from the simulation results, the dynamic neural network was able to drive the plant towards the desired performance under the changed nonlinear characteristics.



(a)



(b)

Figure 4.10: Simulation results, Example 2.

(a): The error and output responses of a nonlinear plant in Model II configuration,

(b): The error and output responses when the nonlinear function f[.] was changed at k = 500.

Example 3: Model III, Equation (4.31c)

In this example, a nonlinear plant represented by the equation

$$y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i\, y(k-i) \right] + g\left[\sum_{j=0}^{2} \beta_j\, u(k-j) \right], \tag{4.35}$$

with the following nonlinear functions

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-y(k)}}{1 + y^2(k-1) + y^2(k-2)}, \text{ and} \tag{4.36a}$$

$$g[.] = \frac{\sqrt{\left|\left\{u^2(k) + u^2(k-1) + u^2(k-2)\right\}\right|}}{\left[1 + u^3(k)\right]}. \tag{4.36b}$$

was considered. The input to the system was $s(k) = \sin(2\pi k / 250)$ in the interval [-1, 1]. The plant parameters were the same as in Example 1. The simulation results, depicted in Fig. 4.11a, show the plant was able to follow the desired response with a MSE of 0.26.

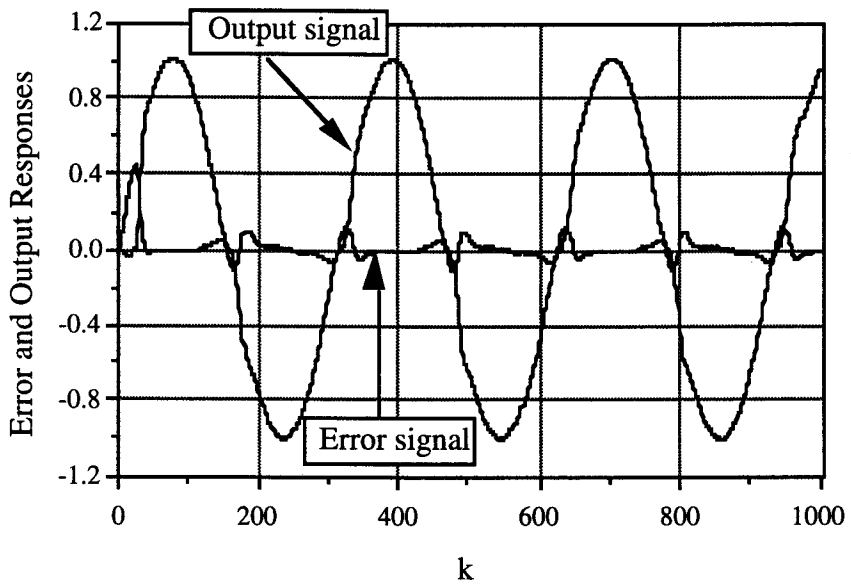


Figure 4.11: The error and output responses of a nonlinear plant configured in Model III, Example 3.

To study the performance of the control scheme under varying input signal and plant parameter perturbations, the input signal and plant parameters were changed as follows:

$s(k) = \sin(2\pi k / 250)$, for $0 \le k < 250$,
$s(k) = 1.0$, for $250 \le k < 500$,
$s(k) = -1.0$, for $500 \le k < 750$, and
$s(k) = 1.0$, for $750 \le k \le 1000$.

The plant parameters were

$\beta_{ff} = [1.2, 1, 0.8]^T$, $\alpha_{fb} = [1, 0.9, 0.7]^T$, for $0 \le k < 400$,
$\beta_{ff} = [1.2, 1, 0]^T$, $\alpha_{fb} = [1, 0, 0.7]^T$, for $400 \le k < 600$, and
$\beta_{ff} = [1.2, 1, 0.4]^T$, $\alpha_{fb} = [1, -0.5, 0.7]^T$, for $600 \le k \le 1000$.

The error and output responses for the above varying conditions are shown in Fig. 4.12.

It can be seen from these results that the effect of the variations of the input signal on the plant output was significant, while that of the plant parameter perturbations was not. This, of course, depended on the magnitude of perturbations. A very large magnitude of perturbation could drive the system unstable.

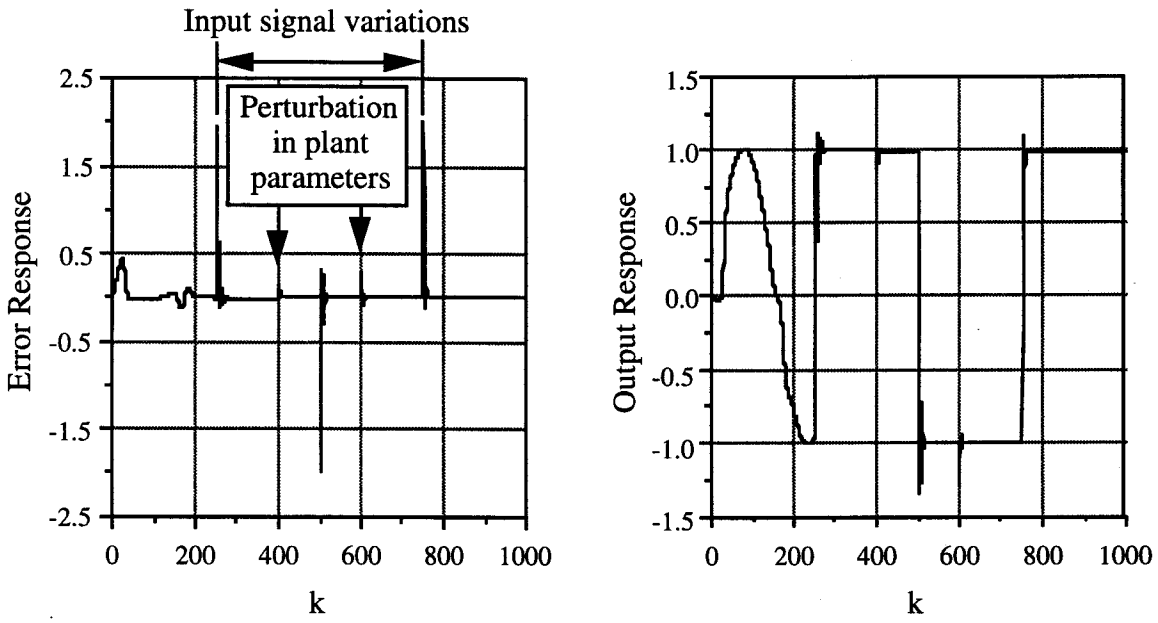


Figure 4.12: The error and output responses under different input signal and plant parameter variations, Example 3.

Example 4: Model IV, Equation (4.31d)

In this simulation example, a nonlinear plant of Model IV with the following nonlinear function

$$f[.] = \frac{\left[e^{(y^2(k-1)+y^2(k-2))} + \sqrt{|\{u^2(k) + u^2(k-1) + u^2(k-2)\}|} \right]}{\left[1+ u^3(k) \right]} \qquad (4.37)$$

and with the same plant parameters and input signal as in Examples 1 and 2 was considered. From the simulation results presented in Fig. 4.13, it is observed that the plant was able to follow the input signal with a MSE of 0.106 after 2000 iterations.

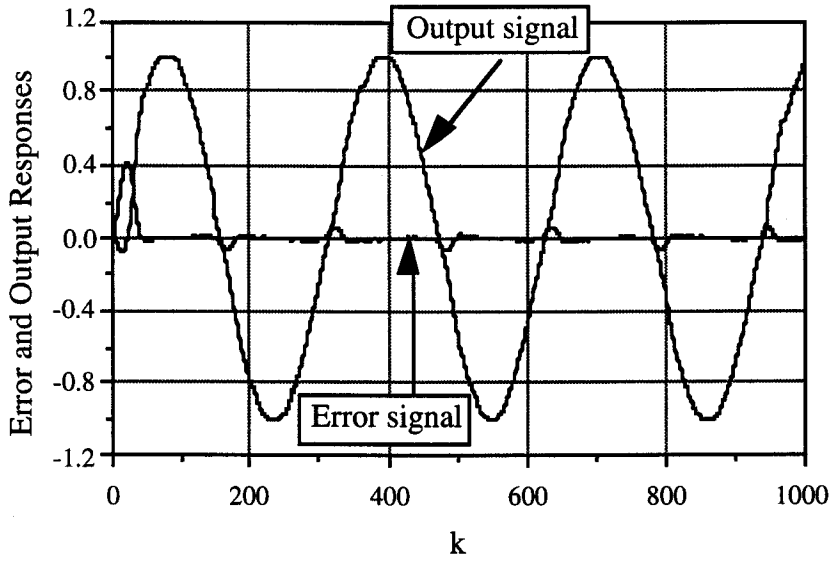


Figure 4.13: The error and output responses of a nonlinear plant configured in Model IV, Example 4.

The performance of this neural network-based controller was compared with a model-reference adaptive controller (MRAC) [14 - 16] shown in Fig. 4.14. The MRAC was originally proposed to solve a control problem in which the specifications are given in terms of a reference model that tells how the plant output ideally should respond to the command signal. The error e(k) is the difference between the outputs of the plant and the reference model. The parameter of the regulator, namely the feedforward gain, was adjusted based on the MIT rule [15]. The dynamic neural network, developed in this chapter, was used as the reference model. The plant was assumed to be represented by the Model IV configuration. The performance of the MRAC in terms of the error and output responses is shown in Fig.

4.15. The MSE, after 2000 iterations, was found to be 3.694. From this simulation study it may be observed that the performance of the dynamic neural network-based controller was much better than that of the MRAC.

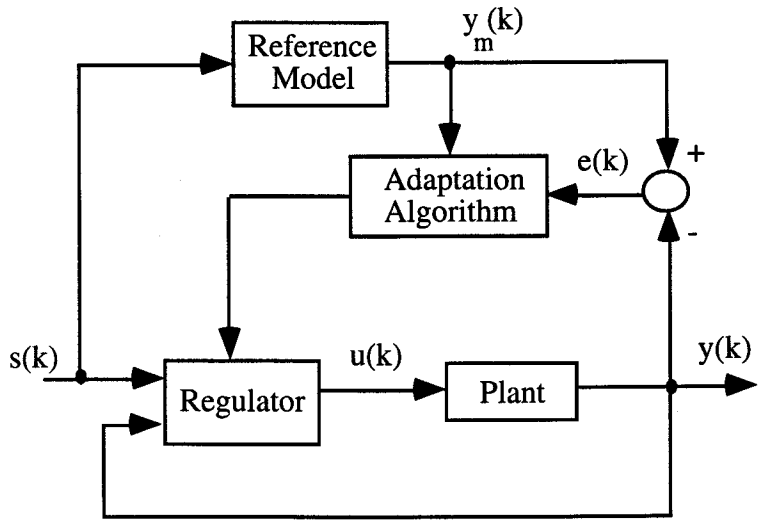


Figure 4.14: Block diagram of a model-reference adaptive controller (MRAC).

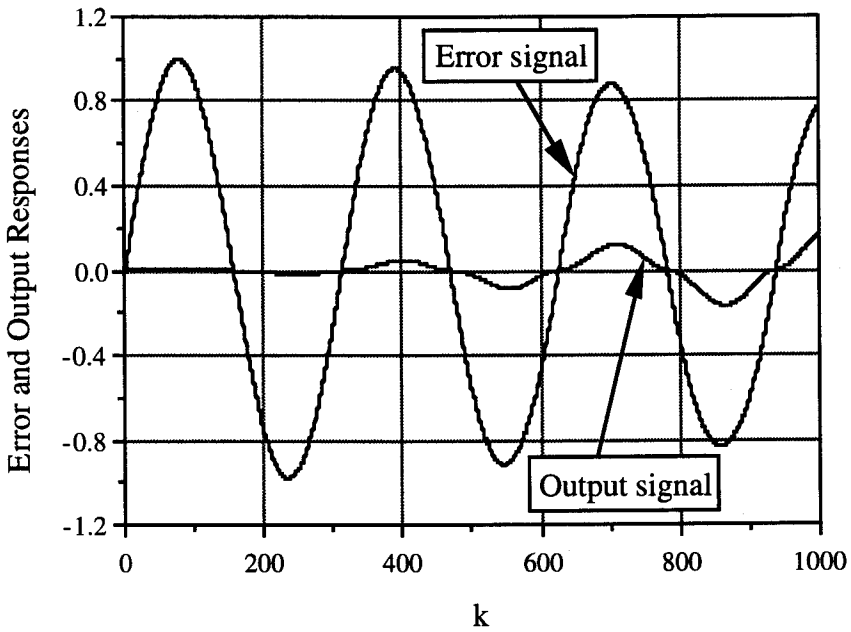


Figure 4.15: The error and output responses of a nonlinear plant configured in Model IV controlled by the model-reference adaptive controller (MRAC).

One of the main advantages of designing a controller based upon a neural network architecture is that the failures of a few neurons in the network do not cause significant effects on the overall system performance. This characteristic is called fault-tolerance. To demonstrate this feature, the intermediate stage of the dynamic neural network shown in Fig. 4.8 was removed, and the simulation was carried out. The corresponding error and output responses are shown in Fig. 4.16. The system responses were improved as the learning trials were increased, for example up to 2000. This simulation example shows that the neural network could control the plant with fewer neurons in the network, demonstrating the fault tolerance characteristic of the dynamic neural network.

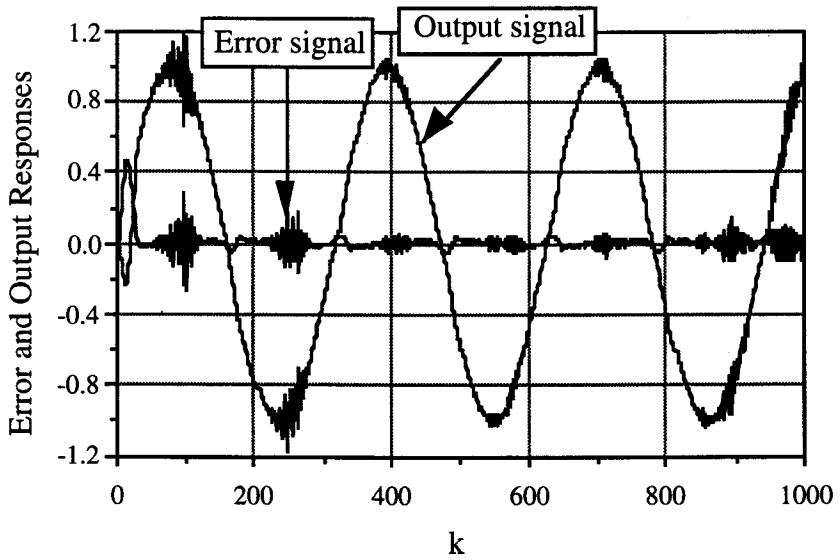


Figure 4.16: The error and output responses of a nonlinear plant with a neural network consisting of only input and output stages with two DNUs in each stage.

Example 5: Adaptation to system model representations

Although major advances have been made in the design of conventional adaptive controllers for linear systems with unknown parameters, such controllers can not provide a solution for a wide range of nonlinear control systems. The great diversity of nonlinear systems is the primary reason why no systematic and generally applicable theory for nonlinear control has yet evolved [28]. The existing control techniques for nonlinear systems, such as the phase plane, feedback linearization, and the describing functions, are system specific. In other words, a control methodology suitable for one class of nonlinear systems may be completely unacceptable for some other class of nonlinear systems. Since neural network-based control schemes exhibit learning and adaptive capabilities, the control law is independent of the plant configuration. This ability of the dynamic neural structure is

demonstrated in this example by changing plant models arbitrarily during the control process. The changes in plant configurations were made as follows:

$$\textbf{Model III: } y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i\, y(k-i)\right] + g\left[\sum_{j=0}^{2} \beta_j\, u(k-j)\right], \text{ for } 0 \le k < 250: \tag{4.38}$$

$$\text{where } f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1)+y^2(k-2)\right)\right\}\right] + e^{-y(k-1)}}{\left[1+y^2(k-1)+y^2(k-2)\right]}, \tag{4.39a}$$

$$g[.] = \frac{\sqrt{\left|\left\{u^2(k)+u^2(k-1)+u^2(k-2)\right\}\right|}}{\left[1+u^3(k)\right]}, \tag{4.39b}$$

$$\textbf{Model I: } y(k+1) = \sum_{i=0}^{2} \alpha_i\, y(k-i) + g\left[\sum_{j=0}^{2} \beta_j\, u(k-j)\right], \text{ for } 250 \le k < 750: \tag{4.40}$$

$$\text{where } g[.] = u^3(k) + 0.3\sin(2\pi u(k-1)) + 0.1\sin(5\pi u(k-2)), \tag{4.41}$$

$$\textbf{Model IV: } y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i\, y(k-i) + \sum_{j=0}^{2} \beta_j\, u(k-j)\right], \text{ for } 750 \le k < 1050: \tag{4.42}$$

$$\text{where } f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1)+y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{\left[1+u^2(k-1)+u^2(k-2)\right]}, \text{ and} \tag{4.43}$$

$$\textbf{Model II: } y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i\, y(k-i)\right] + \sum_{j=0}^{2} \beta_j\, u(k-j), \text{ for } 1050 \le k \le 1500 \tag{4.44}$$

$$\text{where } f[.] = \frac{0.1\sin\pi\sqrt{\left|y^2(k)\right|}}{\left[1+y^2(k-1)+y^2(k-2)\right]}. \tag{4.45}$$

The plant parameters were

$$\beta_{ff} = [1.2,\ 1,\ 0.8]^T, \quad \alpha_{fb} = [1,\ 0.9,\ 0.7]^T, \quad \text{for } 0 \le k \le 1500.$$

The input to the system is the same as in the earlier examples. The error and output responses obtained for this simulation study are shown in Fig. 4.17. It is observed from this figure that the neural network was able to adapt very quickly to the changing models of the nonlinear plant.

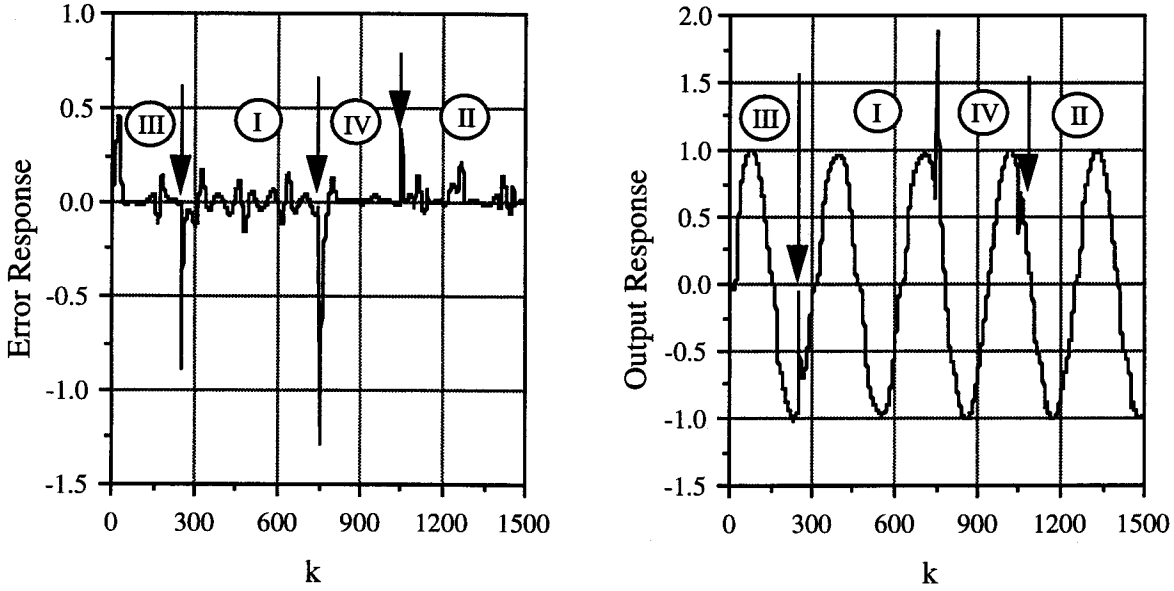


Figure 4.17: The error and output responses of a nonlinear plant with varying model representations of a nonlinear plant, Example 5.

Example 6: Effect of the slope of sigmoidal function on system performance

In the application of neural networks to control systems the slope of the nonlinear function, usually sigmoid, is determined by trial and error. This heuristic selection of the slope of the sigmoid function may limit the application of neural networks to complex systems involving nonlinear dynamics. An improper selection of the slope of the sigmoid function may lead to either unacceptable approximation of nonlinear functions or to system instability. There is no report of any systematic study of the effect of the slope of the nonlinear function on overall system performance. In this section, an attempt is made to study this effect by considering a general nonlinear dynamic plant for the various sigmoidal slopes.

The nonlinear plant considered in this example is of the type Model IV, and the nonlinear function and the plant parameters were respectively as follows:

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1)+y^2(k-2)\right)\right\}\right]+e^{-u(k)}}{1+u^2(k-1)+u^2(k-2)}, \text{ and}$$

$$\beta_{ff} = [1.2, 1, 0.8]^T, \quad \alpha_{fb} = [1, 0.9, 0.7]^T.$$

In this simulation study, the plant response was observed for different slopes of the sigmoidal function. As shown in Fig. 4.18, small increases in the slope significantly affected the plant response. A large increase in the slope resulted in an unstable response. The initial learning of the neural network was also found to depend on the slope. A small slope of the sigmoid function made the neural network respond slowly to the command input. Plant responses for four different values of the slope are shown Fig. 4.18. In the previous example it was shown that the neural network could adapt to nonlinear system models. However, the underlying assumption was that the slope of the nonlinear function had been chosen appropriately. An improper selection of the slope may result in an undesirable system response. In the examples discussed earlier in this chapter, the slope was kept constant at 0.4. A slight change in the slope value from 0.4 to 0.6 resulted in the output response shown in Fig. 4.18e. From this simulation example it is evident that the slope of the nonlinear function, in addition to the synaptic weights, determines the stability and convergence of the system performance.

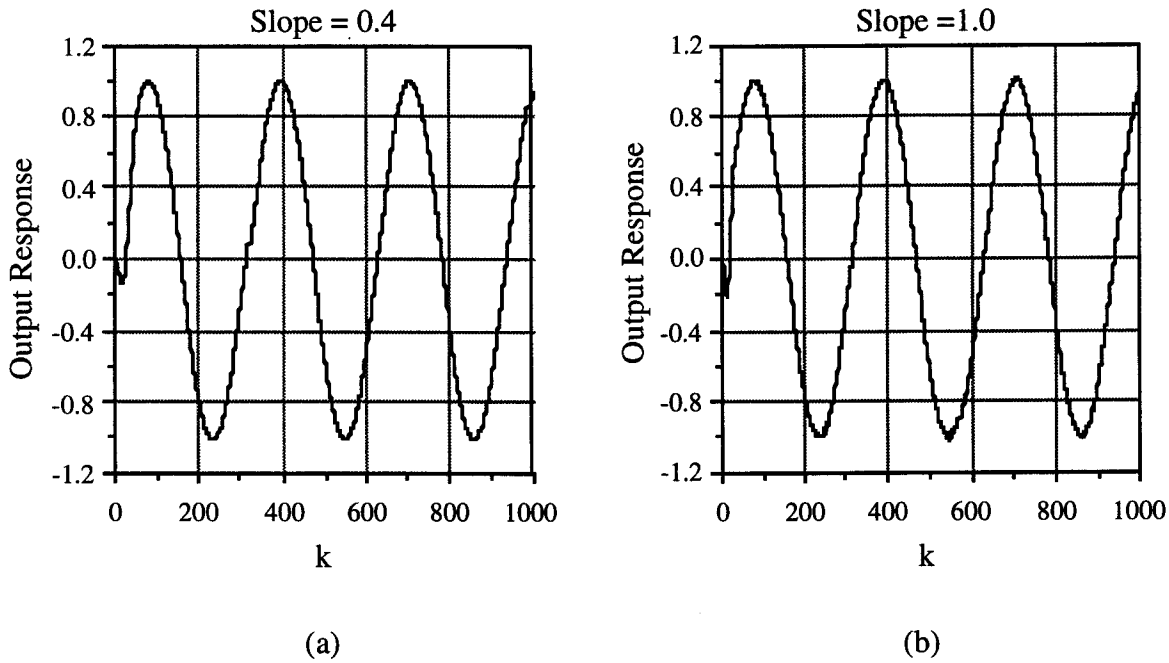
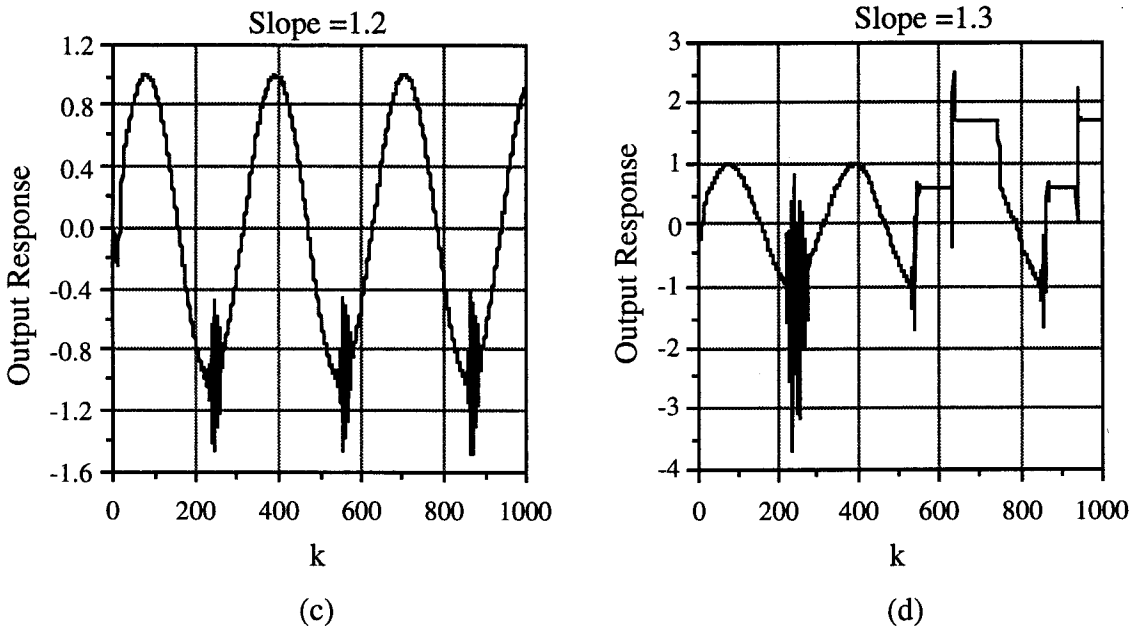


Figure 4.18: (Continued)



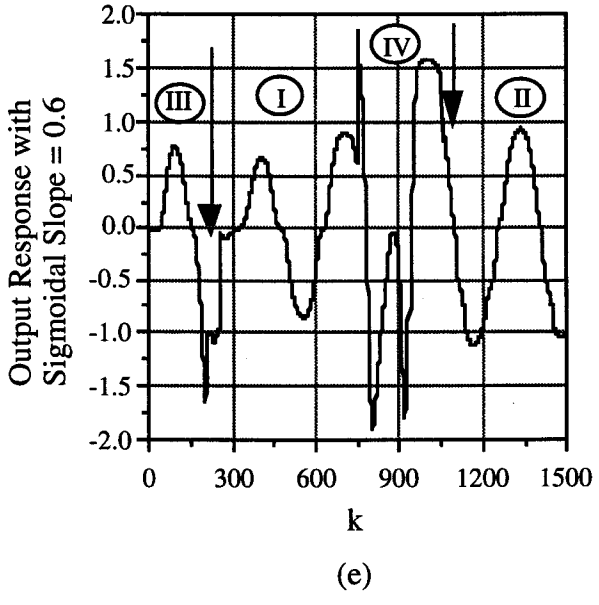


Figure 4.18: The output responses for different values of slope of the sigmoidal function, Example 6.

4.5 Summary

A multi-stage dynamic neural network with DNU as the basic computing element has been developed. An implementation for a three-stage neural network has been presented. The functional approximation theory for this dynamic neural structure was developed using the linear and trigonometric polynomials. This was substantiated by computer simulation studies. However, it is the author's opinion that the neural network with only feedforward connections may be approximated by the linear systems theory as was discussed in Section 4.2, which, in turn, limits the functional approximation capability. This was evident from Fig. 4.6d. To obtain better performance, it may be necessary to have feedback connctions between the neural layers. Four models of a nonlinear dynamic system are also discussed in this chapter. A three-stage dynamic neural network has been used to control adaptively a set of nonlinear plants. In Examples 1 to 4, the learning and adaptive capabilities of the dynamic neural network under varying conditions have been demonstrated. From Example 5 it is observed that the control technique based on the neural network approach is independent of the system representation. It was shown in Example 6 that the slope of the nonlinear activation function has a considerable effect on the performance of the system. An improper selection of this parameter may lead to instability. It was proposed in [89] that the slope of the nonlinear activation function could be considered as an adaptive parameter in addition to the neural weights. This is discussed in more detail in the next chapter.

5. Dynamic Neural Unit With Somatic Adaptation

5.1 Introduction

The optimum slope of a nonlinear activation operator, usually the sigmoidal function used to model the current conduction mechanism of the biological neuron, is determined by trial and error in conventional static neural structures. It was demonstrated in the preceding chapter that the performance of neural networks degrades considerably if the slope of the sigmoidal function is not chosen properly. The selection of the parameter that determines the slope of the nonlinear function needs more attention than what is presently given in the field of neural networks.

Toward this objective, Yamada and Yabuta [61] recently studied the effect of auto-tuning the slope of the sigmoid function on the performance of static neural networks with applications to linear and simple nonlinear systems. Independently, Gupta and Rao proposed [39, 89] that the parameter which controls the slope of the nonlinear function can be considered as one of the adjustable parameters of the neural structure in addition to the synaptic weights. This component contributes to what is generally referred to as *somatic adaptation*. The purpose of this chapter is to develop a dynamic neural structure with somatic adaptation, and to examine briefly how it affects the neural network performance as applied to the control of unknown nonlinear dynamic systems.

This chapter is organized as follows: The biological basis for somatic adaptation is briefly described in Section 5.2. The modified DNU architecture and the algorithm to modify parameters of the DNU are developed in Section 5.3. The implementation scheme of the modified algorithm is also presented in this section. A three-stage dynamic neural network, using the DNU as the basic computing element, is developed in Section 5.4. Computer simulation studies for nonlinear dynamic systems are presented in Section 5.5. The concluding remarks are given in Section 5.6.

5.2 Biological Basis for Somatic Adaptation

The biological neuron is currently understood to provide two distinct mathematical operations distributed over the synapse, the junction point between an axon and the dendrite, and the soma, the main body of the neuron [20, 31]. These two neuronal mathematical operations are called respectively the synaptic operation and the somatic operation [39]. From the biological point of view, these two operations are physically separate, but in the modeling of a biological neuron, these operations have been combined [20] (for example, thresholding in the soma is transferred to the synaptic operation).

At the macroscopic level, the dendrites of each neuron in the biological neural network receive pulses at the synapses and convert them to a variable dendritic current. The flow of the current through the axon membrane subsequently modulates the axonal firing rate. For each neuron there is a time-varying nonlinear relationship between the pulse rate at the synapse and the amplitude of the dendritic current [90]. This leads to a plausible inference that the main body of the neuron, the soma, may also change during neural activities, such as learning, adaptation, and vision perception. This morphological change of the neuron during the learning process may be modeled by considering the slope of the nonlinear function in a neural network as one of the adaptable parameters in addition to the synaptic weights [39, 61]. This component of neuronal learning and adaptation is called the *somatic adaptation* [89].

A sigmoidal function has been used in this thesis as the nonlinear activation function in the architecture of the DNU. Mathematically, a time-varying sigmoid function can be expressed as

$$\Psi[.] = \frac{\exp\left(g_s v_1(k)\right) - \exp\left(-g_s v_1(k)\right)}{\exp\left(g_s v_1(k)\right) + \exp\left(-g_s v_1(k)\right)} = \tanh\left[g_s v_1(k)\right] = \tanh\left[v(k)\right] \quad (5.1)$$

where $v(k) = g_s v_1(k)$. Figure 5.1 shows $\Psi[.]$ and its derivative $\Psi'[.]$ which provides the axonal gain for different values of the slope.

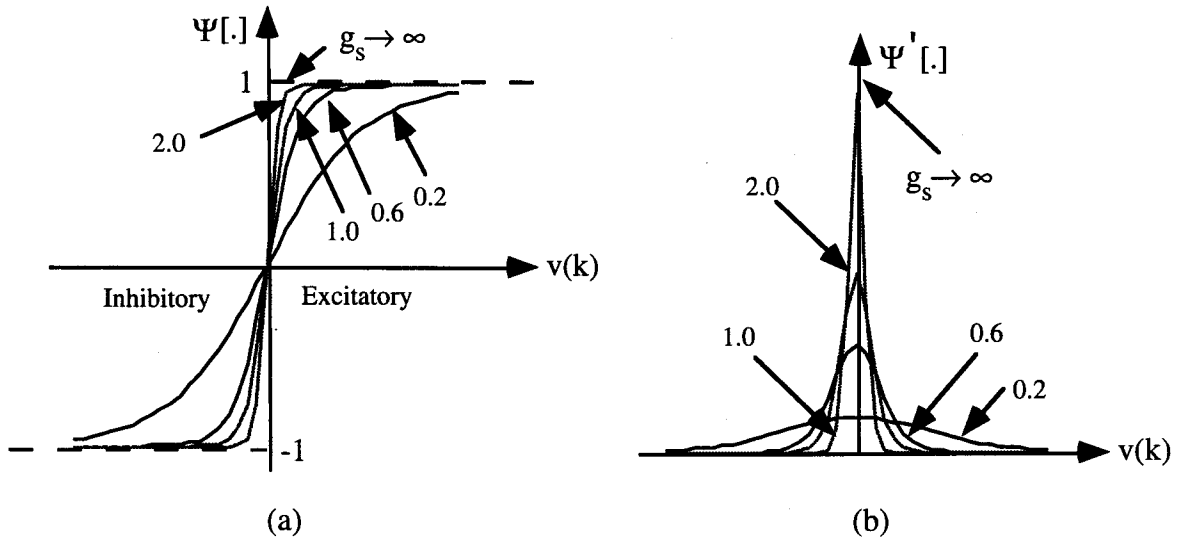


Figure 5.1: Sigmoidal activation function and its derivative.

(a) Sigmoid function $\Psi[v(k)] = \tanh [g_s v_1(k)]$, for different values of g_s,

(b) The derivative, $\Psi'[v(k)]$, of the sigmoidal function. This function tends to become a sign function as $g_s \to \infty$, that is, $\tanh [g_s v_1(k)]|_{g_s \to \infty}$. The slope $\Psi'[v](k)$ tends to become very narrow with an increasing value of g_s.

5.3 Modified Structure of Dynamic Neural Unit

5.3.1 Architectural Details

The DNU introduced in Chapter 2 accounts for only the synaptic component of the neuronal learning process. The modified DNU structure consisting of both the synaptic and somatic components is shown in Fig. 5.2.

The neural dynamics of the DNU can be expressed in the form of a transfer relation as

$$w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) = \frac{v_1(k)}{s(k)} = \frac{[a_0 + a_1 z^{-1} + a_2 z^{-2}]}{[1 + b_1 z^{-1} + b_2 z^{-2}]} \qquad (5.2a)$$

where $s(k) = \left[\sum_{i=1}^{n} w_i s_i - \theta \right]$ is the neural input to the DNU, $s_i \in \Re^n$ are the inputs from other neurons or from sensors, $w_i \in \Re^n$ are the corresponding input weights, θ is an internal threshold, $v_1(k) \in \Re^1$ is the output of the dynamic structure, $u(k) \in \Re^1$ is the neural output, and $\mathbf{a}_{ff} = [a_0, a_1, a_2]^T$ and $\mathbf{b}_{fb} = [b_1, b_2]^T$ are the vectors of adaptable feedforward and

feedback weights respectively. Alternatively, Eqn. (5.2a) may be described by the following difference equation

$$v_1(k) = - b_1\, v_1(k-1) - b_2\, v_1(k-2) + a_0\, s(k) + a_1\, s(k-1) + a_2\, s(k-2)\,. \qquad (5.2b)$$

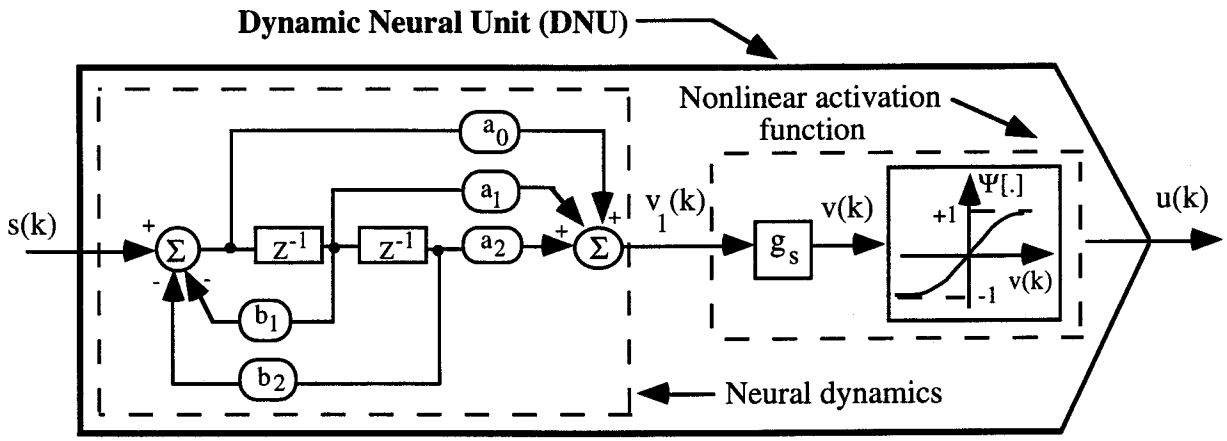


Figure 5.2: The modified DNU structure with variable slope of the sigmoid function.

The vectors of the input signals and adaptable weights of the modified DNU are redefined as

$$\Gamma(k,v_1, s) = \left[v_1(k-1)\;\; v_1(k-2)\;\;\; s(k)\;\; s(k-1)\;\; s(k-2)\right]^T, \qquad (5.3)$$

and

$$\Phi_{(\mathbf{a}_{ff},\mathbf{b}_{fb})} = \left[-b_1\;\; -b_2\;\; a_0\;\; a_1\;\; a_2\right]^T. \qquad (5.4)$$

Using (5.3) and (5.4), Eqn. (5.2b) is rewritten as

$$v_1(k) = \Phi^T_{(\mathbf{a}_{ff},\mathbf{b}_{fb})}\, \Gamma(k,v_1, s) = \left[-b_1\;\; -b_2\;\; a_0\;\; a_1\;\; a_2\right] \begin{bmatrix} v_1(k-1) \\ v_1(k-2) \\ s(k) \\ s(k-1) \\ s(k-2) \end{bmatrix}. \qquad (5.5)$$

The nonlinear mapping operation on $v_1(k)$ yields a neural output u(k) given by

$$u(k) = \Psi\left[g_s\, v_1(k)\right]. \qquad (5.6)$$

5.3.2 The Modified Learning and Adaptive Algorithm

The algorithm to modify the synaptic (feedforward and feedback) weights of the DNU was derived in Chapter 2. In this section, a brief description of the modified algorithm that accounts for both the synaptic and somatic adaptations is presented.

The feedforward parameters a_{ff_i}, i = 0, 1, 2, and the feedback parameters b_{fb_j}, j = 1, 2, are modified based on the following set of equations (derived in Chapter 2)

$$a_{ff_i}(k+1) = a_{ff_i}(k) + \mu_{a_i} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,\mathbf{P}_{ff_i}(k)\right], \quad i = 0,1,2, \tag{5.7a}$$

and

$$b_{fb_j}(k+1) = b_{fb_j}(k) + \mu_{b_j} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,\mathbf{P}_{fb_j}(k)\right], \quad j = 1,2 \tag{5.7b}$$

where the modified parameter-state signals for the feedforward and the feedback weights are given by the relations

$$\mathbf{P}_{ff_i}(k) = g_s\,[s(k-i)], \quad i = 0, 1, 2, \quad \text{and} \tag{5.8a}$$

$$\mathbf{P}_{fb_j}(k) = -\,g_s\,[v_1(k-j)], \quad j = 1,2 \quad \text{(See Appendix I for proof).} \tag{5.8b}$$

The modified parameter-state signals may be derived from the DNU structure as shown in Fig. 5.3.

Similarly, the other adjustable parameter of the DNU, namely the somatic gain g_s of the activation function, may be modified as follows:

$$g_s(k+1) = g_s(k) - \mu_{g_s} \frac{\partial J(\Phi)}{\partial g_s(k)} \tag{5.9}$$

where μ_{g_s} is the adaptive gain. The gradient of the performance index with respect to the somatic gain g_s is given by

$$\frac{\partial J(\Phi)}{\partial g_s} = \frac{1}{2} E\left[\frac{\partial [y_d(k) - u(k)]^2}{\partial g_s}\right] = E\left\{-e(k)\left[\frac{\partial \Psi(v)}{\partial v}\,\frac{\partial v}{\partial g_s}\right]\right\}$$

$$= E\left\{-e(k)\left[\mathrm{sech}^2[v(k)]\,\frac{\partial v}{\partial g_s}\right]\right\} = E\left\{-e(k)\left[\mathrm{sech}^2[v(k)]\;v_1(k)\right]\right\}. \tag{5.10}$$

Therefore, from Eqn. (5.9), the following equation may be written

$$g_s(k+1) = g_s(k) + \mu_{g_s} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,v_1(k)\right] . \qquad (5.11)$$

The modified DNU symbol and the implementation scheme of the modified algorithm are shown in Figs 5.4a and 5.4b respectively.

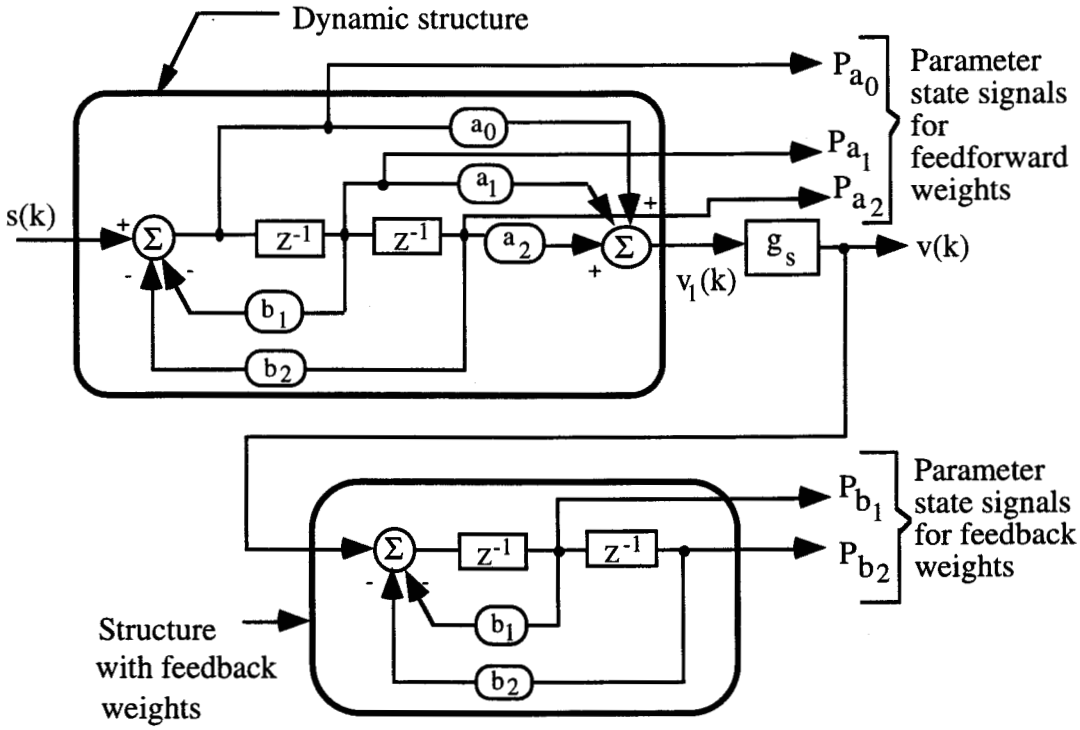


Figure 5.3: Generation of parameter-state signals from the modified structure of DNU.

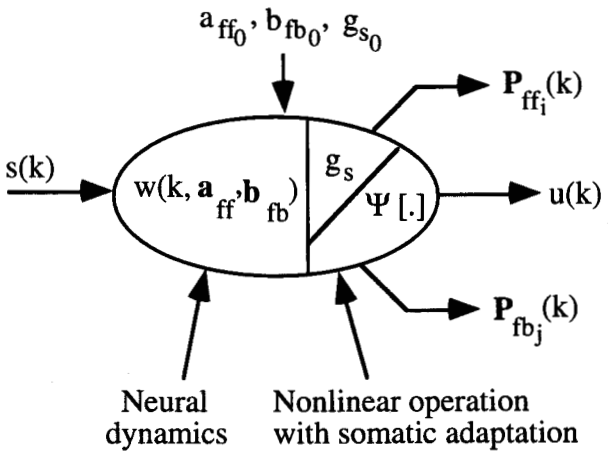


Figure 5.4a: Symbolic representation of the DNU with both synaptic and somatic components.

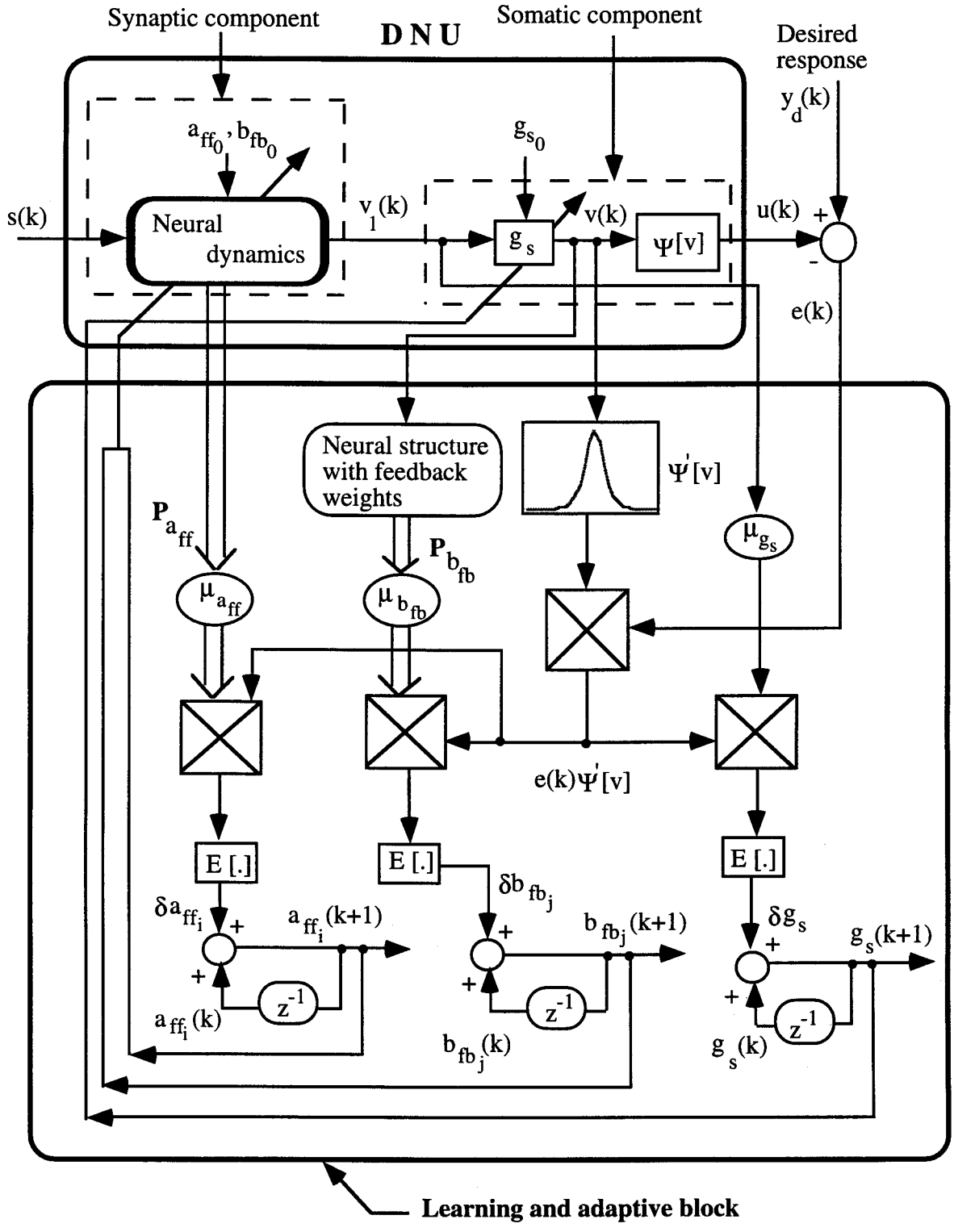


Figure 5.4b: The implementation scheme of the modified learning and adaptive algorithm.

5.4 Multi-Stage Dynamic Neural Network with Somatic Adaptation

Let the output of a DNU with the somatic component be written as

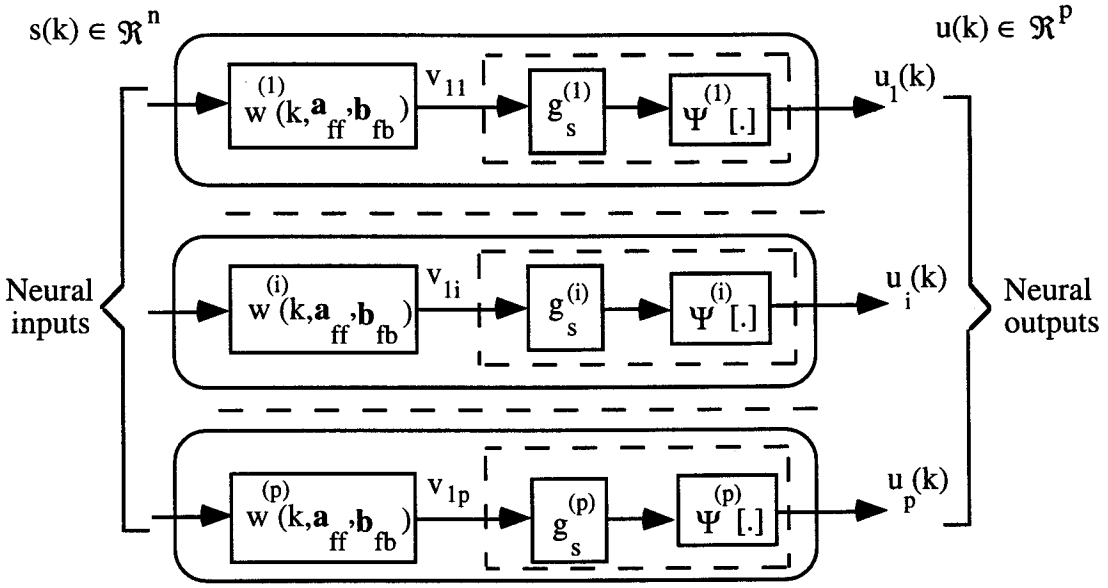
$$\omega(s) = \Psi\left[g_s\left(w(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}\right)s(k)\right] \tag{5.11}$$

where $\Psi[.]$ is a sigmoidal function with the varying slope g_s.

The input-output mapping of a single-stage dynamic neural network, shown in Fig 5.5a, with the DNUs in sigma configuration can be expressed as

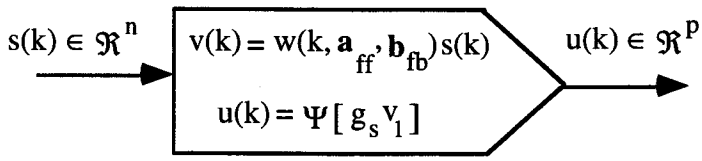
$$\begin{aligned} u(k) = \Psi^{(l)}\left[g_s^{(l)}\left(w^{(l)}(k, \mathbf{a}_{ff}^{l}, \mathbf{b}_{fb}^{l})\, s^{(l)}(k)\right)\right] + \ldots + \Psi^{(i)}\left[g_s^{(i)}\left(w^{(i)}(k, \mathbf{a}_{ff}^{i}, \mathbf{b}_{fb}^{i})\, s^{(i)}(k)\right)\right] \\ + \ldots + \Psi^{(p)}\left[g_s^{(p)}\left(w^{(p)}(k, \mathbf{a}_{ff}^{p}, \mathbf{b}_{fb}^{p})\, s^{(p)}(k)\right)\right] \\ = \omega_1(s) + \ldots + \omega_i(s) + \ldots + \omega_p(s) \\ = \sum_{n=1}^{p} \omega_n(s), \quad n = 1,..,i,..,p. \end{aligned} \tag{5.12}$$

The equivalent representation of an one-stage dynamic neural network with 'p' DNUs in parallel (in sigma mode) is shown in Fig. 5.5b.



(a)

Figure 5.5: (Continued)



(b)

Figure 5.5: Dynamic neural structure with DNU as the basic computing node.
(a): The sigma connection of 'p' DNUs to form a single-stage dynamic neural network, and
(b): The equivalent representation of a single-stage dynamic neural network with 'p' DNUs in parallel (in sigma mode).

Similarly, the input-output mapping of a single-stage of DNUs in pi (series or cascaded) configuration can be expressed as

$$u(k) = \left\{ \Psi^{(l)}\left[g_s^{(l)}\left(w^{(l)}(k, \mathbf{a}_{ff}^{l}, \mathbf{b}_{fb}^{l})\, s^{(l)}(k)\right)\right]\right\} \ldots \left\{ \Psi^{(i)}\left[g_s^{(i)}\left(w^{(i)}(k, \mathbf{a}_{ff}^{i}, \mathbf{b}_{fb}^{i})\, s^{(i)}(k)\right)\right]\right\}$$

$$\ldots \left\{ \Psi^{(p)}\left[g_s^{(p)}\left(w^{(p)}(k, \mathbf{a}_{ff}^{p}, \mathbf{b}_{fb}^{p})\, s^{(p)}(k)\right)\right]\right\}$$

$$= \omega_1(s) \ldots \omega_i(s) \ldots \omega_p(s)$$

$$= \prod_{n=1}^{p} \omega_n(s), \quad n = 1,..,i,..,p. \tag{5.13}$$

The dynamic neural system described by Eqns. (5.12) and (5.13) maps an n-dimensional input vector $s(k) \in \Re^n$ into a p-dimensional neural output vector $u(k) \in \Re^P$ where the mapping operation of an i-th neuron is given by

(i) The linear mapping operation (synaptic operation):

$$v_i(k) = w^{(i)}(k, \mathbf{a}_{ff}^{i}, \mathbf{b}_{fb}^{i})\, s^{(i)}(k), \quad \text{and} \tag{5.14a}$$

(ii) The nonlinear mapping operation (somatic operation):

$$u_i(k) = \Psi^{(i)}\left[g_s^{(i)}\left(v_i(k)\right)\right], \; i = 1, 2,.., p. \tag{5.14b}$$

The output of the dynamic neural network with a fully connected neural structure, with H stages and N_h DNUs in each layer where the output of the p DNU in layer h is connected to the input of the neuron r in the next layer, is

$$u_r^{h+1}(k) = \sum_{p=1}^{N_{h-1}} u_{(p,r)}^{h}(k) = \sum_{p=1}^{N_{h-1}} \left[\Psi^h \left\{ g_s^{(i)} \left(w_{(p,r)}^{h}(k, \mathbf{a}_{ff}, \mathbf{b}_{fb}) \, x_p^h(k) \right) \right\} \right] \quad (5.15)$$

where $1 \le p \le N_h$, $1 \le r \le N_{h+1}$, and $1 \le h \le H$.

The mathematical description of the three-stage dynamic neural network developed in Chapter 4 is applicable to the modified structure of the DNU as well.

5.5 Control of Unknown Nonlinear Systems: Simulation Studies

In the computer simulation studies discussed in this section, a nonlinear dynamic system of the form

$$y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i \, y(k-i) + \sum_{j=0}^{2} \beta_j \, u(k-j) \right] \quad (5.16)$$

was cascaded with a multi-stage dynamic neural network presented in Section 5.4. The neural network used for the computer simulation studies consisted of three-stages with two DNUs in each stage. Five simulation examples are presented, each one demonstrating a particular control objective. In Example 1, a nonlinear dynamic plant governed by the difference Eqn. (5.16) was considered. The parameters of the plant, β_{ff} and α_{fb}, and the nonlinear function f[.] were assumed unknown. As the neural network was trained, the plant response followed the desired command signal very closely. In Example 2, two cases were considered where a study of the robustness of the dynamic neural network was carried out by changing the nonlinear functions at different instants of the control process. Input signal adaptation is one of the important features of a good adaptive system. The effectiveness of the dynamic neural network to make the plant follow the input signal variations was demonstrated in Example 3. The adaptive capability of the dynamic neural network-based control scheme under dynamic perturbations, such as variations in nonlinear function, input signal and plant parameters, was demonstrated in Example 4. It was discussed in the previous chapter that a nonlinear plant can be represented by four different models. As the neuro-control scheme exhibits learning and adaptive capabilities, it was not mandatory to have an *a priori* knowledge about the nonlinear system under control. It was demonstrated in Example 5 that the neural network could adapt to different models of a nonlinear system.

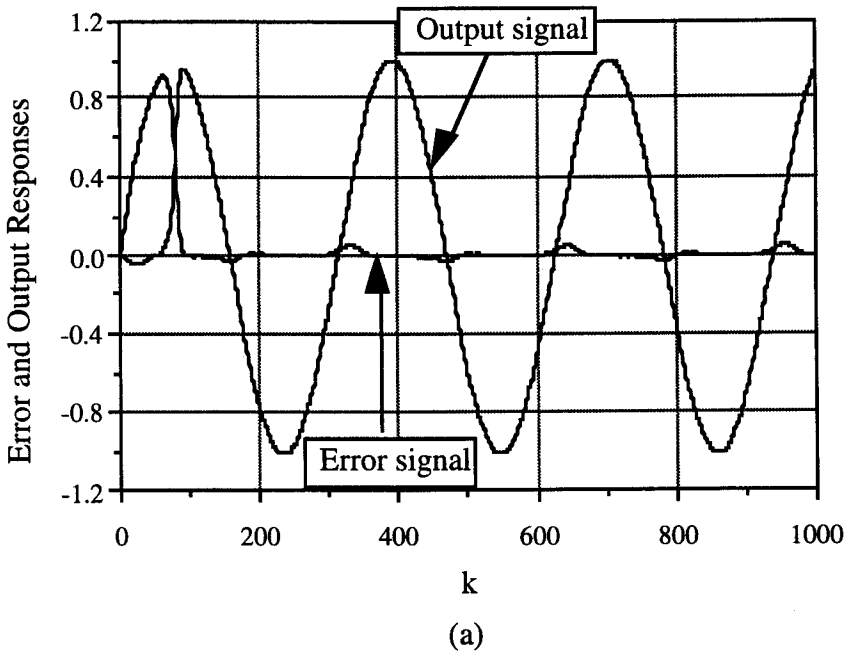
Example 1: Control of an unknown nonlinear plant

A general nonlinear plant described by Eqn. (5.16) was considered. The nonlinear characteristic of this plant was described by the following equation

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{1 + u^2(k-1) + u^2(k-2)}, \qquad (5.17)$$

with parameter values: $\beta_{ff} = [1.2, 1, 0.8]^T$ and $\alpha_{fb} = [1, 0.9, 0.7]^T$. The input to the system was $s(k) = \sin(2\pi k / 250)$ in the interval [-1, 1]. The simulation results obtained for this case are shown in Fig. 5.6. Figure 5.6a shows the error and output responses. Figures 5.6b and 5.6c show the adaptation in the somatic gain (slope) g_s of the activation function with respect to the learning trials k and the performance index J(.) respectively.

It is observed from Fig. 5.6a that the training of the neural network was slow during the initial period which resulted in a large initial error. However, this behavior depended upon the initial settings of adjustable parameters ($\mathbf{a}_{ff}$, $\mathbf{b}_{fb}$, g_s) of the DNUs. As the learning continued, the nonlinear plant followed the command input very closely with a small error. The adaptation in the somatic gain, g_s, as observed from Figs. 5.6c and 5.6d, was initially large and settled down to an average of 0.28 as the error signal converged to the preset tolerance limits of $\pm$ 0.1.



(a)

Figure 5.6: (Continued)

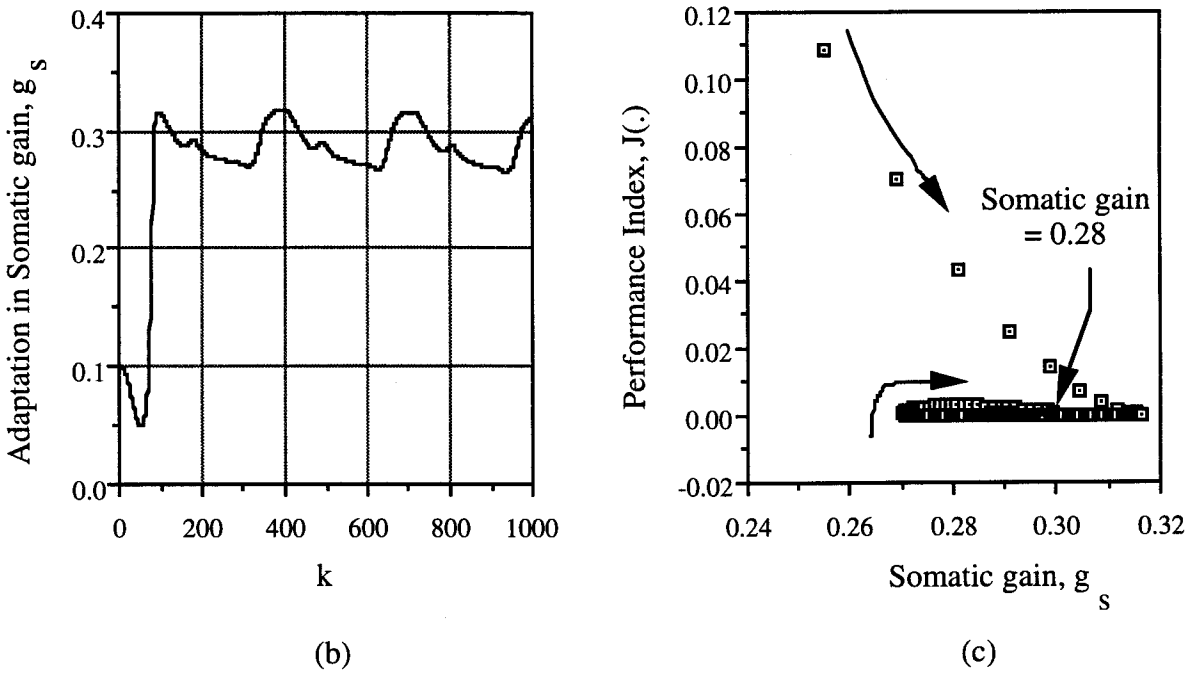


Figure 5.6: Simulation results with somatic adaptation, Example 1.

(a): The error and output responses,

(b): The adaptation in somatic gain g_s, and

(c): Performance index variation with respect to somatic gain.

Example 2: Control of a nonlinear plant with variations in nonlinear characteristics

In this example two cases were considered. In the first case, the nonlinear function f[.] was arbitrarily changed to a new function during the control process. In the second case, three different nonlinear functions were considered and the simulation was carried out for these different nonlinear functions to investigate the robustness of the dynamic neural network.

Case (i): The nonlinear plant to be controlled was the same as in Example 1 with the following nonlinearity

$$f[.] = \frac{\left[\sin\left\{\pi\left(y^2(k-2) + 0.5\right)\right\}\right] + 0.3\sin(2\pi u(k))}{1 + u^2(k-1) + u^2(k-2)}, \text{ f or } k < 250. \qquad (5.18a)$$

During the control process, the nonlinearity f[.] was changed to

$$f[.] = \sin\left\{\pi\left(y^2(k-1) + y^2(k-2)\right)\right\} + \sqrt{\left|\left\{u^2(k) + u^2(k-1) + u^2(k-2)\right\}\right|}, \text{ for } k \geq 250. \qquad (5.18b)$$

The resulting error and plant output responses are shown in Fig. 5.7a and the adaptation in the somatic gain in Figs. 5.7b and 5.7c. It is observed from the simulation results that the neural network was able to drive the plant towards the desired response even in the presence of changing nonlinear characteristics. The effect of changing the nonlinear function on the somatic gain is shown in Fig. 5.7c. The average somatic gain in this case was found to be 0.56.

Case (ii): The nonlinear plant considered in this case was the same as in Case (i). The objective of this simulation was to demonstrate the robustness of the dynamic neural network. Initially, the neural network was trained to a particular nonlinear function f[.]. While the control operation was in progress, the functions used to model the nonlinear system were changed. The neural network was then presented with these different nonlinear functions in a random sequence.

It was observed that the effect of introducing different nonlinear functions on the system performance was smaller on the subsequent occurrences of variations in the nonlinear functions. This illustrates that the neural network adapted to the different nonlinear functions very quickly. The three nonlinear functions used in this simulation example were as follows:

$$f_1[.] = \frac{\left[\sin\left\{\pi\left(y^2(k-2) + 0.5\right)\right\}\right] + 0.3\sin(2\pi u(k))}{1 + u^2(k-1) + u^2(k-2)} \tag{5.19a}$$

for $0 \le k < 350$, and $2000 \le k < 2500$,

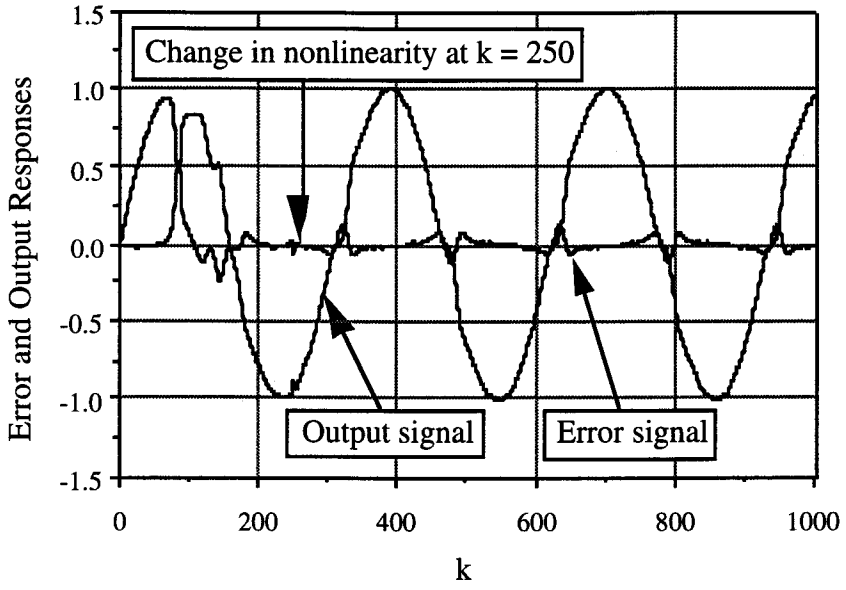
$$f_2[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{1 + u^2(k-1) + u^2(k-2)} \tag{5.19b}$$

for $350 \le k < 800$, $1200 \le k < 1600$, and $k > 2500$,

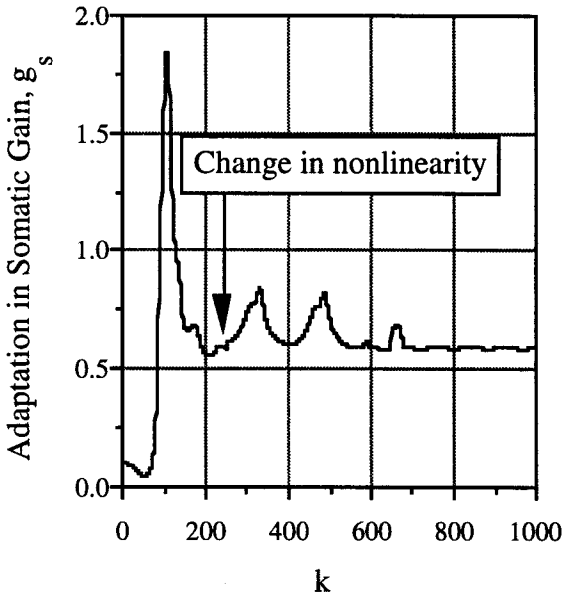
$$f_3[.] = e^{(y^2(k-1)+y^2(k-2))} + \sqrt{\left|\left\{u^2(k) + u^2(k-1) + u^2(k-2)\right\}\right|} \tag{5.19c}$$

for $800 \le k < 1200$, and $1600 \le k < 2000$.

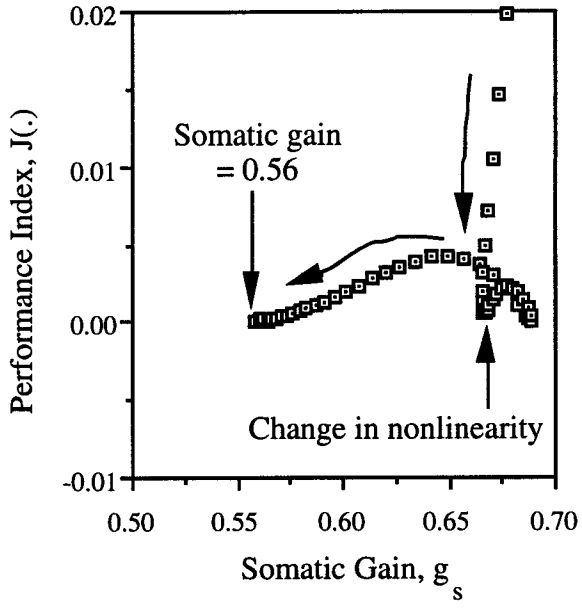
The error and output responses are shown in Figs. 5.8a and 5.8b respectively, and the corresponding variation in the somatic gain in Fig. 5.8c.



(a)



(b)



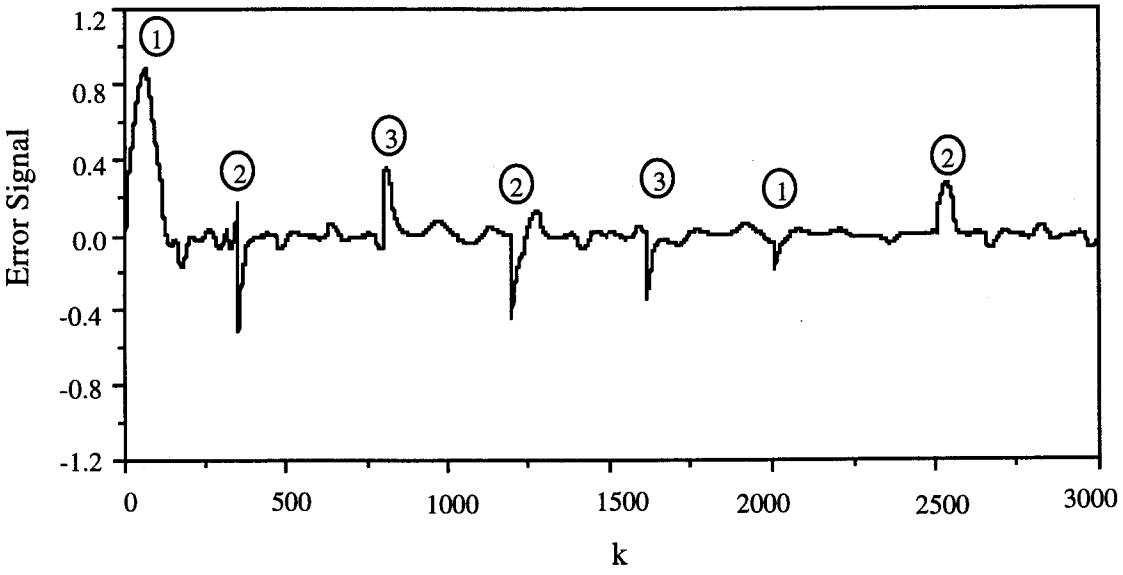
(c)

Figure 5.7: Simulation results with somatic adaptation, Example 2, Case (i).

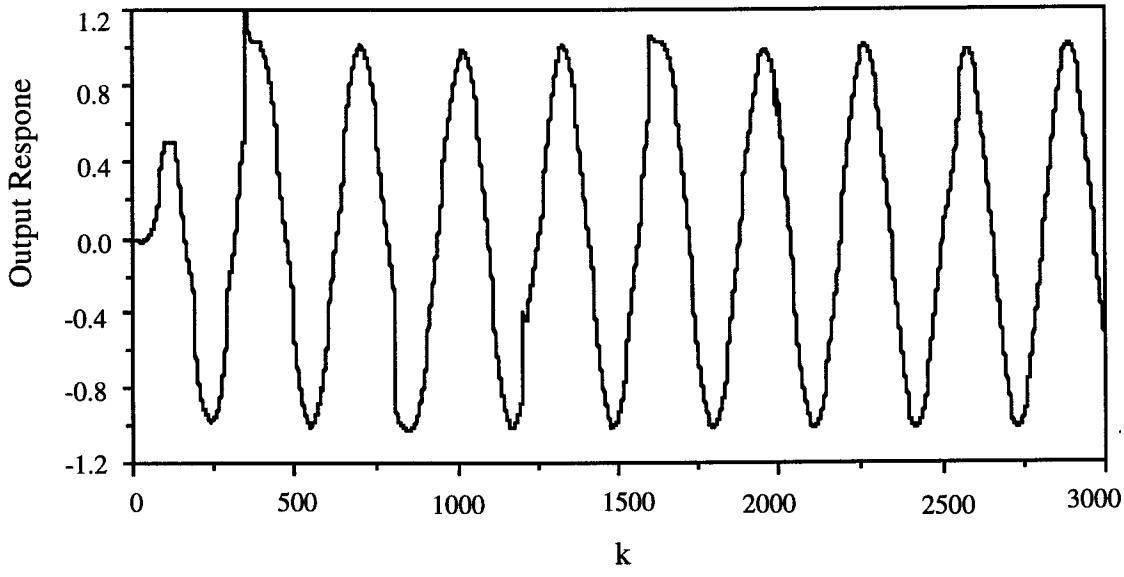
(a): The error and output responses,

(b): The adaptation in somatic gain g_s, and

(c): Performance index variation with respect to somatic gain.



(a)



(b)

Figure 5.8: (Continued)

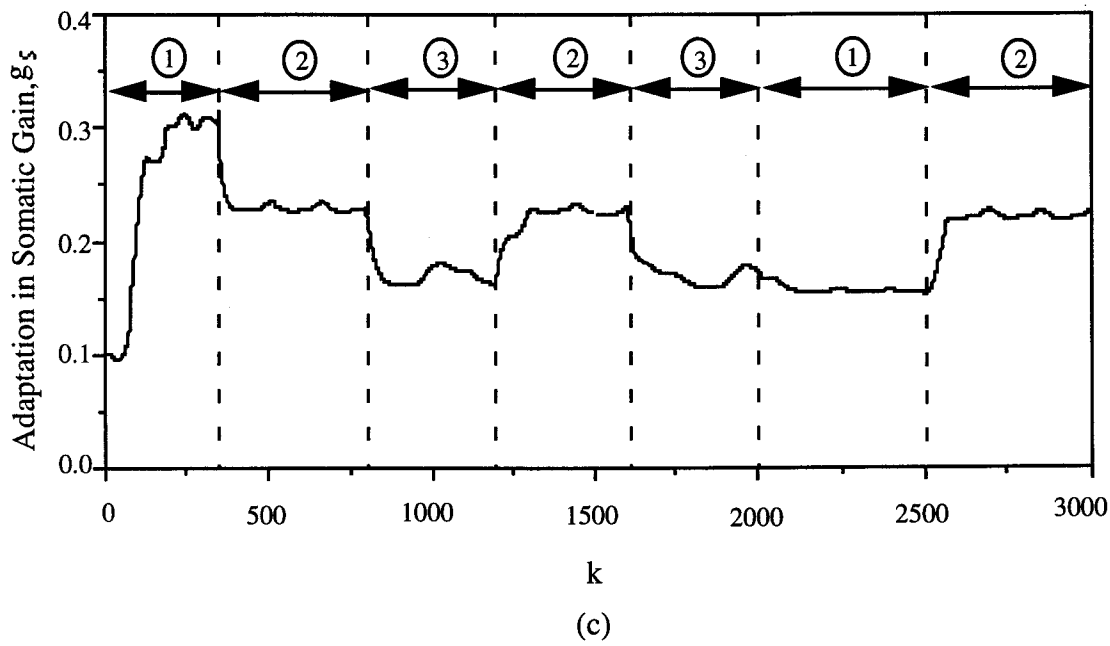


Figure 5.8: Simulation results with somatic adaptation, Example 2, Case (ii).

(a): Error response to the changes in nonlinearity characteristics. Numbers in circles, 1, 2 and 3, represent different nonlinearity characteristics as described in Equations (5.19a), (5.19b) and (5.19c) respectively,

(b): The output response, and

(c): The adaptation in somatic gain.

Example 3: Control of a nonlinear plant with variations in input signal

The objective of this simulation example was to demonstrate the input signal adaptive capability of the dynamic neural network. The plant and the nonlinear function f[.] in this example were the same as in Example 1, but in this example the input signal s(k) was varied in the interval [-1.2, 1.2] as follows:

$$
\begin{aligned}
&s(k) = \sin(2\pi k / 250), \quad 0 \le k < 350 \\
&s(k) = 1.2, \quad \text{for} \quad 350 \le k < 500 \\
&s(k) = 0.4, \quad \text{for} \quad 500 \le k < 600 \\
&s(k) = -0.2, \quad \text{for} \quad 600 \le k < 800 \\
&s(k) = -0.6, \quad \text{for} \quad 800 \le k < 1000 \\
&s(k) = 1.2 \cos(2\pi k / 150), \quad \text{for} \quad k \ge 1000.
\end{aligned}
\tag{5.20}
$$

The corresponding error and output responses are shown in Figs. 5.9a and 5.9b respectively. The adaptation in the somatic gain for the input signal variations is shown in Fig. 5.9c.

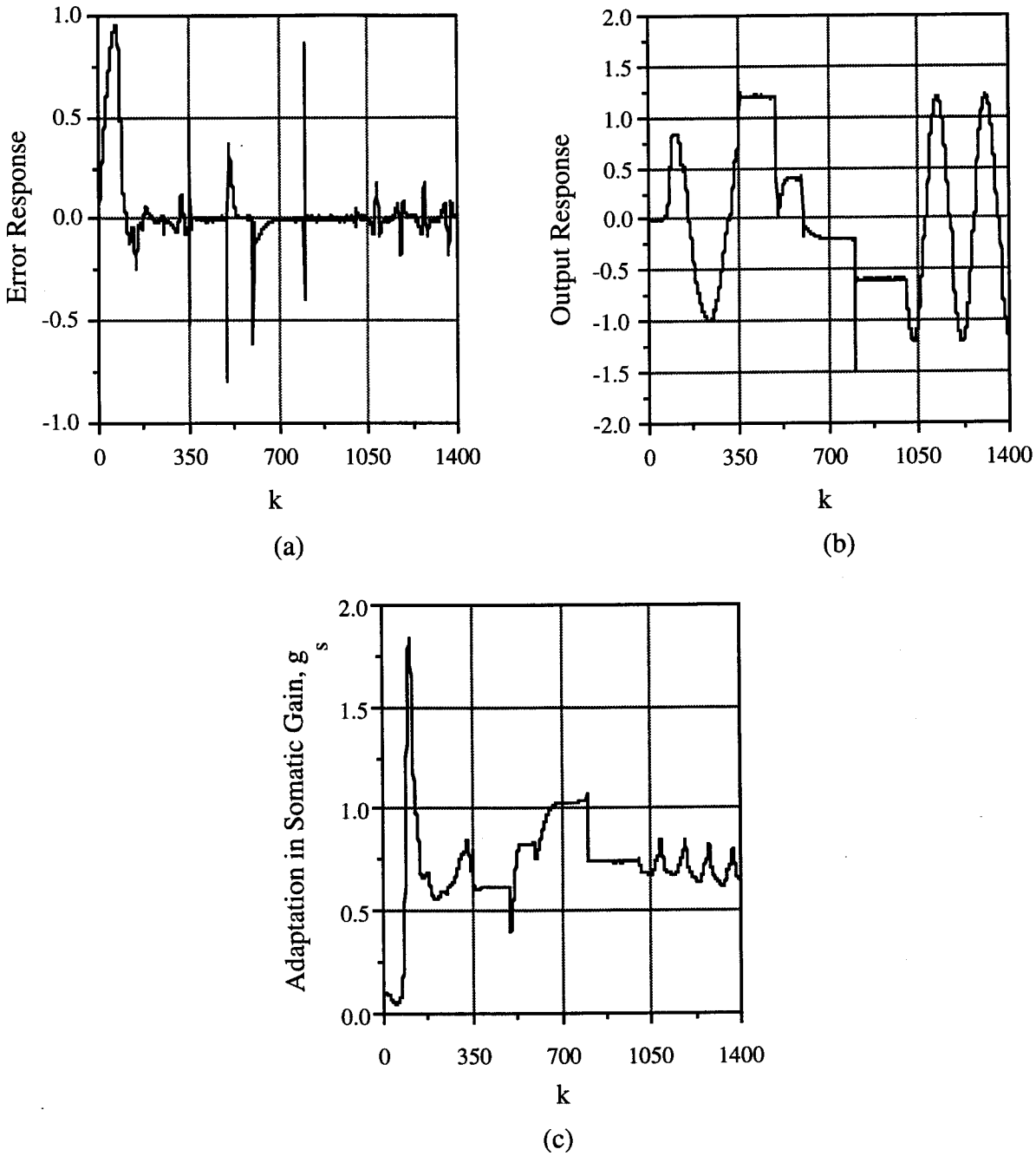


Figure 5.9: Simulation results for input signal variations, Example 3.

(a): The error response,

(b): The output response, and

(c): The adaptation in somatic gain for variations in input signal.

Example 4: Control of a nonlinear plant with variations in nonlinear function, input signal and plant parameters

In the above examples it was shown that the dynamic neural network was able to make a nonlinear plant follow the changes in the nonlinear characteristics and the input signal. The objective of this simulation example was to demonstrate the adaptive control capability of the neural network under the following situations: (i) time-varying nonlinear functions, (ii) varying pattern of input signals, and (iii) perturbations in the plant parameters. The nonlinear function used in this example was

$$f[.] = \frac{\sin\{\pi(y^2(k-2) + 0.5)\} + \sqrt{|\{u^2(k) + u^2(k-1) + u^2(k-2)\}|}}{1 + y^2(k-1) + y^2(k-2)}, \quad 0 \le k < 400 \tag{5.21a}$$

which was perturbed at k = 400 to

$$f[.] = \frac{[2+\cos\{7\pi(y^2(k-1) + y^2(k-2))\}] + e^{-u(k)}}{1 + u^2(k-1) + u^2(k-2)}, \quad 400 \le k < 1400. \tag{5.21b}$$

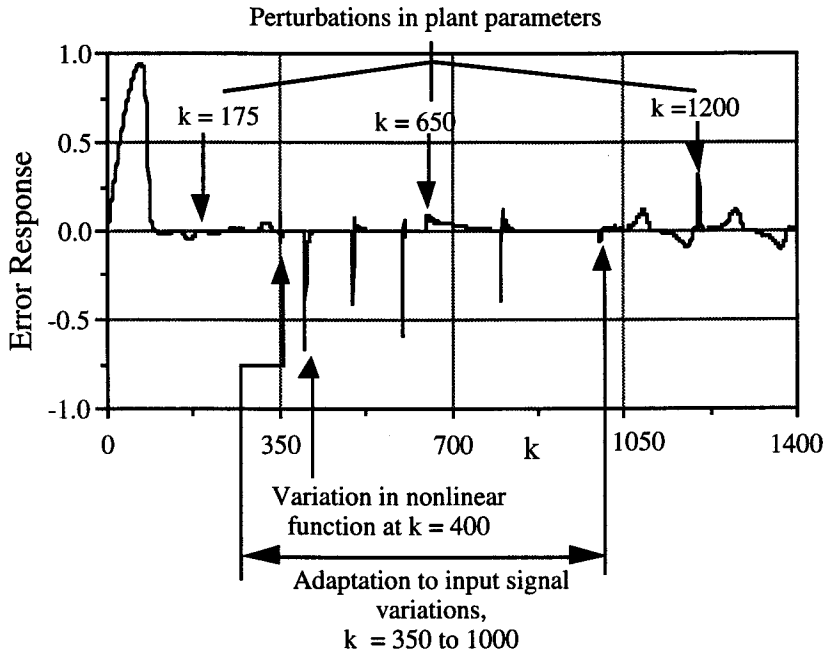
The input to the system, $s(k) = \sin(2\pi k / 250)$ in the interval [-1, 1], was changed as follows

$$\begin{aligned}
&s(k) = \sin(2\pi k / 250), && \text{for } 0 \le k < 350,\\
&s(k) = 0.8, && \text{for } 350 \le k < 500,\\
&s(k) = 0.4, && \text{for } 500 \le k < 600,\\
&s(k) = -0.2, && \text{for } 600 \le k < 800,\\
&s(k) = -0.6, && \text{for } 800 \le k < 1000, \text{ and}\\
&s(k) = 1.2\cos(2\pi k / 150), && \text{for } 1000 \le k \le 1400.
\end{aligned} \tag{5.22}$$

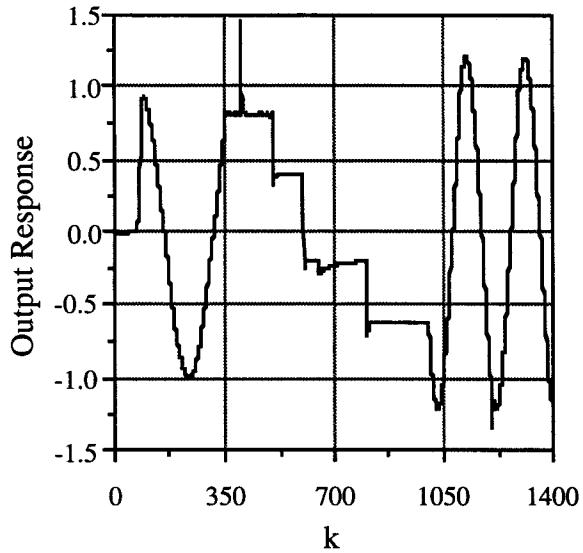
The plant parameters were:

$$\begin{aligned}
&\beta_{ff} = [1.4, 1.2, 0]^T, && \alpha_{fb} = [1, 1, 0]^T, && \text{for } 0 \le k < 175,\\
&\beta_{ff} = [1.4, 1.2, 0.3]^T, && \alpha_{fb} = [1, 1, 0.3]^T, && \text{for } 175 \le k < 650,\\
&\beta_{ff} = [1.4, 1.2, 1.0]^T, && \alpha_{fb} = [1, 1, -0.1]^T, && \text{for } 650 \le k < 1200, \text{ and}\\
&\beta_{ff} = [1.4, 1.2, 0]^T, && \alpha_{fb} = [1, 0, 0.3]^T, && \text{for } k \ge 1200.
\end{aligned} \tag{5.23}$$

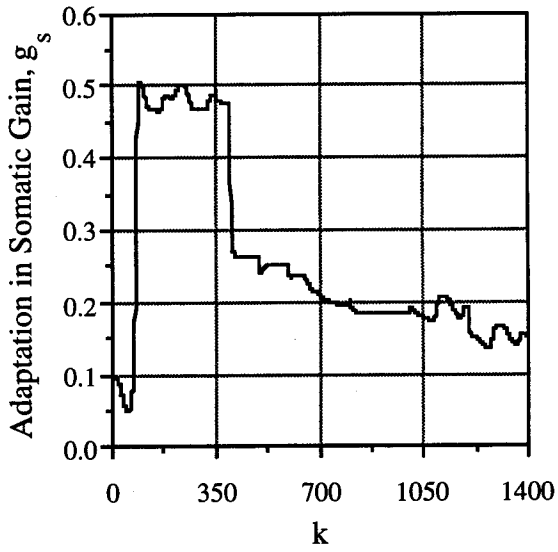
The simulation results obtained for this example are shown in Fig. 5.10. This example demonstrates the robustness of the neural network for variations in the nonlinearity characteristics, input signal, and changes in the dynamic characteristics of the plant.



(a)



(b)



(c)

Figure 5.10: Simulation results for dynamic perturbations in the plant, Example 4.

(a): The error response to variations in the nonlinearity characteristics, input signal and plant parameters,

(b): The corresponding output response, and

(c): The adaptation in somatic gain.

Example 5: Adaptations to system model representations

In this example the adaptive capability of the neural network with somatic adaptation was demonstrated by changing the plant models during the control process. The same changes were made as discussed in Chapter 4. The error and output responses obtained for this simulation example are shown in Figs. 5.11a and 5.11b respectively. The effect of changing the nonlinear plant models on the somatic gain is shown in Figure 5.11c. It is observed from this figure that the neural network was able to adapt very quickly to the changing models of the nonlinear plant.

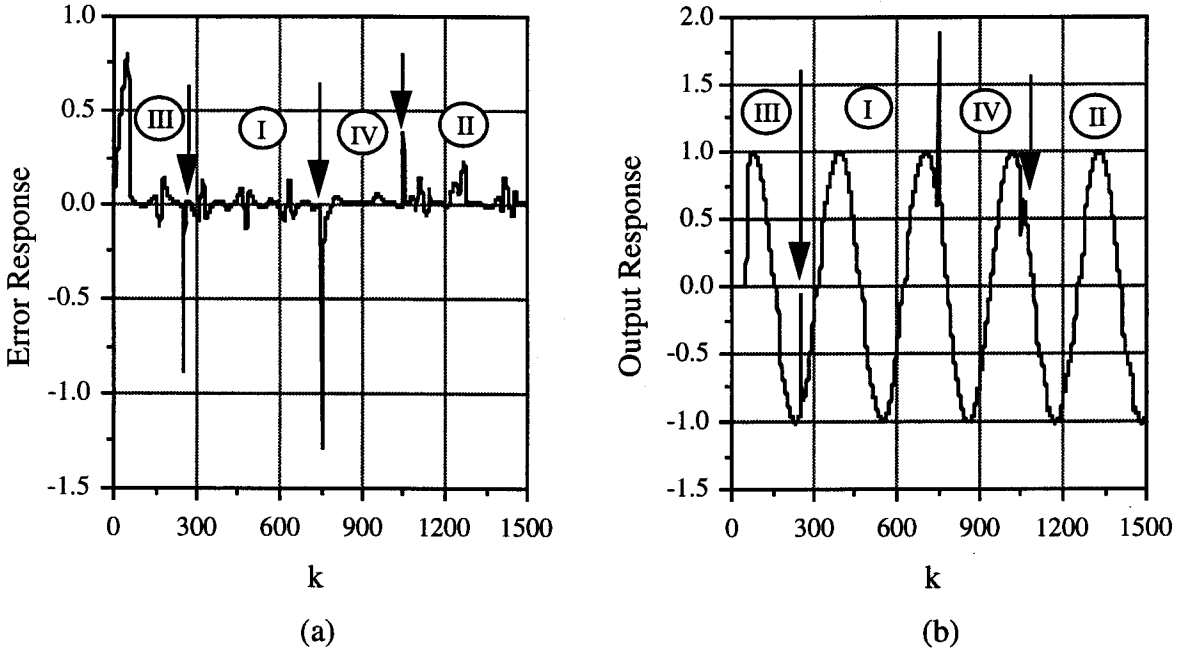
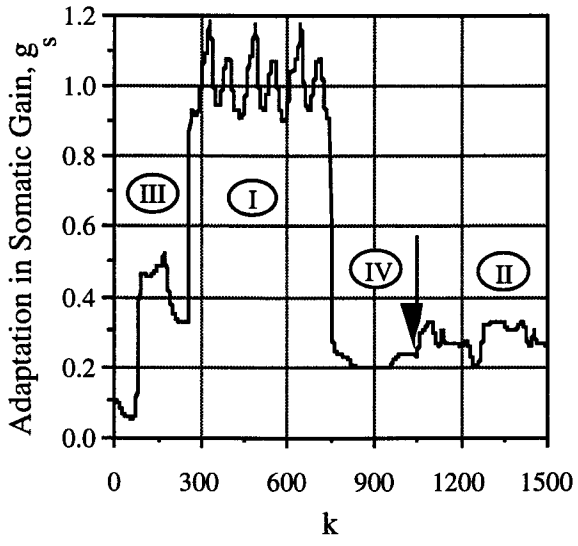


Figure 5.11: (Continued)



(c)

Figure 5.11: Simulation results for model variations, Example 5.

(a): The error response. Circles with numbers I, II, III and IV denoted the different nonlinear models as described by Eqns. (4.14a) through (4.14d) in Chapter 4,

(b): The corresponding output response, and

(c): The adaptation in somatic gain, g_s, for variations in nonlinear plant models.

5.6 Summary

The importance of changing the slope of the nonlinear function in a neural network has been emphasized in this chapter. A biological basis for such a change in the neural network structure has been discussed. Through computer simulation studies, it was demonstrated that the capability of a neural network can be enhanced with somatic adaptation compared to a neural network with non-adaptable sigmoidal functions. A modified DNU structure having both the synaptic and somatic components has been proposed. A modified algorithm to update the adjustable parameters, namely the feedforward and feedback synaptic weights, and the slope of the sigmoid function, has been presented. An implementation scheme for this algorithm has also been presented. A few simulation examples have been presented and discussed demonstrating a particular control objective.

This, and the preceding chapters, have described the dynamic neural unit (DNU) and the multi-stage dynamic neural structures with the DNU as the basic computing element.

However, it is postulated in neuro-physiology that the collective activity generated by large numbers of locally redundant neurons is more significant in a computational context than the activity generated by a single neuron [41]. The total neural activity results from a collective assembly of neural cells called *neural subpopulations.* A subpopulation contains a large class of similar neurons that lie in close spatial proximity. A simple approach for the study of subpopulations of neurons would be to consider a neural model consisting of excitatory and inhibitory neurons. A dynamic neural structure based on the concept of neural subpopulations is developed in the next chapter.

6. Dynamic Neural Processor with Excitatory and Inhibitory Neurons

6.1 Introduction

It was demonstrated in the previous chapters, through computer simulation studies, that a dynamic neural network with a DNU as the basic computing element can control unknown nonlinear systems. This neural structure is different from the conventionally assumed structure of both feedforward and feedback networks in the sense that the basic functional node in a neural network, the DNU, developed in this thesis consists of two delay operators with feedforward and feedback connections forming a second-order linear structure. The output of this linear structure constitutes an argument to a time-varying sigmoid function. A number of such DNUs are connected in a conventional feedforward network comprising of input, intermediate and output stages without any feedback between the stages. The implicit assumption in traditional feedforward and feedback neural networks is that the behavior of each neuron in the networks is deterministic.

Experiments in neuro-physiology have shown that the response of a biological neuron at the single cell level is unpredictable, and that the total neural activity results from a collective assembly of cells called neural subpopulations. A subpopulation contains a large class of similar neurons that lie in close spatial proximity. As mentioned in the previous chapter, a simple approach for the study of subpopulations of neurons would be to consider a neural model consisting of excitatory and inhibitory neurons.

In view of the above, a neural structure, called the dynamic neural processor (DNP), consisting of two DNUs coupled in excitatory and inhibitory modes is proposed in this chapter. The DNP emphasizes the collective action of the subpopulation of neurons. The biological basis for the development of DNP is detailed in the next section. A detailed mathematical model and an algorithm to modify the parameters of the DNP are also described in this section. Four applications of the DNP are discussed in Section 6.3. The first application, discussed in Section 6.3.1, involves the use of the DNP for the approximation of arbitrary nonlinear functions. A comparative study of the DNP with single- and two-layer recurrent neural networks is also discussed in this section. The next subsection contains a discussion of the use of the DNP for computing the inverse kinematic transformations of a two-link robot. This is followed in Section 6.3.3 by the application of the DNP to the control of unknown nonlinear dynamic systems. In the fourth application, the DNP is employed for

the coordination and control of multiple systems. A generalized dynamic neural model based on the concept of neural subpopulations is developed in Section 6.4, followed by the concluding remarks in the last section.

6.2 Architectural Details of the Proposed Neural Structure

6.2.1 The Biological Basis

The neural network models described in the existing literature often consider the behavior of a single neuron as the basic computing unit in neural information processing operations. Each computing unit in the network is based on the concept of an idealized neuron. An ideal neuron is assumed to respond optimally to the applied inputs. However, experimental studies in neuro-physiology have shown that the response of a biological neuron is random [91], and only by averaging many observations is it possible to obtain predictable results. This observed variability in the response of a neuron is determined by the uncontrolled or extraneous electrical signals that are received from the activated neurons in other parts of the nervous system. As well, this variability is enhanced by the intrinsic fluctuations of the electrical membrane potential within the neuron [41, 90, 91] .

In general, the states of a biological neuron can be considered as a random process. However, mathematical analysis has shown that these cells can transmit reliable information if they are sufficiently redundant in number. It is postulated [41], therefore, that the collective activity generated by large numbers of locally redundant neurons is more significant in a computational context than the activity generated by a single neuron. Furthermore, the study of neurons at the individual cell level may be appropriate to emulate some functions of the biological neural network. However, the study of neurons at the individual cell level is not necessarily suited for investigations of complex cognitive functions, such as sensory information processing, learning, memory storage and recall, and vision. This shift in emphasis from the study of single neurons to the study of neural mass is warranted since the sensory information is introduced into the nervous system in the form of large scale spatio-temporal activity in the sheets of cells. The number of cells involved is simply too large for any approach starting at the single cell to be tractable [41, 90, 92].

The total neural activity generated within a tissue layer is a result of the spatially localized assemblies of the densely interconnected nerve cells called a neural population, or a neural mass. The neural population is composed of neurons, and its properties have a generic resemblance to those of individual neurons, but it is not identical to them and its properties

cannot be predicted from measurements on single neurons [41, 90]. This is due to the fact that the properties of a neural population depend on various parameters of individual neurons and also on the interconnections between neurons. The conceptual gap between the functions of single neurons and those of a neural mass is still very wide.

Each neural population may be further divided into several coexisting subpopulations. A subpopulation contains a large class of similar neurons that lie in close spatial proximity. The neurons in each subpopulation are assumed to receive a common set of inputs and provide the corresponding outputs. The individual synaptic connections within any subpopulation are random, but dense enough to ensure that at least one mutual connection exists between any two neurons. The most common neural mass is the mixture of excitatory (positive) and inhibitory (negative) subpopulations of neurons [41, 91 - 93]. The excitatory neural subpopulation increases the electro-chemical potential of the post-synaptic neuron, while the inhibitory subpopulation reduces the electro-chemical potential. The individual neurons within the subpopulations that generate $\mathbf{Y}(k)$ receive stimuli from other neurons in the nervous-tissue layer, self-feedback signals and a common signal space $\mathbf{S}(k)$ external to the tissue layer, as depicted in Fig. 6.1.

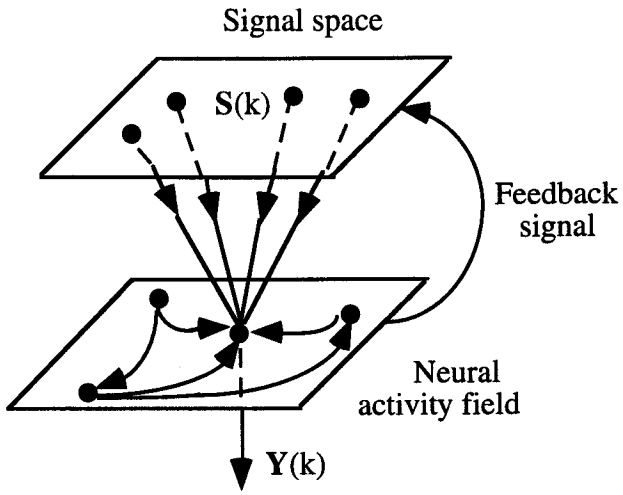


Figure 6.1: Schematic diagram of a neural activity field $\mathbf{Y}(k)$, in response to a signal space $\mathbf{S}(k)$. This structure represents the functional dynamics of a nervous-tissue layer.

The minimum topology of such a neural mass contains excitatory (positive), inhibitory (negative), excitatory - inhibitory (synaptic connection from excitatory to inhibitory), and inhibitory - excitatory (synaptic connection from inhibitory to excitatory)

feedback loops. The morphology of a neural mass is shown in Fig. 6.2. One of the important attributes of a neural mass is that its fundamental characteristics may be described in terms of linear systems theory [93]. This property implies that the operations in a neural mass, within the appropriate limits of amplitude, conform to the principle of superposition that make the responses to two or more inputs additive.

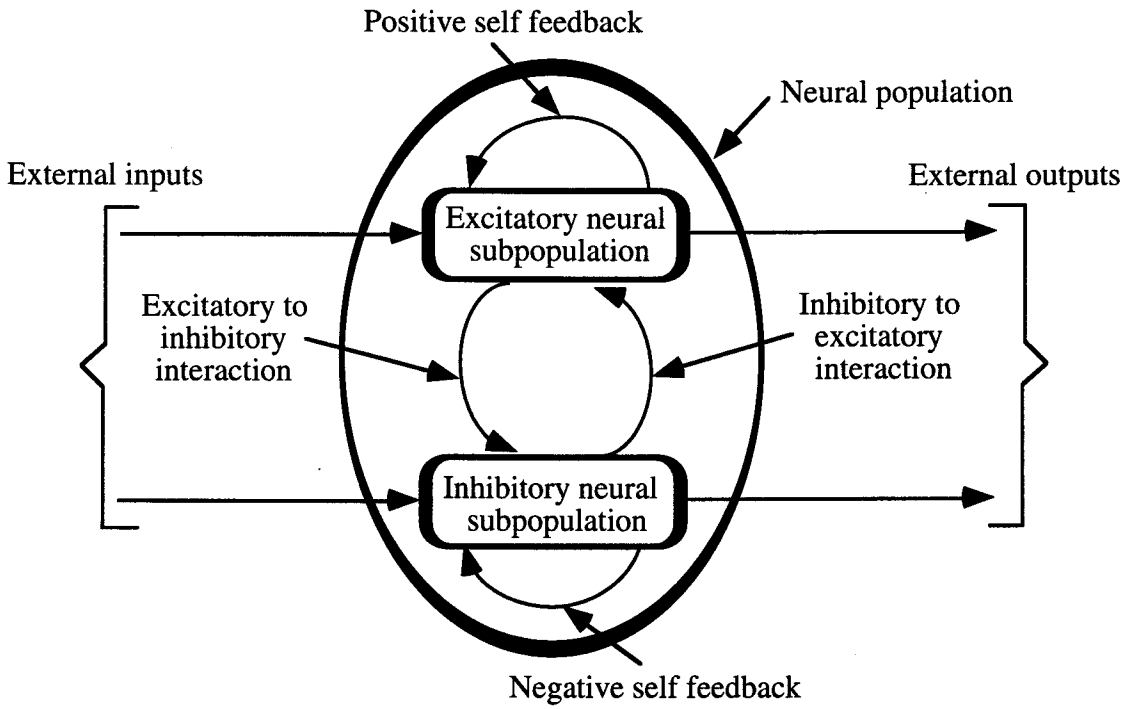


Figure 6.2: A schematic diagram of the coupled interactions between excitatory and inhibitory neural subpopulations within a neural population.

Strumillo and Durani used a neural model based on neural population to study cardiac arrhythmia [94]. Gupta and Knopf [42] proposed a neural structure, called the P-N Neural Processor (PNNP), for machine vision applications. The state-space model of this visual information processor corresponds to a bi-layered two-dimensional array of densely interconnected nonlinear first-order dynamic neurons called processing elements (PEs). An individual PE consists of a multi-modal sigmoidal function followed by a first-order dynamic structure. The connections between the neural subpopulations are fixed. This neural structure has been used successfully for many applications such as spatio-temporal filtering, motion detection, spatio-temporal stabilization, short-term visual memory, content-addressable memory, and pulse frequency modulation [42]. No control applications of the P-N neural processor have been reported.

6.2.2 Mathematical Model of Dynamic Neural Processor

In view of the above remarks, a neural model called the dynamic neural processor (DNP), which emphasizes the dynamic properties of a subpopulation, is proposed in this chapter. The important assumption in this model is that all nervous processes of any complexity depend upon the interaction of excitatory and inhibitory neurons.

The DNP consists of two DNUs, developed in the preceding chapter, which are configured to function as antagonistic neural units as depicted in Fig. 6.3. In this structure, the following notations are used:

- subscript λ indicates either an excitatory, E, or inhibitory, I, state,

- $s_\lambda(k)$ and $u_\lambda(k)$ represent respectively the neural stimulus (input) and neural output of the computing unit,

- $s_{t\lambda}(k)$ denotes the total input to the neural units,

- $w_{\lambda\lambda}$ represent the strength of the self-synaptic connections (w_{EE}, w_{II} in Fig. 6.3),

- $w_{\lambda\lambda'}$ represent the strength of the cross synaptic or inter-subpopulation connections from one neural unit to another (w_{IE}, w_{EI} in Fig. 6.3),

- z^{-1} elements denote communication delays in the self- and inter-subpopulation paths,

- $s_E(k)$ and $s_I(k)$ are the excitatory and inhibitory neural inputs respectively,

- w_E and w_I are the input weights for the excitatory and inhibitory neural inputs respectively,

- $u_E(k)$ and $u_I(k)$ represent the responses of the excitatory and inhibitory neural subpopulations respectively, and

- θ_E and θ_I represent the thresholds of excitatory and inhibitory neurons respectively.

The functional dynamics exhibited by a neural computing unit, the DNU, is defined by a second-order difference equation as represented in Chapter 5 by Eqn. (5.2b). The state variables $u_E(k+1)$ and $u_I(k+1)$ generated at time (k+1) by the excitatory and inhibitory neural units of the proposed neural processor are modeled by the nonlinear functional relationships

$$u_E(k+1) = \mathbf{E}\,[u_E(k)\,,\ v_E(k)],\ \text{and} \tag{6.1a}$$

$$u_I(k+1) = \mathbf{I}\,[u_I(k)\,,\; v_I(k)] \qquad (6.1b)$$

where $v_E(k)$ and $v_I(k)$ represent the proportion of neurons in the neural unit that receive inputs greater than an intrinsic threshold, and **E** and **I** represent the nonlinear excitatory and inhibitory actions of the neurons. The neurons that receive inputs greater than a threshold value are represented by a nonlinear function $\Psi\,[v_\lambda(k)]$. This nonlinear function is related to the distribution of neural thresholds within the neural unit [41, 42]. Ideally, the neural model based on the neural population should consist of a number of identical neurons in each neural subpopulation. However for simplicity, it is assumed that the DNP consists of only one dynamic neuron (DNU) in each subpopulation.

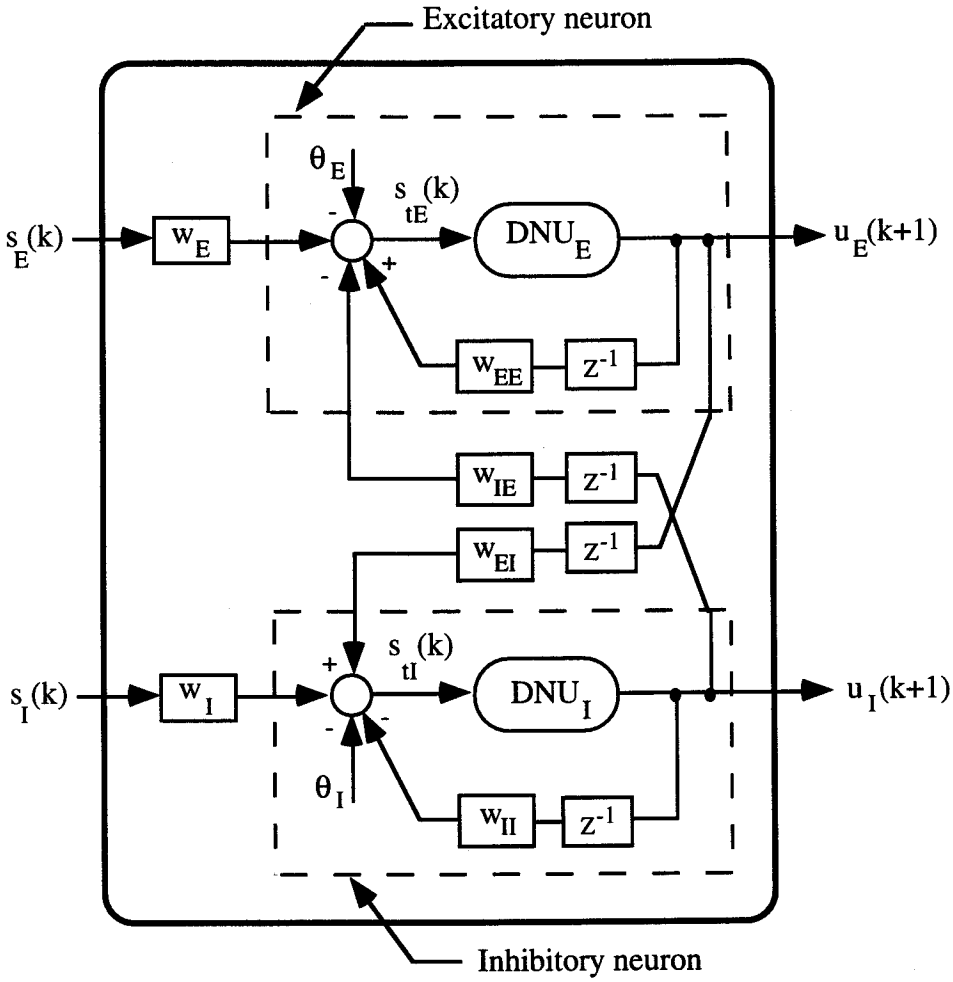


Figure 6.3: The dynamic neural processor with two dynamic neural units coupled as excitatory and inhibitory neurons represented as DNU_E and DNU_I respectively.

The inputs incident on the excitatory and inhibitory neural units are respectively

$$s_{tE}(k) = w_E s_E(k) + w_{EE} u_E(k) - w_{IE} u_I(k) - \theta_E \quad , \text{ and} \tag{6.2a}$$

$$s_{tI}(k) = w_I s_I(k) - w_{II} u_I(k) + w_{EI} u_E(k) - \theta_I \tag{6.2b}$$

where w_E and w_I are the weights associated with the excitatory and inhibitory neural inputs respectively, w_{EE} and w_{II} represent the self-synaptic connection strengths, w_{IE} and w_{EI} represent the inter-neuron synaptic strengths, and θ_E and θ_I represent the thresholds of excitatory and inhibitory neurons respectively. The above equations may be written in matrix form as follows:

$$\begin{bmatrix} s_{tE}(k) \\ s_{tI}(k) \end{bmatrix} = \begin{bmatrix} w_E & 0 \\ 0 & w_I \end{bmatrix} \begin{bmatrix} s_E(k) \\ s_I(k) \end{bmatrix} + \begin{bmatrix} w_{EE} & -w_{IE} \\ w_{EI} & -w_{II} \end{bmatrix} \begin{bmatrix} u_E(k) \\ u_I(k) \end{bmatrix} - \begin{bmatrix} \theta_E \\ \theta_I \end{bmatrix}. \tag{6.3}$$

A general expression for the dynamic neural activity of the DNP may be represented in a compact form by the following equation:

$$\mathbf{S}_{t\lambda} = \mathbf{W}^{\lambda}_{\lambda'} \mathbf{S}_{\lambda} + \mathbf{W}^{\lambda\lambda}_{\lambda\lambda'} \mathbf{U}_{\lambda} - \theta_{\lambda}, \tag{6.4}$$

where, $\mathbf{S}_{t\lambda} = \begin{bmatrix} s_{tE}(k) \\ s_{tI}(k) \end{bmatrix}$: input incident vector ; $\mathbf{S}_{\lambda} = \begin{bmatrix} s_E(k) \\ s_I(k) \end{bmatrix}$: stimulus (input) vector;

$\mathbf{W}^{\lambda}_{\lambda'} = \begin{bmatrix} w_E & 0 \\ 0 & w_I \end{bmatrix}$: input scale matrix; $\mathbf{W}^{\lambda\lambda}_{\lambda\lambda'} = \begin{bmatrix} w_{EE} & -w_{IE} \\ w_{EI} & -w_{II} \end{bmatrix}$: synaptic weight matrix;

and $\mathbf{U}_{\lambda} = \begin{bmatrix} u_E(k) \\ u_I(k) \end{bmatrix}$: response (output) vector; $\theta_{\lambda} = \begin{bmatrix} \theta_E \\ \theta_I \end{bmatrix}$: threshold vector;

assuming that $\mathbf{W}^{\lambda\lambda}_{\lambda\lambda'}$ is a nonsingular matrix. From Eqn. (6.4), the responses of the neural units, $u_{\lambda}(k)$, in terms of the stimulus (input) $s_{\lambda}(k)$, the input $s_{t\lambda}(k)$, and the strength of the synaptic connections $w_{\lambda\lambda}$ and $w_{\lambda\lambda'}$ may be obtained as

$$\mathbf{U}_{\lambda} = \left[\mathbf{W}^{\lambda\lambda}_{\lambda\lambda'} \right]^{-1} \left[\mathbf{S}_{t\lambda} - \mathbf{W}^{\lambda}_{\lambda'} \mathbf{S}_{\lambda} + \theta_{\lambda} \right]. \tag{6.5}$$

A direct analytical solution for determining the steady-state and temporal behavior exhibited by the DNP is not possible because of the inherent nonlinearities in Eqns. (6.2a) and (6.2b). However, these nonlinear equations can be analyzed qualitatively by obtaining the phase trajectories in the u_E - u_I phase plane [41, 42, 98]. These trajectories enable the system characteristics to be observed without solving the nonlinear equations. The locus of points where the phase trajectories have a given slope is called an isocline. The steady-state activity exhibited by the DNUs of the neural processor can be investigated by determining the isoclines corresponding to $u_E(k+1) = u_E(k)$ and $u_I(k+1) = u_I(k)$. A typical plot of the isoclines for $s_E(k) = 0$, $s_I(k) = 0$ is shown in Fig. 6.4. The weight parameters for these curves are: $w_{EE} = 20$, $w_{IE} = 5$, $w_{EI} = 8$, $w_{II} = 10$, $\theta_E = \theta_I = 0.5$. In this case, there is only one steady-state solution corresponding to the one intersection of the two curves. However, depending upon the strength of the synaptic connections, there may be more than one solution, and the solution may be stable (+) or unstable (-) depending upon where the two isoclines intersect [41].

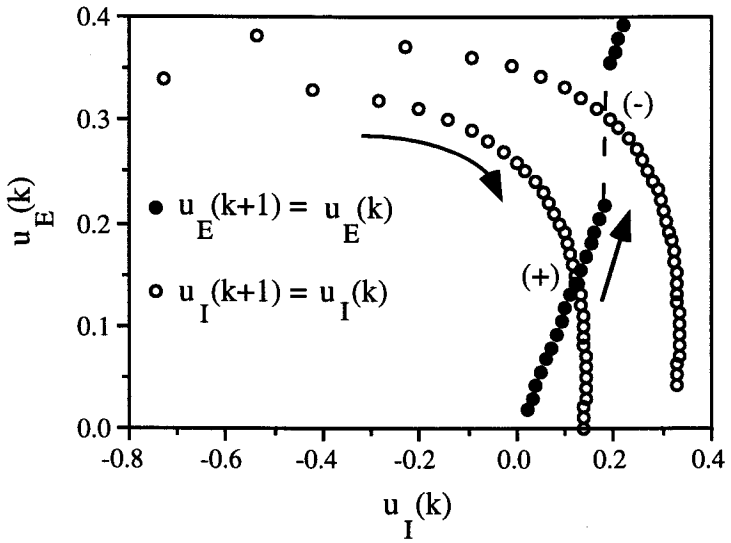


Figure 6.4: The isoclines for $s_E(k) = 0$, $s_I(k) = 0$, and (+) denotes stability and (-) instability of steady-state.

The steady-state behavior of the DNP is the superposition of the individual responses, $u_\lambda(k)$, of the excitatory and inhibitory neural subpopulations and is given by

$$u(k) = u_E(k) + u_I(k)$$

$$= \sum_{i=1}^{n_E} \left\{ \Psi_{Ei} \left[g_{sEi} \, v_{1Ei}(k) - \theta_{Ei} \right] \right\} + \sum_{j=1}^{n_I} \left\{ \Psi_{Ij} \left[g_{sIj} \, v_{1Ij}(k) - \theta_{Ij} \right] \right\} \qquad (6.6)$$

where n_E and n_I represent the population of the antagonistic neural units (number of excitatory and inhibitory neurons respectively) in a neural population. The interpretation of Eqn. (6.6) is that the total activity of the neural population is the summation of the responses of the antagonistic neural subpopulations. This describes the steady-state behavior of the DNP. A brief description of its transient behavior is given below.

The limit cycle phenomenon of the DNP is briefly discussed in this section. The limit cycle oscillations are observed in response to a constant stimulation [41, 42]. Figure 6.5(a) shows the limit cycle oscillations for a constant excitatory stimulus $s_E(k)$ of 5 units, $s_I(k)$ set equal to zero. This simulation was carried out for various stimulus intensities, and the following points were observed: (i) a threshold value exists for the stimulus intensity which must be exceeded in order to evoke such oscillatory behavior, (ii) there is a higher value of the stimulus above which the system saturates and the limit cycle activity is extinguished, and (iii) between these two values, the frequency of oscillations increases monotonically with increasing stimulus intensity.

In general, the dynamic behavior of the neural processor depends upon the strength of the synaptic connections and the stimulus intensity as demonstrated in Figs. 6.5b and 6.5c.

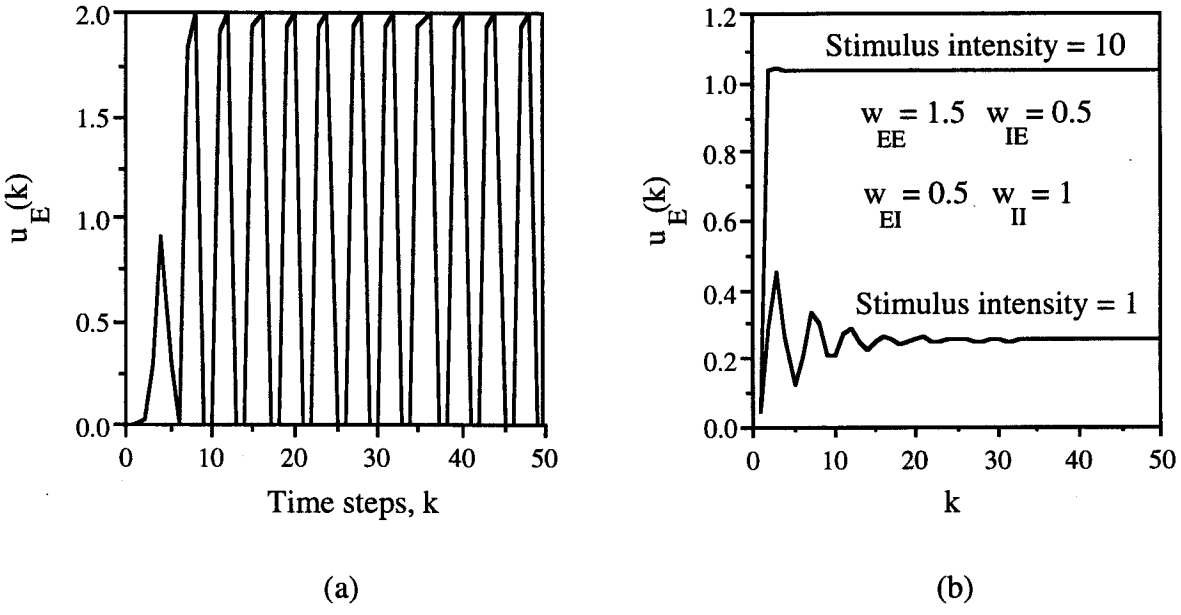


Figure 6.5: (Continued)

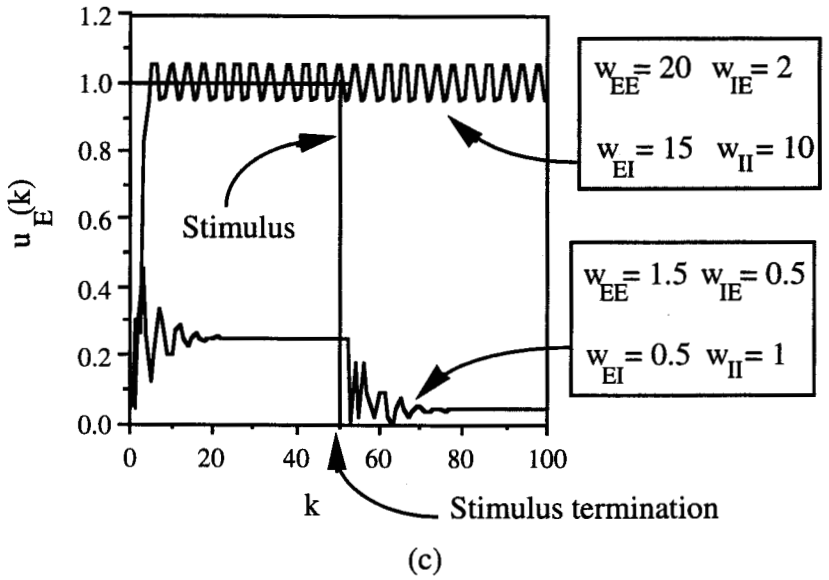


Figure 6.5: The response of a neural subpopulation.

(a) Temporal response of the excitatory neural unit to a stimulus with constant intensity. The parameters used in this simulation are shown in Fig. 6.5b,

(b) and (c): the dynamic behavior of the DNP for the various synaptic strengths and stimulus intensities.

This nonlinear characteristic of the processor can be employed to approximate nonlinear functions and to model some aspects of complex systems such as robots. In order to achieve the computational power of the DNP, it is necessary to develop a learning and adaptive algorithm to update the parameters of the DNP. An algorithm has been derived in Chapters 2 and 5 to modify the parameters of the DNU; in the following sub-section, an extension of this algorithm to other parameters, namely the self- and inter-subpopulation synaptic feedback-weights, of the DNP is derived.

6.2.3 Learning and Adaptive Algorithm

It is believed [24] that the connectivity strength, that is the neural weights, between the neural subpopulations changes as the brain learns to perform a new task. Due to the complexity, and the incomplete knowledge of the biological learning process, many concepts and algorithms have been developed in the field of neural networks to mimic the learning process of the biological neural networks [22 - 26, 44, 45]. One of the methods involves minimizing a performance index with respect to the weights of the neural network. Based on this principle, an algorithm to modify the DNU parameters has been developed earlier in this

thesis. An extension of this algorithm to other parameters of the DNP is briefly described in the following paragraphs.

Let the vector of the adjustable weights of the DNP be $\Omega_{(\mathbf{a}_{ff}, \mathbf{b}_{fb}, g_s, w_{\lambda\lambda'})}$ (represented here after as $\Omega_{(.)}$). The task of the algorithm is to modify each component of the vector $\Omega_{(.)}$ in such a way so as to minimize the performance index $J(\Omega)$ using the steepest-descent technique. This algorithm may be written as

$$\Omega_{(.)}(k+1) = \Omega_{(.)}(k) + \delta\Omega_{(.)}(k) \tag{6.7}$$

where $\Omega_{(.)}(k+1)$ is the new parameter vector, $\Omega_{(.)}(k)$ is the present parameter vector, and $\delta\Omega_{(.)}(k)$ is an adaptive adjustment in the parameter vector. In the steepest-descent method, the adjustment of the parameter vector is made proportional to the negative of the gradient of the performance index J; that is,

$$\delta\Omega_{(.)}(k) \propto (-\nabla J), \text{ where } \nabla J = \frac{\partial J}{\partial \Omega_{(.)}} .$$

$$\text{Thus, } \delta\Omega_{(.)}(k) = -\,\text{dia}[\mu]\,\frac{\partial J}{\partial \Omega_{(.)}} = -\,\text{dia}[\mu]\,\nabla J \tag{6.8}$$

where dia[μ] is the matrix of the individual adaptive gains. In the above equation, the dia[μ] is defined as

$$\text{dia}[\mu] \triangleq \begin{bmatrix} \mu_{a_i} & 0 & 0 & 0 \\ 0 & \mu_{b_j} & 0 & 0 \\ 0 & 0 & \mu_{g_s} & 0 \\ 0 & 0 & 0 & \mu_{\lambda\lambda'} \end{bmatrix} \tag{6.9}$$

where μ_{a_i}, i = 0,1,2, μ_{b_j}, j =1,2, μ_{g_s} are the individual learning gains of the adaptable parameters of the DNU, and $\mu_{\lambda\lambda'}$ denote the learning gains for the self- and inter-subpopulation synaptic weights. The parameters of the DNU are adjusted based on the following equations derived in the preceding chapter:

$$a_{ff_i}(k+1) = a_{ff_i}(k) + \mu_{a_i} E\left[e(k)\,\text{sech}^2[v(k)]\,\mathbf{P}_{ff_i}(k)\right] , \; i = 0,1,2, \tag{6.10a}$$

$$b_{fb_j}(k+1) = b_{fb_j}(k) + \mu_{b_j} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,\mathbf{P}_{fb_j}(k)\right],\ j = 1,2,\ \text{and} \tag{6.10b}$$

$$g_s(k+1) = g_s(k) + \mu_{g_s} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,v_1(k)\right] \tag{6.10c}$$

where $\mathbf{P}_{ff_i}(k)$ and $\mathbf{P}_{fb_j}(k)$ are the parameter-state signals of the DNU.

Similarly, equations to modify the self- and inter-subpopulation feedback weights can be obtained as follows. The gradient of the performance index with respect to these weights is given by

$$\frac{\partial J}{\partial w_{\lambda\lambda'}} = \frac{1}{2} E\left[\frac{\partial [y_d(k) - u(k)]^2}{\partial w_{\lambda\lambda'}}\right] = E\left[-e(k)\left\{\frac{\partial \Psi(v)}{\partial v}\,\frac{\partial v}{\partial w_{\lambda\lambda'}}\right\}\right]$$

$$= E\left[-e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_\lambda(k)\right\}\right]. \tag{6.11}$$

From Eqns. (6.7) and (6.11), the following equation may be written

$$w_{\lambda\lambda'}(k+1) = w_{\lambda\lambda'}(k) + \mu_{\lambda\lambda'} E\left[e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_\lambda(k)\right\}\right]. \tag{6.12}$$

For clarity, Eqn. (6.12) is written for the individual synaptic weights as

$$w_{EE}(k+1) = w_{EE}(k) + \mu_{EE} E\left[e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_E(k)\right\}\right] \tag{6.13a}$$

$$w_{IE}(k+1) = w_{IE}(k) + \mu_{IE} E\left[e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_I(k)\right\}\right] \tag{6.13b}$$

$$w_{EI}(k+1) = w_{EI}(k) + \mu_{EI} E\left[e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_E(k)\right\}\right] \tag{6.13c}$$

$$w_{II}(k+1) = w_{II}(k) + \mu_{II} E\left[e(k)\left\{\mathrm{sech}^2[v(k)]\,g_s\,u_I(k)\right\}\right] \tag{6.13d}$$

Equations (6.10a) and (6.10b) provide adaptation in the synaptic weights, while Eqn. (6.10c) and Eqn. (6.12) provide adaptation in the sigmoidal gain of the DNU and the external synaptic weights respectively. Using these equations, four applications of the DNP are discussed in the following section.

6.3 Applications of Dynamic Neural Processor

Although many applications of the DNP, in equalization of communication channels, robotics and control, short-term memory, and pattern recognition are possible, only some applications to robotics and control were emphasized in this study. In this context, four applications of the DNP are discussed. The first application, discussed in Section 6.3.1, involves the use of the DNP for the approximation of arbitrary nonlinear functions. A comparative study of the DNP with single- and two-layer recurrent neural networks is also discussed in this section. The next subsection details the use of the DNP for computing the inverse kinematic transformations of a two-link robot. This is followed in Section 6.3.3 by the application of the DNP to the control of some unknown nonlinear dynamic systems. Finally, the application of DNP to the coordination and control of multiple systems is discussed in the last section.

6.3.1 Functional Approximation

Although different theoretical bases and approaches are reported in the literature to show the functional approximation capabilities of neural networks, one clear feature is that neural networks have great promise in nonlinear system modeling and control. In this section, the performances of recurrent neural networks and the DNP as applied to the approximation of nonlinear functions are compared. The general learning scheme that is employed to achieve this objective is shown in Fig. 6.6. In the computer simulation studies discussed in this section, both the single- and two-layer recurrent neural networks shown in Figs. 6.7 and 6.8 respectively are considered. In many control applications, only single-layer recurrent neural networks have been employed. Much less has been reported regarding the performance of two-layer recurrent neural networks.

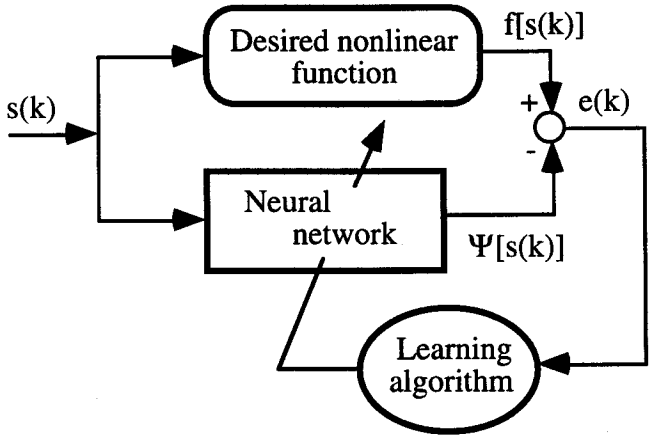


Figure 6.6: A general learning scheme for functional approximation using neural networks.

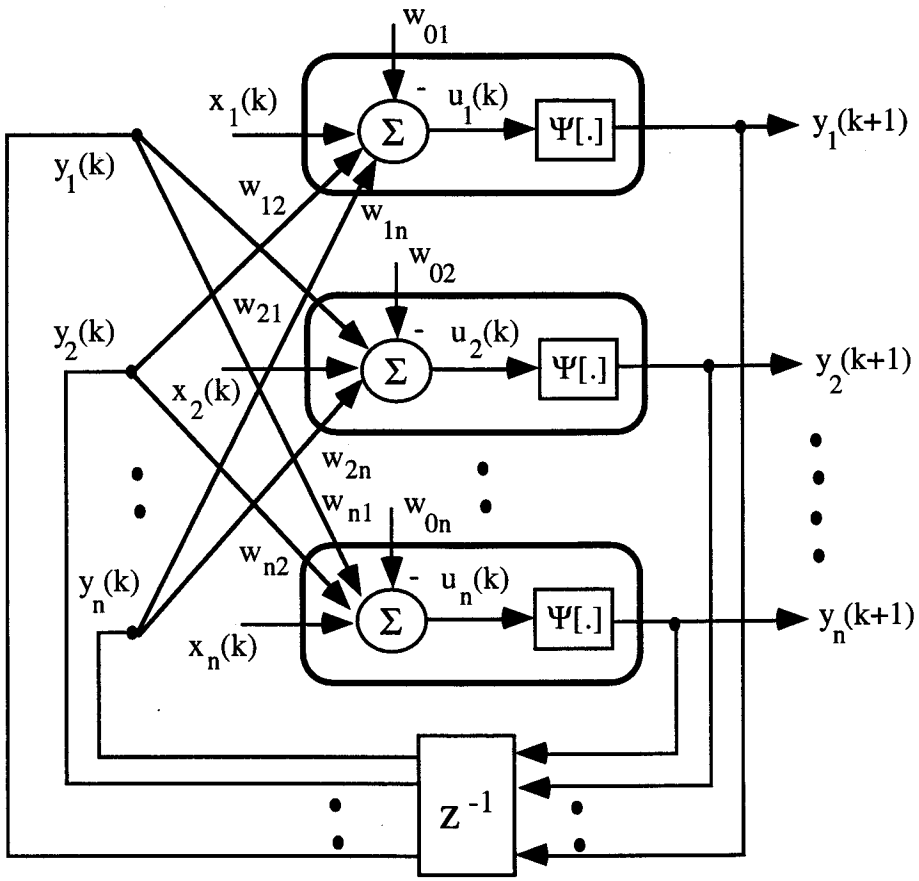


Figure 6.6: Single-layer recurrent neural network with no self-feedback connections.

In Fig. 6.6, the feedback input to the i-th neuron is equal to the weighted sum of neural outputs y_j, where j = 1, 2, ... , n. If w_{ij} is the weight value which connects the output of the j-th neuron to the input of the i-th neuron, the total input u_i of the i-th neuron can be expressed as

$$u_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij}\, y_j + x_i - w_{0i} \quad , i = 1, 2, \ldots , n. \tag{6.14a}$$

In vector form, Eqn. (6.14a) can be rewritten as

$$u_i = \mathbf{w}_i^T \mathbf{y} + x_i - w_{0i} \quad , i = 1, 2, \ldots , n \tag{6.14b}$$

where $\mathbf{w}_i \triangleq \begin{bmatrix} w_{i1} \\ w_{i2} \\ \cdots \\ w_{in} \end{bmatrix}$ and $\mathbf{y} \triangleq \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$.

The linear portion of the recurrent neural network can be described in matrix form as

$$\mathbf{U} = \mathbf{W}\,\mathbf{y} + \mathbf{X} - \mathbf{w_0} \tag{6.15a}$$

where $\mathbf{U} \triangleq \begin{bmatrix} u_1 \\ u_2 \\ \cdots \\ u_n \end{bmatrix}$, $\mathbf{X} \triangleq \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$ and $\mathbf{w_0} \triangleq \begin{bmatrix} w_{01} \\ w_{02} \\ \cdots \\ w_{0n} \end{bmatrix}$.

The matrix $\mathbf{W}$ in Eqn. (6.15a), called the *connectivity matrix*, is an $(n \times n)$ matrix and may be written as

$$\mathbf{W} = \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & 0 & w_{23} & \cdots & w_{2n} \\ w_{31} & w_{32} & 0 & \cdots & w_{3n} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ w_{n1} & w_{n2} & w_{n3} & \cdots & 0 \end{bmatrix}. \tag{6.15b}$$

This matrix is symmetrical, $w_{ij} = w_{ji}$, and with the diagonal entries equal to zero, $w_{ii} = 0$, indicating that no connection exists from any neuron back to itself. This condition is equivalent to there being no self-feedback in the neural structure shown in Fig. 6.7. As an extension of this structure, a two layer recurrent neural network, shown in Fig. 6.8, has been developed for bidirectional associative memory and pattern recognition applications [99].

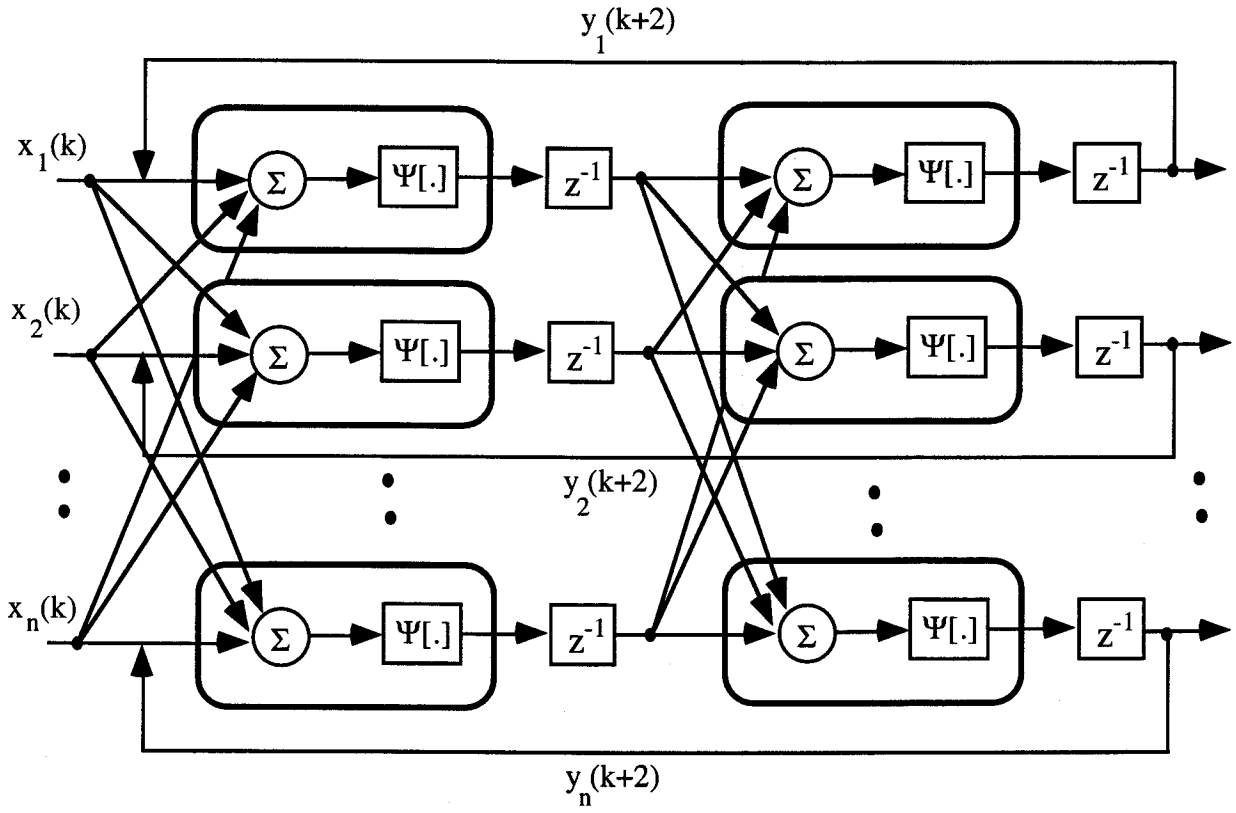


Figure 6.7: A two-layer recurrent neural network.

For performance comparison, a single-layer recurrent neural network with five neurons and a two-layer recurrent neural network, which consisted of two neurons in each stage as shown in Fig. 6.7, were considered in the simulation studies. The conventional backpropagation learning algorithm [100] was implemented to modify the neural network weights. The input used in these simulation studies was s(k) = sin ($2\pi k/250$) in the interval [-1,1]. The initial values of the synaptic connections of the DNP were arbitrarily set to w_{EE} = 1, w_{EI} = 0.5, w_{IE} = 0.5, w_{II} =1, and the components of the scaling vector, $\mathbf{w} = [w_E \; w_I]^T$, to 1. The parameters of the DNP, namely $\mathbf{a}_{ff}$, $\mathbf{b}_{fb}$, g_s and $w_{\lambda\lambda'}$, were adjusted based on the learning algorithm derived in the preceding section.

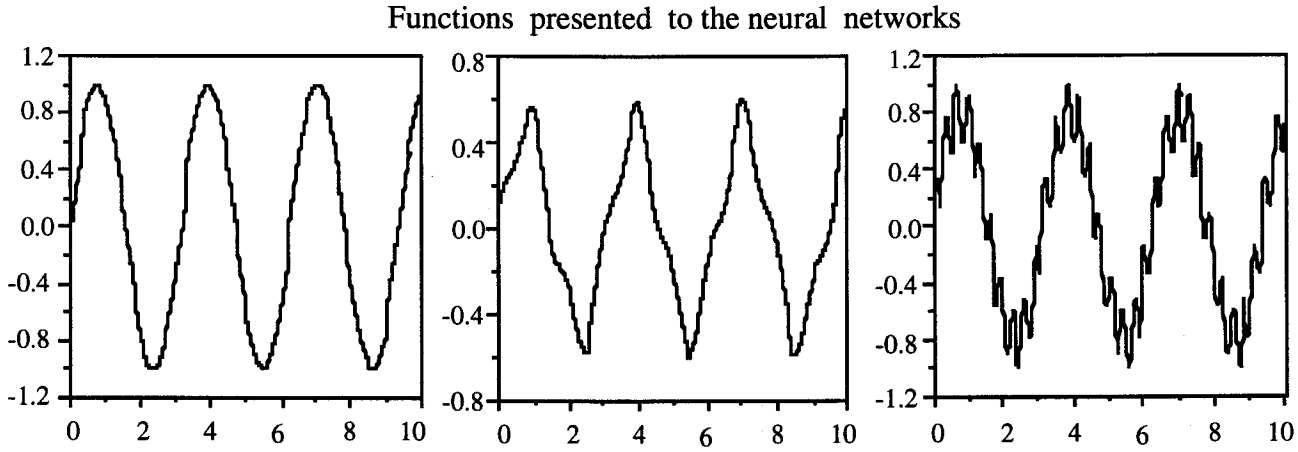
Different arbitrary nonlinear functions of s(k) were used to evaluate the performance of these dynamic neural structures. Some of the functions and their approximations are shown in Fig. 6.8. Figure 6.8a shows the desired nonlinear functions presented to the neural networks. The functional approximations obtained using the single- and two-layer recurrent neural networks are shown in Figs. 6.8b and 6.8c respectively, while Fig. 6.8c illustrates the functional approximation obtained using the DNP. The performance of each neural network

was observed for 1000 time intervals. The number of iterations required for each neural network to approximate the nonlinear functions is indicated in each figure. The functions shown in Fig. 6.9a can be mathematically represented as

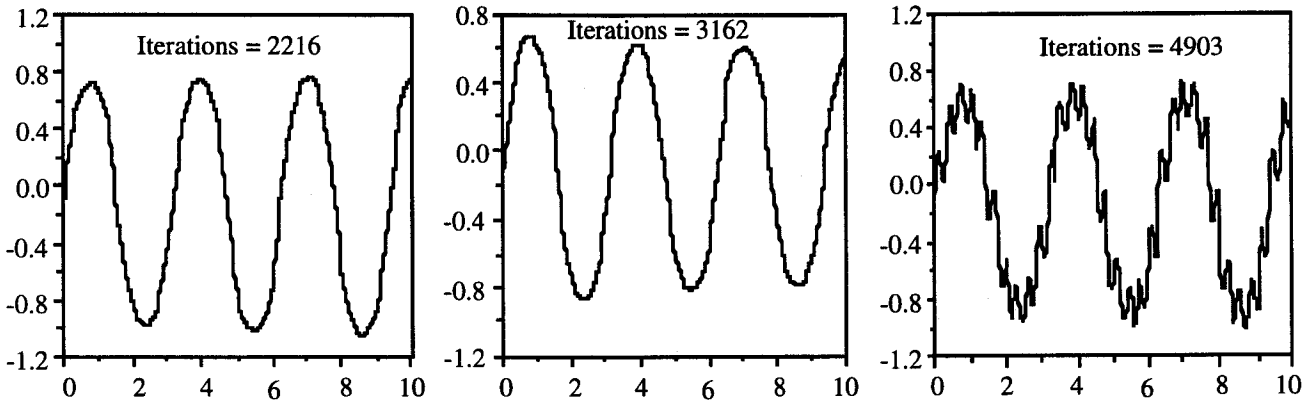
(i) $f[.] = s(k)$, (6.16a)

(ii) $f[.] = 0.5\, s(k) + 0.1 \cos(2\pi k/250)$, and (6.16b)

(iii) $f[.] = 0.8\, s(k) + 0.2 \sin(2\pi k/25)$. (6.16c)



6.9(a)



6.9(b)

Figure 6.9: (Continued)

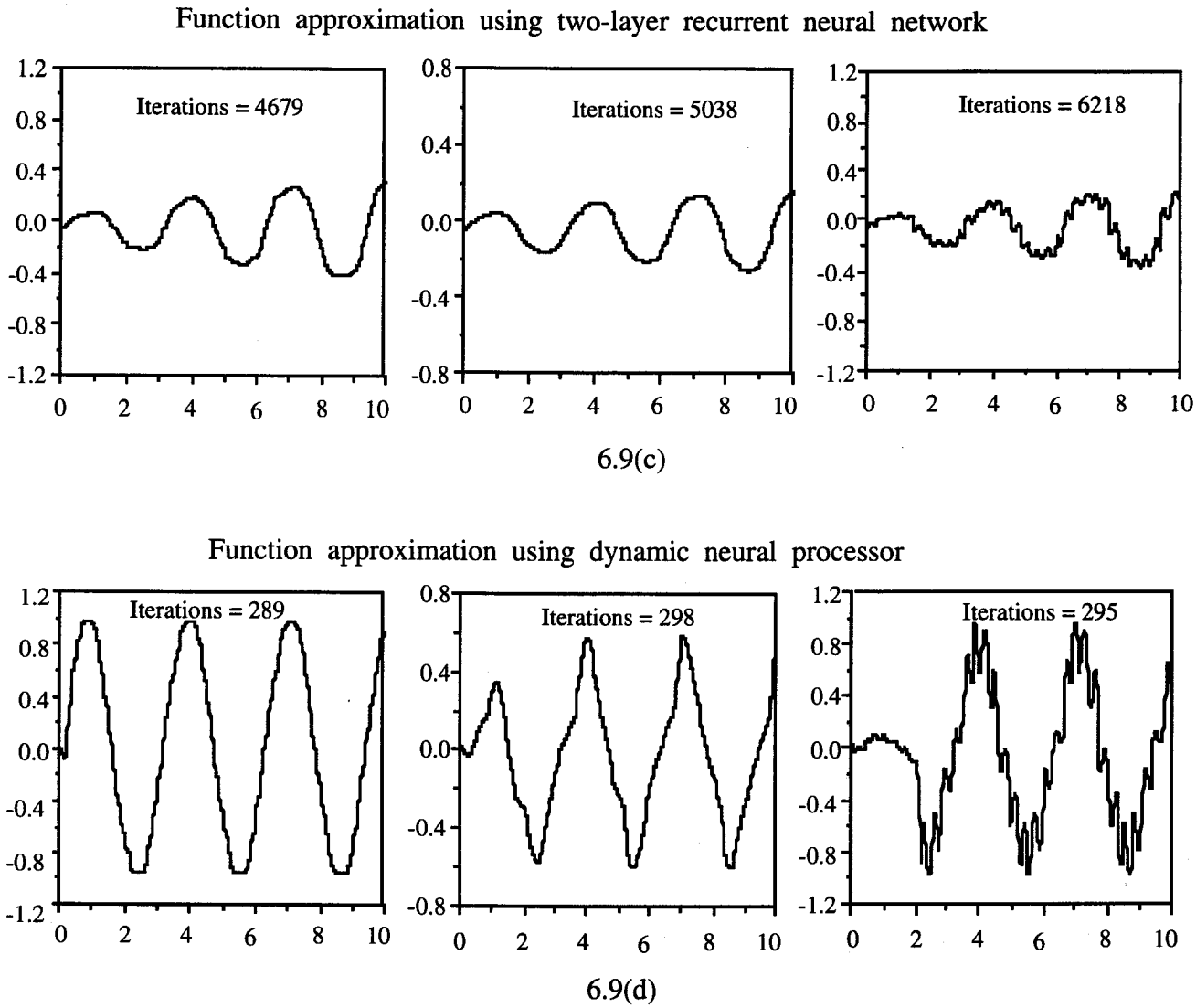


Figure 6.9: Nonlinear functions and their approximations using different neural networks.
(a) Arbitrary nonlinear functions applied to neural networks for performance comparison,
(b) Functional approximation using a single-layer recurrent neural network,
(c) Functional approximation using a two-layer recurrent neural network, and
(d) Functional approximation using the DNP.

Figure 6.10 shows some complex nonlinear functions and their approximations achieved by the DNP. The performance of the recurrent neural networks for these functions was very poor in terms of both the accuracy of approximation and the speed of convergence. The nonlinear functions shown in Figs. 6.10a and 6.10b can be represented mathematically as

$$f[.] = s^3(k) + 0.3\sin(2\pi s(k)) + 0.1\sin(5\pi s(k)) \tag{6.16d}$$

$$f[.] = \sin\left[\pi\left(s^2(k) + 0.3\right)\right] + \frac{0.3\sin(2\pi s(k))}{1+s^2(k)} \quad . \tag{6.16e}$$

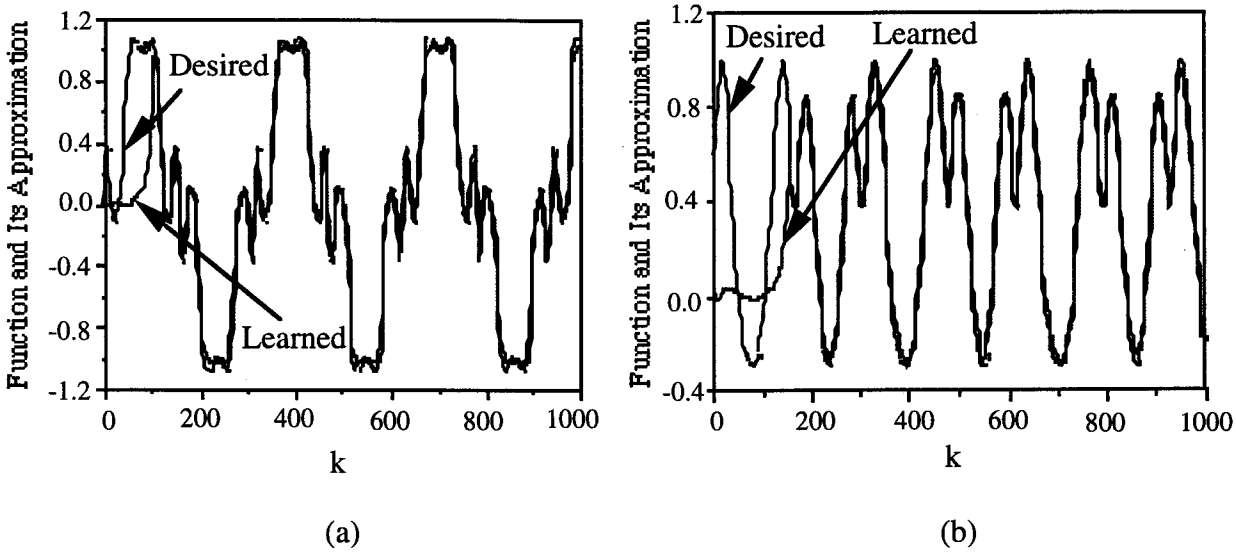


Figure 6.10: Arbitrary nonlinear functions and their approximations using the DNP.

Extensive simulation studies were conducted for various nonlinear functions. From these studies and from the simulation results presented in this chapter, it was observed that the DNP could approximate arbitrary nonlinear functions much more quickly than the recurrent neural networks which was evident from the number of iterations required for each neural network. The speed of convergence of the recurrent neural networks depended upon the functions being approximated. On the other hand, the performance of DNP was found to be almost independent of the functions being approximated. A single-layer recurrent neural network was found to be faster than its two-layer counterpart. A plausible explanation for the slow convergence of the multi-layer recurrent neural network is that the delay operators were employed in the forward path of the information flow, and the feedback signals arrived from the last layer to the input nodes. A detailed comparative study of the recurrent neural networks and the DNP is reported in [101]. These initial findings imply that the greater the number of layers, the slower the convergence of the recurrent neural networks to the desired function. However, more work needs to be done to generalize this characteristic of recurrent neural networks.

In order to compare the performance of the recurrent neural networks and the DNP under noisy conditions, nonlinear functions represented in Fig. 6.9a were corrupted with different levels of noise. Figure 6.11 shows the performance of the recurrent neural networks and the DNP in terms of the percentage of the convergence error with respect to the percentage of the noise level in the functions. These observations were made for a time interval of 1000 iterations.

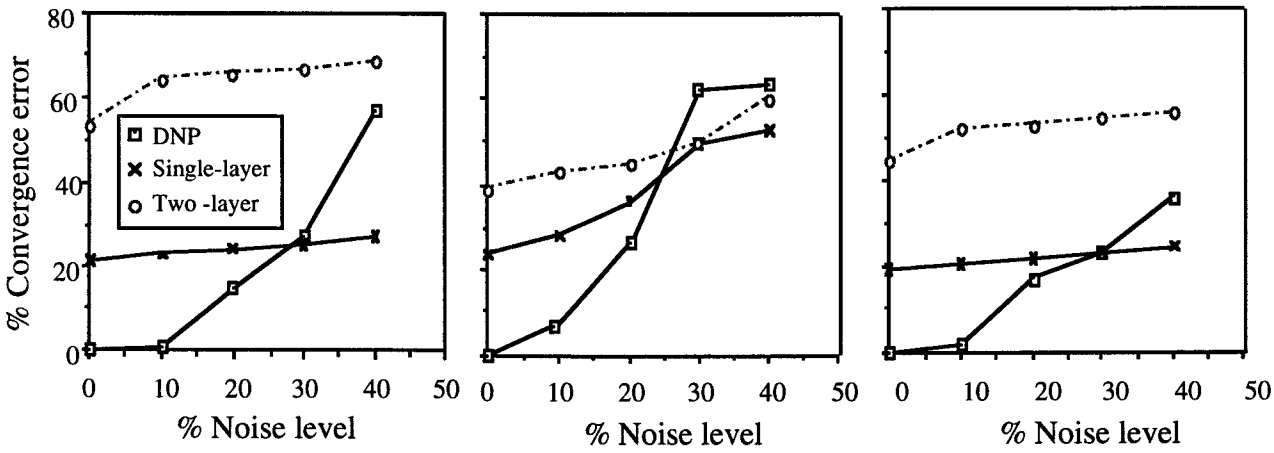


Figure 6.11: Performance comparison of dynamic neural networks under noisy conditions.

Figure 6.11 shows that as the noise level in the functions that were approximated was increased gradually the performance of the DNP was degraded compared to the recurrent neural networks. The recurrent neural networks were found to be more noise tolerant than the DNP. This demonstrates the limitation of the DNP which was developed with only one DNU in each neural subpopulation. The performance of the DNP may improve under noisy conditions if the neural subpopulations contain more neurons.

6.3.2 Neural Learning of Robot Inverse Kinematic Transformations

6.3.2.1 Neural Networks in Robotics

Advances in the area of neural networks have given a different direction to robotic control. By virtue of their functional mapping and iterative capabilities, neural networks can be employed for learning coordinate transformations [101 - 103]. The advantage of using the neural approach over the conventional inverse kinematics algorithms is that neural networks can avoid time consuming calculations. The features of neural networks, such as learning, adaptation, fault-tolerance and parallelism, provide strong incentives for choosing them to compute inverse kinematic transformations in the field of robotics.

Neural networks, because of their parallel and distributed computational abilities, have the ability to learn associations between patterns. These patterns could represent, for example, the task space coordinates and the corresponding joint angles of the model leg. The association between these two sets of patterns basically amounts to inverse kinematics computations in robotics [104, 105]. Neural networks have an advantage over the traditional inverse kinematics algorithms in that the neural networks can 'learn' the transformation through examples. This would avoid time consuming numerical calculations and provide, more or less, instant recall. Furthermore, in a manner that is typical of neural networks, it would be very easy to modify the learned associations as the structure of the mechanism changes. It is advantageous, therefore, to employ neural networks for learning the inverse kinematics transformation of robots. In this section, the recurrent neural network and the DNP are employed to obtain the inverse kinematics transformation of a two-link robot.

In the context of the above observations, consider the two-link robot shown in Fig. 6.12a. The joints at which the rotary motion occurs (within limits) are analogous to the hip and the knee joints of the human leg. The point P(x,y), the free tip of the second link, also called the 'end point', describes the end-effector trajectories based on a Cartesian coordinate system. The origin of the coordinate system is the first (hip) joint, which is assumed to be fixed in space, while the end point coordinates (x, y) are located with respect to the two perpendicular axes, X, Y. The hip joint is considered as the anchor (fixed) point. The position of the leg can also be restricted using the angles formed at the two joints with the reference axes as shown. The relationship between these two angles, defined as θ_1, θ_2, and the end point coordinates, x and y, form the kinematic equations of the two-link leg. Specifically, the coordinates x and y are defined as

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2), \text{ and}$$

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2). \qquad (6.17)$$

These two equations are the 'forward' kinematic equations of the model leg. For the lengths (L_1, L_2), the point coordinates (x, y) of the end-effector are uniquely determined by the two variable joint angles (θ_1, θ_2). The inverse relationships, namely, the definition of the joint angles with respect to the coordinates are

$$\theta_2 = \tan^{-1}\left[\frac{S2}{C2}\right]; \text{ where } S2 = \sin(\theta_2) = \pm\sqrt{1-C_2^2} \text{ and}$$

$$C2 = \cos(\theta_2) = \left[\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1 L_2}\right],$$

$$\theta_1 = \tan^{-1}\left[\frac{y}{x}\right] - \tan^{-1}\left[\frac{L_2 S_2}{L_1 + L_2 C2}\right]. \qquad (6.18)$$

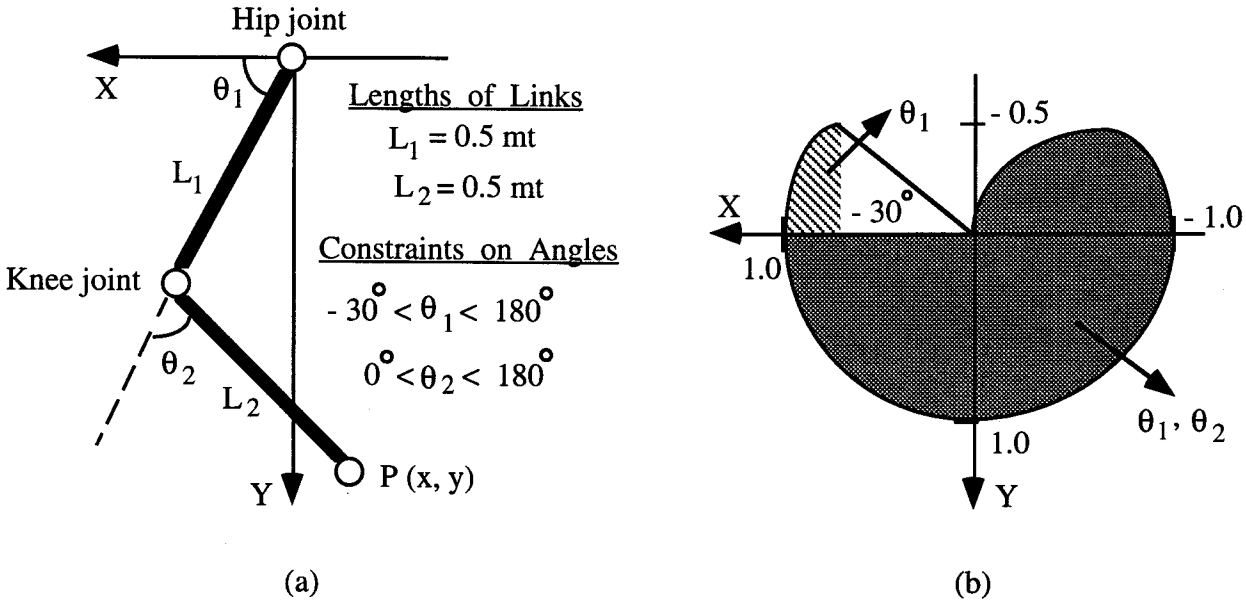


Figure 6.12: A two-link robot as a model of the human leg.
(a) An illustration of the two-linked model leg with the joint angles θ_1 and θ_2 in a two-dimensional task space.
(b) Constraints on the two-dimensional task space of the model leg.

The equations in θ_1 and θ_2 are called the 'inverse' kinematic equations and are nonlinear because of the trigonometric functions and the squared terms. The periodicity and symmetry of the tangent function and the multiple roots of the squared terms result in an ambiguous determination of the end point location. The problem becomes more severe as the number of links increases. One way of circumventing the problem of multiple solutions is to constrain the movement of the two links to certain convenient angular ranges which will usually avoid the occurrence of multiple solutions [104]. In this regard, it is to be noted that the structure of the human leg, with its hinge-like joint at the knee, permits only a constrained motion of the shank. Taking into consideration the above constraints, the two joints of the two-link leg model were constrained to move within the specific angular ranges:

$-30° < \theta_1 < 180°$, and $0° < \theta_2 < 180°$. The introduction of these constraints results in a two-dimensional task space as shown in Fig. 6.12b.

The standard methodology for computing inverse kinematic transformations employs training the neural network off-line for possible data patterns within the robot task space to obtain solutions to the inverse kinematics problem. Because of the generalization property, neural networks can learn the associated patterns and recall the learned patterns. The trained network is then used to achieve the desired voluntary movements. This technique, therefore, basically involves two modes of operations, namely the training phase and the performing phase. However, the major drawback of this technique is a very long training procedure in addition to the fact that the static neural networks based on the backpropagation learning algorithms require a very large convergence time.

An on-line learning scheme for computing the inverse kinematic transformations was proposed by Gupta and Rao [107]. This learning scheme, shown in Fig. 6.13, uses the neural network to determine the joint angles for a given set of desired Cartesian coordinates. These estimated joint angles, which act as inputs to the forward kinematics, are checked against the pre-defined robot task space. This additional level of control makes the robot operate within a specified work-space. Additional rules and inferences may be incorporated into the first-level thereby making it a knowledge-based robotic control system [1]. Although the computation of the inverse kinematics transformations is a static problem, the use of dynamic neural networks generally decreases the time required to compute the transformations compared to that of the static (feedforward) neural networks.

6.3.2.2 Computer Simulation Studies

The desired x-y positions of the end-effector were applied to the excitatory and inhibitory neural units of the DNP. The initial values of the synaptic connections were arbitrarily set to $w_{EE} = 1$, $w_{EI} = 0.5$, $w_{IE} = 0.5$, $w_{II} = 1$, and the components of the scaling vector, $[w_E \;\; w_I]^T$, to 1. The initial position of the end-effector was set arbitrarily at $x = 0.2$ and $y = 0.4$. In this section, a brief comparative study of the recurrent neural network and the DNP applied to the inverse kinematic computations of a two-link robot is provided. The recurrent neural network used in simulation studies consisted of five neurons configured in a single-layer as depicted in Fig. 6.7. Five simulation examples are discussed in the following paragraphs.

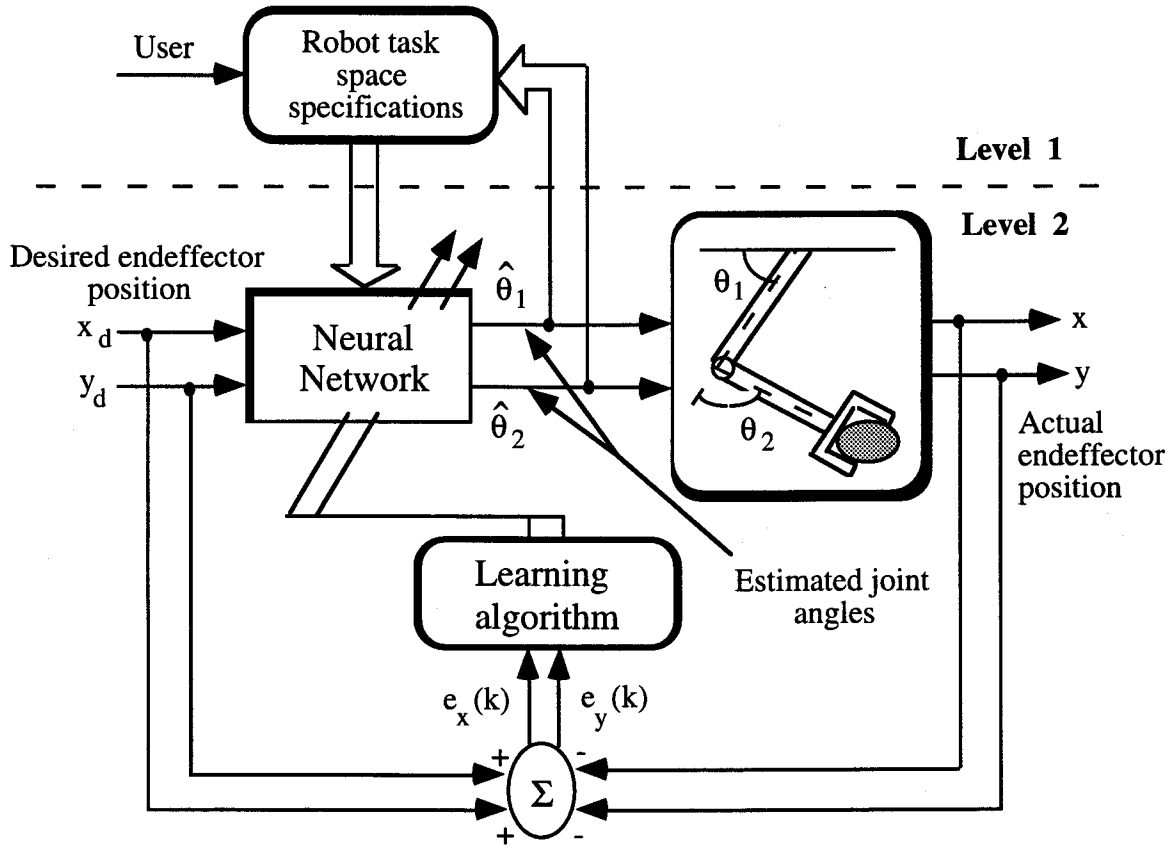


Figure 6.13: The learning scheme, with two hierarchical levels, for on-line learning of inverse kinematic transformations.

Example 1: In this example, the x and y coordinates of the end-effector were selected at random and applied to the processor. The neural weights were adjusted until the output error decreased to a pre-determined value of 0.05. Figure 6.14a shows the actual and the learned x-y coordinates of a two-linked robot, while Fig. 6.14b shows the trajectories of the X-Y coordinates and the corresponding joint angle trajectories for one position of the end-effector.

Different end-effector positions were presented to the neural processor. The results obtained are shown in Fig. 6.15. In this figure, P_1 and P_2 are the two out-of-reach positions of the end-effector. The neural processor could not learn these positions because of the pre-defined robot task-space, but as can be seen from the results, the leg (link) orientations were in that direction. One may note that for some of the 'out-of-reach' inputs, the processor's corresponding outputs were located within the task-space in such a way as to indicate the leg's intention to reach out to those points.

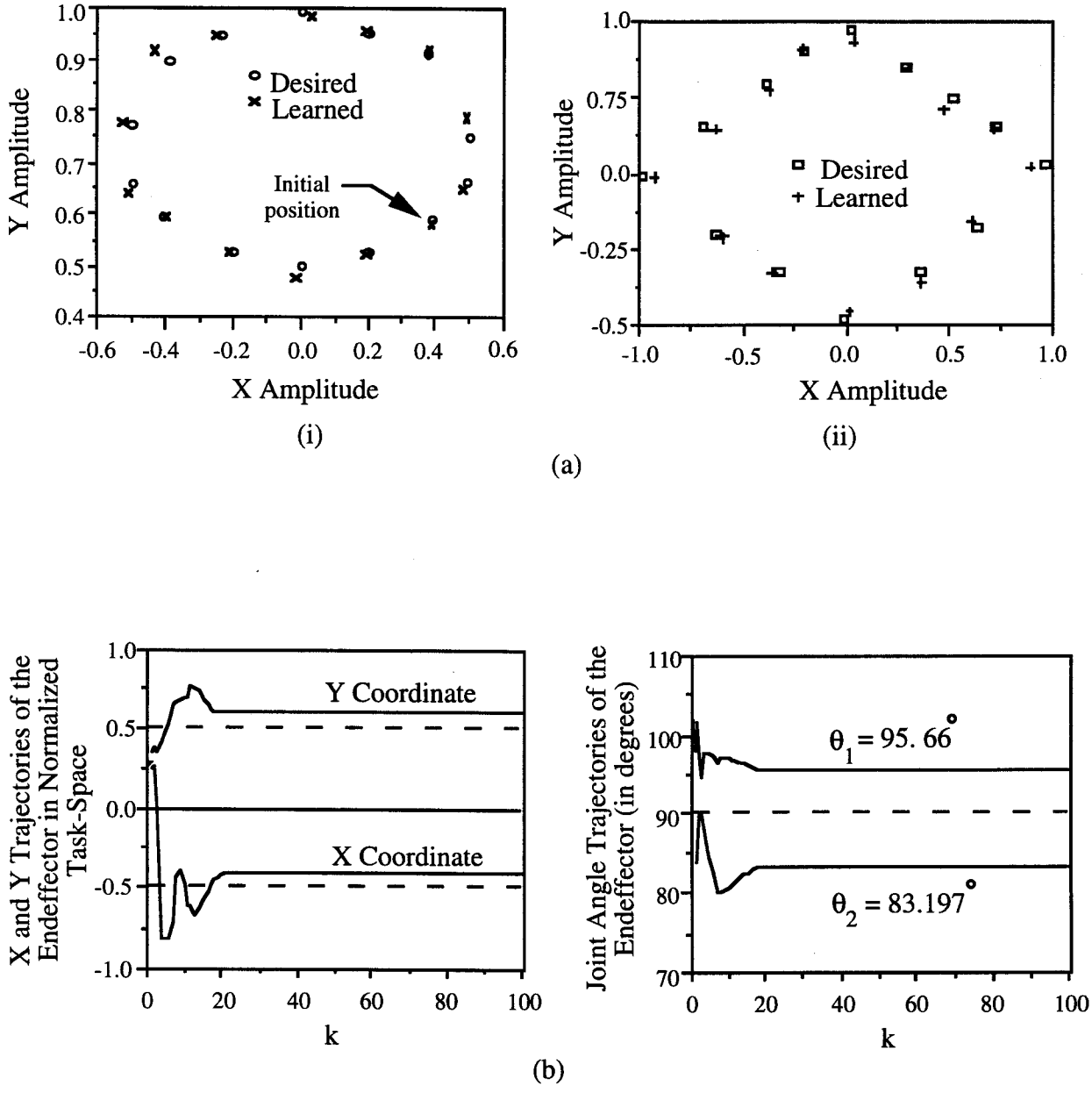


Figure 6.14: Simulation results, Example 1.

(a) Illustration of the actual and the learned positions of different trajectories.

(b) Trajectories of the end-effector's X and Y coordinates for a desired position of $x_d = -0.4$, $y_d = 0.6$, and the corresponding joint angle trajectories.

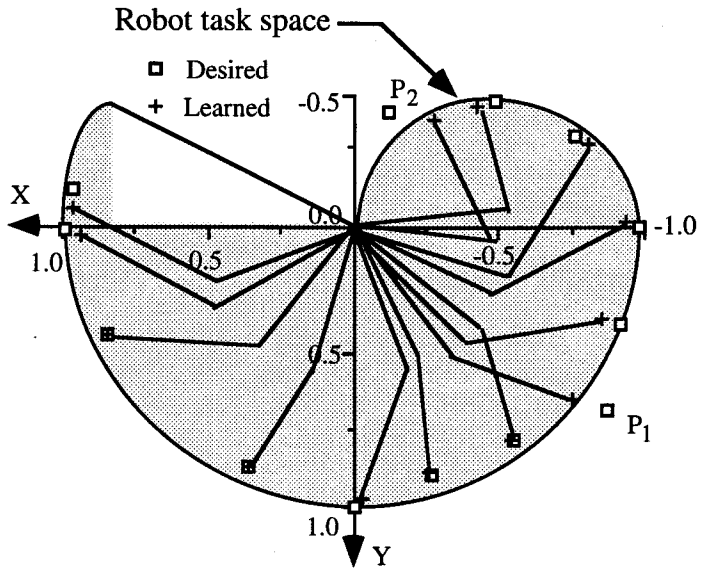


Figure 6.15: Representation of the actual and the learned positions in- and out-side of the task-space in a sagattial plane.

Example 2:

Case (i): In this example, a single-layer recurrent neural network consisting of five neurons was used to compute the inverse kinematic transformations. The desired position of the end-effector x-y coordinates were selected to be the same as in Example 1, and were applied to the first two neurons of the neural network. The neural weights of the neural networks were adjusted based on the backpropagation learning algorithm. The trajectories of the x and y coordinates is shown in Fig. 6.16. From this figure, it is clear that the computing time required to achieve the desired end-effector position using the recurrent neural network was very high compared to the DNP.

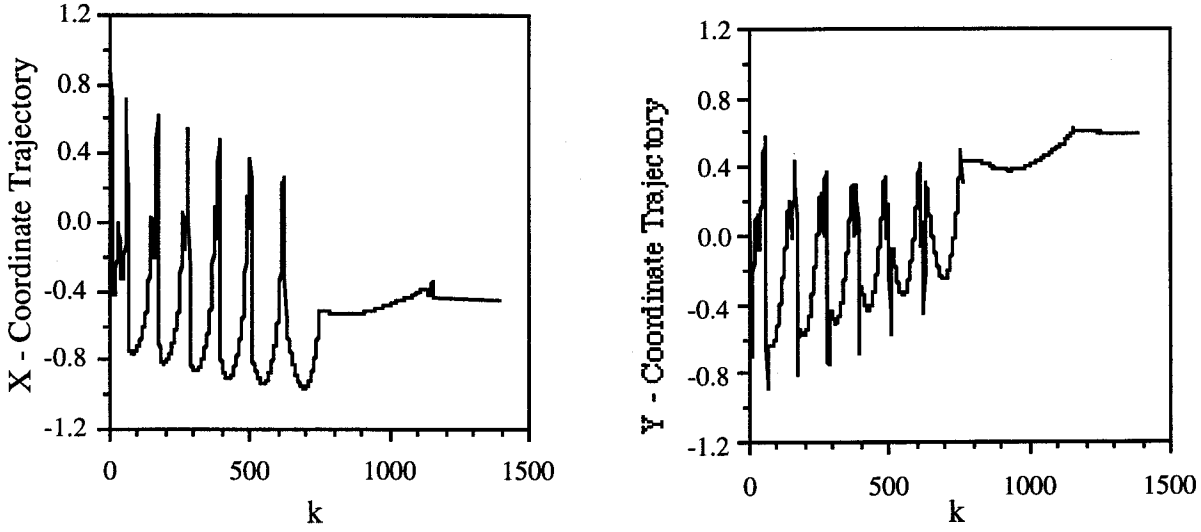


Figure 6.15: Convergence results obtained using recurrent neural network, Example 2.

Case (ii): The purpose of this simulation study was to compare the performance of the recurrent neural network and that of the DNP. Different end-effector positions were presented to the two neural structures. The accuracy and the speed of convergence obtained are tabulated in Table 6.1. These results reinforce the results obtained in Example 1 in that the DNP provided much faster convergence compared to the recurrent neural network.

Example 3: To study the performance of neural structures under noisy conditions, the robot dynamics were corrupted with a random signal bounded in the interval [0,1]. Tables 6.2 and 6.3 show the targeted and the observed end-effector positions for 20% and 50% noise respectively. These results show that both of the neural structures could accurately learn the desired patterns in the presence of noise. However, it was observed that as the noise level increased, the accuracy of the end-effector positions from the recurrent neural network was higher than that obtained from the DNP. This is possibly due to the fact that the noise signal corrupted the DNU parameters and inter-feedback synaptic weights $(w_{EE}, w_{II}, w_{IE}, w_{EI})$. Since the recurrent neural network has a static feedforward and hard (non-adaptable) feedback paths, the probability of the noise signal corrupting the weights was less. However, this statement is only speculative, and needs further investigation.

Table 6.1: Performance comparison of recurrent neural network and DNP

Desired Coordinates		Recurrent Neural Network			Dynamic Neural Processor		
x_d	y_d	x	y	Learning Iterations	x	y	Learning Iterations
- 0.6	0.5	- 0.604	0.51	4218	- 0.602	0.51	422
- 0.5	0.75	- 0.5	0.753	991	- 0.504	0.753	104
- 0.38	0.58	- 0.388	0.587	1821	- 0.377	0.579	101
- 0.2	0.53	- 0.196	0.52	1112	- 0.209	0.533	80
0.0	0.5	0.02	0.51	796	0.07	0.499	144
0.5	0. 75	0.503	0.74	1269	0.492	0.744	319
0.3	0.85	0.29	0.86	802	0.302	0.86	199
0.2	0.95	0.21	0.94	539	0.192	0.952	174
0.0	0.99	0.009	0.987	1095	0.09	0.993	164
- 0.25	0.9	- 0.248	0.897	655	- 0.258	0.903	36
- 0.3	0.8	- 0.296	0.79	2033	- 0.309	0.793	31
- 0.6	0.6	- 0.59	0.594	1255	- 0.601	0.59	137
- 0.2	0.8	- 0.21	0.81	843	- 0.207	0.812	36
0.38	0.58	0.38	0.59	290	0.378	0.579	31
- 0.8	- 0.2	- 0.798	- 0.21	470	- 0.792	- 0.195	69
- 0.3	- 0.45	- 0.304	- 0.44	2130	- 0.312	- 0.45	382
0.5	0.4	0.498	0.41	251	0.492	0.404	53
0.3	0.7	0.306	0.691	166	0.297	0.694	24
0.0	0.96	0.02	0.97	1620	0.08	0.961	60

Table 6.2: 20% Noise

Desired Coordinates		Recurrent Neural Network			Dynamic Neural Processor		
x_d	y_d	x	y	Learning Iterations	x	y	Learning Iterations
- 0.2	0.53	- 0.202	0.534	1236	- 0.205	0.523	169
0.38	0.58	0.375	0.586	352	0.378	0.571	31
0.2	0.95	0.193	0.95	164	0.196	0.954	122
- 0.25	0.9	- 0.243	0.893	1298	- 0.259	0.904	36
- 0.6	0.6	- 0.593	0.604	1375	- 0.609	0.608	335
- 0.2	0.8	- 0.199	0.802	1001	- 0.207	0.802	36
0.5	0.4	0.504	0.404	269	0.492	0.405	53
0.6	0.3	0.602	0.301	377	0.61	0.292	69
0.3	0.7	0.307	0.693	180	0.297	0.695	24
0.0	0.96	0.09	0.97	2512	0.09	0.963	282

Table 6.3: 50% Noise

Desired Coordinates		Recurrent Neural Network			Dynamic Neural Processor		
x_d	y_d	x	y	Learning Iterations	x	y	Learning Iterations
- 0.2	0.53	- 0.203	0.537	2919	- 0.21	0.52	866
0.38	0.58	0.381	0.586	651	0.371	0.571	54
0.2	0.95	0.206	0.953	304	0.19	0.96	210
- 0.25	0.9	- 0.241	0.904	1343	- 0.26	0.904	46
- 0.6	0.6	- 0.62	0.605	1736	- 0.5	0.7	335
- 0.2	0.8	- 0.208	0.805	2594	- 0.207	0.802	36
0.5	0.4	0.495	0.405	606	0.482	0.405	98
0.6	0.3	0.604	0.295	606	0.62	0.284	100
0.3	0.7	0.298	0.708	216	0.297	0.695	24
0.0	0.96	0.02	0.966	3382	0.101	0.86	1282

Example 4: The successful operation of an intelligent robot depends upon its ability to cope with perturbations that may cause dynamic changes in its structure. Such a case is considered in this example where one of the links, L_2, of the robot undergoes a stretching effect during the learning process. Due to these dynamic perturbations, the observed end-effector position may not match the desired position which necessitates readjustments in the neural weights. Due to the adaptive capability of the neural network-based learning schemes, the DNP could modify its weights so that the desired end-effector position was achieved. Computer simulation studies were carried out for the various dynamic perturbations. The simulation results for one such perturbation in the form of error trajectories of the robot links are shown in Figs. 6.17a and 6.76b. From these results it can be seen that the DNP adapted to the change in robot dynamics, thereby demonstrating the robustness of the learning scheme.

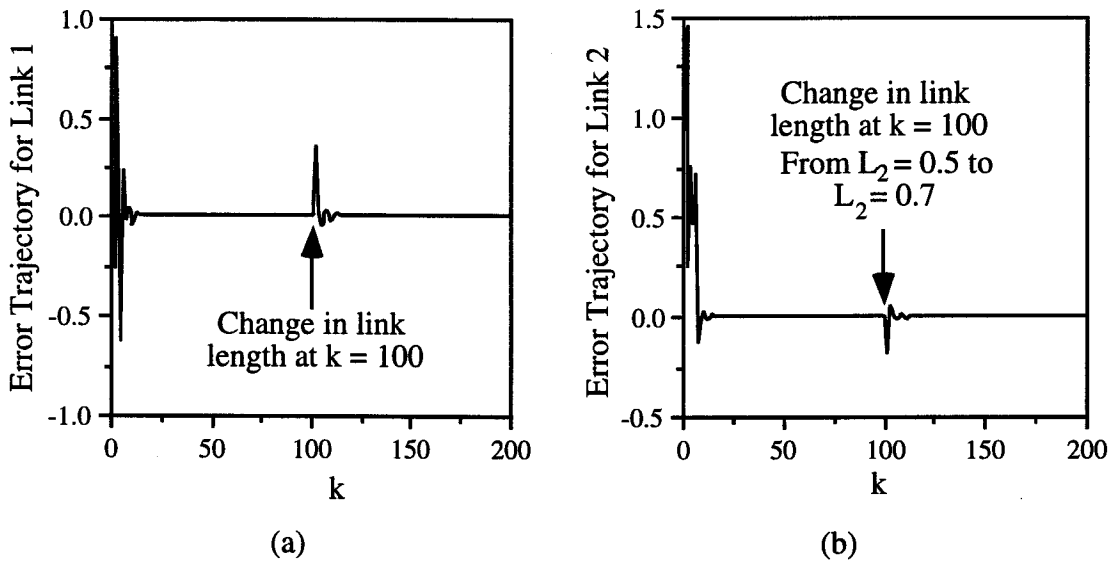


Figure 6.17: Error trajectories of robot links when the length of link L_2 was changed from 0.5 to 0.7 units at time instant k = 100, Example 4.

Example 5: In this example, the adaptive capability of the DNP was demonstrated by changing the desired end-effector positions during the learning process. Initially, the desired end-effector locations presented to the neural processor were: $x_d = 0.3$ and $y_d = 0.7$. At time instant, k = 75, the values were changed to $x_d = -0.8$ and $y_d = -0.2$. The neural processor learned this new pattern as shown in Fig. 6.18.

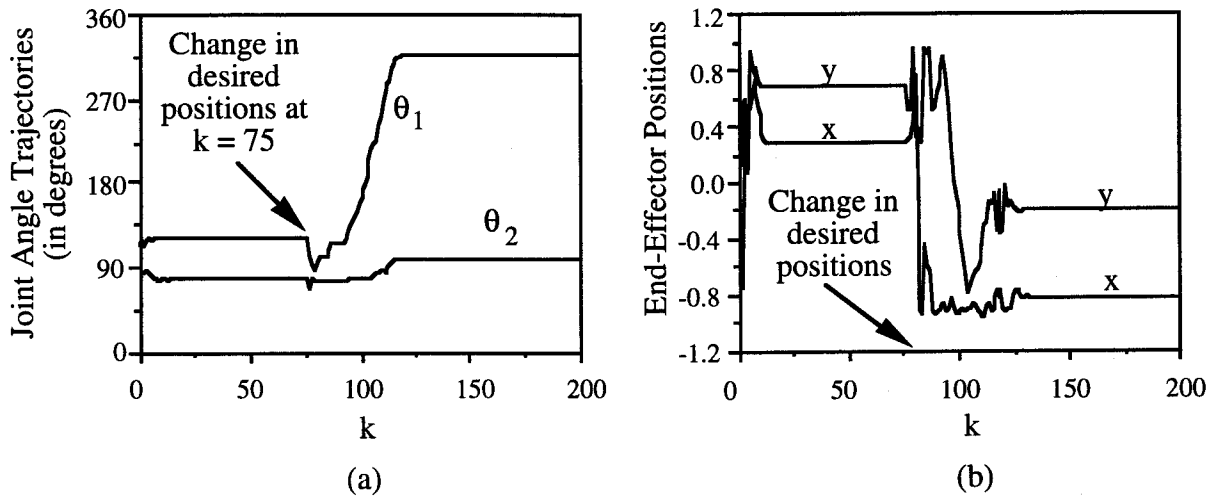


Figure 6.18: Simulation results, Example 5.

(a) Adaptation in joint angle trajectories when the desired end-effector positions were changed from x = 0.3 and y = 0.7 to x = - 0.8 and y = - 0.2 at time instant k = 75,

(b) The corresponding variations in x-y coordinates of the end-effector.

These simulation results indicate that the DNP, by virtue of its functional mapping capabilities, can be suitably employed for learning the coordinate transformations. Compared to the conventional analytical schemes which need intensive computing, the DNUs in the neural processor provide the required transformation very quickly. In the event that it is impossible for the model leg or robot to reach points within its task space, due to some physical limitations, the DNP can be trained to move to the nearest point still within its domain. Any modifications occurring in the structure of the robot can easily be taken care of by continuous learning. This reflects the adaptive nature of the neural network-based learning schemes. On the other hand, the conventional approach would have involved solving new potentially difficult transformations theoretically, generating new software to implement the new transformations, and then installing it in the new robot controller.

6.3.3 Control of Unknown Nonlinear Systems: Simulation Studies

A large number of neural network structures have been proposed and used for control applications. Broadly, these control schemes can be classified into two groups, (i) indirect adaptive control and (ii) direct adaptive control [34]. In the indirect adaptive control technique [34, 84 - 86], the neural network is trained first to attain the same dynamic behavior as the controlled plant, and a controller is then designed using the neural network's outputs to cancel the nonlinear part of the controlled plant [63]. In the direct adaptive control

technique [28, 39, 54, 58, 61, 63, 87, 95, 109], the neural network is cascaded with the controlled system, and the weights are adjusted based on the system output error. Both techniques have advantages and disadvantages. The problems associated with these techniques have been addressed in [28, 63].

In this section, the DNP was used to directly control unknown nonlinear systems. For this application, the command signal was applied to the excitatory unit, and the delayed plant output was fed back to the inhibitory unit serving as a feedback signal. This control scheme is shown in Fig. 6.19. The choice of applying the input and feedback signals to excitatory and inhibitory neurons was arbitrary however. The DNP settings were arbitrarily set to $w_{EE} = 1.0$, $w_{EI} = 0.5$, $w_{II} = 1.0$ and $w_{IE} = 0.5$. The components of the scaling vector, $\mathbf{w} = [w_E \; w_I]^T$, were set to [1, -1].

Example 1: In this simulation example, the nonlinear plant under control was assumed to be governed by the difference equation

$$y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i \, y(k-i) + \sum_{j=0}^{2} \beta_j \, u(k-j) \right], \; i = j = 0, 1, 2 \tag{6.19a}$$

with an arbitrary unknown function of the form

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{\left[1 + u^2(k-1) + u^2(k-2)\right]} \tag{6.19b}$$

and the plant parameters $\beta_{ff} = [1.2, \; 1, \; 0.8]^T$ and $\alpha_{fb} = [1, \; 0.9, \; 0.7]^T$. The input to the system (the desired response) used in this simulation was $s(k) = \sin(2\pi k / 250)$ in the interval [-1, 1]. The error and output responses are shown in Fig. 6.20. From the error response it is seen that the error between the target and the observed trajectories was initially large, but converged quickly within the pre-set tolerance limits.

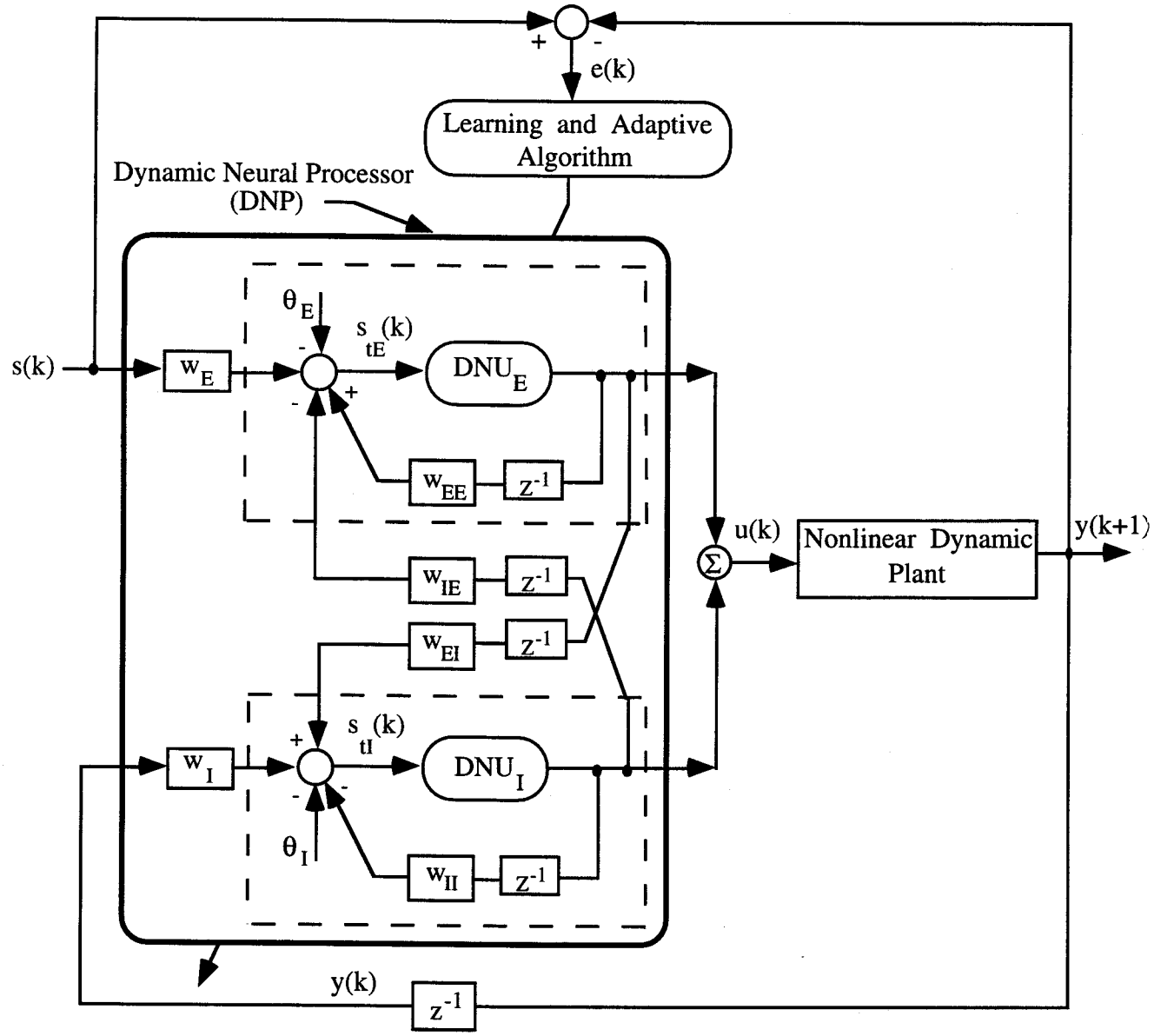


Figure 6.19: The control scheme used for controlling unknown nonlinear dynamic systems using the DNP where the reference input s(k) is used as the target response for nonlinear systems to track.

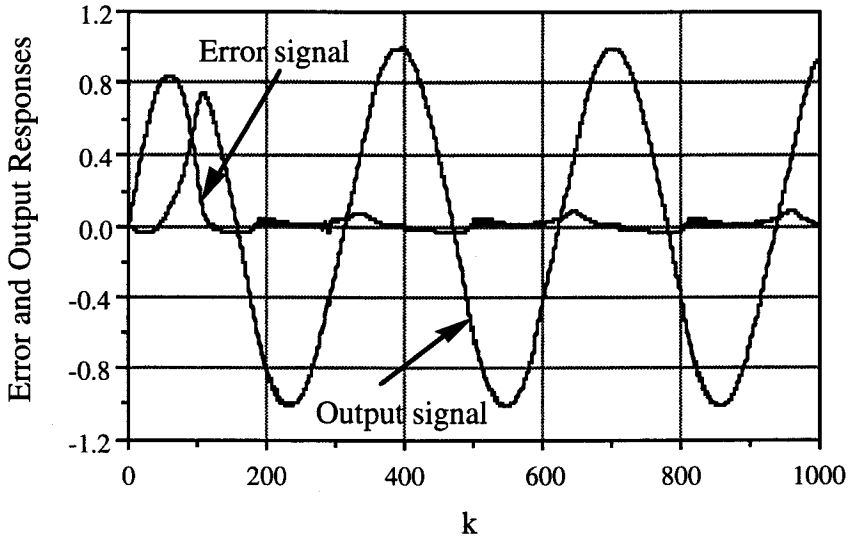


Figure 6.20: The error and output responses of a nonlinear plant represented by Eqn. (6.19), Example 1.

Example 2: Following the initial training described in Example 1, a nonlinear plant described by the following equation

$$y(k+1) = f\left[\sum_{i=0}^{2} \alpha_i\, y(k-i) + \sum_{j=0}^{2} \beta_j\, u(k-j) \right] + \sum_{i=0}^{2} g[y(k-i)]\, u(k) \qquad (6.20)$$

was considered. The input signal, plant parameters and DNP settings were the same as in Example 1. The nonlinear functions f[.] and g[.] in Eqn. (6.20) were as follows:

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{\left[1 + u^2(k-1) + u^2(k-2)\right]}\ , \text{ and}$$

$$g[.] = \sqrt{\left|\left\{\ y^2(k) + y^2(k-1)\right\}\right|}\ .$$

The error and the output responses obtained for this simulation example are shown in Fig. 6.21. In this case, it required about 2000 iterations before the error settled within the tolerance limits of ± 0.05. This example reinforces the main features, namely the learning and adaptive capabilities, of the DNP-based control scheme.

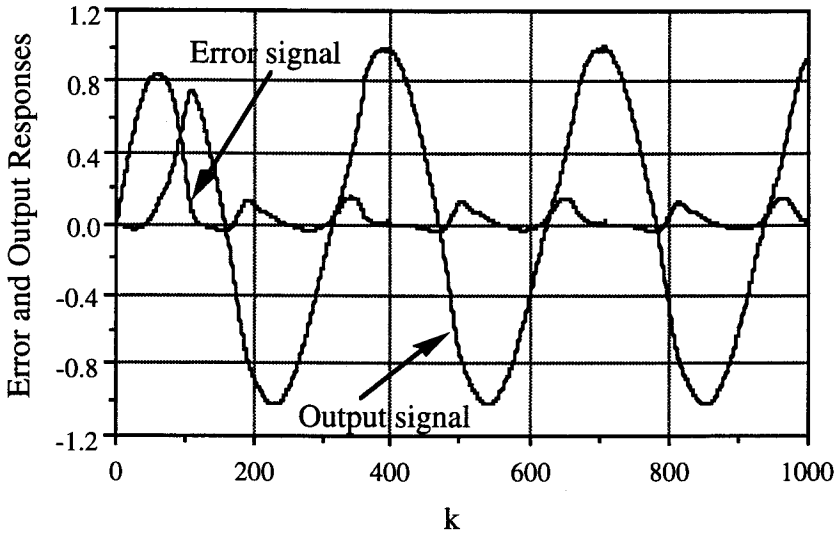


Figure 6.21: The error and output responses of a nonlinear plant represented by Eqn. (6.20), Example 2.

Example 3: The purpose of this simulation was to demonstrate the adaptive capability of the neuro-controller scheme (Fig. 6.19) for the following situations: (i) time-varying nonlinear functions, (ii) varying pattern of input signals, and (iii) perturbations in the plant parameters and changes in the configuration of the plant dynamics. The nonlinear function used in this example was

$$f[.] = e^{(y^2(k-1)+y^2(k-2))} + \sqrt{\left|\{u^2(k) + u^2(k-1) + u^2(k-2)\}\right|}\,, \qquad 0 \le k < 400 \tag{6.21}$$

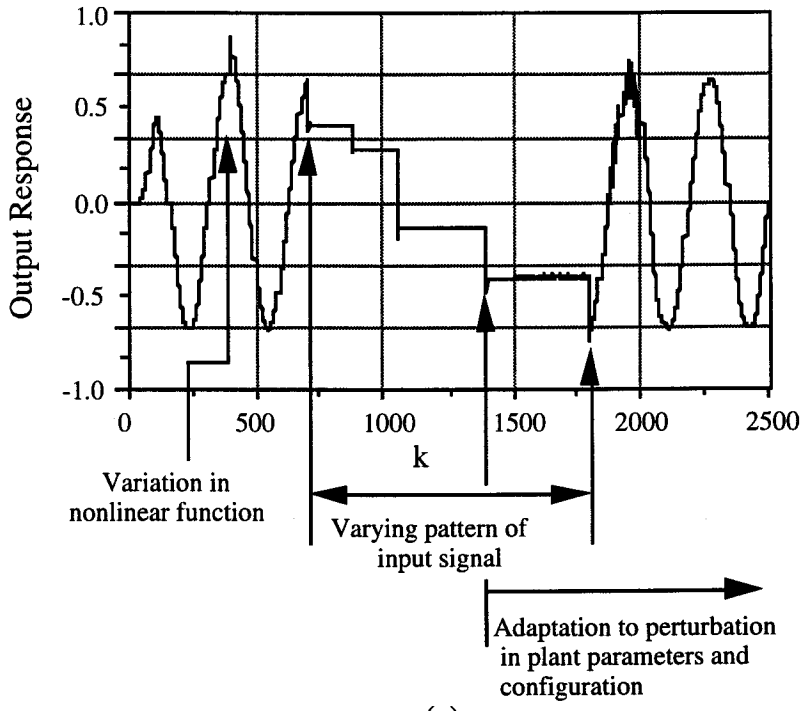
which was changed at k = 400 to

$$f[.] = \frac{\left[2+\cos\left\{7\pi\left(y^2(k-1) + y^2(k-2)\right)\right\}\right] + e^{-u(k)}}{1 + u^2(k-1) + u^2(k-2)}\,, \quad 400 \le k < 2500. \tag{6.22}$$

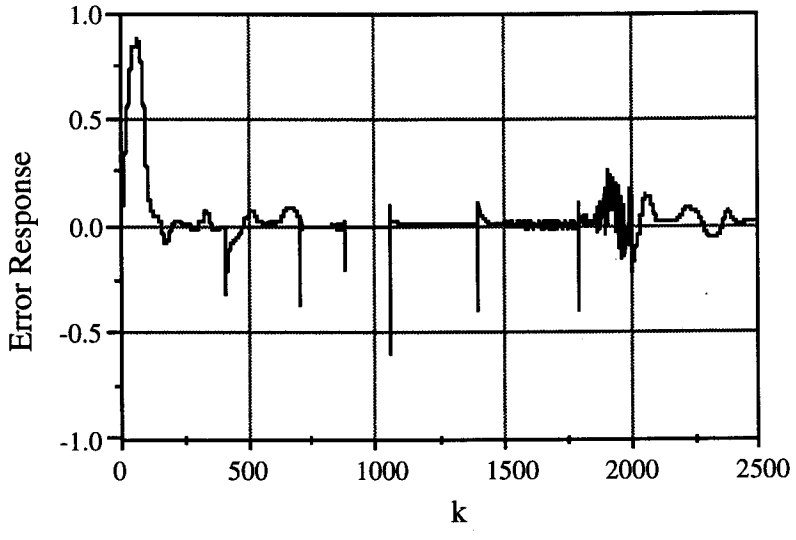
The input to the system, s(k) = sin (2πk / 250) in the interval [-1, 1], was changed as follows: s(k) = sin (2πk / 250), for $0 \le k < 700$, s(k) = 0.6, for $700 \le k < 875$, s(k) = 0.4, for $875 \le k < 1050$, s(k) = - 0.2, for $1050 \le k < 1400$, s(k) = - 0.6, for $1400 \le k < 1800$, and s(k) = 0.6 sin (2πk / 250) for $1800 \le k \le 2500$.

The plant parameters were $\beta_{ff} = [1.2, 1, 0.8]^T$, $\alpha_{fb} = [1, 0.9, 0.7]^T$, for $0 \le k < 1400$, $\beta_{ff} = [1.2, 1, 1.4]^T$, $\alpha_{fb} = [1, 0.9, 1.3]^T$, for $1400 \le k < 2000$, and $\beta_{ff} = [1.2, 1, 0]^T$, $\alpha_{fb} = [1, 0.9, 0]^T$, for $2000 \le k \le 2500$.

The simulation results obtained for this example are shown in Fig. 6.22. This example demonstrates the robustness of the DNP-based control scheme in the presence of variations in the nonlinearities, input signal, and the dynamic characteristics of the plant.



(a)



(b)

Figure 6.22: Simulation results, Example 3.

(a) The plant output under variations in nonlinearity characteristics, input signal variations and plant parameter perturbations.

(b) The corresponding error response.

The above simulation studies show that an unknown nonlinear system followed the desired signals well. The initial settings of the synaptic weights, $w_{\lambda\lambda}$, $w_{\lambda\lambda'}$, and the initial values of the somatic gains of the excitatory and inhibitory neurons in the DNP structure determine the transient behavior of the control system. It was difficult, however, to obtain a general relationship between the initial values of the adaptable parameters and the behavior of the DNP system.

6.3.4 Coordination and Control of Multiple Systems

A complex control system, in general, consists of two or more independently designed and mutually affecting subsystems. Proper coordination and control of the multiple subsystems is necessary for the overall functioning of the system. This necessitates the development of control schemes for multivariable systems. The task of controlling multivariable, also referred to as multi-input-multi-output (MIMO), systems is a complex problem, and has received much attention. This is due to the fact that a multivariable system may involve a nonlinear system with unknown dynamics having two or more inputs and outputs, or may consist of two or more independently designed, separately located, but mutually affecting subsystems. One may observe such complex systems in multi-robot operating situations, or in process industries. In addition to the good behavior of each subsystem, the effective coordination of these subsystems is extremely important to achieve the overall system performance. The main difficulty in coordinating multiple subsystems comes from the lack of precise *a priori* knowledge of the system models and parameters, as well as the lack of efficient tools for system analysis, design, and real-time computation of optimal solutions [108]. Much of the earlier work in the area of control systems has concentrated on linear MIMO systems with unknown parameters [13, 14]. New methods for analysis and design are thus required for the coordination and control of nonlinear multivariable systems.

Neural networks provide alternative and efficient control schemes to deal with uncertainties and nonlinearities in the systems under control. The potential of neural networks for control applications lies in the fact that (i) neural networks can be used to approximate any continuous mapping through learning, and (ii) they can realize parallel processing and fault tolerance. Due to their inherent parallel architecture, neural networks can be effectively employed to control multivariable systems. One of the most popular neural network architectures is a multi layer feedforward network with the back propagation algorithm. For this neural architecture, the weights of the network need to be updated using the network's output error. For a neural network-based controller, the network's output is the

control command to the multiple-systems under control. However, when the neural network is serially connected to the controlled plant, the network's output error is unknown because the desired control action is unknown. This implies that the back propagation algorithm for the neural network can not be applied directly to the control problems [63]. Cui and Shin [63] have developed a direct adaptive control scheme and algorithm using a feedforward neural network for multivariable systems. In this section, the DNP is used in a direct adaptive control scheme for the coordination and control of multivariable systems.

6.3.4.1 The problem of multiple system coordination

Fig. 6.23 describes two interacting systems, and this description can be easily generalized to the case of more than two systems.

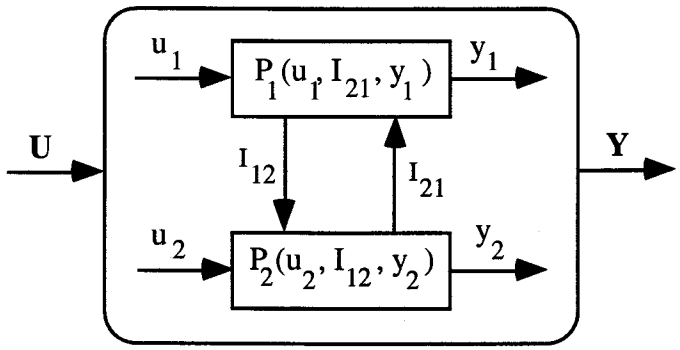


Figure 6.23: Interaction of two subsystems with connection strengths represented as I_{21} and I_{12}.

In the above figure, the two subsystems under control are represented by $P_1(u_1, I_{21}, y_1)$ and $P_2(u_2, I_{12}, y_2)$, where $\mathbf{U} = \left[u_1^T \quad u_2^T \right]^T \in \Re^q$ is the control input vector, $\mathbf{Y} = \left[y_1^T \quad y_2^T \right]^T \in \Re^p$ is the system output vector, and $\mathbf{I} = \left[I_1^T \quad I_2^T \right]^T \in \Re^m$ is the vector of interacting weights between the two subsystems. Usually, the cost function of a multiple system is the sum of the cost functions of all the component subsystems [63], and is given by

$$J(u, I, y) = J_1(u_1, I_{21}, y_1) + J_2(u_2, I_{12}, y_2). \tag{6.23}$$

The problem of coordinating multiple systems can be treated as an optimization problem; that is , of obtaining min J(u, I, y). This may be achieved by treating the interacting signal $I_{k,l}$, $k \neq l$, as an ordinary input variable to each of the interacting subsystem as depicted in Fig. 6.24.

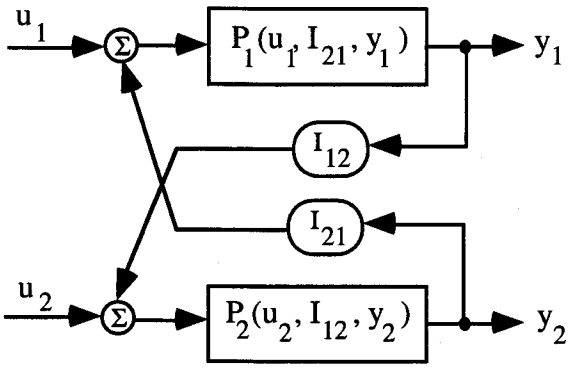


Figure 6.24: The configuration of two interacting systems.

Obviously, minimizing the cost function J(.), defined in Eqn. (6.23), depends on knowledge of the structure and/or dynamic parameters of the subsystems. As the neural-network based control schemes exhibit learning and adaptive capabilities for nonlinear systems, it is not necessary to know *a priori* the system configuration, the parameters of the subsystems or the nature of interaction between them. In this section, the DNP is used to make the linear and nonlinear subsystems follow the desired command trajectories.

6.3.4.2 Computer Simulation Studies

The DNP is used in the direct adaptive control mode [110] as shown in Fig. 6.25. The desired command signals $s_1(k)$ and $s_2(k)$ were respectively sin $(2\pi k / 250)$ in the interval [-1, 1] and cos $(2\pi k / 250)$ in the interval [-0.5, 0.5]. The DNP settings were $w_{EE} = 1.0$, $w_{EI} = 0.5$, $w_{II} = 1.0$ and $w_{IE} = 0.5$. The components of the scaling vector were set to 1.

Example 1: The purpose of this example was to demonstrate that a complex system consisting of two linear sub-systems can be adaptively controlled in the presence of input signal variations, parameter perturbations, and with nonlinear coupling. The two interacting linear systems are governed by the following difference equations

$$y_1(k+1) = \sum_{i=0}^{2} \alpha_{i1}\, y(k-i) + \sum_{j=0}^{2} \beta_{j1}\, u_1(k-j) \quad : \text{ System 1} \tag{6.24a}$$

and

$$y_2(k+1) = \sum_{i=0}^{2} \alpha_{i2}\, y(k-i) + \sum_{j=0}^{2} \beta_{j2}\, u_2(k-j) \quad : \text{ System 2.} \tag{6.24b}$$

The parameter values in the above equations were as follows:

$$\beta_{ff1} = [1.2, 1, 0.8]^T, \alpha_{fb1} = [1.3\ 0.9, 0.7]^T: \text{ System 1}$$

and

$$\beta_{ff2} = [1.3, 0.7, 0]^T, \ \alpha_{fb2} = [1.2\ 0, 0.8]^T: \text{ System 2.}$$

The components of the interaction vector $\mathbf{I}_{k,l}$ were set to -0.1. The simulation results obtained for this case are shown Fig. 6.26.

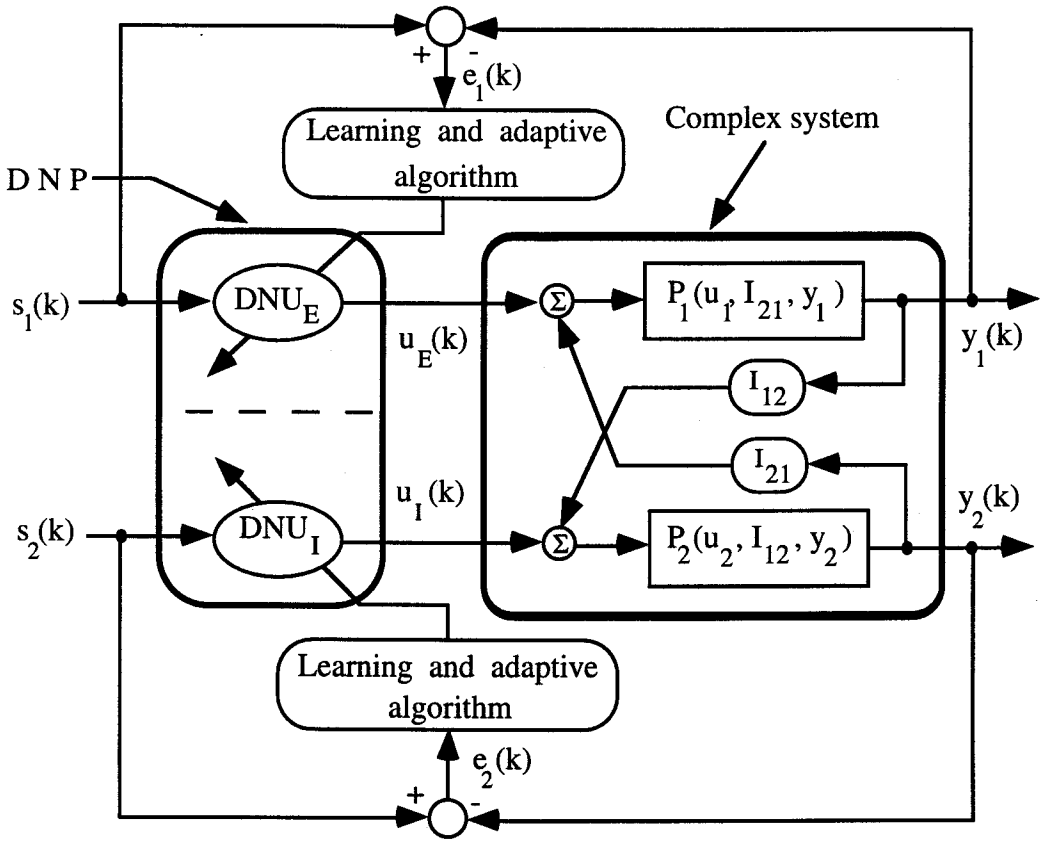


Figure 6.25: The direct adaptive control scheme for the coordination and control of two subsystems.

It is seen from these results that the two systems followed the command signals very closely. The system behavior was good even when the input signal of system 1 was changed, and the parameter perturbations were introduced for system 2. At k = 500, $s_1(k)$ was changed to a sum of two sinusoids, sin $(2\pi k / 250)$ + 0.2 sin $(2\pi k / 25)$. The dynamics of system 2 at k = 500 were also changed to $\beta_{ff2} = [1.3, 0.7, 1.0]^T$, $\alpha_{fb2} = [1.2\ 0.9, 0.8]^T$. The error and output responses of the two systems are shown in Figs. 6.27a and 6.27b respectively.

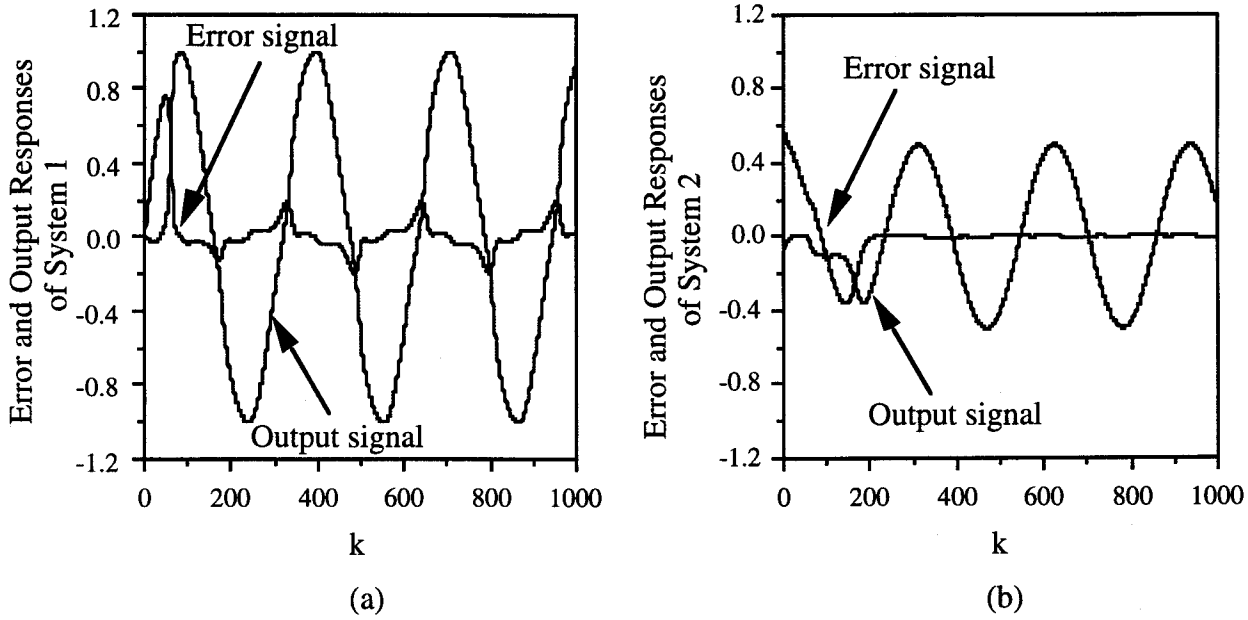


Figure 6.26: Simulation results for two interacting linear systems 1 and 2, Example 1.
(a) The error and output responses of system 1, and
(b) The error and output responses of system 2.

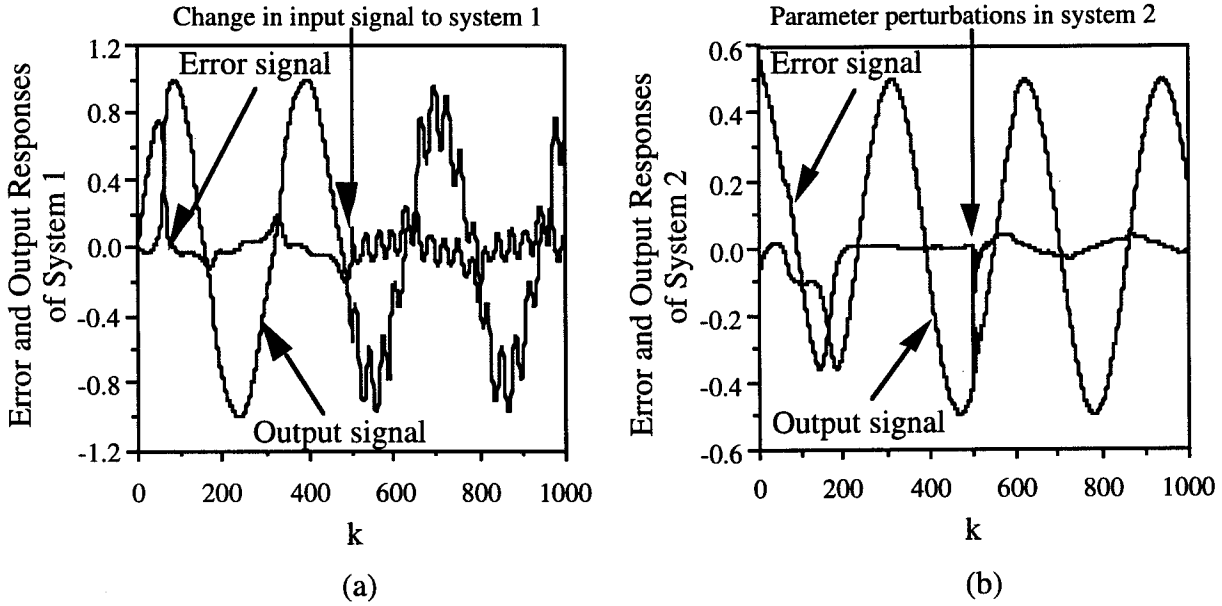


Figure 6.27: Simulation results with input signal and parameter variations, Example 1.
(a) The error and output responses of system 1, and
(b) The error and output responses of system 2.

From the results shown in Fig. 6.27 it can be observed that the effects of changing the input signal of system 1 on system 2, and introducing the parameter perturbations in system 2 on the performance of system 1 were not very significant. However, this depends on the strength of the interaction between the systems. This is shown in the following figures where the components of the interaction strength vector **I** were changed at k = 500 as follows:

$I_{12} = 0.3\ y_1^3(k)$, from system 1 to system 2, and

$I_{21} = 0.1\ e^{-(y_2(k))}$, from system 2 to system 1.

The simulation results obtained for this case are shown in Fig. 6.28. Figures 6.28a and 6.28b show the error and output responses of systems 1 and 2 respectively. Figure 6.28c shows the adaptation in the somatic gains of the excitatory and inhibitory neurons. The optimum somatic gains after 2500 iterations were found be 0.858 and 0.795 for the excitatory and inhibitory neurons respectively.

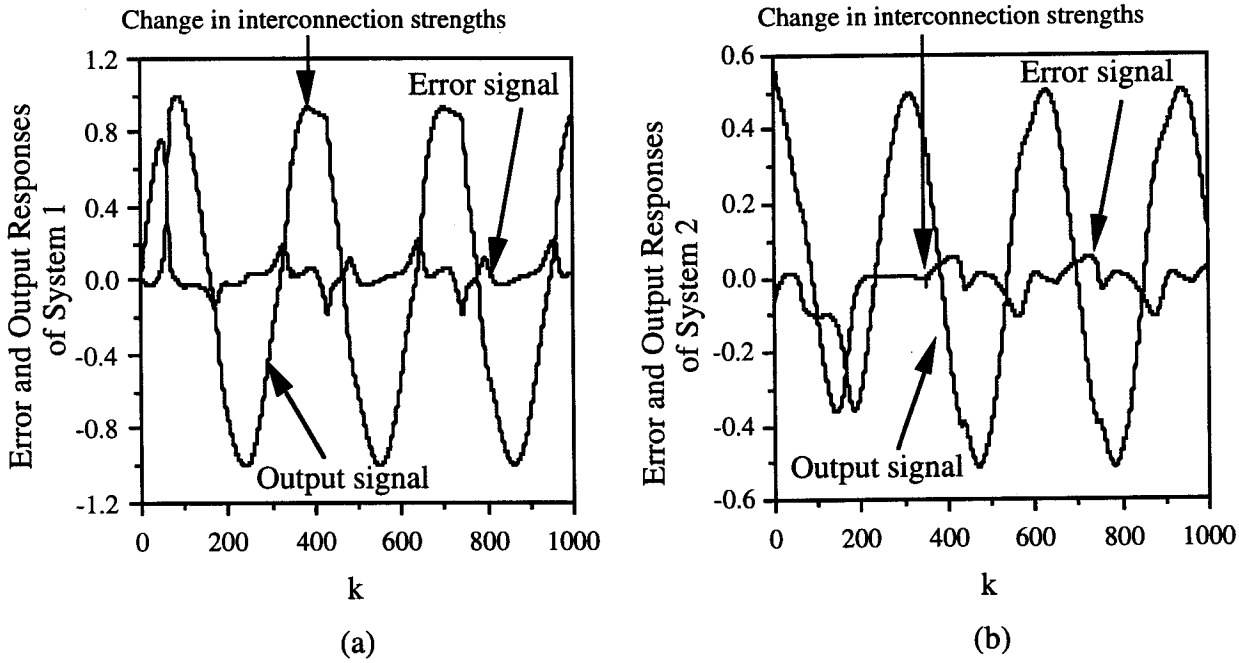
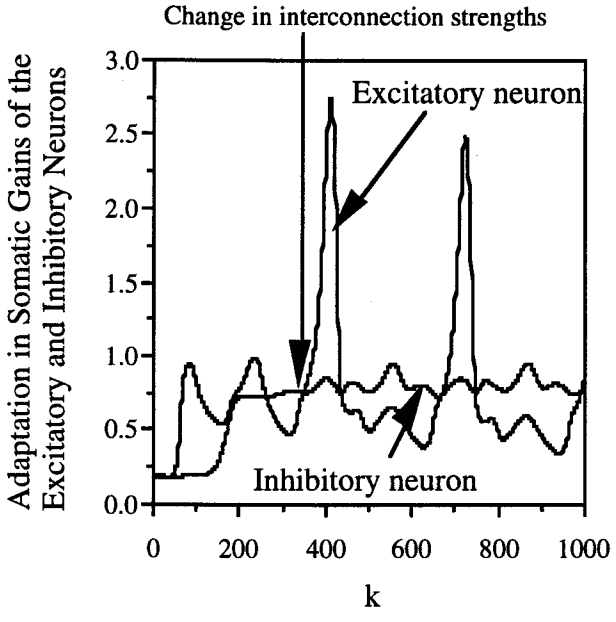


Figure 6.28: (Continued)



(c)

Figure 6.28: The simulation results for a case when systems 1 and 2 were interconnected through nonlinear coupling.

Example 2: The purpose of this simulation example was to demonstrate that the direct adaptive control scheme shown in Fig. 6.25 could be used without any modifications in the algorithm and the initial values of the weights even for the nonlinear multivariable systems.

Case (i): In this case, the two interacting systems were nonlinear and were governed by the following difference equations:

$$y_1(k+1) = \sum_{i=0}^{2} \alpha_{i1}\, y(k-i) + \sum_{j=0}^{2} \beta_{j1}\, u_1(k-j) + f_1\,[y_1(k), y_1(k-1),.., y_1(k-n+1); u_E(k), u_E(k-1),.., u_E(k-m+1)] : \text{System 1} \qquad (6.25a)$$

and,

$$y_2(k+1) = \sum_{i=0}^{2} \alpha_{i1}\, y(k-i) + \sum_{j=0}^{2} \beta_{j1}\, u_1(k-j) + f_2[y_2(k), y_2(k-1),.., y_2(k-n+1); u_I(k), u_I(k-1),.., u_I(k-m+1)] : \text{System 2.} \qquad (6.25b)$$

The nonlinear functions in the above equations were as follows

$$f_1[.] = \frac{\left[2+\cos\left\{7\pi\left(y_1^2(k-1) + y_1^2(k-2)\right)\right\}\right] + \sqrt{\left|\left\{u_E^2(k) + u_E^2(k-1) + u_E^2(k-2)\right\}\right|}}{\left[1 + u_E^2(k-1) + u_E^2(k-2)\right]}, \tag{6.26a}$$

and

$$f_2[.] = e^{\frac{y_2^2(k-1) + y_2^2(k-2)}{2}}. \tag{6.26b}$$

The plant parameter values and the input signals used in this example were the same as in Example 1. The error and output responses of systems 1 and 2 are shown in Figs. 6.29a and 6.29b respectively. The adaptation in the somatic gains of the excitatory and inhibitory neurons of the DNP are shown in Fig. 6.29c. The optimum somatic gains after 2500 iterations were found to be 0.25 and 2.08 for the excitatory and inhibitory neurons respectively.

Case (ii): In this case, two nonlinear subsystems described by the following equations

$$y_1(k) = 0.2\, y_1(k-1) + 0.6\, y_1(k-2) - 0.1\, y_2(k) - 0.1\, y_2(k-1) + 0.6\, u_E(k-1) + u_E(k)$$

: System 1 (6.27a)

$$y_2(k) = -0.1\, y_1(k) + \frac{0.3\, y_2(k-1) + 0.5\, y_2(k-2)}{1 + y_2^2(k-2)} + u_I(k) \quad : \text{System 2} \tag{6.27b}$$

were considered. The performance of each system was observed for a time duration of 1000 learning iterations. The behavior of the interconnected nonlinear systems is shown in Fig. 6.30.

The DNP-based control scheme can quickly adapt to the changing system configuration. To demonstrate this adaptive capability, the configuration of system 1 was changed at time step k = 500 to

$$y_1(k) = 0.2\, y_1(k-1) + 0.6\, y_1(k-2) - 0.1\, y_2(k) - 0.1\, y_2(k-1) + 0.1\, y_1(k-1)\cos(2\pi k / 250) + u_E(k). \tag{6.28}$$

The simulation results are shown in Fig. 6.31, and from these results it is seen that systems 1 and 2 followed the desired trajectories very closely demonstrating the adaptive feature of the control scheme.

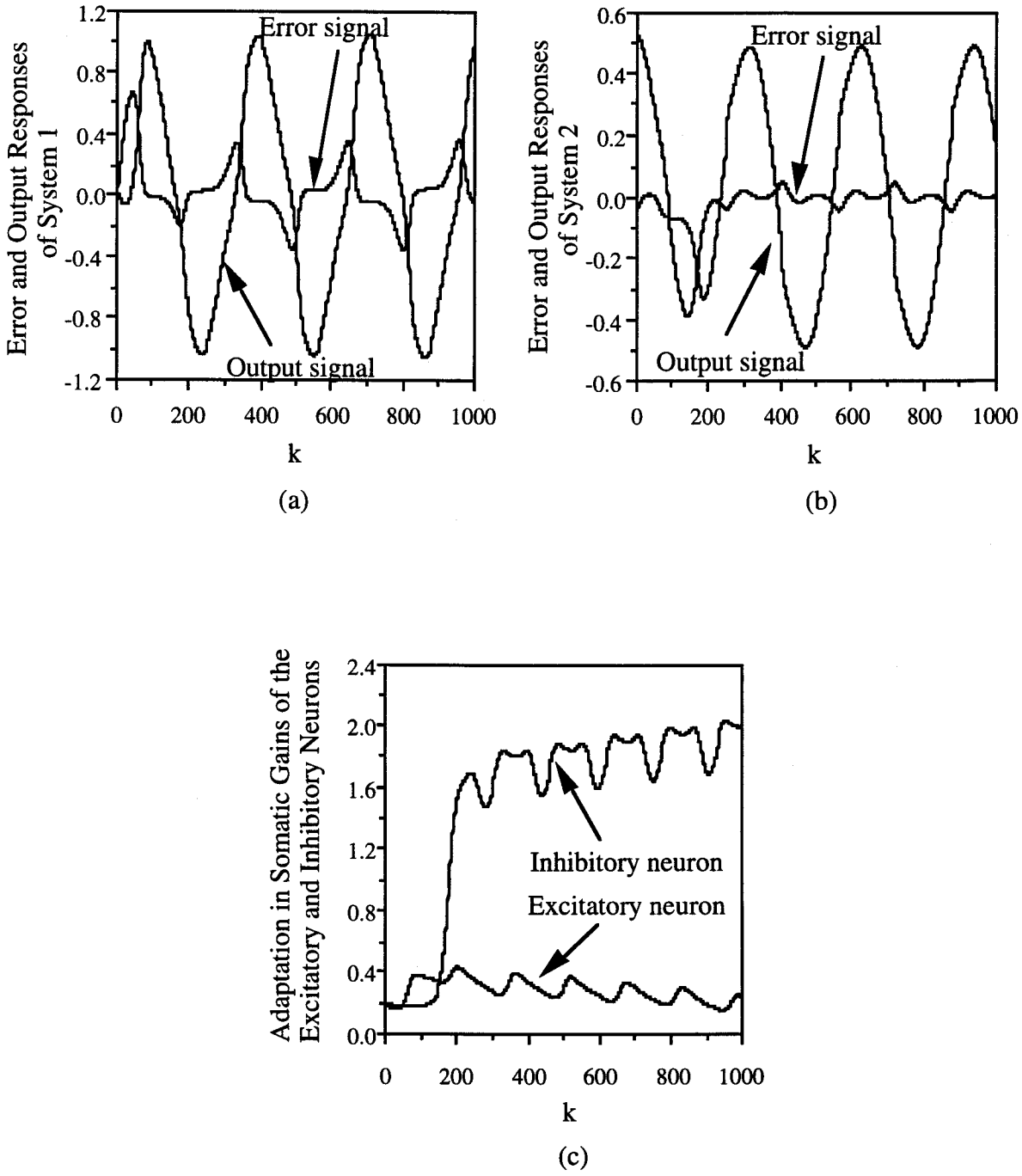


Figure 6.29: The simulation results for a case when the two interconnected systems were nonlinear, Case (i), Example 2.

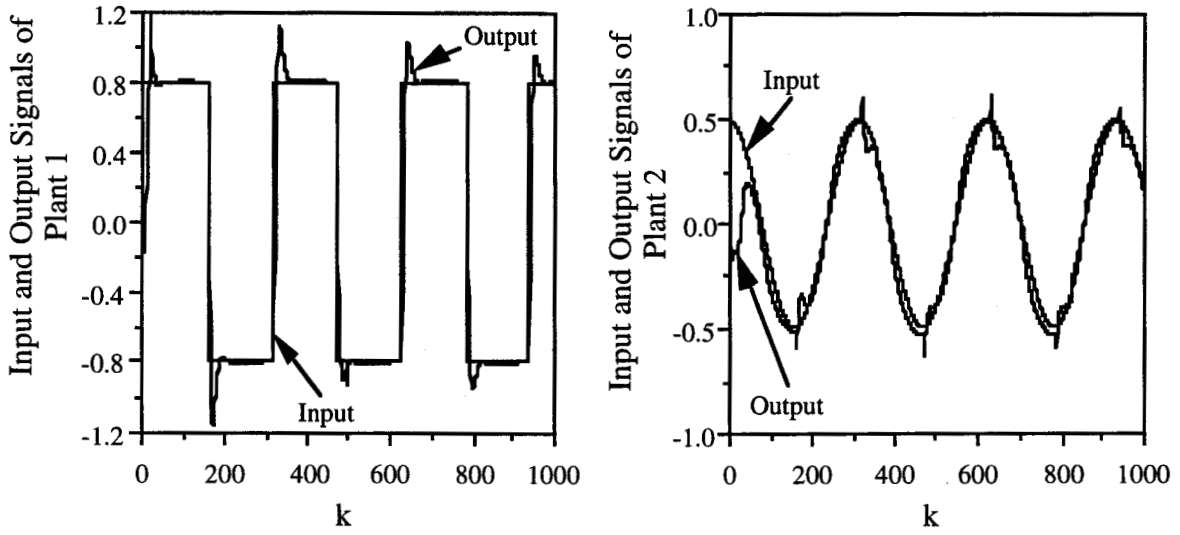


Figure 6.30: The simulation results for a case when the two interconnected systems were nonlinear, Case (ii), Example 2.

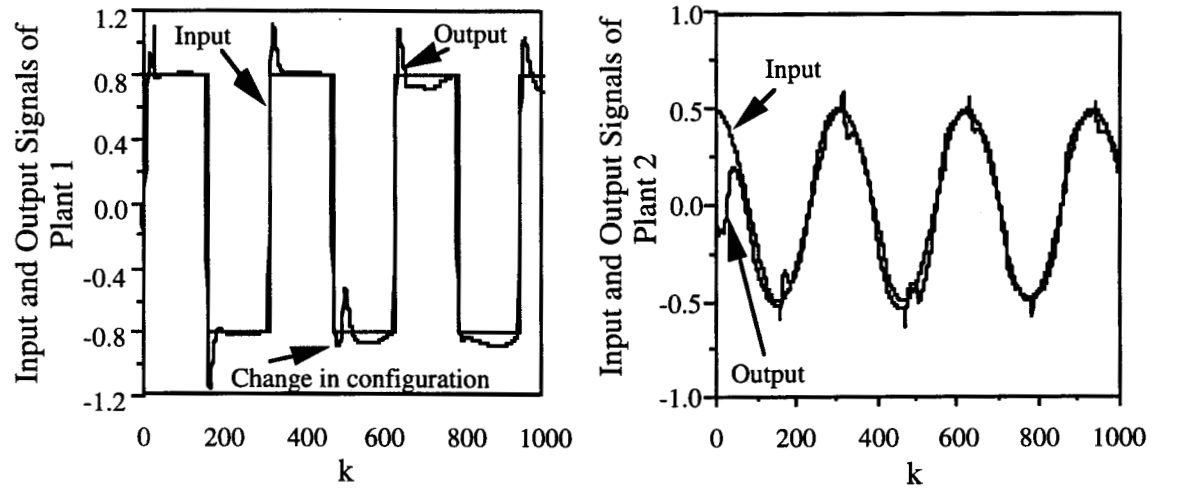


Figure 6.31: The simulation results for two nonlinear interconnected systems with dynamic perturbations, Case (ii), Example 2.

Example 3: In the above examples, multivariable systems consisting of two subsystems were considered. In this example, the problem of truck "backer-upper" control, which is a nonlinear MIMO system, proposed originally by Nguyen and Widrow [111] was considered. Backing a truck to the loading dock is a difficult nonlinear control problem for which no traditional control system design methods exist [112]. Nguyen and Widrow used two static neural networks one as an emulator and the second as a controller to guide the truck to the loading truck. The controller network produced the appropriate steering angle of the truck

given any initial position. The emulator network computed the next position of the truck. The inputs to the emulator network were the previous truck position and the current steering angle output computed by the controller network. As reported in [111], the number of back ups required to train the controller was about 20,000. Kong and Kosko [113] proposed a fuzzy control strategy for the same problem. Wang and Mendel [112] developed a 'numerical-fuzzy approach' for this problem. In this approach, they determined the control angle θ based on the 'common sense', and after some trials they chose the desired input-output pairs corresponding to the smoothest successful trajectory.

The simulated truck and loading zone are shown in Fig. 6.32. The truck position is exactly determined by the variables x, y, and ϕ, where ϕ is the angle of the truck with the ground. The control signal to the truck is the steering angle θ. Only backing up was considered in the simulation study. The truck moved backwards by a fixed unit distance at every stage. For simplicity, enough clearance between the truck and the loading dock was assumed such that 'y' did not have to be considered as an input. The task was to generate proper steering angles of the truck for the input variables $x \in [0, 20]$ and $\phi \in [-90^0, 270^0]$ such that the final truck position was $(x_f, \phi_f) = (10, 90^0)$. The following dynamic equations of the truck backer-upper control system [112] were used during the simulation studies.

$$x(k+1) = x(k) + \cos[\phi(k) + \theta(k)] + \sin[\theta(k)] \sin[\phi(k)] \tag{6.29a}$$

$$y(k+1) = y(k) + \sin[\phi(k) + \theta(k)] - \sin[\theta(k)] \cos[\phi(k)] \tag{6.29b}$$

$$\phi(k+1) = \phi(k) - \sin^{-1}\left[\frac{2\sin\theta(k)}{L}\right] \tag{6.29c}$$

where L is the length of the truck which was assumed to be 4 in the simulation studies. Equations (6.29a - c) were used to obtain the next state when the present state and control are given. Since y was not considered a state, only Eqns. (6.29a) and (6.29c) were used in the simulations. Figure 6.33 shows the simulation results for backing up the truck to the loading dock from a given initial position $(x_i, \phi_i) = (5, 220)$. The x, ϕ and θ trajectories of the truck for this starting position are shown in Figs. 6.33a and 6.33b. In order to compare the performance of the DNP with recurrent neural networks, a two-layer recurrent neural network was used to steer the truck to the loading zone from the initial position $(x_i, \phi_i) = (5, 220)$. The docking-error, defined as the Euclidean distance from the actual final position (x, ϕ) to the desired final position (x_f, ϕ_f) [112], obtained from the DNP and the recurrent neural network was compared for 300 iterations as shown in Figs. 6.33c and 6.33d. The latter successfully steered the truck to the desired position after about 7000 iterations.

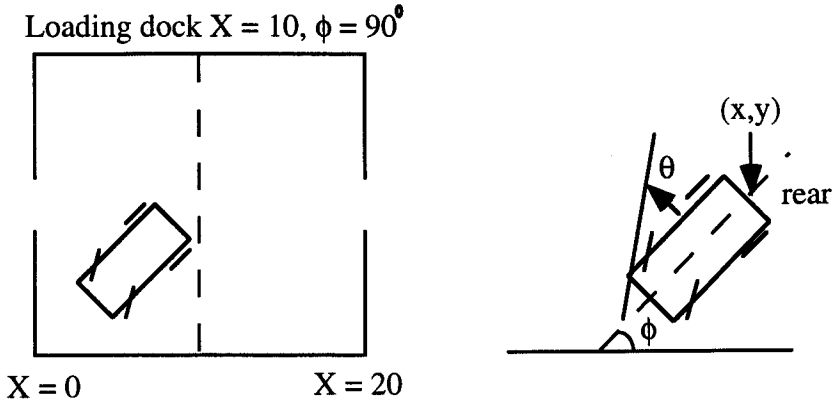


Figure 6.32: Diagram of the simulated truck and loading zone.

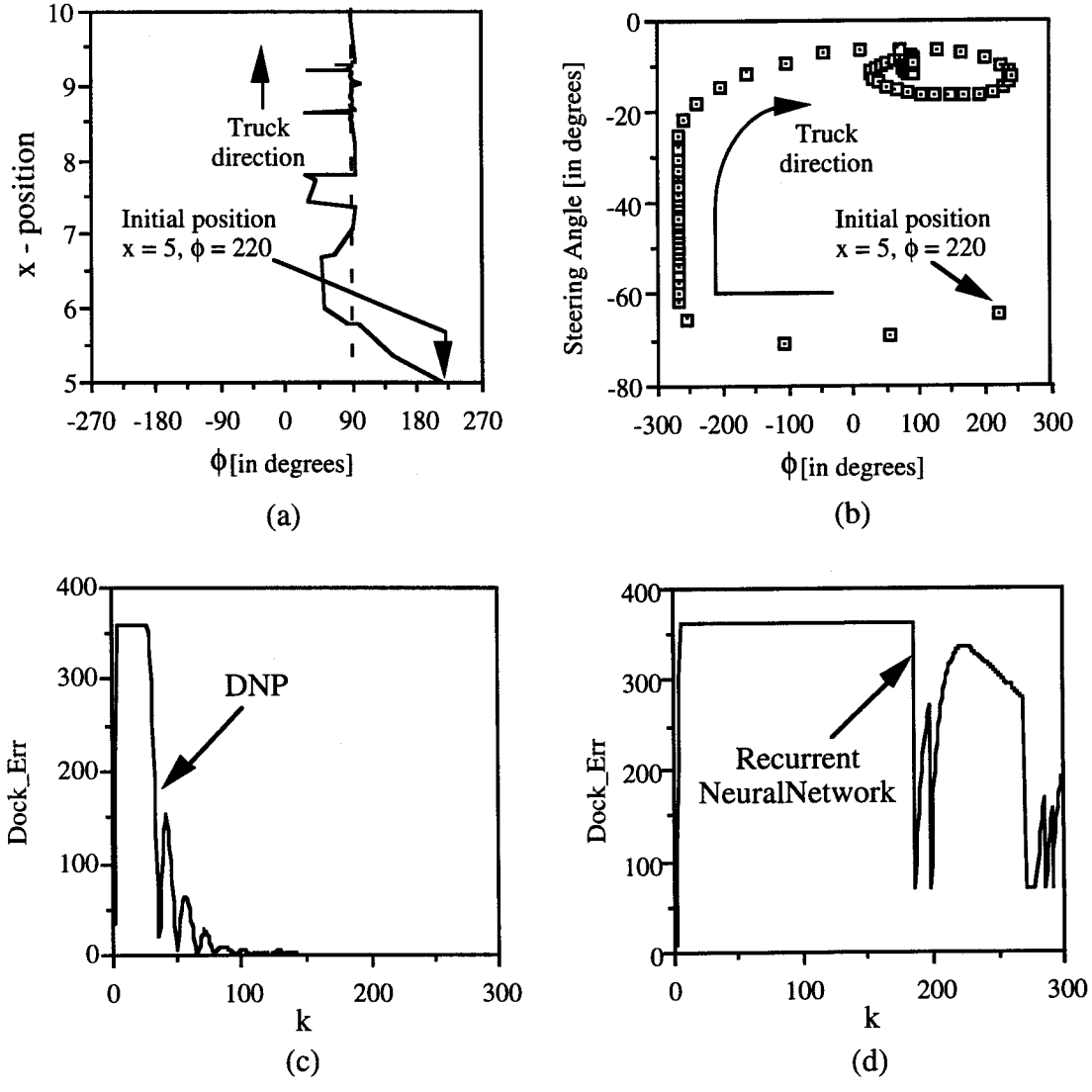


Figure 6.33: Truck trajectories from an initial position $(x_i, \phi_i) = (5, 220)$.

Figure 6.34 shows the x and ϕ trajectories of the truck obtained from the DNP and the recurrent neural network from an initial position $(x_i, \phi_i) = (0, -90)$. About 8000 iterations were required for the recurrent neural network before the truck reached the target position. Figure 6.35a shows the x and ϕ trajectories from an initial position $(x_i, \phi_i) = (3, -30)$ and Fig. 6.35b compares the docking-error obtained from the DNP and the recurrent neural network. The latter required about 7500 iterations to steer the truck to the loading dock. Extensive simulations were carried out for the DNP and for the recurrent neural network from different initial positions of the truck. Some of the results are shown in Table 6.4.

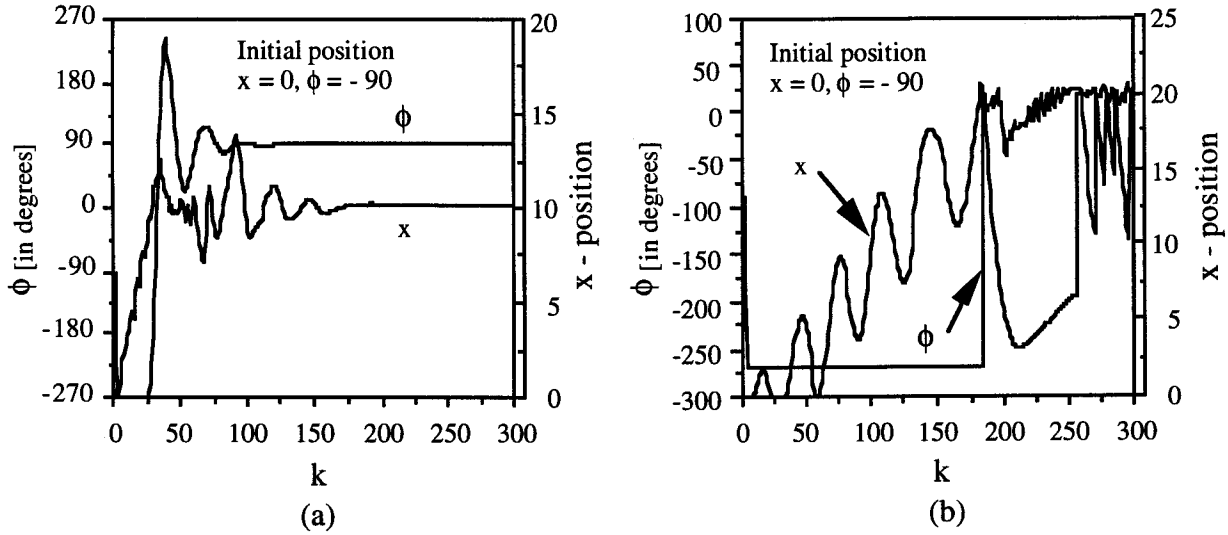


Figure 6.34: Truck trajectories from an initial position $(x_i, \phi_i) = (0, -90)$: (a) using the DNP and (b) using the recurrent neural network.

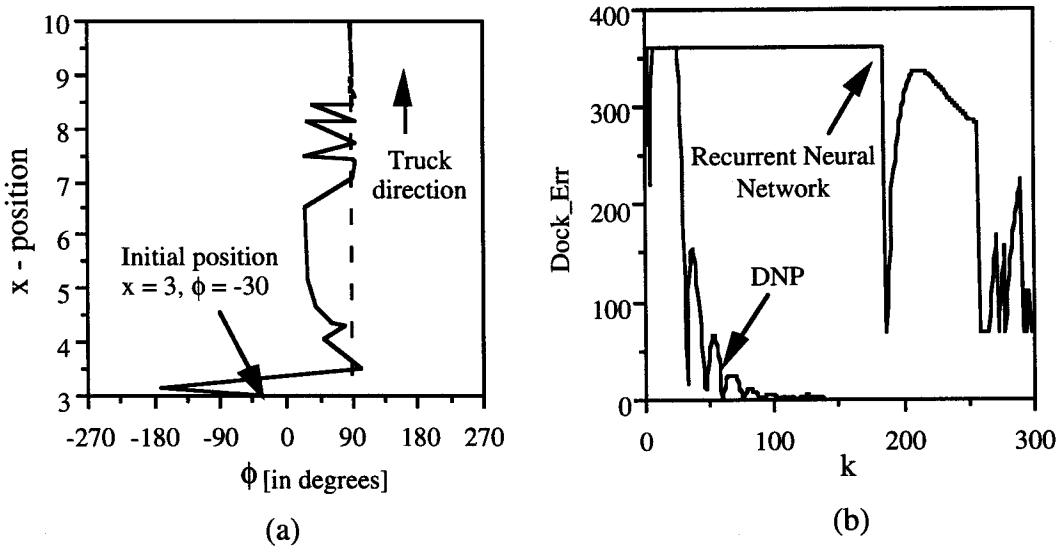


Figure 6.35: Truck trajectories from an initial position $(x_i, \phi_i) = (3, -30)$.

Table 6.4: Performance comparison

Desired Position: $x_d = 10$ and $\Phi_d = 90^0$

Initial Positions		DNP Estimations				Recurrent Neural Network Estimations		
x_i	Φ_i^0	x	Φ^0	θ^0	Learning Iterations	x	Φ^0	Learning Iterations
1.00	0	10.08	90.42	- 18.56	436	*	*	----
3.00	- 30	9.97	88.54	- 11.3	308	9.968	89.03	7756
5.00	- 90	9.97	88.95	- 11.01	383	9.93	88.98	9575
20.00	90	10.06	89.3	- 10.87	421	*	*	----
10.00	120	9.91	88.66	- 11.41	386	9.947	89.16	9647
0.00	270	9.9	89.06	- 11.86	248	10.02	90.24	6233
10.00	220	9.95	89.01	- 9.0	274	9.856	88.92	6528
13.00	30	10.01	91.45	- 11.28	333	9.93	90.85	8210
20.00	270	9.91	91.63	- 11.12	382	*	*	----
20.00	- 90	9.92	91.19	- 10.49	381	*	*	----
0.00	- 90	9.99	90.71	- 10.55	315	9.99	91.03	7649
10.00	- 90	10.01	89.51	- 10.82	261	10.00	90.02	5705
5.00	90	10.02	90. 18	- 8.45	156	9.97	89.93	4964
9.88	89.44	10.01	90.01	- 2.14	11	10.02	90.01	2664

* indicates the initial positions from which the recurrent neural network could not converge.

From the simulation results shown in Figs. 6.33, 6.34 and 6.35, and Table 6.4 it is clear that the DNP could steer the truck to the target position very quickly compared to the recurrent neural network. In some cases, the recurrent neural network could not converge. On the other hand, the DNP could coerce the truck from different initial positions to the target position. As the DNP was used in the direct control mode, off-line training was not necessary in sharp contrast with the methodology involving conventional neural structures. It was found necessary that the conventional neural networks, with error back-propagation learning algorithm, be trained off-line for different initial positions and use the trained network to drive the truck to the target position.

6.4 Generalized Dynamic Neural Model

As was mentioned in Chapter 1, there are numerous possible structures of computational (artificial) neural networks. Currently, there is no single architecture of a computational neuron from which all of the existing neural structures can be derived. It is desirable in the field of neural networks to develop a general computational neural morphology that can represent the characteristics and emulate the functional capabilities observed in biological neural networks. In this section, a generalized dynamic neural model based on the concept of neural subpopulations described earlier in this chapter has been described. This generalized model was proposed in [114]. It is demonstrated in this section that the existing neural models, such as the feedforward (static) neural network, feedback (recurrent) neural network, time delay neural network (TDNN) and dynamic neural unit (DNU) are a subclass of this generic model. The generalized model is shown in Fig. 6.36. The model is of second-order and its output forms an argument to a time-varying nonlinear activation function. This part of the generalized model is simply the DNU structure discussed earlier. The generalized model is an extension of the DNU, and incorporates the delayed feedback that represents the soft (adaptable) connectivity between the subpopulations of neurons.

As shown in Fig. 6.36, the feedforward synaptic matricies are denoted as G, H, P, while the feedback synaptic matricies A, B form the neural dynamics with an internal threshold θ. The matrix C denotes the self- and inter-neuron feedback strength. The matricies F and D represent the scaling matricies of the input and output signals respectively. In conventional static neural networks, the matrix F would represent the synaptic weights. The output of the neural dynamics forms an argument to a nonlinear activation function, usually sigmoidal, with varying slope. This adaptation in the slope of the sigmoidal function, called the somatic adaptation, provides a self-tuning feature to the neural model.

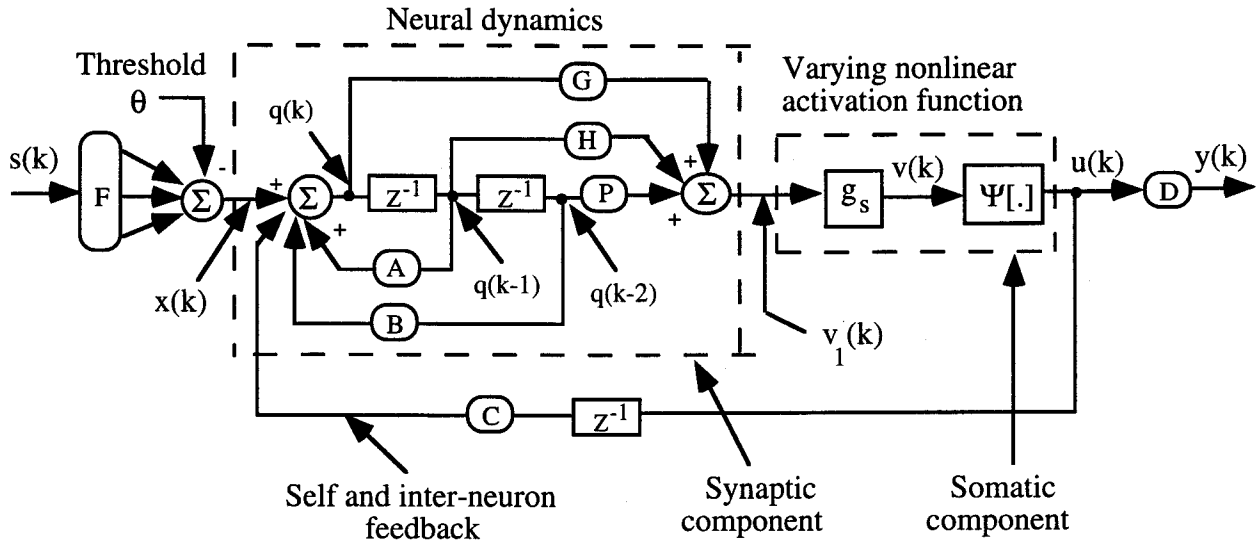


Figure 6.36: A generalized neural model based on excitatory - inhibitory antagonistic neural subpopulations.

The functional dynamics of this neural architecture are represented by the following difference equations:

$$x(k) = F\, s(k) - \theta \tag{6.30a}$$

$$q(k) = x(k) + A\, q(k-1) + B\, q(k-2) + C\, u(k-1) \tag{6.30b}$$

$$v_1(k) = G\,[q(k)] + H\,[q(k-1)] + P\,[q(k-2)] \tag{6.30c}$$

$$v(k) = g_s\, v_1(k) \tag{6.30d}$$

$$u(k) = \Psi[v(k)] \tag{6.30e}$$

$$y(k) = D\, u(k). \tag{6.30f}$$

It is demonstrated in the following paragraphs that existing neural structures can be derived from this generalized neural model.

(i) **Feedforward (Static) Neural Network**

As was described in Section 1.2, the static structure of an artificial neuron receives its inputs from a number of other neurons or from sensors. A weighted sum of these inputs constitutes the argument of an 'activation' function. The resulting value of the activation function, if it exceeds an internal threshold θ, is the neural output. This output is distributed along weighted connections to other processing units. The static neural network is a subset

of the generalized structure with A = B = C = H = P = C = 0, G = 1, and g_s = 1 (nonlinear function with constant slope), as shown in Fig. 6.37.

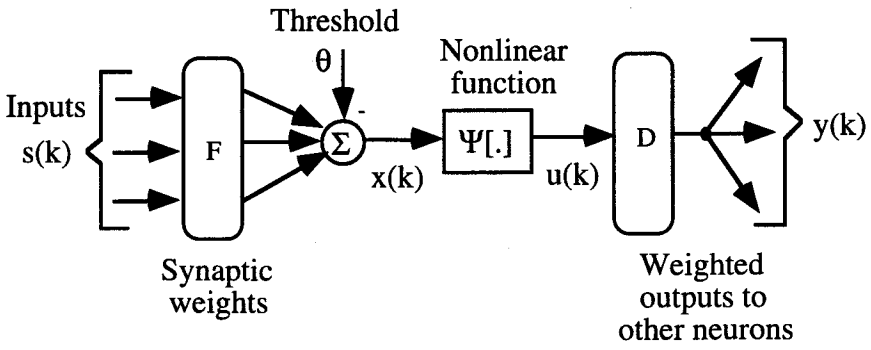


Figure 6.37: The structure of a static neuron as a special case of the dynamic structure shown in Fig. 6.36.

(ii) **Feedback (Recurrent) Neural Network**

The conventional dynamic neural structure, shown in Fig. 6.38, can be obtained as a special case of the generalized structure shown in Fig. 6.36, with A = B = H = P = 0, G = 1, and g_s = 1.

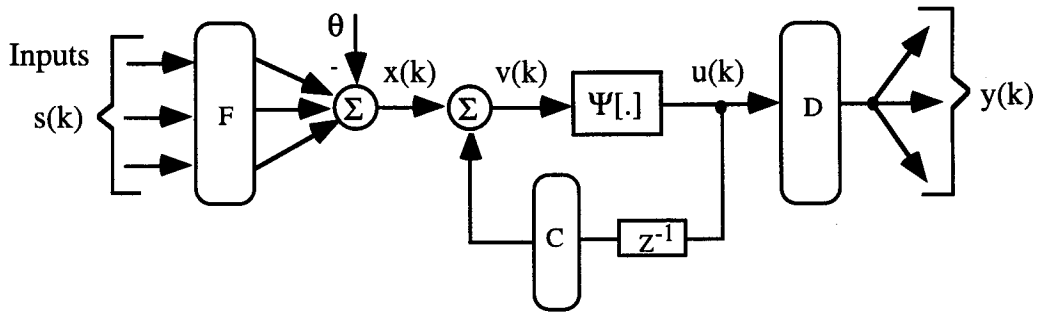


Figure 6.38: A feedback (recurrent) neural network derived from the generalized model.

(iii) **Time-Delay Neural Network (TDNN)**

The dynamics of a time-delay neural network can be described as a special case of the generalized dynamic structure with A = B = C = 0, and g_s = 1. These equations lead to a TDNN structure as shown in Fig. 6.39.

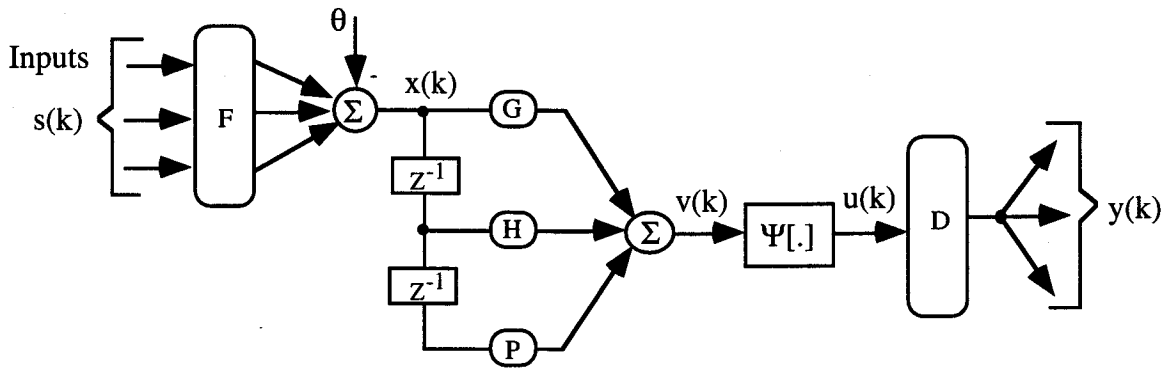


Figure 6.39: A time-delay neural network (TDNN) as a special case of Fig. 6.36.

(iv) **Dynamic Neural Unit (DNU)**

The structure of the DNU is identical to that shown in Fig. 6.36 except that there is no feedback path from the neural output u(k), to the input, that is, C = 0. The DNU structure thus obtained is shown in Fig. 6.40.

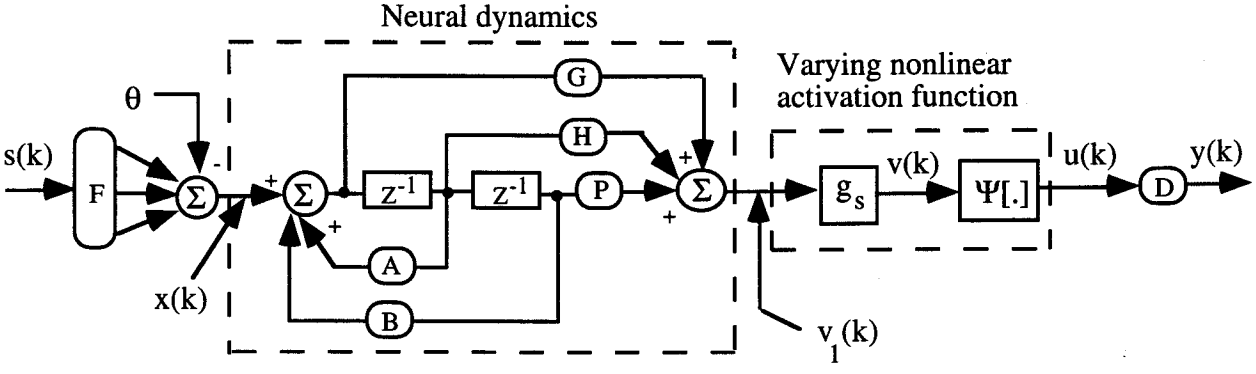


Figure 6.40: The structure of DNU as a special case of the generalized model.

A learning algorithm for the generalized neural model is derived in Appendix II. Although it is shown in this section that several computational neural networks can be obtained from the proposed generalized model, more work needs to be done with regard to the stability and convergence analysis of the model.

6.5 Summary

In this chapter a dynamic neural processor (DNP) that implements the dynamic properties of a subpopulation of neurons has been developed. The basic motivation for this neuronal model has been the observation in neurophysiology that the neural activity of any complexity depends upon the interaction between antagonistic (excitatory and inhibitory) neural subpopulations. Dynamic neural units (DNUs), coupled as excitatory and inhibitory neurons, have been used as the basic computing elements in the DNP architecture. A mathematical model and an algorithm to modify the parameters of the DNP have been discussed. A commonly used dynamic neural structure is the recurrent neural network consisting of a single layer feedforward network included in a feedback configuration with a time delay. Except for the delay operator, this neural network does not employ any dynamic elements in the forward path. The feedback paths are also non-adaptable. On the other hand, the DNP consists of a dynamic structure in the forward path and adaptable feedback connections. Thus, the structure of the DNP is different from the conventionally assumed structures of neural networks in that the former uses two second-order nonlinear dynamic systems, while the latter are developed based on the concept of an idealized single static neuron.

Four applications of the DNP have been discussed in this chapter. The first application involved the functional approximation of arbitrary nonlinear functions. In the second application, the DNP was employed to compute the inverse kinematic transformations of a two-link robot that modeled a human leg. A brief comparison of recurrent neural networks and the DNP, as applied to functional approximation and inverse kinematic computations of a two-link robot, was also made. The DNP approximated the arbitrary nonlinear functions much more quickly than the single- and two-layer recurrent neural networks. The simulation studies demonstrated that the single-layer recurrent neural network performed better, in terms of speed of convergence, than the two-layer network. In the presence of noise, however, the performance of the recurrent neural networks was better than that of the DNP. It was demonstrated in the third application that the DNP could also be used for the adaptive control of unknown nonlinear dynamic systems. As was demonstrated in the last application, the DNP could be easily employed for the coordination and control of multiple systems due to its parallel architecture. A generalized dynamic neural model based on the concept of neural subpopulations has been developed in this chapter. It was shown that many existing neural structures can be obtained from this generalized model.

7. Conclusions

7.1 Concluding Remarks

The computational architectures employed in artificial neural networks are generally based upon mathematical models used to describe the behavior of individual biological neurons or population of neurons. Although all neural network models claim to share a theoretical foundation with biology, they often vary greatly in both complexity and scope. These differences are largely influenced by the goals and academic background of the individual developer. For example, very elaborate models of neural population dynamics have been proposed by researchers in the area of theoretical biology. Alternatively, many of the neural network structures proposed for pattern recognition, system identification and control purposes are simple nonlinear summation circuits. In conventional neural structures, the neuron receives its inputs either from other neurons or from sensors. A weighted sum of these inputs constitutes the argument of a 'fixed' nonlinear activation function producing the neural output. This output is distributed with weighted connections to other processing units.

The above model is a highly simplified but useful first approximation of the biological neuron. Neural networks developed based on this static model respond instantaneously to inputs because these neural networks have no dynamic elements. As has been pointed by Hopfield [35], present models of neural networks are a feeble imitation of biological neural structures. These models ignore many of the salient features of biological neurons, such as time delays and feedback paths, that are very important in the activity of neural models. An attempt has been made in this thesis to develop neural structures based on the dynamic neural model. However, it is not claimed that the neural model proposed in this thesis satisfies all the characteristics of a biological neuron. It is a small but significant step toward the development of dynamic neural networks.

A dynamic model of the neuron, called the *dynamic neural unit* (DNU), was proposed in the second chapter. The topology of DNU was based on the reverberating circuit in the neuronal pool of the central nervous system. It is only analogous to the reverberating circuit and does not represent any specific anatomical region within the biological nervous system. The dynamic structure of the DNU consisted of internal feedforward and feedback synaptic weights, followed by a nonlinear activation operator. A learning and adaptive algorithm was developed to modify the DNU parameters for a given task.

A control technique, called *inverse dynamic adaptive control* (IDAC) using the DNU was developed in the third chapter. The IDAC technique was based on the concept of adaptive inverse control in which an unknown plant can be made to follow a desired trajectory by precascading the plant with its inverse model. It was demonstrated in this chapter that the DNU can be used to obtain an approximate inverse model of the plant under control, thereby achieving almost unity mapping between the input and output signal spaces. Through computer simulation studies it was shown that the DNU, after the initial learning, could cause a linear plant to follow a desired command signal. A feedback-error learning scheme was also described where the control signal to the plant consisted of two components, one from a linear PD controller in the feedback mode and the second from a DNU in the feedforward mode. It was demonstrated through simulation studies that the control signal from the PD controller was more significant than that from the DNU during the initial phases of learning and control. As learning and control actions continued, the DNU became functionally more significant compared to the PD controller. This learning scheme was employed to control linear and simple nonlinear systems.

Although the basic processing element in the human neural system is the neuron, the power of the human brain comes from the massive parallel structure of neural networks [22, 32]. Inspired by this fact, a dynamic neural structure, with the DNU as the basic processing element, was developed in the fourth chapter. A mathematical model of a three-stage dynamic neural network and the implementation of the learning algorithm were discussed. The theory of functional approximation occupies a significant place in the field of neural networks. In the existing literature, the emphasis has been on the study of functional approximation using static neural networks. The approximation theory involving dynamic neural networks has not been developed before. Toward this goal, both theoretical and computer simulation studies of functional approximation for a multi-stage dynamic neural network, using linear and trigonometric polynomials, have been discussed in this chapter. This feature of dynamic neural networks has been exploited in synthesizing a control scheme for the direct control of unknown nonlinear systems. It was demonstrated through computer simulation studies considering different nonlinear system models that a neural network-based control scheme was system independent. A brief performance comparison of this control scheme with the conventional model-reference adaptive controller (MRAC) was made. It was also shown that the slope of the nonlinear function in the DNU structure plays an important role in overall system performance. An improper selection of the slope may lead to instability.

The focus of the fifth chapter was to develop a dynamic neural network with an adaptable slope for the nonlinear activation operator. In this context, a modified DNU architecture was proposed. Thus, the DNU was comprised of two operations: (i) the synaptic operation and (ii) the somatic operation. The synaptic operation provided adaptation in the feedforward and feedback weights, while the somatic operation provided an optimal slope of the sigmoidal function. The latter operation was referred to as *somatic adaptation*. Modifications to the learning algorithm and the implementation scheme developed in the second chapter were also made. A three-stage dynamic neural network, with the modified DNU as the functioning element, was used in synthesizing a controller for unknown nonlinear systems.

A new neural network structure called the *dynamic neural processor* (DNP) was developed in the sixth chapter. The motivation for the development of this model was on the fact that neural activities of any complexity in the human brain [41, 91] depend upon the interaction of antagonistic neural subpopulations, namely excitatory and inhibitory neurons. The DNP consisted of two DNUs configured to function as excitatory and inhibitory neurons. An algorithm was developed in this chapter to make the self- and inter-subpopulation feedback connections adaptable. The transient behavior of DNP was briefly discussed. Four applications of the DNP were elucidated in this chapter. The functional approximation capability of the DNP was demonstrated in the first application. In the second application, the DNP was employed to compute the inverse kinematic transformations of a two-link robot used as a model of the human leg. A brief performance comparison of recurrent neural networks and the DNP, as applied to the functional approximation and inverse kinematic transformations of a two-link robot, was also made. It was demonstrated in the third application that the DNP could be used for the adaptive control of unknown nonlinear dynamic systems. Due to the parallel architecture of the DNP, it could be easily employed for the coordination and control of multiple subsystems as was demonstrated in the last application. Based on the concept of neural subpopulations, a generalized dynamic neural model was also proposed in this chapter.

7.2 Contributions of the Thesis

1. The development of a dynamic neural unit (DNU) in this thesis is a unique contribution in modeling biological neurons incorporating synaptic delays with feedforward and feedback paths. The topology of the DNU was inspired by the structure of reverberating circuits in the CNS, and the development suggests the biological plausibility toward the design of artificial neural networks.

2. The development of functional approximation for the dynamic neural networks is also an original contribution of this work. The utilization of this concept could possibly lead to a generalized approximation theory for neural networks to facilitate learning of a given nonlinear function to a desired degree of accuracy.

3. A major contribution of the thesis was the development of the dynamic neural processor (DNP) based on the concept of antagonistic subpopulations of neurons. This structure may lead to a different direction of research in the area of neural networks. This is because the DNP could possibly provide insights into some of the questions clouding the neural network field, such as the biological basis, and the number of layers required in the network for a given application. The DNP structure is completely different from the conventionally used recurrent (Hopfield) dynamic neural network in the following ways: (i) the Hopfield network involves a static neuron in the feedforward path; (ii) the network lacks self-feedback for each neuron; and (iii) the feedback paths are non-adaptable.

The performance of dynamic neural network structures developed in this thesis was compared, through computer simulation studies, with conventional structures. In particular, the dynamic neural network-based control scheme was compared with proportional-plus-derivative (PD) and model-reference adaptive controllers. The performance of the former was found to be much better compared the traditional control techniques. In some situations, the latter failed to provide the desired performance. As well, the performance of the DNP was compared with recurrent neural networks for the following tasks: functional approximation, computation of inverse kinematic transformations of a two-link robot, and backing a trailer truck to the loading dock. In all these tasks, the performance of the DNP was found to be much better compared to recurrent neural networks.

In fulfilling the objectives of the thesis to formulate and develop dynamic neural structures, and to bring about an interaction of ideas and insights from biology and control systems, the relevance of the work in the fields of artificial neural networks, adaptive control

and robotics was evident. Though researchers are still a long way from making any significant breakthroughs into an understanding of animal (human and nonhuman) behavior, the work hints at the possibilities of developing useful biologically motivated dynamic neural structures for engineering applications.

7.3 Directions for Future Research

The theoretical analysis presented in this dissertation is only introductory. Additional mathematical studies into the stability of the DNU and dynamic neural networks are required. A detailed phase-plane analysis of the DNP needs further work. These studies would help to improve further the selection of suitable initial values of parameters for generating a global optimal solution. The major drawback of dynamic neural networks is their very limited explanation capability [115]. The solutions offered by these networks are hard to track back, unlike in feedforward neural networks. System-type procedures have to be developed to explain the internal functioning of dynamic networks.

From a technological perspective important questions regarding the overall performance, speed, stability and flexibility of dynamic neural networks need to be addressed. The basic premise of the DNU, and its related neural structures, is an eventual incorporation into hardware circuitry. Recent advances in microelectronics and opto-electronics could make this objective a reality in the near future.

This thesis has presented the basic concept of a multi-functional dynamic neural unit, and its associated dynamic neural networks. This is only the initial step because more extensive theoretical and experimental studies still must be performed on the dynamic neural networks in order to make them viable for real-time control system applications. Application of dynamic neural networks to practical problems will be a significant contribution to the field of neural networks, for it involves the study of stability and convergence in real-time.

Neural network structures can deal with imprecise data and ill-defined activities. However, the subjective phenomena such as reasoning and perceptions are often regarded beyond the domain of conventional neural network theory. It is interesting to note that *fuzzy logic* is another powerful tool for modeling uncertainties associated with human thinking and perception. In fact, the neural network approach fuses well with fuzzy logic, and some research endeavors have given birth to the so called '*fuzzy neural networks*' or '*fuzzy neural systems*' [116 - 119]. It would be very interesting and challenging to integrate the principles of fuzzy logic and dynamic neural networks to develop a completely new area of research.

References

[1] A.D. Handelman, H.L. Stephen and J.J. Gelfand, " Integrating Neural Networks and Knowledge -Based Systems for Intelligent Robotic Control ", *IEEE Control Systems Magazine*, pp. 77-87, April 1990.

[2] R.A. Jacobs, M.A. Jordan and A.G. Barto, "Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks", *Cognitive Science*, Vol. 15, No. 2, pp. 219 - 250, April/June 1991.

[3] A.G. Barto, "Connectionist Learning for Control", in Neural Networks for Control, T. Miller, R.S. Sutton and P.J. Werbos, Ed., *MIT Press*, pp. 6 - 58, March/April 1991.

[4] G.E. Hinton, "How Neural Networks Learn From Experience", *Scientific American*, pp. 145-151, Sept. 1992.

[5] B. Widrow and M.A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Medaline, and Backpropagation", *Proc. of IEEE*, Vol. 78, No. 9, pp. 1415-1442, Sept. 1990.

[6] M.M. Gupta and D.H. Rao, "Neuro-Control Systems: A Tutorial", in Neuro-Control Systems: Theory and Applications, pp. 1-44, Eds., M.M. Gupta and D.H. Rao, *IEEE Press*, March 1994.

[7] K.G. Shin and X. Cui, "Design of a Knowledge-Based Controller for Intelligent Control Systems", *IEEE Trans.Systems, Man and Cybernetics*, Vol. 21, No. 2, pp. 368 - 375, March/April 1991.

[8] R. Ortega and Y. Tang, "Robustness of Adaptive Controllers - a Survey", *Automatica*, Vol. 25, No. 5, pp. 651-677, 1989.

[9] M. Vidyasagar, Control Systems Synthesis: A Factorization Approach, *MIT Press Cambridge*, Massachusetts, 1985.

[10] D.C. McFarlance and K. Glover, "Robust Controller Design Using Normalized Coprime Factor Plant Descriptions", in Thoma, M. and Wyner, A., Ed., Lecture Notes in Control and Information Sciences, No. 138, *Springer-Verlag,* Berlin, 1989.

[11] C. Abdallah, D. Dawson and M. Jamshidi, "Survey of Robust Control for Rigid Robots", *IEEE Control System Magazine*, pp. 24-30, Feb. 1991.

[12] S.P. Bhattacharya, Robust Stabilization Against Structured Perturbations, in M. Thoma and A. Wyner, Ed., Lecture Notes in Control and Information Sciences, No. 99, *Springer-Verlag,* Berlin, 1989.

[13] P. Dorato and R.K. Yedavalli, Recent Advances in Robust Control, Ed., *IEEE Press*, New York, 1990.

[14] M.M. Gupta, Adaptive Methods for Control System Design, Ed., *IEEE Press*, New York, 1986.

[15] K.J. Astrom, "Adaptive Feedback Control", *Proc. IEEE*, Vol. 75, No. 2, pp. 185 - 217, Feb. 1987.

[16] K.S. Narendra and A.M. Annaswamy, Stable Adaptive Systems, *Prentice Hall*, Englewood, Cliffs, New Jersey, 1989.

[17] P.J. Antsaklis and K.M. Passino, "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues", *Int. J. of Intelligent and Robotic Systems*, pp. 315 - 342, 1989.

[18] M.D. Peek and P.J. Antsaklis, "Parameter Learning for Performance Adaptation", *IEEE Control Systems Magazine*, pp. 3 -11, Dec. 1990.

[19] K.S. Fu, "Learning Control Systems - Review and Outlook", *IEEE Trans. on Automatic Control*, pp. 210-221, April 1970.

[20] P.K. Simpson, Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations, *Pregamon Press, New York*, 1990.

[21] D.J. Burr, "Experiments on Neural Net Recognition of Spoken and Written Text", *IEEE Trans. Acoustics, Speech and Signal Processing*, Vol. 36, No. 7, pp. 1162 - 1168, July 1988.

[22] K. Fukushima, S. Miyake and T. Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 13, No. 5, pp. 826 - 834, Sept/Oct. 1983.

[23] W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115 - 133, 1943.

[24] D.O. Hebb, The Organization of Behavior, *John Wiley and Sons*, New York, 1949.

[25] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, Vol. 65, pp. 386 - 408, 1959.

[26] B. Widrow and M.E. Hoff, "Adaptive Switching Circuits", *IREWESCON Convention Record*, IRS, New York, 1960.

[27] M.L. Minsky and S.A. Papert, Perceptrons, *MIT Press*, Cambridge, MA, 1969.

[28] K.J. Hunt, D. Sbarbaro, R. Zbikowski and P.J. Gawthrop, "Neural Networks for Control Systems- A Survey", *Automatica*, Vol. 28, No. 6, pp. 1083-1112, 1992.

[29] D.R. Hush and B.G. Horne, "Progress in Supervised Neural Networks", *IEEE Signal Processing Magazine*, Vol. 10, No. 1, pp. 8-39, Jan. 1993.

[30] J.A. Anderson, "Cognitive and Psychological Computation with Neural Models", *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 13, pp. 799-815, 1983.

[31] P.D. Wasserman, Neural Computing: Theory and Practice, *Van Nostrand*, New York, 1989.

[32] R. Hecht-Nielsen, "Neurocomputing: Picking the Human Brain", *IEEE Spectrum*, Vol. 25, pp. 36 - 41, 1988.

[33] J.J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like of Those Two-State Neurons", *Proc. of the National Academy of Sciences*, Vol. 81, pp. 3088- 3092, 1984.

[34] K.S. Narendra and K. Parthasarthy, "Identification and Control of Dynamical Systems Using Neural Networks", *IEEE Trans. Neural Networks*, Vol. 1, No. 1, pp. 4-27, March 1990.

[35] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K.J. Lang, "Phoneme Recognition Using Time-Delay Neural Networks", *IEEE Trans. Acoustics, Speech and Signal Processing*, Vol. 37, No. 3, pp. 328-339, March 1989.

[36] G.E. Fischbach, "Mind and Brain", *Scientific American*, pp. 48-57, Sept. 1992.

[37] J. J. Hopfield, "Artificial Neural Networks are Coming", *IEEE Expert*, An Interview by W. Myers, pp. 3-6, April 1990.

[38] M.M. Gupta and D.H. Rao, " Dynamic Neural Units in the Control of Linear and Nonlinear Systems", *Int. Joint Conference on Neural Networks (IJCNN)*, pp. 100-105, Baltimore, June 9-12, 1992.

[39] M.M. Gupta and D.H. Rao, "Dynamic Neural Units With Applications to the Control of Unknown Nonlinear Systems", *The Journal of Intelligent and Fuzzy Systems*, Vol. 1, No. 1, pp. 73-92, Jan. 1993.

[40] A.C. Guyton, Text Book of Medical Physiology, *W.B. Saunders Company,* Philadelphia, 1987.

[41] H.R. Wilson and J.D. Cowan, "Excitatory and Inhibitory Interactions in Localized Populations of Model Neurons", *Biophysical Journal*, Vol. 12, pp. 1-24, 1972.

[42] M.M. Gupta and G.K. Knopf, "A Multitask Visual Information Processor with a Biologically Motivated Design", *Journal of Visual Communication and Image Representation*, Vol. 3, No. 3, pp. 230-246, Sept. 1992.

[43] K. Kishimoto and S.-I. Amari, "Existence and Stability of Local Excitations in Homogeneous Neural Fields", *Journal of Mathematical Biology*, Vol. 7, pp. 303-318, 1979.

[44] S. Grossberg, "Nonlinear Neural Networks: Principles, Mechanisms and Architectures", *Neural Networks,* Vol. 1, pp. 17-61, 1988.

[45] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. of the National Academy of Sciences*, Vol. 79, pp. 2554 - 2558, 1982.

[46] Y.Z. Tsypkin, Adaptation and Learning in Automatic Systems, *Academic Press*, New York, 1971.

[47] K.S. Narendra and R.M. Wheeler, Jr., "Recent Advances in Learning Automata", Adapative and Learning Systems, K.S. Narendra, Ed., *Plenum Press*, New York, 1985.

[48] J.J. Shynk, "Adaptive IIR Filtering", *IEEE ASSP Magazine*, pp. 4-21, April 1989.

[49] B. Widrow and R. Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition", *IEEE Computer*, pp. 25-39, March 1988.

[50] C.P. Tou, "Inverse Adaptive Modeling of Satellite Communication Channels", *IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, pp. 471-474, Victoria, June 1-2, 1989.

[51] B. Widrow, D. Shur and S. Shaffer, "On Adaptive Inverse Control", *15th Asilomar Conf. on Circuits, Systems and Computers*, pp. 185-189, Nov. 9-11, 1981.

[52] B. Widrow, "Adaptive Inverse Control", *IFAC Adaptive Systems in Control and Signal Processing*, Sweden, pp. 1-5, 1986.

[53] C.R. Johnson Jr, "Admissibility in Blind Adaptive Channel Equalization", *IEEE Control Systems Magazine*, pp. 3-15, Jan. 1991.

[54] K.J. Hunt and D. Sbarbaro, "Neural Networks for Nonlinear Internal Model Control", *IEE Proceedings-D*, Vol. 138, No. 5, pp. 431-438, Sept. 1991.

[55] D.A. Hoskins, J.N. Hwang and J. Vagners, "Iterative Inversion of Neural Networks and Its Application to Adaptive Control", *IEEE Trans. on Neural Networks*, Vol. 3, No. 2, pp. 292-301, March 1992.

[56] D.H. Rao and M.M. Gupta, "Dynamic Neural Adaptive Control Schemes", *American Control Conference*, San Francisco, pp. 1450-1454, June 2-4, 1993.

[57] M.M. Gupta, D.H. Rao and P.N. Nikiforuk, "Neuro-Controller with Dynamic Learning and Adaptation", *Int. Journal of Intelligent and Robotic Systems*, Vol. 7, No. 2, pp. 151-173, April 1993.

[58] H. Miyamoto, M. Kawato, T. Setoyama and R. Suzuki, "Feedback-Error-Learning Neural Network for Trajectory Control of a Robotic Manipulator", *Neural Networks*, Vol. 1, pp. 251-265, 1988.

[59] B. Mulgrew and C.F.N. Cowan, Adaptive Filters and Equalizers, *Kluwer Academic Publishers*, Boston, 1988.

[60] W.B. Mikhael, F.H. Wu, L.G. Kazovsky, G.S. Kang and L. J. Fransen, "Adaptive Filters with Individual Adaptation of Parameters", *IEEE Trans. on Circuits and Systems*, Vol. 33, No. 7, pp. 677-686, 1986.

[61] T. Yamada and T. Yabuta, "Neural Network Controller Using Autotuning Method for Nonlinear Functions", *IEEE Trans. on Neural Networks* , Vol. 3, No. 4, pp. 595-601, July 1992.

[62] R. E. Nordgren and P.H. Meckl, "An Analytical Comparison of a Neural Network and a Model-Based Adaptive Controller", *IEEE Trans. on Neural Networks* , Vol. 4, No. 4, pp. 595-601, July 1993.

[63] X. Cui and K.G. Shin, "Direct Control and Coordination Using Neural Networks", *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 23, No. 3, pp. 686-698, May/June 1993.

[64] C.L. Philips and R.D. Harbour, Feedback Control Systems, *Prentice Hall*, New Jersey, 1988.

[65] D.H. Rao, P.N. Nikiforuk, M.M. Gupta and H.C. Wood, "Neural Equalization of Communication Channels", *IEEE Conf. on Communications, Computers and Power in the Modern Environment,* Saskatoon, pp. 282-290, May 17-18, 1993.

[66] M.M. Gupta, D.H. Rao and J. Gao, "Learning and Adaptation in Neural Control of Higher-Order Linear Plants", *American Control Conference*, Chicago, pp. 3044 - 3048, June 24-26, 1992.

[67] T. Poggio and F. Girosi, "Networks for Approximation and Learning", *Proc. IEEE*, Vol. 78, No. 9, pp. 1481-1497, Sept. 1990.

[68] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals, and Systems* , pp. 303-314, Vol. 2, 1989.

[69] K. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, Vol. 2, pp. 183-192, 1989.

[70] K. Hornik, M. Stinchecombe and H. White, "Multi-Layer Feed forward Networks are Universal Approximators", *Neural Networks*, Vol. 2, pp. 359-366, 1989.

[71] N.E. Cotter, "The Stone-Weierstrass Theorem and Its Applications to Neural Networks", *IEEE Trans. on Neural Networks*, Vol. 1, No. 4, pp. 290-295, 1990.

[72] E.K. Blum and L.K. Li, "Approximation Theory and Feedforward Networks", *Neural Networks*, Vol. 4, pp. 511-515, 1991.

[73] A.R. Gallant and H. White, "There Exists a Neural Network That Does Not Make Avoidable Mistakes", in *Proc. IEEE on Neural Networks*, Vol. I, pp. 657-664, San Diego, 1988.

[74] P. Cardaliaguet and G. Euvard, "Approximation of a Function and Its Derivative with a Neural Network", *Neural Networks,* Vol. 5, pp. 207-220, 1992.

[75] R. Hecht-Nielsen, "Kolomogorov's Mapping Neural Network Existence Theorem", in *Proc. IEEE on Neural Networks*, Vol. II, pp. 11-14, San Diego, 1987.

[76] N.E. Cotter and T.J. Gullerm, "The CMAC and a Theorem of Kolmogorov", *Neural Networks*, Vol. 5, pp. 221-228, 1992.

[77] V. Kurkova, "Kolmogorov's Theorem and Multilayer Neural Networks", *Neural Networks,* Vol. 5, pp. 501-506, 1992.

[78] F. Girosi and T. Poggio, "Representation of Properties of Networks: Kolomogorov's Theorem is Irrelevant", *Neural Computation,* Vol. 63, pp. 169-176, 1990.

[79] D.H. Rao and M.M. Gupta, "Dynamic Neural Units and Function Approximation", *IEEE Conf. on Neural Networks*, San Francisco, pp. 743-748, March 28-April 1, 1993.

[80] L.V. Kantorovich and G.P. Akilov, Functional Analysis, Translated by H.L. Silcock, *Pergamon Press*, NY, 1982.

[81] P.P. Korovkin, Linear Operators and Approximation Theory, *Hindustan Publishing Corp.*, Delhi, 1960.

[82] G.A. Watson, Approximation Theory and Numerical Methods, *John Wiley and Sons*, New York, 1980.

[83] A.N. Kolmogorov and S.V. Fomin, Introductory Real Analysis, Translated by R.A. Silverman, *Prentice-Hall Inc.*, NJ, 1970.

[84] S.R. Chi, R. Shoureshi and M. Tenorio, "Neural Networks for System Identification", *IEEE Control Systems Magazine*, Vol. 10, pp. 31-34, 1990.

[85] S. Chen, S.A. Billings and P.M. Grant, "Nonlinear System Identification Using Neural Networks", *Int. Journal of Control*, Vol. 51, No. 6, pp. 1191-1214, 1990.

[86] S.A. Billings, H.B. Jamaluddin and S. Chen, "Properties of Neural Networks with Applications to Modeling Nonlinear Dynamical Systems, *Int. Journal of Control*, Vol. 55, No. 1, pp. 193-224, 1992.

[87] T. Yabuta and T. Yamada, "Neural Network Controller Characteristics with regard to Adaptive Control", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 22, No. 1, pp. 170-176, Jan/Feb. 1991.

[88] M. Vidyasagar, Nonlinear Systems Analysis, *Prentice-Hall*, New Jersey, 1978.

[89] M.M. Gupta and D.H. Rao, "Synaptic and Somatic Adaptations in Dynamic Neural Networks", *Second Int. Conf. on Fuzzy Logic and Neural Networks*, Fukuoka, Japan, pp. 173-177, July 17-22, 1992.

[90] W.J. Freeman, "Dynamics of Image Formation by Nerve Cell Assemblies", in E. Basar, H. Flohr, H. Haken and A.J. Mandell, Ed., Synergitcs of the Brain, Berlin, *Springer Verlag*, 1983.

[91] W.J. Freeman, *Mass Action in the nervous System*, Academic Press, NY, 1975.

[92] S.-I. Amari, "Neural Theory of Association and Concept Formation", *Biological Cybernetics*, Vol. 26, pp. 175-185, 1977.

[93] W.J. Freeman, "Linear Analysis of the Dynamics of Neural Masses", *Biophysical Journal*, Vol. 1, *Ann. R. Biophy.*, pp. 225-256, 1972.

[94] P. Strumillo and T.S. Durani, "Simulations of Cardiac Arrhythmia Based on Dynamical Interactions Between Neural Models of Cardiac Pacemakers", *IEE Publication No. 349, Second Int. Conf. on Artificial Neural Networks*, pp. 195-199, Nov. 18-20, 1991.

[95] D.H. Rao and M.M. Gupta, "A Multi-Functional Dynamic Neural Processor for Control Applications", *American Control Conference*, San Francisco, pp. 2902-2906, June 2-4, 1993.

[96] D.H. Rao, P.N. Nikiforuk and M.M. Gupta, "A Central Pattern Generator Model Using Dynamic Neural Processor", *World Congress on Neural Networks*, Portland, Vol. IV, pp. 533-536, July 11-15, 1993.

[97] D.H. Rao, M.M. Gupta and H.C. Wood, "Adaptive Tracking in Nonlinear Systems Using Neural Networks", *IEEE Conf. on Control Applications,* Sept. 13-16, 1993, Vancouver.

[98] J. Rinzel and G.B. Ermentrout, "Analysis of Neural Excitability And Oscillations", in Methods in Neuronal Modeling: From Synapses to Networks, Eds., C. Koch and I. Segev, pp. 135-169, *The MIT Press*, Cambridge, 1989.

[99] B. Kosko, "Bidirectional Associative Memories", *IEEE Trans. on Circuits and Systems*, Vol. 18, No. 1, pp. 49-60, Jan/Feb. 1988.

[100] P.J. Werbos, "Backpropagation Through Time: What It Does and How to Do It", *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550-1560, Oct. 1990.

[101] D.H. Rao, M.M. Gupta and P.N. Nikiforuk, "Performance Comparison of Dynamic Neural Processor and Recurrent Neural Networks", *Journal of Neural, Parallel and Scientific Computations,* Invited paper, (In Press, 1994)

[102] A. Guez and Z. Ahmad, "Solution to the Inverse Kinematics Problem in Robotics by Neural Networks", *IEEE Int. Conf. on Neural Networks*, San Diego, Calif., pp. 617-624, March 1988.

[103] J. Barhen, S. Gulati and M. Zak, "Neural Learning of Constrained Nonlinear Transformations", *IEEE Computer*, pp. 67-76, June 1989.

[104] G.E. Stelmach (Ed.), Motor Control: Issues and Trends, *Academic Press*, N Y, 1976.

[105] W.T. Powers, "The Nature of Robots", *Byte*, Vol. 82, pp. 96-111, 1979.

[106] W.A. Wolovich, Robotics: Basic Analysis and Design, *Holt, Rinehart and Winston,* New York, 1986.

[107] M.M. Gupta and D.H. Rao, "General Learning Scheme for Robot Coordinate Transformations Using Dynamic Neural Network", *SPIE 's Conference on Intelligent Robots and Computer Vision XI*, Boston, Vol. 2055, pp. 524-535, Sept. 7-10, 1993.

[108] X. Cui and K.G. Shin, "Intelligent Coordination of Multiple Systems With Neural Networks", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, 1991.

[109] L. Jin, M.M. Gupta and P.N. Nikiforuk, "Direct Adaptive Tracking Control Using Multilayered Neural Networks", *IEE Proceedings -D*, Vol. 140, No. 6, pp. 393-398, Nov. 1993.

[110] D.H. Rao and M.M. Gupta, "A Neural Processor for Coordinating Multiple Systems with Dynamic Uncertainties", *International Symposium on Uncertainty and Management* (ISUMA), Maryland, pp. 633-640, April 25-28, 1993.

[111] D. Nguyen and B. Widrow, "The Truck Backer-Upper: An Example of Self-Learning in Neural Network", *IEEE Control Systems Magazine*, Vol. 10, pp. 18-23, 1990.

[112] L.X. Wang and J.M. Mendel, "Generating Fuzzy Rules by Learning Through Examples", *IEEE Trans.SMC,* Vol. 22, No. 6, pp. 1414-1427, Nov/Dec 1992.

[113] S. G. Kong and B. Kosko, "Comparison of Fuzzy and Neural Truck Backer-Upper Control Systems" in B. Kosko, Neural Networks and Fuzzy Systems, *Prentice Hall*, Englewood Cliffs, New Jersey, 1992.

[114] D.H. Rao and M.M. Gupta, "A Generic Neural Model Based on Excitatory-Inhibitory Neural Population", *IEEE Joint Conf. on Neural Networks (IJCNN)*, Nagoya, Japan, Oct. 25-29, pp. 1393-1396, 1993.

[115] J.M. Zurada, Introduction to Artificial Neural Systems, *West Publishing Company*, St. Paul, MN, 1992.

[116] M.E. Cohen and D.L. Hudson, "An Expert System on Neural Network Techniques", in I.B. Turksen, Ed., *The Proceedings of NAFIP*, pp. 117-12, Toronto, June 1990.

[117] M.M. Gupta and G.K. Knopf, "Fuzzy Neural Network Approach to Control Systems", *Proc. of First Int. Symposium on Uncertainty Modeling and Analysis*, Maryland, pp. 483-488, Dec. 3-5, 1990.

[118] B. Kosko, Neural Networks and Fuzzy Systems, *Prentice Hall*, Englewood Cliffs, New Jersey, 1992.

[119] M.M. Gupta and D.H. Rao, "On the Principles of Fuzzy Neural Networks", *Journal of Fuzzy Sets and Systems* , Vol. 61, No. 1, pp. 1-18, Jan. 1994 (Invited Paper).

Appendix I: Parameter-State Signals for the Feedforward and Feedback Weights of the Modified DNU Structure

The modified DNU algorithm that accounts for both the synaptic and somatic adaptations was presented in Section 5.3.2. The feedforward parameters a_{ff_i}, i = 0, 1, 2, and the feedback parameters b_{fb_j}, j = 1, 2, were updated based on the following set of equations

$$a_{ff_i}(k+1) = a_{ff_i}(k) + \mu_{a_i} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,\mathbf{P}_{ff_i}(k)\right], \quad i = 0,1,2, \tag{5.7a}$$

and

$$b_{fb_j}(k+1) = b_{fb_j}(k) + \mu_{b_j} E\left[e(k)\,\mathrm{sech}^2[v(k)]\,\mathbf{P}_{fb_j}(k)\right], \quad j = 1,2 \tag{5.7b}$$

where the modified parameter-state signals for the feedforward and the feedback weights were given by the relations

$$\mathbf{P}_{ff_i}(k) = g_s[s(k-i)], \quad i = 0, 1, 2, \quad \text{and} \tag{5.8a}$$

$$\mathbf{P}_{fb_j}(k) = -g_s[v_1(k-j)], \quad j = 1,2. \tag{5.8b}$$

In this Appendix, the proof of Eqns. (5.8a) and (5.8b) is given in the following paragraphs.

Proof of Eqns. (5.8a) and (5.8b): From Eqn. (5.5)

$$\mathbf{P}_{ff_i}(k) = g_s \frac{\partial}{\partial a_{ff_i}(k)} \left\{ \begin{bmatrix} -b_1 & -b_2 & a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} (v_1(k-1)) \\ (v_1(k-2)) \\ (s(k)) \\ (s(k-1)) \\ (s(k-2)) \end{bmatrix} \right\}$$

$$= g_s \frac{\partial}{\partial a_{ff_i}(k)} \left\{ \begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} (s(k)) \\ (s(k-1)) \\ (s(k-2)) \end{bmatrix} \right\}, \quad i = 0, 1, 2.$$

Thus, the individual parameter-state signals for the feedforward weights are

For i = 0, $\mathbf{P}_{a_0}(k) = g_s[s(k)]$,

For i = 1, $\mathbf{P}_{a_1}(k) = g_s[s(k-1)]$, and

For i = 2, $\mathbf{P}_{a_2}(k) = g_s\,[s\,(k - 2)]$.

Therefore, the parameter-state signals for the feedforward weights are

$$\mathbf{P}_{ff_i}(k) = g_s\,[s\,(k - i)], \quad i = 0, 1, 2. \tag{I.1}$$

Similarly, to obtain the parameter-state signals for feedback weights, from Eqn. (5.5)

$$\mathbf{P}_{fb_j}(k) = g_s \frac{\partial}{\partial b_{fb_j}(k)} \left\{ \begin{bmatrix} -b_1 & -b_2 & a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} (v_1(k-1)) \\ (v_1(k-2)) \\ (s(k)) \\ (s(k-1)) \\ (s(k-2)) \end{bmatrix} \right\}.$$

Thus, the individual parameter state signals for the feedback weights are

For j = 1, $\mathbf{P}_{b_1}(k) = -\,g_s\,[v_1(k - 1)]$, and

For j = 2, $\mathbf{P}_{b_2}(k) = -\,g_s\,[v_1(k - 2)]$.

Therefore, the parameter state signals for the feedback weights may be written as

$$\mathbf{P}_{fb_j}(k) = -\,g_s\,[v_1(k - j)], \quad j = 1,2. \tag{I.2}$$

Appendix II: Generalized Learning Algorithm

In this Appendix, a learning algorithm for the generalized neural model described in the preceding subsection is derived. In an iterative learning scheme, the parameters are modified in each iteration to cause the neural output y(k) to approach the desired state $y_d(k)$. If the error between the targetted and the observed responses can be reduced to an acceptable tolerance limit, the learning scheme is said to be convergent. Let the parameter vector of the generalized model be defined as

$$\Omega \triangleq [w^F, \; w^G, w^H, w^P, \; w^A, w^B, w^C, \; g_s]^T .$$

Each component of the vector Ω is adapted in such a way so as to minimize the performance index J, defined as a square of the error, using the steepest-descent algorithm. This adaptation algorithm may be written as

$$\Omega(k + 1) = \Omega(k) + \delta\Omega(k) \qquad \text{(II.1)}$$

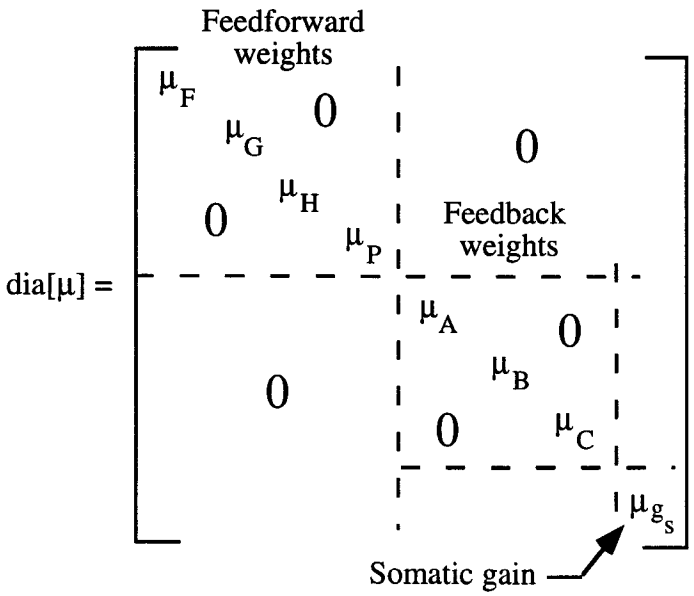
where $\Omega(k + 1)$ is the new parameter vector, $\Omega(k)$ is the present parameter vector, and $\delta\Omega(k)$ is an adaptive adjustment in the parameter vector. In the steepest-descent method, the adjustment of the parameter vector is made proportional to the negative of the gradient of the performance index J; that is,

$$\delta\Omega(k) \propto (- \nabla J), \text{ where } \nabla J = \frac{\partial J}{\partial \Omega} .$$

Thus,

$$\delta\Omega(k) = - \text{dia}[\mu] \frac{\partial J}{\partial \Omega} = - \text{dia}[\mu] \nabla J \qquad \text{(II.2)}$$

where dia[μ] is the matrix of individual adaptive gains. In the above equation, the dia[μ] is defined as



The gradient of the performance index with respect to the parameter vector Ω is given by

$$\frac{\partial J}{\partial \Omega} = \frac{1}{2} E\left[\frac{\partial [y_d(k) - y_i(k)]^2}{\partial \Omega}\right] = E\left\{ e(k)\left[- \frac{\partial y_i(k)}{\partial \Omega}\right]\right\} \qquad \text{(II.3)}$$

where

$$\frac{\partial y_i(k)}{\partial \Omega} = \left[\frac{\partial y_i(k)}{\partial w^F} \;\; \frac{\partial y_i(k)}{\partial w^G} \;\; \frac{\partial y_i(k)}{\partial w^H} \;\; \frac{\partial y_i(k)}{\partial w^P} \;\; \frac{\partial y_i(k)}{\partial w^A} \;\; \frac{\partial y_i(k)}{\partial w^B} \;\; \frac{\partial y_i(k)}{\partial w^C} \;\; \frac{\partial y_i(k)}{\partial g_s}\right]^T$$

and E[.] is an expectation operator defined as:

$$E[x(k)] = \frac{1}{K} \sum_{k=t+1-K}^{k} x(k) \quad . \qquad \text{(II.4)}$$

Feedforward Weights: The feedforward weights, represented as F in the generalized dynamic neural model, shown in Fig. 6.36, are normally referred to as the synaptic weights in a static (feedforward) neural network. However, the generalized neural model consists of three forward paths with weights G, H and P.

(i) $$\frac{\partial y_i(k)}{\partial w^F} = \frac{\partial}{\partial w^F}\left[\sum_{n=1}^{N} w_i^{Dn} \; \Psi\left[v_n(k)\right]\right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \left[\frac{\partial \Psi[v_n(k)]}{\partial v_n(k)} \frac{\partial v_n(k)}{\partial w^F} \right] = \sum_{n=1}^{N} w_i^{Dn} \Psi' [v_n(k)] \mathbf{S}_F(k) \quad \text{(II.5)}$$

where $\mathbf{S}_F(k) = \frac{\partial v_n(k)}{\partial w^F} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^F}$ represents the sensitivity signals which may be obtained as follows.

$$\frac{\partial v_n(k)}{\partial w^F} = g_{sn} \frac{\partial}{\partial w^F}$$

$$\left[\sum_{n=1}^{N} w_i^{Gn} \sum_{n=1}^{I} w_i^{Fn} s_n(k) + \sum_{n=1}^{N} w_i^{Hn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-1) + \sum_{n=1}^{N} w_i^{Pn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-2) \right]$$

$$= g_{sn} \left[\sum_{n=1}^{N} w_i^{Gn} \sum_{n=1}^{I} s_n(k) + \sum_{n=1}^{N} w_i^{Hn} \sum_{n=1}^{I} s_n(k-1) + \sum_{n=1}^{N} w_i^{Pn} \sum_{n=1}^{I} s_n(k-2) \right]$$

Therefore, Eqn. (II.5) becomes

$$\frac{\partial y_i(k)}{\partial w^F} = \sum_{n=1}^{N} w_i^{Dn} \Psi' [v_n(k)]$$

$$g_{sn} \left[\sum_{n=1}^{N} w_i^{Gn} \sum_{n=1}^{I} s_n(k) + \sum_{n=1}^{N} w_i^{Hn} \sum_{n=1}^{I} s_n(k-1) + \sum_{n=1}^{N} w_i^{Pn} \sum_{n=1}^{I} s_n(k-2) \right] \quad \text{(II.6)}$$

For a static neural network, Eqn. (II.6) simplifies to

$$\frac{\partial y_i(k)}{\partial w^F} = \sum_{n=1}^{N} w_i^{Dn} \Psi' [v_n(k)] \sum_{n=1}^{I} s_n(k) .$$

(ii) $$\frac{\partial y_i(k)}{\partial w^G} = \frac{\partial}{\partial w^G} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi [v_n(k)] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' [v_n(k)] \mathbf{S}_G(k) \quad \text{(II.7)}$$

where $\mathbf{S}_G(k)$ is obtained as follows:

$$\mathbf{S}_G(k) = \frac{\partial v_n(k)}{\partial w^G} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^G}$$

$$= g_{sn} \frac{\partial}{\partial w^G} \left[\sum_{n=1}^{N} w_i^{Gn} \sum_{n=1}^{I} w_i^{Fn} s_n(k) \right].$$

Therefore, Eqn. (II.7) becomes

$$\frac{\partial y_i(k)}{\partial w^G} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k) \qquad \text{(II.8)}$$

(iii) $$\frac{\partial y_i(k)}{\partial w^H} = \frac{\partial}{\partial w^H} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \mathbf{S}_H(k) \qquad \text{(II.9)}$$

where $\mathbf{S}_H(k)$ is obtained as follows:

$$\mathbf{S}_H(k) = \frac{\partial v_n(k)}{\partial w^H} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^H}$$

$$= g_{sn} \frac{\partial}{\partial w^H} \left[\sum_{n=1}^{N} w_i^{Hn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-1) \right].$$

Therefore, Eqn. (II.9) becomes

$$\frac{\partial y_i(k)}{\partial w^H} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-1) \qquad \text{(II.10)}$$

(iv) $$\frac{\partial y_i(k)}{\partial w^P} = \frac{\partial}{\partial w^P} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \mathbf{S}_P(k) \qquad \text{(II.11)}$$

where $\mathbf{S}_P(k)$ is obtained as follows:

$$\mathbf{S}_P(k) = \frac{\partial v_n(k)}{\partial w^P} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^P}$$

$$= g_{sn} \frac{\partial}{\partial w^P} \left[\sum_{n=1}^{N} w_i^{Pn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-2) \right].$$

Therefore, Eqn. (II.11) becomes

$$\frac{\partial y_i(k)}{\partial w^P} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-2) . \qquad \text{(II.12)}$$

Feedback Weights: The weight matrices A and B represent the internal feedback weights, while the matrix C denotes the self- and inter-subpopulation feedback connections in the generalized dynamic neural model.

(v) $$\frac{\partial y_i(k)}{\partial w^A} = \frac{\partial}{\partial w^A} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \mathbf{S}_A(k) \qquad \text{(II.12)}$$

where $\mathbf{S}_A(k)$ is obtained as follows:

$$\mathbf{S}_A(k) = \frac{\partial v_n(k)}{\partial w^A} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^A}$$

$$= g_{sn} \frac{\partial}{\partial w^A} \left[\sum_{n=1}^{N} w_i^{An} v_{1n}(k-1) \right]$$

Therefore, Eqn. (II.12) becomes

$$\frac{\partial y_i(k)}{\partial w^A} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} v_{1n}(k-1) . \qquad \text{(II.13)}$$

(vi) $$\frac{\partial y_i(k)}{\partial w^B} = \frac{\partial}{\partial w^B} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \mathbf{S}_B(k) \qquad \text{(II.14)}$$

where $\mathbf{S}_B(k)$ is obtained as follows:

$$\mathbf{S}_B(k) = \frac{\partial v_n(k)}{\partial w^B} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^B}$$

$$= g_{sn} \frac{\partial}{\partial w^B} \left[\sum_{n=1}^{N} w_i^{Bn} \; v_{1n}(k-2) \right]$$

Therefore, Eqn. (II.14) becomes

$$\frac{\partial y_i(k)}{\partial w^B} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \; v_{1n}(k-2) \; . \qquad \text{(II.15)}$$

(vii) $$\frac{\partial y_i(k)}{\partial w^C} = \frac{\partial}{\partial w^C} \left[\sum_{n=1}^{N} w_i^{Dn} \; \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \mathbf{S}_C(k) \qquad \text{(II.16)}$$

where $\mathbf{S}_C(k)$ is obtained as follows:

$$\mathbf{S}_C(k) = \frac{\partial v_n(k)}{\partial w^C} = g_{sn} \frac{\partial v_{1n}(k)}{\partial w^C}$$

$$= g_{sn} \frac{\partial}{\partial w^C} \left[\sum_{n=1}^{N} w_i^{Cn} \; u_n(k-1) \right]$$

Therefore, Eqn. (II.16) becomes

$$\frac{\partial y_i(k)}{\partial w^C} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \; u_n(k-1) \; . \qquad \text{(II.17)}$$

Somatic Gain: This parameter that controls the slope of the nonlinear activation function is normally kept constant in conventional neural networks. It was demonstrated in Chapters 4 and 5 that the somatic gain plays a significant role in the overall performance of the neural network. Modification of this parameter leads to what is called the somatic adaptation.

(viii) $$\frac{\partial y_i(k)}{\partial g_{si}} = \frac{\partial}{\partial g_{si}} \left[\sum_{n=1}^{N} w_i^{Dn} \; \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] S_{g_s}(k) \qquad \text{(II.18)}$$

where $S_{g_s}(k)$ is obtained as follows:

$$S_{g_s}(k) = \frac{\partial v_n(k)}{\partial g_{si}} = \frac{\partial}{\partial g_{si}} \left[g_{si} \, v_{1n}(k) \right]$$

$$= v_{1n}(k)$$

Therefore, Eqn. (II.18) becomes

$$\frac{\partial y_i(k)}{\partial g_{si}} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] v_{1n}(k) \, . \qquad \text{(II.19)}$$

Bias Term: Modification of the bias terms of the neurons provides a shift in the nonlinear activation functions which may be useful in the approximation of functions.

(ix) $$\frac{\partial y_i(k)}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \left[\sum_{n=1}^{N} w_i^{Dn} \Psi \left[v_n(k) \right] \right]$$

$$= \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] S_{\theta_i}(k) \qquad \text{(II.20)}$$

where $S_{\theta_i}(k)$ is obtained as follows:

$$S_{\theta_i}(k) = \frac{\partial v_n(k)}{\partial \theta_i} = g_{sn} \frac{\partial v_{1n}(k)}{\partial \theta_i} = - g_{sn} \, .$$

Therefore, Eqn. (II.20) becomes

$$\frac{\partial y_i(k)}{\partial \theta_i} = \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \left[- g_{sn} \right]. \qquad \text{(II.21)}$$

The learning algorithm to update the adaptable parameters of the generalized neural model may be summarised from Eqns. (1), (3) and (4) as follows:

$$w^F(k+1) = w^F(k) + \mu_F \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \sum_{n=1}^{I} s_n(k) \right]$$

$$w^{G}(k+1) = w^{G}(k) + \mu_{G} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k) \right]$$

$$w^{H}(k+1) = w^{H}(k) + \mu_{H} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-1) \right]$$

$$w^{P}(k+1) = w^{P}(k) + \mu_{P} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} \sum_{n=1}^{I} w_i^{Fn} s_n(k-2) \right]$$

$$w^{A}(k+1) = w^{A}(k) + \mu_{A} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} v_{1n}(k-1) \right]$$

$$w^{B}(k+1) = w^{B}(k) + \mu_{B} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} v_{1n}(k-2) \right]$$

$$w^{C}(k+1) = w^{C}(k) + \mu_{C} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] g_{sn} u_n(k-1) \right]$$

$$g_s(k+1) = g_s(k) + \mu_{g_s} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] v_{1n}(k) \right]$$

$$\theta_i(k+1) = \theta_i(k) + \mu_{\theta_i} \frac{1}{K} \sum_{k=t+1-K}^{k} \left[e(k) \sum_{n=1}^{N} w_i^{Dn} \Psi' \left[v_n(k) \right] \left[- g_{sn} \right] \right] .$$
