

**Model-based Identification and Control of
Nonlinear Dynamic Systems Using
Neural Networks**

by

Ssu-Hsin Yu

B.S., Mechanical Engineering
National Chiao Tung University (1987)

M.S., Mechanical Engineering
University of Wisconsin-Madison (1991)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

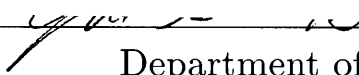
Doctor of Philosophy

at the

Massachusetts Institute of Technology

February 1996

© Massachusetts Institute of Technology 1996. All rights reserved.

Signature of Author  _____
Department of Mechanical Engineering
January 15, 1996

Certified by _____
Anuradha M. Annaswamy
Assistant Professor of Mechanical Engineering
Thesis Supervisor

Accepted by _____
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
Ain A. Sonin
Chairman, Department Graduate Committee

MAR 19 1996

LIBRARIES

Eng.



Model-based Identification and Control of Nonlinear Dynamic Systems Using Neural Networks

by

Ssu-Hsin Yu

Submitted to the Department of Mechanical Engineering
on January 15, 1996, in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Mechanical Engineering

Abstract

The difficulties in identification and control of engineering systems are due to several factors including the presence of several types of nonlinearities, and significant and unknown sources of variations in the operating conditions of the system. In many of these problems, the underlying physics that contributes to the nonlinear system characteristics can be modeled using physical laws. However, due to analytical tractability, the traditional approach has not always made effective use of available physically-based models.

In this thesis, new parameter estimation and control techniques, which take advantage of prior physical knowledge of dynamic systems, are presented. The tools used are artificial neural networks (ANN). For parameter estimation problems, the scheme denoted as θ -adaptive neural networks (TANN) is developed. TANN is useful for systems where the unknown parameters occur nonlinearly. For control problems, we consider two classes of nonlinear stabilization problems including systems with unknown parameters and systems with unknown controller structures. TANN is implemented on the former class to adjust the controller parameters so that stabilization can be achieved in the presence of parametric uncertainty. For systems with unknown controller structures, ANN are trained off-line to generate nonlinear controllers that contribute to the decrease of specified positive definite functions of state variables. Conditions under which these two controllers result in stable closed-loop systems are given.

The framework introduced in this thesis naturally lends itself to a mathematically tractable problem formulation and can be applied to general nonlinear systems. The neural network training procedure, whether it is for the purpose of parameter estimation or control, also permits the use of more efficient training algorithms and leads to a larger region of stability for a wide class of dynamic systems.

Thesis Supervisor: Anuradha M. Annaswamy
Title: Assistant Professor of Mechanical Engineering

Acknowledgments

First of all, I am deeply grateful to Professor Annaswamy for her guidance during the last three and a half years. Her persistence and passion on this research have been a great inspiration for me over the course of my doctoral thesis. I would like to express my gratitude to my thesis committee members, Professor Haruhiko Asada, Professor Michael Jordan and Professor Thomas Sheridan for their constructive comments and invaluable suggestions.

Many people have contributed to the completion of this dissertation and made my stay at MIT rewarding and surprisingly enjoyable. In particular, I would like to thank my colleagues at the adaptive control laboratory, Rienk Bakker, Fredrik Skantze, Steingrimur Karason, Kathleen Misovec and Aleksandar Kojic for many fruitful discussions with them. Their diverse backgrounds also helped me appreciate different cultures and thinkings. I am indebted to Hwa-Ping Chang for his encouragement and friendship both at MIT and at University of Wisconsin-Madison. He was always one step ahead of me to smooth my transition from Taiwan to Wisconsin to MIT.

Finally, and above all, I thank my parents and my wife for their love, support and encouragement. They have made all these meaningful.

自序

對我而言，家人是我最大的精神支柱。特別是我的父母。不論在精神上或物質上，他們始終全力支持我所做的每一個決定。而他們豁達樂觀的人生態度，面對挫折打擊時展現的毅力與韌性，更激勵我渡過低潮，克服困難。在此要向他們獻上最深的謝意和敬意。沒有他們的支持，我絕對無法堅持到底。

另外也要感謝我的妻子。感謝她對我全然不變的信任。二人同心，其利斷金，共同走過「死蔭的幽谷」後，相信必定有「豐美的水草」。

Contents

1	Introduction	12
1.1	Motivation	12
1.2	Contribution of Thesis	13
1.3	Related Work and Previous Literature	15
1.4	Synopsis of Thesis	18
2	Preliminaries	20
2.1	Introduction	20
2.2	Artificial Neural Networks	20
2.3	Mathematical Preliminaries	23
2.4	The Quadratic Optimization Method	29
3	Parameter Estimation Using θ-Adaptive Neural Networks	33
3.1	Introduction	33
3.2	Problem Statement	35
3.3	The Block Parameter Estimation Method	37
3.3.1	The Block Estimation Algorithm	37
3.3.2	Training Scheme	40
3.4	The Recursive Parameter Estimation Method	41
3.4.1	The Recursive Estimation Algorithm	41
3.4.2	Training Scheme	44
3.4.3	Convergence Analysis	49
3.5	Comparison with Extended Kalman Filter	52

3.6	Summary and Remarks	56
4	Adaptive Control of Nonlinear Dynamic Systems Using θ-Adaptive Neural Networks	59
4.1	Introduction	59
4.2	Problem Statement	62
4.2.1	Assumptions	62
4.2.2	The Adaptive Neural Controller	64
4.3	TANN Algorithms for Control	67
4.4	Training of TANN for Control	71
4.5	Proof of Stability	76
4.5.1	Discussions	89
4.6	Complexity Reduction for Partially Linear Systems	93
4.7	Summary and Remarks	96
5	Stable Control of Nonlinear Dynamic Systems Using Neural Networks	98
5.1	Introduction	98
5.2	Problem Statement	99
5.3	Stable Neural Controller	100
5.3.1	Training Scheme	102
5.3.2	Stability Analysis	105
5.4	Summary and Remarks	110
6	Simulation Results	112
6.1	Introduction	112
6.2	Parameter Estimation Using TANN	112
6.2.1	Example 1: A Simple System	113
6.2.2	Example 2: Parameter Estimation for a Regression Model	121
6.2.3	Example 3: Parameter Estimation for a Discrete-time State Space Model	124

6.2.4	Example 4: Parameter Estimation for a Continuous-time State Space Model	131
6.3	Adaptive Control Using TANN	136
6.4	Stable Neural Controller	139
6.5	Summary and Remarks	141
7	Conclusion	151

Chapter 1

Introduction

1.1 Motivation

An increasing demand in the performance specifications and the concomitantly present complexity of dynamic systems mandate the use of sophisticated information processing and control in almost all branches of engineering systems. The promise of fast computation, versatile representational ability of nonlinear maps, fault-tolerance, and the capability to generate quick, robust, sub-optimal solutions from artificial neural networks (ANN) make the latter an ideal candidate for carrying out such a sophisticated identification or control task.

The field of dynamic systems theory deals with identification, prediction, and control problems where variables interact over time. In many of these problems, the task invariably is to determine a nonlinear map between two quantities in the system. For instance, the function of the nonlinear map is that of a controller if the concerned quantities are system errors and the control input; the nonlinear relation may correspond to that of an identifier if the quantities are system inputs and system outputs. Difficulties arise in carrying out the relevant tasks since there are uncertainties associated with these nonlinear maps. Given the potential and proven success of neural networks in tasks related to pattern recognition, classification, and such [32, 36], the obvious question that arises is then the applicability of neural networks in approximating such nonlinear maps.

In many of these problems, the underlying mechanisms that contribute to the nonlinear system characteristics can be modeled using physical laws. Conservation equations due to mass, momentum, and energy balance relations together with the constitutive laws provide a framework for accurately representing the governing nonlinear time-varying characteristics. Despite the presence of complex physically based models, identification and control procedures that have been developed for dynamic systems have not always made effective use of them. For the sake of analytical tractability, simplifications are often introduced in the system model. The question is whether the use of a neural network will allow the development of a more powerful identification and control procedure which requires few assumptions and less simplifications.

Adaptive control addresses dynamic systems with parametric uncertainty. A common assumption in this field is that the uncertain parameters occur linearly. Furthermore, even without parametric uncertainty, the controller structure itself could be very complex for nonlinear systems. Hence, attention has been focused on linear systems or some special classes of nonlinear systems. Unfortunately, many system models do not fall into these categories without further simplification. In addition, such simplification would usually limit the applicability of the resulting model such as around some neighborhood of an operating point. To avoid this problem, we propose using neural networks to construct such nonlinear maps, whether the map is from measurable signals to parameters or from system outputs to control inputs, by taking advantage of their ability in function approximation. The approach should be general enough so as to be applicable to wide classes of nonlinear systems and, nonetheless, retain mathematical rigor of the analysis. Hence, when the schemes are implemented, performance can be guaranteed over a wide range of operating points.

1.2 Contribution of Thesis

The contribution of this thesis consists of three parts:

1. Develop neural network based estimation schemes, which are capable of determining parameters occurring nonlinearly.
2. Establish an adaptive control algorithm and its corresponding stability property for nonlinear systems with parametric uncertainty.
3. For nonlinear systems with unknown controller structures, a new strategy for neural network training is developed to construct a stable nonlinear controller.

They are explained in detail below.

A significant portion of parameter estimation methods concerns linear systems and only a limited class of nonlinear systems can be solved analytically using available tools. Whether linear or nonlinear, in most of these problems, the identification approach requires the unknown parameters to occur linearly. However, many system models derived based on physical laws involve nonlinear parametrization. To bridge the gap, two neural network based parameter estimation algorithms are proposed in this thesis. The local conditions under which such algorithms exist are examined. The convergence property of the algorithms are analyzed. A neural network training procedure that would achieve such property is also suggested. The algorithms are shown to be suitable for a wide class of nonlinear systems where existing methods cannot be used.

Control of nonlinear systems are even more challenging especially for those with uncertain parameters. For this class of problems, as in the parameter estimation case, research in adaptive control focuses on linear parametrization. In this thesis, a neural network algorithm to update unknown controller parameters that occur nonlinearly is incorporated into the controller structure to stabilize nonlinear systems with parametric uncertainty. Conditions under which the algorithm guarantees stability are given explicitly. Moreover, these conditions are general enough so that they can also serve as design guidelines for adaptive controllers under plant nonlinearity.

Unlike linear adaptive control where a general controller structure to stabilize a system can be obtained with only the knowledge of relative degrees, stabilizing controllers for nonlinear systems are hard to determine. A few approaches have been

suggested for utilizing neural networks in the context of a general controller structure. However, there are several limitations for these approaches. First of all, a majority of them can only be applied to discrete-time systems. Secondly, stability of the closed-loop systems is left unanswered. To avoid these problems, a new approach for nonlinear controller designs is proposed in this thesis. This approach can deal with discrete-time as well as continuous-time nonlinear systems. Furthermore, stability of the resulting systems becomes transparent following the proposed method. As a consequence, the approach achieves large region of convergence by utilizing the function approximation ability of neural networks while still maintaining mathematical rigor of the stability analysis.

1.3 Related Work and Previous Literature

Significant research has been done in the fields of system identification and adaptive control for parameter estimation. Most of the problems that can be solved exactly require the unknown parameters to enter linearly [42, 37, 18]. This also includes nonlinear systems where the unknown parameters are linearly related to the output [17, 11, 16, 56, 27, 5, 28]. The recursive least-squares algorithm and many of its variants (e.g. [18, 37]) are among the most popular in the discrete case. For its continuous-time counterpart, gradient type schemes are most commonly used [42], though some faster updating algorithms were also proposed [28]. On the other hand, for problems where the parameters occur nonlinearly, only approximate algorithms are developed for general models. These algorithms usually make use of the current parameter estimate to establish an approximate direction in improving estimation. This includes the nonlinear least-squares algorithm and the bootstrap methods [62, 18]. The extended Kalman filter method for parameter estimation also falls into this class (e.g. [23]). Whether the estimate would converge to the true value depends on the underlying nonlinearity and how good the initial estimate is. When these algorithms are implemented on adaptive control of nonlinear systems, stability property of the closed-loop system is usually unknown.

Artificial neural networks (ANN) had been recognized for their capability in representing complex functions. In most of their applications to system identification, ANN are used as general model structures, replacing standard linear or bilinear models. The role of the networks is to mimic the output of unknown systems given the same input signals. Due to their representative ability, many successful applications have been reported (e.g. [43, 10, 9, 14, 51, 58]).

In recent years, there has been much research on the use of neural networks in control, especially in problems where nonlinearity dominates. A significant portion of them utilizes neural networks as nonlinear models of the underlying nonlinearity, for example, [43, 55, 13, 12, 24, 26, 44, 34, 1, 46, 39, 35, 54, 3]. Broadly, the results of these papers can be classified into two classes on the basis of the functionality of the neural network. In the first class [43, 55, 13, 12, 24, 35, 54], the neural network is to be trained on-line so that it mimics an adaptive controller and meet a regulation or tracking objective, whereas in the second class [26, 44, 34, 1, 46, 39, 3], the network is required to function as an optimal controller with an appropriately chosen cost function.

In the first class, with the networks as identifiers and/or controllers, the problem reduces to finding a stable algorithm for adjusting the weights of the network. Both the development of these algorithms as well as the stability analysis are carried out in these papers in a manner similar to linear adaptive control methods as in [42]. For example, in [43, 55], unknown nonlinear parts in dynamic systems are modeled using neural networks. The information is then used by the controllers to adaptively cancel those nonlinearities and replace them by desired dynamics with either multi-layered neural networks (MNN) [43] or radial basis functions (RBF) networks [55]. However, there is no stability proof given in [43], while the proof of stability in [55] is for a special class of affine systems. For systems which are feedback linearizable but with unknown nonlinearities, the results in [13, 12, 24] pertain to using neural networks to perform the task of a feedback-linearizing controller. However, the conditions under which the system is stable require that the initial weights of the network be close to a certain set of desired weights [13, 12]. This is extremely difficult to check, if

not impossible, since the weights in the network do not have any physical meaning. Based on the knowledge gained from the linear adaptive control, modification in the adaptive control algorithms to improve stability has also been studied. Since a neural network is only an approximation to the underlying nonlinear system, there is always a residual error between the true system and the network model. This effect is similar to that of an adaptive system under bounded external noise, which has long been understood in linear adaptive control to result in unstable mechanism if the standard adaptive law is used [49]. To overcome this problem, modifications in the adaptation algorithms, similar to those in the adaptive control, was introduced in the weight updating algorithm to avoid instability in the presence of small residual error [13, 35]. Furthermore, it is well known in adaptive control that a brute-force correction of controller parameters based on the gradient of the output error can result in instability even for some classes of linear systems [48, 45]. To avoid such instabilities, some researchers proposed only varying network weights that are linearly related to the outputs of the network, such as the RBF networks, so as to achieve a stable updating rule [55, 35, 54].

Up to this point, the development of nonlinear adaptive control using neural networks parallels that of linear adaptive control, and many ideas can be carried directly over. Unfortunately, unlike linear adaptive control where a general controller structure to stabilize a system can be obtained with only the knowledge of relative degrees, stabilizing controllers for nonlinear systems are hard to determine. As a consequence, all of the above researches focus on nonlinear systems whose stabilizing controllers are readily available once some unknown nonlinear parts are identified, such as $x^n = f(x^{n-1}, \dots, x) + bu$ with full state feedback where f is to be estimated by a neural network. Even though some approaches have been suggested for utilizing neural networks in the context of a general controller structure [29], the stability implications are unknown.

In the absence of general controller structures for nonlinear systems, using ANN to construct stabilizing controllers also appears in the literature, especially in the discrete-time case. A straightforward approach is to find a neural network map that

approximates the inverse dynamics of the nonlinear system. Once the network is trained successfully, given a desired trajectory, the network would then send appropriate input signals to drive the system to follow that trajectory [26]. Since inverting dynamics may be too restrictive and may sometimes result in huge control signals that are not practical for some systems, several researchers have posed the problem as one of optimization to make it more tractable, whose results can be grouped under the second class of neural control methods. For example, in [44, 34], using a series of neural networks, it is sought to control a dynamic system to the desired target gradually, where each network in the series corresponds to a map from the measurement to the input signals of the system at a particular time. The complexity of training, not surprisingly, increases dramatically as the number of steps increase. This idea can be extended to solve the N-stage optimal control problem by finding the controller mapping that minimizes a cost function of states and inputs as is done in [1, 46, 39, 3]. In all these papers, however, the issue of stability of the closed-loop system with the neural network as a controller is not addressed.

1.4 Synopsis of Thesis

This thesis is organized into seven chapters as follows.

Background material relating to ANN is introduced in Chapter 2. Results in analysis, control theory and optimization useful in later derivations are also given in this chapter.

In Chapter 3, θ -adaptive neural networks (TANN) for parameter estimation in nonlinear systems is proposed. Two methods, the block estimation method and the recursive estimation method, are discussed. The idea behind the two algorithms and the structures, training procedure, on-line implementation as well as convergence analysis of the algorithms are also explained. One common method for estimating parameters in nonlinear systems is the extended Kalman filter. This method is compared in this chapter to the recursive method to illustrate the fundamental difference between TANN and those approximate algorithms.

TANN for parameter estimation developed in Chapter 3 is extended to adaptive control of nonlinear systems in Chapter 4. In this chapter, the class of systems under consideration and its assumptions are first described. Conditions that TANN has to satisfy in order to guarantee stability of the closed-loop system are given. We also discuss a training method for the neural networks to satisfy these conditions.

For the parameter estimation problems in Chapter 3, the goal is to reduce the parameter estimation error. Whereas, for the stabilization problems, the goal is to decrease the output error. To achieve this, the idea of the recursive method for parameter estimation can be modified to construct controllers for systems with unknown controller structures. This is described in Chapter 5. In this chapter, Training procedure as well as the stability analysis of this approach are given.

The algorithms described in Chapters 3, 4 and 5 are verified by simulation in Chapter 6. The simulation examples include discrete-time and continuous-time systems in either state space or regression forms. Comparison of simulation results with other schemes is also presented in this chapter.

Finally, concluding remarks and recommended future work are given in Chapter 7.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter, materials that are useful in later chapters are discussed. Representations of multi-layered neural networks (MNN) and Gaussian networks and their property that is essential to development of the algorithms in this thesis are presented in section 2.2. Some known results in analysis and differential equations to be used later are delineated in Section 2.3. Training of neural networks is often a nonlinear programming problem. The quadratic penalty function method, which deals with constrained optimization, used throughout the thesis is summarized in Section 2.4.

2.2 Artificial Neural Networks

ANN are usually referred to computational units composed of simple and interconnected linear or nonlinear functions. They initially attracted attention of researchers due to resemblance in structure to their namesake in living beings. Later, ANN found applications in diverse disciplines such as pattern recognition, task classification, identification and control for various reasons. In this thesis, we mainly take advantage of their capability of function approximation, that is, efficiency in representing functions to desired degree of accuracy using finitely parametrized units. We focus on two classes of neural networks, MNN and Gaussian networks, though other neural net-

work architectures possessing the property called “universal approximator”, which is explained later in this section, can be applied equally well to the methods in this thesis. The analytical forms of these two networks and some of the related formula are presented below.

A typical n -layered MNN with input u and output y can be written in the following equations [25]:

$$\begin{aligned} z(i+1) &= W(i)x(i) + b(i) \\ x(i) &= \Gamma(z(i)), \quad i = 1, \dots, n \end{aligned}$$

where $W(i)$ and $b(i)$ denote the weight matrix and the bias vector of the i -th layer respectively, $x(1) = u$, $z(n+1) = y$ and $\Gamma(x) = [\gamma(x_1), \gamma(x_2), \dots, \gamma(x_m)]^T$ with γ usually a smooth nonlinear function such as

$$\gamma(x) = \frac{1 - e^{-x}}{1 + e^x}$$

To train a MNN, we often need to calculate the gradients of y with respect to $W(i)$ and $b(i)$. They are given as follows:

$$\begin{aligned} \delta^{(k)}(i) &\triangleq \left(\frac{\partial y_k}{\partial z(i)} \right)^T \\ &= \Gamma'(z(i))W^T(i)\delta^{(k)}(i+1) \\ \left(\frac{\partial y_k}{\partial w_j(i)} \right)^T &= x_j(i)\delta^{(k)}(i+1) \\ \left(\frac{\partial y_k}{\partial b(i)} \right)^T &= \delta^{(k)}(i+1), \quad i = 2, \dots, n \end{aligned}$$

where $\delta^{(k)}(n+1) = [0, \dots, 0, 1, 0, \dots, 0]^T$, y_k is the k -th element of y , $w_j(i)$ is the j -th column of $W(i)$ and $\Gamma'(x) = \text{diag}\{\gamma'(x_1), \dots, \gamma'(x_m)\}$. These gradient information is used in different schemes for updating weights. For example, the well-known back-propagation method [63] is simply correcting weights along the opposite directions of the gradients. Due to slow convergence of the gradient descent method, other

higher order methods such as the extended Kalman filter algorithm are applied to the training of neural networks as well [57].

On the other hand, for a Gaussian network with p centers and the input u , the corresponding output y can be represented in the following form [55]:

$$y = \sum_{j=1}^p w_j z_j$$

$$z_j = \exp\left(-\sum_{k=1}^I \frac{|u_k - c_{jk}|^2}{\alpha_k^2}\right)$$

where w_j is a column weight vector and has the same dimension as y , u_k denotes the k -th element of the column vector u , α_k is the variance associated with u_k , and I is the total number of the network inputs. The gradients of y with respect to w_j and α_k are:

$$\frac{\partial y}{\partial w_j} = z_j I$$

$$\frac{\partial y}{\partial \alpha} = W \begin{bmatrix} z_1 \frac{2(u_1 - c_{11})^2}{\alpha_1^2}, & \cdot & \cdot & \cdot, & z_1 \frac{2(u_I - c_{1I})^2}{\alpha_I^2} \\ \cdot & \cdot & & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & & \cdot & \cdot \\ z_p \frac{2(u_1 - c_{p1})^2}{\alpha_1^2}, & \cdot & \cdot & \cdot, & z_p \frac{2(u_I - c_{pI})^2}{\alpha_I^2} \end{bmatrix}$$

where $W = [w_1, \dots, w_p]$.

As is mentioned earlier, the role of neural networks in this thesis is to approximate nonlinear functions. Furthermore, they must be able to do so to desired degree of accuracy if enough units are used. This ability is often called the universal approximator. To quantify the requirement, the following definition describing such a class of neural networks \mathcal{N} is used. Let Ω be the set of all continuous mappings of \mathfrak{R}^n into \mathfrak{R}^m , and \mathcal{N} be a subset of Ω .

Definition 2.1 *Given $\epsilon > 0$ and a compact set $K \subset \mathfrak{R}^n$, for every $f \in \Omega$, there exists a $N \in \mathcal{N}$ such that*

$$|N(x) - f(x)| < \epsilon \quad \forall x \in K$$

Definition 2.1 implies that for any continuous function $f \in \Omega$, we can find a $N \in \mathcal{N}$ to approximate the function uniformly to the desired degree of accuracy in a compact set. MNN and the Gaussian network discussed previously have been shown by various authors [15, 21, 47] to qualify as such a class. Without loss of generality, we assume that all neural networks considered in this thesis have this property. Another reason why neural networks are used here is due to their efficiency in representing nonlinear functions. According to [6], if parameters of nonlinear combinations are adjusted, sigmoidal networks can achieve integrated square error of order $\frac{1}{N}$, where N is the number of basis functions. Hence, by varying the nonlinear coefficients in the network, more compact representations of nonlinear functions can be achieved compared to linear models such as power series. This is especially useful when the dimension of network inputs is high.

2.3 Mathematical Preliminaries

In this section, we first show some results in ordinary differential equations in order to establish conditions for transformation from continuous-time to discrete-time representations, followed by results in analysis and stability of dynamic systems.

The parameter estimation algorithms developed in this thesis as well as many other algorithms appearing in the literature focus on discrete-time representations. Since most physical systems are modeled in the continuous time, transformation between these two representations is often necessary. While the transformation is trivial in linear systems, its counterpart in nonlinear systems is much more involved, and an analytical form may not be easily obtained. Nevertheless, TANN algorithms only require either the continuous-time or the discrete-time model be known, provided that the discrete-time representation exists and is continuous. To establish existence and continuity of the transformation, the following two theorems from ordinary differential equations shall prove useful.

Consider the following initial-value problem:

$$\dot{\xi} = f(t, \xi(t)), \quad \xi(\sigma^0) = x^0 \quad (2.1)$$

where $f : \mathcal{I} \times \mathcal{X} \rightarrow \mathfrak{R}^n$, $\mathcal{X} \subset \mathfrak{R}^n$ is open and \mathcal{I} is an interval in \mathfrak{R} . In addition, f satisfies the following two properties:

(H1) $f(\cdot, x) : \mathcal{I} \rightarrow \mathfrak{R}^n$ is measurable for each $x \in \mathcal{X}$.

(H2) $f(t, \cdot) : \mathfrak{R}^n \rightarrow \mathfrak{R}^n$ is continuous for each $t \in \mathcal{I}$.

(H3) f is locally Lipschitz on x ; that is, there are for each $x^0 \in \mathcal{X}$ a real number $\rho > 0$ and a locally integrable function $\alpha : \mathcal{I} \rightarrow \mathfrak{R}_+$ such that the ball $B_\rho(x^0)$ of radius ρ centered at x^0 is contained in \mathcal{X} and

$$\|f(t, x) - f(t, y)\| \leq \alpha(t)\|x - y\|$$

for each $t \in \mathcal{I}$ and $x, y \in B_\rho(x^0)$.

(H4) f is locally integrable on t ; that is, for each fixed $x^0 \in \mathcal{X}$ there is a locally integrable function $\beta : \mathcal{I} \rightarrow \mathfrak{R}_+$ such that

$$\|f(t, x^0)\| \leq \beta(t)$$

for almost all t .

Existence and uniqueness of solutions of the initial-value problem in Eq. (2.1) can be stated as follows [60]:

Theorem 2.2 (Existence and Uniqueness Theorem) *Assume that $f : \mathcal{I} \times \mathcal{X} \rightarrow \mathfrak{R}^n$ satisfies the assumptions (H1), (H2), (H3) and (H4). Then, for each pair $(\sigma^0, x^0) \in \mathcal{I} \times \mathcal{X}$ there is some nonempty subinterval $J \subset \mathcal{I}$ open relative to \mathcal{I} and there exists a solution ξ of Eq. (2.1) on J , with the following property: If $\zeta : J' \rightarrow \mathcal{X}$ is any other*

solution of Eq. (2.1), where $J' \subset \mathcal{I}$, then necessarily

$$J' \subset J \quad \text{and} \quad \xi = \zeta$$

on J' .

The solution ξ is called the maximal solution of the initial-valued problem on the interval \mathcal{I} .

Under more assumptions on f , continuous dependence of the solution on initial conditions and on the right-hand side of Eq. (2.1) can be established. Consider the new initial-value problem as follows:

$$\dot{\zeta} = f(t, \zeta) + h(t, \zeta), \quad \zeta(\sigma^0) = z^0 \quad (2.2)$$

where f and h are mappings satisfying the hypotheses (H1), (H2), (H3) and (H4) and in addition

$$\left\| \int_{\sigma^0}^t h(s, \xi(s)) ds \right\| < \delta$$

for all $t \in [\sigma^0, \tau]$, and $z^0 \in \mathcal{X}$,

$$\|z^0 - x^0\| < \delta$$

Theorem 2.3 [60] *Let ξ be a solution of Eq. (2.1) on the interval $[\sigma^0, \tau] \subset \mathcal{I}$. Then, there exist numbers $c, \Delta > 0$ so that if $\delta \in (0, \Delta]$, the solution ζ of Eq. (2.2) is defined on the entire interval $[\sigma^0, \tau]$, and is uniformly close to that of Eq. (2.1), i.e.,*

$$\|\xi - \zeta\|_{\infty} < c\delta$$

Based on Theorems 2.2 and 2.3, the result on the transformation between continuous-time and discrete-time representations using piecewise constant control signals can be formulated below. Consider a continuous-time system as follows:

$$\dot{x} = f(x, r, \theta), \quad x(0) = x^0 \quad (2.3)$$

where $f : \mathcal{X} \times \mathcal{U} \times \Theta \rightarrow \mathbb{R}^n$, and $x^0 \in \mathcal{X}$, $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{U} \subset \mathbb{R}^s$ and $\Theta \subset \mathbb{R}^m$ are open.

Assume the system is under a discrete exogenous input r , where $r : \mathcal{I} \rightarrow \mathcal{U}$ is a piecewise constant function defined as

$$r(t) = c_i, \quad \text{if } t \in [iT, (i+1)T)$$

where $T > 0$ and $i = 0, \dots, l-1$.

Furthermore, the following conditions are satisfied:

(CD1) f is uniformly continuous on $\mathcal{X} \times \mathcal{U} \times \Theta$.

(CD2) there are for each $x^0 \in \mathcal{X}$ a real number $\rho > 0$ and a function $\alpha : \mathcal{U} \times \Theta \rightarrow \mathfrak{R}_+$ locally integrable for every $r \in \mathcal{U}$ and $\theta \in \Theta$ such that the ball $B_\rho(x^0)$ of radius ρ centered at x^0 is contained in \mathcal{X} and

$$\|f(x, r, \theta) - f(y, r, \theta)\| \leq \alpha(r, \theta)\|x - y\|$$

for every $r \in \mathcal{U}$, $\theta \in \Theta$, and $x, y \in B_\rho(x^0)$.

(CD3) there exists $l > 0$ such that for every $(t, x^0, \theta, r) \in [0, lT) \times \mathcal{X} \times \Theta \times \mathcal{U}$, solutions of (2.3) exist.

Theorem 2.4 *For the dynamic system in Eq. (2.3), there exists a continuous function $f_d : \mathcal{X} \times \mathcal{U} \times \Theta \rightarrow \mathfrak{R}^n$ such that for every $x(kT) \in \mathcal{X}$, $r(kT) \in \mathcal{U}$ and $\theta \in \Theta$,*

$$x((k+1)T) = f_d(x(kT), r(kT), \theta)$$

where $k < l-1$ is a non-negative integer.

Proof: Define $\hat{f} : \mathcal{I} \times \mathcal{X} \times \Theta \rightarrow \mathfrak{R}^n$ as $\hat{f}(t, x, \theta) = f(x, r(t), \theta)$. Consider the following initial-value problem:

$$\dot{x} = \hat{f}(t, x, \theta), \quad x(0) = x^0$$

Since r is a piecewise constant function of t and, from (CD1), $f(x, \cdot, \theta)$ is continuous for every $x \in \mathcal{X}$ and $\theta \in \Theta$, \hat{f} is measurable for every $x \in \mathcal{X}$ and $\theta \in \Theta$. Hence,

(H1) of Theorem 2.2 is satisfied. (H2) is obtained directly from (CD1). Since α in (CD2) is locally integrable and $r(t)$ is a piecewise constant function, the function $\beta : \mathcal{I} \times \Theta \rightarrow \mathbb{R}_+$ defined as $\beta(t, \theta) = \alpha(r(t), \theta)$ is thus locally integrable. Hence, from (CD2),

$$\|\widehat{f}(t, x, \theta) - \widehat{f}(t, y, \theta)\| \leq \beta(t, \theta)\|x - y\|$$

for every $t \in \mathcal{I}$, $\theta \in \Theta$ and $x, y \in B_\rho(x^0)$, which implies (H3). Moreover, since $f(x, r, \theta)$ is continuous and r is integrable on \mathcal{I} , $\widehat{f}(t, x, \theta)$ is integrable on \mathcal{I} and thus, (H4) is satisfied. Therefore, we can conclude from (CD3) and Theorem 2.2 that the solution of (2.3) is unique on $[0, lT)$. Thus, there exists a function $f_d : \mathcal{X} \times \mathcal{U} \times \Theta \rightarrow \mathbb{R}^n$:

$$x(T) = f_d(x(0), r, \theta)$$

Next, we want to prove that f_d is continuous on $\mathcal{X} \times \mathcal{U} \times \Theta$. Since $f(x, r, \theta)$ is uniformly continuous on $\mathcal{X} \times \mathcal{U} \times \Theta$, given $\epsilon > 0$ and $c > 0$, there exists a $\delta > 0$ such that for every $(x, r_0, \theta_0) \in \mathcal{X} \times \mathcal{U} \times \Theta$ if $\|((r - r_0), (\theta - \theta_0))\| < \delta$, then

$$\|f(x, r, \theta) - f(x, r_0, \theta_0)\|_\infty < \frac{\epsilon}{cT}$$

If, in addition, $x(0) \in \mathcal{X}$ is chosen so that $\|x(0) - x^0\| < \delta$, then, according to Theorem 2.3, the solution of the following two initial-value problems:

$$\begin{aligned} \dot{\xi} &= f(\xi, r_0, \theta_0), & \xi(0) &= x^0 \\ \dot{\zeta} &= f(\zeta, r, \theta), & \zeta(0) &= x \end{aligned}$$

are uniformly close:

$$\|\xi - \zeta\|_\infty < \epsilon$$

Therefore, if $\|((x - x^0), (r - r_0), (\theta - \theta_0))\| < \delta$, then $\|f_d(x, r, \theta) - f_d(x^0, r_0, \theta_0)\| < \epsilon$, implying that f_d is continuous on $\mathcal{X} \times \mathcal{U} \times \Theta$

Let $\xi_1(t)$ be the maximal solution of Eq. (2.3) on J_1 with $x(\sigma + T) = x^0$ and $r = r_1(t)$ where $(\sigma + T, x^0, \theta) \in J_1 \times \mathcal{X} \times \Theta$; let $\xi_2(t)$ be the maximum solution of Eq. (2.3) on J_2 with $x(\sigma) = x^0$ and $r = r_1(t + T)$ where $\sigma \in J_2$. We want to prove

that $\xi_1(t+T) = \xi_2(t)$, for every $t \in J_1 \cap J_2$, and thus, $x(kT+T) = f_d(x(kT), r_1, \theta)$ for every $(x(kT), r_1, \theta) \in \mathcal{X} \times \mathcal{U} \times \Theta$ and $k < l-1$. Since $\xi_1(t)$ and $\xi_2(t)$ are solutions of Eq. (2.3) with $x(\sigma+T) = x^0$ and $x(\sigma) = x^0$ respectively, they satisfy the following equations:

$$\begin{aligned}\xi_1(t) &= x^0 + \int_{\sigma+T}^t f(\xi_1(\tau), r(\tau))d\tau, & \forall t \in J_1 \\ \xi_2(t) &= x^0 + \int_{\sigma}^t f(\xi_2(\alpha), r(\alpha+T))d\alpha, & \forall t \in J_2\end{aligned}$$

By comparing the above two equations, we can conclude that $\xi_2(t-T) = \xi_1(t)$. ■

In one of the TANN for parameter estimation, the neural network is trained to approximate the implicit function between measurable signals and parameters. The following result from analysis states the conditions under which the implicit function exists and its corresponding property.

Theorem 2.5 (Implicit Function Theorem) [40] *Let A be open in \mathbb{R}^{k+n} ; let $f : A \rightarrow \mathbb{R}^n$ be of class C^r . Write f in the form $f(x, y)$, for $x \in \mathbb{R}^k$ and $y \in \mathbb{R}^n$. Suppose that (a, b) is a point of A such that $f(a, b) = 0$ and*

$$\det \frac{\partial f}{\partial y}(a, b) \neq 0$$

Then, there is a neighborhood B of a in \mathbb{R}^k and a unique continuous function $g : B \rightarrow \mathbb{R}^n$ such that $g(a) = b$ and

$$f(x, g(x)) = 0$$

for all $x \in B$. The function g is in fact of class C^r .

The following two theorems states stability results for dynamic systems. Theorem 2.6 concerns linear systems, while Theorem 2.7 can be applied to general nonlinear systems.

Theorem 2.6 [42] *The equilibrium state $x = 0$ of the linear time-invariant system*

$$\dot{x} = Ax$$

is asymptotically stable if, and only if, given any symmetric positive-definite matrix Q , there exists a symmetric positive-definite matrix P , which is the unique solution of the set of $\frac{n(n+1)}{2}$ linear equations

$$A^T P + P A = -Q \quad (2.4)$$

Therefore, $V(x) = x^T P x$ is a Lyapunov function for the system.

A discrete version of the above theorem also exists [18], with Eq. (2.4) being replaced by

$$A^T P A - P = -Q \quad (2.5)$$

For stability of general nonlinear systems, consider the following dynamic system:

$$\dot{x} = X(x) \quad (2.6)$$

where $x \in \mathcal{O} \subset \mathfrak{R}^n$ and $X : \mathcal{O} \rightarrow \mathfrak{R}^n$. The positive definite function $V : \mathcal{O} \rightarrow \mathfrak{R}$ has continuous first partials on \mathcal{O} .

Theorem 2.7 [31] *Let Ω denote the set defined by $V(x) \leq \alpha$, and let Ω^c denote the complement of Ω . If*

- (i) $\dot{V}(x) \leq 0$ for all x in Ω^c ,
- (ii) \dot{V} does not vanish identically along any trajectory that starts in Ω^c ,
- (iii) The system in Eq. (2.6) is Lagrange stable,

then every solution of Eq. (2.6) approaches Ω as $t \rightarrow \infty$.

2.4 The Quadratic Optimization Method

The training problems of neural networks for parameter estimation and control in this thesis can usually be formulated as constrained optimization. For the sake of

brevity in the later discussion, we review in this section results from nonlinear programming which are pertinent to these problems. Part of the material in this section is summarized from [8].

Consider the following constrained optimization problem:

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && h_1(x) = 0, \dots, h_m(x) = 0, \quad g_1(x) \leq 0, \dots, g_r(x) \leq 0 \end{aligned} \quad (2.7)$$

The above inequality constrained problem can be converted to a equality constrained one by adding r variables z_i :

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && h_1(x) = 0, \dots, h_m(x) = 0, \quad g_1(x) + z_1^2 = 0, \dots, g_r(x) + z_r^2 = 0 \end{aligned}$$

If the quadratic penalty function method is used, this problem becomes

$$\begin{aligned} & \min_{x,z} \bar{L}_c(x, z, \lambda, \mu) = \\ & \min_{x,z} \left(f(x) + \lambda' h(x) + \frac{c}{2} \|h(x)\|^2 + \sum_{j=1}^r \left\{ \mu_j (g_j(x) + z_j^2) + \frac{c}{2} |g_j(x) + z_j^2|^2 \right\} \right) \end{aligned} \quad (2.8)$$

where $\lambda = [\lambda_1, \dots, \lambda_m]^T$ and $\mu = [\mu_1, \dots, \mu_r]^T$ are the Lagrange multipliers and c is a positive constant sometimes referred to as the penalty parameter. It can be observed from (2.8) that for large c , L_c can be large if the constraints are not satisfied. Hence, as c increases, the solution tends to be near the constraints. As a matter of fact, if λ is equal to the optimal value of Lagrange multiplier, there is a finite \bar{c} such that when $c > \bar{c}$, the W that minimizes the augmented Lagrangian \bar{L}_c is also the solution of the original optimization problem in (2.7) [8]. The above minimization can be performed by first minimizing \bar{L}_c with respect to z , which can be carried out in closed form for each x . This yields

$$\begin{aligned} & \min_x L_c(x, \lambda, \mu) = \\ & \min_x \left[f(x) + \lambda' h(x) + \frac{c}{2} \|h(x)\|^2 + \frac{1}{2c} \sum_{j=1}^r \left\{ (\max\{0, \mu_j + cg_j(x)\})^2 - \mu_j^2 \right\} \right] \end{aligned} \quad (2.9)$$

To improve convergence rate and avoid numerical ill-conditioning, the method of mul-

multipliers is commonly used to update λ and μ in (2.9) without requiring c approaching infinity. The algorithm is summarized below:

$$\begin{aligned}\lambda^{(k+1)} &= \lambda^{(k)} + c^{(k)}h(x^{(k)}) \\ \mu_j^{(k+1)} &= \max\{0, \mu_j^{(k)} + c^{(k)}g_j(x^{(k)})\}\end{aligned}\tag{2.10}$$

where $x^{(k)}$ minimizes

$$\begin{aligned}L_{c^{(k)}}(x, \lambda^{(k)}, \mu^{(k)}) &= f(x) + \lambda^{(k)'}h(x) + \frac{c^{(k)}}{2}\|h(x)\|^2 \\ &+ \frac{1}{2c^{(k)}}\sum_{j=1}^r\left\{\left(\max\{0, \mu_j^{(k)} + c^{(k)}g_j(x)\}\right)^2 - (\mu_j^{(k)})^2\right\}\end{aligned}\tag{2.11}$$

and the limit point of the sequence of $\{x^{(k)}\}$ is the global minimum of the problem in (2.7).

In the derivation of the training algorithms in the following chapters, the resulting optimization problem is often a special case of that in (2.7). That is, the problem involves only inequality constraints and no cost function or equality constraints:

$$g_1(x) \leq 0, \dots, g_r(x) \leq 0\tag{2.12}$$

For this special case, it can be observed from (2.9) that for a particular μ , as long as every constraint $g_j \leq 0$ is met, the minimal cost $-\sum_{i=1}^r \frac{\mu_i^2}{2c}$ can always be achieved with a large enough c . Hence, (2.9) can be further reduced to the following problem:

$$\min_x J \triangleq \min_x \frac{1}{2} \sum_{i=1}^r (\max\{0, g_i(x)\})^2\tag{2.13}$$

Obtaining x^* that solves the above problem requires less computation than that in (2.9) since only one optimization step is necessary in (2.13) as opposed to solving a series of optimization problems in (2.10) and (2.11).

The unconstrained optimization problems in (2.11) and (2.13) can be solved in many ways. In this thesis, the Levenberg-Marquardt [38] method is often used. The

algorithm is summarized below [52]. Consider minimizing the following cost function:

$$\chi^2 = \sum_{i=1}^N \sum_{j=1}^n \left[\frac{y_{ij} - y_j(u_i; x)}{\sigma_i} \right]^2$$

where N is the total number of data elements, n is the dimension of $y(u_i; x)$, y_{ij} is the j -th coordinate of the i -th data element, and $y_j(u_i; x)$ denotes the j -th element of the model output with the input u_i and the model parameter x . The goal is to adjust x so that χ^2 is minimized. For the Levenberg-Marquardt method, the adjustment $\delta x \equiv x^{(new)} - x^{(current)}$ is obtained by solving the M simultaneous linear equations:

$$\sum_{l=1}^M \alpha'_{kl} \delta x = \beta_k, \quad k = 1, \dots, M$$

where M denotes the total number of elements in x , $\alpha'_{jj} = \alpha_{jj}(1 + \lambda)$, $\alpha'_{kl} = \alpha_{kl}$ for $k \neq l$,

$$\begin{aligned} \alpha_{kl} &= \frac{1}{2} \frac{\partial^2 \chi^2}{\partial x_k \partial x_l} \\ \beta_k &= -\frac{1}{2} \frac{\partial \chi^2}{\partial x_k} \end{aligned}$$

and x_k is the k -th element of x . When $x^{(new)}$ results in smaller χ^2 , λ is decreased, which moves the algorithm towards the Newton method. Otherwise, if the new x yields larger χ^2 , the original x is kept and λ is increased so that the next δx is closer to the direction of the gradient. The procedure then repeats as described above. In practice, α_{kl} and β_k are evaluated as:

$$\begin{aligned} \beta_k &= \sum_{i=1}^N \sum_{j=1}^n \frac{[y_{ij} - y_j(u_i; x)]}{\sigma_i^2} \frac{\partial y_j(u_i; x)}{\partial x_k} \\ \alpha_{kl} &\approx \sum_{i=1}^N \sum_{j=1}^n \frac{1}{\sigma_i^2} \left[\frac{\partial y_j(u_i; x)}{\partial x_k} \frac{\partial y_j(u_i; x)}{\partial x_l} \right] \end{aligned}$$

Chapter 3

Parameter Estimation Using θ -Adaptive Neural Networks

3.1 Introduction

The term parameter is typically used to denote an element that characterizes the dynamic characteristics of a system. Determining the parameter value is therefore very useful in several contexts. In the case of linear systems, the parameter value is sufficient for representation, prediction, and control in the entire state-space. For nonlinear systems, the specific value of a parameter characterizes an operating point as well as the corresponding linearized map about the operating point. For instance, in the context of the heat exchange dynamics in air-conditioners, the heat transfer coefficient between the tube wall and the air varies due to different operating conditions [19]; in order to determine a suitable controller at a particular operating condition, the coefficient needs to be estimated on-line. In several process control problems, one might be interested in determining actual physical parameters in a system instead of predicting its response. For example, the determination of the control parameters such as the PID gains might be the task assigned to a skilled controller, since that suffices to generate the requisite performance over a wide range of operation. For all these problems, it would be more appropriate and useful to employ the neural network to identify the parameters of a dynamic system rather than simply match

the output of a network with that of the plant.

A significant portion of parameter estimation methods concerns linear systems and only a limited class of nonlinear systems can be solved analytically using available tools. Whether linear or nonlinear, in most of these problems, the identification approach requires the parameters to occur linearly. In general, a nonlinear parametrization is inevitable due to a number of reasons. For instance, one often needs to transform the state space representation to the regression form

$$y_t = f_r(y_{t-1}, \dots, y_{t-n}, u_{t-1}, \dots, u_{t-n}, \theta) \quad (3.1)$$

for the sake of convenience or because all the states may not be accessible. In such a case, even if f is linear in the parameter θ in the state space form, f_r can be nonlinear in θ due to nonlinearity of the states. Also, it may not always be possible to determine the analytical form of f_r based on the nonlinear state space representation. In all these cases, parameter estimation methods, which can generate an estimate of θ using either the regression form in Eq. (3.1) or the state space form, and does not require f_r to be linear in θ , is highly attractive. A parameter estimate thus obtained can be used to carry out tasks such as fault detection, prediction, or control in a nonlinear system.

In this chapter, neural network based algorithms are developed to perform parameter estimation. Most of the instances where a neural network has been used to represent dynamic systems have been in the context of identifying input-output representations (e.g. [58, 43]). The focus in this chapter, however, is on a different relationship that illustrates the system characteristic. This corresponds to the relationship between the system variables and the parameter θ . If ω is a vector of system variables that can be measured, then we can express this relationship as

$$\theta = h(\omega)$$

It is proposed in this chapter to use a neural network to identify the nonlinear map h , thereby generating a parameter estimate of θ . Two approaches are given. In the first

approach, defined as the block estimation method, the neural network attempts to approximate the implicit function between the response and the parameters directly. In the second approach, denoted as the recursive estimation method, the parameter estimates are updated by incorporating new samples of system response. For both approaches, the inputs of the network are the observable system variables. The outputs are the parameter estimates for the first approach, and the update of the parameter estimates for the second approach. The final algorithms obtained represent the map between the system variables and the parameter estimate. We denote this model as θ -adaptive neural networks (TANN).

This chapter is organized as follows. The problem is described in Section 3.2. The block estimation method is discussed in Section 3.3, and the recursive method in Section 3.4. The structure and the on-line implementation procedure for the block and recursive methods are explained in Section 3.3.1 and Section 3.4.1 respectively, followed by the training procedure of the neural network in Section 3.3.2 and Section 3.4.2 respectively. The recursive algorithm is compared to the extended Kalman filter in Section 3.5 to illustrate the difference between TANN and those schemes based on linearization. Finally, concluding remarks are given in Section 3.6.

3.2 Problem Statement

Our focus is on a class of nonlinear dynamic systems which can be represented in the form of

$$\begin{aligned} \dot{x} &= f_c(x, u, \theta) \\ y &= g_c(x, u, \theta) \end{aligned} \tag{3.2}$$

in continuous-time or

$$\begin{aligned} x_t &= f_d(x_{t-1}, u_{t-1}, \theta) \\ y_t &= g_d(x_{t-1}, u_{t-1}, \theta) \end{aligned} \tag{3.3}$$

in discrete-time. Eqs. (3.2) and (3.3) are typically obtained from physical models of the system under consideration, and the parameter θ corresponds to the physical constants that characterizes the system dynamics. The regression form of representation

of the system in Eq. (3.2) (or (3.3)) is

$$y_t = f_r(\phi_{t-1}, \theta) \quad (3.4)$$

where ϕ_{t-1} represents measurable signals upto time $t - 1$ including past inputs and outputs. It is also assumed that $\phi_{t-1} \in \Phi^c \subset \mathbb{R}^m$, $\theta \in \Theta^c \subset \mathbb{R}^n$ and $y_t \in \Psi^c \subset \mathbb{R}$ for every $t \geq 1$, where Φ^c, Θ^c and Ψ^c are known compact sets and $f_r : \Phi^c \times \Theta^c \rightarrow \Psi^c$ is twice continuously differentiable on $\Phi^c \times \Theta^c$. In this paper, we restrict our attention to the nonlinear system in the form of Eq. (3.4). That this is not a significant restriction can be justified as follows. By using results in Theorem 2.4, it follows that there is a unique transformation between Eq. (3.2) and Eq. (3.3). Furthermore, according to [33], under the condition that the Hankel matrix of the linearized f_d around some equilibrium point has the rank equal to the dimension of the states, Eq. (3.3) can be transformed into the form of Eq. (3.4) with $\phi_{t-1} = [y_{t-1}, \dots, y_{t-k}, u_{t-1}, \dots, u_{t-k}]^T$ where k denotes the number of states in Eq. (3.3). Thus, under these conditions, Eqs. (3.2) and (3.3) can be transformed into the form of Eq. (3.4). Even though the transformation from one form to another may be very complex and may not even be easily obtained analytically in some cases, the training of the neural networks in our algorithms does not require explicit knowledge of the transformation map as we shall see later. Therefore, the development based on the regression form f_r does not limit application of our algorithms.

There are two important aspects regarding the training stage for the identification procedure we shall develop in this chapter. First, in order to train the neural network to learn the mapping between the system observations and θ , the vector $[\phi_{t-1}, y_t]$ as well as the corresponding true parameter θ must be known or calculable. Since the training process typically occurs off-line and in a controlled environment, it is reasonable to assume that these quantities are available for measurement or known. For instance, using the model in Eq. (3.2) or (3.3), at specific known operating points of the system, if f_c or f_d is completely known, and $x(t)$ can be measured corresponding to the known value of the parameter θ , the set $\{x(t), \theta\}$ suffices to accomplish the

training. On the other hand, when only a subset $\phi(t-1)$ can be measured, a known f_r , leading to the measurement of the set $\{y(t), \phi(t-1), \theta\}$ can be used to train the neural network. In the absence of accurate knowledge regarding f_c , f_d or f_r , an extensive simulation model depicting the system interaction for a specific parameter needs to be available to select the data set $\{y(t), \phi(t-1), \theta\}$. An experimental setup which is a prototype model of the process or a pilot experiment leading to the same measurable data set would suffice as well to accomplish the training. [50] describes that one such experiment is carried out to collect data for the purpose of designing a gain scheduling controller in aircrafts. Second, during the training process data collection has to be accomplished by varying the operating point and the parameter over all typical values that would be encountered during the systems' on-line operation. Therefore, the model, whether it is obtained from analysis, simulation, or experiments, must be amenable to the parameter as well as the operating point being varied. Such an off-line training procedure is a necessary ingredient of the approach.

3.3 The Block Parameter Estimation Method

3.3.1 The Block Estimation Algorithm

Consider the nonlinear system in Eq. (3.4). By collecting data for n continuous time steps, we can write Eq. (3.4) as

$$\left\{ \begin{array}{l} y_{t-n+1} = f_r(\phi_{t-n}, \theta) \\ \quad \quad \quad \vdots \\ \quad \quad \quad \quad \quad \quad \vdots \\ y_{t-1} = f_r(\phi_{t-2}, \theta) \\ y_t = f_r(\phi_{t-1}, \theta) \end{array} \right. \quad (3.5)$$

Eq. (3.5) can be written in a compact form as

$$F(\Phi_{t-1}, \theta) = 0 \quad (3.6)$$

where $F(\Phi_{t-1}, \theta) = [f_r(\phi_{t-1}, \theta) - y_t, \dots, f_r(\phi_{t-n}, \theta) - y_{t-n+1}]^T$ and Φ_{t-1} denotes the vector of independent variables of $\phi_{t-1}, \phi_{t-1}, \dots, \phi_{t-n}$ and $y_t, y_{t-1}, \dots, y_{t-n+1}$. Since F is continuous, according to the implicit function theorem in Theorem 2.5, if $\det(D_\theta F(\Phi_0, \theta_0)) \neq 0$ and $F(\Phi_0, \theta_0) = 0$, then there exists a neighborhood Φ° of Φ_0 and a unique continuous map $G : \Phi^\circ \rightarrow \Theta^c$ such that for all $\Phi \in \Phi^\circ$,

$$\theta = G(\Phi_{t-1}) \quad (3.7)$$

In other words, if the implicit function G can be determined, a θ which satisfies Eq. (3.6) can be uniquely determined using the measurement Φ . However, for most systems, either $F(\cdot)$ is unknown or G can not be found analytically. Nevertheless, using Φ_{t-1} as a input, the implicit function G can still be reconstructed using the training method described in Section 3.3.2. The relation between the parameter estimate so derived and the input Φ is given by:

$$\hat{\theta}_t = N(\Phi_{t-1}) \quad (3.8)$$

where N represents a neural network mapping and $\hat{\theta}_t$ is the corresponding parameter estimate. The goal is to train the neural network so that given Φ_{t-1} , its output converges to the desired parameter θ . In Theorem 3.1, we state that such a neural network N exists, which ensures that $\hat{\theta}$ converges to θ under certain conditions on Φ_t .

Consider the equation $F(\Phi, \theta) = 0$. The continuity of F follows from the assumption that f_r is continuous on $\Phi^c \times \Theta^c$.

Theorem 3.1 *Let $D_\theta F(\Phi_0, \theta_0)$ be nonsingular at interior points Φ_0 of Φ^c and θ_0 of Θ^c . Given the class of neural network \mathcal{N} which maps Φ^c into \mathbb{R}^n , there exist $\epsilon_1 > 0$ and a neural network $N \in \mathcal{N}$ such that, given $\epsilon > 0$,*

$$\sup_{\Phi \in B_1} |\theta - N(\Phi)| < \epsilon$$

where $B_1 = \{\Phi \mid |\Phi - \Phi_0| \leq \epsilon_1\}$

Proof: Since F is continuous and $D_\theta F(\Phi_0, \theta_0)$ is nonsingular, according to the implicit function theorem, there exist $\rho_1 > 0$ and a unique continuous map $\Theta(\Phi)$ such that for every $\Phi \in B_2$, $F(\Phi, \Theta(\Phi)) = 0$, where $B_2 = \{\Phi \mid |\Phi - \Phi_0| < \rho_1\}$. If we choose $\epsilon_1 = \rho_1/2$, then $B_1 \subset B_2$. Hence, for every $\Phi \in B_1$, $\Theta(\Phi)$ is continuous. Since B_1 is compact and Θ is continuous on the compact set B_1 , according to Definition 2.1, there exists a neural network $N \in \mathcal{N}$ such that given $\epsilon > 0$

$$|\Theta - N(\Phi)| < \epsilon \quad \forall \Phi \in B_1$$

■

The proof of the theorem simply utilizes the implicit function theorem and the universal approximator property of neural networks as stated in Definition 2.1.

Theorem 3.1 states that, with an appropriate choice of a neural network, the condition under which the block method will guarantee parameter identification is existence of the implicit function. This is analogous to the persistent excitation condition (e.g. [18]) in linear parameter estimation problems.

The convergence properties of the estimation algorithm obviously depends on the amount of observation noise present and how well the neural network is trained. In practice, the measurement Φ_{t-1} is almost always corrupted by noise. This may necessitate post-processing of the output of the neural network so that the effect of the noise is averaged out in some manner. One way of accomplishing this is by using a K-means like algorithm [20]. That is, instead of using Eq. (3.8) directly, we generate $\hat{\theta}_t$ as as

$$\begin{aligned} \bar{\theta}_t &= N(\Phi_{t-1}) \\ \hat{\theta}_t &= \hat{\theta}_{t-1} + \rho e^{-\frac{1}{\sigma^2}|\bar{\theta}_t - \hat{\theta}_{t-1}|^2} (\bar{\theta}_t - \hat{\theta}_{t-1}) \end{aligned} \tag{3.9}$$

where $0 < \rho < 1$ is the step size, σ^2 is the variance, and $\hat{\theta}_t$ is the resulting parameter estimate. The choice of ρ and σ^2 depends on the tradeoff between fast convergence (by choosing ρ close to 1 and small $\frac{1}{\sigma^2}$) and small variance (by choosing ρ close to 0 and large $\frac{1}{\sigma^2}$). The reason of adding the exponential term in Eq. (3.9) is that when the new estimate is too far away from the previous one, the correction is weighted less,

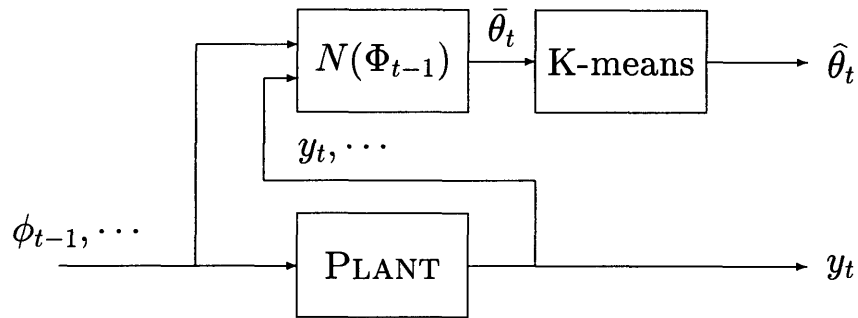


Figure 3-1: On-line Estimation using the Block Estimation Method

which usually occurs when the singularity points are encountered. In this case, larger $\frac{1}{\sigma^2}$ should therefore be used. σ is thus the parameter that controls the weighting. After the neural network is successfully trained, the block estimation algorithm is implemented as shown in Figure 3-1.

The nonsingularity of $D_{\theta}F(\Phi_0, \theta_0)$ cannot be checked on-line since it depends on θ for general nonlinear systems. The parameter estimate resulting from a data point where the implicit function does not exist could be less accurate. To increase the accuracy when the data is not rich enough, we can select a wider window over which the data is collected to determine the parameters, i.e. choose a larger n in Eq. (3.6). With more data available, we can reduce the possibility of encountering singular points, which is analogous to using pseudo-inverse to solve noisy linear equations. However, there is a direct tradeoff between accuracy and network complexity. As n gets larger, the network becomes larger in size and hence takes more time to train. Another approach to avoid the problem of singularity is to improve the estimate gradually only when new information is available and stop adaptation otherwise. This leads to the recursive estimation scheme discussed in Section 3.4.

3.3.2 Training Scheme

As shown in Eqs. (3.7) and (3.8), the neural network is used to model the mapping $\Phi_{t-1} \rightarrow \theta$ in the hyperspace (Φ_{t-1}, θ) . Since it is known that θ belongs to a compact

region Θ^c , we set $\theta = \theta_1 \in \Theta^c$. The corresponding Φ in Eq. (3.6) can be measured, and we denote this values as Φ_1 . A data element can be formed as (Φ_1, θ_1) , where these values are related as $F(\Phi_1, \theta_1) = 0$. By continuing the above procedure for $\Phi = \Phi_i$ where $i = 1, \dots, q$, a typical set of data T_1 can then be formed as

$$T_1 = \{(\Phi_i, \theta_1) | 1 \leq i \leq q\}$$

The process can be repeated for other $\theta = \theta_j \in \Theta^c$ where $j = 1, \dots, p$ to form the corresponding data sets T_j respectively. The complete set of training data is defined as:

$$T_{train} = \bigcup_{1 \leq j \leq p} T_j$$

Similarly, we can also form a testing set T_{test} for cross-validation during training.

Once the training set is obtained, we can train the weights of a neural network with the input Φ_i and the corresponding target θ_j . If a multi-layered neural network is used, for example, the training can be done by the back-propagation algorithm. If a radial basis function is used, the recursive least squares method can train the network efficiently. After the training procedure is completed, the network can be implemented as shown in Figure 3-1 using the algorithm in Eq. (3.9).

3.4 The Recursive Parameter Estimation Method

In contrast to the block method in Section 3.3, we introduce, in this section, an alternative procedure wherein the neural network produces the parameter update at every iteration rather than the actual parameter estimate itself. The algorithm, its training, and the stability analysis are presented in Sections 3.4.1 through 3.4.3.

3.4.1 The Recursive Estimation Algorithm

As is mentioned earlier, the goal is to construct a recursive scheme to identify θ in Eq. (3.4), which should depend only on signals that can be directly measured and contain information regarding θ . Since the value of θ is reflected at y_t through the

relation in Eq. (3.4), to determine the direction of improving the estimate $\hat{\theta}_{t-1}$ of θ at time $t - 1$, one way is to compare y_t with the predicted response \hat{y}_t , where $\hat{y}_t = f_r(\phi_{t-1}, \hat{\theta}_{t-1})$. Since \hat{y}_t depends on ϕ_{t-1} and $\hat{\theta}_{t-1}$, we choose the estimation algorithm as a function of y_t, ϕ_{t-1} and $\hat{\theta}_{t-1}$. Therefore, the algorithm is written in the following form:

$$\Delta \hat{\theta}_t \triangleq \hat{\theta}_t - \hat{\theta}_{t-1} = R(y_t, \phi_{t-1}, \hat{\theta}_{t-1})$$

where the function R is to be chosen such that $\hat{\theta}_t$ converges to θ asymptotically. In the case of a linear system $y_t = \theta^T \phi_{t-1}$, for example, a well-known linear parameter estimation method is to adjust $\Delta \hat{\theta}$ as (e.g. [18])

$$\Delta \hat{\theta}_t = \frac{a\phi_{t-1}}{c + \phi_{t-1}^T \phi_{t-1}} [y_t - \phi_{t-1}^T \hat{\theta}_{t-1}] \quad (3.10)$$

In other words, the mechanism for carrying out parameter estimation is realized by R . In the case of general nonlinear systems, the task of determining such a function R is quite difficult, especially when the parameters occur nonlinearly. The goal here is to find R corresponding to a general f_r in Eq. (3.4). Towards this end, we propose the use of a neural network N . That is, $\hat{\theta}_t$ is adjusted as

$$\Delta \hat{\theta}_t = N(y_t, \phi_{t-1}, \hat{\theta}_{t-1}) \quad (3.11)$$

where the inputs of the neural network are y_t, ϕ_{t-1} and $\hat{\theta}_{t-1}$ and the output is $\Delta \hat{\theta}_t$. The neural network is to be trained so that the resulting network can improve the parameter estimation over time for any possible $\theta \in \Theta^c$.

In order for the recursive scheme in Eq. (3.11) to result in a successful parameter estimation, it is sufficient to establish that the parameter error, $\tilde{\theta}_t$, or a continuous function thereof, decreases monotonically, where $\tilde{\theta}_t = \hat{\theta}_t - \theta$. If we define a positive definite function V_t as

$$V_t = \tilde{\theta}_t^T \tilde{\theta}_t \quad (3.12)$$

and the neural network in Eq. (3.11) is trained such that $V_t < V_{t-1}$ for all possible θ , once the algorithm is implemented on line, the estimated parameter $\hat{\theta}_t$ would

eventually approach the true parameter θ no matter what θ is. Based on the above idea, the change ΔV_t is first calculated to see how the neural network affects the estimation error. This is given by

$$\Delta V_t = V_t - V_{t-1} = 2\Delta\hat{\theta}_t^T \tilde{\theta}_{t-1} + \Delta\hat{\theta}_t^T \Delta\hat{\theta}_t$$

The choice of $\Delta\hat{\theta}_t$ as in Eq. (3.11) implies that

$$\Delta V_t = 2N^T \tilde{\theta}_{t-1} + N^T N.$$

The training procedure for adjusting weights in the network should therefore be such that $\Delta V_t < 0$ if $\tilde{\theta}_{t-1} \neq 0$. Since the output error $\tilde{y}_t = \hat{y}_t - y_t$ is indicative of how large the parameter error is, we define as a target of ΔV_t ,

$$\Delta V_{d_t} = -a \frac{2 + |C(y_t, \phi_{t-1}, \hat{\theta}_{t-1})|^2}{(1 + |C(y_t, \phi_{t-1}, \hat{\theta}_{t-1})|^2)^2} \tilde{y}_t^2 \quad (3.13)$$

where $0 < a < 1$, C is a continuous function of y_t , ϕ_{t-1} and $\hat{\theta}_{t-1}$ in \mathbb{R}^n , and $\Delta V_e = \Delta V - \Delta V_d$. The choice of C is discussed in Section 3.4.3. If N can be found such that $\Delta V_e \leq 0$ for all possible system responses and parameters in a compact region, the algorithm in Eq. (3.11) would gradually improve parameter estimation for any θ in that region, since this implies that $\Delta V \leq \Delta V_d \leq 0$. However, as ΔV_d approaches zero, when ΔV_e is small, ΔV may become positive due to the approximating nature of neural networks. This in turn would cause the parameter error to increase. To avoid this situation, the recursive estimation algorithm is modified as

$$\begin{aligned} \hat{\theta}_t &= \hat{\theta}_{t-1} + \Delta\bar{\theta}_t \\ \Delta\bar{\theta}_t &= \begin{cases} N(y_t, \phi_{t-1}, \hat{\theta}_{t-1}) & \text{if } |\tilde{y}_t| > \delta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (3.14)$$

similar to the dead-zone modification in adaptive controllers [42]. In general, the choice of δ depends on how well the neural network approximates and the magnitude

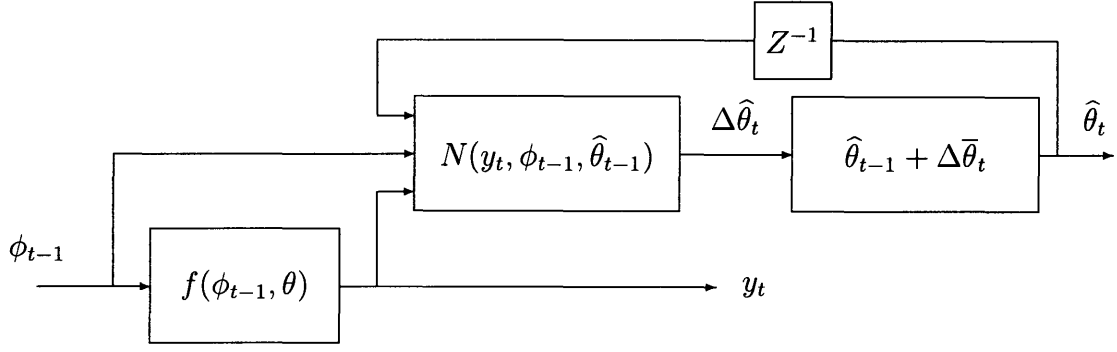


Figure 3-2: On-line Estimation Using the Recursive Estimation Method

of measurement noise. If the neural network can guarantee stability ($\Delta V < 0$) for small \tilde{y} and the noise is weak, a smaller δ can be selected, which results in more accurate estimation. The algorithm is implemented in the following manner (Figure 3-2): ϕ_{t-1} , y_t are measured first, and along with the past estimate $\hat{\theta}_{t-1}$ constitute the network inputs. Using this information, $\Delta\hat{\theta}_t$ is obtained as the output of the network. If $|\tilde{y}_t| > \delta$, $\hat{\theta}$ is updated as $\hat{\theta}_t = \hat{\theta}_{t-1} + \Delta\hat{\theta}_t$. Otherwise, the estimate remains unchanged.

3.4.2 Training Scheme

To establish $\Delta V_t \leq \Delta V_{d_t}$ for every $\theta, \hat{\theta} \in \Theta^c$ and $\phi \in \Phi^c$, it is necessary that for every samples $\theta_s, \hat{\theta}_s \in \Theta^c$ and $\phi_s \in \Phi^c$, $\Delta V_s \leq \Delta V_{d_s}$, where

$$\Delta V_s = 2N^T(y_s, \phi_s, \hat{\theta}_s)\tilde{\theta}_s + N^T N(y_s, \phi_s, \hat{\theta}_s) \quad (3.15)$$

$$\Delta V_{d_s} = -a \frac{2 + |C(y_s, \phi_s, \hat{\theta}_s)|^2}{(1 + |C(y_s, \phi_s, \hat{\theta}_s)|^2)^2} (y_s - \hat{y}_s)^2 \quad (3.16)$$

$y_s = f(\phi_s, \theta_s)$ and $\hat{y}_s = f(\phi_s, \hat{\theta}_s)$. In other words, $\Delta V \leq \Delta V_d$ at the sample points. Thus, we need to find N such that for every $\theta_s, \hat{\theta}_s \in \Theta^c$ and $\phi_s \in \Phi^c$,

$$\Delta V_{e_s} = \Delta V_s - \Delta V_{d_s} \leq 0$$

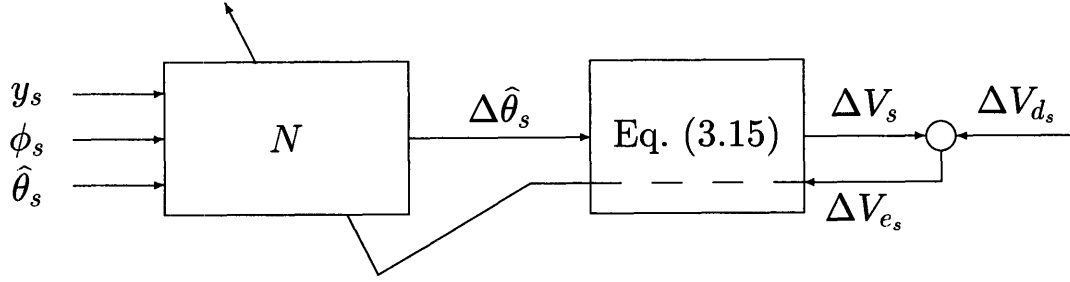


Figure 3-3: Training of the Recursive Estimation Network

Finding weights of a neural network to satisfy $\Delta V_{e_s} < 0$ can be considered as an optimization problem with no cost function and subject to the inequality constraints

$$\Delta V_{e_i}(W) \leq 0, \quad i = 1, \dots, M \quad (3.17)$$

on the weights space, where M is the total number of patterns in the training set. As discussed in Section 2.4, this can be accomplished by solving the following unconstrained optimization problem:

$$\min_W J \triangleq \min_W \frac{1}{2} \sum_{i=1}^M (\max\{0, \Delta V_{e_i}(W)\})^2 \quad (3.18)$$

To find a W which minimizes the above cost function J , we can apply algorithms such as the gradient method and the Gauss-Newton method. The gradient of the cost function with respect to a weight (or bias) w_j can be obtained as

$$\frac{\partial J}{\partial w_j} = \sum_{s \in P} 2\Delta V_{e_s} [\tilde{\theta}_s + N(y_s, \phi_s, \hat{\theta}_s)]^T \frac{\partial N}{\partial w_j} \quad (3.19)$$

where P denotes the set of patterns where $\Delta V_{e_s} > 0$. The term $\frac{\partial N}{\partial w_j} = [\frac{\partial N_1}{\partial w_j}, \dots, \frac{\partial N_n}{\partial w_j}]^T$ is the gradient of the output vector of the neural network with respect to a weight w_j in the network. The explicit forms of $\frac{\partial N_n}{\partial w_j}$ for the MNN and Gaussian networks are given in Section 2.2. If the gradient descent method is used for example, the weights

in N are changed as

$$\Delta w_j = -\rho \frac{\partial J}{\partial w_j}$$

where ρ is the step size. This training procedure can be graphically shown in Fig. 3-3. In the forward path, the neural network N has the inputs y_s , ϕ_s and $\hat{\theta}_s$, and its output is the adjustment $\Delta\hat{\theta}_s$. By using this preliminary $\Delta\hat{\theta}_s$, we can calculate ΔV_s from Eq. (3.15). ΔV_s is then compared with ΔV_{d_s} to obtain the error signal ΔV_{e_s} for correcting the weights W of the network. If ΔV_{e_s} is already less than zero, no correction is made in W . Otherwise, training is done by back-propagating ΔV_{e_s} through the intermediate function ΔV_s . In this formulation, as opposed to common neural network training, the outputs of the network do not have a direct target to compare with. Instead, after the outputs pass through another function, there is a distal target ΔV_{d_s} . This kind of structure is also known as “training with a distal teacher” [26]. By continuously applying the algorithm on a training set which contains all the sampled data required to calculate Eq. (3.19), the cost function J can then be minimized. It should be noted that since it is a nonlinear optimization problem, the solutions may converge to a local minimum.

Since we want the algorithm in Eq. (3.11) to be valid in the ranges of possible values of θ and ϕ as well as different initial estimates $\hat{\theta}$, the training set should cover those variables in their respective ranges. The training set also needs to include all the values required to calculate Eq. (3.19), specifically $\tilde{\theta}_s, \hat{\theta}_s, y_s, \hat{y}_s$ and ϕ_s , in order to apply the training algorithm. The procedure for gathering such a set of data is delineated below.

Similar to the procedure described in Section 3.3.2, by selecting $\theta = \theta_1$, we can formulate the triple $(\phi_1, y_1^1, \theta_1)$, where these values are related as $y_1^1 = f_r(\phi_1, \theta_1)$, and the subscript and superscript of y denote that the response is due to the specific θ_1 and ϕ_1 respectively. We also need to find out how the system responds to a different parameter value, say $\hat{\theta}_1$, so that we can train the network to learn the underlying relation between $(\hat{\theta}_1 - \theta_1)$ and y_1^1, \hat{y}_1^1 for the particular ϕ_1 , where

$$\hat{y}_1^1 = f_r(\phi_1, \hat{\theta}_1)$$

and the subscript of \hat{y} corresponds to the parameter $\hat{\theta}_1$. By combining θ_1 , ϕ_1 , y_1^1 , $\hat{\theta}_1$ and \hat{y}_1^1 , a data element can be formed as $(\theta_1, \phi_1, \hat{\theta}_1, y_1^1, \hat{y}_1^1)$ for a true parameter value θ_1 . This data element suffices to determine every term inside the summation of Eq. (3.19). Proceeding in the same manner, we can obtain a number of estimated outputs \hat{y}_j^1 as

$$\hat{y}_j^1 = f(\phi_1, \hat{\theta}_j), \quad j = 1, \dots, p$$

and collect p data sets $(\theta_1, \phi_1, \hat{\theta}_1, y_1^1, \hat{y}_1^1), \dots, (\theta_1, \phi_1, \hat{\theta}_p, y_1^1, \hat{y}_p^1)$ for various $\hat{\theta}_j \in \Theta^c$. By repeating the above procedure for $\phi = \phi_i$ where $i = 1, \dots, q$, we can also obtain y_1^i and \hat{y}_j^i of the plant not only for θ_1 and various $\hat{\theta}_j$ respectively, but also for different ϕ_i . A typical set of data T_1 can therefore be formed as

$$T_1 = \{(\theta_1, \phi_i, \hat{\theta}_j, y_1^i, \hat{y}_j^i) | 1 \leq i \leq q; 1 \leq j \leq p\}$$

We can also repeat the process for other possible parameters $\theta = \theta_k \in \Theta^c$ where $k = 2, \dots, r$ and form the corresponding sets T_k respectively, where $T_k = \{(\theta_k, \phi_i, \hat{\theta}_j, y_k^i, \hat{y}_j^i) | 1 \leq i \leq q; 1 \leq j \leq p\}$. The complete set of training data is then defined as

$$T_{train} = \bigcup_{1 \leq k \leq r} T_k$$

Similarly, a testing set T_{test} for cross-validation during network training can also be formed.

By utilizing the sets T_{train} and T_{test} , we can train the neural network as depicted in Figure 3-3. The procedure is summarized below, where we assume the stochastic gradient descent method is used to minimize the cost function (3.18). If other nonlinear optimization algorithms are used, *Step 4* below can be modified accordingly.

Step 1 Consider the s -th data element $(\theta_s, \phi_s, \hat{\theta}_s, y_s, \hat{y}_s)$ of T_{train} , where s denotes the index of the training set. By using $(y_s, \phi_s, \hat{\theta}_s)$ as inputs of the neural network, we can obtain the output vector of the network as, say, $\Delta \hat{\theta}_s$. If the neural network is thoroughly trained, it is expected that $|\hat{\theta}_s + \Delta \hat{\theta}_s - \theta_s| \leq |\hat{\theta}_s - \theta_s|$.

Step 2 The change, ΔV_s , corresponding to $\Delta \hat{\theta}_s$ is calculated from Eq. (3.15) by replacing $N(\cdot)$ with $\Delta \hat{\theta}_s$ as

$$\Delta V_s = 2\Delta \hat{\theta}_s^T \tilde{\theta}_s + \Delta \hat{\theta}_s^T \Delta \hat{\theta}_s$$

where $\tilde{\theta}_s = \hat{\theta}_s - \theta_s$.

Step 3 Calculate ΔV_{d_s} from Eq. (3.16).

Step 4 If $\Delta V_{e_s} > 0$, update w_j as

$$\Delta w_j = -\rho \Delta V_{e_s} \left[\tilde{\theta}_s + \Delta \hat{\theta}_s \right]^T \frac{\partial N}{\partial w_j}$$

The above procedure is repeated for every data element of the training set and for several epochs until J stops improving.

It is worth noting that \hat{y}_t , which appears in both the training phase in ΔV_{d_s} and the on-line implementation in Eq. (3.14), can be attained in several ways. First, if the plant model f_r in Eq. (3.4) is available, \hat{y}_t can be simply calculated from the model. Otherwise, during the training phase, if ϕ_s can be freely chosen, \hat{y}_s can be obtained by performing another experiment with $\hat{y}_s = f_r(\phi_s, \hat{\theta}_s)$. This avoids an inaccurate N due to model uncertainty in the training phase, while limiting using the plant model only to check whether $|\tilde{y}_t| > \delta$ during the on-line implementation phase. If neither is the model available nor can ϕ be chosen easily, we can construct another network N_y to estimate the output using ϕ_{t-1} and θ as inputs to mimic $f_y(\phi_{t-1}, \theta)$. To form the training set for N_y , we can follow the similar procedure discussed in this section. The data set $T_p(\theta_1) = \{(\theta_1, \phi_j, y_1^j) | 1 \leq j \leq p\}$ is obtained by measuring $y_1^j = f_r(\phi_j, \theta_1)$ for the corresponding ϕ_j and θ_1 . By repeating the process for different θ , we can form the training set as

$$T_{p_{train}} = \bigcup_{1 \leq i \leq q} T_p(\theta_i)$$

$N_y(\phi, \theta)$ is then trained on the set with inputs ϕ_j and θ_i , and target y_i^j using standard network training methods. After N_y is trained thoroughly, \hat{y}_t can be obtained as

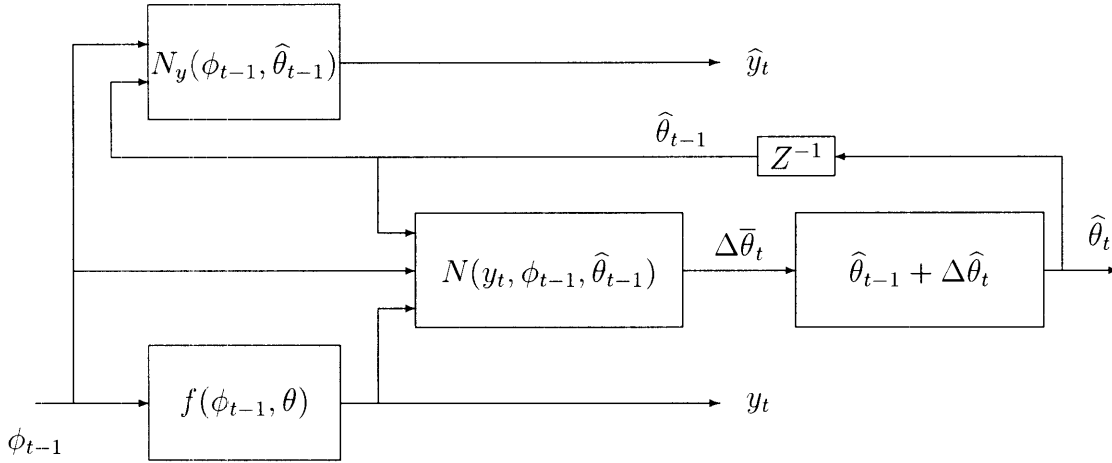


Figure 3-4: The Recursive Estimation Method (f_r unknown)

$\hat{y}_t = N_y(\phi_{t-1}, \hat{\theta}_{t-1})$. On-line implementation of the recursive method when f_r is unknown is shown schematically in Figure 3-4.

3.4.3 Convergence Analysis

For the procedure proposed in Section 3.4.2, the network is trained to make $\Delta V_e \leq 0$ by using all the measurable signals to determine $\Delta \hat{\theta}$. To guarantee the success of training, it must be shown that for any θ , $\Delta \hat{\theta}$ can indeed be found from those measurable signals. Furthermore, if the network is successfully trained, the algorithm in (3.14) can guarantee stability of parameter estimation. In this section, we will prove that the above statement is true under some assumptions. Moreover, those assumptions turn out to be satisfied for a wide class of nonlinear systems.

Consider the nonlinear system in Eq. (3.4) and the corresponding predictor for the estimate $\hat{\theta}_{t-1}$:

$$\begin{aligned} y_t &= f_r(\phi_{t-1}, \theta) \\ \hat{y}_t &= f_r(\phi_{t-1}, \hat{\theta}_{t-1}) \end{aligned} \quad (3.20)$$

where $\phi_{t-1} \in \Phi^c$, $\theta, \hat{\theta}_{t-1} \in \Theta^c$ and $y_t, \hat{y}_t \in \Psi^c$.

Theorem 3.2 *For the system in Eq. (3.20), let there exist continuous functions $C : \Psi^c \times \Phi^c \times \Theta^c \rightarrow \mathbb{R}^n$ and $h : \Psi^c \times \Phi^c \times \Theta^c \rightarrow \mathbb{R}$, and a function $R : \Phi^c \times \Theta^c \times \Theta^c \rightarrow \mathbb{R}$*

such that the following condition is satisfied:

$$\tilde{\theta}^T C(y, \phi, \hat{\theta}) + R(\phi, \theta, \hat{\theta}) = h(y, \phi, \hat{\theta}) \quad (3.21)$$

Then, the equilibrium point $\tilde{\theta} = 0$ of the parameter identification algorithm

$$\tilde{\theta}_t = \tilde{\theta}_{t-1} + ak\nu h(y_t, \phi_{t-1}, \hat{\theta}_{t-1}) C(y_t, \phi_{t-1}, \hat{\theta}_{t-1}) \quad (3.22)$$

is stable, where

$$\nu = \begin{cases} 1 & \text{if } |h(y, \phi, \hat{\theta})| > \delta \\ 0 & \text{otherwise} \end{cases}$$

$$k = -\frac{1}{1 + C^T C}$$

$$\alpha = \sup_{\phi \in \Phi, \theta \in \Theta, \hat{\theta} \in \Theta} |R(\phi, \theta, \hat{\theta})|, \delta \geq \frac{2}{b}\alpha, 0 < b \leq 1 \text{ and } 0 < a \leq 1.$$

Proof: Let $V_t = \tilde{\theta}_t^T \tilde{\theta}_t$ be a Lyapunov function candidate, where $\tilde{\theta}_t = \hat{\theta}_t - \theta$. If $|h(y_t, \phi_{t-1}, \hat{\theta}_{t-1})| \leq \delta$, then $\tilde{\theta}_t = \tilde{\theta}_{t-1}$ and therefore $V_t = V_{t-1}$ implying stability. Hence, we need to consider the case when $|h(y, \phi, \hat{\theta})| > \delta$. By triangular inequality, we can conclude from Eq. (3.21) that, if $|h(y, \phi, \hat{\theta})| > \delta$, then $\left| \frac{2R}{\tilde{\theta}^T C + R} \right| < b$. Thus,

$$\begin{aligned} \Delta V_t &\triangleq V_t - V_{t-1} \\ &= 2akh\tilde{\theta}_{t-1}^T C + a^2k^2h^2C^T C \\ &= \frac{ah^2}{(1 + |C|^2)^2} \left[-2 - (2 - a)|C|^2 + \frac{2R}{\tilde{\theta}^T C + R}(1 + |C|^2) \right] \\ &\leq -a(1 - b) \frac{2 + |C|^2}{(1 + |C|^2)^2} h^2 \\ &\leq 0 \end{aligned}$$

Hence, $\hat{\theta}_t \leq \hat{\theta}_{t-1}$. $\tilde{\theta} = 0$ is thus a stable equilibrium point. ■

Theorem 3.2 implies that if the condition in (3.21) is satisfied, then the parameter update law, determined as in Eq. (3.22), is a stable one. The question naturally arises as to whether this condition is indeed satisfied by a general nonlinear system. Since

f_r is twice continuously differentiable, if we expand the system in Eq. (3.20) around a point θ_0 using Taylor expansion, the following equation can be obtained:

$$\hat{y}_t - y_t = \left. \frac{\partial f_r(\phi_{t-1}, \theta)}{\partial \theta} \right|_{\theta_0} \tilde{\theta}_{t-1} + \theta_1^T H_1 \theta_1 - \theta_2^T H_2 \theta_2$$

where $\theta_1 = \hat{\theta}_t - \theta_0$ and $\theta_2 = \theta - \theta_0$; H_1 and H_2 are Hessian matrices of f_r with respect to θ evaluated at $\theta = \theta_0 + \rho_1 \theta_1$ and $\theta = \theta_0 + \rho_2 \theta_2$ respectively with $\rho_1, \rho_2 \in [0, 1]$. By comparing the above equation with (3.21), we can define C , R and h as follows:

$$\begin{aligned} C &= \left(\left. \frac{\partial f_r(\phi_{t-1}, \theta)}{\partial \theta} \right|_{\theta_0} \right)^T \\ R &= \theta_1^T H_1 \theta_1 - \theta_2^T H_2 \theta_2 \\ h &= \hat{y}_t - y_t \end{aligned} \tag{3.23}$$

Since f_r is continuous and Φ^c , Θ^c and Ψ^c are compact, δ in Theorem 3.2 exists.

The above argument and Theorem 3.2 establish the existence of a stabilizing continuous algorithm for a general nonlinear system. However, if a function C given by Eq. (3.23) is indeed chosen in (3.22), the region where $\nu = 1$ is satisfied can be very small. This is because this specific C is only a linear approximation around θ_0 . When θ or $\hat{\theta}$ is away from θ_0 , h is dominated by R and thus a large δ results. Fortunately, the neural network N is trained so as to have $\Delta V \leq \Delta V_d$. This allows more freedom in the function that N has to approximate and hence may result in a larger region where $\nu = 1$. The following theorem summarizes the stability of the algorithm when N is trained so that $\Delta V - \Delta V_d \leq \epsilon$.

Theorem 3.3 *For the system in Eq. (3.20), suppose a neural network N is trained such that $\Delta V_s - \Delta V_{d_s} \leq \epsilon$ for an $\epsilon > 0$ and for every $\phi_s \in \Phi^c$ and $\theta_s, \hat{\theta}_s \in \Theta^c$. If $\phi_{t-1} \in \Phi^c$ and $\hat{\theta}_{t-1} \in \Theta^c$ for every t , and $\theta \in \Theta^c$, then there exists a $\delta > 0$ so that the following algorithm*

$$\tilde{\theta}_t = \begin{cases} \tilde{\theta}_{t-1} + N(y_t, \phi_{t-1}, \hat{\theta}_{t-1}) & \text{if } |\tilde{y}_t| > \delta \\ \tilde{\theta}_{t-1} & \text{otherwise} \end{cases} \tag{3.24}$$

is stable at $\tilde{\theta}_t = 0$.

Proof: From the definition of ΔV_{d_t} , we have the following inequality:

$$\begin{aligned} |\Delta V_{d_t}| &= a \frac{2 + |C|^2}{(1 + |C|^2)^2} \tilde{y}^2 \\ &\geq a \frac{\tilde{y}^2}{1 + |C|^2} \end{aligned}$$

Since C is continuous and Φ^c , Θ^c and $f_r(\Phi^c, \Theta^c)$ are compact, the supremum of $|C|$ exists on $f_r(\Phi^c, \Theta^c) \times \Phi^c \times \Theta^c$. Thus, we can find a $\alpha > 0$ such that

$$|\Delta V_{d_t}| \geq \alpha \tilde{y}^2$$

Hence, if $\tilde{y}^2 \geq \frac{\epsilon}{\alpha}$,

$$\Delta V_t \leq \Delta V_{d_t} + \epsilon \leq 0$$

By choosing $\delta = \sqrt{\frac{\epsilon}{\alpha}}$, we have $V_t = V_{t-1}$ for $\tilde{y} \leq \delta$ and $V_t \leq V_{t-1}$ if $|\tilde{y}| > \delta$. $\tilde{\theta} = 0$ is thus stable for every $\phi_{t-1} \in \Phi^c$ and $\theta, \hat{\theta}_{t-1} \in \Theta^c$. ■

3.5 Comparison with Extended Kalman Filter

Since many efficient parameter estimation algorithms have been developed for linear systems, a natural extension is to modify nonlinear problems so that these algorithms can be applied to them. Most methods attempt to overcome this difficulty by linearizing the nonlinear system around the most recent parameter estimate. One such approach is the extended Kalman filter. In this section, the extended Kalman filter is used as an example to illustrate the difference between TANN and these linearized methods.

The extended Kalman filter for parameter estimation can be summarized as follows [23, 18]. The regression form in Eq. (3.4) can be rewritten in the following state space

representation with the unknown parameters θ as state variables:

$$\begin{aligned}\theta_{t+1} &= I\theta_t \\ y_{t+1} &= f_r(\phi_t, \theta_t)\end{aligned}\tag{3.25}$$

where I is an identity matrix and $\theta_0 = \theta$. If we linearize f_r around some estimate of θ , say, $\hat{\theta}_t$, the linear approximation of (3.25) can be written as

$$\begin{aligned}\theta_{t+1} &= I\theta_t \\ y_{t+1} &= \left. \frac{\partial f_r(\phi_t, \theta)}{\partial \theta} \right|_{\theta=\hat{\theta}_t} \theta_t\end{aligned}\tag{3.26}$$

With the above formulation, a standard Kalman filter can then be easily applied. The result is

$$\begin{aligned}\hat{\theta}_{t+1} &= \hat{\theta}_t + \frac{P(t-1)H^T(t)}{r+H(t)P(t-1)H^T(t)}(y_{t+1} - f(\phi_t, \hat{\theta}_t)) \\ P(t) &= P(t-1) - \frac{P(t-1)H^T(t)H(t)P(t-1)}{r+H(t)P(t-1)H^T(t)}\end{aligned}\tag{3.27}$$

where $H(t) = \left. \frac{\partial f(\phi(t), \theta)}{\partial \theta} \right|_{\theta=\hat{\theta}_t}$ and r denotes the variance of measurement noise.

A major difference between the extended Kalman filter as well as other algorithms that uses information from the linearized system and TANN is that the later utilizes the knowledge of the true parameter value during construction of the algorithms. This can be elaborated as follows. When the extended Kalman filter is applied on-line, since the true parameter value is unknown, the direction to correct parameter estimate is based on the gradient calculated at the current parameter estimate. If the estimate is far away from the true value, this gradient may not point in the right direction. As a result, even worse estimate could occur. Hence, there is no guarantee that these classes of algorithms would converge to the right value if at all. On the other hand, the fact that TANN is trained off-line is made full use of. Since during the off-line training phase the parameter value is known, the direction where the parameter update should be made at each measurement and for each parameter value is determined without resorting to estimation. Once the neural network has stored this information off-line, on-line estimation becomes simply choosing the appropriate correction in parameter estimate, which has already been obtained during training,

corresponding to the current measurement. Hence, more stable result can be expected from TANN algorithms. Furthermore, how the parameter estimate is updated is determined off-line. As shall be seen in Chapter 4, this property helps establish the stability of the controlled systems in a rigorous manner.

The above discussion can be illustrated geometrically in Figures 3-5, 3-6 and 3-7, which sketch respectively how parameter estimates using the TANN, the projection algorithm and the extended Kalman filter evolve during estimation. The projection algorithm is also derived from the linearized system at the most recent parameter estimate. In this fictitious example $y_t = f(\phi_{t-1}, \theta)$, it is assumed that only two different measurements $(y^{(1)}, \phi^{(1)})$ and $(y^{(2)}, \phi^{(2)})$ are available, and they appear alternately. The two thick solid lines L_1 and L_2 in the figures represent all possible combinations of the two parameters to be estimated that satisfy the relationships $y^{(1)} = f(\phi^{(1)}, \theta)$ and $y^{(2)} = f(\phi^{(2)}, \theta)$ respectively. The intersection of the two lines is certainly the true value. The thin solid curves represent the values of θ that satisfies $c_i = f(\phi^{(1)}, \theta)$ for various c_i , while the dotted curves correspond to those of satisfying $d_i = f(\phi^{(2)}, \theta)$. For the recursive TANN algorithm, since the parameter values are known off-line, the neural network can be trained to update parameter estimate approximately as what is shown in Figure 3-5. If the initial estimate is at the point a , the following estimates would gradually move toward the true value. However, for the projection algorithm, since the correction is always along the gradient calculated at the most recent estimate, the estimate would gradually diverge for this particular example. On the other hand, Figure 3-7 shows the trajectory of parameter estimate using the extended Kalman filter. For the sake of simplicity in illustration, it is assumed that $r = 0$ in Eq. (3.27), and thus, the extended Kalman filter becomes the orthogonalized projection algorithm [18]. The algorithm updates parameter estimate along the direction that is orthogonal to all previous directions of updates. Hence, after the first two updates, the new estimate stops at the point b . From this example, it can be seen that the quality of the results clearly depends on how well the linearized system resembles the original nonlinear system.

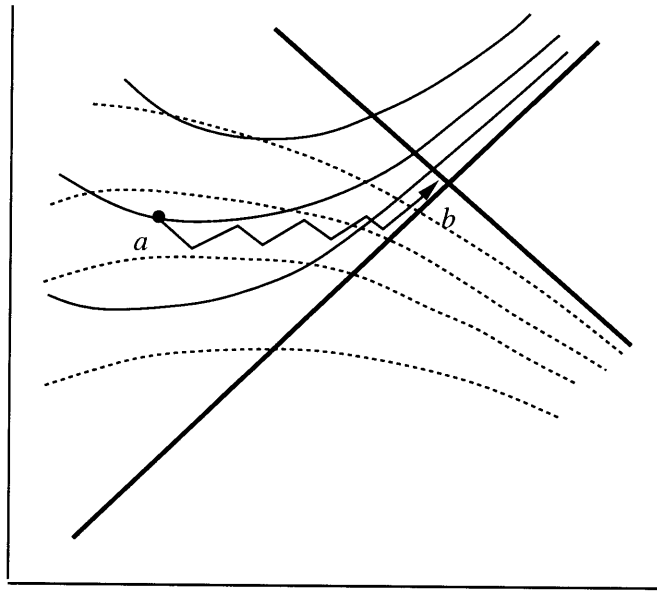


Figure 3-5: Illustration of the TANN Algorithm

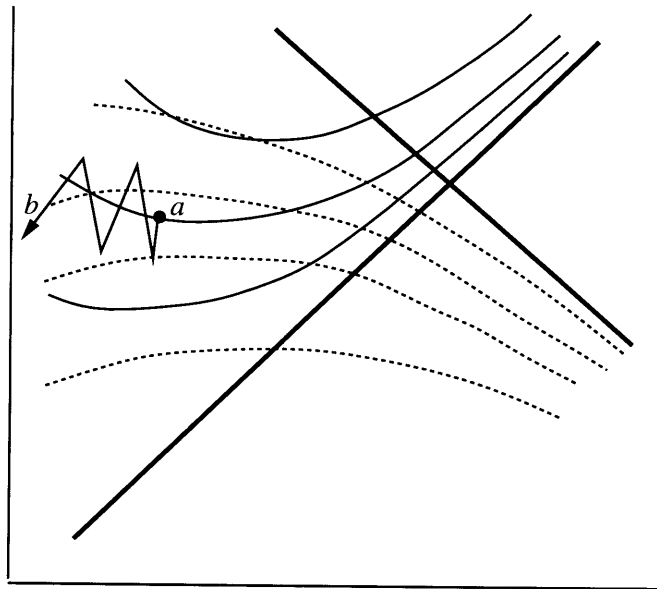


Figure 3-6: Illustration of the Projection Algorithm

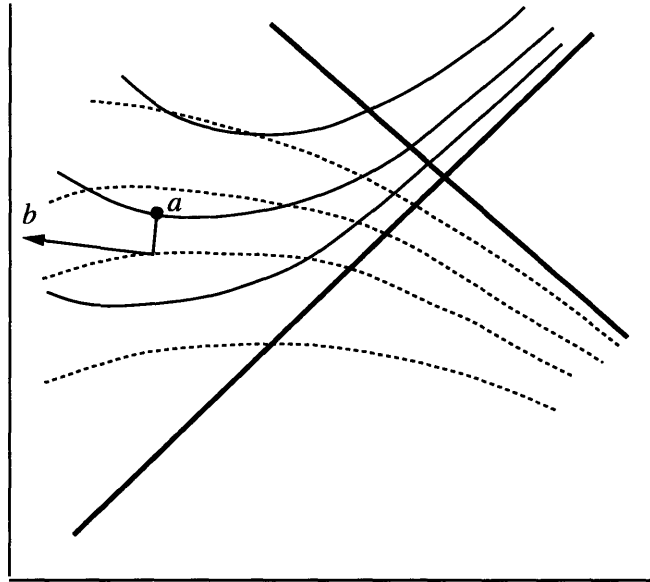


Figure 3-7: Illustration of the Extended Kalman Filter

3.6 Summary and Remarks

In this chapter, we have introduced a novel use of neural networks for parameter estimation. These parameters can occur nonlinearly and the structure of the nonlinearities may or may not be known. We make use of the nonlinear representative capability of neural networks to approximate the map between system variables and system parameters. By training the network off-line in a controlled environment where data can be collected regarding the system corresponding to known system parameters, it can be used on-line to estimate unknown parameters of a nonlinear and perhaps time-varying system. We denote the class of neural networks trained thus as θ -adaptive neural networks. Two different methods, based on the block and the recursive parameter estimation, are proposed. The analytical conditions under which the parameter estimates converge to their true values are presented. We also show that these conditions are indeed satisfied locally by a broad class of nonlinear systems.

The block estimation method, described in Section 3.3, uses a neural network to

construct the implicit function describing the relation between the responses and the parameters for general nonlinear dynamic systems. This method can be viewed as being analogous to the concept of the look-up table with the entry of the table Φ . The table is coded into the network using the training procedure described in Section 3.3.2. When the network is implemented on-line, it then “interpolates” the table to find the best parameter approximation that fits the measured responses. Since the neural network is constructed off-line, the on-line identification can be carried out fairly quickly. Furthermore, the algorithm requires no specific knowledge of the analytic form of the system for parameter identification since either an extensive simulation model or an experimental setup can substitute the analytical model to form T_{train} . Therefore the algorithm is useful in applications where the systems are too complex to model accurately.

We provide some comparisons between the recursive parameter estimation algorithm and the parameter identification procedures carried out in adaptive control [42, 18]. The role of the neural network in this case, as shown in Figure 3-2, is to generate the map between the quantities $\{y, \phi, \hat{\theta}\}$ and $\Delta\hat{\theta}$. In the case of linear parameter estimation, this mapping is defined by Eq. (3.10) using which convergence and other stability properties can be established. For parameter estimation in general nonlinear systems which is of interest here, we use a neural network as described in Section 3.4.1. As in the block parameter estimation method, the fact that the true parameter is known during training is directly used, as shown in Section 3.4.2. The stability properties of the algorithm proposed was established in Section 3.4.3, using a target function ΔV_d . Since the estimation is being carried out recursively, using instantaneous values of $\{y_t, \phi_t, \hat{\theta}_t\}$, ΔV_d was chosen to be a function of the scalar error function \tilde{y} rather than the parameter error vector $\tilde{\theta}$. Due to the approximating nature of neural networks, the algorithm is modified to incorporate a dead-zone like feature as in Eq. (3.14), similar to adaptive control algorithms [42]. Yet another feature of commonality between the adaptive algorithms and those proposed here is the effect of persistent excitation. As it shall be seen in Chapter 6, this leads to smaller parameter error. Unlike the block parameter estimation, when there is a lack of persistent exci-

tation, the parameter estimate from the recursive algorithm simply ceases to update, i.e. $\Delta\hat{\theta} \approx 0$, which again is similar to linear adaptive algorithms. Furthermore, since the recursive algorithm uses only the most recent values of $\{y_t, \phi_t, \hat{\theta}_t\}$, the dimension of the input space of the neural network is smaller, which results in a less complex network structure.

A comparison between the θ -adaptive neural networks and those in [43] is also worth making. Neural networks have been used in [43] as well as in many other papers to represent the nonlinear map from inputs to outputs. In contrast, we have suggested the use of neural networks for estimating the map between system observations and parameters. In [43], the neural network has been shown to represent different classes of nonlinear systems (denoted as models I-IV), which are input-output models, derived perhaps on an empirical basis. In this paper, our starting point is the model as in Eq. (3.2) or Eq. (3.3), which is based on the physical characteristics of the dynamic system, thereby providing a closer connection between neural network models and the actual physical models of the system. The target for training the neural network in [43] is the output error. More specifically, the performance index used for back propagation is a quadratic function of the output error which is an obvious target for the proposed use of the neural network. In our case, the goal is to reduce the parameter error. The advantage to be gained is that the resulting output, which is the true parameter of the nonlinear system, is more useful for adaptive control tasks.

Chapter 4

Adaptive Control of Nonlinear Dynamic Systems Using θ -Adaptive Neural Networks

4.1 Introduction

In many of the engineering problems, governing dynamics of the systems can be modeled using physical laws such as conservation of mass, momentum and energy. Physical constants in these models characterize key aspects of the behavior of the systems. Due to several factors including change in operating conditions, these physical constants may vary during operation. How to compensate for such variations is a challenging task for control engineers. The field of adaptive control deals specifically with this class of problems, where a control action has to take place despite the presence of uncertain and varying system parameters. For the sake of analytical tractability, simplifications are often introduced in the system models despite the presence of physically-based models.

In this chapter, we introduce a neural controller which makes direct use of the physical model of the system under consideration. The problem we focus on is the control of systems with parametric uncertainty while the structure of the system models is assumed to be known. This occurs in many systems where the underlying

physics that contributes to the system dynamics is well-known and can be modeled using physical laws such as conservation of energy and momentum. Yet, the parameters that characterize the operating points vary and need to be determined on-line. Since the system model is used in designing the controller, designers have more freedom in choosing the controller structure. However, difficulty arises in estimating the unknown parameters in the system if the physical model is used, since these parameters usually occur nonlinearly in the model. The approach that we suggest to overcome this problem is to use TANN discussed in Chapter 3. In contrast to common approaches in applying neural networks to control problems where the neural network is used as a controller model with its weights updated on-line, our approach consists of applying TANN to adjust the parameters of a controller whose structure is designed using the available physical model. In other words, the neural networks in our approach act as parameter adaptation mechanisms as opposed to controller models. The adaptation algorithm for a nonlinear system is constructed using a neural network rather than being solved analytically. Outputs of the neural network are thus correction in the controller parameters. There are several advantages for this kind of approach. Besides allowing a more flexible choice of controllers as the control design is based on the system model, there are much fewer number of parameters to be adjusted on-line compared to most neural control methods where typically hundreds of weights need to be varied on-line. This would significantly improve transient performance of the resulting closed-loop system especially when nonlinearity dominates, and in turn result in a larger region of applicability in state space and parameter space.

In addition to proposing a new way of incorporating neural networks for control purposes, another main contribution is the proof of stability of the closed-loop system. In contrast to much of the published literature on control using neural networks, we show that the proposed controller with a neural algorithm leads to closed-loop stability. While the stability is local, since the parameters adjusted on-line are closely associated with the physical model and usually have a physical meaning, the bounds on the initial parameter estimates that we shall establish in the proof are more tangible

(compared to the weights of a neural network) and can thus be easily checked for a given process. Furthermore, the assumption that is required in most adaptive control approaches that the system model is affine in the control input, is not needed in our approach.

In order for the neural network to adjust the controller parameters on-line in a stable manner, training of the network must be performed off-line for our proposed approach. Such an off-line training prior to proceeding to on-line parameter adaptation is a distinct departure from most other adaptive algorithms suggested for identification and control in the literature, which are simply implemented on-line with no prior training requirements. As shall be seen later, the off-line training is analogous to the process of finding a stable parameter adaptation algorithm for a linear adaptive system. Since the goal of the network training is to find such an algorithm, the targets of the training are certain properties that would guarantee stability of the closed-loop system when the network is implemented on-line, as opposed to the controller mapping that would minimize a performance cost function. The properties to be satisfied and the training process to achieve them are described in detail in this chapter.

This chapter is organized as follows. The problem as well as assumptions on the system model are described in Section 4.2. Properties that the neural network has to satisfy in order to guarantee stability of the overall system are discussed in Section 4.3. We describe a training method for the neural network to satisfy those properties in Section 4.4. Outlines of the proof of stability followed by the proof itself are shown in Section 4.5. The assumptions and sufficient conditions under which the stability holds are also discussed in this section. For systems where part of the model is linearly parametrized, simplification can be done to reduce complexity of the problem. We discuss one such scheme in Section 4.6.

4.2 Problem Statement

The focus of the nonlinear adaptive controller to be developed in this paper is on dynamic systems that can be written in the d -step ahead predictor form as follows:

$$y_{t+d} = f_r(\omega_t, u_t, \theta) \quad (4.1)$$

where

$$\omega_t^T = [y_t, \dots, y_{t-n+1}, u_{t-1}, \dots, u_{t-m-d+1}] \quad (4.2)$$

$n \geq 1$, $m \geq 0$, $d \geq 1$, $m + d = n$, $\mathcal{Y}_1, \mathcal{U}_1 \subset \mathfrak{R}$ containing the origin and $\Theta_1 \subset \mathfrak{R}^k$ are open, $f_r : \mathcal{Y}_1^n \times \mathcal{U}_1^{m+d} \times \Theta_1 \rightarrow \mathfrak{R}$ is a known function, y_t and u_t are the output and the input of the system at time t respectively, and θ is an unknown parameter and occurs nonlinearly in f_r .¹ The goal is to choose a control input u such that the system in (4.1) is stabilized and the plant output is regulated around zero.

4.2.1 Assumptions

Let

$$x_t^T \triangleq [y_{t+d-1}, \dots, y_{t+1}, \omega_t^T], \quad (4.3)$$

$$A_m = \left[\begin{array}{ccc|ccc} 0 & \dots & 0 & & & \\ & \mathbf{I} & \vdots & & & \\ & & 0 & & \mathbf{0}_{(n+d-1) \times (m+d-1)} & \\ \hline & & & 0 & \dots & 0 \\ \mathbf{0}_{(m+d-1) \times (n+d-1)} & & & & \mathbf{I} & \vdots \\ & & & & & 0 \end{array} \right], \quad B_m = \left[\begin{array}{cc} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \hline 0 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{array} \right] \quad (4.4)$$

We make the following assumptions regarding the system in Eq. (4.1).

¹Here, as well as in the following sections, A^n denotes the n -th product space of the set A .

(A1) For every $\theta \in \Theta_1$,

$$f_r(0, 0, \theta) = 0 \quad (4.5)$$

(A2) There exist open and convex neighborhoods of the origin $\mathcal{Y}_2 \subset \mathcal{Y}_1$ and $\mathcal{U}_2 \subset \mathcal{U}_1$, an open and convex set $\Theta_2 \subset \Theta_1$, and a function $K : E_1 \rightarrow \mathcal{U}_1$ such that for every $\omega_t \in \Omega_2$, $y_{t+d} \in \mathcal{Y}_2$ and $\theta \in \Theta_2$, Eq. (4.1) can be written as

$$u_t = K(\omega_t, y_{t+d}, \theta). \quad (4.6)$$

where

$$\Omega_2 \triangleq \mathcal{Y}_2^n \times \mathcal{U}_2^{m+d-1}, \quad E_1 \triangleq \Omega_2 \times \mathcal{Y}_2 \times \Theta_2.$$

(A3) K is twice differentiable and has bounded first and second derivatives on E_1 , while f_r is differentiable and has a bounded derivative on $\Omega_2 \times K(E_1) \times \Theta_2$.

(A4) There exists a $\delta_g > 0$ such that for every $y_1 \in f_r(\Omega_2, K(\Omega_2, 0, \Theta_2), \Theta_2)$, $\omega \in \Omega_2$ and $\theta, \hat{\theta} \in \Theta_2$,

$$\left| 1 - \left(\frac{\partial K(\omega, y, \theta)}{\partial y} - \frac{\partial K(\omega, y, \hat{\theta})}{\partial y} \right) \Big|_{y=y_1} \cdot \frac{\partial f_r(\omega, u, \theta)}{\partial u} \Big|_{u=u_1} \right| > \delta_g$$

(A5) There exist positive definite matrices P and Q of dimensions $(n + m + 2d - 2)$ such that for every $\theta \in \Theta_1$,

$$\begin{aligned} & x_t^T (A_m^T P A_m - P) x_t + [0, K(\omega_t, 0, \theta)] B_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} \\ & + 2x_t^T A_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} \leq -x_t^T Q x_t \end{aligned}$$

Assumptions (A1) and (A2) together imply $K(0, 0, \theta) = 0$ for every $\theta \in \Theta$. It is worth noting that if $\frac{\partial f_r}{\partial u_t} \neq 0$ at $\omega_t = 0$ and for some $\theta \in \Theta_1$ in Eq. (4.1), the existence of \mathcal{Y}_2 , \mathcal{U}_2 , Θ_2 and K is guaranteed by the implicit function theorem [40]. The smoothness assumption in (A3) is needed to apply the mean-value theorem to obtain upper bounds on growth rates of signals in the closed-loop system. Assumption (A4) is automatically satisfied if the gain multiplying u_t is known. This is similar to conditions on the high frequency gain in linear adaptive control. The assumption in (A5) essentially implies the decrease of

$$W_t \triangleq x_t^T P x_t \tag{4.7}$$

along the trajectory to be no slower than $-x_t^T Q x_t$ if $u_t = K(\omega_t, 0, \theta)$. A necessary condition for this assumption is for the zero dynamics of the closed-loop system to be stable.

4.2.2 The Adaptive Neural Controller

Since the objective is to control the system in (4.1) where θ is unknown, in order to stabilize the output y at the origin, we choose the control input as

$$u_t = K(\omega_t, 0, \hat{\theta}_t) \tag{4.8}$$

where $\hat{\theta}_t$ is the estimate of θ at time t . It can be seen from Eqs. (4.1) and (4.6) that the following relation holds:

$$0 = f_r(\omega_t, K(\omega_t, 0, \theta), \theta)$$

which implies that if the value of θ is known, the controller in Eq. (4.8) can drive the output y to the origin in one step. Since θ in Eq. (4.1) is unknown, the stabilizing controller needs to be combined with an adaptive algorithm that generates the parameter estimate $\hat{\theta}_t$.

Suppose the algorithm for updating $\hat{\theta}_t$ is defined recursively as

$$\Delta\hat{\theta}_t \triangleq \hat{\theta}_t - \hat{\theta}_{t-1} = R(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1})$$

the problem is to determine the function R such that $\hat{\theta}_t$ converges to θ asymptotically. In general, R is chosen to depend on y_t , ω_{t-d} , u_{t-d} and $\hat{\theta}_{t-1}$ since they are measurable and contain information regarding θ . For example, in the case of linear systems which can be cast in the input predictor form, $u_t = \phi_t^T \theta$, a well-known linear parameter estimation method to adjust $\Delta\hat{\theta}$ is the projection algorithm [18]

$$\Delta\hat{\theta}_t = \frac{\phi_{t-d}}{1 + \phi_{t-d}^T \phi_{t-d}} [u_{t-d} - \phi_{t-d}^T \hat{\theta}_{t-1}]. \quad (4.9)$$

In other words, the mechanism for carrying out parameter estimation is realized by R . In the case of general nonlinear systems, the task of determining such a function R is quite difficult, especially when the parameters occur nonlinearly. Towards this end, we propose the use of an artificial neural network N . That is, we adjust $\hat{\theta}_t$ as

$$\Delta\hat{\theta}_t = N(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1}) \quad (4.10)$$

where the inputs of the neural network are y_t , ω_{t-d} , u_{t-d} and $\hat{\theta}_{t-1}$, and the output is $\Delta\hat{\theta}_t$. The structure of the controller and the adaptive algorithm are shown in Fig. 4-1. The neural network is to be trained so that the resulting network can improve the parameter estimation over time for any possible $\theta \in \Theta_2$. In addition, the trained network must ensure that the overall system in Eqs. (4.1), (4.8) and (4.10) is stable. We show in the following sections that these are indeed achievable. By using a procedure similar to what we developed for the TANN algorithms [2] in parameter estimation, we show in Section 4.4 that we can successfully train the network. Under the conditions made in this section regarding f_r and K , we prove in Section 4.5 that the output of the closed-loop system converges to a neighborhood of the origin.

It must be noted that in order to apply the TANN controller, the controller structure specified by the function K in Eq. (4.6), which satisfies (A2)-(A5), should be

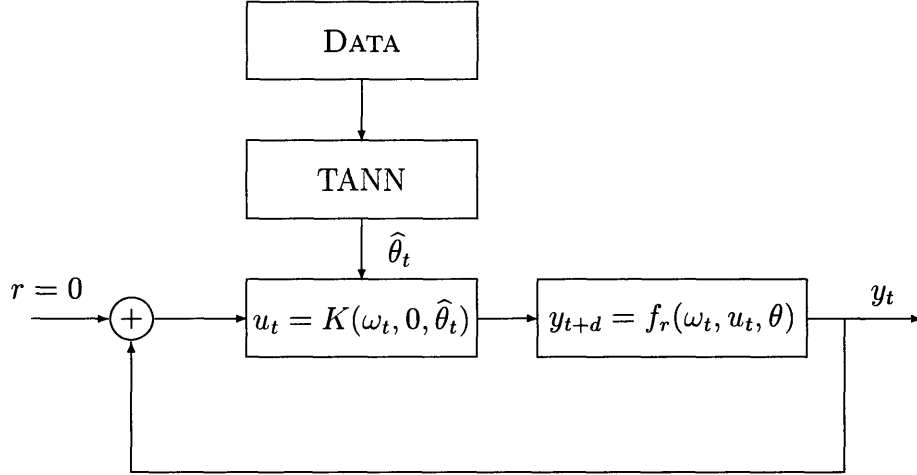


Figure 4-1: Structure of Adaptive Control Using TANN

explicitly known. A question arises as to under what conditions on f_r does such a function K exist. While we do not explicitly address the issue here, we argue that the class of such functions is quite large. For instance, consider the class of nonlinear systems with a triangular structure discussed in [28, 27, 56] of the form

$$\begin{aligned}
 x_{1_{t+1}} &= x_{1_t} + f_1(x_{1_t}, x_{2_t}, \theta) \\
 x_{2_{t+1}} &= x_{2_t} + f_2(x_{1_t}, x_{2_t}, x_{3_t}, \theta) \\
 &\cdot \\
 &\cdot \\
 x_{d_{t+1}} &= x_{d_t} + f_d(x_{1_t}, \dots, x_{d+1_t}, \theta) + u_t \\
 x_{d+1_{t+1}} &= x_{d+1_t} + f_d(x_{1_t}, \dots, x_{n_t}, u_t, \theta) \\
 &\cdot \\
 &\cdot \\
 x_{n_{t+1}} &= x_{n_t} + f_n(x_{1_t}, \dots, x_{n_t}, u_t, \theta) \\
 y_t &= x_{1_t}
 \end{aligned} \tag{4.11}$$

Assume $\frac{\partial f_i}{\partial x_{i+1}}(0, \dots, 0, \theta_0) \neq 0$ for $1 \leq i < d$. We can transform (4.11) into the form in (4.6) by following the steps described below. Since $\frac{\partial f_i}{\partial x_{i+1}}(0, \dots, 0, \theta_0) \neq 0$, we can

write

$$x_{i+1_{t+d-i}} = g_i(x_{1_{t+d-i}}, \dots, x_{i_{t+d-i}}, x_{i_{t+d-i+1}}, \theta) \quad (4.12)$$

for some function $g_i : \mathcal{Y}_1^{i+1} \times \Theta_1$, where \mathcal{Y}_1 and Θ_1 are open neighborhoods of the origin and θ_0 respectively, and $1 \leq i < d$. It can be observed from (4.11) that by continuously substituting $x_{i_{t+d-i+1}} = x_{i_{t+d-i}} + f_i(x_{1_{t+d-i}}, \dots, x_{i_{t+d-i}})$ into the state equation $x_{i-1_{t+d-i+2}}$ and utilizing (4.12) we can obtain

$$u_t = g(x_{1_{t+d}}, x_{1_{t+d-1}}, \dots, x_{1_t}, \dots, x_{i_{t+d-i}}, \dots, x_{i_t}, \dots, x_{d_t}, \theta)$$

By applying (4.12) again, we can represent $x_{1_{t+d}}, \dots, x_{d_t}$ in terms of $x_{1_t}, \dots, x_{1_{t-d+1}}$ and $u_{t-1}, \dots, u_{t-d+1}$. The above equation is then in a predictor model form

$$u_t = K(y_{t+d}, y_t, y_{t-1}, \dots, y_{t-d+1}, u_{t-1}, \dots, u_{t-d+1}, \theta) \quad (4.13)$$

which is similar to that in Eq. (4.6) with $\omega_t^T = [y_t, y_{t-1}, \dots, y_{t-d+1}, u_{t-1}, \dots, u_{t-d+1}]$.

If the internal dynamics

$$\begin{aligned} x_{d+1_{t+1}} &= x_{d+1_t} + f_d(x_{1_t}, \dots, x_{n_t}, u_t, \theta) \\ &\cdot \\ &\cdot \\ x_{n_{t+1}} &= x_{n_t} + f_n(x_{1_t}, \dots, x_{n_t}, u_t, \theta) \end{aligned}$$

is stable for $u_t \in \mathcal{U}_2$ and $\theta \in \Theta_2$, the predictor model in Eq. (4.13) suffices to be used as a controller. Though outside the scope of this paper, it is possible to determine conditions on f_i in (4.11) under which (A3)-(A5) are satisfied.

4.3 TANN Algorithms for Control

A question that naturally arises is whether the same algorithm and training procedure developed for parameter estimation in the previous chapter can be applied directly for estimating θ in the adaptive control problem stated in Section 4.2. Since in the

control problem, we are interested not only in generating bounded parameter estimates $\hat{\theta}_t$ but also in establishing stability of the closed-loop system, we may need to impose additional conditions on the neural network and its training. In this section, we introduce two sufficient conditions that the neural network must satisfy for stability. In Section 4.4, we show how the network can be trained so as to satisfy these conditions. In Section 4.5, we show that under these conditions, closed-loop stability follows.

To quantify the parameter estimation error, we use the same norm as in Eq. (3.12). By comparing Eq. (4.6) with Eq. (3.4), we can see that $[\omega_t, y_{t+d}]$ in Eq. (4.6) have the role similar to ϕ_{t-1} in Eq. (3.4), while y_t in Eq. (3.4) is similar to u_t in Eq. (4.6). Hence, referring to Eqs. (3.11), (3.13) and (3.23), similar $\Delta\hat{\theta}_t$, ΔV_{dt} and $C(\bar{\phi}_t)$ can be defined for the control parameter estimation as

$$\begin{aligned}\Delta\hat{\theta}_t &= N(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1}) \\ \Delta V_{dt} &= -a_1 \frac{2 + |C(\bar{\phi}_{t-d})|^2}{(1 + |C(\bar{\phi}_{t-d})|^2)^2} \tilde{u}_{t-d}^2 \\ C(\bar{\phi}_t) &= \left(\frac{\partial K}{\partial \theta} (\omega_t, y_{t+d}, \theta) \Big|_{\theta=\theta_0} \right)^T\end{aligned}\tag{4.14}$$

where

$$\tilde{u}_t = u_t - K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1})\tag{4.15}$$

$$\bar{\phi}_t = [\omega_t^T, y_{t+d}]^T,\tag{4.16}$$

$a_1 \in (0, 1)$ and $\theta_0 \in \Theta_2$ as in (3.23) is the point where K is linearized. Unlike TANN for parameter estimation where $\Delta V_t \leq 0$ is sufficient to guarantee stability, the control problem does require that $\Delta V_t \leq \Delta V_{dt}$, or at least

$$\Delta V_t - \Delta V_{dt} < \epsilon_1\tag{4.17}$$

where ϵ_1 is a small positive constant. Besides making the proof tractable due to the knowledge of the minimum rate of parameter convergence, another reason for

requiring $\Delta V_t - \Delta V_{d_t} < \epsilon_1$ is that if the convergence rate is too slow, the system output may diverge before the controller parameter converges to a stable set. Therefore, ΔV_t should be bounded above by a negative definite function of \tilde{u}_{t-d} . These points shall be made clear when we prove stability of the system in Section 4.5.

The choices of the functions ΔV_{d_t} and C in (4.14) for the control problem are motivated by the projection algorithm (4.9) used for linear parameter estimation. That is, if θ is close to θ_0 , the projection algorithm yields a $\Delta V \approx \Delta V_d$ given by (4.14), and hence a stable estimation procedure. In the problem under consideration, we selected these functions to provide a negative definite target for ΔV since the goal is to achieve $\Delta V \leq 0$. The training procedure, derived in detail in Section 4.4, seeks to train N in (4.14) so that ΔV satisfies (4.17). The parameter estimation algorithm in (4.14) however is not determined as a specific continuous function with a closed form but is constructed using a neural network which is trained to approximate this continuous function. Since the algorithm attempts to generate a ΔV which is less than or equal to ΔV_d and not achieve an exact equality, its construction requirements are less stringent and therefore, it is amenable to generating satisfactory performance over a range of parameter values of θ and not merely for θ close to θ_0 . This is illustrated in Section 6.3.

For the parameter estimation problem in Chapter 3, it was only required that the parameter error norm be non-increasing. In the control problem, in addition to having $\Delta V_t \leq \Delta V_{d_t}$, we need yet another property, which is that the magnitude of the parameter correction $\Delta \hat{\theta}_t$ at each step has to be small enough if \tilde{u}_{t-d} is small. Since $\Delta \hat{\theta}_t$ is the output of the network, we limit the output vector such that

$$|N(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1})|^2 \leq a_2 \frac{|C(\bar{\phi}_{t-d})|^2}{(1 + |C(\bar{\phi}_{t-d})|^2)^2} \tilde{u}_{t-d}^2 \quad (4.18)$$

where $a_1 < a_2 \leq 1$.

Feasible corrections of parameter estimation satisfying the two properties can be graphically shown in Figure 4-2. In the figure, (θ_1, θ_2) denoted by ‘×’ is the true value of parameter, while $(\hat{\theta}_1, \hat{\theta}_2)$ labeled by ‘*’ represents the current estimate. Hence, the

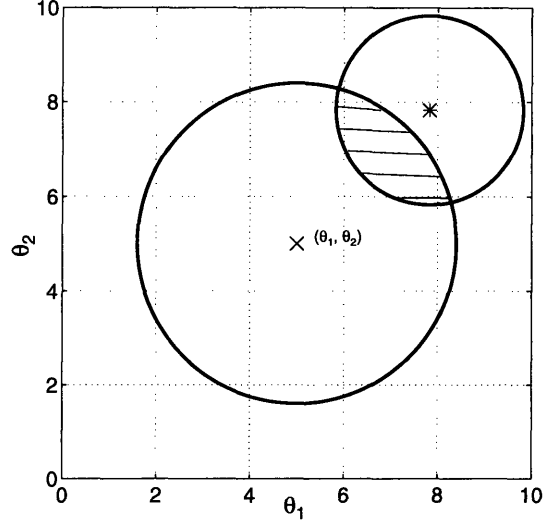


Figure 4-2: Feasible Region of Neural Network Output for Adaptive Control

requirement $\Delta V_t \leq \Delta V_{d_t}$ limits the new estimate to be somewhere inside the large disk whose radius is determined by ΔV_{d_t} . For parameter estimation, this would be enough. However, for the control problem, the additional property in Eq. (4.18) specifies the maximum distance where the next estimate can move, which is defined by the right-hand side of Eq. (4.18). It therefore restricts the new estimate to within the small disk centered at '*'. The intersection of the two disks is the feasible region of the neural network output.

Similar to the parameter estimation case, a dead-zone modification is also made in TANN for the control problem to guarantee $\Delta V_t \leq 0$. Suppose there is an $\epsilon_1 > 0$ so that $\Delta V_t - \Delta V_{d_t} < \epsilon_1$. The algorithm in Eq. (4.10) is changed to

$$\hat{\theta}_t = \begin{cases} \hat{\theta}_{t-1} + N(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1}) & \text{if } \Delta V_{d_t} < -\epsilon \\ \hat{\theta}_{t-1} & \text{otherwise} \end{cases} \quad (4.19)$$

where $\epsilon > \epsilon_1$.

In summary, we are interested in finding a function $N : \mathcal{Y}_3^{n+1} \times \mathcal{U}_3^{m+d} \times \Theta_3 \rightarrow \mathbb{R}^k$, where $\mathcal{Y}_3 \subset \mathcal{Y}_2$, $\mathcal{U}_3 \subset \mathcal{U}_2$ and $\Theta_3 \subset \Theta_2$ are compact sets, and \mathcal{Y}_3 , \mathcal{U}_3 contain neighborhoods of the origin, such that for every $\omega_{t-d} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $\theta \in \Theta_3$ and

$\hat{\theta}_{t-1} \in \Theta_3$, N satisfies the following two properties:

$$(P1) \quad |N(y_t, \omega_{t-d}, u_{t-d}, \hat{\theta}_{t-1})|^2 \leq a_2 \frac{|C(\bar{\phi}_{t-d})|^2}{(1+|C(\bar{\phi}_{t-d})|^2)^2} \tilde{u}_{t-d}^2$$

$$(P2) \quad \Delta V_t - \Delta V_{d_t} < \epsilon_1, \quad \epsilon_1 > 0.$$

if $u_{t-d} \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_{t-d}, \theta}(\mathcal{Y}_3)$, where $f_r^{-1}|_{\omega_{t-d}, \theta}(\mathcal{Y}_3)$ denotes the inverse image of the set \mathcal{Y}_3 under the mapping $f_r(\omega, u, \vartheta)$ at $\omega = \omega_{t-d}$ and $\vartheta = \theta$. With this neural network N , TANN algorithm is then implemented as in (4.19) with $\epsilon > \epsilon_1$.

The reason why we restrict u_{t-d} in $\mathcal{U}_3 \cap f_r^{-1}|_{\omega_{t-d}, \theta}(\mathcal{Y}_3)$ is that for arbitrary u_{t-d} , y_t does not necessarily stay within \mathcal{Y}_3 . As a result, the subsequent ω_{t-d+1} may be outside the set $\mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$ where N is trained. If this happens, the network output needs to be discarded since its approximation is unreliable outside the compact set. Hence, we do not train the network on those patterns which result in such a scenario. However, when (4.19) is actually implemented on the closed-loop system, the question remains whether those patterns can actually be avoided. In Section 4.5, we prove this is indeed true as long as the initial estimation error is small enough.

It is also worth noting that although we require the above two properties to establish stability of the closed-loop system, it is still possible for algorithms violating these two properties to result in a stable system since they are only sufficient conditions. Nevertheless, these two properties provide us with mathematical tractability in the proof.

4.4 Training of TANN for Control

In the previous section, we proposed an algorithm using a neural network for adjusting the control parameters. We introduced two properties (P1) and (P2) of the identification algorithm that the neural network needs to possess in order to maintain stability of the closed-loop system. In this section, we discuss the training procedure by which the weights of the neural network are to be adjusted so that the network retains these properties.

As in the training of TANN in Chapter 3, a training set is constructed off-line

first. Since we want the algorithm in Eq. (4.19) to be valid on the specified sets \mathcal{Y}_3 and \mathcal{U}_3 for different θ and $\hat{\theta}$ in Θ_3 , the training set should cover those variables appearing in Eq. (4.19) in their respective ranges. Hence, we first sample ω in the set $\mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, and $\theta, \hat{\theta}$ in the set Θ_3 . Their values are, say, ω_1, θ_1 and $\hat{\theta}_1$ respectively. For the particular $\hat{\theta}_1$ and θ_1 we sample $\hat{\theta}$ again in the set $\{\theta \in \Theta_3 \mid |\theta - \theta_1| \leq |\hat{\theta}_1 - \theta_1|\}$, and its value is, say, $\hat{\theta}_1^d$. The reason for having two different samples of $\hat{\theta}$ (i.e. $\hat{\theta}_1$ and $\hat{\theta}_1^d$) is elaborated as follows. It can be observed from Eqs (4.1), (4.8) and (4.19) that $\hat{\theta}$ appears at two different places in the closed-loop system: the parameter estimate at time t is used to assign u_t and is used at time $t + d - 1$, together with y_{t+d} , ω_t and u_t , to obtain the new estimate $\hat{\theta}_{t+d}$ as well as \hat{u}_t . Hence, we need $\hat{\theta}_1$ and $\hat{\theta}_1^d$ to represent samples of the variables $\hat{\theta}_t$ and $\hat{\theta}_{t+d-1}$ respectively. Since $\hat{\theta}_{t+d-1}$ is $(d - 1)$ steps ahead of $\hat{\theta}_t$, if the parameter estimation is successful, $\hat{\theta}_1^d$ should always be closer to the true parameter than $\hat{\theta}_1$. Therefore, we limit the samples of $\hat{\theta}_1^d$ in the set $\{\theta \in \Theta_3 \mid |\theta - \theta_1| \leq |\hat{\theta}_1 - \theta_1|\}$. However, for the special case $d = 1$ in Eq. (4.1), only $\hat{\theta}_1$ is required since $t + d - 1 = t$.

Once $\omega_1, \theta_1, \hat{\theta}_1$ and $\hat{\theta}_1^d$ are sampled, other data can then be calculated, such as $u_1 = K(\omega_1, 0, \hat{\theta}_1)$ and $y_1 = f_r(\omega_1, u_1, \theta_1)$, where u_1 corresponds to a sample of u at time t , whereas y_1 corresponds to a sample of y at time $t + d$. If $y_1 \notin \mathcal{Y}_3$, this pattern is ignored for reasons explained in the end of Section 4.3. Otherwise, we can also obtain the corresponding $C(\bar{\phi}_1), \Delta V_{d_1}$ and L_1 as follows:

$$\begin{aligned} C(\bar{\phi}_1) &= \frac{\partial K}{\partial \theta}(\omega_1, y_1, \theta_0) \\ \Delta V_{d_1} &= -a_1 \frac{2 + |C(\bar{\phi}_1)|^2}{(1 + |C(\bar{\phi}_1)|^2)^2} (u_1 - \hat{u}_1)^2 \\ L_1 &= a_2 \frac{|C(\bar{\phi}_1)|^2}{(1 + |C(\bar{\phi}_1)|^2)^2} (u_1 - \hat{u}_1)^2 \end{aligned}$$

where $\bar{\phi}_1 = [\omega_1^T, y_1]^T$ and $\hat{u}_1 = K(\omega_1, y_1, \hat{\theta}_1^d)$. A data element can then be formed as $(y_1, \omega_1, u_1, \hat{\theta}_1^d, \theta_1, \Delta V_{d_1}, L_1)$. This data element suffices to determine every term on the right-hand side of Eq. (4.22). Proceeding in the same manner, by choosing various

$\omega_s, \theta_s, \hat{\theta}_s$ and $\hat{\theta}_s^d$ in their respective ranges, we form a typical training set T_{train} as

$$T_{train} = \left\{ (y_s, \omega_s, u_s, \hat{\theta}_s^d, \theta_s, \Delta V_{d_s}, L_s) \mid 1 \leq s \leq M \right\}$$

where M denotes the total number of points in the training set. Similarly, a testing set T_{test} can also be formed for cross-validation during the network training.

Let s denote an index of the pattern in the training set. ΔV_s and ΔV_{e_s} for the particular pattern s are

$$\begin{aligned} \Delta V_s &= |\hat{\theta}_s + N_s(W) - \theta_s|^2 - |\hat{\theta}_s - \theta_s|^2 \\ \Delta V_{e_s} &= \Delta V_s - \Delta V_{d_s} \end{aligned}$$

where $N_s(W)$ is an abbreviation of $N(y_s, \omega_s, u_s, \hat{\theta}_s; W)$ and W denotes weights in the network.

To establish (P1) and (P2) in the previous section, it is necessary that for every pattern in the training set, W must be such that

$$\begin{aligned} |N_s(W)|^2 &\leq L_s \\ \Delta V_{e_s}(W) &\leq 0, \quad s = 1, \dots, M \end{aligned} \tag{4.20}$$

In order to satisfy the inequalities in (4.20), we view the problem as one of constrained optimization with no cost function. We first convert the inequality constraints in (4.20) into equality constraints by adding additional variables v_i and z_i :

$$|N_i(W)|^2 - L_i + z_i^2 = 0, \quad \Delta V_{e_i}(W) + v_i^2 = 0, \quad i = 1, \dots, M$$

If the quadratic penalty function method [7] is used, we can formulate the optimization problem as follows:

$$\begin{aligned} \min_{W, v, z, \mu, \zeta} \sum_{i=1}^M \left\{ \mu_i (\Delta V_{e_i}(W) + v_i^2) + \frac{\epsilon}{2} |\Delta V_{e_i}(W) + v_i^2|^2 \right. \\ \left. + \frac{1}{b^2} \left(\zeta_i (|N_i(W)|^2 - L_i + z_i^2) + \frac{\epsilon}{2} ||N_i(W)|^2 - L_i + z_i^2|^2 \right) \right\} \end{aligned}$$

where μ_i and ζ_i are Lagrange multipliers, c is a constant and $\frac{1}{b^2}$ is a weighting constant. Minimization with respect to v and z can be carried out in closed form [7], the result can be written as

$$\min_{W, \mu, \zeta} \sum_{i=1}^M \left\{ (\max\{0, \mu_i + c\Delta V_{e_i}\})^2 - \mu_i^2 + \frac{1}{b^2} \left[(\max\{0, \zeta_i + c(|N_i(W)|^2 - L_i)\})^2 - \zeta_i^2 \right] \right\}$$

Since the optimization involves only inequality constraints, we can further reduce the problem into the following form:

$$\min_W J \triangleq \min_W \frac{1}{2} \sum_{i=1}^M \left\{ (\max\{0, \Delta V_{e_i}\})^2 + \frac{1}{b^2} (\max\{0, |N_i(W)|^2 - L_i\})^2 \right\} \quad (4.21)$$

To find a W which minimizes the above unconstrained cost function J , we can apply algorithms such as the gradient method and the Gauss-Newton method. The gradient of J with respect to a weight (or bias) w_j , which is required for either algorithm, can be obtained as

$$\begin{aligned} \frac{\partial J}{\partial w_j} &= \sum_{s \in P_1} 2\Delta V_{e_s} [\tilde{\theta}_s + N(y_s, \omega_s, u_s, \hat{\theta}_s)]^T \frac{\partial N}{\partial w_j} \\ &\quad + \frac{1}{b^2} \sum_{s \in P_2} 2(|N_s|^2 - L_s) N^T(y_s, \omega_s, u_s, \hat{\theta}_s) \frac{\partial N}{\partial w_j} \end{aligned} \quad (4.22)$$

where P_1 and P_2 denote the sets of patterns for which $\Delta V_{e_s} > 0$ and $|N_s|^2 - L_s > 0$ respectively. The term $\frac{\partial N}{\partial w_j} = [\frac{\partial N_1}{\partial w_j}, \dots, \frac{\partial N_n}{\partial w_j}]^T$ is the gradient of the output vector of the neural network with respect to a weight w_j in the network. The procedure of neural network training is summarized below.

Step 1. Consider the s -th data element $(y_s, \omega_s, u_s, \hat{\theta}_s^d, \theta_s, \Delta V_{d_s}, L_s)$ of T_{train} , where s is an index of the training set. By using y_s, ω_s, u_s and $\hat{\theta}_s^d$ as inputs of the neural network, we can obtain the output vector of the network as, say, $\Delta \hat{\theta}_s$.

Step 2. ΔV_s corresponding to this particular $\Delta \hat{\theta}_s$ is calculated from the definition as:

$$\Delta V_s = 2\Delta \hat{\theta}_s^T (\hat{\theta}_s^d - \theta_s) + \Delta \hat{\theta}_s^T \Delta \hat{\theta}_s$$

If the neural network is thoroughly trained, it is expected that $\Delta V_s \leq \Delta V_{d_s}$ and $|\Delta \hat{\theta}_s|^2 \leq L_s$.

Step 3. Update the j -th weight w_j of the network as

$$\Delta w_j \triangleq w_j^{\text{new}} - w_j^{\text{old}} = -\rho \left(\delta[\Delta V_{e_s}] \left[\tilde{\theta}_s + N(y_s, \omega_s, u_s, \hat{\theta}_s) \right]^T + \frac{1}{b^2} \delta[|N_s|^2 - L_s] N^T(y_s, \omega_s, u_s, \hat{\theta}_s) \right) \frac{\partial N}{\partial w_j}$$

where $\delta[x] = x$ if $x > 0$ and $\delta[x] = 0$ otherwise. Repeat the correction for every weight in the network.

The above procedure is repeated for every data element of the training set and for several epochs until J in Eq. (4.21) is within tolerance.

In Step 3, Δw_j is calculated using the stochastic gradient descent method. If the Gauss-Newton method is used for example [52], we can replace Step 3 with the following steps:

Step 3a Calculate α_{kl} and β_k as

$$\begin{aligned} \alpha_{kl} &= \sum_{i \in P_1} \left[\frac{\partial \Delta V_i}{\partial w_k} \frac{\partial \Delta V_i}{\partial w_l} \right] + \sum_{j \in P_2} \frac{1}{b^2} \left[\frac{\partial |N_j|^2}{\partial w_k} \frac{\partial |N_j|^2}{\partial w_l} \right] \\ \beta_k &= - \sum_{i \in P_1} \Delta V_{e_i} \frac{\partial \Delta V_i}{\partial w_k} - \sum_{j \in P_2} (|N_j|^2 - L_j) \frac{\partial |N_j|^2}{\partial w_k} \end{aligned}$$

where w_k is the k -th element of the weights in the network, and

$$\begin{aligned} \frac{\partial \Delta V_i}{\partial w_k} &= 2 \left[\tilde{\theta}_i + N(y_i, \omega_i, u_i, \hat{\theta}_i) \right]^T \frac{\partial N}{\partial w_k} \\ \frac{\partial |N_i|^2}{\partial w_k} &= 2 N^T(y_i, \omega_i, u_i, \hat{\theta}_i) \frac{\partial N}{\partial w_k} \end{aligned}$$

Step 3b Solve the following n_w linear equations for Δw_l :

$$\sum_{l=1}^{n_w} \alpha_{kl} \Delta w_l = \beta_k, \quad k = 1, 2, \dots, n_w$$

where n_w is the total number of variable coefficients (weights) in the network and

Δw_l is the update of the l -th weight.

The Gauss-Newton method or its variants such as the Levenberg-Marquardt method [38] are very efficient in finding a local minimum of the cost function J . Since the optimization problem is nonlinear, there is no guarantee that the local minimum is also a global one unless the initial choice of weights are close to the optimal ones. Hence, a common practice is to start from several different initial weights to increase the chance of reaching a global minimum. An alternative is to use global search techniques such as the simulated annealing method [52], albeit the global methods are usually computationally more expensive than their local counterparts. After a minimum of J is reached, whether it is local or global, a question that arises is if the resulting upper bound on ΔV_{e_i} of the training set is small enough. To ensure stability of the closed-loop system when TANN is implemented on-line, however, what is relevant is not whether or not J reaches a global minimum, but how small the resulting upper bound on ΔV_{e_i} is after a minimum of J is reached. This is discussed at the end of Section 4.5.

4.5 Proof of Stability

With the plant given by Eq. (4.1), the controller by Eq. (4.8), the TANN parameter estimation algorithm by Eq. (4.19), and the TANN trained using Steps 1-3 (presented in Section 4.4) off-line, we proceed to show in this section that the resulting closed-loop system is stable. Before providing the proof, two points that are specific to neural controllers must be mentioned. First, since a neural network can only approximate a nonlinear function in a compact set, care must be taken to ensure that all signals entering the network in the closed loop indeed belong to the compact set where it is trained. Second, due to the approximating nature of the networks and nonlinearity of the functions to be approximated, there can be a nonzero error between the target and the approximator. In other words, only a positive ϵ can be achieved such that $(\Delta V - \Delta V_d) \leq \epsilon$. In order to avoid an increasing parameter estimation error, a dead-zone is incorporated in (4.19). While proving stability, the effect of the dead zone

should also be taken into account.

The closed-loop system, specified by Eqs. (4.1), (4.8) and (4.19), is rewritten below:

$$\begin{aligned} y_{t+d} &= f_r(\omega_t, u_t, \theta) \\ u_t &= K(\omega_t, 0, \hat{\theta}_t) \\ \hat{\theta}_{t+d} &= \begin{cases} \hat{\theta}_{t+d-1} + N(y_{t+d}, \omega_t, u_t, \hat{\theta}_{t+d-1}) & \text{if } \Delta V_{d_{t+d}} < -\epsilon \\ \hat{\theta}_{t+d-1} & \text{otherwise} \end{cases} \end{aligned}$$

To analyze the stability of the system, we use the state variable x_t as defined in Eq. (4.3). The closed-loop system excluding the dynamics of the parameter estimates can be written in the following state space form:

$$\begin{aligned} x_{t+1} &= A_m x_t + B_m \begin{bmatrix} f_r(\omega_t, K(\omega_t, 0, \hat{\theta}_t), \theta) \\ K(\omega_t, 0, \hat{\theta}_t) \end{bmatrix} \\ &= A_m x_t + B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} + B_m F_t \end{aligned} \quad (4.23)$$

where A_m and B_m are defined in Eq. (4.4), and

$$F_t = \begin{bmatrix} f_r(\omega_t, K(\omega_t, 0, \hat{\theta}_t), \theta) \\ K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta) \end{bmatrix} \quad (4.24)$$

Also, the closed-loop system is assumed to satisfy Assumptions (A1)-(A5) stated and discussed in Section 4.2.1.

Before proceeding to the proof, we enumerate the key steps which lead to the stability of the closed-loop system. To analyze the stability, we focus on the evolution of W_t , defined in Eq. (4.7), over time.

(C1) If $\Delta W_t = W_{t+1} - W_t$, then

$$\begin{aligned} \Delta W_t &< -\lambda_{\min}(Q)|x_t|^2 + c_9|x_t| \left| K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta) \right| \\ &\quad + c_{10} \left| K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta) \right|^2 \end{aligned} \quad (4.25)$$

which is established by utilizing the smoothness assumptions of f_r and K in the bounded sets as in (A3).

(C2) The following two properties of the TANN algorithm, pertaining to parameter convergence, can be established. If all the signals entering the network remain within the sets where the network is trained, then

- (i) the parameter estimation error is non-increasing;
- (ii) given an $\epsilon > 0$, there are only certain number of iterations such that $|\Delta V_{d_t}| > \epsilon$ for any length of interval.

(C3) Based on property (P1) of TANN and assumption (A4), we can derive from (C2) that given an $\epsilon > 0$, the maximum number of iterations where

$$\frac{|K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)|^2}{1 + |C(\hat{\phi}_t)|^2} > \epsilon$$

cannot exceed certain value for any length of window of time.

(C4) Based on the result from (C3), it can be proved that if x_t is outside a neighborhood of the origin there are only finite number of iterations between $1 \leq t < \infty$ where $\Delta W_t \geq 0$. In addition, even for these iterations, the maximum increase in W_t is bounded. Hence, the total amount of change in W_t is negative over a large enough interval. Thus, x_t eventually converges to the neighborhood of the origin. Based on this idea, we divide the state space into three sets:

$$\begin{aligned} S_1 &= \left\{ x_t \in \Omega_3 \mid |K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)| < k_1 |x_t| \right\} \\ S_2 &= \left\{ x_t \in \Omega_3 \mid |K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)| \geq k_1 |x_t|, \quad |x_t|^2 > \frac{k_2}{k_1 - k_2 c_6^2} \right\} \\ S_3 &= \left\{ x_t \in \Omega_3 \mid |K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)| \geq k_1 |x_t|, \quad |x_t|^2 \leq \frac{k_2}{k_1 - k_2 c_6^2} \right\} \end{aligned} \quad (4.26)$$

where $\Omega_3 = \mathcal{Y}_3^{n+d-1} \times \mathcal{U}_3^{m+d-1}$ and k_1 is chosen so as to guarantee that $\Delta W_t < 0$ if $x_t \in S_1$. Whereas, k_2 is selected so that if $|x_t|^2 > \frac{k_2}{k_1 - k_2 c_6^2}$, then the inequality $|K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)| \geq k_1 |x_t|$ implies $\frac{|K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)|}{1 + |C(\hat{\phi}_t)|^2} > k_2$, which ensures that x_t belongs to the set S_2 for only finite number of iterations between

$$1 \leq t < \infty.$$

(C5) The statement in the previous step is valid if signals entering the neural network do not leave the set where the network is trained. It is shown by induction that this is indeed true by choosing the initial parameter estimate close enough to the true value and small initial conditions for the state variables. Moreover, the larger the set over which the neural network is trained, the farther the initial estimate can be from the true parameter value and larger the initial conditions from the origin.

□

The proof is organized as follows. The upper bound on ΔW_t as described in (4.25) is derived first. The two properties of the parameter estimation algorithm described in (C2) are established in Proposition 4.1. Based on the results in (C2), (C3) is achieved using Propositions 4.2, 4.3 and 4.4. Upper bounds on the change of ΔW_t , in term of W_t , for x_t belonging to the sets S_1 , S_2 and S_3 defined in (4.26) are subsequently obtained in (4.35). Using these results, we prove by induction in Theorem 4.5 that all the signals entering TANN remain in the set where it is trained. Furthermore, the output converges to a neighborhood of the origin.

Evolution of W_t : W_t can be evaluated along trajectories of the closed-loop system as:

$$\begin{aligned}
\Delta W_t &= W_{t+1} - W_t \\
&= x_t^T (A_m^T P A_m - P) x_t + [0, K(\omega_t, 0, \theta)] B_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} \\
&\quad + 2x_t^T A_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} + 2(x_t^T A_m^T + [0, K(\omega_t, 0, \theta)] B_m^T) P B_m F_t \\
&\quad + F_t^T B_m^T P B_m F_t \\
&\leq -x_t^T Q x_t + 2(\sigma_{\max}(A_m^T P B_m) |x_t| + \sigma_{\max}(B_m^T P B_m) |K(\omega_t, 0, \theta)|) |F_t| \\
&\quad + \sigma_{\max}(B_m^T P B_m) |F_t|^2 \tag{4.27}
\end{aligned}$$

where assumption (A5) is used in obtaining the last inequality. It is worth noting that, if $\hat{\theta}_t = \theta$ for $t \geq t_0$, $y_{t+d} = 0$, $F_t = 0$ and hence,

$$\begin{aligned} \Delta W_t &= x_t^T (A_m^T P A_m - P) x_t + [0, K(\omega_t, 0, \theta)] B_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} \\ &\quad + 2x_t^T A_m^T P B_m \begin{bmatrix} 0 \\ K(\omega_t, 0, \theta) \end{bmatrix} \\ &\leq -x_t^T Q x_t. \end{aligned}$$

for $t \geq t_0$. Thus, (A5) implies that the unobservable state variables of x_t , $(u_{t-1}, \dots, u_{t-m-d+1})$, are stable and, furthermore, converge to the origin, as in the linear case, at a rate determined by Q .

Based on assumption (A3) that f_r and K are differentiable and have bounded derivatives, inequalities regarding f_r and K can be derived as follows using the mean-value theorem [40], where c_i represents a positive constant, $i = 1, \dots, 11$. First, since $K(0, 0, \theta) = 0$ from (A2), provided that $\omega_t \in \Omega_2$ and $\theta, \hat{\theta} \in \Theta_2$,

$$|K(\omega_t, 0, \theta)| = |K(\omega_t, 0, \theta) - K(0, 0, \theta)| \leq c_1 |\omega_t| \leq c_2 |x_t| \quad (4.28)$$

Moreover, by using the relation $f_r(0, K(0, 0, \hat{\theta}), \theta) = 0$, we have

$$\begin{aligned} |f_r(\omega, K(\omega, 0, \hat{\theta}), \theta)| &= |f_r(\omega, K(\omega, 0, \hat{\theta}), \theta) - f_r(0, K(0, 0, \hat{\theta}), \theta)| \\ &\leq c_3 |[\omega, K(\omega, 0, \hat{\theta}), \theta] - [0, K(0, 0, \hat{\theta}), \theta]| \\ &\leq c_4 |\omega| \end{aligned}$$

since $K(E_1)$ is convex which in turn follows from the facts that E_1 is convex and K is continuous on E_1 .

The function C , defined below, satisfies the following inequalities if $y_{t+d} \in \mathcal{Y}_2$,

$$|C(\bar{\phi}_t)| \triangleq \left| \frac{\partial K}{\partial \theta}(\omega_t, y_{t+d}, \theta_0) \right|$$

$$\begin{aligned}
&= \left| \frac{\partial K}{\partial \theta} (\omega_t, f_r (\omega_t, K (\omega_t, 0, \hat{\theta}_t), \theta), \theta_0) - \frac{\partial K}{\partial \theta} (0, 0, \theta_0) \right| \\
&\leq c_5 \left| [\omega_t^T, f_r (\omega_t, K (\omega_t, 0, \hat{\theta}_t), \theta)]^T \right| \\
&\leq c_6 |x_t|
\end{aligned} \tag{4.29}$$

Above, we have used the fact that $\frac{\partial K}{\partial \theta}(0, 0, \theta_0) = 0$ since $K(0, 0, \theta) = 0$ for every $\theta \in \Theta_2$. The first inequality in (4.29) is obtained from the assumption that K is twice differentiable and have bounded second derivatives on $\Omega_2 \times \mathcal{Y}_2 \times \Theta_2$.

Furthermore, since $f_r(\omega, K(\omega, 0, \theta), \theta) = 0$, and $K(E_1)$ is convex,

$$|f_r(\omega, K(\omega, 0, \hat{\theta}), \theta)| \leq c_7 |K(\omega, 0, \hat{\theta}) - K(\omega, 0, \theta)| \tag{4.30}$$

Hence,

$$\begin{aligned}
|F_t| &\leq c_7 |K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)| + |K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)| \\
&= c_8 |K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)|
\end{aligned}$$

The inequality in (4.27) can thus be written as

$$\begin{aligned}
\Delta W_t &\leq -\lambda_{\min}(Q)|x_t|^2 + c_9 |x_t| \left| K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta) \right| \\
&\quad + c_{10} \left| K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta) \right|^2
\end{aligned} \tag{4.31}$$

$$\leq \lambda_1 W_t \tag{4.32}$$

where $c_9 = 2c_8 (\sigma_{\max}(A_m^T P B_m) + c_2 \sigma_{\max}(B_m^T P B_m))$, $c_{10} = c_8^2 \sigma_{\max}(B_m^T P B_m)$ and $\lambda_1 = -\frac{\lambda_{\min}(Q)}{\lambda_{\max}(P)} + \frac{2c_2 c_9 + 4c_2^2 c_{10}}{\lambda_{\min}(P)}$. The inequality $|K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)| \leq 2c_2 |x_t|$ obtained from (4.28) and the triangular inequality are used to obtain (4.32).

By applying the mean-value theorem again, we can obtain

$$\begin{aligned}
y_{t+d} &= f_r(\omega_t, K(\omega_t, 0, \hat{\theta}_t), \theta) \\
&= f_r(\omega_t, K(\omega_t, 0, \hat{\theta}_t), \theta) - f_r(\omega_t, K(\omega_t, 0, \theta), \theta)
\end{aligned}$$

$$= \frac{\partial f_r(\omega_t, u, \theta)}{\partial u} \Big|_{u=u_1} \cdot (K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta))$$

where $u_1 \in K(\Omega_2, 0, \Theta_2)$ since $K(\Omega_2, 0, \Theta_2)$ is convex. Hence,

$$\begin{aligned} & K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t) \\ = & \left(K(\omega_t, 0, \theta) - K(\omega_t, 0, \hat{\theta}_t) \right) + \left(\frac{\partial K(\omega_t, y, \theta)}{\partial y} - \frac{\partial K(\omega_t, y, \hat{\theta}_t)}{\partial y} \right) \Big|_{y=y_1} \cdot y_{t+d} \\ = & \left\{ 1 - \left(\frac{\partial K(\omega_t, y, \theta)}{\partial y} - \frac{\partial K(\omega_t, y, \hat{\theta}_t)}{\partial y} \right) \Big|_{y=y_1} \cdot \frac{\partial f_r(\omega_t, u, \theta)}{\partial u} \Big|_{u=u_1} \right\} \\ & \cdot (K(\omega_t, 0, \theta) - K(\omega_t, 0, \hat{\theta}_t)) \quad (4.33) \end{aligned}$$

where $y_1 \in f_r(\Omega_2, K(\Omega_2, 0, \Theta_2), \Theta_2)$ since \mathcal{Y}_2 is convex and contains the origin.

We recall from Eq. (4.19) that ϵ_1 is the upper bound of $\Delta V - \Delta V_{d_t}$ after network training and ϵ is the width of the dead-zone in the parameter estimation algorithm. With ϵ chosen to be larger than ϵ_1 and provided that the signals entering the network remain in the set where it is trained, certain properties of the parameter estimation algorithm can be established, as is done in the following proposition:

Proposition 4.1 *For the system in Eqs. (4.1), (4.8) and (4.19) satisfying properties (P1) and (P2), if $\epsilon_1 < \epsilon$, $\theta \in \Theta_3$, $y_i \in \mathcal{Y}_3$ for $0 \leq i \leq t$, $u_i \in \mathcal{U}_3$ for $0 \leq i \leq t-d$ and $\hat{\theta}_i \in \Theta_3$ for $n+d-2 \leq i \leq t-1$, then the following statements hold:*

(i) $|\tilde{\theta}_t| \leq |\tilde{\theta}_{t-1}| \leq \dots \leq |\tilde{\theta}_{n+d-2}|$

(ii) *The set $T_g \triangleq \{i \in \mathcal{T} | \Delta V_{d_i} < -\epsilon\}$ cannot have more than n_1 elements.*

where n_1 is the largest integer that is less than or equal to $\frac{|\tilde{\theta}_{n+d-2}|^2}{\epsilon - \epsilon_1}$ and $\mathcal{T} = \{n+d-1, \dots, t-1, t\}$.

Proof: Based on the assumptions in Proposition 4.1, we have $\omega_{i-d} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $u_{i-d} \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_{i-d}, \theta}(\mathcal{Y}_3)$ and $\hat{\theta}_{i-1} \in \Theta_3$ for $n+d-1 \leq i \leq t$. According to (4.19) and the property (P2), $\Delta V_i = 0$ if $\Delta V_{d_i} \geq -\epsilon$. Otherwise, if $\Delta V_{d_i} < -\epsilon$,

$$\Delta V_i < \Delta V_{d_i} + \epsilon_1 < 0$$

since $\epsilon > \epsilon_1$. Hence, (i) follows.

We also have the following inequality:

$$0 \leq \left| \sum_{i=n+d-1}^t \Delta V_i \right| = \left| |\tilde{\theta}_t|^2 - |\tilde{\theta}_{n+d-2}|^2 \right| \leq |\tilde{\theta}_{n+d-2}|^2$$

Since $\Delta V_i < -(\epsilon - \epsilon_1)$ and the set $\{i \in \mathcal{T} | \Delta V_i < -(\epsilon - \epsilon_1)\}$ cannot have more than $\frac{|\tilde{\theta}_0|^2}{\epsilon - \epsilon_1}$ elements, (ii) can be established. \blacksquare

From the definition of ΔV_{d_t} in Eq. (4.14), we can obtain

$$|\Delta V_{d_{t+d}}| > a_1 \frac{|K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1}) - K(\omega_t, y_{t+d}, \theta)|^2}{1 + |C(\bar{\phi}_t)|^2}$$

Comparing the numerator of the above inequality with (4.31) and (4.33), in order to make use of the property of ΔV_{d_t} developed in Proposition 4.1, we need to first relate $|K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1}) - K(\omega_t, y_{t+d}, \theta)|$ to $|K(\omega_t, y_{t+d}, \hat{\theta}_t) - K(\omega_t, y_{t+d}, \theta)|$. This issue is addressed in the following proposition, where the requirement of limiting the magnitude of parameter correction in (P1) comes in.

Proposition 4.2 *There exists a continuous function $h : \mathfrak{R} \rightarrow \mathfrak{R}$ satisfying $\lim_{\delta \rightarrow 0} h(\delta) = 0$ such that for every $\delta > 0$, if*

$$\frac{|K(\omega_{i-d+1}, y_{i+1}, \theta) - K(\omega_{i-d+1}, y_{i+1}, \hat{\theta}_i)|^2}{1 + |C(\bar{\phi}_{i-d+1})|^2} \leq \frac{\delta}{a_1}, \quad \text{for } i = t, t+1, \dots, t+d-1$$

where $\omega_{i-d+1} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $\theta, \hat{\theta}_i \in \Theta_3$ and $y_{i+1} \in \mathcal{Y}_3$ for $t \leq i \leq t+d-1$, and $\hat{\theta}_i$ is adjusted using (4.19), then

$$\frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} \leq h(\delta)$$

Proof: Since $y_{i+1} \in \mathcal{Y}_3$, $u_{i-d+1} \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_{i-d+1}, \theta}(\mathcal{Y}_3)$, $\omega_{i-d+1} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$ and $\theta, \hat{\theta}_i \in \Theta_3$ for $i = t, \dots, t+d-2$, from Eq. (4.19) and the property (P1), we have, for $i = t, \dots, t+d-2$,

$$|\tilde{\theta}_{i+1} - \tilde{\theta}_i|^2 = |N(y_{i+1}, \omega_{i-d+1}, u_{i-d+1}, \hat{\theta}_i)|^2$$

$$\begin{aligned}
&\leq a_2 \frac{|C(\bar{\phi}_{i-d+1})|^2}{(1 + |C(\bar{\phi}_{i-d+1})|^2)^2} \tilde{u}_{i-d+1}^2 \\
&\leq \frac{a_2}{a_1} \delta.
\end{aligned}$$

Hence,

$$|\tilde{\theta}_{t+d-1} - \tilde{\theta}_t| < (d-1) \sqrt{\frac{a_2}{a_1} \delta}$$

Furthermore, since $\omega_t \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $y_{t+d} \in \mathcal{Y}_3$, $\hat{\theta}_t, \hat{\theta}_{t+d-1} \in \Theta_3$, and K is differentiable and has a bounded derivative on E_1 ,

$$|K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1}) - K(\omega_t, y_{t+d}, \hat{\theta}_t)| \leq c_{11} |\hat{\theta}_{t+d-1} - \hat{\theta}_t| < c_{11} (d-1) \sqrt{\frac{a_2}{a_1} \delta}$$

Therefore,

$$\begin{aligned}
&\frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|}{(1 + |C(\bar{\phi}_t)|^2)^{\frac{1}{2}}} \\
&\leq \frac{|K(\omega_t, y_{t+d}, \hat{\theta}_t) - K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1})|}{(1 + |C(\bar{\phi}_t)|^2)^{\frac{1}{2}}} + \frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1})|}{(1 + |C(\bar{\phi}_t)|^2)^{\frac{1}{2}}} \\
&< c_{11} (d-1) \sqrt{\frac{a_2}{a_1} \delta} + \sqrt{\frac{\delta}{a_1}}
\end{aligned}$$

Hence, $h(\delta)$ can be chosen as $h(\delta) = \left(1 + c_{11}(d-1)\sqrt{a_2}\right)^2 \frac{\delta}{a_1}$. ■

Using Proposition 4.2, we prove in Proposition 4.3 that given $\epsilon > 0$ there are only finite number of sequences when $\frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} > \epsilon$. Given $T \in \mathcal{Z}^+$, $\epsilon > 0$, and $\bar{T} = \{n+d-2, n+d-1, \dots, T\}$, we partition \bar{T} into T_1 and T_2 , where

$$T_1 \triangleq \left\{ t \in \bar{T} \mid \frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} \leq h(\epsilon) \right\}, \quad T_2 \triangleq \bar{T} - T_1 \quad (4.34)$$

Proposition 4.3 states that the set T_2 has at most certain number of elements no matter how large T is.

Proposition 4.3 *For any $T \in \mathcal{Z}^+$, $\theta \in \Theta_3$ and $\epsilon > 0$, if $\epsilon_1 < \epsilon$, and $\omega_{t-d+1} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $y_{t+1} \in \mathcal{Y}_3$ and $\hat{\theta}_t \in \Theta_3$ for $t = n+d-2, \dots, T+d-1$, then T_2 cannot*

have more than $2n_1d$ elements.

Proof: We first partition \bar{T} into segments of d elements as follows:

$$T_{d_k} \triangleq \{(n+d-2)+(k-1)d, (n+d-2)+(k-1)d+1, \dots, \min((n+d-2)+kd-1, T)\}$$

where $1 \leq k \leq n_2$, $k \in \mathcal{Z}$ and n_2 is the largest integer satisfying $(n_2+d-2)+kd-1 \leq T$. Define

$$\begin{aligned} T' &= \{T_{d_i} | i \in \mathcal{Z}^+, 1 \leq i \leq n_2\} \\ T'_1 &= \left\{ T_{d_i} \left| \frac{|K(\omega_{t-d+1}, y_{t+1}, \theta) - K(\omega_{t-d+1}, y_{t+1}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_{t-d+1})|^2} \leq \frac{\epsilon}{a_1}, \forall t \in T_{d_i} \right. \right\} \\ T'_2 &= T' - T'_1 \end{aligned}$$

Since $a_1 \frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1})|^2}{1 + |C(\bar{\phi}_t)|^2} < |\Delta V_{d_{t+d}}|$, we can obtain from Proposition 4.1 (ii) that T'_2 has at most n_1 elements. By utilizing Proposition 4.2, it can be concluded that if $T_{d_i} \in T'_1$ and $T_{d_{i+1}} \in T'_1$ then

$$\frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} \leq h(\epsilon), \quad \forall t \in T_{d_i}$$

Hence, T_2 cannot have more than $2n_1d$ elements. ■

We define a set T_3

$$T_3 \triangleq \left\{ t \in \bar{T} \left| \frac{|K(\omega_t, 0, \theta) - K(\omega_t, 0, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} > \frac{h(\epsilon)}{\delta_g} \right. \right\}$$

Based on Proposition 4.3 and assumption (A4), a property satisfied by the set T_3 is derived in Proposition 4.4.

Proposition 4.4 *For any $T \in \mathcal{Z}^+$, $\theta \in \Theta_3$ and $\epsilon > 0$, if $\epsilon_1 < \epsilon$, and $\omega_{t-d+1} \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $y_{t+1} \in \mathcal{Y}_3$ and $\hat{\theta}_t \in \Theta_3$ for $t = n+d-2, \dots, T+d-1$, then T_3 cannot have more than $2n_1d$ elements.*

Proof: From Eq. (4.33) and the assumption (A4),

$$\begin{aligned} & |K(\omega_t, y_{t+d}, \hat{\theta}_t) - K(\omega_t, y_{t+d}, \theta)| \\ & > \delta_g |K(\omega_t, 0, \hat{\theta}_t) - K(\omega_t, 0, \theta)| \end{aligned}$$

Hence, if $t \in T_3$

$$\begin{aligned} & \frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} \\ & > \delta_g \frac{|K(\omega_t, 0, \theta) - K(\omega_t, 0, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} \\ & > h(\epsilon) \end{aligned}$$

implying $t \in T_2$. Thus, T_3 cannot have more than $2n_1d$ elements. ■

To study the evolution of W_t , we separate the set $\mathcal{Y}_3^{n+d-1} \times \mathcal{U}_3^{m+d-1}$ into the three subsets, S_1 , S_2 and S_3 as defined in (4.26), for given $\theta, \hat{\theta}_t \in \Theta_3$, where $k_1 = \min\left(\frac{1}{4c_9} \lambda_{\min}(Q), \left(\frac{1}{4c_{10}} \lambda_{\min}(Q)\right)^{\frac{1}{2}}\right)$ and $0 < k_2 < \frac{k_1}{c_6^2}$. The value k_1 is chosen such that if $x_t \in S_1$ then $\Delta W_t < 0$, as can be verified from (4.31).

From (4.31) and (4.32), we have

$$\begin{aligned} \Delta W_t & < \lambda_1 W_t & \text{if } x_t \in S_2 \cup S_3 \\ \Delta W_t & < -\lambda_2 W_t & \text{if } x_t \in S_1 \end{aligned} \tag{4.35}$$

where $\lambda_2 = \frac{\lambda_{\min}(Q)}{2\lambda_{\max}(P)} < 1$. Furthermore, from (4.29), if $x_t \in S_2$, then

$$\begin{aligned} \frac{|K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)|}{1 + |C(\bar{\phi}_t)|^2} & \geq \frac{|K(w_t, 0, \hat{\theta}_t) - K(w_t, 0, \theta)|}{1 + c_6^2 |x_t|^2} \\ & \geq \frac{k_1 |x_t|^2}{1 + c_6^2 |x_t|^2} \\ & > k_2. \end{aligned}$$

Under the conditions of Proposition 4.4, given $\epsilon > 0$ and $\epsilon_1 < \epsilon$, if $k_2 \geq \frac{h(\epsilon)}{\delta_g}$, then for any $T \in \mathcal{Z}^+$, there are at most $2n_1d$ elements in T_3 . In other words, for any period of time, x_t belongs to S_2 for at most $2n_1d$ times if the conditions of Proposition 4.4 is

satisfied. Therefore, using (4.35) and the definitions of S_1 , S_2 and S_3 , we can conclude that for any $T \in \mathcal{Z}^+$,

$$\begin{aligned} W_T &< \max \left(\frac{\lambda_{\max}(P)k_2}{k_1 - k_2c_6^2} (1 + \lambda_1)^{2n_1d+2} (1 - \lambda_2)^{\max(0, T-2n_1d-2)}, \right. \\ &\quad \left. (1 + \lambda_1)^{2n_1d+1} (1 - \lambda_2)^{\max(0, T-2n_1d-1)} W_{t_0}, \frac{k_2}{k_1 - k_2c_6^2} \right) \\ &\leq \max \left(\frac{\lambda_{\max}(P)k_2}{k_1 - k_2c_6^2} (1 + \lambda_1)^{2n_1d+2}, (1 + \lambda_1)^{2n_1d+1} W_{t_0}, \frac{k_2}{k_1 - k_2c_6^2} \right) \end{aligned} \quad (4.36)$$

provided that $y_i \in \mathcal{Y}_3$ for $0 \leq i \leq T + d - 2$, and $u_i \in \mathcal{U}_3 \cap f^{-1}|_{\omega_t=\omega_i, \theta=\theta}(\mathcal{Y}_3)$ for $0 \leq i \leq T - 2$, and $\hat{\theta}_i \in \Theta_3$ for $n + d - 2 \leq i \leq T + d - 2$.

Based on the above results, the stability property of the closed-loop system in Eqs. (4.1), (4.8) and (4.19) can be concluded in the following theorem:

Theorem 4.5 *For the closed-loop adaptive system given by (4.1), (4.8) and (4.19), under assumptions (A1)-(A5), given the compact sets $\mathcal{Y}_3^{n+1} \times \mathcal{U}_3^{m+d} \times \Theta_3$ where the neural network in Eq. (4.19) is trained so that properties (P1) and (P2) hold, there exist $\epsilon_1, \epsilon > 0$ such that the following statement is true: For any interior point θ of Θ_3 , there exist open sets $\mathcal{Y}_4 \subset \mathcal{Y}_3$, $\mathcal{U}_4 \subset \mathcal{U}_3$ and a neighborhood Θ_4 of θ such that if $y_0, \dots, y_{n+d-2} \in \mathcal{Y}_4$, $u_0, \dots, u_{n-2} \in \mathcal{U}_4$, and $\hat{\theta}_{n-1}, \dots, \hat{\theta}_{n+d-2} \in \Theta_4$, then all the signals in the closed-loop system remain bounded and y_t converges to a neighborhood of the origin.*

Proof: Since the compact sets \mathcal{Y}_3 and \mathcal{U}_3 contain neighborhoods of the origin, we can find a $\delta > 0$ such that the set $V_1 = \{x \in \mathbb{R}^{n+m+2d-2} \mid |x| = \delta\}$ is a subset of the product space $\mathcal{Y}_3^{n+d-1} \times \mathcal{U}_3^{m+d-1}$. Since V_1 is compact and W_t is a continuous, positive definite function of x_t , $\alpha = \min_{x \in V_1} W(x)$ exists and $\alpha > 0$ where $W(x) = x^T P x$. Hence, the set $V_2 = \{x \in \mathcal{Y}_3^{n+d-1} \times \mathcal{U}_3^{m+d-1} \mid W(x) < \alpha\}$ is open and nonempty, and contains the origin. Choose $\epsilon_1 < \epsilon_1^*$ where

$$h(\epsilon_1^*) \triangleq \delta_g \min \left(\frac{\alpha k_1}{\alpha c_6^2 + \lambda_{\max}(P)(1 + \lambda_1)^2}, \frac{\alpha k_1}{1 + \alpha c_6^2}, \frac{k_1}{c_6^2} \right). \quad (4.37)$$

Hence, there exist an integer $n_0 \geq 0$ and $\epsilon > \epsilon_1$ such that

$$h(\epsilon) < \delta_g \min \left(\frac{\alpha k_1}{\alpha c_6^2 + \lambda_{\max}(P)(1 + \lambda_1)^{2n_0 d + 2}}, \frac{\alpha k_1}{1 + \alpha c_6^2}, \frac{k_1}{c_6^2} \right)$$

For the particular n_0 and ϵ , k_2 is then chosen so that

$$\frac{h(\epsilon)}{\delta_g} < k_2 < \min \left(\frac{\alpha k_1}{\alpha c_6^2 + \lambda_{\max}(P)(1 + \lambda_1)^{2n_0 d + 2}}, \frac{\alpha k_1}{1 + \alpha c_6^2}, \frac{k_1}{c_6^2} \right) \quad (4.38)$$

where the first term in the bracket of $\min(\cdot)$ is chosen such that for any non-negative integer $n_1 \leq n_0$, the first term in the bracket of (4.36) is less than α , whereas the second term in $\min(\cdot)$ implies that the third term in the bracket of (4.36) is less than α .

Since θ is an interior point of Θ_3 , we can find a neighborhood $\Theta_4 \subset \Theta_3$ of θ such that for every $\hat{\theta} \in \Theta_4$

$$|\hat{\theta} - \theta| < \max(1, n_0)(\epsilon - \epsilon_1)$$

Therefore, if the initial estimate $\hat{\theta}_{n-1} \in \Theta_4$, then $n_1 \leq n_0$ since $n_1 \in Z^+ \cup \{0\}$ and $n_1 \leq \frac{|\hat{\theta}_{n-1} - \theta|^2}{\epsilon - \epsilon_1}$. Consider the open set $V_3 = \{x \in V_2 \mid W(x) < \frac{\alpha}{(1 + \lambda_1)^{2n_1 d + 1}}\}$. We can find open sets $\mathcal{Y}_4 \subset \mathcal{Y}_3$ and $\mathcal{U}_4 \subset \mathcal{U}_3$ containing the origin such that $\mathcal{Y}_4^{n+d-1} \times \mathcal{U}_4^{m+d-1} \subset V_3$. For the open sets \mathcal{Y}_4 , \mathcal{U}_4 and Θ_4 constructed above, we prove below by induction that if the initial estimate $\hat{\theta}_t \in \Theta_4$ for $t_0 \leq t \leq t_0 + d - 1$, $y_t \in \mathcal{Y}_4$ for $0 \leq t \leq t_0 + d - 1$ and $u_t \in \mathcal{U}_4$ for $0 \leq t \leq t_0 - 1$, where $t_0 = n - 1$, then $y_t \in \mathcal{Y}_3$, $u_t \in \mathcal{U}_3$ and $\hat{\theta}_{t+t_0} \in \Theta_3$ for every $t \geq 0$.

At $t = t_0$, since $y_i \in \mathcal{Y}_4$ for $0 \leq i \leq t_0 + d - 1$, $u_i \in \mathcal{U}_4$ for $0 \leq i \leq t_0 - 1$ and $\hat{\theta}_{t_0} \in \Theta_4$, based on (4.32) $W_{t_0+1} < \frac{\alpha}{(1 + \lambda_1)^{2n_1 d}}$, which implies $x_{t_0+1} \in \mathcal{Y}_3^{n+d-1} \times \mathcal{U}_3^{m+d-1}$ and thus $y_{t_0+d} \in \mathcal{Y}_3$ and $u_{t_0} \in \mathcal{U}_3$. In addition, $u_{t_0} \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_{t_0}, \theta}(\mathcal{Y}_3)$, since $f_r(\omega_{t_0}, u_{t_0}, \theta) = y_{t_0+d} \in \mathcal{Y}_3$. Hence, $\hat{\theta}_{t_0+d} \in \Theta_4$ according to Proposition 4.1(i).

At $t = T > t_0$, if $y_i \in \mathcal{Y}_3$ for $0 \leq i \leq T + d - 1$, $u_i \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_i, \theta}(\mathcal{Y}_3)$ for $0 \leq i \leq T - 1$ and $\hat{\theta}_i \in \Theta_4$ for $t_0 \leq i \leq T + d - 1$, then according to (4.36),

$$W_{T+1} < \max \left(\frac{\lambda_{\max}(P)k_2}{k_1 - k_2 c_6^2} (1 + \lambda_1)^{2n_1 d + 2}, \alpha, \frac{k_2}{k_1 - k_2 c_6^2} \right)$$

$$= \alpha$$

due to the choices of k_2 in (4.38) and $\max(1, n_0) > \frac{|\hat{\theta}_{n+d-2}-\theta|^2}{\epsilon-\epsilon_1} \geq n_1$. It implies $y_{T+d} \in \mathcal{Y}_3$ and $u_T \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_T, \theta}(\mathcal{Y}_3)$. Hence, $\hat{\theta}_{T+d} \in \Theta_4$ according to Proposition 4.1(i). Therefore, we can conclude by induction that $y_{t+d} \in \mathcal{Y}_3$, $u_t \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_t, \theta}(\mathcal{Y}_3)$ and $\hat{\theta}_{t+d} \in \Theta_4$ for every $t \geq t_0$. All the signals in the closed-loop system are thus bounded for all time.

Since $\Delta V_t \leq 0$ and $\sum_{t=t_0}^{\infty} \Delta V_t$ is bounded below according to Proposition 4.1, $\lim_{t \rightarrow \infty} \Delta V_t = 0$ and, thus, there exists a $t_1 > 0$ such that $|\Delta V_{d_t}| \leq \epsilon$ for $t \geq t_1$. Hence,

$$a_1 \frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_{t+d-1})|^2}{1 + |C(\bar{\phi}_t)|^2} < \epsilon, \quad \forall t \geq t_1$$

It follows from the above inequality and Proposition 4.2 that

$$\frac{|K(\omega_t, y_{t+d}, \theta) - K(\omega_t, y_{t+d}, \hat{\theta}_t)|^2}{1 + |C(\bar{\phi}_t)|^2} < h(\epsilon), \quad \forall t \geq t_1 + d - 1 \quad (4.39)$$

From the inequality (4.29) and the boundedness of $|x_t|$ proved earlier, we can establish:

$$|C(\bar{\phi}_t)|^2 < \frac{c_6^2 \alpha}{\lambda_{\min}(P)} \quad (4.40)$$

Furthermore, since $u_t = K(\omega_t, y_{t+d}, \theta) = K(\omega_t, 0, \hat{\theta}_t)$, by combining (4.39), (4.40) and Assumption (A4), it can be established that

$$|y_{t+d}| < \frac{1}{\delta_h} \left[h(\epsilon) \left(1 + \frac{c_6^2 \alpha}{\lambda_{\min}(P)} \right) \right]^{\frac{1}{2}}, \quad \forall t \geq t_1 + d - 1 \quad (4.41)$$

Therefore, we conclude that as $t \rightarrow \infty$, y_t converges to a neighborhood around the origin. ■

4.5.1 Discussions

The proof of Theorem 4.5 implies that closed-loop stability follows if properties (P1) and (P2) are satisfied by the TANN, with $\epsilon < \epsilon_1^*$, where ϵ_1^* is given by (4.37). The

question to be answered then is whether such an ϵ_1^* can be determined for a given f_r and K . The constants c_i , k_1 and λ_1^* which appear in the right-hand side of (4.37) depend on smoothness of the system model, and can be estimated. Hence, the left-hand side of (4.37) can be calculated from the definition of $h(\cdot)$ in Proposition 4.2 for a particular ϵ_1 , while the right-hand side of the inequality can be determined once the set V_1 is chosen. Therefore, whether or not the network training is successful enough to result in a stable adaptation algorithm can be checked explicitly before implementation. It is in this regard that the proposed approach in this paper is more tangible and verifiable.

Since the proof of stability follows if a continuous function N exists such that properties (P1) and (P2) with $\epsilon_1 < \epsilon_1^*$ are satisfied, whether or not such a continuous function exists needs to be guaranteed. To answer this question, let us consider the projection algorithm obtained by linearizing Eq. (4.6) with respect to θ . For this algorithm, it can be easily checked that (P1) is automatically satisfied. If V_1 is chosen, the right-hand side of (4.37) is fixed. Hence, we can always find a Θ_3 such that the projection algorithm can satisfy (P2) on Θ_3 for the required ϵ_1 in (4.37) based on the linearization principle. Therefore, we can conclude that such a continuous function does exist.

Since the continuous function exists, the existence of networks in the chosen class such as MNN or RBF which can approximate the continuous function to a desired degree of accuracy so that (P1) and (P2) continue to hold can be ensured from the universal approximator property described in Definition 2.1. Such networks can be constructed using the training procedure suggested in Section 4.4 and by increasing the complexity of the network structure until $\epsilon_1 < \epsilon_1^*$. It is noted that it is not necessary for the network to solve the global minimum of J in (4.21). Any local minimum that satisfies $\epsilon_1 < \epsilon_1^*$ suffices as well for stability.

The region Θ_3 for which the projection algorithm guarantees (P1) and (P2) may be very small. However, since our approach does not explicitly use the projection algorithm but attempts to construct a continuous function that satisfies (P1) and (P2), the applicable region of parameters and initial conditions can be made larger as long

as such functions exist. The same kind of comparison can be made between our neural controller and the extended Kalman filter algorithm which uses the gradient of f_r with respect to the most recent parameter estimate to determine the update (see Section 6.3 for more details). For practical purposes, in order to ensure that the network training can be accomplished, the training set should initially include patterns sampled from a smaller Θ_3 , and could be gradually enlarged as training is successful.

Even though Theorem 4.5 is stated in terms of the neural network parameter estimation algorithm, the result in the theorem can be applied to much broader adaptive control schemes. It is noted that the only requirements for the parameter adaptation scheme in Theorem 4.5 are the properties (P1) and (P2) discussed in Section 4.3. They are accomplished in this paper using a neural network. Nonetheless, other algorithms can be used as well, as long as the requirements in (P1) and (P2) are met, and stability of the closed-loop system is assured from Theorem 4.5.

We provide the following comments on some of technical aspects of the proof of Theorem 4.5:

1. The proof was established by showing that over any arbitrary interval, W_t , the norm of the system state, can grow at most over a finite period and decays almost everywhere else. This is similar to the arguments used in [30, 41].
2. In order to ensure that all the signals entering the TANN controller are within the region where it is trained, it is shown that given a neighborhood of the origin, an upper bound on the output can be estimated for *all* initial conditions starting in the neighborhood. This is a stronger result than most of the proofs of the adaptive systems, where only global boundedness for each initial condition is established.
3. Eq. (4.41) defines an upper bound on the steady state value of the output, which, among others, depends on the size of the dead-zone, ϵ , in the TANN algorithm. The smallness of ϵ in turn is limited by ϵ_1 , which is an upper bound of $\Delta V_t - \Delta V_{d_t}$, as well as the size of V_3 . If $\epsilon_1 = 0$, ϵ can be chosen to be zero and,

thus, there is no steady state error. Nevertheless, it does not imply that if ϵ is positive, there will be a non-zero output error during implementation. Thus, in general, the final output error varies with the initial parameter estimate as well as the initial state of the system. Depending upon where the control parameters converge to when the adaptation stops, the output error may converge to zero. In the simulation studies reported in Section 6.3, it can be seen that the output indeed converges to zero since in steady state the control parameter corresponded to an asymptotically stable closed-loop system.

4. In the above proof, the sets \mathcal{Y}_4 and \mathcal{U}_4 where the initial conditions of the system should lie in order to ensure stability is determined by the set V_3 . The larger the set V_3 , the larger the sets \mathcal{Y}_4 and \mathcal{U}_4 . However, the size of V_3 is limited, among others, by the constants α and n_1 . The implications of the values of the two constants on the network training and initial parameter estimation error are explained below:
 - (a) The value α is decided by the compact set $\mathcal{Y}_3^{n+1} \times \mathcal{U}_3^{m+d} \times \Theta_3$ where the network is trained successfully. If the compact set is large, so is α as well as the allowable initial conditions.
 - (b) Since n_1 is the largest integer that is less than or equal to $\frac{|\hat{\theta}_{n-1} - \theta|^2}{\epsilon - \epsilon_1}$, if the initial parameter estimation error is large, n_1 becomes large, which in turn results in a small V_3 . Hence, there is a direct tradeoff between allowable initial parameter estimation error and allowable initial conditions of the system for a given TANN controller.
5. When we established the inequality in (4.36), $y_i \in \mathcal{Y}_3$ is required only for $0 \leq i \leq T + d - 2$ instead of for $0 \leq i \leq T + d - 1$. The reason is that $\Delta W_{T-1} < \lambda_1 W_{T-1}$ is valid as long as, among others, $\omega_{T-1} \in \Omega_2$. The same is true that $u_i \in \mathcal{U}_3$ for $0 \leq i \leq T - 2$ rather than for $0 \leq i \leq T - 1$.

4.6 Complexity Reduction for Partially Linear Systems

For the control algorithm developed in Section 4.3, all the unknown parameters in the system are estimated using a TANN. However, in many problems, only some of the unknown parameters occur nonlinearly while others occur linearly. It is therefore desirable to exploit the linear structure to reduce complexity of the neural networks. In this section, instead of the model in Eq. (4.6), we consider a subclass of (4.6) in the following form:

$$u_t = K(\phi_{N_t}, \theta_N) + \phi_{L_t}^T \theta_L \quad (4.42)$$

where ϕ_{N_t} and ϕ_{L_t} are vectors of measurable signals up to time t and, possibly, y_{t+d} ; θ_N denotes unknown parameters which enter the system nonlinearly and θ_L linearly. Comparing Eq. (4.42) with Eq. (4.6), we can see that ϕ_{N_t} and ϕ_{L_t} together comprises $[\omega_t, y_{t+d}]$ while $[\theta_N, \theta_L]$ is equivalent to θ . In [59], a parameter estimation method, which utilizes the projection algorithm [18] to identify parameters occurring linearly and the TANN to estimate nonlinearly occurring parameters, deals specifically with this class of parameter estimation problems. In this section, the training criteria of the network in [59] is modified to accommodate the properties (P1) and (P2) needed in the control problem.

As proposed in [59], the TANN algorithm of the following form is used to update $\hat{\theta}_{N_t}$:

$$\hat{\theta}_{N_t} = \hat{\theta}_{N_{t-1}} + \nu k_{t-d} N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) \quad (4.43)$$

The linear parameter estimate $\hat{\theta}_{L_t}$ is adjusted based on the projection algorithm:

$$\hat{\theta}_{L_t} = \hat{\theta}_{L_{t-1}} + \nu k_{t-d} \phi_{L_{t-d}} \quad (4.44)$$

where

$$k_t = \frac{\alpha \tilde{u}_t}{1 + C_{N_t}^T C_{N_t} + \phi_{L_t}^T \phi_{L_t}}, \quad \alpha \in (0, 1)$$

$$\begin{aligned}\nu &= \begin{cases} 1 & \text{if } \Delta V_{d_t} < -\epsilon \\ 0 & \text{otherwise} \end{cases} \\ C_{N_t} &= \left. \frac{\partial K(\phi_{N_t}, \theta_N)}{\partial \theta_N} \right|_{\theta_{N_0}} \\ \tilde{u}_t &= u_t - \left(K(\phi_{N_t}, \hat{\theta}_{N_{t+d-1}}) + \phi_{L_t}^T \hat{\theta}_{L_{t+d-1}} \right)\end{aligned}$$

In order to satisfy (P1) and (P2), the network N in Eq. (4.43) is subjected to the following constraints:

$$(PL1) \quad |N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}})| \leq |C_{N_{t-d}}|$$

$$(PL2) \quad N^T(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) \tilde{\theta}_{N_{t-1}} - \left(K(\phi_{N_t}, \hat{\theta}_{N_{t+d-1}}) - K(\phi_{N_t}, \theta_N) \right) \leq \epsilon_2, \quad \epsilon_2 > 0.$$

for every $\omega_{t-d}^T \in \mathcal{Y}_3^n \times \mathcal{U}_3^{m+d-1}$, $\theta \in \Theta_3$ and $\hat{\theta}_{t-1}^T \in \Theta_3$ and for which $u_{t-d} \in \mathcal{U}_3 \cap f_r^{-1}|_{\omega_{t-d}, \theta}(\mathcal{Y}_3)$ where $\tilde{\theta}_{N_t} = \hat{\theta}_{N_t} - \theta_N$. From (PL1) we can derive that, if $\nu = 1$,

$$\begin{aligned}|\Delta \hat{\theta}_{N_t}|^2 + |\Delta \hat{\theta}_{L_t}|^2 &= k_{t-d}^2 \left(|N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}})|^2 + |\phi_{L_{t-d}}|^2 \right) \\ &\leq k_{t-d}^2 \left(|C_{N_{t-d}}|^2 + |\phi_{L_{t-d}}|^2 \right) \\ &= \frac{\alpha^2 \left(C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}} \right)}{\left(1 + C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}} \right)^2} \tilde{u}_{t-d}^2\end{aligned}$$

This in turn implies that (P1) is satisfied since $C(\bar{\phi}_t) = [C_{N_t}, \phi_{L_t}^T]^T$. Moreover, if (PL1) and (PL2) are satisfied and there exists a ϵ_3 such that $|2k_t \epsilon_2| \leq \epsilon_3$, then, if $\nu = 1$,

$$\begin{aligned}\Delta V_{N_t} &\triangleq \tilde{\theta}_{N_t}^T \tilde{\theta}_{N_t} - \tilde{\theta}_{N_{t-1}}^T \tilde{\theta}_{N_{t-1}} \\ &\leq 2k_{t-d} \left(K(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) - K(\phi_{N_{t-d}}, \theta_N) + \epsilon_2 \right) \\ &\quad + k_{t-d}^2 N^T(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) \\ &\leq 2k_{t-d} \left(K(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) - K(\phi_{N_{t-d}}, \theta_N) \right) \\ &\quad + k_{t-d}^2 N^T(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) + \epsilon_3\end{aligned}$$

It is also noted that

$$\begin{aligned}\Delta V_{L_t} &\triangleq \tilde{\theta}_{L_t}^T \tilde{\theta}_{L_t} - \tilde{\theta}_{L_{t-1}}^T \tilde{\theta}_{L_{t-1}} \\ &= 2k_{t-d} \phi_{L_{t-d}}^T \tilde{\theta}_{L_{t-1}} + k_{t-d}^2 \phi_{L_{t-d}}^T \phi_{L_{t-d}}\end{aligned}$$

Hence,

$$\begin{aligned}\Delta V_t &= \Delta V_{L_t} + \Delta V_{N_t} \\ &\leq 2k_{t-d} \left(\phi_{L_{t-d}}^T \hat{\theta}_{L_{t-1}} + K(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}}) - K(\phi_{N_{t-d}}, \theta_N) \right) \\ &\quad + k_{t-d}^2 \left(|N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}})|^2 + |\phi_{L_{t-d}}|^2 \right) + \epsilon_3 \\ &= -2k_{t-d} \tilde{u}_{t-d} + k_{t-d}^2 \left(|N(\phi_{N_{t-d}}, \hat{\theta}_{N_{t-1}})|^2 + |\phi_{L_{t-d}}|^2 \right) + \epsilon_3 \\ &\leq -\frac{2\alpha \tilde{u}_{t-d}^2}{1 + C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}}} + \frac{\alpha^2 \tilde{u}_{t-d}^2 (C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}})}{(1 + C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}})^2} + \epsilon_3 \\ &\leq -\frac{\alpha \tilde{u}_{t-d}^2}{1 + C_{N_{t-d}}^T C_{N_{t-d}} + \phi_{L_{t-d}}^T \phi_{L_{t-d}}} + \epsilon_3\end{aligned}$$

implying (P2) is satisfied. Therefore, the stability result in Theorem 4.5 can be carried directly over to the current approach provided that (PL1) and (PL2) are satisfied.

To achieve (PL1) and (PL2), we follow the similar procedure as introduced in Section 4.4. That is, for every pattern in the training set, the weights W are to be found to satisfy

$$\begin{aligned}|N_i(W)|^2 &\leq L_i \\ N_i^T(W) \tilde{\theta}_{N_i} &\leq D_i\end{aligned}$$

where $L_i = |C_{N_i}|^2$ and $D_i = K(\phi_i, \hat{\theta}_{N_i}) - K(\phi_i, \theta_{N_i})$. As shown in Section 4.3, the problem is equivalent to minimizing the following cost function J :

$$J = \frac{1}{2} \sum_{i=1}^M \left\{ \left(\max\{0, N_i^T(W) \tilde{\theta}_{N_i} - D_i\} \right)^2 + \frac{1}{b^2} \left(\max\{0, |N_i(W)|^2 - L_i\} \right)^2 \right\} \quad (4.45)$$

As in the previous section, $\frac{1}{b^2}$ is the relative weight for the two targets. A similar training procedure to that in Section 4.4 can be applied to find W in Eq. (4.45).

As is pointed out in [59], a major advantage of separating terms that are linearly parametrized from the rest of the model is reduction in dimensions of the training space. In this case, if the original algorithm is used, we need to vary ω_s , θ_s , $\hat{\theta}_s$ and $\hat{\theta}_s^d$ in their respective ranges. By comparison, if we take advantage of the linear structure, only a subset of them need to be varied. The dimensions of the training set are smaller, and, thus, it takes less time to train the network, reducing the computational complexity significantly.

4.7 Summary and Remarks

In this chapter, we considered the adaptive control of nonlinear dynamic systems where parameters occur nonlinearly. A neural network is used to generate the parameters of the adaptive controller with the structure of the controller determined using the nonlinear model. The notable property of the proposed neural controller is that it leads to stability of the overall adaptive system. In particular, it is shown that (a) given a neighborhood of the origin, upper bounds on the input and output of the closed-loop system can be established for all initial conditions starting in the neighborhood, (b) for small initial parameter estimation error, all signals entering the network remain within the sets where TANN is trained, and (c) the output of the system converges to a neighborhood of the origin. We also showed that by exploiting the linear structure, complexity of TANN can be significantly reduced.

Training of neural networks is formulated as a constrained optimization problem. The constraints that the neural network has to satisfy are the sufficient conditions that guarantee closed-loop stability. This approach deviates considerably from most of the neural network literature where the role of the neural networks is to meet tracking or stabilization objectives. This constrained problem is eventually reduced to an unconstrained one, which can be solved by various optimization methods.

An important aspect of the proposed approach is the training of the neural network that needs to be carried out off-line. Since the neural network is required to estimate the control parameters on-line for a particular controller structure, it needs to be

trained a priori to learn the map between the system variables and the controller parameters. This implies that during the training data collection phase, the parameter value as well as the corresponding response of the system must be known. This is a reasonable assumption since the data collection is often done off-line and in a controlled environment. For instance, an accurate system model, an extensive simulation model or an experimental setup all suffice to accomplish the formulation of the training set.

Chapter 5

Stable Control of Nonlinear Dynamic Systems Using Neural Networks

5.1 Introduction

In many of the applications of neural networks in control problems for cancelling unknown nonlinearity, such as in [43, 55], a prerequisite is that a controller structure is readily available such that the system can be stabilized if this nonlinearity is known. Unfortunately, unlike linear systems, determining a nominal controller for a nonlinear system itself is difficult even if the system model is well-known. To overcome this problem, a stability based approach for designing a neural controller is proposed in this chapter. The goal of this approach is to develop a new way of designing neural controllers which can stabilize more general nonlinear systems. The technique used to achieve this is similar to the recursive parameter estimation method discussed in the previous two chapters except that now the desired target is the change of a positive definite function of state variables instead of that of parameter estimation error. Nevertheless, the resulting controller and the class of problems considered in this chapter is quite different from the parameter estimation case.

This chapter is outlined as follows. The problem under consideration is described

in Section 5.2. The new controller design method is explained in Section 5.2, in which the training procedure is described in Section 5.3.1, and the existence of a controller and stability property of the closed-loop system is shown in Section 5.3.2. Several remarks of this approach are given in Section 5.4

5.2 Problem Statement

Consider the following nonlinear dynamic system

$$\begin{aligned}\dot{x} &= f(x, u) \\ y &= h(x)\end{aligned}\tag{5.1}$$

where $x \in R^n$, $u \in R^m$. Determination of a nonlinear controller

$$u = \gamma(y, t)$$

to stabilize (5.1) for general f and h is a difficult task even if f and h are known. For systems which are feedback linearizable, although such a γ exists, closed form solutions for γ cannot always be obtained. Our goal in this chapter is to construct a neural controller as

$$u = N(y; W)\tag{5.2}$$

where N is a neural network with weights W , and establish the conditions under which the closed-loop system is stable.

The nonlinear system in (5.1) is expressed as a combination of a linear part and a higher-order nonlinear part as

$$\begin{cases} \dot{x} = Ax + Bu + R_1(x, u) \\ y = Cx + R_2(x) \end{cases}\tag{5.3}$$

where $f(0, 0) = 0$ and $h(0) = 0$. We make the following assumptions:

(SA1) f, h are twice continuously differentiable and are completely known.

(SA2) There exists a K such that $(A - BKC)$ is asymptotically stable.

The problem is therefore to determine a controller as in Eq. (5.2), the structure of N , and a procedure for training the weights of N so that the closed-loop system is stable.

Again, in order for the neural network to approximate any continuous function, the class of neural networks used in this thesis must satisfy the universal approximator property defined in Definition 2.1.

5.3 Stable Neural Controller

With an input y in Eq. (5.2), the neural controller is therefore completely determined once the weights of the network are selected. Since this selection should be such that the closed loop is stable, the following approach is adopted. It is noted that while the controller and the training procedure is described for the continuous system in (5.1), a similar result can be derived for nonlinear discrete-time systems as well.

In order for the nonlinear controller in Eq. (5.2) to result in an asymptotically stable closed-loop system, it is sufficient to establish that a continuous positive definite function of the state variables decreases monotonically through output feedback. In other words, if we can find a scalar positive definite function with a negative definite derivative of all points in the state space, we can guarantee stability of the overall system. Here, the choices of the Lyapunov function candidates are limited to the quadratic form, i.e. $V = x^T P x$, where P is positive definite, and the goal is to choose the controller so that $\dot{V} < 0$ where

$$\dot{V} = 2x^T P f(x, N(h(x), W)). \quad (5.4)$$

To ensure $\dot{V} < 0$, we define a *desired* time-derivative \dot{V}_d is defined as

$$\dot{V}_d = -x^T Q x \quad \text{where } Q = Q^T > 0 \quad (5.5)$$

P and Q matrices are chosen as follows. First, a matrix K to make $(A - BKC)$

asymptotically stable exists according to assumption (SA2). In general, such a K can be easily obtained from linear controller design methods such as the LQ scheme. A P, Q pair can subsequently be found by choosing an arbitrary positive definite matrix Q and solving the following Lyapunov equation to obtain the positive definite P (Theorem 2.4):

$$(A - BKC)^T P + P(A - BKC) = -Q \quad (5.6)$$

Based on the linearization principle (ref. pp. 170-173 of [60]), it follows that there is at least a linear controller $u = -Ky$ which can make the derivative of $V = x^T P x$ as close to $-x^T Q x$ as possible in a neighborhood of the origin with the choices of P and Q . Our aim is to use a controller as in Eq. (5.2) so that by allowing nonlinearity, the neural controller can achieve a larger basin of convergence.

With the controller of the form of Eq. (5.2), the goal is to find W in Eq. (5.2) which yields

$$\dot{V} \leq \dot{V}_d \quad (5.7)$$

along the trajectories in a compact set $\mathcal{X} \subset \mathfrak{R}^n$ containing the origin in the state space. Let x_i denote the value of a sample point of $x \in \mathcal{X}$ in the state space where i is an index. To establish (5.7), it is necessary that for every x_i in a neighborhood $\mathcal{X} \subset \mathfrak{R}^n$ of the origin, $\dot{V}_i \leq \dot{V}_{d_i}$, where $\dot{V}_i = 2x_i^T P f(x_i, N(h(x_i), W))$ and $\dot{V}_{d_i} = -x_i^T Q x_i$. That is, the goal is to find a W such that the inequality constraints

$$\Delta V_{e_i} \leq 0, \quad i = 1, \dots, M \quad (5.8)$$

are satisfied, where $\Delta V_{e_i} = \dot{V}_i - \dot{V}_{d_i}$ and M denotes the total number of sample points in \mathcal{X} . Alternatively, this can be posed as an optimization problem with no cost function and subject to the inequality constraints in (5.8). This problem is similar to those in the last two chapters. If the same quadratic penalty function method is used, the problem becomes finding W to minimize the following cost function:

$$\min_W J \triangleq \min_W \frac{1}{2} \sum_{i=1}^M (\max\{0, \Delta V_{e_i}\})^2 \quad (5.9)$$

It is worth noting that, though we focus on satisfying the constraints in (5.7) in the above derivation, $\dot{V}_i \leq 0$ is all that is needed to assure nominal stability. There are several other ways to achieve this. For example, training the network to minimize $\sum_{i=1}^M \dot{V}_i$ subject to the constraint $\dot{V}_i < 0$ for $1 \leq i \leq M$ suffices to ensure the condition. If the penalty function method is used, as shown in Section 2.4, the problem involves finding a sequence of weights, $W^{(k)}$, which minimizes the following augmented Lagrangian

$$L_{c^{(k)}}(W, \lambda^{(k)}) = \sum_{j=1}^M \dot{V}_j(W) + \frac{1}{2c^{(k)}} \sum_{j=1}^M \left\{ \left(\max\{0, \lambda_j^{(k)} + c^{(k)} \dot{V}_j(W)\} \right)^2 - \lambda_j^{(k)} \right\}$$

where the Lagrange multipliers $\lambda_j^{(k)}$ and $c^{(k)}$ are updated recursively as

$$\begin{aligned} \lambda_j^{(k+1)} &= \max\{0, \lambda_j^{(k)} + c^{(k)} \dot{V}_j(W^{(k)})\} \\ c^{(k+1)} &= \beta c^{(k)} \end{aligned}$$

and $\beta > 1$. Since the process of finding weights is equivalent to solving a sequence of unconstrained problems due to the presence of $\sum_{i=1}^M \dot{V}_i$ in the original cost function, it takes longer time to train the network. Nevertheless, this approach can potentially result in large stable region since less stringent constraints are imposed.

5.3.1 Training Scheme

To minimize J in Eq. (5.9), we can use the gradient method or higher order methods such as the Levenberg-Marquardt method. If the gradient method is used, W can be updated as

$$\Delta w_j = -\rho \frac{\partial J}{\partial w_j} \quad (5.10)$$

where ρ is the step size and w_j denotes j -th element of W . $\frac{\partial J}{\partial w_j}$ in turn can be obtained as:

$$\frac{\partial J}{\partial w_j} = \sum_{i \in \mathcal{A}} \frac{\partial J}{\partial N(y_i; W)} \frac{\partial N(y_i; W)}{\partial w_j}$$

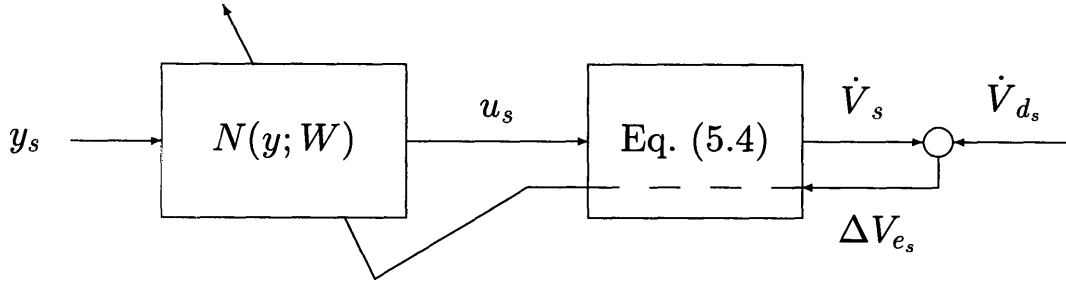


Figure 5-1: Block Diagram of the Network Training

$$= 2 \sum_{i \in \mathcal{A}} \Delta V_{e_i} x_i^T P \frac{\partial f(x_i, u_i)}{\partial u} \frac{\partial N(y_i; W)}{\partial w_j} \quad (5.11)$$

where \mathcal{A} denotes the set of patterns where $\Delta V_{e_i} > 0$ and the term $\frac{\partial N(y_i; W)}{\partial w_j} = [\frac{\partial N_1(y_i; W)}{\partial w_j}, \dots, \frac{\partial N_n(y_i; W)}{\partial w_j}]^T$ represents the gradient of the outputs of the neural network with respect to the parameter w_j of the network, which are given in Section 2.4. To train a neural network to minimize Eq. (5.9), we can form a set, which is composed of all the data needed in the training algorithm, and continuously apply Eq. (5.10) on the set. This procedure is graphically shown in Figure 5-1. Suppose for a state x_s , the output is y_s and the corresponding control input is $u_s = N(y_s; W)$. In the forward path, the neural network N has as its input vector y_s , and its output is u_s . By using this preliminary u_s , we can calculate \dot{V}_s from Eq. (5.4). \dot{V}_s is then compared with \dot{V}_{d_s} to obtain the error signal ΔV_{e_s} . Only if $V_{e_s} > 0$ is the pattern used to update W in the network.

The procedure of training a neural network in Eq. (5.2) to stabilize Eq. (5.1) is summarized in the following steps, where the stochastic gradient descent method is assumed to be used.

Step 1 Find the linear approximation matrices A, B and C of the system in Eq. (5.2).

Choose a matrix K to make $(A - BKC)$ asymptotically stable.

Step 2 Define a negative definite function $\dot{V}_d = -x^T Q x$, which is our desired change of Lyapunov function. The corresponding Lyapunov function candidate $V =$

$x^T P x$ is obtained by solving the Lyapunov equation (5.6) to find the positive definite matrix P .

Step 3 Sample x in a subspace $\mathcal{X} \in \mathfrak{R}^n$. Its value is, say, x_1 . The corresponding output vector y_1 and \dot{V}_{d_1} at the particular point x_1 can be evaluated as $y_1 = h(x_1)$ and $\Delta V_{d_1} = -x_1^T Q x_1$ respectively. A typical data element is of the form $(x_1, y_1, \dot{V}_{d_1})$. By repeating the above procedure for different $x_i \in \mathcal{X}$ where x_i can be either chosen randomly or evenly spaced, we can then formulate the training set as:

$$T_{train} = \{(x_i, y_i, \dot{V}_{d_i}) | 1 \leq i \leq M\}$$

Similarly, a testing set T_{test} can also be formed for cross-validation. In order to ensure existence of a map $N(\cdot)$, the training set should initially compose of samples in a smaller region around the origin, and could be gradually enlarged as training is successful.

Step 4 For each element x_i in the training set, the outputs of the network can be obtained as $u_i = N(y_i; W)$. Based on this preliminary u_i , we can calculate $\dot{V}_i = 2x_i^T P f(x_i, u_i)$ and subsequently $\frac{\partial J}{\partial w_j}$ using Eq. (5.11).

Step 5 The weights of the neural network is updated with the step size ρ as

$$\Delta w_j = -\rho \frac{\partial J}{\partial w_j}$$

Step 6 Repeat *Step 4* and *Step 5* until J stops improving. □

After the training is finished, it is expected that $\dot{V}_i - \dot{V}_{d_i} \leq \epsilon$ for a small $\epsilon > 0$ of all the sample points x_i . It implies that $\dot{V}_i < 0$ except possibly at some points where \dot{V}_{d_i} is small, which are in a small neighborhood of the origin. This ensures that states of the closed-loop system converges to the neighborhood when the neural controller in Eq. (5.2) is implemented as we shall see in the next section.

The procedure discussed in this section can also be applied to discrete-time systems with some minor modification. In particular, the discrete version of Lyapunov equation in Eq. (2.5):

$$(A - BKC)^T P(A - BKC) - P = -Q \quad (5.12)$$

replaces Eq. (5.6) to determine the matrix P . In addition, \dot{V} in (5.4) is substituted by the difference of the Lyapunov function candidate:

$$\begin{aligned} \Delta V_t &\triangleq x_{t+1}^T P x_{t+1} - x_t^T P x_t \\ &= f^T(x_t, N(h(x_t), W)) P f^T(x_t, N(h(x_t), W)) - x_t^T P x_t \end{aligned}$$

Otherwise, the rest of the training procedure follows the same line as its continuous-time counterpart.

5.3.2 Stability Analysis

In this section, that the proposed neural controller and the corresponding training procedure lead to closed-loop stability is established. First of all, in order to guarantee the training is successful, it is necessary that there indeed exists a smooth function which, when used as a controller, can make $|\dot{V} - \dot{V}_d|$ as small as possible in a neighborhood of the origin. Theorem 5.1 establishes this by making use of the linearization principle for stability (ex. [60]). When this smooth function is approximated by a neural network, the universal approximator property in Definition 2.1 ensures stability of the closed-loop system using the neural controller. For systems where a more complex controller exists to achieve $\dot{V} - \dot{V}_d \leq 0$ in a larger region, Theorem 5.2 says that the states of the closed-loop system with a well-trained neural controller converges to a neighborhood of the origin.

In the following theorem, $x(t_0) = x_0$ corresponds to the initial conditions of the system in (5.1).

Theorem 5.1 *Consider the closed-loop system in Eqs. (5.1) and (5.2), which satis-*

fies the assumptions (SA1) and (SA2).

(i) For every $\epsilon > 0$, there exist $\gamma_1 > 0$, $0 < \gamma_2 < \gamma_1$ and a neural network $N \in \mathcal{N}$ such that

$$\dot{V} - \dot{V}_d < \epsilon \quad \forall x \in \mathcal{X}$$

where \dot{V} and \dot{V}_d are given by Eqs. (5.4) and (5.5), and $\mathcal{X} = \{x \in \mathbb{R}^n \mid \gamma_2 \leq \|x\| \leq \gamma_1\}$.

(ii) There exist a neural network $N \in \mathcal{N}$ and an open set \mathcal{U} containing the origin such that the solutions converge to a neighborhood $\mathcal{O} \subset \mathcal{U}$ of the origin for every $x_0 \in \mathcal{U}$.

Proof: From (SA2) it follows that $K \in \mathbb{R}^n$ exists such that Eq. (5.6) is satisfied. Since \mathcal{X} is a closed annulus region in \mathbb{R}^n and h is continuous on \mathcal{X} , the set $\mathcal{Y} = \{y \mid h(x) = y, x \in \mathcal{X}\}$ is compact. Therefore from Definition 2.1, it follows that we can find an $N \in \mathcal{N}$ such that $|N(y) - (-Ky)|$ can be made as small as possible for every $y \in \mathcal{Y}$. Let $N(y) = -Ky + E(y)$. Using Eq. (5.3), we obtain that

$$u = -KCx - KR_2(x) + E(y) \quad (5.13)$$

and the closed-loop system to be

$$\dot{x} = (A - BKC)x + R_4(x)$$

where $R_4(x) = R_1(x, u(x)) - BKR_2 + BE$. Since $V = x^T Px$ and $\dot{V} = -x^T Qx + 2x^T PR_4(x)$, it follows that

$$\dot{V} - \dot{V}_d = 2x^T PR_4(x)$$

We need to establish therefore that a annulus region \mathcal{X} exists such that for every $x \in \mathcal{X}$, $|2x^T PR_4(x)| < \epsilon$.

Since $f \in \mathcal{C}^{(2)}$, according to Taylor's theorem [53], $\lim_{\|(x,u)\| \rightarrow 0} \frac{\|R_1(x,u)\|}{\|(x,u)\|} = 0$. Hence, for every $\epsilon_1 > 0$, there exists $\delta_1 > 0$ such that

$$\|R_1(x, u)\| < \epsilon_1 \|(x, u)\| \quad \text{if } \|(x, u)\| < \delta_1 \quad (5.14)$$

Similarly, since $h \in \mathcal{C}^{(2)}$, $\lim_{\|x\| \rightarrow 0} \frac{\|R_2(x)\|}{\|x\|} = 0$ and thus, for every $\epsilon_2 > 0$, there exists

$\delta_2 > 0$ such that

$$\|R_2(x)\| < \epsilon_2 \|x\| \quad \text{if } \|x\| < \delta_2 \quad (5.15)$$

From Eq. (5.13), it follows that

$$\begin{aligned} \|(x, u)\| &< (1 + \|KC\| + \epsilon_2 \|K\|) \|x\| + \|E\| \\ &= c_2 \|x\| + \|E\| \end{aligned} \quad (5.16)$$

The above inequality is valid if $\|x\| < \delta_2$. By using (5.14) and (5.16), we can derive

$$\|R_1(x, u)\| < \epsilon_1 (c_2 \|x\| + \|E\|)$$

if $\|x\| < \delta_2$ and $\|(x, u)\| < \delta_1$.

For a given $c_1 > 0$, let $\epsilon_2 = \frac{c_1}{3\|BK\|}$, $\epsilon_1 = \frac{c_1}{3c_2}$, $\gamma_1 = \min\left(\frac{\delta_1}{2c_2}, \frac{\delta_2}{2}\right)$ and $0 < \gamma_2 < \gamma_1$. For the choices of ϵ_1 , ϵ_2 , γ_1 and γ_2 , if $x \in \mathcal{X}$, then $\|x\| < \delta_2$ and $\delta_1 > c_2 \|x\|$, which implies that $\|(x, u)\| < \delta_1$ can indeed be satisfied for a small enough $\|E\|$. From Definition 2.1 and since \mathcal{Y} is a compact set, we can find $N \in \mathcal{N}$ such that

$$\|E(y)\| < \epsilon_3 \quad (5.17)$$

If $\epsilon_3 = \min\left(\frac{c_1}{3(\epsilon_1 + \|B\|)}, \delta_1 - c_2 \gamma_1\right)$, then $\|(x, u)\| < \delta_1$ and furthermore

$$\begin{aligned} \|R_4(x)\| &\leq \|R_1(x, u)\| + \|BKR_2(x)\| + \|BE(y)\| \\ &< (c_2 \epsilon_1 + \|BK\| \epsilon_2) \|x\| + (\epsilon_1 + \|B\|) \epsilon_3 \\ &\leq c_1 \|x\| \end{aligned}$$

for every $x \in \mathcal{X}$. Since c_1 is arbitrary, we can choose c_1 so that

$$\|2x^T P R_4(x)\| < \epsilon \quad \forall x \in \mathcal{X}$$

which establishes (i).

If we choose $c_1 < \frac{\lambda_{\min}(Q)}{2\lambda_{\max}(P)}$, then $\dot{V} < -(\lambda_{\min}(Q) - 2c_1 \lambda_{\max}(P)) \|x\|^2 < 0$ for every

$x \in \mathcal{X}$. Define $\mathcal{U}_1 = \{x \in \mathbb{R}^n \mid \|x\| = \gamma_1\}$, $\mathcal{U}_2 = \{x \in \mathbb{R}^n \mid \|x\| = \gamma_2\}$, $\alpha_1 = \min_{x \in \mathcal{U}_1} V(x)$, $\alpha_2 = \max_{x \in \mathcal{U}_2} V(x)$ and $\mathcal{O} = \{x \in \mathbb{R}^n \mid V(x) < \alpha_2\}$, where $\alpha_1 > 0$ and $\alpha_2 > 0$ exist since \mathcal{U}_1 and \mathcal{U}_2 are compact. Choose $\mathcal{U} = \{x \in \mathbb{R}^n \mid V(x) < \alpha_1\}$ and γ_2 to be small enough so that $\alpha_2 < \alpha_1$. Such a γ_2 can always be found since $V(x) \rightarrow 0$ as $x \rightarrow 0$. Furthermore, $\dot{V} < 0$ if $x \in \mathcal{U} - \mathcal{O}$. Therefore, according to Theorem 2.7, the solutions of Eqs. (5.1) and (5.2) converge to the neighborhood \mathcal{O} of the origin for every $x \in \mathcal{U}$. ■

The above theorem establishes the existence of a stabilizing neural network controller around the origin such that $\dot{V} - \dot{V}_d < \epsilon$, for every $x \in \mathcal{X}$. The proof was given essentially along the same lines as the proof of the linearization principle for stability (ex. [60]). The important modification is the introduction of a neural network which replaces the stabilizing linear controller. For systems where a continuous but unknown function $K(y)$ exists such that for $V = x^T P x$, the control input $u = K(y)$ yields $\dot{V} \leq -x^T Q x$, we can find a neural network $N(y)$ which approximates $K(y)$ arbitrarily closely in a compact set leading to closed-loop stability. This is summarized in Theorem 5.2.

Theorem 5.2 *Let there be a continuous function $K(h(x))$ such that $2x^T P f(x, K(h(x))) + x^T Q x \leq 0$ for every $x \in \mathcal{X}$ where $\mathcal{X} \subset \mathbb{R}^n$ is a compact set containing neighborhoods of the origin. Then, given a neighborhood $\mathcal{O} \subset \mathcal{X}$ of the origin, there exists a neural controller $u = N(h(x); W)$ and an open set $\mathcal{Y} \subset \mathcal{X}$ such that the solutions of*

$$\dot{x} = f(x, N(h(x); W))$$

converge to \mathcal{O} , for every $x_0 \in \mathcal{Y}$.

Proof: Since $f \in \mathcal{C}^2$,

$$f(x, K(y) + E(y)) = f(x, K(y)) + \frac{\partial f}{\partial u}(x, K(y))E(y) + R_1(E(y))$$

where $\lim_{E(y) \rightarrow 0} \frac{R_1(E(y))}{\|E(y)\|} = 0$. Hence, for every $\epsilon_1 > 0$, there exists a $\delta_1 > 0$ such that

$$\|R_1(E(y))\| < \epsilon_1 \|E(y)\| \quad \text{if } \|E(y)\| < \delta_1$$

Since $f \in \mathcal{C}^2$, $h \in \mathcal{C}^2$ and K is continuous, $\frac{\partial f}{\partial u}(x, K(y))$ is continuous on \mathcal{X} . Furthermore, since \mathcal{X} is compact, there is a $c_1 > 0$ so that $\|\frac{\partial f}{\partial u}(x, K(y))\| < c_1$ for every $x \in \mathcal{X}$.

Define $\mathcal{Y} = \{x \in \mathbb{R}^n \mid V(x) < \alpha_1\}$, $\mathcal{U}_1 = \{x \in \mathbb{R}^n \mid \|x\| = \gamma_1\}$, $\alpha_2 = \max_{x \in \mathcal{U}_1} V(x)$ and $\mathcal{U}_2 = \{x \in \mathbb{R}^n \mid V(x) < \alpha_2\}$. Choose α_1 and γ_1 to be small enough so that $\mathcal{Y} \subset \mathcal{X}$, $\mathcal{U}_1 \subset \mathcal{O}$ and $\alpha_2 < \alpha_1$. Since \mathcal{X} is compact, there exists a $N \in \mathcal{N}$ such that for every $x \in \mathcal{X}$

$$\|E(y)\| \triangleq \|N(y) - K(y)\| < \min(\delta_1, c_2 \gamma_1)$$

where $c_2 = \frac{\lambda_{\min}(Q)}{4\lambda_{\max}(P)(c_1 + \epsilon_1)}$.

Thus, for every $x \in \mathcal{Y} - \mathcal{U}_2$, $x \geq \gamma_1$ and

$$\begin{aligned} \dot{V} &= 2x^T P f(x, N(y)) \\ &= 2x^T P f(x, K(y)) + 2x^T P \frac{\partial f}{\partial u}(x, K(y)) + 2x^T P R_1(E(y)) \\ &\leq -x^T Q x + 2x^T P \frac{\partial f}{\partial u}(x, K(y)) + 2x^T P R_1(E(y)) \\ &\leq -\lambda_{\min}(Q) \|x\|^2 + 2c_1 \lambda_{\max}(P) \|x\| \|E\| + 2\epsilon_1 \lambda_{\max}(P) \|x\| \|E\| \\ &< -[\lambda_{\min}(Q) - 2\lambda_{\max}(P)(c_1 + \epsilon_1)c_2] \|x\|^2 \\ &< 0 \end{aligned}$$

According to Theorem 2.7, the solutions converge to \mathcal{O} for every $x_0 \in \mathcal{Y}$. ■

The following are several remarks regarding training and performance of the resulting controller:

1. It should be noted that the region \mathcal{X} of initial conditions for which the stability result in Theorem 5.1 is valid can be small. This is because the neural network was chosen so as to approximate a linear controller $-Ky$. Nevertheless, as long

as a continuous controller which results in $\dot{V} \leq \dot{V}_d$ along every trajectory in a larger region exists, we can use the proposed procedure to search for such a function. And, as stated in Theorem 5.2, with proper training, we can ensure that the states of the resulting closed-loop system converge to a neighborhood of the origin.

2. It can be seen from Theorem 5.1 and 5.2 that, as opposed to most analytically expressed controllers where zero steady state error can be achieved under the disturbance-free condition, the neural controllers generally result in some small steady state error. To ensure the states converging to the origin as close as possible, small γ_2 and, hence, ϵ_3 are required in Theorem 5.1. To achieve this, we need a more complicated network especially around the origin, and it inevitably takes longer time to train the network. The same conclusion can also be drawn from Theorem 5.2. Hence, there is a tradeoff between network complexity and accuracy of the neural controller.

5.4 Summary and Remarks

In this chapter, a stability based approach is taken to design a neural controller. With a quadratic Lyapunov function candidate V of state variables, the neural controller is chosen such that a negative definite time-derivative \dot{V} is resulted along trajectories of the closed-loop system. A stability proof is given, which establishes the conditions under which such a neural controller can be found. The proposed procedure allows us to design stabilizing controllers for systems where conventional methods may be inadequate. The framework chosen naturally lends itself to a mathematically more tractable problem formulation and can be applied to more general nonlinear systems. Training such a neural controller off-line permits more efficient algorithms in nonlinear programming being used. One such algorithm is discussed in this chapter as well. Furthermore, the nonlinear representational ability of neural networks allows the controller to achieve a larger stable region, compared to a linear controller. This shall be confirmed by the simulation results provided in Chapter 6.

The approach proposed in this chapter starts from selecting a Lyapunov function candidate. The role of the neural controller is then to make this candidate indeed a Lyapunov function. It is known from [4] that if there is a smooth Lyapunov function of the closed-loop system in (5.1), then there must exist a corresponding continuous controller. [61] further proposed a way to construct explicitly the controller. However, the method can only be applied to systems which are affine in control and with full-state feedback. The approach suggested here, on the other hand, does not require such assumptions. Furthermore, it is much easier to incorporate other requirements into the approach of constructing the neural controller, which is essential in improving performance of the closed-loop systems.

There are several classes of systems where existence of a nonlinear controller can be guaranteed while difficulty still remains in actually solving such a controller. For example, the feedback linearization technique establishes a constructive procedure for finding nonlinear controllers for systems affine in control under some conditions [22]. However, obtaining such controllers through the procedure requires solving simultaneous nonlinear partial differential equations, which is notoriously difficult. Hence, even for those classes of problems, the proposed neural controller still offers an alternative solution.

Chapter 6

Simulation Results

6.1 Introduction

In the last three chapters, the model-based algorithms and their convergence properties for parameter estimation and controller design are developed. In this chapter, the detailed procedure for construction of the training sets and on-line implementation of those algorithms are illustrated through extensive simulations. The examples chosen are intended to show how these methods can be applied to different classes of systems.

In Section 6.2, both the block and the recursive versions of TANN are implemented on systems represented in regression as well as state space forms. To stabilize a system with unknown parameters occurring nonlinearly, the TANN controller proposed in Chapter 4 can be applied. In Section 6.3, stabilization of one such system is shown. For a system whose controller structure is unknown, the simulation example in Section 6.4 demonstrates how a stable neural controller can be constructed, based on the method derived in Chapter 5.

6.2 Parameter Estimation Using TANN

In this section, four simulation examples are presented to demonstrate parameter estimation using TANN. These examples cover system models represented in regression

forms and state space forms in discrete time as well as continuous time. Both the block and the recursive versions of TANN are implemented. The simulation results of outputs corrupted by noise are also compared to those of the noise-free cases to illustrate robustness of the algorithms to disturbance.

6.2.1 Example 1: A Simple System

In this section, a simple nonlinear system with one parameter to be estimated is introduced. The value of the parameter can indeed be calculated by simply inverting the function. The purpose of this example is to demonstrate graphically the idea behind the two methods and how they differ in achieving the estimate.

Consider the system as follows:

$$y_t = \frac{2u_{t-1}}{1 + 0.1u_{t-1}^4} e^{-\frac{\theta^2}{2}} \quad (6.1)$$

where θ is the parameter to be estimated on-line, and y_t and u_{t-1} are all measurable signals. Prior knowledge regarding u and θ is that $1 < u < 3$ and $0.8 < \theta < 2$. With some manipulation, it can be found that Eq. (6.1) can be rewritten as

$$\theta = F(u_{t-1}, y_t)$$

where

$$F(u_{t-1}, y_t) = \left[2 \ln \left(\frac{2u_{t-1}}{(1 + 0.1u_{t-1}^4)y_t} \right) \right]^{\frac{1}{2}} \quad (6.2)$$

The goal is to determine θ based on measurable signals using both the block and the recursive TANN.

The Block Method

The block method for this example can be written in the following form

$$\hat{\theta} = N(y_t, u_{t-1})$$

where $N(\cdot)$ denotes a neural network with inputs y_t and u_{t-1} . The output of the network is the parameter estimate $\hat{\theta}$. In this example, a Gaussian network of the form shown in Eq. (2.1) is used to construct the mapping, where $\alpha_1^2 = 2.045$, $\alpha_2^2 = 3.615$, $p = 400$ and $I = 2$. Locations of the centers are randomly distributed in the range of the training set.

For N to model the mapping F , we choose the training set to contain data points (u_{t-1}, y_t, θ) where y_t , u_{t-1} and θ are related by Eq. (6.1). The procedure for constructing such a set is by first randomly choosing a θ , say θ_1 , in its range. By varying the input u_t between 1 and 3, we can gather the corresponding y_t for this particular θ_1 . A series of (u_{t-1}, y_t, θ_1) triples can then be formed such as (u_1, y_2, θ_1) , (u_2, y_3, θ_1) , $(u_3, y_4, \theta_1), \dots$. Repeating the same procedure by selecting different θ , we can form a training set composed of the triples as

$$T_{train} = \{(u_i, y_j^i, \theta_j) | 1 \leq i \leq q, 1 \leq j \leq p\}$$

In this example, $q = 40$ and $p = 40$, with a total of 1,600 data points in the training set. A similar testing set can also be constructed for cross-validation purpose.

After the training set is formed, the Gaussian network is trained on the set using the recursive least squares method. Figure 6-1 shows the target implicit function map calculated from Eq. (6.2). After one epoch of training, the resulting map from the neural network is shown in Figure 6-2, where the implicit function has not been learned completely. Since a neural network only approximates a function well in the region where it is trained, it is important for the training set to cover regions where signals during on-line operation might be present. In this example, if the neural network is trained only for $1.75 < u < 2.25$ and $1 < \theta < 2$ as opposed to $1 < u < 3$ and $0.8 < \theta < 2$, the function constructed from the network does not approximate the implicit function well outside the trained region, as shown in Figure 6-3. With a network thoroughly trained in the appropriate region, the implicit function can be fully reconstructed in that region. Figure 6-4 shows the resulting neural network approximation of the implicit function, which is nearly identical to

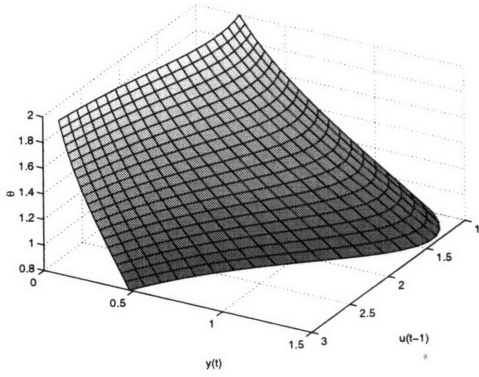


Figure 6-1: The Target Function

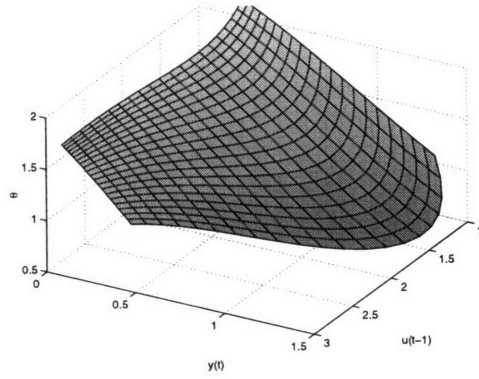


Figure 6-2: Without Enough Training

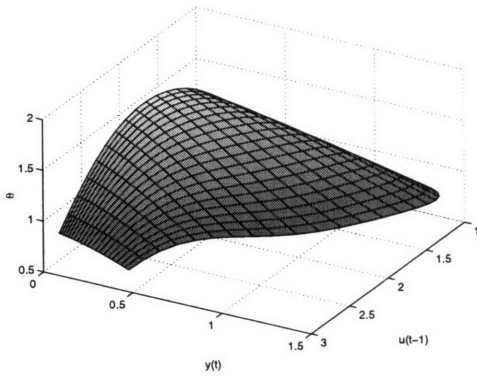


Figure 6-3: Inappropriate Training Set

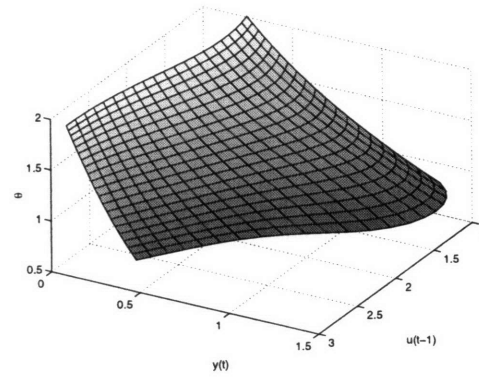


Figure 6-4: Correct Approximation

the desired nonlinear map in Figure 6-1.

During on-line operation, the value of θ might vary due to different operating conditions. The trained network can then be implemented on-line to estimate the current value of θ . The on-line process starts by measuring u_{t-1} and y_t , which are in turn used as inputs of the network. The output of the network is the current estimate of θ . To reduce the variance of estimation error, the output from the network is postprocessed by a K-means like algorithm as described in Eq. (3.9). The resulting algorithm is shown as follows:

$$\bar{\theta} = N(y_t, u_{t-1}) \quad (6.3)$$

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \rho e^{-\frac{1}{\sigma^2} \|\bar{\theta}_t - \hat{\theta}_{t-1}\|^2} (\bar{\theta}_t - \hat{\theta}_{t-1}) \quad (6.4)$$

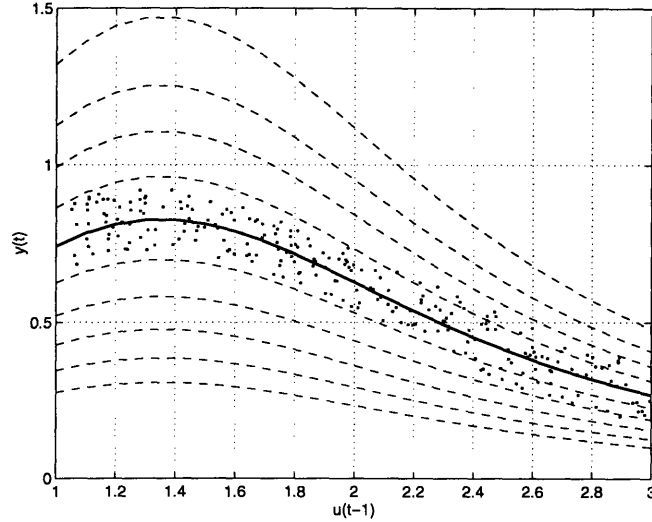


Figure 6-5: Parameter Estimate $\bar{\theta}$ under Noisy Measurement in Example 1

where, in this example, $\rho = 0.2$, $\frac{1}{\sigma^2} = 5$. The simulation is done with $\theta = 1.34$ and y_t corrupted by a random signal uniformly distributed between ± 0.1 . Figure 6-5 shows the estimate directly from the output of the network (i.e. $\bar{\theta}$ in Eq. (6.3)), where the dashed lines are constant θ contours, the solid line corresponds to the actual value $\theta = 1.34$ and the dots represent the estimates during simulation. Under the same conditions except with the output from the network being processed by (6.4), the result is shown in Figure 6-6. By comparing the two figures, we can see that the variance of the estimation error is reduced due to the postprocessing scheme.

The Recursive Method

The recursive estimation algorithm for this particular example is written as follows:

$$\begin{aligned} \hat{\theta}_t &= \hat{\theta}_{t-1} + \Delta \hat{\theta}_t \\ \Delta \hat{\theta}_t &= \begin{cases} N(y_t, u_{t-1}, \hat{\theta}_{t-1}) & \text{if } |\tilde{y}_t| > \delta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (6.5)$$

where $\hat{\theta}_t$ and $\hat{\theta}_{t-1}$ represent the current and previous estimates of θ respectively, and $\tilde{y}_t = \hat{y}_t - y_t$, with $\hat{y}_t = \frac{2u_{t-1}}{1+0.1u_{t-1}^4} e^{-\frac{\hat{\theta}_{t-1}^2}{2}}$. A Gaussian network of the form in Eq. (2.1)

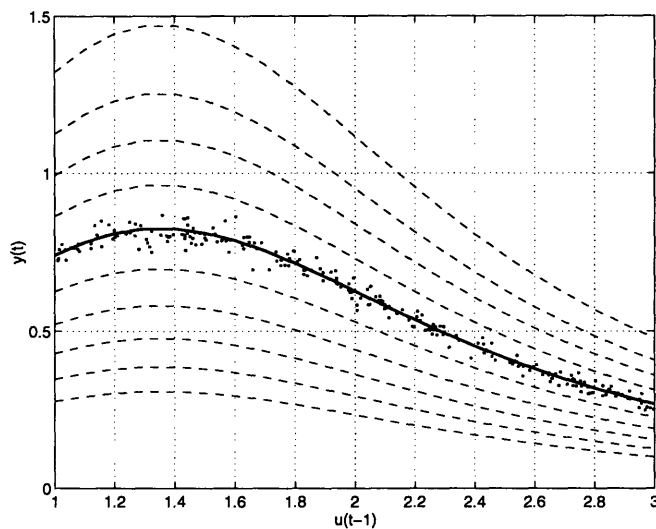


Figure 6-6: Parameter Estimate $\hat{\theta}$ under Noisy Measurement with Post-processing in Example 1

with $p = 500$ and variances $\alpha_i = 8.535, 12.370$ and 7.481 corresponding to y_t, u_{t-1} and $\hat{\theta}_{t-1}$ respectively are used to model $N(\cdot)$.

The training set consists of data points $(\theta_k, u_i, \hat{\theta}_j, y_k^i, \hat{y}_j^i)$, where they are related as

$$y_k^i = \frac{2u_i}{1+0.1u_i^4} e^{-\frac{\theta_k^2}{2}}, \quad \hat{y}_j^i = \frac{2u_i}{1+0.1u_i^4} e^{-\frac{\hat{\theta}_j^2}{2}}$$

The procedure of forming the training set is similar to that of the block method. By selecting $\theta = \theta_1$ and varying u , we can formulate the triple (y_1^i, u_i, θ_1) . For the same u_i but with $\theta = \hat{\theta}_1$, where $\hat{\theta}_1$ is selected in the same range as θ , the triple $(\hat{y}_1^i, u_i, \hat{\theta}_1)$ can also be attained. The process is repeated for different θ and $\hat{\theta}$. A training set of the following form is obtained.

$$T_{train} = \{(\theta_k, u_i, \hat{\theta}_j, y_k^i, \hat{y}_j^i) \mid 1 \leq i \leq q; 1 \leq j \leq p; 1 \leq k \leq r\}$$

In this example, a total of 1,600 data points are used in the training set. A similar testing set can also be constructed for cross-validation purpose.

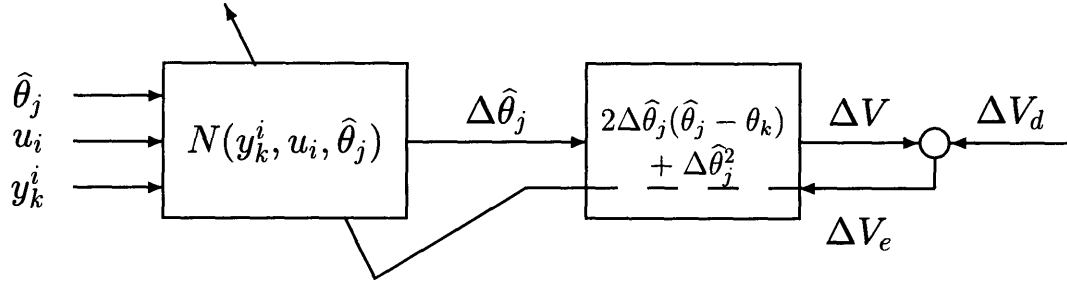


Figure 6-7: Block Diagram of the Network Training

N is trained on the training set such that $\Delta V \rightarrow \Delta V_d$ at the data points, where

$$\begin{aligned} \Delta V &= 2N^T(\hat{\theta}_j - \theta_k) + N^T N \\ \Delta V_d &= -\frac{2 + C^T(\cdot)C(\cdot)}{(1 + C^2(\cdot))^2} (y - \hat{y})^2 \\ C(\cdot) &= \left. \frac{\partial}{\partial \theta} \left(\frac{u}{1 + 0.1u^4} e^{-\frac{\theta^2}{2}} \right) \right|_{\theta=\theta_0} = -\frac{u\theta_0}{1 + 0.1u^4} e^{-\frac{\theta_0^2}{2}} \end{aligned}$$

and $\theta_0 = 1.4$. It can be seen that all the data required to calculate ΔV and ΔV_d can be found in T_{train} . A schematic drawing of the training process is shown in Figure 6-7. As shown in the figure, each data point in the training set is sequentially applied to calculate the corresponding ΔV and target ΔV_d . $N(\cdot)$ is then adjusted accordingly to make ΔV_e approach zero using either the gradient descent algorithm with a cost function $J = \sum \frac{1}{2} \Delta V_e^2$ or other optimization schemes. For this simulation, the Levenberg-Marquardt method is used to train the network.

When the trained network is implemented on-line, it uses the measurement y_t, u_{t-1} and $\hat{\theta}_{t-1}$ as inputs, and the output of the neural network is the current adjustment of the parameter estimate. The actual change of the estimate depends on $\tilde{y}_t = \hat{y}_t - y_t$, as shown in Eq. (6.5). If $|\tilde{y}_t| > \delta$, the adjustment is made according to the output of the neural network. Otherwise, the estimate remains unchanged. The procedure is shown in Figure 3-2. The first simulation is done with $\theta = 1.34$ and $\delta = 0.01$ in Eq. (6.5). Figure 6-8 shows the simulation result for the noise-free case, where the dashed and solid lines are constant $\hat{\theta}$ contours while the solid line represents the true

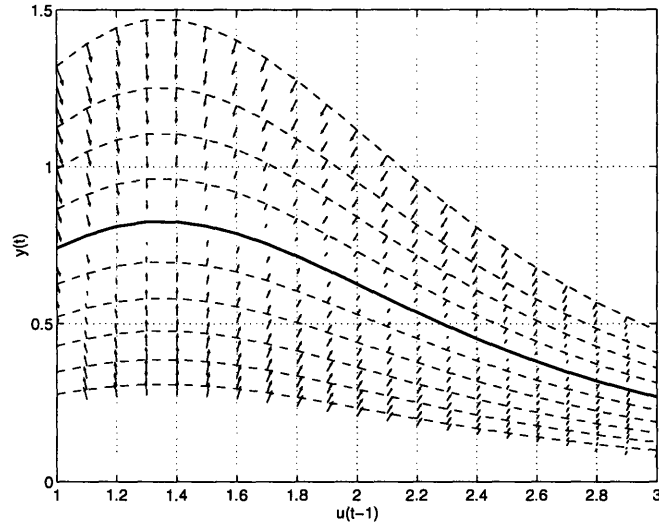


Figure 6-8: Correction of Parameter Estimate $\Delta\theta$ under Noise-free Measurement in Example 1

value of θ . Directions and lengths of the arrows in the figure denote the signs and relative magnitude of $\Delta\hat{\theta}$ given $(y, u, \hat{\theta})$ at the corresponding point as inputs of the network. It can be seen that the corrections $\Delta\hat{\theta}$ point in the directions of decreasing estimation error except in the region around the solid contour. In other words, the estimation error outside the region always decreases no matter what $(y_t, u_{t-1}, \hat{\theta}_{t-1})$ is. For the system under the same measurement noise as the block method, the contour plots are shown in Figures 6-9 and 6-10 for $\delta = 0$ and 0.12 respectively. It can be observed from Figure 6-9 that there are arrows along the solid line, which implies that the algorithm continues adapting even if the correct parameter value has already been reached. This behavior might result in instability in a dynamic system as explained in Chapter 3. With the dead zone modification ($\delta \neq 0$) in Eq. (6.5), the estimate stops changing when it is close to the true value, and the effect of noise starts to dominate, as shown in Figure 6-10. Therefore, instability caused by noise and error of neural network approximation can be avoided.

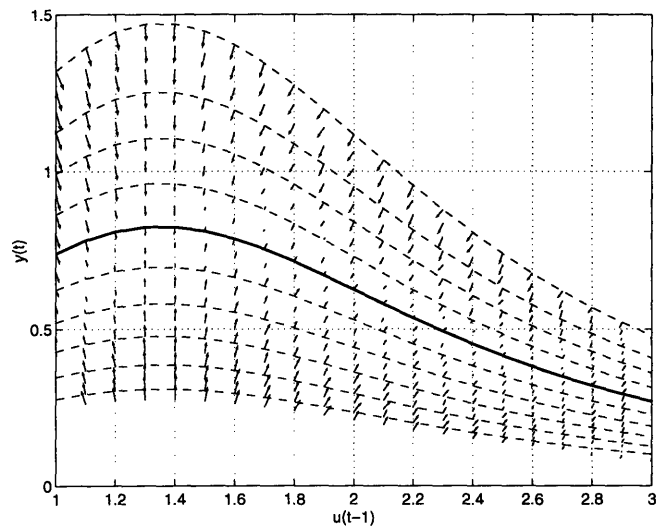


Figure 6-9: Correction of Parameter Estimate $\Delta\theta$ under Noisy Measurement in Example 1

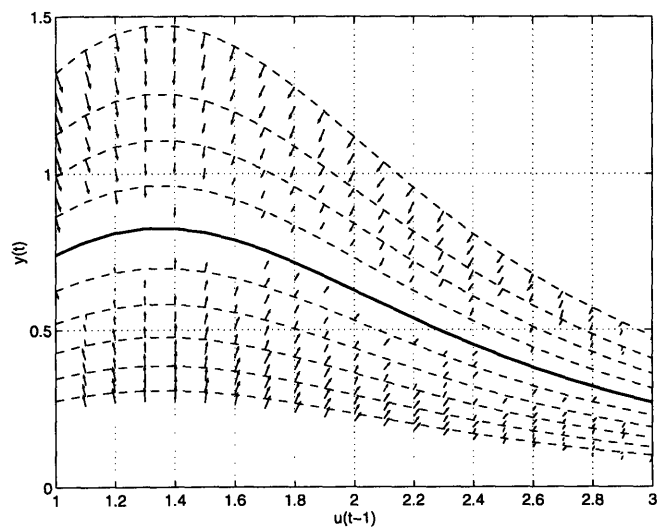


Figure 6-10: Correction of Parameter Estimate $\Delta\theta$ under Noisy Measurement with Dead-zone Modification in Example 1

6.2.2 Example 2: Parameter Estimation for a Regression Model

In this section, the behavior of TANN in the context of a nonlinear system in regression form with two unknown parameters is studied:

$$y_t = \frac{\theta_1 y_{t-1}}{1 + e^{-\theta_1 y_{t-1}^2}} + \theta_2 u_{t-1} + n_t \quad (6.6)$$

where θ_1 and θ_2 are the parameters to be estimated on-line, and are known to be between 1 and 2. The input u of the system is a sinusoidal wave varying between 1 and 3 with a frequency of 0.4 rad/sec. n_t , the measurement noise, is a random signal uniformly distributed between ± 0.05 .

The Block Method

Since there are two unknown parameters in this system, by choosing $\Phi_{t-1} = [y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}]^T$ in Eq. (3.9), the block estimation algorithm can be derived as:

$$\begin{aligned} \begin{Bmatrix} \bar{\theta}_{1t} \\ \bar{\theta}_{2t} \end{Bmatrix} &= N(y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}) \\ \begin{Bmatrix} \hat{\theta}_{1t} \\ \hat{\theta}_{2t} \end{Bmatrix} &= \begin{Bmatrix} \hat{\theta}_{1t-1} \\ \hat{\theta}_{2t-1} \end{Bmatrix} + \rho e^{-\frac{1}{\sigma^2} \|\bar{\theta}_t - \hat{\theta}_{t-1}\|^2} (\bar{\theta}_t - \hat{\theta}_{t-1}) \end{aligned} \quad (6.7)$$

where $\hat{\theta}_t = [\hat{\theta}_{1t}, \hat{\theta}_{2t}]^T$ is the estimate of $[\theta_1, \theta_2]^T$ at time t .

The training set is composed of data points $(y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2)$, where the subscripts of y and u denote their relative causality in time. To form the training set, u_t is chosen to be the same sinusoid as that during on-line estimation, and y_t is determined from Eq. (6.6) for some θ_1 and θ_2 . The process is repeated by varying both θ_1 and θ_2 between 1 and 2. In this example, the training set consists of 2,940 data points. A Gaussian network as shown in Eq. (2.1) with 500 centers and variances $\alpha_i = 1$ is used as the network structure. The network is trained using the recursive least-squares method with inputs $(y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2})$ and targets (θ_1, θ_2) .

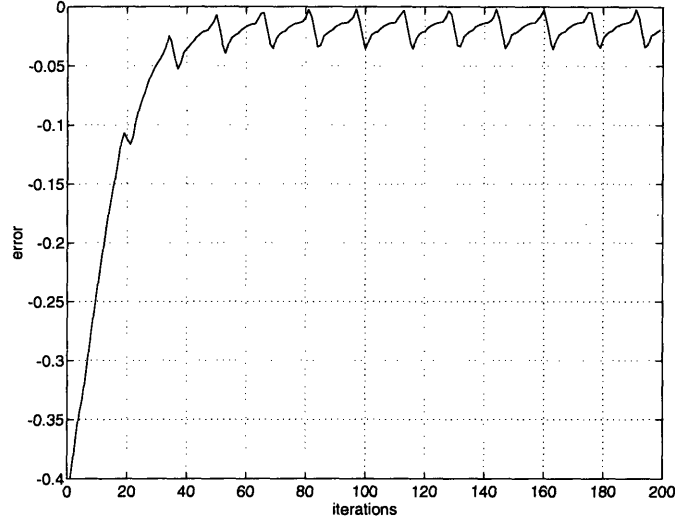


Figure 6-11: Estimation Error $\tilde{\theta}_1$ under Noise-free Measurement Using the Block Method in Example 2

During simulation, ρ and $\frac{1}{\sigma^2}$ in Eq. (6.7) were chosen as 0.1 and 5 respectively. The estimation errors of θ_1 and θ_2 for the system without measurement noise, $n_t = 0$, are shown in Figures 6-11 and 6-12 respectively. For the same system with the output corrupted by noise uniformly distributed between ± 0.05 , the estimation errors of θ_1 and θ_2 are shown in Figures 6-13 and 6-14 respectively.

The Recursive Method

For this example, by substituting $\phi_t = [y_{t-1}, u_{t-1}]$ and $\hat{\theta}_t = [\hat{\theta}_{1t}, \hat{\theta}_{2t}]^T$ into Eq. (3.11), the recursive algorithm becomes:

$$\Delta \hat{\theta}_t = \hat{\theta}_t - \hat{\theta}_{t-1} = \begin{cases} N(y_t, y_{t-1}, u_{t-1}, \hat{\theta}_{1t-1}, \hat{\theta}_{2t-1}) & \text{if } |\tilde{y}_t| > \delta \\ 0 & \text{otherwise} \end{cases}$$

where $\delta = 0.1$, $N(\cdot)$ is a Gaussian network with 600 centers, and $\tilde{y}_t = \hat{y}_t - y_t$.

A typical point in the training set is $(\theta_1, \theta_2, y_{t-1}, u_{t-1}, \hat{\theta}_1, \hat{\theta}_2, y_t, \hat{y}_t)$, where the subscripts of y and u denote their relative causality in time. A complete set, composed of 10,000 data points, is obtained by varying $\theta_1, \theta_2, \hat{\theta}_1$ and $\hat{\theta}_2$ in their respective ranges

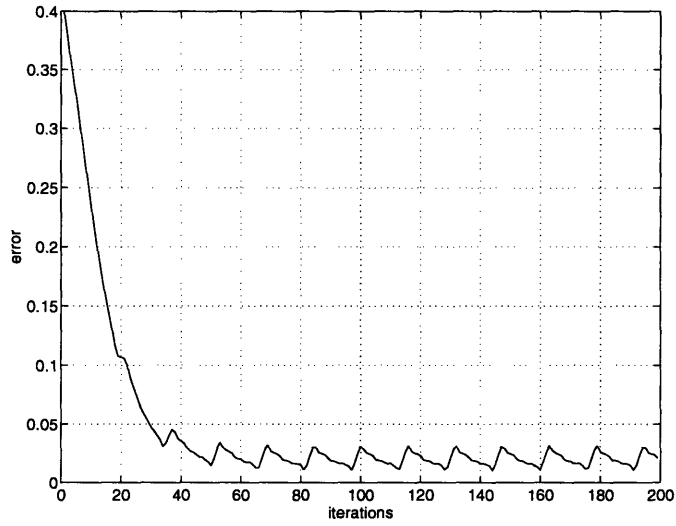


Figure 6-12: Estimation Error $\tilde{\theta}_2$ under Noise-free Measurement Using the Block Method in Example 2

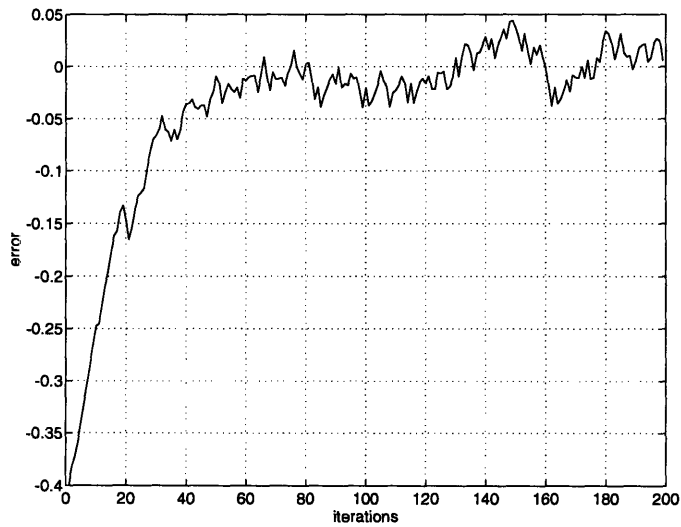


Figure 6-13: Estimation Error $\tilde{\theta}_1$ under Noisy Measurement Using the Block Method in Example 2

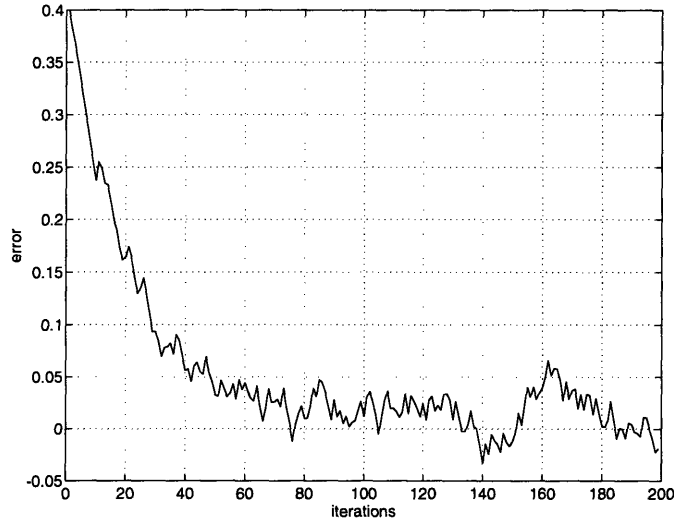


Figure 6-14: Estimation Error $\tilde{\theta}_2$ under Noisy Measurement Using the Block Method in Example 2

and choosing u_t as the specified sinusoidal wave. The corresponding y_t and \hat{y}_t can then be obtained.

For each data point, ΔV and ΔV_d can be calculated as in Eq. (3.15) and (3.16). The resulting ΔV_e is then used to adjust the network. Once again, the network is trained using the Levenberg-Marquardt method.

Under the same situation as the block method, the estimation errors of θ_1 and θ_2 for the system without measurement noise are shown in Figures 6-15 and 6-16 respectively. For the system with noisy measurement, the estimation errors of θ_1 and θ_2 are shown in Figures 6-17 and 6-18 respectively. It can be seen that the parameter estimates stop adapting when the estimation error is small. This is due to that, in order to ensure stability, as $|\tilde{y}| < \delta$, $\Delta \hat{\theta}$ becomes zero according to Eq. (3.11).

6.2.3 Example 3: Parameter Estimation for a Discrete-time State Space Model

For systems in the state space representation, the two methods can be applied equally well even though the corresponding regression form is unknown. In this section, an

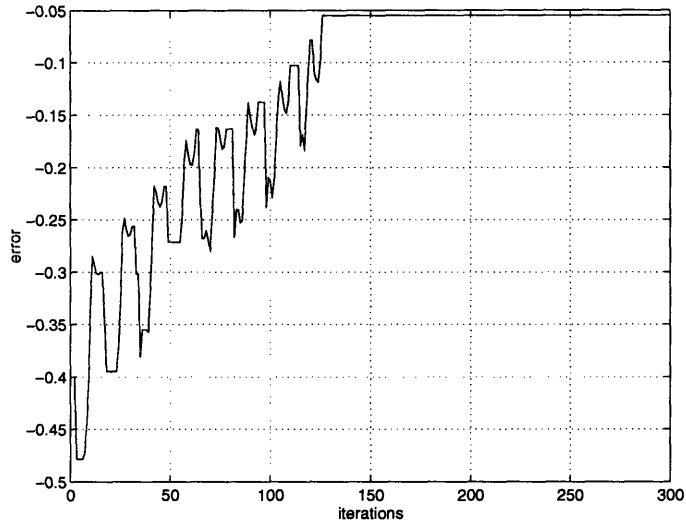


Figure 6-15: Estimation Error $\tilde{\theta}_1$ under Noise-free Measurement Using the Recursive Method in Example 2

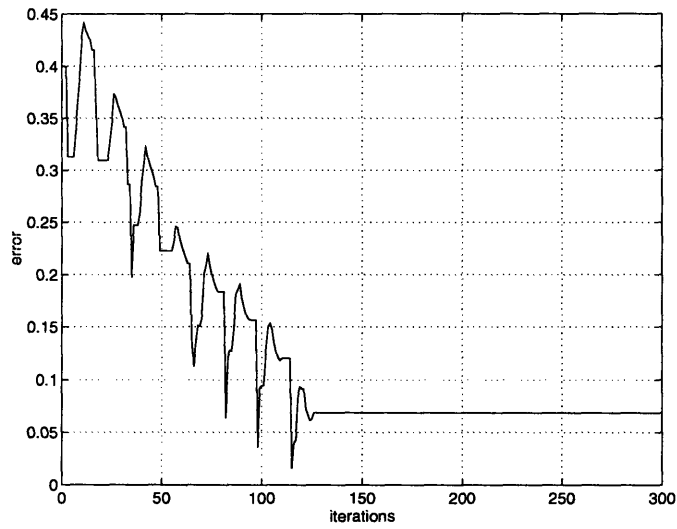


Figure 6-16: Estimation Error $\tilde{\theta}_2$ under Noise-free Measurement Using the Recursive Method in Example 2

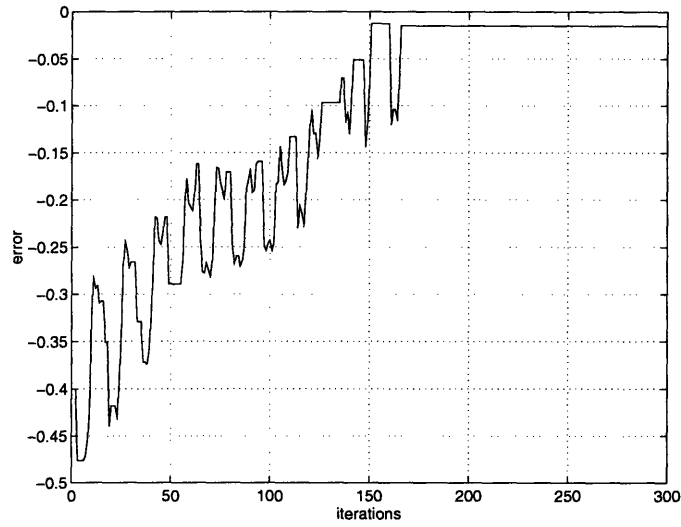


Figure 6-17: Estimation Error $\tilde{\theta}_1$ under Noisy Measurement Using the Recursive Method in Example 2

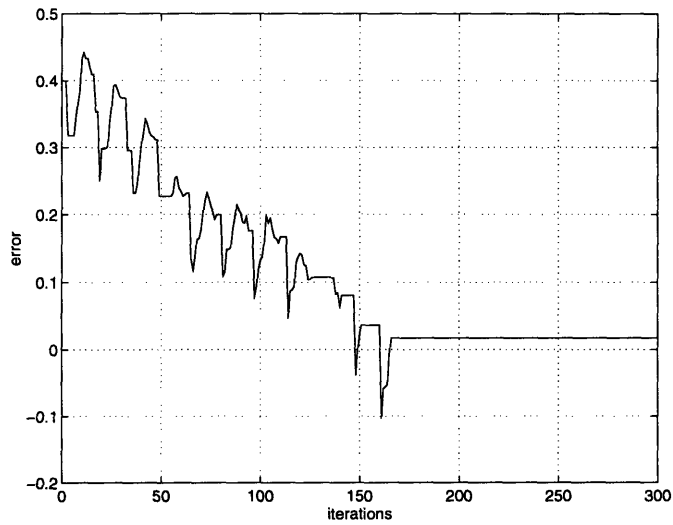


Figure 6-18: Estimation Error $\tilde{\theta}_2$ under Noisy Measurement Using the Recursive Method in Example 2

example of such systems is shown. The system is in the discrete-time state space form with two unknown parameters given by

$$\begin{cases} x_{1,t+1} &= \frac{5\theta_1 x_{1,t}}{1+x_{1,t}^2} + u_t \\ x_{2,t+1} &= x_{1,t} + \frac{5\theta_2 x_{2,t}(1+x_{1,t})}{1+\theta_2 x_{2,t}^2} \end{cases} \quad (6.8)$$

$$y_t = x_{2,t}$$

where θ_1 and θ_2 are the parameters to be estimated on-line, and are known to be within $[0.1, 0.7]$ and $[0.5, 1.5]$ respectively. The input u_t of the system is $u_t = \sin \pi t/10 + \sin \pi t/5$. It is assumed that only y and u can be measured at each instant of time.

Linearized around the equilibrium point $(x_1, x_2) = (0, 0)$, the linear approximation of Eq. (6.8) can be derived as follows:

$$\begin{cases} x_{1,t+1} \\ x_{2,t+1} \end{cases} = \begin{bmatrix} 5\theta_1 & 0 \\ 1 & 5\theta_2 \end{bmatrix} \begin{cases} x_{1,t} \\ x_{2,t} \end{cases} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t$$

$$y_t = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{cases} x_{1,t} \\ x_{2,t} \end{cases}$$

It can be easily checked that the linearized system is both controllable and observable for any θ_1 and θ_2 in their respective range of variation. Therefore, the rank of the Hankel matrix of the linearized system is 2, which is the dimension of the state space. According to [33], Eq. (6.8) can be represented as:

$$y_t = f_r(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2) \quad (6.9)$$

However, the exact representation of f_r for this particular example is unknown.

The Block Method

By choosing $\Phi_{t-1} = [y_t, y_{t-1}, \dots, y_{t-n}, u_{t-1}, \dots, u_{t-n}]^T$ in Eq. (3.9), the block estimation method for Eq. (6.9) can be written as follows:

$$\begin{cases} \bar{\theta}_{1_t} \\ \bar{\theta}_{2_t} \end{cases} = N(y_t, y_{t-1}, \dots, y_{t-n}, u_{t-1}, \dots, u_{t-n}) \quad (6.10)$$

$$\begin{cases} \hat{\theta}_{1_t} \\ \hat{\theta}_{2_t} \end{cases} = \begin{cases} \hat{\theta}_{1_{t-1}} \\ \hat{\theta}_{2_{t-1}} \end{cases} + \rho e^{-\frac{1}{\sigma^2} \|\bar{\theta}_t - \hat{\theta}_{t-1}\|^2} (\bar{\theta}_t - \hat{\theta}_{t-1})$$

where $\hat{\theta}_t = [\hat{\theta}_{1_t}, \hat{\theta}_{2_t}]$ is the estimate of $[\theta_1, \theta_2]$ at time t . The smallest n that can be used in the above algorithm is 3. In order to achieve a smaller error variance, we choose $n = 5$ in the simulation.

The training set is formed by using the input $u_t = \sin \pi t/10 + \sin \pi t/5$ and performing simulations for various θ_1 and θ_2 in the ranges of variation. The resulting training set consists of 9,000 data points of the form $(y_t, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \theta_1, \theta_2)$, where the subscripts of y and u represent their relative causality in time. A similar testing set with 2,000 data points is also constructed for cross-validation. Again, the Gaussian network in Eq. (2.1) with 720 centers is used to represent N in Eq. (6.10). The training is done by using the recursive least-squares and the Levenberg-Marquardt methods to vary weights and variances of the network respectively.

After the network is thoroughly trained, we simulate the system for $\theta_1 = 0.5$ and $\theta_2 = 1$. ρ and σ in Eq. (6.10) are selected as $\rho = 0.05$ and $\frac{1}{\sigma^2} = 10$. The parameter estimation errors for θ_1 and θ_2 are shown in Figures 6-19 and 6-20 respectively. It is seen that the parameter estimation can be obtained even though the corresponding regression form is unknown and the parameters enter nonlinearly. Further reduction in estimation error can be achieved by more training or adding centers to the network.

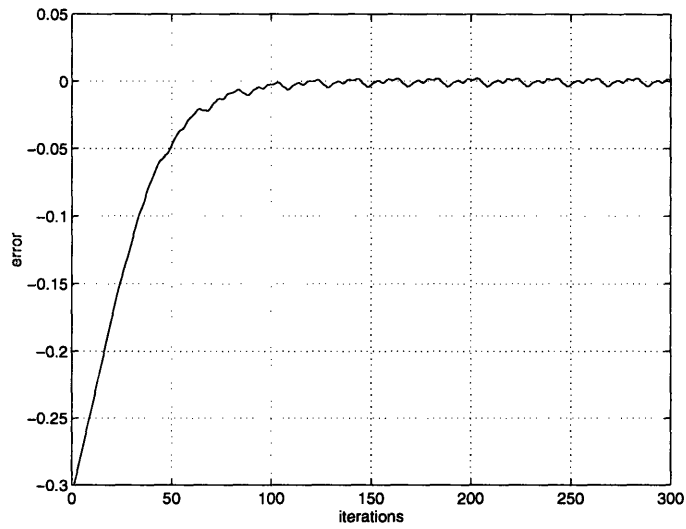


Figure 6-19: Estimation Error $\tilde{\theta}_1$ Using the Block Method in Example 3

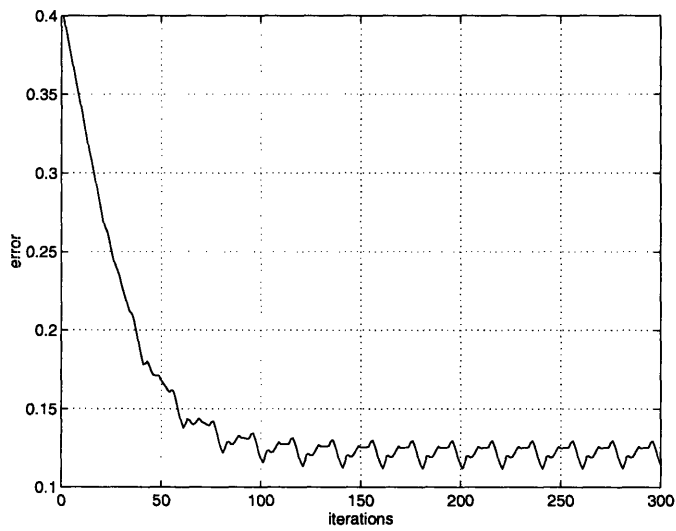


Figure 6-20: Estimation Error $\tilde{\theta}_2$ Using the Block Method in Example 3

The Recursive Method

By substituting $\phi_{t-1} = [y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}]$ into Eq. (3.11), the recursive algorithm for estimating θ_1 and θ_2 in this example, can be written as

$$\Delta \hat{\theta}_t = \hat{\theta}_t - \hat{\theta}_{t-1} = \begin{cases} N(y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2) & \text{if } |\tilde{y}_t| > \delta \\ 0 & \text{otherwise} \end{cases} \quad (6.11)$$

where $\hat{\theta}_t = [\hat{\theta}_{1t}, \hat{\theta}_{2t}]^T$ and $\tilde{y}_t = \hat{y}_t - y_t$.

The training and testing sets, consisting of data points $(\theta_1, \theta_2, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \hat{\theta}_1, \hat{\theta}_2, y_t, \hat{y}_t)$, are constructed similar to those in Section 6.2.2 with θ_1 and $\hat{\theta}_1$ varying between $[0.1, 0.7]$, and θ_2 and $\hat{\theta}_2$ between $[0.5, 1.5]$. The resulting training and testing sets, referred as S_1 and S_2 , contain 25,920 and 1,620 data points respectively.

Since the function f_r in Eq. (6.9) is not explicitly known, in order to calculate $\frac{\partial f_r}{\partial \theta}$ and subsequently ΔV_d , a neural network of the following form is used to model f_r :

$$y_t = N_y(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2)$$

where the inputs of the neural network are $y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1$ and θ_2 . A training set for N_y , constructed similarly to the training sets for the block method, contains 8,000 data points of the form $(y_t, y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2)$. Another set of 500 data points is also formed for cross-validation. We use Eq. (2.1) with 720 centers to model N_y . After N_y is trained, C in Eq. (3.23) is calculated by taking derivative around $[\theta_1, \theta_2] = [0.4, 1]$ through N_y as

$$C(\cdot) = \left(\begin{array}{c} \frac{\partial N_y(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2)}{\partial \theta_1} \\ \frac{\partial N_y(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \theta_1, \theta_2)}{\partial \theta_2} \end{array} \right) \Bigg|_{(\theta_1, \theta_2) = (0.4, 1)}$$

Each data point in S_1 and S_2 is substituted into the above equation to evaluate $C(\cdot)$ vector and then the corresponding ΔV_d in Eq. (3.13).

Because ΔV_d with respect to each data point in S_1 and S_2 are available as described earlier, $N(\cdot)$ in Eq. (6.11) can then be trained as described in the previous examples

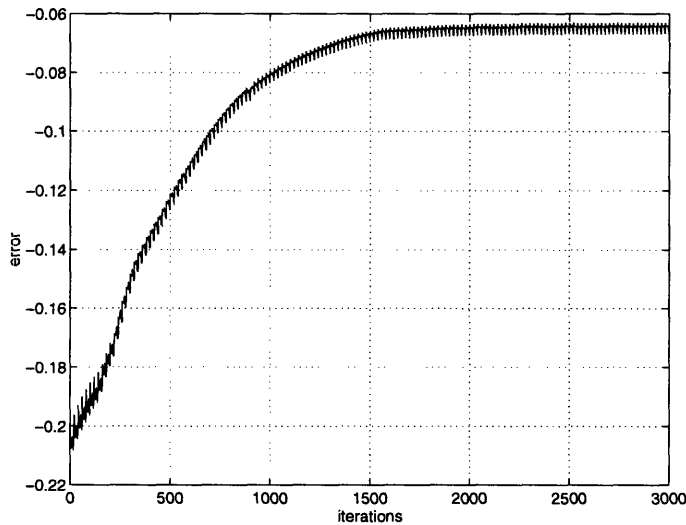


Figure 6-21: Estimation Error $\tilde{\theta}_1$ Using the Recursive Method in Example 3

for the recursive method. In this example, the Gaussian network $N(\cdot)$ in Eq. (2.1) has seven inputs, two outputs and 600 centers. The training is done using the Levenberg-Marquardt method.

We then simulate the recursive method in Eq. (6.11) for the same actual θ_1 and θ_2 as the block method, where δ is chosen as 0.35. In order to compute \tilde{y} in Eq. (6.11), N_y is also implemented on-line to estimate \hat{y} as $N_y(y_{t-1}, y_{t-2}, u_{t-1}, u_{t-2}, \hat{\theta}_{1,t-1}, \hat{\theta}_{2,t-1})$ based on the measurement at every instant. A schematic drawing of this procedure is shown in Figure 3-4. The resulting estimation error for θ_1 and θ_2 are shown in Figures 6-21 and 6-22 respectively.

6.2.4 Example 4: Parameter Estimation for a Continuous-time State Space Model

One of the most commonly encountered models in dealing with parameter estimation for physical systems is in the continuous-time state space form. In such case, the processes of sampling and transformation make the regression representation nonlinear in parameters. Moreover, the corresponding regression relation is extremely difficult to obtain analytically. The example in this section shows how unknown parameters

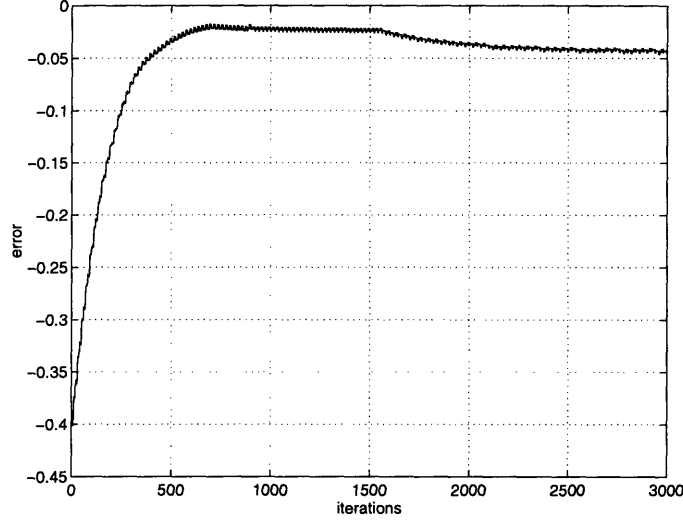


Figure 6-22: Estimation Error $\tilde{\theta}_2$ Using the Recursive Method in Example 3

for this kind of problems can be determined.

The following model corresponds to the dynamics of the two-phase flow heat exchange in an air-conditioning system [19]:

$$\dot{x} = D^{-1}(x)f_a(x, u; \alpha) \quad (6.12)$$

$$y = [0 \ 1 \ 0 \ 0 \ 0]x$$

$$f_a = \begin{bmatrix} \dot{m}_i u - \dot{m}_i h_g + \alpha_{i1} \pi D_i L_1 (T_{w1} - T_{r1}) \\ \dot{m}_o h_g - \dot{m}_o h_o + \alpha_{i2} \pi D_i (L - L_1) (T_{w2} - T_{r2}) \\ \dot{m}_i - \dot{m}_o \\ \alpha_{i1} \pi D_i (T_{r1} - T_{w1}) + \alpha \pi D_o (T_a - T_{w1}) \\ \alpha_{i2} \pi D_i (T_{r2} - T_{w2}) + \alpha \pi D_o (T_a - T_{w2}) \end{bmatrix}$$

In the above equations, $x = [L_1 \ P \ h_o \ T_{w1} \ T_{w2}]^T$ is a state vector; $D(x)$ is a 5 by 5 matrix and depends nonlinearly on x ; $\dot{m}_i, \dot{m}_o, \alpha_{i1}, \alpha_{i2}, D_i, D_o$ and L are all known constants; T_{r1} and T_{r2} are approximated as known linear functions of the states P and h_o , while h_g is proportional to P . Most of the parameters affecting the dynamics of the system can be determined by off-line experiment. However, the equivalent heat

transfer coefficient, α , between the tube wall and the air, varies due to many factors including the ambient temperature and air speed. Therefore, the actual value of α can only be measured on-line. Yet, to carry out this measurement, numerous sensors have to be installed in each machine which makes it somewhat impractical. It is therefore necessary to estimate α from the input and output signals of the system during on-line operation. The prior knowledge of the system is that α is between 56.68 and 66.68 during operation, and the input u is a square wave varying between 39 and 41 with a period of 60 seconds. The sampling interval is 2 seconds.

The Block Method

While α occurs linearly in Eq. (6.12), the corresponding regression form includes α in a nonlinear way. According to Theorem 2.4, Eq. (6.12) can be transformed into a discrete-time state space form with five state variables. If the linearization of the discrete-time state space form does not lose rank, the system can be further transformed into a regression form with five delayed outputs and inputs as

$$y_t = f_r(y_{t-1}, y_{t-2}, \dots, y_{t-5}, u_{t-1}, u_{t-2}, \dots, u_{t-5}; \alpha) \quad (6.13)$$

However, due to complexity of the system, as are most nonlinear systems represented in the continuous-time state space form, the resulting mapping f_r is unknown.

From Eq. (6.13), the block estimation algorithm can be written as:

$$\begin{aligned} \bar{\alpha}_t &= N(y_t, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}) \\ \hat{\alpha}_t &= \hat{\alpha}_{t-1} + \rho e^{-\frac{1}{\sigma^2} \|\bar{\alpha}_t - \hat{\alpha}_{t-1}\|^2} (\bar{\alpha}_t - \hat{\alpha}_{t-1}) \end{aligned}$$

where, in this simulation, N is a Gaussian network with 300 centers and variances equal to 1, $\rho = 0.1$ and $\frac{1}{\sigma^2} = 0.1$.

The training set is composed of 850 data points of the form $(y_t, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \alpha)$, where y and u are sampled every 2 seconds. N is then trained using the set of inputs $(y_t, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5})$ and target α using the recursive

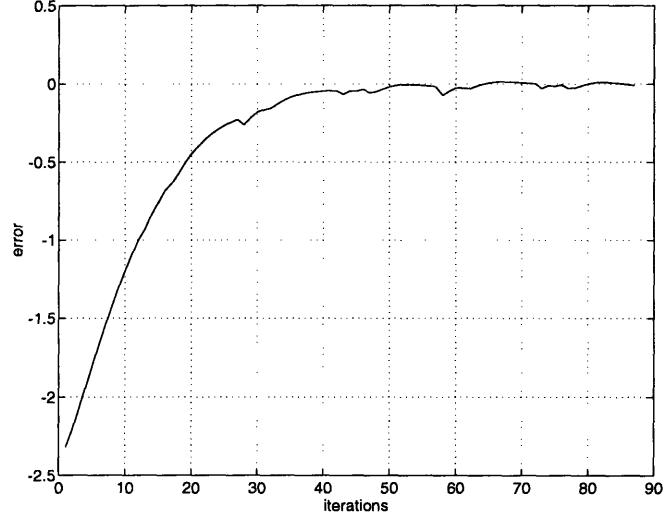


Figure 6-23: Estimation Error $\tilde{\alpha}$ under Noise-free Measurement Using the Block Method in Example 4

least-squares algorithm. The parameter estimation error for the system without measurement noise is shown in Figure 6-23, while that with the output corrupted by noise uniformly distributed between ± 0.05 is shown in Figure 6-24.

The Recursive Method

According to the regression form of the system in Eq. (6.13), the recursive algorithm can be written as

$$\begin{aligned} \hat{\alpha}_t &= \hat{\alpha}_{t-1} + \Delta\hat{\alpha}_t \\ \Delta\hat{\alpha}_t &= \begin{cases} N(y_t, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \hat{\alpha}_{t-1}) & \text{if } |\tilde{y}| > \delta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (6.14)$$

where $N(\cdot)$ is a Gaussian network with 500 centers, and $\delta = 0.1$. The training set is composed of data points $(\alpha, y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \hat{\alpha}, y_t, \hat{y}_t)$. They are obtained by first choosing α and $\hat{\alpha}$ in the specified range of variation, and u_t the square wave obtained from the prior knowledge. The corresponding y_t, \hat{y}_t and u_t are then measured every 2 seconds.

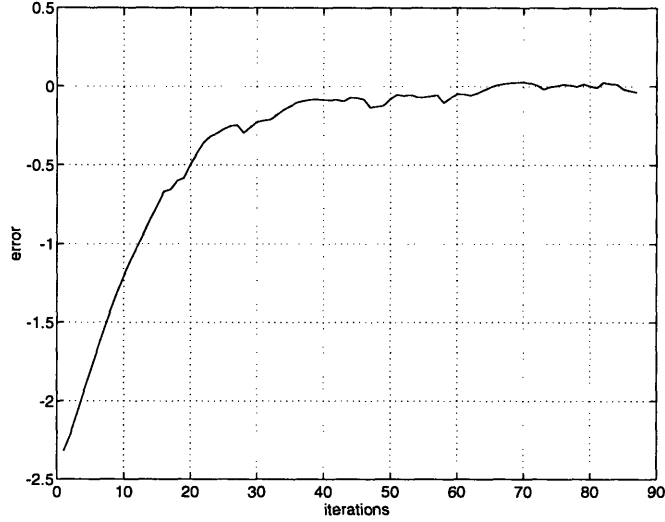


Figure 6-24: Estimation Error $\tilde{\alpha}$ under Noisy Measurement Using the Block Method in Example 4

Since f_r is unknown in Eq. (6.13), in addition to a network used to construct the recursive algorithm in Eq. (6.14), another network N_y is necessary to model f_r in (6.13) in order to calculate $C(\cdot)$ in Eq. (3.13) required during training and \tilde{y} in Eq. (6.14) during on-line operation. The structure of N_y is given by

$$y_t = N_y(y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \alpha)$$

where N_y is a Gaussian network with 500 centers. The training of N_y is achieved by using $(y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \alpha)$ in the training set as inputs and y_t as target. By following the same procedure as that of the recursive method in the previous section, ΔV_d corresponding to each data point can be obtained. N in Eq. (6.14) can then be trained to make $\Delta V \rightarrow \Delta V_d$.

The diagram for the complete on-line process containing the two networks N and N_y is shown in Figure 3-4. As shown in the figure, N_y is used to calculate \hat{y}_t , where

$$\hat{y}_t = N_y(y_{t-1}, \dots, y_{t-5}, u_{t-1}, \dots, u_{t-5}, \hat{\alpha})$$

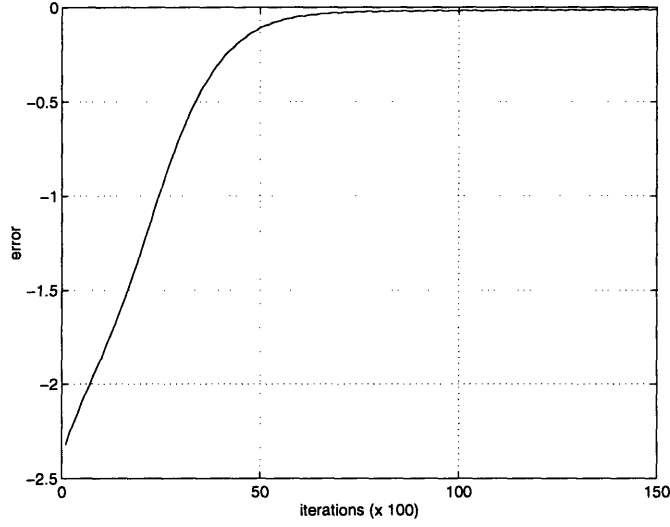


Figure 6-25: Estimation Error $\tilde{\alpha}$ under Noise-free Measurement Using the Recursive Method in Example 4

in order to check $|\tilde{y}_t| > \delta$ in Eq. (6.14). Under the same conditions as the block estimation method, the simulation result for the estimation error of α without measurement noise is shown in Figure 6-25, while that for the system with noisy measurement is shown in Figure 6-26.

6.3 Adaptive Control Using TANN

In this section, we present a simulation example of the control algorithm proposed in Chapter 4. The system is of the form

$$y_{t+1} = \frac{\theta y_t(1 - y_t)}{1 + e^{-0.05\theta y_t}} + u_t$$

where θ is the parameter to be determined on-line, and y_t and u_t are the output and input at time t respectively. Prior information regarding the system is that $\theta \in [4, 10]$. Based on Eq. (4.8), the controller is chosen to be

$$u_t = -\frac{\hat{\theta}_t y_t(1 - y_t)}{1 + e^{-0.05\hat{\theta}_t y_t}} \quad (6.15)$$

where $\hat{\theta}_t$ denotes the parameter estimate at time t . According to Eq. (4.19), θ is estimated using the TANN algorithm with inputs y_{t+1} , y_t , u_t and $\hat{\theta}_t$ as shown below:

$$\hat{\theta}_{t+1} = \begin{cases} \hat{\theta}_t + N(y_{t+1}, y_t, u_t, \hat{\theta}_t) & \text{if } \Delta V_{d_t} < -\epsilon \\ \hat{\theta}_t & \text{otherwise} \end{cases}$$

where $\epsilon = 0.01$. N is a Gaussian network in Eq. (2.1) with 700 centers, and variances $\alpha_1 = 0.51462$, $\alpha_2 = 0.94078$, $\alpha_3 = 22.30859$ and $\alpha_4 = 1.20583$ corresponding respectively to y_{t+1} , y_t , u_t and $\hat{\theta}_t$. Centers of the Gaussian networks are chosen to be the centers of clusters on the training set using the K-means clustering technique [20]. To form the training set and the testing set, we varied y_t between ± 3 , as well as θ and $\hat{\theta}$ between 4 and 10, and formulated data elements of the form $(y_{t+1}, y_t, u_t, \hat{\theta}_t, \theta, \Delta V_{d_t}, L_t)$. However, as discussed in Section 4.4, only those elements with y_{t+1} within ± 3 were retained. The final training set and the testing set were composed of 6,040 and 720 data elements respectively. We then performed training on the sets using the Levenberg-Marquardt method [38].

After the training was completed, we tested the TANN controller on the system with six different values of θ , 4.5, 5.5, 6.5, 7.5, 8.5 and 9.5, while the initial parameter estimate and the initial output were chosen as $\hat{\theta}_1 = 7$ and $y_0 = -0.9$ respectively. The results are plotted in Figure 6-27. It can be seen that y_t can be stabilized at the origin for all these values of θ . For comparison, we also simulated the system under the same conditions using the controller in Eq. (6.15) but with $\hat{\theta}$ determined as follows:

1. $\hat{\theta}_t = 7$
2. the recursive least squares algorithm [18] with

$$\begin{aligned} \hat{\theta}_t &= \hat{\theta}_{t-1} + \frac{P_{t-2}C(y_{t-1}, \theta_0)}{1 + C(y_{t-1}, \theta_0)P_{t-2}C(y_{t-1}, \theta_0)} [u_{t-1} - (y_t + C(y_{t-1}, \theta_0)\hat{\theta}_{t-1})] \\ P_{t-1} &= P_{t-2} - \frac{P_{t-2}C^2(y_{t-1}, \theta_0)P_{t-2}}{1 + C(y_{t-1}, \theta_0)P_{t-2}C(y_{t-1}, \theta_0)} \end{aligned}$$

3. the extended Kalman filter [18] with

$$\begin{aligned}\hat{\theta}_t &= \hat{\theta}_{t-1} + \frac{P_{t-2}C(y_{t-1}, \hat{\theta}_{t-1})}{1+C(y_{t-1}, \hat{\theta}_{t-1})P_{t-2}C(y_{t-1}, \hat{\theta}_{t-1})} \left[u_{t-1} - (y_t + C(y_{t-1}, \hat{\theta}_{t-1})\hat{\theta}_{t-1}) \right] \\ P_{t-1} &= P_{t-2} - \frac{P_{t-2}C^2(y_{t-1}, \hat{\theta}_{t-1})P_{t-2}}{1+C(y_{t-1}, \hat{\theta}_{t-1})P_{t-2}C(y_{t-1}, \hat{\theta}_{t-1})}\end{aligned}$$

where $C(y_{t-1}, \theta') = -\frac{\partial}{\partial \theta} \frac{\theta y(1-y)}{1+e^{-0.05\theta y}} \Big|_{y=y_{t-1}, \theta=\theta'}$. The difference between (2) and (3) is that in the corresponding algorithms, the gradients of the system with respect to θ was evaluated at a fixed point θ_0 in the recursive least squares algorithm, while at the most recent parameter estimate in the extended Kalman filter. Figures 6-28, 6-29 and 6-30 show respectively the output responses of the three methods. It is not surprising that they perform well for certain values of θ . However, for other values of θ , especially when the initial estimation error is large, the responses either diverge or exhibit steady state error. It is worth mentioning that with the TANN controller, although the output converges to zero (see Figure 6-27) for all parameter values, the parameter estimation error of the TANN controller does not, as can be seen from Figure 6-31, which is a common phenomenon in adaptive control. The reason can be observed from Figure 6-32 that when $\hat{\theta}$ converges to a neighborhood of θ , u_t approaches 0, which in turn leads to $\Delta V_{dt} \geq -\epsilon$. As a result, parameter adaptation stops.

Since the system model is affine in control, an adaptive neural controller similar to that in [13] was also implemented. The controller is a RBF network with the input y_t , the output u_t and 700 centers as shown below:

$$u_t = N(y_t; w_r) \tag{6.16}$$

where w_r denotes weights in the network. The centers of the RBF were distributed evenly between ± 20 . During simulation, w_r was updated using the projection algo-

rithm of the following form:

$$w_r(t) = w_r(t-1) + \begin{cases} \frac{R(y_{t-1})}{1+R^T(y_{t-1})R(y_{t-1})} [u_{t-1} - (y_t + R^T(y_{t-1})w_r(t-1))] & \text{if } |e_t| > \Delta \\ 0 & \text{otherwise} \end{cases} \quad (6.17)$$

where $w_r(t)$ is the value of w_r at time t , $\Delta = 0.01$ is the width of the dead-zone and $R(\cdot)$ is the output vector from the centers of the RBF. The simulation results are shown in Fig. 6-33. It can be seen that the transient responses of the adaptive neural controller for the same values of θ are much worse than those of the TANN controller due to the fact that significantly more number of parameters were varied on-line. For the same neural controller, we performed a second simulation by first training the network extensively off-line before implementation to learn the mapping from y_t to $-\frac{\theta y_t(1-y_t)}{1+e^{-0.05\theta y_t}}$ for $\theta = 7$ and y_t between ± 20 . Such training set is composed of 3,000 patterns. The neural controller was then implemented under the same conditions as above, with weights adjusted on-line using the projection algorithm in (6.17). The results are shown in Fig. 6-34. Since the RBF network had been trained for $\theta = 7$, the performance of the adaptive neural controller and the TANN controller is comparable around $\theta = 7$ as expected. However, for $\theta = 4.5$, large transients occurred in the simulation.

6.4 Stable Neural Controller

The stable neural controller presented in Chapter 5 is demonstrated in this section through a simulation example. The neural controller is also compared with a linear controller to illustrate the difference. This example concerns a second-order nonlinear system:

$$x_t = f(x_{t-1}, u_{t-1}) \quad (6.18)$$

where

$$f = \begin{bmatrix} x_{1t-1}(1 + x_{2t-1}) + x_{2t-1}(1 - u_{t-1} + u_{t-1}^2) \\ x_{1t-1}^2 + 2x_{2t-1} + u_{t-1}(1 + x_{2t-1}) \end{bmatrix}$$

u and $x = [x_1, x_2]^T$ denote the input and the state vector respectively. It is assumed that x is measurable, and we wish to stabilize the system around the origin. It can be verified that the above system is not feedback-linearizable and no other stabilizing nonlinear controllers can be found by inspection.

A neural controller of the following form is used to stabilize Eq. (6.18):

$$u_t = N(x_{1_t}, x_{2_t})$$

where N is a Gaussian network with inputs x_{1_t} and x_{2_t} , output u_t , 289 centers, and variances α_1 and α_2 both equal to 0.0120417. The linearized system is given by

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t \\ &= \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix} x_{t-1} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_{t-1} \end{aligned}$$

A choice of $K = [0.1429, 1.7752]$ results in the eigenvalues of $(A - BK)$ being within the unit circle. By choosing Q as an identity matrix, we can calculate the matrix P in the discrete-time Lyapunov equation in (5.12). The training set was constructed by sampling both x_1 and x_2 between ± 0.2 with a total of 961 points. We also formed a testing set composed of 121 points distributed in the same range. The network was then trained using the Levenberg-Marquardt method to make $\Delta V \leq -x^T Q x$ at all points in the training set, where $\Delta V = f^T(x, N(x; W)) P f(x, N(x; W)) - x^T P x$.

After the training was finished, the actual changes of the function, ΔV , using the linear controller $u = -Kx$ and the neural controller were plotted in Figures 6-35 and 6-36 respectively. It can be observed from the two figures that if the neural controller is used, ΔV is negative definite except in a small neighborhood of the origin, which, according to Theorem 2.7, assures that the closed-loop system would converge to vicinity of the origin; whereas, if the linear controller is used, ΔV becomes positive in some region away from the origin, which implies that the system may be unstable for some initial conditions. The reason for larger region of stability of the neural controller can be seen from Figures 6-37 and 6-38. As shown in Figure 6-

37, the linear control function is restricted to a plane in the space. On the other hand, the neural controller allows the control input to be nonlinear. As a result, the neural controller can stabilize regions where the linear controller fails. Figure 6-38 shows the difference of the input signals between the linear controller and the neural controller with respect to the state vector x . It can be seen that nonlinearity has the most effect in the region where the linear controller fails to stabilize. Simulation results for an initial state located in the region where ΔV of the linear controller is positive are shown in Figures 6-39 and 6-40 for the linear and the neural controllers respectively. Instability of the closed-loop system using the linear controller confirms our observation.

6.5 Summary and Remarks

In this chapter, TANN for parameter estimation and adaptive control as well as the stable neural controller introduced in the last three chapters are verified through extensive computer simulations. In the examples of parameter estimation, different classes of system models where unknown parameters relate nonlinearly to the output are simulated using both the block and the recursive methods of TANN. For systems with uncertain parameters, the TANN controller can be used to stabilize such systems even though these parameters enter nonlinearly. Comparisons of the TANN controller with other linear schemes are also made in the simulation examples. On the other hand, when the controller structure itself is unknown, the simulation result of the neural controller demonstrates superior performance compared to a linear controller.

In the parameter estimation using TANN, if the system model is represented in the state-space form, the corresponding regression model might not be available, such as those in Examples 3 and 4 of Section 6.2. For this class of problems, another network N_y is required to model the input-output map of the system. The purpose of the additional network is to compute ΔV_d during the training phase and \hat{y} during the on-line estimation phase. Training of such networks is similar to that appearing in other approaches (such as [43]) utilizing neural networks as model structures. The

only difference is that, in addition to past inputs and outputs of the system, the unknown parameters become part of the inputs to the neural network as well. In other words, the network N_y can predict the output of the system not only for a particular parameter value but also for every value in its range of variation. This is required when we compute \hat{y} and thus, \tilde{y} in the on-line phase, since \hat{y} corresponding to various parameter estimates has to be readily available.

All of the simulations done in this chapter use the Gaussian network as the model structures. Nevertheless, they can also be performed, for example, using MNN since they satisfy the universal approximator property as well. If a MNN is used instead of a Gaussian network, the procedure of forming the initial network structure is simplified since we do not have to worry about the locations of the centers. Another benefit expected to be gained from using a MNN is reduction in complexity of the network structure. For a Gaussian network, the contribution of each Gaussian function to the outputs is limited to the region of radius approximately four times the standard deviation of the Gaussian function. Therefore, the number of Gaussian functions required to approximate a function within certain error bound grows exponentially as the dimensions of the inputs. In contrast, the number of sigmoidal terms in a MNN increases at a much slower rate for functions bounded on a spectral norm, according to [6]. Therefore, we expect fewer neurons to be needed if MNN were used in the simulation. On the other hand, deterioration in the speed of convergence during training especially for the block method would probably occur if a MNN is used. The weights of a Gaussian network are linearly related to the desired targets in the block method of TANN. Therefore, linear algorithms such as the least-squares method can be applied to adjust weights of the network. This results in fast convergence during training. However, the weights and biases in the hidden layers of a MNN are nonlinear in the outputs. Therefore, only nonlinear optimization schemes such as the gradient descent and the Newton methods can be used. The rate of convergence of those algorithms is usually slower. Thus, tradeoff between complexity and training speed has to be kept in mind when choosing an appropriate network structure.

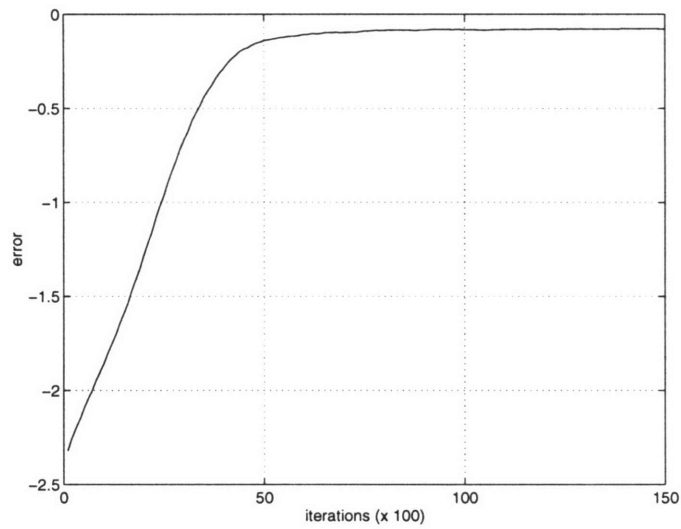


Figure 6-26: Estimation Error $\tilde{\alpha}$ under Noisy Measurement Using the Recursive Method in Example 4

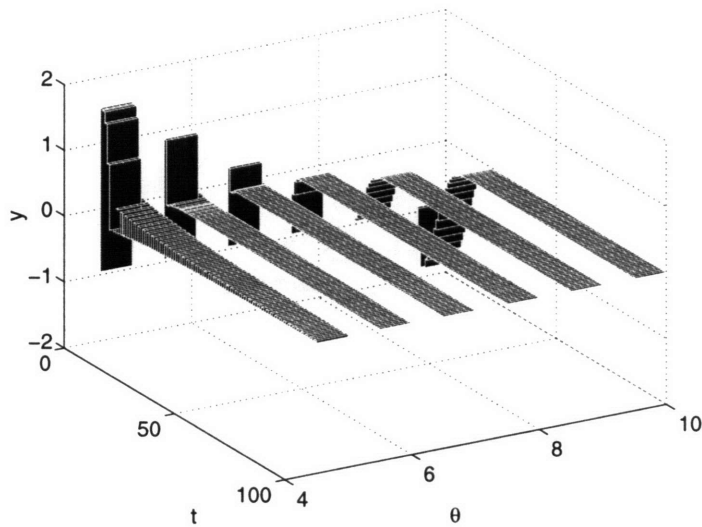


Figure 6-27: y_t (Output of the Closed-loop System with the TANN Controller)

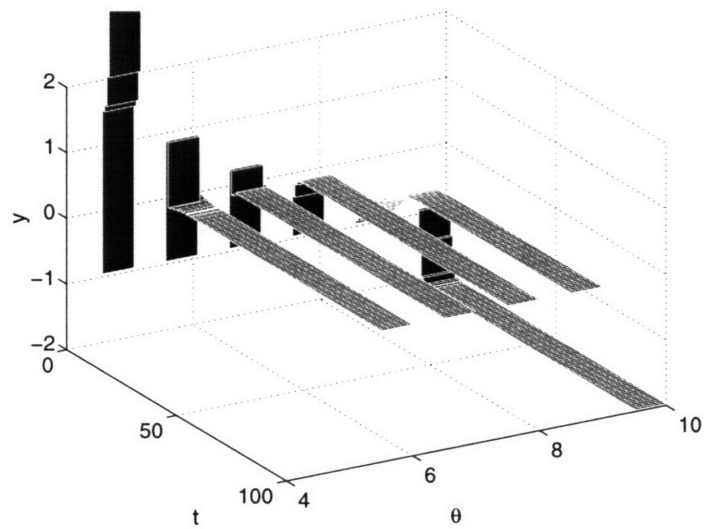


Figure 6-28: y_t (Output of the Closed-loop System with the Parameter Estimate Fixed at $\hat{\theta} = 7$)

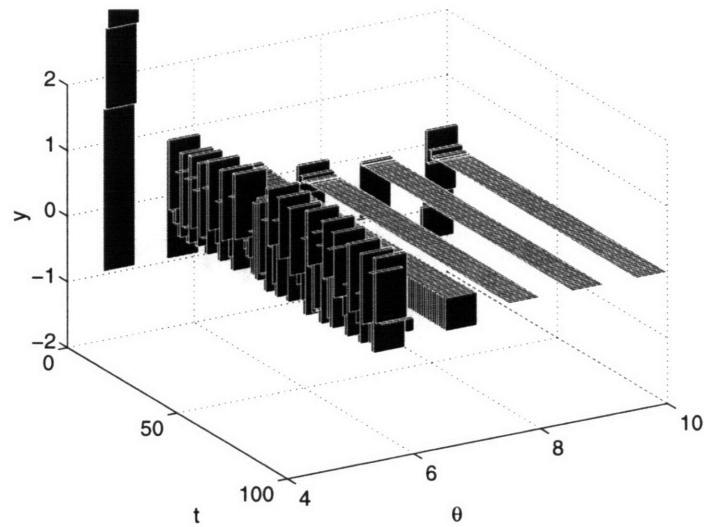


Figure 6-29: y_t (Output of the Closed-loop System with $\hat{\theta}_t$ Determined by the Recursive Least Squares Algorithm)

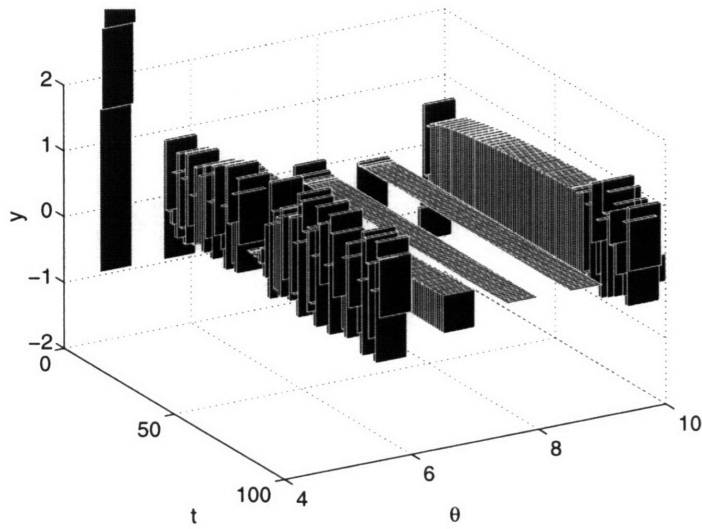


Figure 6-30: y_t (Output of the Closed-loop System with $\hat{\theta}_t$ Determined by the Extended Kalman Filter)

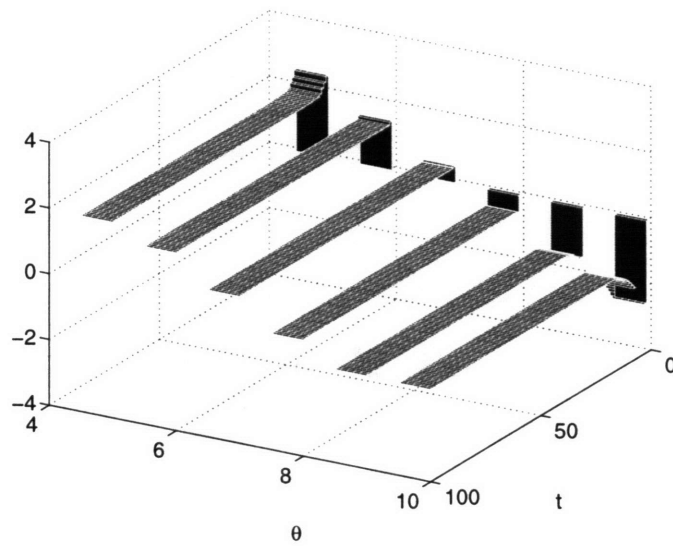


Figure 6-31: $\tilde{\theta}_t$ (Parameter Estimation Error of the TANN Controller)

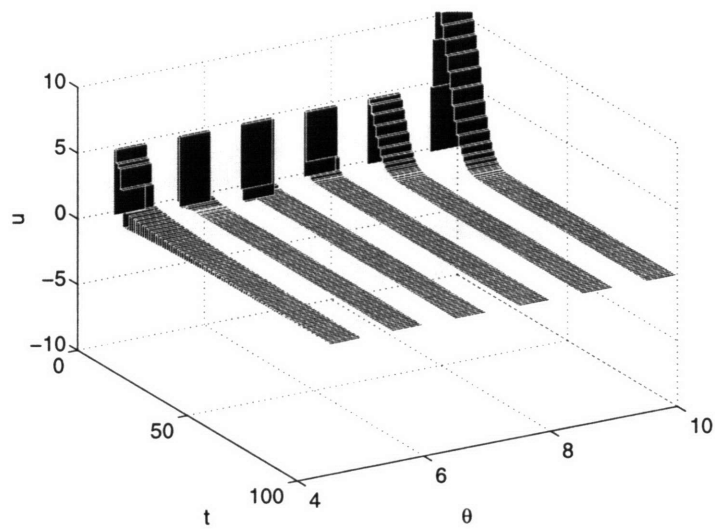


Figure 6-32: u_t (TANN Control Signal)

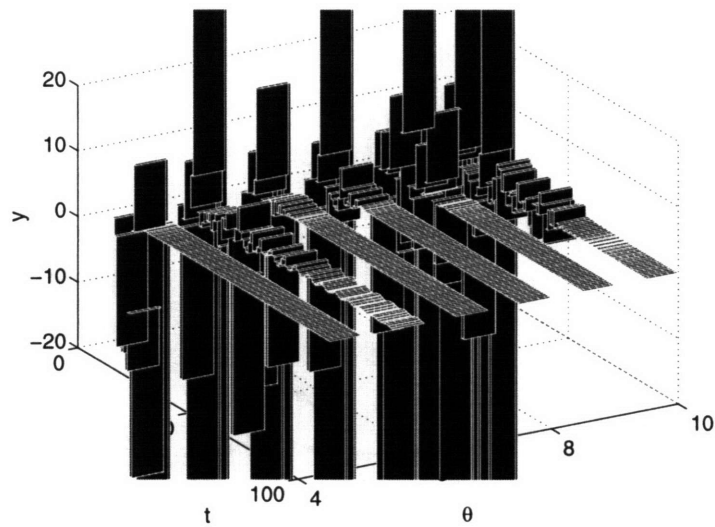


Figure 6-33: y_t (Output of the Closed-loop System with the Adaptive Neural Controller in Eq. (6.16) — Without Off-line Training)

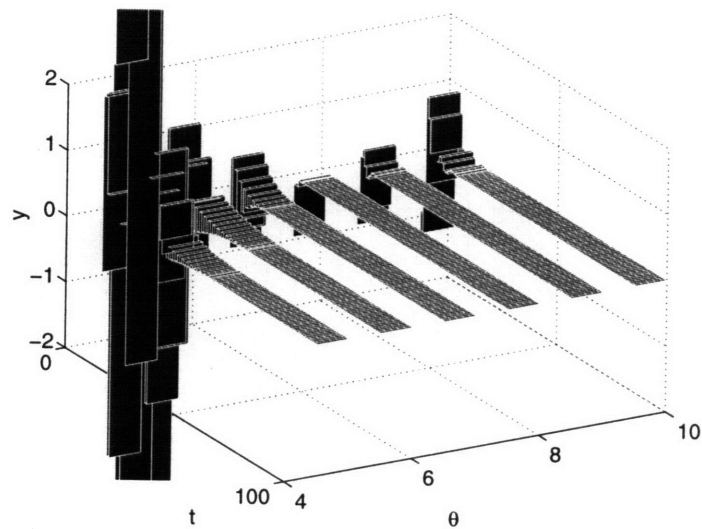


Figure 6-34: y_t (Output of the Closed-loop System with the Adaptive Neural Controller in Eq. (6.16) — With Off-line Training)

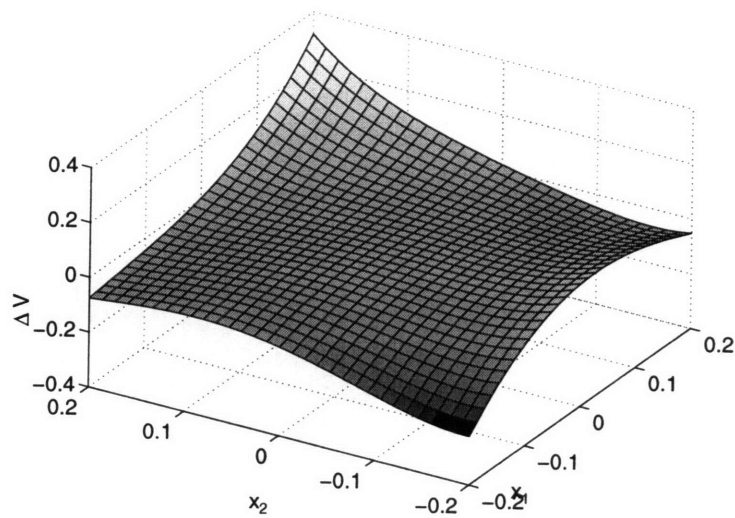


Figure 6-35: $\Delta V(x)$ for the Linear Controller $u = -Kx$

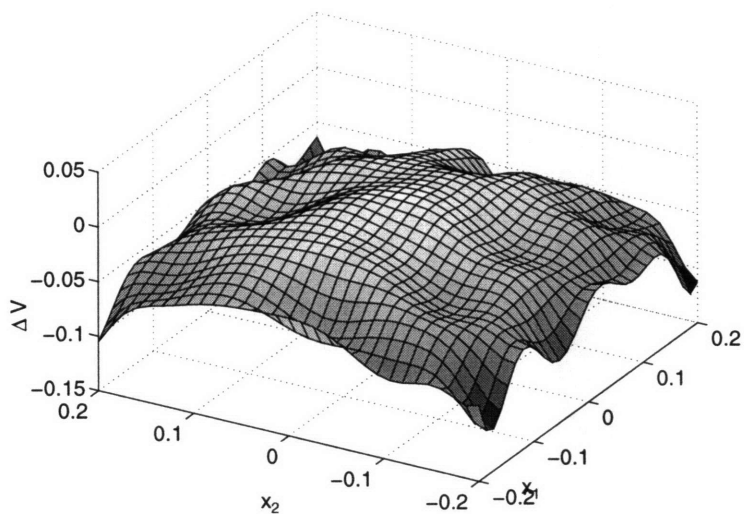


Figure 6-36: $\Delta V(x)$ for the Stable Neural Controller $u = N(x)$

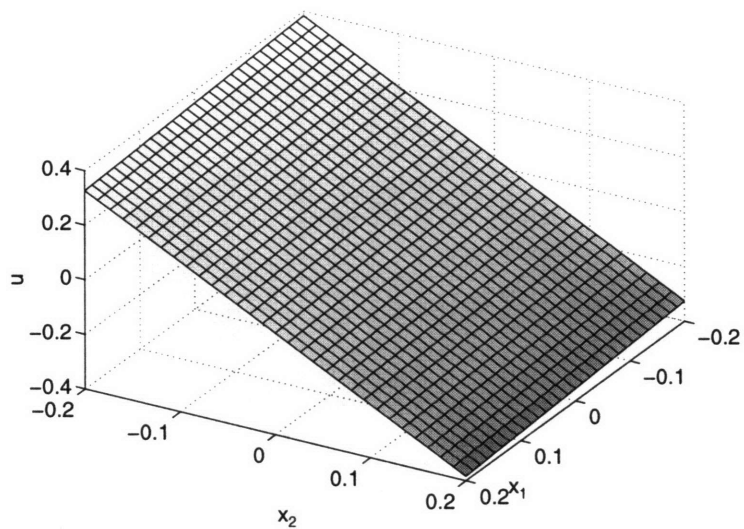


Figure 6-37: Control Surface for the Linear Controller $u = -Kx$

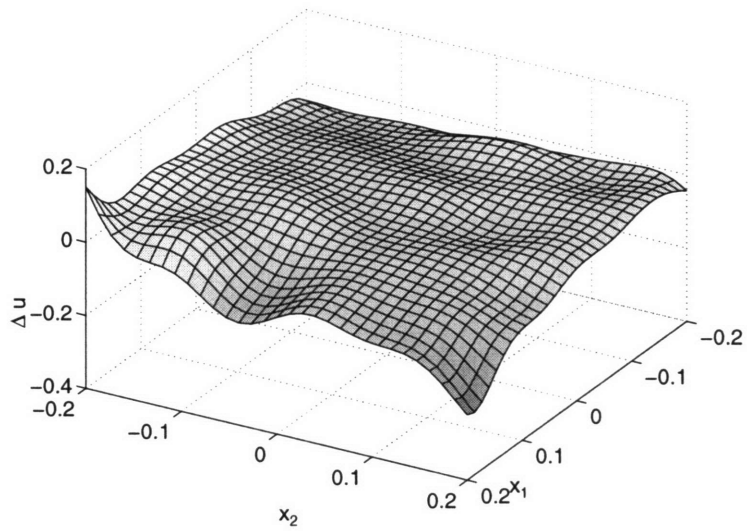


Figure 6-38: Difference of the Linear and Stable Neural Control Signals $\Delta u(x)$ ($= N(x) - (-Kx)$)

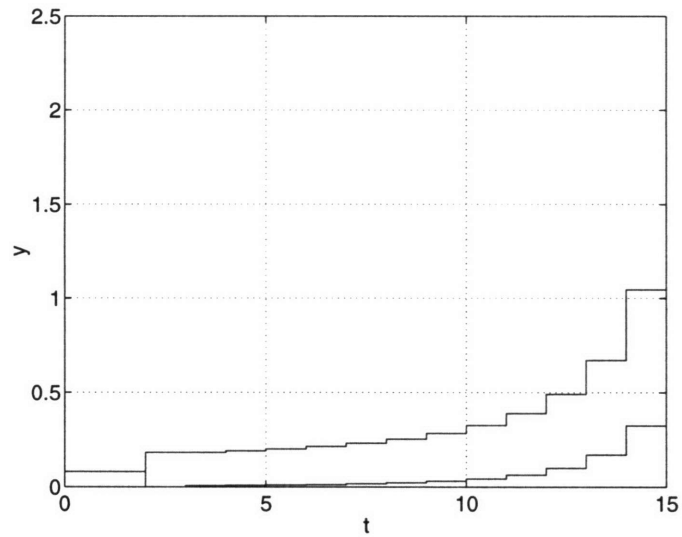


Figure 6-39: State Response x_t of the Closed-loop System Using the Linear Controller

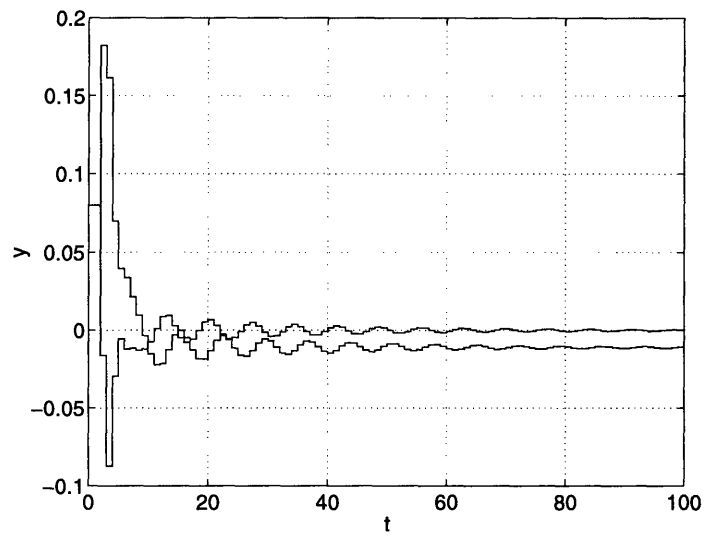


Figure 6-40: State Response x_t of the Closed-loop System Using the Stable Neural Controller

Chapter 7

Conclusion

In this thesis, algorithms are developed using ANN for parameter estimation and control of nonlinear systems. A model-based approach is taken to solve the estimation and control problems. The resulting algorithms are applicable to either discrete-time or continuous-time models in regression and state space forms, which are among the most common model representations in engineering problems. The nonlinear maps that the neural networks attempt to approximate are the nonlinear functions that would satisfy the specified stability indices, rather than explicit input to output mappings. These indices are positive definite functions of parameter estimation errors if the problem under consideration pertain to identification and adaptive control; they are the change of the Lyapunov function candidates if the problems are related to control design. The unified approach in achieving this is formulating the training of neural networks as optimization problems with these indices as either cost functions or constraints. This approach naturally lends itself to a framework that is transparent in determining stability of the resulting algorithms, with the approximating error directly linked to the stability criteria.

An important aspect of the methods proposed in this thesis is that they are all developed for partially known system models as opposed to the black box approach often taken in system identification and control using ANN. The tradeoff is, of course, that some knowledge about the systems is required. Nevertheless, in many engineering problems, analytical models can usually be obtained based on physical laws, and de-

signers are reluctant to replace them by black boxes. What is often needed are means to improve performance by incorporating better knowledge and designing better controllers for the systems. Explicit knowledge of the system models allows flexibility in designing controllers. Furthermore, by using physical models to parametrize the systems, the parameters can be associated closely with physical constants. The parameter estimation and control schemes developed in this thesis, which make full use of the analytical models, can take advantage of them—the TANN controller stabilizes systems under nonlinear parametric uncertainty, while, for systems where a nonlinear controller itself is hard to obtain, the stable neural controller method points a new way of constructing such controllers.

The work in this thesis can be extended in several directions. They are enumerated as follows:

1. The TANN control algorithm is developed for adaptive stabilization of nonlinear systems. For systems where tracking is the major concern, more work still needs to be done in solving these problems. Moreover, the training procedure as well as how the new training set should be constructed require further investigation.
2. The assumption on the systems for which a neural controller can be constructed is that a static controller exists to stabilize the linearized system. This is more restrictive than the stabilizable and detectable conditions. In order to relax the assumption, dynamic compensators are necessary. Hence, the neural controller can be coupled with a linear filter, an extended Kalman filter or a nonlinear filter. As the complexity of the filters increase, the applicable region of the dynamic compensators should also expand. However, how those filters should be designed to ensure closed-loop stability remains to be seen.
3. Whether the problems in this thesis are parameter estimation or control, the positive definite functions quantifying the performance indices are always chosen first. In the control problems, for example, it implies choosing a Lyapunov function candidate. Even though the way the functions are determined guarantees existence of either a stable parameter estimation algorithm or a neural

controller, different choices of the positive definite function may lead to better performance or larger applicable region. Hence, other alternative functions or even constructing these functions by training are worth investigating.

Bibliography

- [1] L. Acosta, A. Hamilton, L. Moreno, J. L. Sanchez, J. D. Pineiro, and J. A. Mendez. Two approaches to nonlinear systems optimal control by using neural networks. In *IEEE International Conference on Neural Networks*, volume 7, pages 4583–4587, 1994.
- [2] A. M. Annaswamy and S. Yu. θ -adaptive neural networks: A new approach to parameter estimation. *IEEE Transactions on Neural Networks*, (to appear) 1996.
- [3] Jaipaul K. Antony and Levent Acar. Real-time nonlinear optimal control using neural networks. In *Proceedings of the 1994 American Control Conference*, volume 3, pages 2926–2930, 1994.
- [4] Z. Artstein. Stabilization with relaxed controls. *Nonlinear Analysis*, 7:1163–1173, 1983.
- [5] R. S. Baheti, R. R. Mohler, and III H. A. Spang. Second-order correlation method for bilinear system identification. *IEEE Transactions on Automatic Control*, AC-25(6):1141–1146, December 1980.
- [6] Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.
- [7] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, Inc., New York, 1982.

- [8] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.
- [9] Naveen V. Bhat, Jr. Peter A. Minderman, Thomas AcAvoy, and Nam Sun Wang. Modeling chemical process systems via neural computation. *IEEE Control Systems Magazine*, pages 24–30, April 1990.
- [10] S. A. Billings, H. B. Jamaluddin, and S. Chen. Properties of neural networks with applications to modelling non-linear dynamical systems. *International Journal of Control*, 55(1):193–224, 1992.
- [11] S. A. Billings and W. S. F. Voon. Least squares parameter estimation algorithms for non-linear systems. *Int. J. Systems Science*, 15(6):601–615, 1984.
- [12] Fu-Chuang Chen and Hassan K. Khalil. Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Transactions on Automatic Control*, 40(5):791–801, May 1995.
- [13] Fu-Chuang Chen and Chen-Chung Liu. Adaptively controlling nonlinear continuous-time systems using multilayer neural networks. *IEEE Transactions on Automatic Control*, 39(6):1306–1310, June 1994.
- [14] S. Reynold Chu, Rahmat Shoureshi, and Manoel Tenorio. Neural networks for system identification. *IEEE Control Systems Magazine*, pages 31–35, April 1990.
- [15] G Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
- [16] Farhat Fnaiech and Lennart Ljung. Recursive identification of bilinear systems. *International Journal of Control*, 45(2):453–470, 1987.
- [17] M. M. Gabr and T. Subba Rao. On the identification of bilinear systems from operating records. *International Journal of Control*, 40(1):121–128, 1984.
- [18] Graham C. Goodwin and Kwai Sang Sin. *Adaptive Filtering Prediction and Control*. Prentice-Hall, Inc., 1984.

- [19] Xiandong He, Sheng Liu, and Haruhiko Asada. A moving-interface model of two-phase flow heat exchanger dynamics for control of vapor compression cycles. In *AES-Vol. 32*, pages 69–75. ASME Winter Annual Meeting, 1994.
- [20] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.
- [21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [22] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, New York, NY, 2nd edition, 1989.
- [23] Andrew H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, 1970.
- [24] Liang Jin, Peter N. Nikiforuk, and Madan M. Gupta. Fast neural learning and control of discrete-time nonlinear systems. *IEEE Transactions on Systems, Man and Cybernetics*, 25(3):478–488, March 1995.
- [25] Michael I. Jordan. Lecture notes of 9.641 connectionist models of cognitive processes, Spring 1993.
- [26] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.
- [27] I. Kanellakopoulos, P.V. Kokotovic, and A.S. Morse. “Systematic design of adaptive controllers for feedback linearizable system”. *IEEE Transactions on Automatic Control*, 34:1241–1253, November 1991.
- [28] Ioannis Kanellakopoulos. A discrete-time adaptive nonlinear system. *IEEE Transactions on Automatic Control*, 39(11):2362–2365, November 1994.
- [29] S. Kawaji, T. Maeda, and N. Matsunaga. Learning control of an inverted pendulum using neural networks. In *Proceedings of the 31st Conference on Decision and Control*, pages 2734–2739, December 1992.

- [30] G. Kreisselmeier and K.S. Narendra. Stable model reference adaptive control in the presence of bounded disturbances. *IEEE Transactions on Automatic Control*, 27:1169–1175, December 1982.
- [31] J.P. LaSalle. “Asymptotic stability criteria”. In *Proceedings of Symposium on Appl. Math.*, pages 299–307, Providence, R.I., vol. 13 1962. Hydrodynamic Instability.
- [32] James A. Leonard and Mark A. Kramer. Radial basis function networks for classifying process faults. *IEEE Control Systems Magazine*, pages 31–38, April 1991.
- [33] I. J. Leontaritis and S. A. Billings. Input-output parametric models for nonlinear systems part I: Deterministic non-linear systems. *International Journal of Control*, 41(2):303–328, 1985.
- [34] Asriel U. Levin and Kumpati S. Narendra. Control of nonlinear dynamical systems using neural networks: Controllability and stabilization. *IEEE Transactions on Neural Networks*, 4(2):192–206, March 1993.
- [35] Frank L. Lewis, Kai Liu, and Aydin Yesildirek. Neural net robot controller with guaranteed tracking performance. *IEEE Transactions on Neural Networks*, 6(3):703–715, May 1995.
- [36] R. P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1:1–38, 1989.
- [37] Lennart Ljung. *System Identification: Theory for the User*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
- [38] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, June 1963.

- [39] Antonio Moran and Masao Nagai. Optimal active control of nonlinear vehicle suspensions using neural networks. *JSME International Journal, Series C: Dynamics, Control, Robotics, Design and Manufacturing*, 37(4):707–718, December 1994.
- [40] James R. Munkres. *Analysis on Manifolds*. Addison-Wesley Publishing Company, 1991.
- [41] K.S. Narendra and A.M. Annaswamy. Robust adaptive control in the presence of bounded disturbances. *IEEE Transactions on Automatic Control*, 31:306–315, April 1986.
- [42] Kumpati S. Narendra and Anuradha M. Annaswamy. *Stable Adaptive Systems*. Prentice-Hall, Inc., 1989.
- [43] Kumpati S. Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–26, March 1990.
- [44] Derrick H. Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10:18–23, April 1990.
- [45] P.V. Osburn, H.P. Whitaker, and A. Kezer. New developments in the design of model reference adaptive control systems. In *Proceedings of the IAS 29th Annual Meeting*, New York, New York, 1961.
- [46] T. Parisini and R. Zoppoli. Neural networks for feedback feedforward nonlinear control systems. *IEEE Transactions on Neural Networks*, 5(3):436–449, May 1994.
- [47] J. Park and I. W. Sandberg. Universal approximation using radial-basis function networks. *Neural Computation*, 3:246–257, 1991.
- [48] P.C. Parks. Liapunov redesign of model reference adaptive control systems. *IEEE Transactions on Automatic Control*, 11:362–367, 1966.

- [49] B.B. Peterson and K.S. Narendra. Bounded error adaptive control. *IEEE Transactions on Automatic Control*, 27:1161–1168, December 1982.
- [50] Gerald Peterson, William Bond, Roger Germann, Barry Streeter, and James Umes. Using neural networks for aerodynamic parameter modeling. In *Proceedings of the American Control Conference*, pages 1360–1361, June 1995.
- [51] Marios M. Polycarpou and Petros A. Ioannou. Neural networks as on-line approximators of nonlinear systems. In *Proceedings of the 31st Conference on Decision and Control*, volume 1, pages 7–12, December 1992.
- [52] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.
- [53] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, Inc., 1976.
- [54] Nader Sadegh. A perceptron network for functional identification and control of nonlinear systems. *IEEE Transactions on Neural Networks*, 4(6):982–988, November 1993.
- [55] Robert M. Sanner and Jean-Jacques E. Slotine. Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, 3(6):837–863, November 1992.
- [56] Danbing Seto, Anuradha M. Annaswamy, and J. Baillieul. Adaptive control of nonlinear systems with a triangular structure. *IEEE Transactions on Automatic Control*, 39(7):1411–1428, July 1994.
- [57] Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended kalman algorithm. In *Advances in Neural Information Processing*, pages 133–140. Morgan-Kaufmann, 1989.
- [58] J. Sjöberg, H. Hjalmarsson, and L. Ljung. Neural networks in system identification. In *10th IFAC Symposium on System Identification*, volume 2, pages 49–72, 1994.

- [59] Fredrik Petter Skantze and Anuradha M. Annaswamy. Parameter estimation for systems with partial nonlinear parameterization using θ -adaptive neural networks. In *Proceedings of the 2nd Joint Conference on Information Sciences*, Wrightsville Beach, NC, September 1995.
- [60] E. D. Sontag. *Mathematical Control Theory*. Springer-Verlag New York, Inc., 1990.
- [61] Eduardo D. Sontag. A ‘universal’ construction of artstein’s theorem on nonlinear stabilization. *Systems and Control Letters*, 13:117–123, 1989.
- [62] P. Stoica and T. Söderström. Asymptotic behavior of some bootstrap estimators. *International Journal of Control*, 33:433–454, 1981.
- [63] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

4536-11