Robust neural training and pruning algorithms for a class of nonlinear tracking control systems

Ni, Jie

2006

Ni, J. (2006). Robust neural training and pruning algorithms for a class of nonlinear tracking control systems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4973>

<https://doi.org/10.32657/10356/4973>

Nanyang Technological University

Downloaded on 26 Apr 2023 16:31:55 SGT

Robust Neural Training and Pruning Algorithms for a Class of Nonlinear Tracking Control Systems

Ni Jie

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirements for the degree of

Doctor of Philosophy

2006

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Ni Jie

Acknowledgements

I would like to express my sincere gratitude to all the people who helped me during the years of my postgraduate study at Nanyang Technological University.

Most of all, to my supervisor, Associate Professor Qing Song, for his intelligent guidance and patient support over the past years. His analytic insight into the problem helped greatly in the accomplishment of this thesis. Without his instructions and help, this thesis will not be possible. It is a pleasure experience to get the consultation from him.

Thanks also go to my colleagues in Intelligent Machine Research Lab and Computer Control Lab, they are all very friendly.

My deepest thanks are to my parents, and my wife LiJun He, thank you for your love, understanding and encouragement.

Finally, I acknowledged the School of EEE, Nanyang Technological University, for providing the financial support, research facilities and opportunity for this study.

Summary

This thesis focuses on developing robust online training and pruning algorithms for a class of neural network tracking control systems. In particular, a complete convergence analysis is presented for all the algorithms with different learning schemes, respectively.

There have been extensive research and significant progress in the area of robust discrete time neural controllers designed for nonlinear systems with different classes of nonlinear functions. For example, a first-order approximation is applied in the convergence proof to deal with the nonlinear activation function. Variable structure and dead zone schemes also have been introduced to design robust adaptive algorithms of neural network control systems to improve the control performance.

However, when a neural system is used to handle unlimited samples, including training and testing data, an important issue is how well it generalizes to patterns of the testing data, which is known as generalization ability. For large discrete time domain sequential signals, such as the signals in the online control applications, it is usually impossible to cover every sample data even with proper training. One would like the system to generalize from training samples to the underlying function and give reasonable answer to novel inputs with the existing training data. A rule of thumb for

obtaining good generalization is to use the smallest system that fits the testing data. When a network has too many free parameters (i.e. weights and/or units), it may end up by just memorizing the training patterns. Both theoretical and practical results show that networks with minimal free parameters exhibit better generalization performance, which can be illustrated by recalling the analogy between neural network learning and curve fitting. Moreover, knowledge embedded in smaller trained networks is presumably easier to interpret and thus the extraction of simple rules can be facilitated. Lastly, from an implementation point of view, smaller networks require fewer resources in any physical computational environment.

Before applying the pruning methods in the closed-loop control system, we applied a robust Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm in a closed-loop control system, with a proper control structure and completed stability proof. The main reason for choosing the SPSA algorithm is the excellent drift-free parameter estimate through simultaneous perturbation of the weights and the inherent relationship with the Hessian-based pruning algorithm, which will be used for the online pruning in the closed-loop control system.

As for the neural network structure optimization, two different incremental approaches are often pursued. The first starts with a small initial network and gradually adds new hidden units or layers until learning takes place. Well-known examples for this kind of growing algorithms are cascade correlation and others. The second approach, referred to as pruning, starts with a large network and excises unnecessary weights and/or units. In this thesis, we focus on an online training and pruning

algorithm for a class of control systems. In particular, completed stability analysis of the online pruning algorithms is provided, which is critically important for a control application and it is believed that few previous work was done in this area.

In summary, this thesis developed robust training algorithms for the online control applications in the closed-loop control system. Then the Hessian based pruning algorithm is integrated in the system to provide an improved generalization ability with completed stability proof. To improve the pruning efficiency, a robust neuron-pruning based algorithm is developed for the online tracking control system. Simulation results in the following chapters show better performance of the proposed algorithms over other existing training only schemes.

Table of Contents

Acknowledgements	i
Summary	ii
List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Objectives and Our Approach	3
1.3 Major Contributions of the Thesis	5
1.4 Report Organization	6
2 An Overview of Related Works	8
2.1 Architecture of Neural Networks	8
2.1.1 Neural Networks Approach to Control	11
2.1.2 Generalization Ability of Neural Networks	13

TABLE OF CONTENTS

vi

2.1.2.1	Bias and Variance	13
2.1.2.2	Generalization	18
2.2	Pruning Algorithms	21
2.2.1	Hessian-based Network Pruning	21
2.2.2	Weight Decay	25
2.2.3	Weight Elimination	26
2.2.4	Minimal Resource Allocation Network	27
2.2.5	Interactive Pruning	28
2.2.6	Tukey-Kramer Multiple Comparison Pruning	28
2.3	Learning Algorithms	29
2.3.1	Back-propagation Algorithm	30
2.3.2	Simultaneous Perturbation Stochastic Approximation Algo- rithm	31
2.4	Concluding Remarks	33
3	Robust Neural Network Tracking Controller Using Simultaneous Perturbation Stochastic Approximation	34
3.1	Introduction	34
3.2	System Modelling	37
3.2.1	The ARX Modelling Representation	37
3.2.2	The NARMAX Model Representation	38
3.3	Control System and SPSA Training Algorithm	40

TABLE OF CONTENTS

vii

3.4	Conic Sector Condition for the Robustness Analysis of the Output Layer	45
3.5	Conic Sector Condition for the Hidden Layer Training	58
3.6	Discussion of the Convergence Properties of the Specified SPSA Controller	65
3.7	Simulation Results	68
3.7.1	Euler's Method	69
3.8	Concluding Remarks	74
4	Dynamic Weight-Pruning Based Training Algorithm for Neural Tracking Control Systems	81
4.1	Introduction	81
4.2	NN Tracking Controller and the Dynamic Training and Pruning Algorithm	83
4.2.1	Design of the Controller	83
4.2.2	Basic Form of the Training Algorithm	84
4.2.3	Hessian Based Pruning	87
4.2.4	The DTP Algorithm	89
4.3	Conic Sector Condition for the Robustness Analysis of the Learning Law in the Output Layer	91
4.3.1	Conic Sector Condition for the Training Process in the Output layer	92

TABLE OF CONTENTS

viii

4.3.2	Conic Sector Condition for the Pruning Process in the Output Layer	97
4.4	Conic Sector Condition for the Robustness Analysis of the Learning Law in the Hidden Layer	103
4.4.1	Conic Sector Condition for the Training Process in the Hidden Layer	104
4.4.2	Conic Section Condition for the Pruning Process in the Hidden Layer	109
4.5	Simulation Results	113
4.6	Concluding Remarks	119
5	Neuron-Pruning Based Robust Backpropagation Training Algorithm for RBF Network Tracking Controller	120
5.1	Introduction	120
5.2	RBF Tracking Controller and Pruning Based Robust Training Algorithm	122
5.2.1	Design of the Controller	122
5.2.2	RBF Unit Pruning	126
5.2.2.1	Weight Pruning	126
5.2.2.2	Unit Pruning	129
5.2.3	Summary of the Pruning Based Robust Training Algorithm	131
5.2.4	Convergence Properties for the Dynamic Training & Pruning Process of the Output Layer	132

TABLE OF CONTENTS

ix

5.2.5	Convergence Property of the Hidden Layer	138
5.3	Simulation Results	143
5.4	Concluding Remarks	153
6	Conclusions and Recommendations	154
6.1	Conclusions	154
6.2	Recommendations	155
6.2.1	Apply to Real Control Plant	156
6.2.2	Incorporation of Growing Technique	156
6.2.3	Use of Online Support Vector Regression	156
	Appendix A	158
	Author's Publications	163
	Bibliography	164

List of Tables

List of Figures

2.1	Architectural graph of a multilayer perceptron	9
2.2	A schematic illustration of the meaning of high bias. Solid circles denotes a set of data points which are generated from an underlying function (represented by dashed line) with the addition of noise. Using a fixed function to model the data, the bias will generally be high while the variance will be zero	17
2.3	In this case the bias is low but the variance is high	18
2.4	Good generalization(correct fitted data)	19
2.5	Bad generalization(over fitted data)	20
2.6	The complexity penalty term	27
3.1	Structure of the control scheme	42
3.2	Feedback system.	47
3.3	The equivalent error feedback system using the conic sector conditions: (a)For the tracking error $s(k)$; (b)For the estimation error $e(k)$	53
3.4	The reference signal for the plant.	74
3.5	Output by using the robust SPSA based neural controller.	75

LIST OF FIGURES

xii

3.6	Tracking error by using the robust SPSA based neural controller. . .	76
3.7	Mean square error by using the robust SPSA based neural controller.	76
3.8	One weight $v_{1,1}$ of the output layer by using the robust SPSA based neural controller.	77
3.9	Output by using the standard BP based neural controller.	77
3.10	Tracking error by using the standard BP based neural controller. . . .	78
3.11	Mean square error by using the standard BP based neural controller.	78
3.12	One weight $v_{1,1}$ of the output layer by using the standard BP based neural controller.	79
3.13	Output by using the PID based controller.	79
3.14	Tracking error by using the PID based controller.	80
3.15	Mean square error by using the PID based controller.	80
4.1	Structure of the control scheme	84
4.2	The equivalent error feedback systems using the conic sector condi- tions for the estimate error e_k^T and the output $\Phi_k^v = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k$, where $H_2=1$;	96
4.3	The equivalent error feedback systems using the conic sector condi- tions for the estimate error e_k^{pT} and the output $\Phi_k^{vp} = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k -$ τ_k^v , where $H_2=1$;	100
4.4	An overfitting example	114
4.5	Reference signal	116

4.6	Output y_k and reference signal using the standard ASP based neural network controller with 100 hidden neurons	117
4.7	Output y_k and reference signal using the DTP based neural network controller	117
4.8	Root mean square error using the standard ASP based neural network controller with 100 hidden neurons	118
4.9	Root mean square error using the DTP based neural network controller	118
5.1	Reference signals	145
5.2	System output $x_2(k)$ by using standard BP algorithm.	146
5.3	Hidden layer weight $w(1,1)$ by using standard BP algorithm.	146
5.4	Hidden layer weight $w(5,1)$ by using standard BP algorithm.	147
5.5	System output $x_2(k)$ by using pruning based robust BP training algorithm by setting $L=5$	147
5.6	Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=5$	148
5.7	Hidden layer weight $w(1,1)$ by using pruning based robust BP training algorithm by setting $L=5$	148
5.8	Hidden layer weight $w(5,1)$ by using pruning based robust BP training algorithm by setting $L=5$	149
5.9	The curve for the neuron pruning by setting $L=5$	150
5.10	System output $x_2(k)$ by using pruning based robust BP training algorithm by setting $L=10$	150

LIST OF FIGURES

xiv

5.11	Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=10$	151
5.12	The curve for the neuron pruning by setting $L=10$	151
5.13	System output $x_2(k)$ by using pruning based robust BP training al- gorithm by setting $L=20$	152
5.14	Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=20$	152
5.15	The curve for the neuron pruning by setting $L=20$	153
1	Extension to conic sector stability theory	160

Chapter 1

Introduction

1.1 Background and Motivation

One of the major problems in the design of control systems is to find a way to regulate the system when there is uncertainty about the nature of the underlying process. Adaptive control procedures (i.e., those that can learn and adapt over time) have been developed in a variety of areas for such problems. Examples of applications in areas such as robot arm manipulation, aircraft control, macroeconomic policy making, and biological systems regulation are given in books and journals [1–6]. Neural networks have attracted much attention for their potential to address a number of difficult problems in modelling and control [7–12]. One of the areas receiving a significant portion of attention is the use of neural networks for controlling and regulating nonlinear dynamical systems [13–17], where the approach is based on

using neural networks to approximate the unknown control law. The basis for this approach is the now well-known fact that any measurable function can be approximated within any accuracy by certain single (or multiple) hidden-layer feed-forward neural networks. [7, 18, 19]

However, when a neural system is trained with training data, an important issue is how well it generalizes to patterns other than the training set. For large discrete time domain problems, it is impossible to provide examples of every input. If the system simply memorizes the training patterns, it performs well during training but may fail miserably when presented with different testing data [7, 20–23]. Therefore, one would like the system to generalize from training samples to the underlying function and give reasonable answers to novel inputs. A rule of thumb for obtaining good generalization is to use the smallest system that fits the data. In particular, a theoretical study [24, 25] has shown that when the number of hidden units equals the number of training examples (minus one), the back-propagation error surface is guaranteed to have no local minima. On the other hand, both theory [26] and experiments [27, 28] show that networks with fewer free parameters exhibit a better generalization performance, and this is explained by recalling the analogy between neural network learning and curve fitting. Moreover, knowledge embedded in small trained networks is presumably easier to interpret and thus the extraction of simple rules can hopefully be facilitated. Lastly, from an implementation point of view, smaller networks requires fewer resources in any physical computational environment.

To solve the problem of optimizing the size of neural networks, two different incremental approaches are often pursued. The first starts with a small initial network and gradually adds new hidden units or layers until learning takes place [29, 30], which is named as growing technique. The second, referred to as pruning, starts with a relatively large network and excises unnecessary weights and/or units. This approach combines the advantages of training large networks and those of running small ones (i.e., improved generalization). Among different pruning algorithms there are methods which reduce the excess weights/nodes during the training process, such as penalty term methods [7, 31], and methods in which the training and pruning process are carried out in completely separate phases [32–36].

1.2 Objectives and Our Approach

Although many neural network training algorithms are reported to be efficient and the results also satisfy the good generalization performance, few are conducted with online control applications. The main reasons are:

- (1) Pruning based on batch mode, which is not suitable for sequential online training;
- (2) Choosing appropriate parameters, such as penalty term methods, which is hard to be optimized;
- (3) Retraining requirement: the stability or convergence is not guaranteed, and it

requires a post-training, which is not applicable for online control applications.

The goal of this thesis is first to apply a novel robust algorithm for the online training of neural networks, then the robust algorithm is integrated with an online pruning procedure for the control system and finally a complete stability proof for the whole system is provided. More specifically, the objectives are as follows:

- (1) To briefly review the growing area of neural networks for control applications.
- (2) To consider the problem of developing adaptive controllers for a class of dynamic systems with unknown (usually nonlinear) governing equations and develop a novel training algorithm for neural networks to provide an improved performance over other existing algorithms.
- (3) To incorporate the pruning concept into the training algorithms, which can improve the generalization ability of the neural networks, and in turn, a better performance of the neural controller.
- (4) To study the convergence properties of these neural net pruning based controller and provide a completed stability proof for the whole system.

Simply speaking, in this thesis, a novel online training and pruning algorithm for arbitrary feed-forward networks is proposed and applied in a closed-loop control system, which aims to select an optimal network size by gradually reducing the network size. The method is based on the simple idea of iteratively removing unimportant or salient weights and neurons while maintaining the least increment of error surface.

During the training process, some useful information is extracted for the pruning when a certain criterion is satisfied. We shall prove that the tracking and parameter estimation errors of the neural network are bounded using the conic sector theory under some mild assumptions.

1.3 Major Contributions of the Thesis

The thesis provides novel algorithms in the field of tracking control applications using neural networks, emphasizing network pruning to improve the generalization ability of neural networks, and stability analysis for the whole control system. The contributions of this thesis are summarized as follows:

- A training algorithm is adopted and applied [37] with a neural network in a closed-loop control system. The training algorithm, named Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm is used to update the weights of the neural network, and due to its inherent drift-free property, the SPSA based neural controller is proved to provide a better performance over a standard Back-propagation(BP) training algorithm. In particular, a complete convergence proof is given for the closed-loop control system, in the framework of classical adaptive control, which is different from other approaches [37,38].
- The Hessian-based pruning algorithm is integrated with the second-order SPSA algorithm for the training and pruning of the neural networks. It is known that the generalization performance is very important for a neural system.

The inherent connection between the Hessian-based pruning and the second-order SPSA is studied for the purpose of convergence analysis. Moreover, the simulation results based on our novel algorithm are demonstrated to be better in terms of better generalization ability and less weight drift.

- To reduce the computational payload and increase the efficiency of the pruning, a robust neuron-pruning based algorithm is proposed for a class of dynamical control systems. The Radial basis function (RBF) neural network is applied here, which is reported to be computationally more efficient, yet is better initialized compared to multilayer perceptron networks. Instead of the first-order approximation of the activation function, mean value theory is used in the convergence proof to explore the limitation and possible improvement for the robustness performance of the pruning based nonlinear training algorithm.

1.4 Report Organization

The content of this report can be divided into the following chapters.

In Chapter 2, a detailed review of neural networks and related training and pruning algorithms is given at Section 2.1 to 2.3, with a preliminary introduction for mathematical notation throughout this thesis in Section 2.4.

In Chapter 3, the SPSA algorithm is investigated. First, a specific control structure is proposed and the proposed training algorithm is extended for this structure in Section 3.2. A complete stability proof is then provided for the layered network from

Section 3.3 to 3.5. In Section 3.6, the simulation results are given for comparison purposes.

Weight-pruning based and neuron-pruning based algorithms are proposed in Chapter 4 and 5, both of which are based on the Hessian-pruning and developed for tracking control applications. In these two chapters, two different stability analysis are presented for different control structures. Conic sector theory is first introduced for the convergence analysis of the weight-pruning based training algorithms, which is based on the second-order SPSA. A dead zone scheme is proposed to guarantee the stability of the neuron-pruning based algorithm described in chapter 5.

Chapter 6 provides the conclusion of the thesis and recommendations for the future work in this research area.

Chapter 2

An Overview of Related Works

2.1 Architecture of Neural Networks

Artificial neural networks (or simply neural networks) are mathematical models that attempt to achieve certain predetermined goal via the adjustment of weighted interconnections between simple computational elements (referred to here as neurons or nodes). The neural network model was inspired by its biological counterpart, the human brain. Just as humans attempt to process all information received by their sensors (e.g., eyes, ears, etc.) to determine the appropriate actions (e.g., talk, move, etc.) [7], neural networks attempt to use input information to derive useful outputs, represent a technology that is rooted in many disciplines: neurosciences, mathematics, statistics, physics, computer science, and engineering [39–45]. The goal of network training is not to learn an exact representation of the training data

Figure 2.1: Architectural graph of a multilayer perceptron

itself, but rather to build a statistical model of the process which generates the data [7, 8].

The multilayer perceptron, a widely used neural network, has three distinctive characteristics:

- (1) The model of each neuron in the network includes a nonlinear activation function. The nonlinearity is smooth (i.e., differentiable everywhere), as opposed to the hard-limiting nonlinearity used in Rosenblatt's perceptron [7]. A commonly used form of nonlinearity that satisfies this requirement is the sigmoidal nonlinearity defined by the logistic function:

$$Y_j = \frac{1}{1 + \exp(-v_j)} \quad (2.1)$$

where the v_j is the induced local field (i.e., the weighted sum of all synaptic inputs plus bias) of neuron j , and Y_j is the output of the neuron. The presence of the nonlinearities is important because otherwise the input-output relation of the network could be reduced to that of a single-layer perceptron. Moreover, the use of the logistic function is biologically motivated, since it attempts to account for the refractory phase of real neurons [7].

- (2) The network contains one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns (vectors).
- (3) The network exhibits a high degree of connectivity, determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections of their weights.

The Radial basis function(RBF) neural network is an important network structure, which was first introduced in the solution of the real multivariate interpolation problem. The early work on RBF networks is surveyed in [46,47]. The construction of a RBF network, in its most basic form, involves three layers with entirely different roles. The input layer is made up of source neurons that connect the network to its environment. The second layer, the only hidden layer in the network, applies a non-linear transformation from the input space to the hidden space; in most applications the hidden space is of high dimensionality. The output layer is linear, supplying the response of the network to the activation pattern applied to the input layer.

2.1.1 Neural Networks Approach to Control

A trained neural network may be viewed as a practical vehicle for performing a non-linear input-output mapping of a general nature. To be more specific, let N denote the number of input nodes of a multilayer perceptron, and let M denote the number of neurons in the output layer of the network. The input-output relationship of the network defines a mapping from an N -dimensional Euclidean input space to an M -dimensional Euclidean output space, which is infinitely continuously differentiable when the activation is likewise. And the following Universal Approximation Theorem [48] can be directly applicable to neural networks.

Let $\varphi(\cdot)$ be a nonconstant, bounded and monotone-increasing continuous function. Let I_{n_i} denote the n_i -dimensional unit hypercube $[0, 1]^{n_i}$. The space of continuous functions on I_{n_i} is denoted by $C(I_{n_i})$. Then, given any function $f \in C(I_{n_i})$ and $\epsilon > 0$, there exist an integer n_h and sets of real constants α_i , β_i , and ω_{ij} , where $i=1, \dots, n_i$ such that define

$$F(x_1, \dots, x_{n_i}) = \sum_{i=1}^{n_h} \alpha_i \varphi\left(\sum_{j=1}^{n_i} \omega_{ij} x_j + b_i\right) \quad (2.2)$$

as an approximate realization of the function $f(\cdot)$; that is,

$$|F(x_1, \dots, x_{n_i}) - f(x_1, \dots, x_{n_i})| < \epsilon$$

for all $x_1, x_2, \dots, x_{n_i}$ that lie in the input space.

First, the logistic function used as the nonlinearity in a neural model for the con-

struction of a multilayer perceptron is indeed a nonconstant, bounded, and monotone-increasing function; it therefore satisfies the conditions imposed on the function $\varphi(\cdot)$. Next, Eq (2.2) represents the output of a multilayer perceptron described as follows:

- (1) The network has n_i input nodes and a single hidden layer consisting of m_1 neurons; the inputs are denoted by $x_1, x_2, \dots, x_{n_i}$.
- (2) Hidden neuron i has synaptic weights $\omega_{i_1}, \dots, \omega_{n_i}$, and bias b_i .
- (3) The network output is a linear combination of the outputs of the hidden neurons, with $\alpha_1, \dots, \alpha_{n_h}$ defining the synaptic weights of the output layer.

Thus the neural network can be viewed as an approximator for the unknown control law. The universal approximation theorem is only an existence theorem in the sense that it provides the mathematical justification for the approximation of an arbitrary continuous function as opposed to exact representation. Eq (2.2), which is the backbone of the theorem, merely generalizes approximations by finite Fourier series. In effect, the theorem states that a single hidden layer is sufficient for a multilayer perceptron to compute a uniform ϵ approximation to a given training set represented by the set of inputs and a desired output. However, the theorem does not say that a single hidden layer is optimum in the sense of learning time, ease of implementation, or more importantly, generalization.

2.1.2 Generalization Ability of Neural Networks

As mentioned above, the goal of network training is not only to learn an exact representation of the training pattern, but to build a statistical model of the process. For a deep comprehension, the concept of the bias-variance trade-off is introduced, in which the generalization error is decomposed into the sum of the *bias* squared plus the *variance*. [8]

2.1.2.1 Bias and Variance

It is convenient to consider the particular case of a model trained using a sum-of-square error function, although the conclusion will be much more general. For notational simplicity, it is considered that the network has a single output y . The sum-of-square error can be written in the form [7]

$$E = \frac{1}{2} \int \{y(x) - \langle t|x \rangle\}^2 dx + \frac{1}{2} \int \{\langle t^2|x \rangle - \langle t|x \rangle^2\} p(x) dx \quad (2.3)$$

in which $p(x)$ is the unconditional density of the input data, and $\langle t|x \rangle$ denotes the conditional average, or regression, of the target data given by

$$\langle t|x \rangle \equiv \int t p(t|x) dt \quad (2.4)$$

where $p(t|x)$ is the conditional density of the target variable t conditioned on the input vector x . Similarly

$$\langle t^2|x \rangle \equiv \int t^2 p(t|x) dt \quad (2.5)$$

Note that the second term of Eq.(2.3) is independent of the network function $y(x)$ and hence is independent of the network weights. The optimal network function $y(x)$, in the sense of minimizing the sum-of-square error, is the one which makes the first term in Eq.(2.3) vanish, and is given by $y(x) = \langle t|x \rangle$. The second term represents the intrinsic noise in the data and sets a lower limit on the error which can be achieved.

Now consider a whole ensemble of the possible data sets, each containing N patterns, and each taken from the same fixed joint distribution $p(x, t)$. It is mentioned above that the optimal network mapping is given by the conditional average $\langle t|x \rangle$. A measurable of how close the actual mapping function $y(x)$ is to the desired one is given by the integrand of the first term in Eq.(2.3)

$$\{y(x) - \langle t|x \rangle\}^2. \quad (2.6)$$

The value of this quantity will depend on the particular data set $\mathbf{D}$ on which it is trained. And this dependence can be eliminated by considering an average over complete ensemble of data sets, which can be written as

$$\xi_D[\{y(x) - \langle t|x \rangle\}^2] \quad (2.7)$$

where $\xi_D[\cdot]$ denotes the expectation, or ensemble average.

If the network function were always a perfect predictor of the regression function $\langle t|x \rangle$ then this error would be zero. It is known that a non-zero error can arise for essentially two distinct reasons.

- (1) The network function is on average different from the regression function, which is called *bias*.
- (2) The network function is very sensitive to the particular data sets $\mathbf{D}$, that is, at a given x , it is larger than the required value for some data sets and smaller for other data sets, which is called *variance*.

The decomposition of bias and variance can be done by writing Eq.(2.7) in a somewhat different, but mathematically equivalent form. First expand the term in Eq.(2.7) to give

$$\begin{aligned} \{y(x) - \langle t|x \rangle\}^2 &= \{y(x) - \xi_D[y(x)] + \xi_D[y(x)] - \langle t|x \rangle\}^2 \\ &= \{y(x) - \xi_D[y(x)]\}^2 + \{\xi_D[y(x)] - \langle t|x \rangle\}^2 + 2\{y(x) - \xi_D[y(x)]\}\{\xi_D[y(x)] - \langle t|x \rangle\} \quad (2.8) \end{aligned}$$

In order to compute the expansion of Eq.(2.7), the expectation of both sides of Eq.(2.8) over the ensemble of data sets $\mathbf{D}$. It is showed that the third term on the right-hand side of Eq.(2.8) vanishes, and left with

$$\xi_D[\{y(x) - \langle t|x \rangle\}^2] = \{\xi_D[y(x)] - \langle t|x \rangle\}^2 + \xi_D[\{y(x) - \xi_D[y(x)]\}^2] \quad (2.9)$$

It is worth studying the expression in Eq.(2.9) closely. The first term of the right-hand side is the square form of the *bias* and the second refers to the *variance*. The bias measures the extent to which the average (over all data sets) of the network function differs from the desired function $\langle t|x \rangle$. Conversely the variance measures the extent to which the network function $y(x)$ is sensitive to the particular choice of the data set. Corresponding average values for bias and variance by integrating over all x can be easily obtained, so that

$$(B)^2 = \frac{1}{2} \int \{\xi_D[y(x)] - \langle t|x \rangle\}^2 p(x) d(x) \quad (2.10)$$

$$V = \frac{1}{2} \int \xi_D[\{y(x) - \xi_D[y(x)]\}^2] p(x) d(x) \quad (2.11)$$

where B refers to “bias” and V means “variance”

The meaning of the bias and variance terms can be illustrated by considering two extreme limits for the choice of functional form for $y(x)$. Suppose that the target data for network training is generated from a smooth function $h(x)$ to which zero mean random noise ε is added, so that

$$t^n = h(x^n) + \varepsilon^n. \quad (2.12)$$

Note that the optimal mapping function in this case is given by $\langle t|x \rangle = h(x)$. One choice of model for $y(x)$ may be some fixed function $g(x)$ that is completely independent of the data set D , as indicated in Figure 2.2

Figure 2.2: A schematic illustration of the meaning of high bias. Solid circles denotes a set of data points which are generated from an underlying function (represented by dashed line) with the addition of noise. Using a fixed function to model the data, the bias will generally be high while the variance will be zero

The opposite extreme is to take a function which fits the training data perfectly, such as the simple exact interpolation in Figure 2.3. In this case the bias term vanishes at the data points themselves since

$$\xi_D[\{y(x)\}] = \xi_D[h(x) + \varepsilon] = h(x) = \langle t|x \rangle \quad (2.13)$$

and the bias will be small in the neighborhood of the data points. The variance, however, will be significant since

$$\xi_D[\{y(x) - \xi_D[y(x)]\}^2] = \xi_D[\{y(x) - h(x)\}^2] = \xi_D\{\varepsilon^2\} \quad (2.14)$$

which is just the variance of the noise on the data, which could be substantial.

Figure 2.3: In this case the bias is low but the variance is high

It is obvious that there is a natural trade-off between bias and variance. A function which is closely fitted to the data set will tend to have a large variance and hence give a large expected error. The variance can be decreased by smoothing the function, but if this is taken too far then the bias becomes large and the expected error is large again. And the situation of Figure(2.3) is also called overfitting or overtraining. But for some given size of data set, there is some optimal balance between them which gives the smallest average generalization error. This highlights the need to optimize the complexity of the model in order to achieve the best generalization.

2.1.2.2 Generalization

Since the learning process (i.e., training of a neural network) can be viewed as a nonlinear regression problem. [7, 8, 20] The network itself may be considered simply as a nonlinear input-output mapping. Such a viewpoint permits to simplify the

concept of generalization and can view it rather simply as the effect of a good non-linear interpolation of the input data. The network performs interpolation primarily because multilayer perceptrons with continuous activation functions lead to output functions that are also continuous.

A neural network that is designed to generalize well will produce a correct input-output mapping even when the input is slightly different from the patterns used to train the network, as illustrated in the Figure(2.4)

Figure 2.4: Good generalization(correct fitted data)

However, when a neural network learns too many input-output examples, the network may end up with memorizing the training data. That is called overfitting. An example of how poor generalization due to overtraining in a neural network may occur is illustrated in Figure(2.5) for the same data depicted in Figure(2.4), which implies the input-output mapping computed by the network is not smooth.

Figure 2.5: Bad generalization(over fitted data)

(Here smoothness means that the simplest function that approximates the mapping for a given error criterion, which also generally demands the fewest computational resources.) Smoothness is also a natural feature of many applications. It is therefore important to seek a smooth nonlinear mapping for input-output, so that the network is able to learn novel patterns correctly with respect to the training patterns. Then the factors that can influence generalization can be summarized as:

- (1) *The size of the training data set, and how it can represent the environment of interest.*
- (2) *The architecture of the neural network.*

Thus two different viewpoints can be taken for the issue of generalization [7]:

- (1) *The architecture of the network is fixed then the issue to be resolved is that*

of determining the size of the training set needed for a good generalization to occur.

(2) The size of the training set is fixed or unknown, and the issue of interest is that of determining the best network architecture to achieve good generalization.

For the problems faced us here, we cannot repeatedly access the training set and cannot tell whether it is training data or generalized data, because the learning is online. One way out is the optimization of the network architecture. As discussed in last chapter, the approach of pruning combines the advantages of training large networks (i.e., avoidance of local minima) and those of running small ones (i.e., improved generalization), which is suitable for our online control applications.

2.2 Pruning Algorithms

2.2.1 Hessian-based Network Pruning

The basic idea of this approach is to use the information on the second-order derivatives of the error surface in order to make a trade-off between network complexity and training error minimization. A similar idea was originated from Optimal Brain Damage (OBD) procedure [36] or Optimal Brain Surgeon (OBS) procedure [35]. The starting point in the construction of such a model is the approximation of the cost

function ξ_{av} using a *Taylor series* about the operating point, described as follows:

$$\begin{aligned}\xi_{av}(\hat{W}(k) + \Delta\hat{W}(k)) &= \xi_{av}(\hat{W}(k)) + G_k^T(\hat{W}(k))\Delta\hat{W}(k) + \frac{1}{2}\Delta\hat{W}^T(k)H\Delta\hat{W}(k) \\ &\quad + O(\|\Delta\hat{W}(k)\|^3)\end{aligned}\tag{2.15}$$

where $\Delta\hat{W}(k)$ is a perturbation applied to the operating point $\hat{W}(k)$, with the $G_k(\hat{W}(k))$ is the gradient vector and H is the Hessian matrix. The requirement is to identify a set of parameters whose deletion from multilayer perceptron that cause the minimal increase in the value of the cost function ξ_{av} .

To solve this problem in practical terms, the following approximations are made:

- (1) *Quadratic Approximation* : The error surface around a local minimum or global minimum is nearly “quadratic”. Then the higher-order terms in Eq.(2.15) may be neglected.
- (2) *Extremal Approximation* : The parameters have a set of values corresponding to a local minimum or global minimum of the error surface. In such a case, the gradient vector $G_k(\hat{W}(k))$ may be set equal to zero and the term $G_k^T(\hat{W}(k))\Delta\hat{W}_k$ on the right-hand of Eq.(2.15) may therefore be ignored.

Under these two assumptions, Eq.(2.15) can be presented approximately as

$$\Delta\xi_{av} = \xi_{av}(\hat{W}(k) + \Delta\hat{W}(k)) - \xi_{av}(\hat{W}(k)) \simeq \frac{1}{2}\Delta\hat{W}(k)^T H \Delta\hat{W}(k)\tag{2.16}$$

The goal of OBS is to set one of the synaptic weights to zero to minimize the

incremental increase in ξ_{av} given in Eq.(2.16). Let $\hat{W}(k, i)$ denote this particular synaptic weight. The elimination of this weight is equivalent to the condition

$$I_i^T \Delta \hat{W}(k) + \hat{W}(k, i) = 0 \quad (2.17)$$

where I_i is the *unit vector* whose elements are all zero, except for the i th element, which is equal to unity. The goal is to minimize the quadratic term $\frac{1}{2} \Delta \hat{W}(k)^T H \Delta \hat{W}(k)$ with respect to the perturbation $\Delta \hat{W}(k)$, subject to the constraint that $I_i^T \Delta \hat{W}(k) + \hat{W}(k, i)$ is zero, and then minimize the result with respect to the index i .

To solve this constraint optimization problem, the following penalty function is used.

$$S = \frac{1}{2} \Delta \hat{W}(k)^T H \Delta \hat{W}(k) - \lambda (I_i^T \Delta \hat{W}(k) + \hat{W}(k, i)) \quad (2.18)$$

where λ is the Lagrange multiplier. Applying the derivative of the risk function S with respect to $\Delta \hat{W}(k)$, subjects to the constraint of Eq.(2.17), the optimum solution of the weight vector $\hat{W}(k)$ can be found as:

$$\Delta \hat{W}(k) = -\frac{\hat{W}(k, i)}{[H_{i,i}^{-1}]} H^{-1} I_i \quad (2.19)$$

where H^{-1} is the inverse of the Hessian matrix H , and $[H_{i,i}^{-1}]$ is the ii -th element of the inverse matrix. The corresponding optimum value of the risk function S for element W_i is

$$S_i = \frac{(\hat{W}(k, i))^2}{2[H_{i,i}^{-1}]} \quad (2.20)$$

The value S_i optimized with respect to $\Delta\hat{W}(k)$, subject to the constraint that the i th synaptic weights $\hat{W}(k, i)$ be eliminated, is called the *saliency* of $\hat{W}(k, i)$.

In this method, the weight corresponding to the smallest saliency is the one selected for deletion. Moreover, the corresponding optimal changes in the remainder of the weights are given in Eq.(2.19), which show that they should be updated along the direction of the i th column of the inverse of the Hessian.

Unlike Hessian-based pruning, some penalty-term like methods minimize the total risk expressed as:

$$R(\hat{W}) = \xi_{av}(\hat{W}) + \lambda\xi_c(\hat{W}) \quad (2.21)$$

The first term, $\xi_{av}(\hat{W})$, is the standard performance measure and the second term, $\xi_c(\hat{W})$, is the *complexity penalty*, which depends on the network(model) alone. In fact, the form of the total risk defined in Eq.(2.21) is simply a statement of *regularization theory*, which is proposed by Tikhonov [49, 50]. In general setting, one choice of complexity-penalty term $\xi_c(\hat{W})$ is the k th order smoothing integral

$$\xi_c(\hat{W}, k) = \frac{1}{2} \int \left\| \frac{\partial^k}{\partial x^k} F(x, \hat{W}) \right\|^2 \mu(x) dx \quad (2.22)$$

where the $F(x, \hat{W})$ is the input-output mapping performed by the model, and $\mu(x)$ is some weighting function that determines the region of the input space over which the function $F(x, \hat{W})$ is required to be smooth. The motivation is to make the k th derivative of $F(x, w)$ with respect to the input vector x small. The larger we choose k , the smoother(i.e., less complex) the function $F(x, w)$ will become.

2.2.2 Weight Decay

The concept of weight decay was proposed by Hinton in 1989 [31]. In this procedure, the complexity penalty term is defined as the squared norm of the weight vector $\hat{W}$ (i.e., all the free parameters) in the network, as shown by

$$\xi_c = \|\hat{W}\|^2 = \sum_{i \in \xi_{total}} \hat{W}_i^2 \quad (2.23)$$

where the set ξ_{total} refers to all the synaptic weights in the network. This procedure operates by forcing some of the synaptic weights in the network to take values close to zero, while permitting other weights to retain their relatively large values. Accordingly, the weights of the network are grouped into two categories: those that have a large influence on the network, and those that have little or no influence on it. The weights in the latter category are referred to as *excess weights*. The use of complexity regularization encourages the excess weights to assume values close to zero, and thereby improve generalization. [51] In this case, all the weights in the multilayer perceptron are treated equally. That is, the prior distribution in weight space is assumed to be centered at the origin. In 1992, Nowlan and Hinton [52] describe a more complex penalty term that models the probability distribution of the weights as a mixture of Gaussians. Strictly speaking, weight decay is not the correct form of complexity regularization for multilayer perceptron since it does not fit into the rationale described in Eq.(2.22).

2.2.3 Weight Elimination

In 1991, Weigend and Rumelhart [7, 53] proposed another complexity penalty term which is defined as

$$\xi_c(\hat{W}) = \sum_{i \in \xi_{total}} \frac{(\hat{W}_i/\hat{W}_0)^2}{1 + (\hat{W}_i/\hat{W}_0)^2} \quad (2.24)$$

where $\hat{W}_0$ is a preassigned parameter, and $\hat{w}_i$ refers to the weight of some synapse i in the network. The set ξ_{total} refers to all the synaptic connections in the network. An individual penalty term varies with $\hat{W}_i/\hat{W}_0$ in a symmetric fashion, as shown in Figure 2.6. When $|\hat{W}_i| \ll \hat{W}_0$, the complexity penalty for that weight approaches zero. The implication of this condition is that insofar as learning from examples is concerned, the i th synaptic weight is unreliable and should be eliminated from the network and vice versa. Thus it can be seen that the complexity penalty term of Eq.(2.24) does serve the desired purpose of identifying the synaptic weights of the network that are of significant influence. Note also that the weight-elimination procedure includes the weight-decay procedure as a special case. With the proper choice of parameter w_0 , it permits some weights in the network to assume values that are larger than with weight decay.

There are also many other these penalty-like methods and the penalty-term methods control the amount of pruning by balancing the scaling factors of the error terms. However, it often needs to be made before training begins, which will be exhausted, especially for the control problem here. So it does not meet the requirement of the control application here.

Figure 2.6: The complexity penalty term

2.2.4 Minimal Resource Allocation Network

This algorithm is actually for a Radial Basis Function(RBF) neural network, which combines the growth criterion of the resource allocating network(RAN) of Platt(1991) [54] with a pruning strategy [27], which was developed by [30, 55].

The pruning criterion of this algorithm is more heuristic than theoretical. For each observation, the contribution of each hidden unit to the output is observed during pruning stage and those hidden units with insignificant contribution will be removed.

More specific, for each observation (x_j, d_j) , the overall output is computed as:

$$y = \sum_{i=0}^N \hat{W}_i \varphi(x_i) \quad (2.25)$$

where N is the number of neurons in hidden layer and here $\varphi(x)$ is the radial basis

function which is defined as:

$$\varphi(x_i) = \exp\left(-\frac{\|x_i - \mu_k\|^2}{\sigma_k^2}\right) \quad (2.26)$$

where μ_k is the center vector for the k th hidden unit and σ_k is the width of the Gaussian function. The hidden units with the least contribution are removed.

2.2.5 Interactive Pruning

Sietsma and Dow [52] describe an interactive method in which the designer inspects a trained network and decides which nodes to remove. Several heuristics are used to identify units which do not contribute to the solution.

- (1) If a unit has a constant output over all the training patterns, then it can be removed.
- (2) If a number of units have highly correlated responses over all patterns, then they can be combined into a single unit.

2.2.6 Tukey-Kramer Multiple Comparison Pruning

Certain evolutionary bootstrap approaches [52,56] has been taken for neural network pruning. Bootstrapping is used as a general framework for estimating objectives out of a sample by redrawing subsets from a training sample. Evolution is used to search the large number of potential network architectures. The combination of

these two methods creates a network estimation and selection procedure which finds parsimonious network structures which generalize well. It also can be regarded as a cross-validation method, which is usually used for off-line learning.

Tukey Kramer multiple comparison pruning [56] uses the bootstrap algorithm to estimate the distribution of the model parameter salience and statistical multiple comparison procedures are then used to make pruning decisions.

2.3 Learning Algorithms

There are many methods to train a neural network, such as the Back-propagation algorithm(BP), the conjugate-gradient method and so on. These methods are suitable for some different training modes respectively. We can divide these methods into two groups:

- (1) Sequential Mode. In this mode of operation, weight updating is performed after the presentation of each training example. This mode is also referred to as an *online mode*.
- (2) Batch Mode. In this mode, weight updating is performed after the presentation of *all* the training examples that constitute an epoch.

From an “on-line” operation point of view, the sequential mode of training is preferred over the batch mode because it requires less local storage for each synaptic connection. A comprehensive introduction to neural networks and various learning

algorithms are given by Bishop [8] and Haykin [7].

2.3.1 Back-propagation Algorithm

Given an initial starting value for W , where W is known as weights(parameters), one can iterate with the Back-propagation method(BP) [7, 8] as

$$W(k+1) = W(k) - \eta \frac{\partial L(W)}{\partial W} \quad (2.27)$$

where η is a predetermined or adaptive learning rate. $L(W)$ is the error function, the typical one is the sum of squared error(SSE) given by

$$L(W) = \sum_{i=1}^N (Y_i - \hat{Y}_i)^2 \quad (2.28)$$

where $\hat{Y}_i$ is the actual output by the neural network for the i th sample and Y_i is the desired value.

In the application of the BP algorithm, two distinct passes of computation are distinguished.

- 1) The first pass is referred to as the forward pass. In the forward pass the weights remain unaltered throughout the network and the function signals of the network are computed on a neuron-by-neuron basis.
- 2) The backward pass, on the other hand, starts at the output layer by passing the error signals through the network, layer by layer, and recursively computing

the local gradient for each neuron. This recursive process permits the weights of the network to undergo changes.

2.3.2 Simultaneous Perturbation Stochastic Approximation

Algorithm

Stochastic approximation(SA) refers to a general set of recursive algorithms for finding minima of functions in the presence of noisy observations [57, 58]. In particular, suppose we wish to minimize some differentiable loss function $L(W)$, $L : R^p \rightarrow R^1$ subjects to the weight vector W . Thus we are searching for a minimum point W^* satisfying the gradient equation: $g(W) = \frac{\partial L}{\partial W} = 0$.

Simultaneous perturbation stochastic approximation (SPSA) was proposed by James C. Spall [37, 59, 60] and widely applied in many different applications [38, 61–65].

The SPSA procedure in the general recursive SA form:

$$W(k+1) = W(k) - \alpha(k)\hat{G}(W(k)) \quad (2.29)$$

where $\alpha(k)$ is the learning rate parameter and $\hat{G}(W(k))$ is the estimate of the gradient $G(W(k))$. The simultaneous perturbation approximation has all elements of $W(k)$ randomly perturbed together to obtain two measurements $L(\cdot)$, but each component is formed from a ratio involving the individual components in the perturbation vector and difference in the two corresponding measurements. For two-sided

simultaneous perturbation, we have

$$\hat{G}(W(k)) = \frac{L(W(k) + c_k \Delta_k) - L(W(k) - c_k \Delta_k)}{2c_k \Delta_k} \quad (2.30)$$

where the distribution of the user-specified p-dimensional random perturbation vector, $\Delta_k = [\Delta_{k1}, \Delta_{k2}, \dots, \Delta_{kp}]^T$, satisfies conditions discussed later.

Spall [59, 60] presents sufficient conditions for convergence of the SPSA iterate using a differential equation approach well known in general SA theory. To establish convergence, conditions are imposed on both gain sequences ($\alpha(k)$ and c_k), the user specified distribution of Δ_k , and the statistical relationship of Δ_k to the measurement $L(\cdot)$. The essence of the main conditions is that $\alpha(k)$ and c_k both go to 0 at rates neither too fast nor too slow, that $L(W(k))$ is sufficiently smooth and that the Δ_{ki} are independent and symmetrically distributed about 0 with finite inverse moments $E(|\Delta_{ki}|^{-1})$ for all k, i . One particular distribution for Δ_{ki} that satisfies these latter conditions is the symmetric Bernoulli ± 1 distribution; two common distributions that do not satisfy the conditions are the uniform and normal distributions. Based on the first order SPSA, Spall proposed the Adaptive Simultaneous Perturbation(ASP) approach [37], which is a second-order version of SPSA. And we will discuss it in the Chapter 4 in details.

2.4 Concluding Remarks

This thesis aims to develop an online training and pruning algorithm for control applications, from which we know that only sequential training is applicable. Moreover, stability is critically important for a control system, which is not investigated in the previous pruning algorithms. So in the following chapters, the SPSA based neural controller is first proposed, which performs better over existing methods. Followed by that, a novel weight-pruning based algorithm is then proposed for the control system, together with a complete stability proof. Finally, the neuron-pruning based algorithm is investigated.

Chapter 3

Robust Neural Network Tracking Controller Using Simultaneous Perturbation Stochastic Approximation

3.1 Introduction

Extensive research and significant progress have been made in the area of robust discrete time neural controller design for a class of nonlinear systems with specific nonlinear functions [66–71]. Variable structure and dead zone schemes have been introduced to design robust adaptive algorithms of neural network control systems

to improve the tracking performance [68–70]. The well-known Persistent Exciting (PE) condition has been removed in the presence of disturbance [71]. More recently, the idea of Simultaneous Perturbation Stochastic Algorithm (SPSA) has been used recently as model free control method for dynamical systems [59–61].

In this part of the thesis, we shall propose a SPSA-based neural control structure and derive a general stability proof when it is applied to a nonlinear input-output dynamical plant. The plant under consideration is nonlinear and the neural network in the system is used to estimate the nonlinear function in the closed-loop. The conic sector theory [72, 73] is introduced to design the robust control system, which aims to provide guaranteed boundedness for both the input-output signals and the weights of the neural network. One of the main advantage of the conic sector approach is that it provides a model free study. Our neural network is trained by the SPSA algorithm in the closed-loop to provide an improved training performance over standard methods, such as back-propagation algorithm, which in turn, will yield good tracking performance for the dynamic control system. The main motivation for using the SPSA instead of the popular back-propagation algorithm is its excellent convergence property. The SPSA algorithm which was proposed by Spall provides a good parameter estimate through simultaneous perturbation of the weights.

Unlike the robust conic sector analysis for a pre-trained neural network [74], we provide an on-line scheme for the robustness analysis of the neural control system. A special normalized cost function is provided to the SPSA algorithm to reject disturbance and solve the so-called vanished cone problem [73]. A two-stage normalized

training strategy is proposed for the SPSA training with guaranteed I/O stability using the conic sector condition. In addition to the general stability proof, one of the most interesting contributions of this chapter is the revelation of the relationship between the conventional adaptive control system and generalization theory, which is mainly developed for neural-network based pattern recognition system and, to the best of our knowledge, is not widely acknowledged by the control community. That is, a relatively large learning rate will contribute to a faster convergence speed of the SPSA training algorithm and in turn yields a good adaptive learning capability. This is closely linked to the concept of generalization of neural network theory. The early neural control system approaches, including the classical back-propagation based training algorithm, tend to emphasize the approximation property of a large neural network. It is only recently that good generalization property has attracted more interests, for it suggests that a network with a reasonable number of neurons may be the best way to approximate a nonlinear system rather than an over-fitted large network. As shown later in the theoretical analysis about the learning rate and the simulation results, an optimal number of neurons can be derived based on the maximum learning rates calculated by the conic sector condition. Thus, it allows one to achieve good generalization performance in terms of reduced control signal error and fast tracking speed. This idea can also be further developed into an adaptive pruning based SPSA algorithm to automatically search for an optimal neural network structure [75]. The performance improvement of the proposed algorithm can be measured in terms of preventing weights shifting, fast convergence and robustness against system disturbance.

3.2 System Modelling

Neural networks can be employed within estimators for all the of the three classes (parametric, nonparametric, semiparametric) of models [76,77]. To understand and analyze real-world physical phenomena, various mathematical models have been developed. Depending on some *a priori* knowledge about the process, data and model, we differentiate between three fairly general modes of modelling. The idea is to distinguish between three levels of prior knowledge and an overview of the white, grey and black box modelling techniques can be found in [76] and [77].

The exact form of the input-output relationship that describes a real-world system is most commonly unknown, and therefore modelling is based upon a chosen set of known functions. In addition, if the model is to approximate the system with an arbitrary accuracy, the set of chosen continuous functions must be dense. In this light, neural networks can be viewed as another mode of functional representations.

3.2.1 The ARX Modelling Representation

Linear systems can be represented in various structural forms such as state-space or auto-regressive moving average models. The state space model of an n^{th} order linear system with m inputs and l outputs can be written as:

$$\dot{x} = Ax + Bu \quad (3.1)$$

$$Y = Hx \quad (3.2)$$

where x is an $(n \times l)$ vector which represents the "states" of the system, y is an $(l \times 1)$ output vector, u is an $(m \times 1)$ input vector, and A is an $(n \times n)$, B is an $(n \times m)$, and H is an $(l \times n)$ matrix.

A LTI discrete-time system can also be modelled by using autoregressive average or autoregressive average exogenous (ARX) difference equations, [78]. This model representation is useful when all the system states are not accessible or only the input/output data is available. Let n denotes the order of the ARX model, the model structure can be written as

$$y(k) = \sum_{i=1}^n (a_i y(k-i) + b_i u(k-i)) \quad (3.3)$$

where y_k is the output of the model at k^{th} iteration, a_i and b_i are the weighting factors.

3.2.2 The NARMAX Model Representation

A well known representation of a wide class of nonlinear models is constituted by the general nonlinear difference equation model, known as the Nonlinear AutoRegressive Moving Average with eXogenous variables (NARMAX) model [79, 80]. It provides a unified representation for a wide class of nonlinear systems. Leontaris and Billings showed that several well known models such as the Hammerstein, Wiener and bilinear models are special cases of the NARMAX model [81]. This

model can be represented as follows:

$$y(k) = F(y(k-1), \dots, y(k-n_y), u(k-d), \dots, u(k-d-n_u), e(k-1), \dots, e(k-n_e)) + e(k) \quad (3.4)$$

where $y(k), u(k)$ and $e(k)$ are the system output, control input, and noise items, respectively. n_y , n_u and n_e are the corresponding maximum lags, with d denotes the delay and F is some nonlinear function, the form of which is usually unknown. Polynomial expansions are linear-in-parameters models, so that algorithms of least squares family can be employed for parameter estimation. The Eq.(3.4) can be expressed as follows:

$$y(k) = \Psi_p^T(k-1)a_p + \Psi_n^T(k-1)a_n + e(k) \quad (3.5)$$

where Ψ_p^T includes all the polynomial terms involving only $u(\cdot)$ and $y(\cdot)$ up to degree l and time $k-1$ (process terms), and Ψ_n^T includes all the remaining terms (noise terms). Vectors a_p and a_n are the corresponding coefficients, since noise terms are not measurable, sequence $e(\cdot)$ is estimated iteratively.

In particular, given a desired trajectory vector of a robot arm $x_{nd}(k) \in R^m$, the tracking error for a robot can be defined as $e_n(k) = x_n(k) - x_{nd}(k) \in R^m$. A filtered tracking error is also used typically in the robot controller as the following:

$$r(k) = e_n(k) + \lambda_1 e_{n-1}(k) + \dots + \lambda_{n-1} e_1(k) \quad (3.6)$$

where $e_{n-1}(k), \dots, e_1(k)$ are the delayed values of the tracking error $e_n(k)$, and $\lambda_{n-1}, \dots, \lambda_1$ are constant matrices selected so that the polynomial

$$\lambda_n z^n - 1 + \dots + \lambda_1 z^{n-2} + z^{n-1} \quad (3.7)$$

is stable (z represents the time delayed factor), based on which, we will propose our control structure in the following sections.

3.3 Control System and SPSA Training Algorithm

Followed by NARMAX model introduced above, a class of dynamical nonlinear plant, which has wide applications in robotics and variable air volume control system [68,70,82], a general form of which can be represented as an multiple-input, multiple-output (MIMO) form as follows:

$$y(k+1) = f(y(k), \dots, y(k-l), u(k-d), \dots, u(k-n)) \quad (3.8)$$

where $y(k) \in R^m$ is measured plant output vector, $u(k) \in R^m$ is the measured plant input vector, n denotes known plant order, d is the known plant delay. Further, an assumption is made on the nonlinear function $f(\cdot) \in R^m$ which is a continuous differentiable function.

To be more specific, a simplified model is studied in this thesis rather than the above

general model, which can be presented as following:

$$y(k+1) = f(y(k), \dots, y(k-l), u(k-1), \dots, u(k-n)) + u(k) + \varepsilon(k) \quad (3.9)$$

where $y(k) \in R^m$ is the output, and $f(\cdot)$ is a dynamic nonlinear function follows the above assumptions, $\varepsilon(k) \in R^m$ denotes the system noise vector which is assumed to be bounded $\|\varepsilon(k)\| \leq \varepsilon_{max}$, and $u(k) \in R^m$ is the control signal vector with a unit discrete time delay, which can also be extended to a d-step delay system by using a linear predictor [82]. The tracking error of the control system can be defined as

$$s(k) = y(k) - d(k) \quad (3.10)$$

where $d(k) \in R^m$ is the command signal. Define the control signal as

$$u(k) = -\hat{f}(k) + d(k+1) + k_v s(k) \quad (3.11)$$

where k_v is the gain parameter of the fixed controller and $\hat{f}(k-1)$ is the estimate of the nonlinear function $f(k-1)$ by the neural network. Then the error vector can be presented as

$$e(k) = f(k-1) - \hat{f}(k-1) + \varepsilon(k) \quad (3.12)$$

to train the neural network as shown in Figure 3.1 Note that the training error $e(k)$ may not be directly measurable, so we should use the tracking error to generate it

Figure 3.1: Structure of the control scheme

using the closed-loop relationship (3.9), (3.11) and (3.12)

$$e(k) = (1 - z^{-1}k_v)s(k) \quad (3.13)$$

The loss function for SPSA is defined as $L(W) : R^p \rightarrow R^1$, where $W \in R^p$ is the *parameter vector* of the neural network. Consider the problem of finding the optimal parameter of the gradient equation

$$g(W) = \frac{\partial L(W)}{\partial W} = 0 \quad (3.14)$$

for the differentiable loss function $L(W)$.

Now we can define the SPSA algorithm to update $\hat{W}(k) \in R^p$, which is an estimate

of an optimal parameter vector W^* as

$$\hat{W}(k) = \hat{W}(k-1) - \alpha(k)\hat{g}(\hat{W}(k-1)) \quad (3.15)$$

where $\alpha(k)$ is the learning rate and $\hat{g}(\hat{W}(k-1))$ is the approximation of the gradient function with

$$\hat{g}(\hat{W}(k-1)) = \frac{L(\hat{W}(k-1) + c_k \Delta_k) - L(\hat{W}(k-1) - c_k \Delta_k) + \varepsilon^w(k)}{2c_k} r_k \quad (3.16)$$

where $\varepsilon^w(k)$ is the measurement disturbance as defined in [59]. In the above equation, $\Delta_k \in R^p$ is a random directional vector, that is used to stimulate the weight vector simultaneously, $c_k > 0$ is a sequence of positive number satisfying certain regularity conditions [59,60]. The random vector Δ_k is generated via Monte Carlo according to conditions specified in [59,60]. If the i th element of Δ_k is denoted as Δ_{ki} , then the sequence of $r_k \in R^p$ is defined as $r_k = (\Delta_{k1}^{-1}, \Delta_{k2}^{-1}, \dots, \Delta_{kp}^{-1})$.

The output of a three-layer neural network can be presented as

$$\hat{f}(k-1) = H(\hat{w}(k-1), x(k-1))\hat{v}(k-1) \quad (3.17)$$

where the input vector $x(k-1) \in R^{n_i}$ of the neural network is

$$x(k-1) = [x(k-1, 1), x(k-1, 2) \dots x(k-1, n_i)] = [y^T(k-1), y^T(k-2), \dots]^T \quad (3.18)$$

$\hat{v}(k-1) \in R^{p_v}$ is the weight vector of the output layer, and $\hat{w}(k-1) \in R^{p_w}$ is

the weight vector of the hidden layer of the neural network with $p_v = m \times n_h$ and $p_w = n_h \times n_i$, where n_i and n_h are the number of the neurons in the input and hidden layers of the network, respectively. $H(\hat{w}(k-1), x(k-1)) \in R^{m \times P_v}$ is the nonlinear activation function matrix:

$$H(\hat{w}(k-1), x(k-1)) = \begin{bmatrix} h_{k-1,1}, h_{k-1,2}, \dots, h_{k-1,n_h} & 0 \dots & \dots 0 \\ 0 \dots & h_{k-1,1}, h_{k-1,2}, \dots, h_{k-1,n_h} & \dots 0 \\ \dots & \dots & \dots \end{bmatrix}$$

with $h_{k-1,i}$ is the nonlinear activation function

$$h_{k-1,i} = h(x^T(k-1)\hat{w}(k-1, i)) = \frac{1}{1 + e^{-4\lambda x^T(k-1)\hat{w}(k-1, i)}} \quad (3.19)$$

with $\hat{w}_{k-1,i} \in R^{n_i}$, $\hat{w}_{k-1} = [\hat{w}_{k-1,1}^T, \dots, \hat{w}_{k-1,n_h}^T]^T$ and $4\lambda > 0$, which is the gain parameter of the threshold function.

Remark 3.1: To simplify the notation, we could define the overall estimate parameter vector $\hat{W}(k) = [\hat{v}^T(k), \hat{w}^T(k)]^T \in R^p$ with $p = p_v + p_w$ in the SPSA algorithm. However, for a multi-layered neural network, it may not be possible to update all the estimated parameters with a single gradient approximation function to meet the stability requirement. Therefore, it is better that the estimate parameter vectors $\hat{v}(k)$ and $\hat{w}(k)$ are updated separately in the SPSA algorithm using different gradient approximation functions as in the standard back-propagation. This point will be explored further in the robustness analysis in the next section.

3.4 Conic Sector Condition for the Robustness Analysis of the Output Layer

The general idea is that we do not deal with the convergence property of the parameter estimate of the SPSA training algorithm directly, which is well established under certain conditions. Rather we shall prove the tracking error and the parameter estimation error of the output layer of the neural network are bounded using the conic sector theory under some mild assumptions. Furthermore, the boundedness condition for the parameter estimation error of the hidden layer is also derived in the next section. In this approach, the estimated weight are not required to converge to the ideal values as in the traditional robust adaptive control system. Unlike the convergence analysis in [60], Theorem 3.2 in this section proves that the positive scalar $c_k > 0$ can be chosen arbitrarily without affecting the system stability. This also provides guidelines for the selection of SPSA learning rate and the number of neurons to obtain an improved performance. The relationship between the measurement noise of SPSA and the overall noise of the control system is also explored.

The purpose of the SPSA training algorithm is to make the estimate parameter vector $\hat{W}(k)$ approximate the optimal one, and in turn, to produce an optimal tracking error for the control system. To do this, one important condition is that the time-varying equation should be bounded as required for adaptive control systems. To guarantee the boundedness condition, the robust neural controller as shown in

Figure(3.1) uses a normalized SPSA training algorithm, which is isolated from the rest of the system and a deterministic analysis is applied. Interestingly, a different deterministic treatment of the SPSA algorithm can also be found in [83]. In this thesis, the robustness of the system is analyzed using the conic sector theory. A two-stage normalized training strategy is proposed for the SPSA training with guaranteed I/O stability using conic sector condition, which also provides guidelines for the selection of the SPSA learning parameters and normalization to obtain an improved performance. The relationship between the measurement noise of SPSA and the overall disturbance of the control system is also explored.

Theorem 3.1: *Consider the following error feedback system 3.2*

$$s(k) = e^*(k) - P(k)$$

$$\Phi(k) = H_1 s(k)$$

$$P(k) = H_2 \Phi(k)$$

with operators $H_1, H_2 : L_{2e} \rightarrow L_{2e}$ and discrete time signal $s(k), P(k), \Phi(k) \in L_{2e}$

and $e^*(k) \in L_2$. If

(a) H_1 : $s(k) \rightarrow \Phi(k)$ satisfies

$$\sum_{k=1}^N [s^T(k)\Phi(k) + \sigma s^T(k)s(k)/2] > -\gamma,$$

(b) H_2 : $\Phi(k) \rightarrow P(k)$ satisfies

$$\sum_{k=1}^N [\sigma P^T(k)P(k)/2 - P^T(k)\Phi(k)] \leq -\eta \|(P(k), \Phi(k))\|_N^2$$

for some $\sigma, \eta, \gamma > 0$, then the above feedback system is stable with $s(k), \Phi(k) \in L_2$.

Figure 3.2: Feedback system.

Proof: see Corollary 2.1, Chapter 2 of this thesis.

Note that operator H_1 represents the SPSA training algorithm, the input error signal is the tracking error $s(k)$ defined in Eq.(3.10) and the output is $\Phi(k)$, which will be defined later and is related to the weight error vectors, and in turn, the estimation parameter error vector $e(k)$ and tracking error $s(k)$ through Eq.(3.13).

H_2 usually represents the mismatched linear model uncertainty in a typical adaptive linear control system and will be defined later in this section.

The first step to use the conic sector stability Theorem 3.1 is to restructure the control system into an equivalent error feedback system as shown below. Then the parameter estimation error vector should be derived and referred to the output signal $\Phi(k)$. For this purpose, define the desired output of the neural network as the plant nonlinear function in Eq.(3.9)

$$f(k-1) = H(w^*(k-1), x(k-1))v^* \quad (3.20)$$

where $v^* \in R^{p_v}$ is the ideal weight vector in the output layer of the neural network, $w^* \in R^{p_w}$ is the desired weight vector of the hidden layer. Therefore, the parameter estimate error vectors can be defined as $\tilde{v}(k) = v^* - \hat{v}(k) \in R^{p_v}$ and $\tilde{w}(k) = w^* - \hat{w}(k) \in R^{p_w}$ for the output and hidden layers, respectively.

Assumptions:

- a) The sum of the system disturbance $\varepsilon(k)$ is L2 norm bounded;
- b) The ideal weight vector v^* and w^* are L2 norm bounded.

Now we are ready to establish the relationship between the tracking error signal $s(k)$ and the parameter estimate vectors of the neural network, which is referred to the operator H_1 in Theorem 3.1 (i.e. the SPSA algorithm). According to Eq.(3.12)

the error signals can be extended as:

$$\begin{aligned}
s(k) &= H_2 e(k) = H_2(f(k-1) - \hat{f}(k-1) + \varepsilon(k)) \\
&= H_2\left(\frac{1}{2}(f(k-1) - \hat{f}^{v+}(k-1)) + \frac{1}{2}(f(k-1) - \hat{f}^{v-}(k-1)) + \varepsilon(k)\right) \\
&= H_2(H(w^*, x(k-1))v^* - H(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + \varepsilon(k)) \\
&= H_2(H(w^*, x(k-1))v^* - H(\hat{w}(k-1), x(k-1))v^* + H(\hat{w}(k-1), \\
&\quad x(k-1))v^* - H(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + \varepsilon(k)) \\
&= H_2(H(\hat{w}(k-1), x(k-1))\tilde{v}(k-1) + \tilde{H}(\hat{w}(k-1), x(k-1))v^* \\
&\quad + \varepsilon(k)) \\
&= H_2 H(\hat{w}(k-1), x(k-1))\tilde{v}(k-1) + \tilde{e}^v(k) \\
&= -H_2 \Phi^v(k) + \tilde{e}^v(k)
\end{aligned} \tag{3.21}$$

where the operator $H_2 = \frac{1}{1-k_v z^{-1}}$, and

$$\Phi^v(k) = -H(\hat{w}(k-1), x(k-1))\tilde{v}(k-1) \tag{3.22}$$

$$\tilde{e}^v(k) = H_2(\tilde{H}(\hat{w}(k-1), x(k-1))v^* + \varepsilon(k)) \tag{3.23}$$

$$\begin{aligned}
\tilde{H}(\hat{w}(k-1), x(k-1))v^* &= (H(w^*, x(k-1)) - H(\hat{w}(k-1), \\
&\quad x(k-1)))v^*
\end{aligned} \tag{3.24}$$

$$\hat{f}(k-1)^{v+} = H(\hat{w}(k-1), x(k-1))(\hat{v}(k-1) + c_k \Delta_k^v) \tag{3.25}$$

$$\hat{f}(k-1)^{v-} = H(\hat{w}(k-1), x(k-1))(\hat{v}(k-1) - c_k \Delta_k^v) \tag{3.26}$$

with Δ_k^v and $r_k^v \in R^{p_v}$, which can be viewed as the first p_v components of Δ_k and r_k defined before.

Remark 3.2: There is an important implication in Eq.(3.21). The tracking error signal $s(k)$ is directly linked to the output signal $\Phi^v(k)$ in Eq.(3.22), and in turn, the parameter estimation error vector $\tilde{v}(k)$ of the output layer of the network, which implies that the training procedure of the output layer of the neural network should be treated separately from the hidden layer of the network to obtain a bounded disturbance term $\tilde{e}^v(k)$ as defined in Eq.(3.23), i.e. $\tilde{e}^v(k) \in L_2$ as required by Theorem 3.1. Therefore, using Eq.(3.21), we are able to form an equivalent error feedback system Figure 2 as the one in Theorem 3.1. Note that H_2 usually represents the mismatched linear model uncertainty in a typical adaptive linear control system, therefore, the operator $H_2 = \frac{1}{1-k_v z^{-1}}$ represents only the fixed controller and is always stable as $|k_v| < 1$. Furthermore, the condition (b) of Theorem 3.1 can be treated as positive real function, i.e. the plot of H_2 should be in the positive half of a complex plane as shown in [72].

We define the operator H_1^v , which represents the SPSA training algorithm of the output layer, and the loss function as

$$L(\hat{v}(k-1), \hat{w}(k-1)) = \|f(k) - \hat{f}(k)\|^2 \quad (3.27)$$

and with definitions in equation, we have a gradient approximation using the simultaneous perturbation vector $\Delta_k^v \in R^{p_v}$ to stimulate the weights of the output

layer:

$$\begin{aligned}
& \hat{g}(\hat{v}(k-1), \hat{w}(k-1), \Delta_k^v) \\
&= \frac{L(\hat{v}(k-1) + c_k^v \Delta_k^v, \hat{w}(k-1)) - L(\hat{v}(k-1) - c_k^v \Delta_k^v, \hat{w}(k-1)) + \varepsilon^v(k)}{2c_k^v \rho^v(k)} r_k^v \\
&= \frac{(f(k-1) - \hat{f}^{v+}(k-1))^2 - (f(k-1) - \hat{f}^{v-}(k-1))^2 + \varepsilon^v(k)}{2c_k^v \rho^v(k)} r_k^v \\
&= \frac{(f(k-1) - \hat{f}^{v+}(k-1) + f(k-1) - \hat{f}^{v-}(k-1))(\hat{f}^{v-}(k-1) - \hat{f}^{v+}(k-1)) + \varepsilon^v(k)}{2c_k^v \rho^v(k)} r_k^v \\
&= \frac{e^T(k)(\hat{f}^{v-}(k-1) - \hat{f}^{v+}(k-1))}{c_k^v \rho^v(k)} r_k^v \\
&= -\frac{e_k^T H(\hat{w}(k-1), x(k-1)) 2\Delta_k^v}{\rho^v(k)} r_k^v \\
&= -\frac{s^T(k)(1 - z^{-1}k_v) H(\hat{w}(k-1), x(k-1)) 2\Delta_k^v}{\rho^v(k)} r_k^v \tag{3.28}
\end{aligned}$$

where the bounded normalization factor, which is traditionally used in adaptive control system to bound the signals in learning algorithms [73], is

$$\begin{aligned}
\rho^v(k) &= \mu \rho^v(k-1) + \max\left(\frac{\alpha(k)}{p_v} \|H(\hat{w}(k-1), x(k-1))\|^2, \bar{\rho}\right) \\
&\quad \|\Delta_k^v\|^2 \|r_k^v\|^2, \bar{\rho})
\end{aligned}$$

with $\bar{\rho} > 0$, $\mu \in (0, 1)$.

The learning rate $\alpha(k)$ (for output layer) has an upper bound as defined below:

$$\alpha(k) < \frac{\rho_{max}^v}{p_v |h_{min}|^2 |\Delta_{min}^v|^2 |r_{min}^v|^2} \tag{3.29}$$

where $h_{min} = \min\{|h_{k-1,i}|\}$, $r_{min}^v = \min\{\|r_k^v\|\}$ and $\Delta_{min}^v = \min\{\|\Delta_k^v\|\}$ are the

non-zero minimum values of the activation function and perturbation vectors of the output layer, respectively.

$\varepsilon^v(k)$ is the bounded measurement error and has a relationship with the overall system disturbance as

$$\varepsilon(k) = \|\hat{f}^{v+}(k-1) - \hat{f}^{v-}(k-1)\|^{-2}(\hat{f}^{v+}(k-1) - \hat{f}^{v-}(k-1))\varepsilon^v(k) \quad (3.30)$$

Note that the gradient approximation function in Eq.(3.28) can be used only for the output layer as justified in remarks 1 and 2. Therefore, the parameter vector $\hat{w}(k)$ can be estimated by the SPSA algorithm with the gradient approximation $\hat{g}(\hat{w}(k-1), \hat{v}(k-1), \Delta_k^v)$, i.e.

$$\begin{aligned} \hat{v}(k) &= \hat{v}(k-1) - \alpha^v(k)\hat{g}(\hat{w}(k-1), \hat{v}(k-1), \Delta_k^v) \\ &= \hat{v}(k-1) + \alpha^v(k) \frac{s^T(k)(1 - z^{-1}k_v)H(\hat{w}(k-1), x(k-1))2\Delta_k^v}{\rho^v(k)} r_k^v \end{aligned} \quad (3.31)$$

Then stability analysis of the robust neural controller can be justified by the conic sector condition, which requires the feedback system in the Figure 3.3 below to meet certain dissipative conditions as in Theorem 3.1 and can be justified as in the following theorem.

Theorem 3.2: *The operator $H_1^v : s(k) \rightarrow \Phi^v(k)$, which represents the SPSA learning algorithm of the output layer(see Figure(3.3)), satisfies the conditions (a) and (b) of Theorem 3.1, i.e. $s(k), \Phi^v(k) \in L_2$.*

Figure 3.3: The equivalent error feedback system using the conic sector conditions: (a) For the tracking error $s(k)$; (b) For the estimation error $e(k)$

Proof: As an easy starting point, we establish the conic sector condition between the estimate error $e(k)$ and the output $\Phi^v(k) = -H(\hat{w}(k-1), x(k-1))\tilde{v}(k)$ first and then extend to the tracking error $s(k)$ later. Using the learning law, we have

$$\begin{aligned}
 & \|\tilde{v}(k)\|^2 - \|\tilde{v}(k-1)\|^2 \\
 &= -2\{\alpha^v(k)e^T(k)H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}\tilde{v}(k-1)\}(\rho^v(k))^{-1} \\
 & \quad + (\alpha^v(k)e^T(k)H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}(\rho^v(k))^{-1})^2 \\
 &\leq 2\{\alpha^v(k)\} | -e^T(k)H(\hat{w}(k-1), x(k-1))\tilde{v}(k-1) | \\
 & \quad |\tilde{v}(k-1)^T(\tilde{v}(k-1)\tilde{v}(k-1)^T)^{-1}\Delta_k^v r_k^{vT}\tilde{v}(k-1)|(\rho^v(k))^{-1} \\
 & \quad + (\alpha^v(k)e^T(k)H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}(\rho^v(k))^{-1})^2 \tag{3.32}
 \end{aligned}$$

Consider the property of gradient of the square error signal around the attractor basin [70]

$$\begin{aligned} -\frac{\partial(e^T(k)e(k))}{\partial(\hat{v}^T(k-1))}\tilde{v}(k-1) &= -e^T(k)H(\hat{w}(k-1), x(k-1))\tilde{v}(k-1) \\ &= e^T(k)\Phi^v(k) \end{aligned} \quad (3.33)$$

and the trace property

$$\begin{aligned} &|\tilde{v}^T(k-1)(\tilde{v}(k-1)\tilde{v}^T(k-1))^{-1}\Delta_k^v r_k^{vT}\tilde{v}(k-1)| \\ &= |tr\{\tilde{v}^T(k-1)(\tilde{v}(k-1)\tilde{v}^T(k-1))^{-1}\Delta_k^v r_k^{vT}\tilde{v}(k-1)\}| \end{aligned}$$

and

$$\begin{aligned} |tr\{\tilde{v}(k-1)\tilde{v}^T(k-1)(\tilde{v}(k-1)\tilde{v}^T(k-1))^{-1}\Delta_k^v r_k^{vT}\}| &= |tr\{\Delta_k^v r_k^{vT}\}| \\ &\triangleq p_v \end{aligned} \quad (3.34)$$

Eq.(3.32) can be rewritten as

$$\begin{aligned} \|\tilde{v}(k)\|^2 - \|\tilde{v}(k-1)\|^2 &\leq 2\alpha^v(k)p_v\{e^T(k)\Phi^v(k)\}(\rho^v(k))^{-1} + \\ &\quad \|(\alpha^v(k)e^T(k)H(\hat{w}(k-1), x(k-1)) \\ &\quad \Delta_k^v r_k^{vT}(\rho^v(k))^{-1})\|^2 \end{aligned}$$

Summing the above equation upon to N steps, we are able to establish the conic

condition with a constant $\check{\sigma}^v$ for the estimate error $e(k)$ as the input in Theorem 3.1 and the output $\Phi^v(k)$:

$$\sum_{k=1}^N \{e^{nT}(k)\check{\Phi}^{vn}(k) + e^{nT}(k)e^n(k)\frac{\check{\sigma}^v}{2}\} \geq -(\tilde{v}(0))^2 \left\{ \frac{\check{\rho}^v(k)}{2\alpha^v(k)p_v} \right\} \quad (3.35)$$

by selecting a suitable normalized factor $\check{\rho}^v(k)$ to obtain the constant number $\check{\sigma}^v$ such that

$$1 > \check{\sigma}^v \geq \frac{\alpha^v(k)}{p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 (\check{\rho}^v(k))^{-1} \quad (3.36)$$

Note that the conic condition (3.35) guarantees that the normalized estimate error $e^n(k) = e(k)\sqrt{\check{\rho}^v(k)}$ and the normalized output $\check{\Phi}^{vn}(k) = \Phi^v(k)\sqrt{\check{\rho}^v(k)} = -H(\hat{w}(k-1), x(k-1))\sqrt{\check{\rho}^v(k)}$ are bounded according to the Theorem 3.1, and in turn, the original $e(k)$ and $\Phi^v(k)$ with the parameter estimate error $\tilde{v}(k)$ [73]. To establish the conic sector condition of the tracking error $s(k)$, we can further extend

Eq.(3.32) into

$$\begin{aligned}
& \|\tilde{v}(k)\|^2 - \|\tilde{v}(k-1)\|^2 \\
& \leq 2\alpha^v(k)p_v\{e^T(k)\Phi^v(k)\}(\rho^v(k))^{-1} + \\
& \quad \|(\alpha^v(k)e_k^T H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}(\rho^v(k))^{-1})\|^2 \\
& = 2\alpha^v(k)p_v\{s^T(k)\Phi^v(k) - k_v s^T(k-1)\Phi^v(k)\}(\rho^v(k))^{-1} + \\
& \quad \|(\alpha^v(k)H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}(\rho^v(k))^{-1})\|^2 \|s(k) - k_v s(k-1)\|^2 \\
& \leq 2\alpha^v(k)p_v\{s^T(k)\Phi^v(k)\}(\rho^v(k))^{-1} + \alpha(k)p_v k_v \|s^T(k-1)\|^2 (\rho^v(k))^{-2} + \\
& \quad \alpha(k)p_v k_v \|\Phi^v(k)\|^2 + \|(\alpha^v(k)H(\hat{w}(k-1), x(k-1))\Delta_k^v r_k^{vT}(\rho^v(k))^{-1})\|^2 (\|s(k)\|^2 \\
& \quad + \|k_v s(k-1)\|^2 + \|s(k)\|^2 + \|k_v s(k-1)\|^2) \\
& \leq 2\alpha^v(k)p_v\{s^T(k)\Phi^v(k)\}(\rho^v(k))^{-1} + (\alpha(k)p_v(\rho^v(k))^{-2}(k_v)^{-1} + \\
& \quad 2(\alpha^v(k)(\rho^v(k))^{-1})^2 \|H(\hat{w}(k-1), x(k-1))\|^2 \|\Delta_k^v\|^2 \|r_k^v\|^2) \|k_v s^T(k-1)\Phi^v(k)\|^2 \\
& \quad \|(\Phi^v(k))^T[\Phi^v(k)(\Phi^v(k))^T]^{-1}\|^2 + \alpha^v(k)p_v k_v \|H(\hat{w}(k-1), x(k-1))\|^2 \|\tilde{v}(k)\|^2 \\
& \quad + 2\|s^T(k)\|^2 (\alpha^v(k)(\rho^v(k))^{-1})^{-2} \|H(\hat{w}(k-1), x(k-1))\|^2 \|\Delta_k^v\|^2 \|r_k^v\|^2 \\
& \leq 2\alpha^v(k)p_v\{s^T(k)\Phi^v(k)\}(\rho^v(k))^{-1} + \alpha^v(k)p_v k_v \|H(\hat{w}(k-1), x(k-1))\|^2 \|\tilde{v}(k)\|^2 \\
& \quad + [\alpha^v(k)p_v(\rho^v(k))^{-2}(k_v)^{-1}\bar{\Phi}^v + \{2\bar{\Phi}^v + 2\}(\alpha^v(k)(\rho^v(k))^{-1})^{-2} \\
& \quad \|H(\hat{w}(k-1), x(k-1))\|^2 \|\Delta_k^v\|^2 \|r_k^v\|^2] \|s^T(k)\|^2
\end{aligned}$$

where $\bar{\Phi}^v = \max(\|\Phi^v(k)\|^4 \|\Phi^v(k)(\Phi^v(k))^T\|^{-2})$, the second last inequality is due to the fact $-2(\rho^v(k))^{-1}s^T(k-1)\Phi^v(k) \leq \|s(k-1)\|^2(\rho^v(k))^{-2} + \|\Phi^v(k)\|^2$, the last inequality is from $e^T(k)\Phi^v(k) \geq 0 \Rightarrow s^T(k)\Phi^v(k) \geq s^T(k-1)\Phi^v(k)$. Then we can

rewrite the above into

$$\begin{aligned}
& -\frac{k_v \|H(\hat{w}(k-1), x(k-1))\|^2 \|\tilde{v}(k)\|^2}{2} + \frac{\|\tilde{v}(k)\|^2 - \|\tilde{v}(k-1)\|^2}{2\alpha^v(k)p_v} \\
& \leq \{s^T(k)\Phi^v(k)(\rho^v(k))^{-1}\} + \frac{\|s(k)\|^2}{2}(\rho^v(k))^{-2} \left\{ \frac{\bar{\Phi}^v}{k_v} + 2(1 + \bar{\Phi}^v) \frac{\alpha^v(k)}{p_v} \right. \\
& \quad \left. \|H(\hat{w}(k-1), x(k-1))\|^2 \|\Delta_k^v\|^2 \|r_k^v\|^2 \right\} \\
& \leq \{s^{nT}(k)\Phi^{vn}(k)\} + \frac{\|s^n(k)\|^2 \sigma^v}{2} \tag{3.37}
\end{aligned}$$

with the normalized tracking error $s^n(k) = s(k)\sqrt{\rho^v(k)}$ and the normalized output $\Phi^{vn}(k) = \Phi^v(k)\sqrt{\rho^v(k)} = -H(\hat{w}(k-1), x(k-1))\tilde{v}(k)\sqrt{\rho^v(k)}$ by choosing the normalized factor $\rho^v(k)$ to guarantee that there exists a constant σ^v such that

$$\begin{aligned}
0 & \leq (\rho^v(k))^{-2} \left\{ \frac{\bar{\Phi}^v}{k_v} + 2(1 + \bar{\Phi}^v) \frac{\alpha^v(k)}{p_v} \|H(\hat{w}(k-1), x(k-1))\|^2 \|\Delta_k^v\|^2 \|r_k^v\|^2 \right\} \\
& \leq \sigma^v < 1
\end{aligned}$$

Summing the above equation to N steps, we have

$$\begin{aligned}
& \sum_{k=1}^N \left\{ s^{nT}(k)\Phi^{vn}(k) + s^{nT}(k)s^n(k) \frac{\sigma^v}{2} \right\} \geq -\frac{(\tilde{v}_0)^2}{2\alpha^v(k)p_v} \\
& - \sum_{k=1}^N \frac{k_v \|H(\hat{w}(k-1), x(k-1))\|^2 \|\tilde{v}(k)\|^2}{2} \tag{3.38}
\end{aligned}$$

Note that the last item in the left hand side of Eq.(3.38) is bounded and furthermore, the specified normalized factor $\rho^v(k)$ plays two important roles. Firstly, it guarantees $\sigma^v < 1$ to avoid the so-called vanished cone problem [73]. Secondly, it guarantees the sector conditions of Theorem 3.1 to be simultaneously satisfied by both the original

feedback system in Figure (3.1) and the normalized equivalent feedback system in Figure (3.3) [72, 73, 84]. Therefore, both conditions of Theorem 3.1 are fulfilled (see remark 2 in this chapter for a discussion on operator H_2).

Remark 3.3: According to the stability conditions above, the stability of the control system in Figure 3.1 depends on the fixed gain parameter k_v , learning rate α_k^v and the number p_v of neurons of the output layer of the SPSA training algorithm. A smaller learning rate and larger number of neurons may increase the stability of the control system. It is interesting to note that the parameter c_k^v does not affect the system's stability, which is more relaxed when compared with the Spall's SPSA convergence proof.

3.5 Conic Sector Condition for the Hidden Layer Training

As justified in Remark 1, the hidden layer parameters of the network should be estimated separately. Therefore, a conic sector condition will be established for the hidden layer training in this section so that we can prove stability of the whole

system. Similar to the error equation (3.21)

$$\begin{aligned}
s(k) &= H_2 e(k) \\
&= H_2 (f(k-1) - \hat{f}(k-1) + \varepsilon(k)) \\
&= H_2 (H(w^*, x(k-1))v^* - H(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + \varepsilon(k)) \\
&= H_2 (H(w^*, x(k-1))v^* - H(w^*, x(k-1))\hat{v}(k-1) + H(\hat{w}^*, x(k-1)) \\
&\quad \hat{v}(k-1) - H(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + \varepsilon(k)) \\
&= H_2 (\tilde{H}(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + H(w^*, x(k-1))\tilde{v}(k-1) + \varepsilon(k)) \\
&= H_2 [\tilde{H}(\hat{w}(k-1), x(k-1))\hat{v}(k-1) + H(w^*, x(k-1))\tilde{v}(k-1) + \varepsilon(k)] \\
&= -H_2 \Phi^w(k) + \tilde{e}^w(k)
\end{aligned}$$

with $\tilde{e}^w(k) = H_2 (H(w^*, x(k-1))\tilde{v}(k-1) + \varepsilon(k))$ and $\Phi^w(k) = -\tilde{\Omega}(k-1)\tilde{w}(k-1)$.

where $\tilde{H}(\hat{w}(k-1)) = H(w^*, x(k-1)) - H(\hat{w}(k-1), x(k-1))$, $\tilde{v}_{k-1,i} = v_i^* - \hat{v}(k-1, i)$, $\tilde{w}_{k-1,i} = w_i^* - \hat{w}(k-1, i)$, and $\tilde{\Omega}(k-1) \in R^{m \times p_w}$ is the following matrix

$$\tilde{\Omega}_{k-1} = \begin{bmatrix} \tilde{\mu}_{k-1,1} x^T(k-1, 1) \hat{v}(k-1, (1, 1)), \dots, \tilde{\mu}_{k-1,n_h} x^T(k-1, n_i) \hat{v}(k-1, (1, n_h)) \\ \vdots \\ \tilde{\mu}_{k-1,1} x^T(k-1, 1) \hat{v}(k-1, (m, 1)), \dots, \tilde{\mu}_{k-1,n_h} x^T(k-1, n_i) \hat{v}(k-1, (m, n_h)) \end{bmatrix}$$

$\in R^{m \times p_w}$ with $\hat{v}(k-1, i) \in R^{n_h}$, $\hat{v}(k-1) = [\hat{v}^T(k-1, 1), \dots, \hat{v}^T(k-1, m)]^T$ where n_h is the number of neurons in the hidden layer of the network.

Note the above equation is derived from the mean value theorem and the activation

function is a non-decreasing function, so there exist unique positive numbers $\tilde{\mu}_{k-1,i}$

$$h_{k-1,i}(w_i^*, x(k-1)) - h_{k-1,i}(\hat{w}(k-1, i), x(k-1)) = \tilde{\mu}_{k-1,i} x^T(k-1)(w_i^* - \hat{w}(k-1, i)) \quad (3.39)$$

where $\hat{w}(k-1, i), w_i^* \in R^{n_i}$ are the estimation and ideal weight vectors linked to the i th hidden layer neuron respectively. The maximum value of the derivative $h'_{k-1,i}$ is λ , therefore

$$\lambda \geq \tilde{\mu}_{k-1,i} \geq 0 (1 \leq i \leq n_h) \quad (3.40)$$

After we define

$$\hat{f}^{w+}(k-1) = H(\hat{w}(k-1) + c_k^w \Delta_k^w, x(k-1)) \hat{v}(k-1) \quad (3.41)$$

$$\hat{f}^{w-}(k-1) = H(\hat{w}(k-1) - c_k^w \Delta_k^w, x(k-1)) \hat{v}(k-1) \quad (3.42)$$

Then the normalized gradient approximation of the hidden layer can be written as

$$\begin{aligned}
& \hat{g}(\hat{v}(k-1), \hat{w}(k-1), \Delta_{k-1}^w) \\
&= \frac{L(\hat{v}(k-1), \hat{w}(k-1) + c_k^w \Delta_k^w) - L(\hat{v}(k-1), \hat{w}(k-1) - c_k^w \Delta_k^w) + \varepsilon_k^w}{2c_k^w \rho^w(k)} r_k^w \\
&= \frac{(f(k-1) - \hat{f}^{w+}(k-1))^2 - (f(k-1) - \hat{f}^{w-}(k-1))^2 + \varepsilon_k^w}{2c_k^w \rho^w(k)} r_k^w \\
&= \frac{(f(k-1) - \hat{f}^{w+}(k-1) + f(k-1) - \hat{f}^{w-}(k-1))(\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1)) + \varepsilon_k^w}{2c_k^w \rho^w(k)} r_k^w \\
&= \frac{e^T(k)(\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1))}{c_k^w \rho^w(k)} r_k^w \\
&= \frac{e^T(k) c_k^w \Delta_k^w \hat{\Omega}(k-1)}{c_k^w \rho^w(k)} r_k^w \\
&= \frac{s^T(k)(1 - k_v z^{-1}) c_k^w \Delta_k^w \hat{\Omega}(k-1)}{c_k^w \rho^w(k)} r_k^w \tag{3.43}
\end{aligned}$$

where $\hat{\Omega}(k-1)$ is defined similarly to $\tilde{\Omega}(k-1)$.

Then the SPSA algorithm of the hidden layer becomes

$$\hat{w}(k) = \hat{w}(k-1) + \alpha^w(k) \frac{s^T(k)(1 - k_v z^{-1})(\hat{f}^{w+}(k-1) - \hat{f}^{w-}(k-1))}{c_k^w \rho^w(k)} r_k^w \tag{3.44}$$

Theorem 3.3: *The operator $H_1^w : s(k) \rightarrow \Phi^w(k)$, which represents the SPSA learning algorithm of the hidden layer, satisfies the conditions (a) and (b) of Theorem 3.1.*

Proof: Using the property of local minimum points of the gradient

$-\frac{\partial e^T(k)e(k)}{\partial(\hat{w}(k-1,i,n_i))}\tilde{w}_{k-1,i,n_i} \geq 0$ [70]. We have

$$\begin{aligned}
0 &\leq -\frac{\partial e^T(k)e(k)}{\partial(\hat{w}^T(k-1))}\tilde{w}_{k-1} = \sum_{i=1}^{n_h} \left\{ \frac{\partial e^T(k)e(k)}{\partial(\hat{w}_{k-1,i})^T} \tilde{w}_{k-1,i} \right\} \\
&= \sum_{i=1}^{n_h} \{h'_{k-1,i} e^T(k) \hat{v}(k-1, i) x^T(k-1) \tilde{w}_{k-1,i}\} \\
&= \sum_{i=1}^{n_h} \left\{ \frac{h'_{k-1,i}}{\tilde{\mu}_{k-1,i}} \tilde{\mu}_{k-1,i} e^T(k) \hat{v}(k-1) x(k-1)^T \tilde{w}_{k-1,i} \right\} \\
&\leq \frac{\lambda}{\lambda_{min}} \sum_{i=1}^{n_h} \{ \tilde{\mu}_{k-1,i} e^T(k) \hat{v}(k-1) x^T(k-1) \tilde{w}_{k-1,i} \} \\
&= -\frac{\lambda}{\lambda_{min}} e^T(k) \Phi^w(k)
\end{aligned} \tag{3.45}$$

where $\hat{v}(k-1, i) \in R^m$, $\hat{v}(k-1) = [\hat{v}_{k-1,1}^T, \dots, \hat{v}_{k-1,n_h}^T]^T$ and $\tilde{w}_{k-1,i} = w_{k-1,i}^* - \hat{w}(k-1, i) \in R^{n_i}$ are weight vector components of the hidden and output layers linked to the i th hidden layer neuron, $\lambda \geq h'_{k-1,i}$ is the maximum value of the derivative $h'_{k-1,i}$ of the activation function in Eq.(3.19), and $\lambda_{min} \neq 0$ is the minimum non-zero value of the parameter $\tilde{\mu}_{k-1,i}$ defined in $\tilde{\Omega}(k-1)$.

Using Eq.(3.44) and (3.45), then similar to the proof of Theorem 3.2, we have

$$\begin{aligned}
& \|\tilde{w}(k)\|^2 - \|\tilde{w}(k-1)\|^2 \\
&= -\{2\alpha^w(k) \frac{e^T(k)(\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1))}{\rho^w(k)c_k^w} (r_k^w)^T \tilde{w}(k-1)\} + \\
& \quad \|\alpha^w(k) \frac{e^T(k)(\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1))}{\rho^w(k)c_k^w} (r_k^w)^T\|^2 \\
&= \sum_{i=1}^{n_h} \left\{ \frac{\mu_{k-1,i}}{h'_{k-1,i}} (-h'_{k-1,i} e^T(k) \hat{v}(k-1) x^T(k-1) \tilde{w}_{k-1,i}) \right\} (\tilde{w}^T(k-1) (\tilde{w}(k-1) \\
& \quad \tilde{w}^T(k-1))^{-1} \Delta_k^w r_k^w \tilde{w}(k-1)) \frac{2\alpha^w(k)}{\rho^w(k)} + \\
& \quad \|\alpha^w(k) \frac{e^T(k)(\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1))}{\rho^w(k)c_k^w} (r_k^w)^T\|^2 \\
&\leq 2\alpha^w(k) \left(\frac{\lambda}{\lambda_{min}}\right)^2 \frac{p_w}{\rho^w(k)} e^T(k) \Phi^w(k) + \|e^T(k)\|^2 \|\hat{f}^{w-}(k-1) - \\
& \quad \hat{f}^{w+}(k-1)\|^2 \|r_k^w\|^2 \left(\frac{\alpha^w(k)}{\rho^w(k)}\right)^2 \\
&\leq 2\alpha^w(k) \left(\frac{\lambda}{\lambda_{min}}\right)^2 \frac{p_w}{\rho^w(k)} s^T(k) \Phi^w(k) + \alpha^w(k) k_v \left(\frac{\lambda}{\lambda_{min}}\right)^2 p_w \|\Phi^w(k)\|^2 \\
& \quad + \|s(k)\|^2 [\alpha^w(k) \left(\frac{\lambda}{\lambda_{min}}\right)^2 \frac{p_w \bar{\Phi}^w}{k_v (\rho^w(k))^{-2}} + 2(1 + \bar{\Phi}^w) \|\hat{f}^{w-}(k-1) - \\
& \quad \hat{f}^{w+}(k-1)\|^2 \|r_k^w - k_v r_{k-1}^w\|^2 \left(\frac{\alpha^w(k)}{\rho^w(k)}\right)^2]
\end{aligned}$$

Note that we take the fact that $\lambda \geq \mu_{k-1,i}$ is the maximum value of the derivative and the increment of the activation function. Similar to section 3, we have

$$\begin{aligned}
& -\frac{k_v \left(\frac{\lambda}{\lambda_{min}}\right)^2 \|\tilde{\Omega}(k-1)\|^2 \|\tilde{w}(k-1)\|^2}{2} + \frac{(\|\tilde{v}(k)\|^2 - \|\tilde{v}(k-1)\|^2) \left(\frac{\lambda}{\lambda_{min}}\right)^2}{2\alpha^w(k) p_w} \\
&\leq \{s^T(k) \Phi^w(k) (\rho^w(k))^{-1}\} \\
& \quad + \frac{\|s(k)\|^2 (\rho^w(k))^{-2} \left\{ \frac{\bar{\Phi}^w}{k_v} + 2(1 + \bar{\Phi}^w) \|\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1)\|^2 \|r_k^w\|^2 \right\}}{2} \\
&\leq \{s^{nT}(k) \Phi^{wn}(k)\} + \frac{\|s^n(k)\|^2}{2} \tag{3.46}
\end{aligned}$$

under the condition $\bar{\Phi}^w = \max(\|\Phi^w(k)\|^4 \|\Phi^w(k)(\Phi^w(k))^T\|^{-2})$ and choosing the normalized factor $\rho^w(k)$ to guarantee that there exists a constant σ^w such that

$$0 < (\rho^w(k))^{-1} \left\{ \frac{\bar{\Phi}^w}{k_v} + 2(1 + \bar{\Phi}^w) \frac{\alpha^w(k)}{p_w} \left(\frac{\lambda_{min}}{\lambda} \right)^2 \|\hat{f}^{w-}(k-1) - \hat{f}^{w+}(k-1)\|^2 \|r_k^w\|^2 \right\} \leq \sigma^w < 1 \quad (3.47)$$

Summing the above equation to N steps we have

$$\begin{aligned} \sum_{k=1}^N s^{nT}(k) \Phi^{nw}(k) + \frac{1}{2} \sigma^w s^{nT}(k) s^n(k) \geq \\ -\frac{1}{2} (\tilde{w}(0) \frac{\lambda_{min}}{\lambda})^2 - \sum_{k=1}^N \frac{k_v \left(\frac{\lambda_{min}}{\lambda} \right)^2 \|\tilde{\Omega}(k-1)\|^2 \|\tilde{w}(k-1)\|^2}{2} \end{aligned} \quad (3.48)$$

The conic sector stability of the closed loop SPSA controller is closely linked to the convergence property of the SPSA training algorithm in a sense of the traditional deterministic neural control system approach. For example, there are a few important properties related to the established conic sector conditions for the robust neural controller and the convergence properties of the SPSA closed-loop training algorithm.

3.6 Discussion of the Convergence Properties of the Specified SPSA Controller

- (a) In addition to the stability proof, one of the most interesting contributions of this chapter is to reveal the relationship between the conventional adaptive control system and generalization theory, which is mainly developed for the neural network pattern recognition system and, to the best of our knowledge, is not widely acknowledged by the control community. A relatively larger learning rate will contribute to faster convergence speed of the SPSA training algorithm, and our equations(3.29) reveals that a relatively small number of neurons will yield relatively larger optimal rates. This concept is closely linked to the generalization property of neural network theory. While the theoretical upper bounds for the learning rate may not be computed, they serve the purpose of illuminating the generalization property of neural network theory. And in the next chapter, we will discuss how to choose an optimal network structure in this online control system.
- (b) The stability condition of the hidden layer training is weaker than the condition of the output layer training in the sense that the constant σ^w is not always satisfying the requirements since the minimum value $\lambda_{min} \rightarrow 0$. This is logical as the output of the hidden layer is always bounded due to the nature of the nonlinear activation function in the hidden layer defined in Eq.(3.19). In addition, the tracking signal $s(k)$ should be bounded as in Theorem 3.2 for

the output layer training without influence from the hidden layer at all. The important role of Theorem 3.3 is that the parameter estimation error $\tilde{w}(k-1)$ should be bounded to meet the stability condition.

- (c) The SPSA measurement noises ε_k^v and ε_k^w are linked dynamically and proportionally to the overall control system noise, i.e.

$$\begin{aligned}
 \varepsilon_k &= \|\hat{f}^{v+}(k-1) - \hat{f}^{v-}(k-1)\|^{-2}(\hat{f}^{v+}(k-1) - \hat{f}^{v-}(k-1))\varepsilon_k^v \\
 &= \frac{(H(\hat{w}(k-1), x(k-1))\Delta_k^v)\varepsilon_k^v}{\|H(\hat{w}(k-1), x(k-1))\Delta_k^v\|^2 2c_k^w} \\
 \varepsilon_k &= [(f(k-1) - \hat{f}(k-1)) + (\hat{f}(k-1) - \hat{f}^{w-}(k-1)) + (\hat{f}(k-1) - \\
 &\quad \hat{f}^{w-}(k-1))] - \|\hat{f}^{w+}(k-1) - \hat{f}^{w-}(k-1)\|^{-2}(\hat{f}^{w+}(k-1) - \\
 &\quad \hat{f}^{w-}(k-1))\varepsilon_k^w
 \end{aligned} \tag{3.49}$$

for the output and hidden layers, respectively, which are always bounded as long as the closed-loop system is stable in the sense that the conic sector conditions in Theorem 3.2 and 3.3 are satisfied.

Theorem 3.1, 3.2 and 3.3 guarantee signals $s(k)$, $\Phi^v(k-1)$, $\Phi^w(k-1) \in L_2$, therefore, the original feedback system in Figure(3.1) is stable in the sense of bounded input-output error signals for the SPSA estimation.

The robust neural control algorithm can be summarized as (refer to Figure(3.1)):

Summary of the SPSA Training Algorithm for Neural Controller

Step 1.

Initializing: Form the new input vector $x(k-1)$ of the neural network defined in Eq.(3.18);

Step 2.

Calculating the output $\hat{f}(k-1)$ of the neural network: Use the input state $x(k-1)$ and the existing or initial weights of the network in the first iteration;

Step 3.

The control input $u(k-1)$ is calculated based on Eq.(3.11);

Step 4.

The new measurement of the system dynamics is taken and the measurable tracking error signal $s(k)$ is fed through a fixed filter to produce the implicit training error signal $e(k)$ of the network;

Step 5.

The tracking error $s(k)$ is used directly to train the neural network and calculates the new weights $\hat{v}(k)$ and $\hat{w}(k)$ using the learning law in Eq.(3.31) and (3.44) for the output and hidden layers, respectively, of the next iteration;

Step 6.

Go back to Step 1 to continue the iteration.

3.7 Simulation Results

One or two-link robot arm model is extensively used in the literature of neural control applications [70, 71, 85, 86] because it is a simple, yet effective example to illustrate all the nonlinear terms arising in a general n-link manipulator. A dynamic equation of the two-link direct drive manipulator is given in matrix form [68] by

$$Tor = M(y)\ddot{y} + V(y, \dot{y}) + F(\dot{y}) \quad (3.50)$$

where the vector $y, \dot{y}, \ddot{y}$ are the joint angle, the joint angular velocity and joint angular acceleration, respectively. $M(y) \in R^{2 \times 2}$ is configuration dependent inertia matrix, $V(y, \dot{y})$ is a 2×1 vector representing centrifugal and coriolis effects, and $F(\dot{y})$ is the 2×1 vector representing coulomb friction.

From the practical point of view, robots are controller by digital computers on discrete time basis. Digital implementation of a solution based on continuous time model can result in degradation of performance and the closed loop system can even become unstable, especially when the sampling time is not small. In order to design the digital controller, a discrete time robot is required, one of the approaches [68] is based on the energy conservation theorem for the Lagrange equations. In the following sections we apply the above mentioned methods to derive the discrete

time models for the direct drive robot.

3.7.1 Euler's Method

When a robot manipulator is controlled by a digital controller, then interfaces such as D/A and A/D converters are needed to couple the plant with the computer. Therefore, the input to the plant is updated at every sampling instant and maintained constant by the converters:

$$Tor(kT + t) = Tor(kT) \quad (3.51)$$

for $0 \leq t < T$ with T is the sampling time. The A/D converters sampling the output state variables of the plant at every instant produce the piecewise constant states:

$$\begin{aligned} y(kT + t) &= y(kT) \\ \dot{y}(kT + t) &= \dot{y}(kT) \end{aligned} \quad (3.52)$$

For simulation and/or control purpose, what we need is a model of the discrete time system consisting of the cascade connection of D/A, holder, robot, sampler and A/D. Usually, approximate model of the discrete time robot are derived by applying numerical discretization procedure to dynamic equation. The most popular approach is the Euler's rule [68]. Using this method, the following equations are

obtained:

$$y(k+1) = y(k) + T\dot{y}(k)$$

$$\dot{y}(k+1) = \dot{y}(k) + T\ddot{y}(k)$$

So consequently the following state space representation of the discrete time model obtained from Euler's method can be presented:

$$\begin{aligned} [y(k+1), \dot{y}(k+1)]^T &= [y(k) + T\dot{y}(k), \dot{y}(k) - TM^{-1}(y(k))[V(y(k), \dot{y}(k)) + F(\dot{y}(k))]]^T \\ &+ [0, TM^{-1}(y(k))]^T * Tor(k) \end{aligned} \quad (3.53)$$

And then we can get the discrete time dynamic model as follows:

$$\begin{aligned} y(k+1, 1) &= y(k, 1) + T * y(k, 2) \\ y(k+1, 2) &= y(k, 2) - T * M^{-1}(y(k, 1)) * [V(y(k, 1), y(k, 2)) \\ &\quad + F(y(k, 1), y(k, 2))] + T * M^{-1}(y(k, 1)) * Tor(k) \\ &= f(y(k, 1), y(k, 2)) + U(k) + \varepsilon(k) \end{aligned} \quad (3.54)$$

where $y(k) = [y(k, 1), y(k, 2)]^T$ and $y(k, 1)$ and $y(k, 2)$ are the joint angle and velocity vectors, respectively, T is the sampling time, $U(k) = T * M^{-1}(y(k, 1)) * Tor(k)$ is the torque control vector signal and here an assumption is made that the $M^{-1}(y(k, 1))$ is approximately known to compute the $U(k)$, $\varepsilon(k)$ is a normally distributed disturbance with a bound $\|\varepsilon(k)\| \leq 0.2$, and the nonlinear function $f(y(k), y(k-1)) =$

$y(k, 2) - T * M^{-1}(y(k, 1))[V(y(k, 1), y(k, 2)) + F(y(k, 1), y(k, 2))]$ with the configuration dependent inertia matrix

$$M(y(k, 1)) = \begin{bmatrix} 3.32 + 0.32\cos(y(k, 1)) & 0.12 + 0.16\cos(y(k, 1)) \\ 0.12 + 0.16\cos(y(k, 1)) & 0.12 \end{bmatrix}$$

centrifugal and coriolis effect $V(y(k), y(k+1)) =$

$$\begin{bmatrix} -(y(k+1, 2) - y(k, 2))(2(y(k+1, 1) - y(k, 1)) + y(k+1, 2) - y(k, 2))0.16\sin(y(k, 2))/T^2 \\ 0.16(y(k+1, 1) - y(k, 1))^2\sin(y(k, 2))/T^2 \end{bmatrix}$$

and coulomb friction

$$F(y(k), y(k+1)) = \begin{bmatrix} 5.3\operatorname{sgn}((y(k+1, 1) - y(k, 1))/T) \\ 1.1\operatorname{sgn}((y(k+1, 2) - y(k, 2))/T) \end{bmatrix}$$

with $T=0.002s$ being the sampling time.

A three-layered neural network is used as defined in section 2 of this chapter for this simulation study with 30 hidden layer neurons and two output neurons, which was trained by the standard back propagation and SPSA training algorithm with same control structures as shown in Figure(3.1). The desired joint trajectory is selected as

$$d(k) = [d_{k1}, d_{k2}]^T = [(\pi/4)\sin(\pi \times kT), (\pi/4)\cos(\pi \times kT)]^T \quad (3.55)$$

which is shown as Figure(3.4). Then the tracking error is defined as:

$$s(k) = y(k, 1) - d(k, 1) \quad (3.56)$$

Define the control signal vector as:

$$U(k-1) = -\hat{f}(k) + y(k, 2) + k_v * s(k) \quad (3.57)$$

where the sampling time is $T=0.002\text{sec}$ and the linear gain parameter of the fixed controller was given as $k_v = \text{diag}[0.5, 0.5]$, the $\hat{f}(k)$ is the output of the neural network. All the initial conditions are chosen to be zero.

We use a variable input tracking signal, which change the magnitude of the reference signal from $\pi/4$ to $\pi/3$ then back to $\pi/4$ at the 1000 and 1600 iterations, respectively. Note that for nonlinear control systems, the change of the input signal magnitude is equivalent to change of the system model setting since the superposition theory is no longer valid. To reveal the problem of standard back-propagation algorithm, the neural controller using the standard back-propagation algorithm performs worst because a relatively larger tracking errors are achieved as a result of the drifting weight, as shown in Figure (3.9), (3.10), and (3.11). Figure (3.5), (3.6) and (3.7) show the output of the plant and the error signal of the robust SPSA based neural controller.

The theoretical stability analysis has been carried out in this chapter for the pro-

posed algorithm, and here for the specific implementation, some other notes need to be note down. Taking the analysis for the output layer as an example, the conic condition would be satisfied only if the normalization factor is chosen properly, which will play two important roles. First, it guarantees the $0 < \check{\sigma}^v < 1$ to avoid the so called vanishing cone problem. Secondly, it also guarantees the condition (a) for the estimation error and output will be satisfied which in turn guarantees the tracking error would be bounded. So in the simulation part, it is critical important to decide this normalization factor for the output layer. According to Equation(3.36)

$$1 > \check{\sigma}^v \geq \frac{\alpha^v(k)}{p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 (\check{\rho}^v(k))^{-1} \quad (3.58)$$

So if

$$\check{\rho}^v(k) > \frac{\alpha^v(k)}{p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 \quad (3.59)$$

where $p_v = m * n_h$ with $m = 2$ denoting the dimension of the input and $n_h = 30$ is the number of neurons. Thus, at every step, if we choose the value of the normalization factor as bigger than $\frac{\alpha^v(k)}{p_v}$, and a good example is $\sqrt{1 + (\frac{\alpha^v(k)}{p_v})^2}$. The same strategy can be applied to the hidden layer training, thus it will lead to a guaranteed stability of the nonlinear system in the simulation process.

As discussed before, a suitable number of neurons with maximum learning rate, achieve a good generalization performance in terms of reduced control signal error and faster tracking performance, compared to the fixed PID controller with the

Figure 3.4: The reference signal for the plant.

initial optimal parameters before the change of setting point, shown in Figure (3.13), (3.14) and (3.15), with the drift free parameter estimate in Figure (3.8).

3.8 Concluding Remarks

In this Chapter, a robust neural controller based on the SPSA algorithm has been developed to obtain the guaranteed stability with a normalized learning algorithm. A complete stability analysis is performed for the closed-loop control system. Simulation results show that the proposed robust neural controller performs better than a neural controller based on the standard back-propagation algorithm or the PID controller.

And as discussed, a network with a minimum necessary number of neurons may be the best to approximate a nonlinear system rather than a over-fitted large network.

Figure 3.5: Output by using the robust SPSA based neural controller.

As shown in the simulation part, a suitable number of neurons (not necessary the biggest network) can have a good generalization performance in terms of reduced control signal error and fast tracking performance, which can be further developed into the adaptive pruning based algorithm for an optimal performance. This point will be explored further in the next chapter.

Figure 3.6: Tracking error by using the robust SPSA based neural controller.

Figure 3.7: Mean square error by using the robust SPSA based neural controller.

Figure 3.8: One weight $v_{1,1}$ of the output layer by using the robust SPSA based neural controller.

Figure 3.9: Output by using the standard BP based neural controller.

Figure 3.10: Tracking error by using the standard BP based neural controller.

Figure 3.11: Mean square error by using the standard BP based neural controller.

Figure 3.12: One weight $v_{1,1}$ of the output layer by using the standard BP based neural controller.

Figure 3.13: Output by using the PID based controller.

Figure 3.14: Tracking error by using the PID based controller.

Figure 3.15: Mean square error by using the PID based controller.

Chapter 4

Dynamic Weight-Pruning Based Training Algorithm for Neural Tracking Control Systems

4.1 Introduction

As justified in the previous chapter, a network with the minimum necessary number of neurons may be the best way to approximate a nonlinear system rather than a large over-fitted network. When a neural system is used to handle unlimited examples, including training data and testing data, an important issue is how well it generalizes to patterns of the testing data, which is known as generalization ability. The generalization ability of neural networks is very important and a rule of thumb

for good generalization in neural systems is that the smallest acceptable system should be used to fit the training data. When a network has too many free parameters (i.e. weights and/or units), it may end up by just memorizing the training patterns. Both theoretical [87] and practical results [88, 89] show that networks with minimal free parameters exhibit better generalization performance, which can be illustrated by recalling the analogy between neural network learning and curve fitting [21, 23, 90]. Moreover, knowledge embedded in smaller trained networks is presumably easier to interpret and thus the extraction of simple rules can hopefully be facilitated. Lastly, from an implementation standpoint, small networks only require limited resources in any physical computational environment. Unfortunately, it is normally difficult to determine the optimal size of networks, particularly, in the sequential training applications such as online control. In this chapter, an online training algorithm with a dynamic pruning procedure is proposed for the online tuning and pruning the neural tracking control system. Similar to Chapter 3, the conic sector theory is also introduced in the design of this robust neural control system, which aims at providing guaranteed boundedness for both the input-output signals and the weights of the neural network in the presence of certain disturbance and pruning. The neural network is trained by this algorithm in a closed-loop manner to provide an improved training and generalization performance over the standard back-propagation algorithm, in terms of guaranteed stability of the weights, which in turn, yields good tracking and generalization performance for the dynamical control system.

4.2 NN Tracking Controller and the Dynamic Training and Pruning Algorithm

4.2.1 Design of the Controller

As in the last chapter (see Chap3 for details), a dynamic control system is presented at an input-output form as the following:

$$y_{k+1} = f(y_k, \dots, y_{k-l}, u_{k-1}, \dots, u_{k-n}) + u_k + \varepsilon_k \quad (4.1)$$

where $y_k \in R^m$ is the output, $f(.) \in R^m$ is a dynamic nonlinear function follows the same settings as described in last chapter, $\varepsilon_k \in R^m$ denotes the system noise vector which is a bounded vector and $u_k \in R^m$ is the control signal vector with a unit discrete time delay, which can also be extended to a d-step delay system by using a linear predictor [82]. The tracking error of the control system can be defined as

$$s_k = y_k - d_k \quad (4.2)$$

where $d_k \in R^m$ is the command signal. Define the control signal as

$$u_k = -\hat{f}_k + d_{k+1} + k_v s_k \quad (4.3)$$

where k_v is the gain parameter of the fixed controller and $\hat{f}_k$ is the estimate of the nonlinear function $f(.)$ by the neural network. Then the error vector can be

presented as

$$e_k = f_{k-1} - \hat{f}_{k-1} + \varepsilon_k \quad (4.4)$$

to train the neural network as shown in Figure 4.1

Figure 4.1: Structure of the control scheme

Note that the training error e_k may not be directly measurable, so we should use the tracking error to generate it using the closed-loop relationship (4.1), (4.3) and (4.4)

$$e_k = (1 - z^{-1}k_v)s_k \quad (4.5)$$

4.2.2 Basic Form of the Training Algorithm

The Adaptive Simultaneous Perturbation(ASP) approach [37] is composed of two parallel recursions: one for the weights W and one for the Hessian of the loss

function, $L(W)$. The two core recursions are, respectively:

$$\hat{W}_k = \hat{W}_{k-1} - a_k(\overline{\overline{H}})_k^{-1}G_k(\hat{W}_{k-1}) \quad (4.6)$$

$$\overline{\overline{H}}_k = M_k(\overline{H}_{k-1}) \quad (4.7)$$

$$\overline{H}_k = \frac{k}{k+1}\overline{H}_{k-1} + \frac{1}{k+1}\hat{H}_k \quad (4.8)$$

where a_k is a nonnegative scalar gain coefficient, $G_k(\hat{W}_{k-1})$ is the input information related to the gradient or the gradient approximation. M_k is a mapping designed to cope with possible non-positive definiteness of $\overline{H}_k$ [37], and $\hat{H}_k$ is a per-iteration estimation of the Hessian discussed below. The parallel recursions can be implemented once $\hat{H}_k$ is specified and the formula for estimating the Hessian at each iteration can be presented as

$$\hat{H}_k = \frac{1}{2}[\frac{\delta G_k^T}{2c_k}r_k + (\frac{\delta G_k^T}{2c_k}r_k)^T] \quad (4.9)$$

where

$$\delta G_k = G_k^{(1)}(\hat{W}_k + c_k\Delta_k) - G_k^{(1)}(\hat{W}_k - c_k\Delta_k) \quad (4.10)$$

$$G_k^{(1)}(\hat{W}_k \pm c_k\Delta_k) = \frac{L(\hat{W}_k \pm c_k\Delta_k + \tilde{c}_k\tilde{\Delta}_k) - L(\hat{W}_k \pm c_k\Delta_k - \tilde{c}_k\tilde{\Delta}_k)}{2\tilde{c}_k} \tilde{r}_k \quad (4.11)$$

with $\Delta_k = (\Delta_{k1}, \Delta_{k1}, \dots, \Delta_{kp})^T$ is generated via Monte Carlo according to conditions specified in [37,59,60] and $r_k = (\Delta_{k1}^{-1}, \Delta_{k2}^{-1}, \dots, \Delta_{kn}^{-1})$. $\tilde{\Delta}_k = (\tilde{\Delta}_{k1}, \tilde{\Delta}_{k2}, \dots, \tilde{\Delta}_{kp})^T$ is generated in the same statistical manner as Δ_k , but independently of Δ_k , and

$\tilde{r}_k = (\tilde{\Delta}_{k1}^{-1}, \tilde{\Delta}_{k2}^{-1}, \dots, \tilde{\Delta}_{kp}^{-1})^T$. c_k is a positive scalar satisfying certain regularity conditions [37, 59, 60] and with $\tilde{c}_k$ satisfying conditions similar to c_k .

The output of a three-layer neural network can be presented as

$$\hat{f}_{k-1} = H(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} \quad (4.12)$$

where the input vector $x_{k-1} \in R^{n_i}$ of the neural network is

$$x_{k-1} = [y_{k-1}^T, y_{k-2}^T, \dots, u_{k-2}^T, u_{k-3}^T \dots]^T \quad (4.13)$$

$\hat{v}_{k-1} \in R^{p_v}$ is the weight vector of the output layer, and $\hat{w}_{k-1} \in R^{p_w}$ is the weight vector of the hidden layer of the neural network with $p_v = m \times n_h$ and $p_w = n_h \times n_i$, where n_i and n_h are the number of the neurons in the input and hidden layers of the network, respectively. $H(\hat{w}_{k-1}, x_{k-1}) \in R^{m \times P_v}$ is the nonlinear activation function matrix:

$$H(\hat{w}_{k-1}, x_{k-1}) = \begin{bmatrix} h_{k-1,1}, h_{k-1,2}, \dots, h_{k-1,n_h} & 0 \dots & \dots 0 \\ 0 \dots & h_{k-1,1}, h_{k-1,2}, \dots, h_{k-1,n_h} & \dots 0 \\ \dots & \dots & \dots \end{bmatrix}$$

with $h_{k-1,i}$ is the nonlinear activation function

$$h_{k-1,i} = h(x_{k-1}^T \hat{w}_{k-1,i}) = \frac{1}{1 + e^{-4\lambda x_{k-1}^T \hat{w}_{k-1,i}}} \quad (4.14)$$

with $\hat{w}_{k-1,i} \in R^{n_i}$, $\hat{w}_{k-1} = [\hat{w}_{k-1,1}^T, \dots, \hat{w}_{k-1,n_h}^T]^T$ and $4\lambda > 0$, which is the gain parameter of the threshold function.

4.2.3 Hessian Based Pruning

The idea of this approach is introduced, with details in the chapter 2. In this method, the weight corresponding to the smallest saliency

$$S_i = \frac{(\hat{W}(k, i))^2}{2[H_{i,i}^{-1}]} \quad (4.15)$$

is the one selected for deletion, where H^{-1} is the inverse of the Hessian matrix H , and $[H_{i,i}^{-1}]$ is the ii -th element of the inverse matrix. The value S_i optimized with respect to $\Delta\hat{W}(k)$, subject to the constraint that the i th synaptic weights $\hat{W}(k, i)$ be eliminated, is called the *saliency* of $\hat{W}(k, i)$.

Moreover, the corresponding optimal changes in the remainder of the weights are given as

$$\Delta\hat{W}(k) = -\frac{\hat{W}(k, i)}{[H_{i,i}^{-1}]} H^{-1} I_i \quad (4.16)$$

And in this chapter, the Hessian matrix is a dynamic term which contains the information of the latest L steps. This point is going to be discussed deeply in the next section. Until now, the idea of this novel Dynamic Training and Pruning (DTP) algorithm is going to be established. The perturbation is applied to update the weights of both layers by using adaptive simultaneous perturbation algorithm

and related useful information is extracted in this process to do the pruning when some criteria is satisfied. The detailed steps for implementing this novel algorithm are illustrated next.

Remark 4.1: Similar to Chapter 3, we could define the overall estimate parameter vector $\hat{W}_k = [\hat{v}_k^T \ \hat{w}_k^T] \in R^p$ with $p = p_v + p_w$ in the DTP algorithm. However, for a multilayered neural network, it may not be possible to update all the estimated parameters with a single gradient approximation function to meet the stability requirement. Therefore, it is better that the estimated parameter vectors $\hat{v}_k$ and $\hat{w}_k$ are updated separately in the DTP algorithm using different gradient approximation functions as in the standard BP training algorithm. This point will be explored further in the robustness analysis in the next section.

4.2.4 The DTP Algorithm

Summary of the DTP Algorithm for Neural Controller

Step 1.

Initializing : Form the new input vector x_{k-1} of the neural network defined in Eq.(4.13);

Step 2.

Calculating the output $\hat{f}_{k-1}$ of the neural network : Use the input state x_{k-1} and the existing or initial weights of the network in the first iteration;

Step 3.

Calculating the control input u_{k-1} by using

$$u_{k-1} = -\hat{f}_{k-1} + d_k + k_v s_{k-1};$$

Step 4.

Evaluating the estimation error e_k by feeding the tracking error signal s_k into a fixed filter;

Step 5.

Evaluate the squared error of L consecutive training samples: If all of them are less than the criteria for pruning ξ (where we assume it reaches a local minimum), goto step 6; else, go to step 7;

Step 6.

Do the pruning by choosing the weight corresponding to the minimum saliency, then goto step7;

Step 7.

Updating the weights for the output layer

$$\hat{\theta}_k^v = \hat{\theta}_{k-1}^v - \frac{a_k^v (M_k^v)^{-1} e_k^{pT} H(\hat{\theta}_{k-1}^w, x_{k-1}) 2\Delta_k^v}{\rho_k^v} r_k^v \quad (\text{see Eq.(4.28) in}$$

section 3 of this chapter)

and hidden layer

$$\hat{\theta}_k = \hat{\theta}_{k-1} - \frac{a_k (M_k^w)^{-1} e_k^{pT} (\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{c_k^w \rho_k^w} r_k^w \quad (\text{see Eq.(4.58) in sec-}$$

tion 4 of this chapter)

respectively, and the role of ρ_k^v and ρ_k^w is also illustrated later;

Step 8.

Go back to step 2 to continue the iteration.

After summarizing the DTP algorithm, it is necessary to investigate the robustness for this algorithm. Since in Chapter 3, we have derived the conic sector conditions for the learning law in both layers already, which are very similar to those for the second-order SPSA (ASP). So here certain repeated proof procedure will be neglected, and put the emphasis on the robustness analysis for pruning procedure in the following sections.

4.3 Conic Sector Condition for the Robustness Analysis of the Learning Law in the Output Layer

As mentioned above, the conic conditions for the estimation error for both layers are similarly derived as last chapter and the extension to the tracking error is illustrated in Chapter 3, thus we will put the emphasis on the analysis for pruning procedure, which is not analyzed yet and so it is more convenient to analyze the conic conditions for the training and pruning processes separately.

4.3.1 Conic Sector Condition for the Training Process in the Output layer

According to Eq.(4.4) the error signals can be extended as :

$$\begin{aligned}
s_k &= H_2 e_k \\
&= H_2 (f_{k-1} - \hat{f}_{k-1} + \varepsilon_k) \\
&= H_2 (f_{k-1} - \hat{f}_{k-1} + \varepsilon_k) \\
&= H_2 (H(w^*, x_{k-1})v^* - H(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + \varepsilon_k) \\
&= H_2 (H(w^*, x_{k-1})v^* - H(\hat{w}_{k-1}, x_{k-1})v^* + H(\hat{w}_{k-1}, x_{k-1})v^* \\
&\quad - H(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + \varepsilon_k) \\
&= H_2 (H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_{k-1} + \tilde{H}(\hat{w}_{k-1}, x_{k-1})v^* + \varepsilon_k) \\
&= H_2 H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_{k-1} + \tilde{e}_k^v \\
&= -H_2 \Phi_k^v + \tilde{e}_k^v
\end{aligned} \tag{4.17}$$

where the operator $H_2 = 1$ and

$$\Phi_k^v = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_{k-1} \quad (4.18)$$

$$\tilde{e}_k^v = H_2(\tilde{H}(\hat{w}_{k-1}, x_{k-1})v^* + \varepsilon_k) \quad (4.19)$$

$$\tilde{H}(\hat{w}_{k-1}, x_{k-1})v^* = (H(w^*, x_{k-1}) - H(\hat{w}_{k-1}, x_{k-1}))v^* \quad (4.20)$$

$$\hat{f}_{k-1}^{v+} = H(\hat{w}_{k-1}, x_{k-1})(\hat{v}_{k-1} + c_k \Delta_k^v) \quad (4.21)$$

$$\hat{f}_{k-1}^{v-} = H(\hat{w}_{k-1}, x_{k-1})(\hat{v}_{k-1} - c_k \Delta_k^v) \quad (4.22)$$

with Δ_k^v and $r_k^v \in R^{p_v}$, which can be viewed as the first p_v components of Δ_k and r_k defined before.

We define the operator H_1^v , which represents the training process of the output layer, and the loss function as

$$L(\hat{v}_{k-1}, \hat{w}_{k-1}) = \|f_k - \hat{f}_k\|^2 \quad (4.23)$$

and with definitions in equation, we have a gradient approximation using the simultaneous perturbation vector $\Delta_k^v \in R^{p_v}$ to stimulate the weights of the output

layer:

$$\begin{aligned}
\hat{g}(\hat{v}_{k-1}, \hat{w}_{k-1}, \Delta_k^v) &= \frac{L(\hat{v}_{k-1} + c_k^v \Delta_k^v, \hat{w}_{k-1}) - L(\hat{v}_{k-1} - c_k^v \Delta_k^v, \hat{w}_{k-1}) + \varepsilon_k^v}{2c_k^v} r_k^v \\
&= \frac{(f_{k-1} - \hat{f}_{k-1}^{v+})^2 - (f_{k-1} - \hat{f}_{k-1}^{v-})^2 + \varepsilon_k^v}{2c_k^v} r_k^v \\
&= \frac{(f_{k-1} - \hat{f}_{k-1}^{v+} + f_{k-1} - \hat{f}_{k-1}^{v-})(\hat{f}_{k-1}^{v-} - \hat{f}_{k-1}^{v+}) + \varepsilon_k^v}{2c_k^v} r_k^v \\
&= \frac{e_k^T (\hat{f}_{k-1}^{v-} - \hat{f}_{k-1}^{v+})}{c_k^v} r_k^v \\
&= e_k^T H(\hat{w}_{k-1}, x_{k-1}) 2\Delta_k^v r_k^v
\end{aligned} \tag{4.24}$$

Since in the basic form of ASP, $\overline{H}_k$ is actually the sample mean of $\hat{H}_k$ during the period, which is

$$\overline{H}_k = \frac{1}{k+1} \sum_{k=0}^k \hat{H}_k \tag{4.25}$$

and according to the definition of $\hat{H}_k$, we can get

$$\begin{aligned}
\hat{H}_k^v &= \frac{1}{2} \left[\frac{\delta G_k^{vT}}{2c_k^v} r_k^v + \left(\frac{\delta G_k^{vT}}{2c_k^v} r_k^v \right)^T \right] \\
\delta G_{k-1}^v &= \frac{L(\hat{w}_{k-1}, \hat{v}_{k-1} + c_k^v \Delta_k^v + \tilde{c}_k^v \tilde{\Delta}_k^v) - L(\hat{w}_{k-1}, \hat{v}_{k-1} + c_k^v \Delta_k^v - \tilde{c}_k^v \tilde{\Delta}_k^v) + \tilde{e}_k^v}{2\tilde{c}_k^v} \tilde{r}_k^v \\
&+ \frac{L(\hat{w}_{k-1}, \hat{v}_{k-1} - c_k^v \Delta_k^v + \tilde{c}_k^v \tilde{\Delta}_k^v) - L(\hat{w}_{k-1}, \hat{v}_{k-1} - c_k^v \Delta_k^v - \tilde{c}_k^v \tilde{\Delta}_k^v) + \tilde{e}_k^v}{2\tilde{c}_k^v} \tilde{r}_k^v \\
&= g(\hat{w}_{k-1}, \hat{v}_{k-1} + c_k^v \Delta_k^v) - g(\hat{w}_{k-1}, \hat{v}_{k-1} - c_k^v \Delta_k^v) \\
&= e(\hat{v}_{k-1} + c_k^v \Delta_k^v) H(\hat{w}_{k-1}, x_{k-1}) 2\tilde{\Delta}_k^v \tilde{r}_k^v \\
&\quad - e(\hat{v}_{k-1} - c_k^v \Delta_k^v) H(\hat{w}_{k-1}, x_{k-1}) 2\tilde{\Delta}_k^v \tilde{r}_k^v \\
&= (f_{k-1} - \hat{f}_{k-1}^{v+} + \varepsilon_k) H(\hat{w}_{k-1}, x_{k-1}) 2\tilde{\Delta}_k^v \tilde{r}_k^v \\
&\quad - (f_{k-1} - \hat{f}_{k-1}^{v-} + \varepsilon_k) H(\hat{w}_{k-1}, x_{k-1}) 2\tilde{\Delta}_k^v \tilde{r}_k^v \\
&= (\hat{f}_{k-1}^{v-} - \hat{f}_{k-1}^{v+}) H(\hat{w}_{k-1}, x_{k-1}) 2\tilde{\Delta}_k^v \tilde{r}_k^v \\
&= 4H(\hat{w}_{k-1}, x_{k-1}) H^T(\hat{w}_{k-1}, x_{k-1}) c_k^v \Delta_k^v \quad (4.26)
\end{aligned}$$

with $\Delta_k^v, \tilde{\Delta}_k^v, r_k^v, \tilde{r}_k^v \in R^{p_v}$, which can be viewed as the first p_v components of $\Delta_k, \tilde{\Delta}_k, r_k, \tilde{r}_k$, respectively.

And then we can get the $\overline{H}_k^v$

$$\overline{H}_k^v = \frac{1}{k+1} \sum_{k=0}^k \hat{H}_k^v = \frac{1}{k+1} \sum_{k=0}^k (2H(\hat{w}_k, x_k) H^T(\hat{w}_k, x_k)) \quad (4.27)$$

After we define the M_k^v , which is after the mapping of $\overline{H}_k^v$ and consider the pruning,

we have a normalized learning law for the weight of the output layer:

$$\hat{v}_k = \hat{v}_{k-1} - \frac{a_k^v (M_k^v)^{-1} e_k^T H(\hat{w}_{k-1}, x_{k-1}) 2\Delta_k^v}{\rho_k^v} r_k^v \quad (4.28)$$

where ρ is the bounded normalization factor, which is traditionally used in adaptive control systems to bound the signals in the learning algorithm [73], is defined as

$$\rho_k^v = \mu \rho_{k-1}^v + \max\{[(M_k^v)^{-1} H(\hat{w}_{k-1}, x_{k-1})]^2, \bar{\rho}\} \quad (4.29)$$

with $\bar{\rho} > 0, \mu \in (0, 1)$.

Then the stability analysis of the robust neural controller can be justified by the conic sector condition, which requires the feedback system in Figure(4.2) to meet certain dissipative condition as in Theorem 4.1 and can be justified as in the following theorem.

Figure 4.2: The equivalent error feedback systems using the conic sector conditions for the estimate error e_k^T and the output $\Phi_k^v = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k$, where $H_2=1$;

Theorem 4.2: The operator $H_1^v: e_k \rightarrow \Phi_k^v$, which represents the training process

of the output layer(see Figure(4.2)), satisfies the condition (a) and (b) of Theorem 4.1, i.e. $e_k, \Phi_k^v \in L_2$.

Proof: As most of the procedure of this part is the same as the one in last chapter, we will simplify the procedures and put the main results here. So the conic conditions here can be justified as

$$\sum_{k=1}^N \{e_k^T \Phi_k^v + e_k^T e_k \frac{\check{\sigma}^v}{2}\} \geq -(\tilde{v}_0)^2 \left\{ \frac{M_k^v \check{\rho}_k^v}{2a_k p_v} \right\} \quad (4.30)$$

by selecting a suitable normalized factor $\check{\rho}_k^v$ to obtain the constant number $\check{\sigma}^v$ such that

$$1 > \check{\sigma}^v \geq \frac{a_k^v (M_k^v)^{-1}}{p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 (\check{\rho}_k^v)^{-1} \quad (4.31)$$

Note that the conic sector condition guarantees that the estimate error e_k and the output $\Phi_k^v = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k$ are bounded according to the Theorem 4.1, and in turn, the parameter estimate error $\tilde{v}_k$.

4.3.2 Conic Sector Condition for the Pruning Process in the Output Layer

However, from the criterion of pruning, we know that there is pruning once per step at most. So here a new term e_k^p is defined to denote the estimation error instead.

$$e_k^p = H_2(f_{k-1} - \hat{f}_{k-1}^p + \varepsilon_k)$$

where $\hat{f}_{k-1}^p$ is the network output with pruning. And from the definition of this perturbation based pruning, it is obvious that

$$\hat{f}_{k-1}^p = \hat{f}_{k-1}(\hat{v}_{k-1} + \Delta v_{k-1}) \quad (4.32)$$

where the Δv_{k-1} is the added perturbation for pruning. According to the pruning criteria and Eq.(4.16), we know

$$\begin{aligned} \Delta v_{k-1} &= -\frac{v_{k-1,i}}{[\bar{H}_{L_{k-1},i,i}^v]^{-1}} \bar{H}_{L_{k-1}}^v I_{k-1,i} \\ \bar{H}_{L_{k-1}}^v &= \sum_{m=k-L}^{k-1} (\hat{H}_m^v)/L \end{aligned}$$

with i denoting the specific weight to be pruned, $I_{k-1,i}$ is a unit vector whose elements are all zero, except for the i th element, which is equal to unity. $\hat{H}_{k-1}^v$ is the per-iteration estimation Hessian matrix for the output layer. From the mean value theory, we have

$$\hat{f}_{k-1}(\hat{v}_{k-1} + \Delta v_{k-1}) - \hat{f}_{k-1}(\hat{v}_{k-1}) = \Delta v_{k-1} \cdot \hat{f}'_{k-1,v} \quad (4.33)$$

where $\hat{f}'_{k-1,v}$ is the first derivative of the network output with respect to the weights of the output layer. It is easy to see that $\|\hat{f}'_{k-1,v}\| < 1$.

From above analysis, e_k^p can be rewritten as

$$\begin{aligned}
 e_k^p &= H_2(f_{k-1} - \hat{f}_{k-1}^p + \varepsilon_k) \\
 &= H_2(f_{k-1} - (\hat{f}_{k-1} + \Delta v_{k-1} \hat{f}'_{k-1,v}) + \varepsilon_k) \\
 &= -H_2(\Phi_k^v - \Delta v_{k-1} \hat{f}'_{k-1,v}) + \tilde{e}_k^v
 \end{aligned} \tag{4.34}$$

From the basic concept of pruning, we know that some weight is deleted during the pruning. For example, let $\hat{v}_{k-1}^p \in R^n$ denote the weight vector after pruning of $\hat{v}_{k-1}$, and we assume that the n -th weight $\hat{v}_{k,n}$ is pruned, so we have

$$\begin{aligned}
 \hat{v}_{k-1}^p &= \Delta v_{k-1} + \hat{v}_{k-1} \\
 \Delta v_{k-1} &= \hat{v}_{k-1}^p - \hat{v}_{k-1} \\
 &= [\hat{v}_{k-1,1}, \dots, 0] - [\hat{v}_{k-1,1}, \dots, \hat{v}_{k-1,n}] \\
 &= [0, \dots, -\hat{v}_{k-1,n}]
 \end{aligned} \tag{4.35}$$

which shows that the Δv_{k-1} is related to the current weight value, so from Theorem 4.2, $\|\Delta v_{k-1}\|$ is a bounded term.

For the convenience of the analysis later, we can rewrite the e_k^p as a term so that it can describe the estimation error with pruning.

$$e_k^p = -H_2 \Phi_k^{vp} + \tilde{e}_k^v$$

where

$$\Phi_k^{vp} = \Phi_k^v - \tau_k^v \quad (4.36)$$

where $\tau_k^v = \Delta v_{k-1} \hat{f}'_{k-1,v}$ and $\|\tau_k^v\| \leq \tau_{max}^v$ with τ_{max}^v is a positive constant.

And similarly, we have the learning law for the output layer after pruning:

$$\hat{v}_k = \hat{v}_{k-1} - \frac{a_k^v (M_k^v)^{-1} e_k^{pT} H(\hat{w}_{k-1}, x_{k-1}) 2\Delta_k^v}{\rho_k^v} r_k^v \quad (4.37)$$

After reconstruct a new feedback system by using the new input e_k^{pT} and output term $\Phi_k^{vp} = \Phi_k^v - \tau_k^v$, we can justify the conic sector condition for the learning law with pruning for the output layer as in Figure (4.3)

Figure 4.3: The equivalent error feedback systems using the conic sector conditions for the estimate error e_k^{pT} and the output $\Phi_k^{vp} = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k - \tau_k^v$, where $H_2=1$;

$$\begin{aligned}
& \|\tilde{v}_k\|^2 - \|\tilde{v}_{k-1}\|^2 \\
= & -2\{a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^{pv}r_k^{vT}\tilde{v}_{k-1}\}(\rho_k^v)^{-1} \\
& + (a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})^2 \\
= & 2\{a_k^v(M_k^v)^{-1}\}[-e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_{k-1}|\tilde{v}_{k-1}^T(\tilde{v}_{k-1}\tilde{v}_{k-1}^T)^{-1}\Delta_k^v r_k^{vT}\tilde{v}_{k-1}|(\rho_k^v)^{-1} \\
& + (a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})^2] \\
= & 2\{a_k^v(M_k^v)^{-1}\}|e_k^{pT}(\Phi_k^{vp} + \tau_k^v)|\|\tilde{v}_{k-1}^T(\tilde{v}_{k-1}\tilde{v}_{k-1}^T)^{-1}\Delta_k^v r_k^{vT}\tilde{v}_{k-1}|(\rho_k^v)^{-1} \\
& + (a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})^2 \tag{4.38}
\end{aligned}$$

Consider the trace property

$$\begin{aligned}
|\tilde{v}_{k-1}^T(\tilde{v}_{k-1}\tilde{v}_{k-1}^T)^{-1}\Delta_k^v r_k^{vT}\tilde{v}_{k-1}| &= |\text{tr}\{\tilde{v}_{k-1}^T(\tilde{v}_{k-1}\tilde{v}_{k-1}^T)^{-1}\Delta_k^v r_k^{vT}\tilde{v}_{k-1}\}| \\
|\text{tr}\{\tilde{v}_{k-1}\tilde{v}_{k-1}^T(\tilde{v}_{k-1}\tilde{v}_{k-1}^T)^{-1}\Delta_k^v r_k^{vT}\}| &= |\text{tr}\{\Delta_k^v r_k^{vT}\}| \triangleq p_v \tag{4.39}
\end{aligned}$$

Eq.(4.38) can be rewritten as

$$\begin{aligned}
& \|\tilde{v}_k\|^2 - \|\tilde{v}_{k-1}\|^2 \\
= & 2a_k(M_k^v)^{-1}p_v\{e_k^{pT}\Phi_k^{vp}\}(\rho_k^v)^{-1} + 2a_k(M_k^v)^{-1}p_v e_k^{pT}\tau_k^v(\rho_k^v)^{-1} \\
& + \|(a_k^v(M_k^v)^{-1}e_k^T H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})\|^2
\end{aligned}$$

And from $\frac{1}{2}a^2 + 2b^2 \geq 2ab$, we have

$$\begin{aligned}
& \|\tilde{v}_k\|^2 - \|\tilde{v}_{k-1}\|^2 \\
& \leq 2a_k(M_k^v)^{-1}p_v\{e_k^{pT}\Phi_k^{vp}\}(\rho_k^v)^{-1} + a_k(M_k^v)^{-1}p_v\left(\frac{1}{2}\|e_k^{pT}\|^2 + 2\|\tau_k^v\|^2\right)(\rho_k^v)^{-1} \\
& \quad + \|(a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})\|^2 \\
& \leq 2a_k(M_k^v)^{-1}p_v\{e_k^{pT}\Phi_k^{vp}\}(\rho_k^v)^{-1} + a_k(M_k^v)^{-1}p_v\left(\frac{1}{2}\|e_k^{pT}\|^2 + 2\|\tau_{max}^v\|^2\right)(\rho_k^v)^{-1} \\
& \quad + \|(a_k^v(M_k^v)^{-1}e_k^{pT}H(\hat{w}_{k-1}, x_{k-1})\Delta_k^v r_k^{vT}(\rho_k^v)^{-1})\|^2
\end{aligned} \tag{4.40}$$

Summing the above equation up to N steps, we are able to establish the conic condition with a constant $\check{\sigma}^v$ for the estimate error e_k^p as the input in Figure(4.3) and the output Φ_k^{vp} :

$$\sum_{k=1}^N \{e_k^{pT}\Phi_k^{vp} + e_k^{pT}e_k^p\frac{\check{\sigma}^v}{2}\} \geq -[(\tilde{v}_0)^2\{\frac{M_k^v\rho_k^v}{2a_kp_v}\} + N\|\tau_{max}^v\|^2] \tag{4.41}$$

by selecting a suitable normalized factor ρ_k^v to obtain the constant number $\check{\sigma}^v$ such that

$$1 > \check{\sigma}^v > \frac{1}{4} + \frac{a_k^v(M_k^v)^{-1}}{2p_v}\|\Delta_k^v\|^2\|r_k^v\|^2(\rho_k^v)^{-1} \tag{4.42}$$

Note that the conic sector condition guarantees that the estimate error e_k^p and the output $\Phi_k^{vp} = -H(\hat{w}_{k-1}, x_{k-1})\tilde{v}_k - \eta_k^v\tau_k^v$ are bounded according to the Theorem 4.1.

The specified normalized factor ρ_k^v plays two important roles. Firstly, it guarantees $\sigma^v < 1$ to avoid the so-called vanished cone problem [73]; secondly, it guarantees the

sector conditions of Theorem 4.1 to be simultaneously satisfied by both the original feedback system and the normalized equivalent feedback system. Therefore, both conditions (a) and (b) of Theorem 4.1 for e_k^p are fulfilled.

This completes the proof.

4.4 Conic Sector Condition for the Robustness Analysis of the Learning Law in the Hidden Layer

As justified in remark 1, the hidden layer parameter of the network should be estimated separately. Therefore, a conic sector condition will be established for the hidden layer training in this section.

4.4.1 Conic Sector Condition for the Training Process in the Hidden Layer

Similar to the error equation Eq.(4.17), one can rewrite it as

$$\begin{aligned}
e_k &= H_2(f_{k-1} - \hat{f}_{k-1} + \varepsilon_k) \\
&= H_2(H(w^*, x_{k-1})v^* - H(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + \varepsilon_k) \\
&= H_2(H(w^*, x_{k-1})v^* - H(w^*, x_{k-1})\hat{v}_{k-1} + H(\hat{w}^*, x_{k-1})\hat{v}_{k-1} - \\
&\quad H(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + \varepsilon_k) \\
&= H_2(\tilde{H}(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + H(w^*, x_{k-1})\tilde{v}_{k-1} + \varepsilon_k) \\
&= H_2[\tilde{H}(\hat{w}_{k-1}, x_{k-1})\hat{v}_{k-1} + H(w^*, x_{k-1})\tilde{v}_{k-1} + \varepsilon_k] \\
&= -H_2\Phi_k^w + \tilde{e}_k^w \tag{4.43}
\end{aligned}$$

with $\tilde{e}_k^w = H_2(H(w^*, x_{k-1})\tilde{v}_{k-1} + \varepsilon_k)$ and $\Phi_k^w = -\tilde{\Omega}_{k-1}\tilde{w}_{k-1}$. where $\tilde{H}(\hat{w}_{k-1}) = H(w^*, x_{k-1}) - H(\hat{w}_{k-1}, x_{k-1})$, $\tilde{v}_{k-1,i} = v_i^* - \hat{v}_{k-1,i}$, $\tilde{w}_{k-1,i} = w_i^* - \hat{w}_{k-1,i}$, and $\tilde{\Omega}_{k-1} \in R^{m \times p_w}$ is the following matrix

$$\tilde{\Omega}_{k-1} = \begin{bmatrix} \tilde{\mu}_{k-1,1}x_{k-1,1}^T\hat{v}_{k-1,(1,1)}, \dots, \tilde{\mu}_{k-1,1}x_{k-1,n_i}^T\hat{v}_{k-1,(1,1)}, \dots, \tilde{\mu}_{k-1,n_h}x_{k-1,n_i}^T\hat{v}_{k-1,(1,n_h)} \\ \vdots \\ \tilde{\mu}_{k-1,1}x_{k-1,1}^T\hat{v}_{k-1,(m,1)}, \dots, \tilde{\mu}_{k-1,1}x_{k-1,n_i}^T\hat{v}_{k-1,(m,1)}, \dots, \tilde{\mu}_{k-1,n_h}x_{k-1,n_i}^T\hat{v}_{k-1,(m,n_h)} \end{bmatrix}$$

$\in R^{m \times p_w}$ with $\hat{v}_{k-1,i} \in R^{n_h}$, $\hat{v}_{k-1} = [\hat{v}_{k-1,1}^T, \dots, \hat{v}_{k-1,m}^T]^T$ where n_h is the number of neurons in the hidden layer of the network.

Note the above equation is derived from the mean value theorem and the activation function is a non-decreasing function, so there exist unique positive numbers $\tilde{\mu}_{k-1,i}$

$$h_{k-1,i}(w_i^*, x_{k-1}) - h_{k-1,i}(\hat{w}_{k-1,i}, x_{k-1}) = \tilde{\mu}_{k-1,i} x_{k-1}^T (w_i^* - \hat{w}_{k-1,i}) \quad (4.44)$$

where $\hat{w}_{k-1,i}, w_i^* \in R^{n_i}$ are the estimation and ideal weight vectors linked to the i th hidden layer neuron respectively. The maximum value of the derivative $h'_{k-1,i}$ is λ , therefore

$$\lambda \geq \tilde{\mu}_{k-1,i} \geq 0 (1 \leq i \leq n_h) \quad (4.45)$$

After we define

$$\hat{f}_{k-1}^{w+} = H(\hat{w}_{k-1} + c_k^w \Delta_k^w, x_{k-1}) \hat{v}_{k-1} \quad (4.46)$$

$$\hat{f}_{k-1}^{w-} = H(\hat{w}_{k-1} - c_k^w \Delta_k^w, x_{k-1}) \hat{v}_{k-1} \quad (4.47)$$

The gradient approximation of the hidden layer can be written as

$$\begin{aligned}
& \hat{g}(\hat{v}_{k-1}, \hat{w}_{k-1}, \Delta_{k-1}^w) \\
&= \frac{L(\hat{v}_{k-1}, \hat{w}_{k-1} + c_k^w \Delta_k^w) - L(\hat{v}_{k-1}, \hat{w}_{k-1} - c_k^w \Delta_k^w) + \varepsilon_k^w}{2c_k^w} r_k^w \\
&= \frac{(f_{k-1} - \hat{f}_{k-1}^{w+})^2 - (f_{k-1} - \hat{f}_{k-1}^{w-})^2 + \varepsilon_k^w}{2c_k^w} r_k^w \\
&= \frac{(f_{k-1} - \hat{f}_{k-1}^{w+} + f_{k-1} - \hat{f}_{k-1}^{w-})(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+}) + \varepsilon_k^w}{2c_k^w} r_k^w \\
&= \frac{[2(f_{k-1} - \hat{f}_{k-1}) + (\hat{f}_{k-1} - \hat{f}_{k-1}^{w+}) + (\hat{f}_{k-1} - \hat{f}_{k-1}^{w-})]^T (\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+}) + \varepsilon_k^w}{2c_k^w} r_k^w \\
&= \frac{e_k^T (\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{c_k^w} r_k^w \tag{4.48}
\end{aligned}$$

$$= \frac{e_k^T c_k^w \Delta_k^w \hat{\Omega}_{k-1}}{c_k^w} r_k^w \tag{4.49}$$

where $\hat{\Omega}_{k-1}$ is defined similarly to $\tilde{\Omega}_{k-1}$. And similarly, after define

$$\hat{f}_{k-1}^{w++} = H(\hat{w}_{k-1} + c_k^w \Delta_k^w + \tilde{c}_k \tilde{\Delta}_k, x_{k-1}) \hat{v}_{k-1} \tag{4.50}$$

$$\hat{f}_{k-1}^{w+-} = H(\hat{w}_{k-1} + c_k^w \Delta_k^w - \tilde{c}_k \tilde{\Delta}_k, x_{k-1}) \hat{v}_{k-1} \tag{4.51}$$

$$\hat{f}_{k-1}^{w-+} = H(\hat{w}_{k-1} - c_k^w \Delta_k^w + \tilde{c}_k \tilde{\Delta}_k, x_{k-1}) \hat{v}_{k-1} \tag{4.52}$$

$$\hat{f}_{k-1}^{w--} = H(\hat{w}_{k-1} - c_k^w \Delta_k^w - \tilde{c}_k \tilde{\Delta}_k, x_{k-1}) \hat{v}_{k-1} \tag{4.53}$$

we can get

$$\begin{aligned}
G_k^{(1)}(\hat{w}_k + c_k \Delta_k) &= \frac{L(\hat{w}_k + c_k \Delta_k + \tilde{c}_k \tilde{\Delta}_k) - L(\hat{w}_k + c_k \Delta_k - \tilde{c}_k \tilde{\Delta}_k)}{2\tilde{c}_k} \tilde{r}_k \\
&= \frac{L(\hat{w}_k^{w+} + \tilde{c}_k \tilde{\Delta}_k) - L(\hat{w}_k^{w+} - \tilde{c}_k \tilde{\Delta}_k)}{2\tilde{c}_k} \tilde{r}_k \\
&= g(\hat{w}_k^{w+})
\end{aligned} \tag{4.54}$$

where $\hat{w}_k^{w+} = \hat{w}_k + c_k^w \Delta_k^w$. Then we can obtain $\delta G^w(k)$ as follows:

$$\delta G^w(k) = \hat{g}(\hat{w}_k^{w+}) - \hat{g}(\hat{w}_k^{w-}) \tag{4.55}$$

So we can get

$$\begin{aligned}
\delta G^w(k) &= e_k^T (w_k + c_k^w \Delta_k^w) \hat{\Omega}_{k-1}^+ - e_k^T (w_k - c_k^w \Delta_k^w) \hat{\Omega}_{k-1}^- \\
&= (f_{k-1} - \hat{f}_{k-1}^{w+} + \varepsilon_k) \hat{\Omega}_{k-1}^+ - (f_{k-1} - \hat{f}_{k-1}^{w-} + \varepsilon_k) \hat{\Omega}_{k-1}^-
\end{aligned}$$

where $\hat{\Omega}_{k-1}^+$ and $\hat{\Omega}_{k-1}^-$ are also similar to $\tilde{\Omega}_{k-1}$.

And we can take it as a new nonlinear function like $R(\hat{w}_k) = (f_{k-1} - \hat{f}_{k-1} + \varepsilon_k) \hat{\Omega}_{k-1}$,

so we can take the equation above as $R(\hat{w}_k + c_k^w \Delta_k^w) - R(\hat{w}_k - c_k^w \Delta_k^w)$. Using the

mean value theorem again, we can get

$$R(\hat{w}_{k-1} + c_k^w \Delta_k^w) - R(\hat{w}_{k-1} - c_k^w \Delta_k^w) = 2c_k^w \Delta_k^w \bar{\Omega}_{k-1} \quad (4.56)$$

with $\bar{\Omega}_{k-1}$ defined similarly to $\tilde{\Omega}_{k-1}$

Thus the per-iteration estimation of Hessian is calculated as

$$\overline{H}_k^w = \frac{1}{k+1} \sum_{k=0}^k (\bar{\Omega}_k) \quad (4.57)$$

Let M_k^w denote $\overline{\overline{H}}_k^w$ which is after the mapping of $\overline{H}_k^w$, so we can rewrite the learning algorithm after considering pruning for hidden layer as:

$$\hat{w}_k = \hat{w}_{k-1} - \frac{a_k (M_k^w)^{-1} e_k^T (\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{c_k^w \rho_k^w} r_k^w \quad (4.58)$$

Theorem 4.3 : *The operator $H_1^w : e_k \rightarrow \Phi_k^w$, which represents the training algorithm of the hidden layer, and H_2 satisfy the condition(a) and (b) of Theorem 4.1.*

Proof: Taking the same strategy as in last section, we can have the conic conditions as:

$$\sum_{k=1}^N \{e_k \Phi_k^w + \frac{\check{\sigma}^w}{2} e_k^T e_k\} \geq -(\tilde{w}_0)^2 \left(\frac{\lambda_{min}}{\lambda}\right)^2 \frac{M_k^w \rho_k^w}{2a_k^w p_w} \quad (4.59)$$

by selecting a suitable normalized factor ρ_k^w to obtain the constant number $\check{\sigma}^w$ such that

$$1 \geq \check{\sigma}^w \geq \frac{a_k^w \|\Delta_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 (\frac{\lambda_{min}}{\lambda})^2}{p_w \rho_k^w M_k^w}$$

4.4.2 Conic Section Condition for the Pruning Process in the Hidden Layer

Similar to the analysis of the output layer, we can rewrite the estimation error after pruning as:

$$e_k^p = -H_2(\Phi_k^w + \Delta w_{k-1} \hat{f}'_{k-1,w}) + \tilde{e}_k^w \quad (4.60)$$

with $\Delta w_{k-1} = -\frac{w_{k-1,i}}{[\sum_{m=k-L}^{k-1} (\bar{\Omega}_{m,i,i})]^{-1} [\sum_{m=k-L}^{k-1} (\bar{\Omega}_m)]^{-1}} I_{k-1,i}$ where I is defined similar to the one in last section and $\bar{\Omega}_{k-1}$ is the per-iteration estimation of the Hessian matrix of the hidden layer, details of which is discussed next.

Note $\hat{f}'_{k-1,w}$ is the first derivative of the network output with respect to the weight value of the hidden layer and we have proven that the weight of the output layer is a bounded term, so $\|\hat{f}'_{k-1,w}\| \leq \lambda \cdot v_{max}$, where $\|\hat{v}_k\| \leq v_{max}$. And taken the similar analysis of the pruning for output layer, we know that Δw_{k-1} is also a bounded term since it is related to the current weight value of the hidden layer. Thus we can

rewrite the e_k^p like

$$\begin{aligned}
 e_k^p &= -H_2(\Phi_k^w - \Delta w_{k-1} \hat{f}'_{k-1,w}) + \tilde{e}_k^w \\
 &= -H_2(\Phi_k^w - \tau_k^w) + \tilde{e}_k^w \\
 &= -H_2 \Phi_k^{pw} + \tilde{e}_k^w
 \end{aligned} \tag{4.61}$$

where $\tau_k^w = \Delta w_{k-1} \cdot \hat{f}'_{k-1,w}$, and $\|\tau_k^w\| \leq \tau_{max}^w$.

So we have the learning law after pruning as:

$$\hat{w}_k = \hat{w}_{k-1} - \frac{a_k (M_k^w)^{-1} e_k^{pT} (\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{c_k^w \rho_k^w} r_k^w \tag{4.62}$$

After reconstructing a new feedback system by using the input e_k^p and output Φ_k^{pw} ,

we can justify the conic sector condition as follows:

$$\begin{aligned}
& \|\tilde{w}_k\|^2 - \|\tilde{w}_{k-1}\|^2 \\
&= -\left\{2a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T \tilde{w}_{k-1}\right\} + \left\|a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T\right\|^2 \\
&= \sum_{i=1}^{n_h} \left\{-\mu_{k-1,i} e_k^{pT} \hat{v}_{k-1} x_{k-1}^T \tilde{w}_{k-1,i}\right\} (\tilde{w}_{k-1}^T (\tilde{w}_{k-1} \tilde{w}_{k-1}^T)^{-1} \Delta_k^w r_k^{wT} \tilde{w}_{k-1}) \frac{2a_k(M_k^w)^{-1}}{\rho_k^w} \\
&\quad + \left\|a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T\right\|^2 \\
&= \sum_{i=1}^{n_h} \left\{\frac{\mu_{k-1,i}}{h'_{k-1,i}} (-h'_{k-1,i} e_k^{pT} \hat{v}_{k-1} x_{k-1}^T \tilde{w}_{k-1,i})\right\} (\tilde{w}_{k-1}^T (\tilde{w}_{k-1} \tilde{w}_{k-1}^T)^{-1} \Delta_k^w r_k^{wT} \tilde{w}_{k-1}) \frac{2a_k(M_k^w)^{-1}}{\rho_k^w} \\
&\quad + \left\|a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T\right\|^2 \\
&\leq \frac{\lambda}{\lambda_{min}} \left\{ \left| -h'_{k-1,i} e_k^{pT} \hat{v}_{k-1} x_{k-1}^T \tilde{w}_{k-1,i} \right| \left| \tilde{w}_{k-1}^T (\tilde{w}_{k-1} \tilde{w}_{k-1}^T)^{-1} \Delta_k^w r_k^{wT} \tilde{w}_{k-1} \right| \frac{2a_k(M_k^w)^{-1}}{\rho_k^w} \right. \\
&\quad \left. + \left\|a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T\right\|^2 \right\} \\
&\leq 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} \sum_{i=1}^{n_h} \left\{ -\mu_{k-1,i} e_k^{pT} \hat{v}_{k-1} x_{k-1}^T \tilde{w}_{k-1,i} \right\} \\
&\quad + \left\|a_k^w(M_k^w)^{-1} \frac{e_k^{pT}(\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+})}{\rho_k^w c_k^w} (r_k^w)^T\right\|^2 \\
&\leq 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} e_k^{pT} \Phi_k^w + \|e_k\|^2 \|\hat{f}_{k-1}^{w-} - \hat{f}_{k-1}^{w+}\|^2 \|r_k^w\|^2 \left(\frac{a_k^w(M_k^w)^{-1}}{\rho_k^w c_k^w} \right)^2 \\
&= 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} e_k^{pT} \Phi_k^w + \|e_k\|^2 \|\Delta_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 \left(\frac{a_k^w(M_k^w)^{-1}}{\rho_k^w} \right)^2 \\
&= 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} e_k^{pT} (\Phi_k^{pw} + \tau_k^w) + \|e_k\|^2 \|\Delta_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 \left(\frac{a_k^w(M_k^w)^{-1}}{\rho_k^w} \right)^2 \\
&\leq 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} e_k^{pT} \Phi_k^{pw} + a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} \left(\frac{1}{2} \|e_k^p\|^2 + 2\|\tau_k^w\|^2 \right) \\
&\quad + \|e_k\|^2 \|\Delta_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 \left(\frac{a_k^w(M_k^w)^{-1}}{\rho_k^w} \right)^2 \\
&\leq 2a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} e_k^{pT} \Phi_k^{pw} + a_k^w(M_k^w)^{-1} \left(\frac{\lambda}{\lambda_{min}} \right)^2 \frac{p_w}{\rho_k^w} \left(\frac{1}{2} \|e_k^p\|^2 + 2\|\tau_{max}^w\|^2 \right) \\
&\quad + \|e_k\|^2 \|\Delta_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 \left(\frac{a_k^w(M_k^w)^{-1}}{\rho_k^w} \right)^2
\end{aligned}$$

Summing the above equation upon to N steps, we are able to establish the conic condition with a constant $\check{\sigma}^w$ for the estimate error e_k as the input in Theorem 4.1 and the output Φ_k^w similar to the output layer:

$$\sum_{k=1}^N \{e_k^p \Phi_k^{pw} + \frac{\check{\sigma}^w}{2} e_k^{pT} e_k^p\} \geq -[(\tilde{w}_0)^2 (\frac{\lambda_{min}}{\lambda})^2 \frac{M_k^w \rho_k^w}{2a_k^w p_w} + N \|\tau_{max}^w\|^2] \quad (4.63)$$

by selecting a suitable normalized factor ρ_k^w to obtain the constant number $\check{\sigma}^w$ such that

$$1 \geq \check{\sigma}^w \geq \frac{1}{4} + \frac{a_k^w \|\triangle_k^w\|^2 \|r_k^w\|^2 \|\hat{\Omega}_{k-1}\|^2 (\frac{\lambda_{min}}{\lambda})^2}{p_w \rho_k^w M_k^w}$$

Although the conic conditions for the learning and pruning of both layers are obtained between the estimation error e_k^p and the output, the result can be easily extended to the tracking error s_k as following steps. Taking the output layer as an example,

Theorem 4.4 : *If the discrete time signal e_k and s_k in the Figure 4.1 satisfy*

$$(a) \quad e_k \in L_2$$

$$(b) \quad e_k = (1 - k_v z^{-1}) s_k$$

with $\|k_v\| \leq 1$ then the above feedback system is stable with $s_k \in L_2$

Proof : see the proof in Chapter 3 and [70].

4.5 Simulation Results

Followed by the simulation in previous chapter, we consider the same two-link direct drive robot model with a discrete-time input-output form as And then we can get the discrete time dynamic model as follows:

$$\begin{aligned}
 y_{k+1,1} &= y_{k,1} + T * y_{k,2} \\
 y_{k+1,2} &= y_{k,2} - T * M^{-1}(y_{k,1}) * [V(y_{k,1}, y_{k,2}) \\
 &\quad + F(y_{k,1}, y_{k,2})] + T * M^{-1}(y_{k,1}) * Tor_k \\
 &= f(y_{k,1}, y_{k,2}) + U_k + \varepsilon_k
 \end{aligned} \tag{4.64}$$

where $y_k = [y_{k,1}, y_{k,2}]^T$, $y_{k,1}$ and $y_{k,2}$ are the joint angle and velocity vectors, respectively, T is the sampling time, $U_k = T * M^{-1}(y_{k,1}) * Tor_k$ is the torque control vector signal and the same assumption must be made here that the $M^{-1}(y_{k,1})$ is approximately known to compute the U_k , ε_k is a normally distributed disturbance with a bound $\|\varepsilon_k\| \leq 0.2$, and the nonlinear function $f(y_k, y_{k-1}) = y_{k,2} - T * M^{-1}(y_{k,1})[V(y_{k,1}, y_{k,2}) + F(y_{k,1}, y_{k,2})]$ with the configuration dependent inertia matrix

$$M(y_k) = \begin{bmatrix} 3.32 + 0.32\cos(y_{k,2}) & 0.12 + 0.16\cos(y_{k,2}) \\ 0.12 + 0.16\cos(y_{k,2}) & 0.12 \end{bmatrix}$$

Figure 4.4: An overfitting example

centrifugal and coriolis effect

$$V(y_k, y_{k+1}) = \begin{bmatrix} -(y_{k+1,2} - y_{k,2})(2(y_{k+1,1} - y_{k,1}) + y_{k+1,2} - y_{k,2})0.16\sin(y_{k,2})/T^2 \\ 0.16(y_{k+1,1} - y_{k,1})^2\sin(y_{k,2})/T^2 \end{bmatrix}$$

and coulomb friction

$$F(y_k, y_{k+1}) = \begin{bmatrix} 5.3\operatorname{sgn}((y_{k+1,1} - y_{k,1})/T) \\ 1.1\operatorname{sgn}((y_{k+1,2} - y_{k,2})/T) \end{bmatrix}$$

with $T=0.002s$ is the sampling time.

Figure (4.4) shows an overfitting example, where the number of neurons applied in the systems is much greater than required.

First, the feed-forward neural network with 100 hidden neurons is initially used. By the definition of the perturbation, Δ_k is generated as a vector with 100 components satisfying some regularity conditions (e.g., Δ_k being a vector of independent

Bernoulli ± 1 random variables satisfies these conditions). Once the perturbation is decided, the gradient approximation of the output layer can be obtained by Eq.(4.24), and in turn, the Hessian matrix for the output layer using Eq.(4.27). Similarly, Eq.(4.49) and Eq.(4.57) can be applied to update the parameters in the hidden layer through Eq.(4.28) and Eq.(4.58).

Figure 4.6 shows the output of the plant using the standard ASP algorithm without pruning using the command signal $y_k^* = [y_{k1}^*, y_{k2}^*]^T = [\sin(\pi/kT), \cos(\pi/kT)]^T$ (as in Figure 4.5) and Figure 4.7 shows the result by using DTP algorithm, in which the overfitting is not so obvious, and we can evaluate the root mean square(RMS) error for both algorithms, which are illustrated in Figure 4.8 and Figure 4.9.

The theoretical stability analysis has been carried out in this chapter for the proposed algorithm, and here for the specific implementation, some other notes need to be note down. Taking the analysis for the output layer as an example, the conic condition would be satisfied only if the normalization factor is chosen properly, which will play two important roles. First, it guarantees the $0 < \check{\sigma}^v < 1$ to avoid the so called vanishing cone problem. Secondly, it also guarantees the condition (a) for the estimation error e_k^{pv} and output will be satisfied which in turn guarantees the tracking error would be bounded. So in the simulation part, it is critical important to decide this normalization factor for the output layer. According to Equation(4.42)

$$1 > \check{\sigma}^v > \frac{1}{4} + \frac{a_k^v (M_k^v)^{-1}}{2p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 (\rho_k^v)^{-1} \quad (4.65)$$

Figure 4.5: Reference signal

So if

$$\check{\rho}^v(k) > \frac{2a_k^v(M_k^v)^{-1}}{3p_v} \|\Delta_k^v\|^2 \|r_k^v\|^2 \quad (4.66)$$

where $p_v = m * n_h$ with $m = 2$ denoting the dimension of the input and $n_h = 100$ is the number of neurons. Thus, at every step, if we choose the value of the normalization factor as bigger than $\frac{2a_k^v(M_k^v)^{-1}}{3p_v}$, and a good example is $\sqrt{1 + (\frac{2a_k^v(M_k^v)^{-1}}{3p_v})^2}$. The same strategy can be applied to the hidden layer training, thus it will lead to a guaranteed stability of the nonlinear system in the simulation process.

One of the improvement of the weight-pruning based algorithm over the one without any pruning is the decrease of certain computation in the sense after deleting the weights, the update on those weights is not necessary thus leads to a computation saving. For our application, the input are 4 dimensional and output are 2 dimensional, so totally there are $(4+2)*100 = 600$ weights and here is a statistical

Figure 4.6: Output y_k and reference signal using the standard ASP based neural network controller with 100 hidden neurons

Figure 4.7: Output y_k and reference signal using the DTP based neural network controller

Figure 4.8: Root mean square error using the standard ASP based neural network controller with 100 hidden neurons

Figure 4.9: Root mean square error using the DTP based neural network controller

summary of the number of weights after pruning.

	Number of weights initially	Average number of weights after training
without pruning	600	600
with pruning	600	126

From above table, we can see the number of weights has been significantly reduced and thus leads to a less computation on the weights updating.

4.6 Concluding Remarks

In this chapter, the DTP based pruning method for neural controller has been developed to obtain the guaranteed stability with improved generalization ability. A complete stability analysis is performed for this closed loop control system. Simulation results show that the proposed neural controller performs better than a neural controller based on the standard back-propagation algorithm or standard ASP algorithm without pruning in case of overfitting. However, the main disadvantage of the weight-pruning is that it is very expensive in time consumption to remove the weight in a large network. So in the next chapter, we will discuss the neuron-pruning based training algorithm, where the basic idea to overcome the disadvantage is to use the Hessian matrix to remove one unit rather than one weight.

Chapter 5

Neuron-Pruning Based Robust Backpropagation Training Algorithm for RBF Network Tracking Controller

5.1 Introduction

As discussed in Chapter 4, the computation of the second-order algorithm is rather heavy for our system and the Radial Basis Function (RBF) neural network is reported to be computationally more efficient compared with multilayer perceptrons in a number of control applications [91–94], yet better initialization compared to

multilayer perceptron networks [95–100]. However, performance of the RBF network is greatly dependent on the number of radial basis functions. To use a fixed number of RBF units has the drawback in that the number of radial basis functions must be determined a priori, which may be over-sized and lead to poor generalization performance [7, 22, 89]. As surveyed in the first two chapters, several researchers attempted to resolve this problem by determining the number and locations of the radial basis function centers using constructive and pruning methods [29, 35, 36, 54, 101]. Platt [54] proposed a sequential learning algorithm for resource allocating network (RAN). Kadirkamanathan and Niranjan [101] improved RAN by using an extended Kalman filter. However, most of the previous approaches of the RBF pruning/growing algorithms are presented without a stability proof, which is critically important in the neural network control system.

Combining the advantages of RBF networks and Hessian-based pruning algorithm, we propose a robust backpropagation training algorithm for the closed-loop RBF tracking controller using a dead zone scheme to reject noise with a guaranteed stability proof. The idea of using the dead zone had been adopted traditionally in the adaptive and neural control systems for a long time [70, 102–104]. We shall prove that the dead zone assures the convergence and robustness of the tracking controller in the presence of disturbance with emphasis of the pruning caused effects. Instead of the first-order approximation of the activation function, we should use the mean value theory in the convergence proof to explore the limitation and possible improvement for the robustness performance of the pruning based nonlinear training

algorithm in this chapter.

5.2 RBF Tracking Controller and Pruning Based Robust Training Algorithm

5.2.1 Design of the Controller

Consider a class of nonlinear discrete-time system given by

$$\begin{aligned}
 x_1(k+1) &= x_2(k) \\
 &\vdots \\
 x_{n-1}(k+1) &= x_n(k) \\
 x_n(k+1) &= f(x(k)) + u(k) + d(k)
 \end{aligned} \tag{5.1}$$

where $f(x(k)) \in R^m$ is a nonlinear function with $x(k) = [x_1(k) \dots x_n(k)]^T$ and $x_i(k) = [x_i^1 \dots x_i^m]^T \in R^m$; $i = 1, \dots, n$, $u(k) \in R^m$ is the control signal vector, and $d(k) \in R^m$ denotes a disturbance vector with the bound $\|d(k)\| \leq d_m$.

Given a desired trajectory vector $x_{nd}(k) \in R^m$, the tracking error for a robot can be defined as $e_n(k) = x_n(k) - x_{nd}(k) \in R^m$. A filtered tracking error is also used typically in the robot controller as the following:

$$r(k) = e_n(k) + \lambda_1 e_{n-1}(k) + \dots + \lambda_{n-1} e_1(k) \tag{5.2}$$

where $e_{n-1}(k), \dots, e_1(k)$ are the delayed values of the tracking error $e_n(k)$, and $\lambda_{n-1}, \dots, \lambda_1$ are constant matrices selected so that the polynomial $\lambda_{n-1} + \dots + \lambda_1 z^{n-2} + z^{n-1}$ is stable (z represents the time-delayed factor).

Define the control signal $u(k) \in R^m$ as

$$u(k) = x_{nd}(k+1) - \hat{f}(x(k)) + k_v r(k) - \lambda_1 e_n(k) - \dots - \lambda_{n-1} e_2(k) \quad (5.3)$$

where $k_v \in R^{m \times m}$ is a diagonal gain matrix, and $\hat{f}(\bar{x}(k)) \in R^m$ is the output of a three-layered RBF network as

$$\begin{aligned} \hat{f}(\bar{x}(k)) &= [\hat{f}_1(k) \dots \hat{f}_m(k)]^T \\ &= [v_0(k) + v(k)^T H(w(k), \bar{x}(k))] \end{aligned} \quad (5.4)$$

where $v_0(k) \in R^m$ and $v(k) \in R^{n_h \times m}$ are the adjustable weights of the output layer of the RBF network, $w(k) \in R^{n_h \times mn+1}$ is an adjustable center matrix of the hidden layer given by

$$w(k) = [w_1^T(k) \dots w_{n_h}^T(k)]^T \quad (5.5)$$

$w_j(k) \in R^{mn+1}$; is the center vector. n_h is the number of neurons in the hidden layer.

In Eq.(5.4), the bounded input vector $\bar{x}(k) \in R^{mn+1}$ with $\|\bar{x}(k)\| \leq \bar{x}_{max}$ is referred

as the first layer of the network and is given by

$$\begin{aligned}\bar{x}(k) &= [1, \bar{x}_1^T(k), \dots, \bar{x}_n^T(k)]^T \\ \bar{x}_1(k) &= [h(x_i^1(k)) \dots h(x_i^m(k))]^T\end{aligned}\quad (5.6)$$

where $H(w(k), \bar{x}(k)) \in R^{n_h}$ is a nonlinear vector of the hidden layer defined as

$$H(w(k), \bar{x}(k)) = [h(w_1^T(k)\bar{x}(k)) \dots h(w_{n_h}^T(k)\bar{x}(k))]^T \quad (5.7)$$

with a nonlinear radial basis function

$$h(w, x) = \exp\left\{-\frac{(x - w)^T(x - w)}{\sigma^2}\right\} \quad (5.8)$$

where $\sigma > 0$ is the width parameter.

Define the ideal weights and centers $v^* \in R^{n_h \times m}$, $v_0^* \in R^m$ and $w^* \in R^{n_h \times (mn+1)}$, so that $f(x(k))$ can be presented by the RBF network as

$$f(x(k)) = f(\bar{x}(k)) = v_0^* + v^{*T} H(w^*, \bar{x}(k)) + \epsilon(k) \quad (5.9)$$

where $\epsilon(k) \in R^m$ is a reconstruction and measurement error vector with the boundary $\|\epsilon\| \leq \epsilon_m$.

Combining Eq.(5.1)(5.2)(5.3), the closed-loop system becomes:

$$r(k+1) = k_v r(k) + e(k) + d(k) \quad (5.10)$$

with the RBF network estimate error $e(k) \in R^m$ given by

$$\begin{aligned} e(k) &= f(\bar{x}(k)) - \hat{f}(\bar{x}(k)) \\ &= \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k)) + \bar{v}^{*T} \tilde{H}(k) + \epsilon(k) \end{aligned} \quad (5.11)$$

where $\tilde{H}(k) \in R^{n_h+1}$ is defined as

$$\begin{aligned} \tilde{H}(k) &= \bar{H}(w^*, \bar{x}(k)) - \bar{H}(w(k), \bar{x}(k)) \\ &= [1, H(w^*, \bar{x}(k))]^T - [1, H(w(k), \bar{x}(k))]^T \end{aligned} \quad (5.12)$$

$\|\bar{H}(w(k), \bar{x}(k))\| \leq \bar{H}_m$, and $\tilde{v}(k) \in R^{(n_h+1)m}$ is the weight error matrix of the output layer

$$\tilde{v}(k) = [\bar{v}^* - \bar{v}(k)] = [v_0^{*T}, v^*]^T - [v_0^T(k), v(k)]^T \quad (5.13)$$

5.2.2 RBF Unit Pruning

5.2.2.1 Weight Pruning

The basic idea of this approach is to use the information on the second-order derivatives of the error surface in order to make a trade-off between network complexity and training error minimization. A similar idea was originated from Optimal Brain Damage (OBD) procedure [36] or Optimal Brain Surgeon (OBS) procedure [35]. The starting point in the construction of such a model is the approximation of the cost function ξ_{av} using a *Taylor series* about the operating point, described as follows:

$$\begin{aligned}
 & \xi_{av}(W(k) + \Delta W(k)) \\
 = & \xi_{av}(W(k)) + G_k^T(W(k))\Delta W(k) + \frac{1}{2}\Delta W(k)^T H_k(W(k))\Delta W(k) \\
 & + O(\|\Delta W(k)\|^3)
 \end{aligned} \tag{5.14}$$

where $\Delta W(k)$ is a perturbation applied to the operating point $W(k)$, with the $G_k(W(k))$ is the gradient vector and $H_k(W(k))$ is the per-iteration estimated Hessian matrix. The requirement is to identify a set of parameters whose deletion from multilayer perceptron that cause the minimal increase in the value of the cost function ξ_{av} . However, for our dynamic pruning algorithm, the criteria for pruning should be based on a consecutive L -step information(the estimation error of all these L steps are smaller than the criteria, so we regard it reaches a local minimum),

where L is a finite positive integer. So

$$\begin{aligned}
 & \xi_{av}(W(k) + \Delta W(k)) \\
 = & \xi_{av}(W(k)) + \bar{G}_L^T(W(k))\Delta W(k) + \frac{1}{2}\Delta W(k)^T \bar{H}_L(W(k))\Delta W(k) \\
 & + O(\|\Delta W(k)\|^3) \\
 \bar{G}_L^T(W(k)) = & \sum_{k-1+L}^k (G_k^T(W(k)))/L \\
 \bar{H}_L(W(k)) = & \sum_{k-1+L}^k (H_k(W(k)))/L
 \end{aligned} \tag{5.15}$$

To solve this problem in practical terms, the following approximations are made:

- (1) *Quadratic Approximation*: The error surface around a local minimum or global minimum is nearly “quadratic”. Then the higher-order terms in Eq.(5.15) may be neglected.
- (2) *Extremal Approximation*: The parameters have a set of values corresponding to a local minimum or global minimum of the error surface. In such a case, the gradient vector $\bar{G}_k(W(k))$ may be set equal to zero and the term $\bar{G}_k^T(W(k))\Delta W(k)$ on the right-hand of Eq.(5.15) may therefore be ignored.

Under these two assumptions, Eq.(5.15) can be presented approximately as

$$\Delta \xi_{av} = \xi_{av}(W(k) + \Delta W(k)) - \xi_{av}(W(k)) \simeq \frac{1}{2}\Delta W(k)^T \bar{H}_L(W(k))\Delta W(k) \tag{5.16}$$

The goal of OBS is to set one of the synaptic weights to zero to minimize the

incremental increase in ξ_{av} given in Eq.(5.16). Let $W(k, i)$ denote this particular synaptic weight. The elimination of this weight is equivalent to the condition

$$I_i^T \Delta W(k) + W(k, i) = 0 \quad (5.17)$$

where I_i is the *unit vector* whose elements are all zero, except for the i th element, which is equal to unity. And the goal is to minimize the quadratic term $\frac{1}{2} \Delta W(k)^T \bar{H}_L(W(k)) \Delta W(k)$ with respect to the perturbation $\Delta W(k)$, subject to the constraint that $I_i^T \Delta W(k) + W(k, i)$ is zero, and then minimize the result with respect to the index i .

To solve this constraint optimization problem, the following penalty function is used.

$$S = \frac{1}{2} \Delta W(k)^T \bar{H}_L(W(k)) \Delta W(k) - \lambda (I_i^T \Delta W(k) + W(k, i)) \quad (5.18)$$

where λ is the Lagrange multiplier. Apply the derivative of the risk function S with respect to $\Delta W(k)$, use the constraint of Eq.(5.17), and matrix inversion, the optimum solution of the weight vector $W(k)$ is

$$\Delta W(k) = -\frac{W(k, i)}{[H_{i,i}^{-1}]} H^{-1} I_i \quad (5.19)$$

and the corresponding optimum value of the risk function S for element W_i is

$$S_i = \frac{(W(k, i))^2}{2[H_{i,i}^{-1}]} \quad (5.20)$$

where H^{-1} is the inverse of the Hessian matrix $\bar{H}_L$, and $[H_{i,i}^{-1}]$ is the ii -th element of the inverse matrix. The value S_i optimized with respect to $\Delta W(k)$, subject to the constraint that the i th synaptic weights $W(k, i)$ be eliminated, is called the *saliency* of $W(k, i)$.

5.2.2.2 Unit Pruning

The main disadvantage of the weight pruning is that it is very expensive in time consumption to remove the weight in a large network. The basic idea to overcome this disadvantage is to use the Hessian matrix to remove one unit rather than one weight.

One main difference is that the condition $I_i^T \Delta W(k) + W(k, i) = 0$ is replaced by the generalized condition

$$(\Delta W(k) + W(k))^T I_{q_i} = 0$$

with $I = [q_1^T, q_2^T, \dots, q_{n_h}^T]^T$, where I is the selection vector (selecting the outgoing weights of the neuron which will be removed) whose elements are all zero, except for the selected element, which is equal to unity. $q_j \in R^m$ are the indices of the neurons whose outgoing weights will be removed. The key idea for that is: if all outgoing connections of a unit can be removed then the whole unit can be deleted since it cannot influence the net output anymore [105].

Consequently, the saliency of the q_i th neuron can be obtained as

$$S_{q_i} = \frac{1}{2} \bar{v}^T(k) I_{q_i} (I_{q_i}^T \bar{H}_L^{-1} I_{q_i})^{-1} I_{q_i}^T \bar{v}(k) \quad (5.21)$$

In this method, the neuron corresponding to the smallest saliency is the one selected for deletion.

5.2.3 Summary of the Pruning Based Robust Training Algorithm

Pruning Based Robust Training Algorithm for RBF Network Controller

Step 1.

Initializing : Form the new input vector $\bar{x}(k)$ of the RBF network;

Step 2.

Calculating the output $\hat{f}(\bar{x}(k))$ of the neural network : Use the input state $\bar{x}(k)$ and the existing or initial weights of the network in the first iteration;

Step 3.

Evaluating the estimation error e_k by feeding the tracking error signal $r(k)$ into a fixed filter;

Step 5.

Evaluate the squared estimation error of L consecutive training samples: If all of them are less than the criteria for pruning ξ (where we assume it reaches a local minimum), goto step 6; else, go to step 7;

Step 6.

Do the pruning once by choosing the neuron corresponding to the minimum saliency, then goto step 7;

Step 7.

Updating the weights for the output layer and hidden layer by using the robust BP training algorithm respectively;

Step 8.

Go back to step 2 to continue the iteration.

5.2.4 Convergence Properties for the Dynamic Training & Pruning Process of the Output Layer

To update the weight $\bar{v}(k)$, the robust backpropagation algorithm can be written in the matrix form

$$\begin{aligned}\bar{v}(k+1) &= \bar{v}(k) - \alpha(k) \frac{\partial E(k)}{\partial \bar{v}(k)} \\ &= \bar{v}(k) + \alpha(k) \bar{H}(w(k), \bar{x}(k)) e^T(k)\end{aligned}\quad (5.22)$$

where $E(k) = e^T(k)e(k)/2$ is the cost function and $\alpha(k)$ is the learning rate.

From the criteria of pruning, we know that there is pruning once per step at most. So here a new term $e(k)^p$ is defined to denote the estimation error instead of Eq.(5.11).

$$e^p(k) = f(\bar{x}(k)) - \hat{f}^p(\bar{x}(k)) \quad (5.23)$$

where $\hat{f}^p(\bar{x}(k))$ is the RBF network output after pruning one specific neuron. And from the definition of Hessian-based neuron pruning, we know that the neuron-

pruning is actually weights-pruning which is done only at the output layer and

$$\hat{f}^p(\bar{x}(k)) = \hat{f}^p(v^p(k), w(k)) = \hat{f}(v(k) + \Delta v(k), w(k)) \quad (5.24)$$

Note above is from that the effect of pruning is equivalent to adding perturbation to the specific weights according to the definition of pruning, where $v^p(k)$ is the weight matrix after adding the perturbation $\Delta v(k)$ for pruning. And the perturbation is defined through the Hessian-based pruning as

$$\begin{aligned} \Delta v(k) &= -\bar{H}_L(k)^{-1} I_{q_i} (I_{q_i}^T \bar{H}_L(k)^{-1} I_{q_i})^{-1} I_{q_i}^T v(k) \\ \bar{H}_L(k) &= \sum_{m=k-L+1}^k \left[\frac{\partial^2 E(m)}{\partial v^T(m) \partial v(m)} \right] / L \\ &= \sum_{m=k-L+1}^k [\bar{H}^T(w(m), \bar{x}(m)) \bar{H}(w(m), \bar{x}(m))] / L \end{aligned} \quad (5.25)$$

$I = [q_1^T, q_2^T, \dots, q_{n_h}^T]^T$, where I is the selection vector (selecting the outgoing weights of the neuron which will be removed) whose elements are all zero, except for the selected element, which is equal to unity. $q_j \in R^m$ are the indices of the neurons whose outgoing weights will be removed. And $\bar{H}_L(k)$ is the sample mean of the Hessian matrix for the consecutive L steps. And from above equations, we know that the $\Delta v(k)$ is related to the current value of certain specific weights, which is proved to be bounded in last sections. From the mean value theory, we have

$$\hat{f}(v(k) + \Delta v(k), w(k)) - \hat{f}(v(k), w(k)) = \Delta v(k) \cdot \hat{f}'(v(k), w(k))_{v(k)} \quad (5.26)$$

where $\hat{f}'(v(k), w(k))_{v(k)}$ is also a bounded term, which can be justified through the nature of the activation function. Thus we can write as

$$\hat{f}(v(k), w(k)) - \hat{f}(v(k) + \Delta v(k), w(k)) = \tau(v(k)) \quad (5.27)$$

with $\|\tau(v(k))\| \leq \tau_{vmax}$.

From Eq.(5.11), we can rewrite the estimation error after neuron pruning $e^p(k)$ as

$$\begin{aligned} e^p(k) &= f(\bar{x}(k)) - \hat{f}^p(\bar{x}(k)) \\ &= \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k)) + \bar{v}^{*T} \tilde{H}(k) + \epsilon(k) + \tau(v(k)) \end{aligned} \quad (5.28)$$

And after pruning, the updating law Eq.(5.22) for the next step of the output layer should become

$$\bar{v}(k+1) = \bar{v}(k) + \alpha(k) \bar{H}(w(k), \bar{x}(k)) e^{pT}(k) \quad (5.29)$$

Here, we first show the requirement of the use of a variable learning rate $\alpha(k)$ to guarantee convergence of the weight training and pruning algorithm after pruning.

From Eq.(5.28), we have

$$\begin{aligned} \|e^p(k)\|^2 &= e^{pT}(k) \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k)) + e^{pT}(k) \Delta_k \\ &= \text{tr}\{e^{pT}(k) \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k))\} + e^{pT}(k) \Delta_k \end{aligned} \quad (5.30)$$

with the noise term

$$\Delta_k = \bar{v}^{*T} \tilde{H}(k) + \epsilon(k) + \tau(v(k)) \quad (5.31)$$

Consider the Lyapunov Function $\|\tilde{v}(k)\|^2$ and use Eq.(5.29) and (5.30), we have

$$\begin{aligned} \Delta \tilde{v}(k) &= \|\tilde{v}(k+1)\|^2 - \|\tilde{v}(k)\|^2 \\ &= -2\alpha(k) \text{tr}\{\bar{H}(w(k), \bar{x}(k)) e^{pT}(k) \tilde{v}^T(k)\} \\ &\quad + (\alpha(k))^2 \|\bar{H}(w(k), \bar{x}(k)) e^{pT}(k)\|^2 \\ &= -2\alpha(k) \text{tr}\{e^{pT}(k) \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k))\} \\ &\quad + (\alpha(k))^2 \|\bar{H}(w(k), \bar{x}(k)) e^{pT}(k)\|^2 \end{aligned} \quad (5.32)$$

$$\begin{aligned} &= -2\alpha(k) (\|e^p(k)\|^2 - e^{pT}(k) \Delta_k) \\ &\quad + (\alpha(k))^2 \|\bar{H}(w(k), \bar{x}(k)) e^{pT}(k)\|^2 \end{aligned} \quad (5.33)$$

Note that the second equality above is from the property of $\text{tr}\{AB\} = \text{tr}\{BA\}$, if A and B^T are the same dimension. [66].

Without loss of generality, we assume temporarily that the network has a single output to simplify the analysis here. To ensure convergence, the last item in Eq.(5.33) must be smaller or equal to zero, i.e.,

$$e^p(k)^2 \leq \frac{2(|e(k)| - \Delta_{\min})}{\alpha(k) \|\bar{H}(w(k), \bar{x}(k))\|^2} \quad (5.34)$$

or $e^p(k)=0, \forall e(k) \in R^1$, where $\Delta_{\min} \leq \text{sign}(e(k)) \Delta_k$ is a constant. The condition

(5.34) is generally not true for a fixed learning rate $\alpha(k)$ and $\Delta_{min} \neq 0$ (please refer to [103, 104] for theoretical details.)

Motivated from the above analysis, we can propose a variable learning rate based on the dead zone scheme

$$\alpha(k) = \begin{cases} 1 & \text{if } \|e^p(k)\| > \Delta_v \\ 0 & \text{if } \|e^p(k)\| \leq \Delta_v \end{cases}$$

with

$$\Delta_v = \frac{2(v_m \tilde{H}_m + \epsilon_m + \tau_{vmax})}{\sqrt{3 - \bar{H}_m^2}} \quad (5.35)$$

with $0 < \bar{H}_m^2 < \frac{3}{2}$, and v_m , $\tilde{H}_m$, ϵ_m , τ_{vmax} and $\bar{H}_m$ are their maximum value defined above respectively.

Theorem 5.1: *The robust training algorithm in (5.29) using the dead zone scheme has the following properties:*

$$\|\tilde{v}(k+1)\| \leq \|\tilde{v}(k)\| \quad (5.36)$$

$$\lim_{k \rightarrow \infty} \sup |e(k)| \leq \Delta_v \quad (5.37)$$

for $k = 0, 1, \dots$

Proof: Following Eq.(5.33), we have

$$\begin{aligned}
& \Delta \tilde{v}(k) \\
& \leq -\alpha(k)(\|e^p(k)\|^2 - \|e^p(k)\| \|\Delta_k\|) + \alpha(k)\|e^p(k)\|^2 \|\bar{H}(w(k), \bar{x}(k))\|^2 \\
& = 2\alpha(k)\|e^p(k)\| \|\Delta_k\| - \alpha(k) \cdot (2 - \|\bar{H}(w(k), \bar{x}(k))\|^2) \|e^p(k)\|^2 \\
& \leq \alpha(k)(\|e^p(k)\|^2/2 + 2\|\Delta_k\|^2) - \alpha(k) \cdot (2 - \|\bar{H}(w(k), \bar{x}(k))\|^2) \|e^p(k)\|^2 \\
& \leq -\frac{\alpha(k)}{2} [(3 - 2\bar{H}_m^2) \|e^p(k)\|^2 - 4(v_m \tilde{H}_m + \epsilon_m + \tau_{vmax})^2] \tag{5.38}
\end{aligned}$$

Note that the first inequality is from the definition that $\alpha(k)$ is either one or zero, while the second inequality is due to the inequality $2ab \leq 2a^2 + b^2/2$. One interesting thing is that from the first step of above equation, if we take the similar approach as described in Chapter 3 and 4 for the conic condition analysis, we can actually reach the same conic conditions, which we will explore it further in the remark section.

According to the definition above, $\alpha(k) \geq 0$ will be zero as long as the item inside the square bracket of the last item in Eq.(5.38) becomes negative. Therefore, we have

$$\alpha(k)[\|e^p(k)\|^2(3 - 2\bar{H}_m^2) - 4(v_m \tilde{H}_m + \epsilon_m + \tau_{vmax})^2] \geq 0 \tag{5.39}$$

Then $\Delta \tilde{v}(k)$ is bounded below by zero and is not increasing. The limit of $\tilde{v}(k)$

therefore exists and the last item of Eq.(5.38) will go to zero, i.e.,

$$\lim_{k \rightarrow \infty} \sup \|e(k)\| \leq \frac{2(v_m \tilde{H}_m + \epsilon_m + \tau_{vmax})}{\sqrt{3 - 2\tilde{H}_m^2}} = \Delta_v \quad (5.40)$$

This completes the proof.

5.2.5 Convergence Property of the Hidden Layer

To update the center vector $w(k)$, the robust backpropagation algorithm can be written in matrix form

$$\begin{aligned} w(k+1) &= w(k) - \beta(k) \frac{\partial E(k)}{\partial w(k)} \\ &= w(k) + \beta(k) H'(k) v(k) e^p(k) \bar{x}^T(k) \end{aligned} \quad (5.41)$$

with

$$\begin{aligned} H'(k) &= \text{diag}[h'_1(k), \dots, h'_{n_h}(k)] \\ &= \text{diag}[h'(w_1(k) \bar{x}(k)), \dots, h'(w_{n_h}(k) \bar{x}(k))] \end{aligned} \quad (5.42)$$

where $h'_i(k)$ is the first derivative of the activation function in Eq.(5.8).

Theorem 5.2 : *The robust training algorithm in Eq.(5.41) using the dead zone scheme*

$$\beta(k) = \begin{cases} 1 & \text{if } |e^p(k)| > \Delta_w \\ 0 & \text{if } \|e^p(k)\| \leq \Delta_w \end{cases}$$

with

$$\begin{aligned}\Delta_w &= \frac{2\tilde{e}_m\sqrt{h_{\min}}}{\sqrt{3h_{\min} - 2n_h h_{\max}^3 v_{\max}^2 \bar{x}_{\max}^2}} \\ 0 &< \frac{n_h h_{\max}^3 v_{\max}^2 \bar{x}_{\max}^2}{h_{\min}} < \frac{3}{2}\end{aligned}\quad (5.43)$$

where $h_{\min} = \min[\|h'_1(k)\|, \dots, \|h'_{n_h}(k)\|]$ and $h_{\max} = \max[\|h'_1(k)\|, \dots, \|h'_{n_h}(k)\|]$
 $\bar{x}(k) \leq \bar{x}_{\max}$ [refer to definition (5.6)], $\|\tilde{e}(k)\| \leq \tilde{e}_m$ with $\tilde{e}(k) = \tilde{v}^T \bar{H}(w^*, \bar{x}(k)) +$
 $\epsilon(k) \in R^m$, and $\|v(k)\| \leq v_{\max}$.

has the following properties:

$$\|\tilde{w}(k+1)\| \leq \|\tilde{w}(k)\| \quad (5.44)$$

$$\lim_{k \rightarrow \infty} \sup |e^p(k)| \leq \Delta_w \quad (5.45)$$

for $k = 0, 1, \dots$

Proof: Since the proof is almost the same to [70], please see the details in [70].

Corollary 1: If the maximum singular value $k_{v\max}$ of k_v is smaller than one, the tracking error $r(k)$ will converge to a compact set, i.e, $\lim_{k \rightarrow \infty} \|r(k)\| \leq \Delta_r$, where

$$\Delta_r > (\sqrt{(1 - k_{v\max}) + \|k_v\|^2} + \|k_v\|) \frac{\Delta_m + d_m}{1 - k_{v\max}} \quad (5.46)$$

is a positive constant with $\Delta_r = \min[\Delta_v, \Delta_w]$.

Proof: Define the Lyapunov function $\|r(k)\|^2$ and use Eq.(5.10) we have

$$\begin{aligned}
& r^T(k+1)r(k+1) - r^T(k)r(k) \\
= & -r^T(k)[I - k_v T T k_v]r(k) + \|e(k)\|^2 + \|d(k)\|^2 \\
& + 2d^T(k)e(k) + 2r^T(k)k_v e(k) + 2r^T(k)k_v d(k) \\
\leq & -(1 - k_{vmax})\|r(k)\|^2 + \|e(k)\|^2 + \|d_m\|^2 \\
& + 2d_m \triangle_m + 2\|k_v\|\|r(k)\|\|e(k)\| + 2\|k_v\|\|r(k)\|d_m \\
& - [(1 - k_{vmax})(\|r(k)\| - \frac{\|k_v\|(\|e(k)\| + d_m)}{1 - k_{vmax}})^2 \\
& - ((1 - k_{vmax}) + \|k_v\|^2) \frac{\|e(k)\| + d_m}{1 - k_{vmax}}] \tag{5.47}
\end{aligned}$$

If we select $k_{vmax} < 1$, the L.H.S of Eq.(5.47) will be non-increasing, whenever

$$\begin{aligned}
& (1 - k_{vmax})(\|r(k)\| - \frac{\|k_v\|(\|e(k)\| + d_m)}{1 - k_{vmax}})^2 \\
\geq & ((1 - k_{vmax}) + \|k_v\|^2) \frac{(\|e(k)\| + d_m)^2}{1 - k_{vmax}} \\
\Rightarrow & \lim_{k \rightarrow \infty} \sup \|r(k)\| \\
\geq & \lim_{k \rightarrow \infty} \sup (\sqrt{((1 - k_{vmax}) + \|k_v\|^2)} + \|k_v\|) \frac{(\|e(k)\| + d_m)^2}{1 - k_{vmax}} \\
\leq & (\sqrt{((1 - k_{vmax}) + \|k_v\|^2)} + \|k_v\|) \frac{(\triangle_m + d_m)^2}{1 - k_{vmax}} \tag{5.48}
\end{aligned}$$

The conclusion from Eq.(5.47) and (5.48) is that, given any

$$\triangle_r > (\sqrt{((1 - k_{vmax}) + \|k_v\|^2)} + \|k_v\|) \frac{(\triangle_m + d_m)}{1 - k_{vmax}} \tag{5.49}$$

then $\lim_{k \rightarrow \infty} \sup \|r(k)\| \leq \Delta_r$ will be a positively invariant set. This completes the proof.

Remark 5.1: Accuracy of the RBF network tracking controller is affected by three parameters k_{vmax} , Δ_m , and d_m . Because the size Δ_m of the dead zone is independent of the tracking error $r(k)$ as shown in Theorem 5.1 and 2, a smaller tracking error can be achieved by selecting the size Δ_m as small as possible. And pruning will make the size increased, that is natural, since the pruning will cause error increment, however, the pruning itself will improve the generalization performance, which is wildly accepted and proved.

As mentioned in the section 5.2.4, we even can actually reach the conic conditions if we take the similar approach as in Chap 3 and 4. Followed by Eq.(5.32)

$$\begin{aligned} & \|\tilde{v}(k+1)\|^2 - \|\tilde{v}(k)\|^2 \\ = & -2\alpha(k) \text{tr}\{e^{pT}(k) \tilde{v}^T(k) \bar{H}(w(k), \bar{x}(k))\} \\ & + (\alpha(k)^2 \|\bar{H}(w(k), \bar{x}(k)) e^{pT}(k)\|^2 \end{aligned} \quad (5.50)$$

$$(5.51)$$

and similarly as in chapter 3 and 4, $-\frac{\partial e^2(k)}{\partial W} \geq 0 \rightarrow e^T(k) \Phi^v(k)$, with $Phi^v(k)$ term

denotes the remaining factor, we can rewrite it as:

$$\begin{aligned}
& \|\tilde{v}(k+1)\|^2 - \|\tilde{v}(k)\|^2 \\
&= 2\alpha(k)\{e^{pT}(k)\Phi^v(k)\}\rho \\
&\quad + (\alpha(k)^2\|\bar{H}(w(k), \bar{x}(k))e^{pT}(k)\|^2 \\
&\leq 2\alpha(k)\{e^{pT}(k)\Phi^v(k)\}\rho \\
&\quad + \|e^{pT}(k)\|^2(\alpha(k)^2\|\bar{H}(w(k), \bar{x}(k))\|^2
\end{aligned} \tag{5.52}$$

where ρ absorbs other terms.

By summing up to N steps, we can reach the conic conditions as in the similar form as:

$$\{e^{pT}(k)\Phi^v(k)\} + \frac{\|e^{pT}(k)\|^2\sigma}{2} \geq -\eta \tag{5.53}$$

where the σ and η are all positive constants.

From the analysis above, we can see that by either of the stability analysis approaches described in this chapter for the dead zone method, and the way for the conic sector conditions as described in both chapter 3 and 4, we can reach the same destination, which in turn prove that our system is robust and stable.

5.3 Simulation Results

In this part of simulation, we still consider a two-link direct drive robot model with a discrete-time input-output form as

$$\begin{aligned} x_1(k+1) &= x_2(k) \\ x_2(k+1) &= f(x(k)) + u(k) + d(k) \end{aligned} \quad (5.54)$$

where $x(k) = [x_1(k), x_2(k)]^T$

$$f(x(k)) = x_2(k) - T^2 M^{-1}(x_2(k)) [V(x_1(k), x_2(k)) + F(x_1(k), x_2(k))] \quad (5.55)$$

where $x_k = [x_{k1}, x_{k2}]^T$ is the joint angle vector and $u_k \in R^2$ is the torque control signal defined later, d_k is normally distributed disturbance with a bound $\|d_k\| \leq 0.2$, with the configuration dependent inertia matrix

$$M(x_k) = \begin{bmatrix} 3.32 + 0.32\cos(x_{k2}) & 0.12 + 0.16\cos(x_{k2}) \\ 0.12 + 0.16\cos(x_{k2}) & 0.12 \end{bmatrix}$$

centrifugal and coriolis effect

$$V(x_k, x_{k+1}) = \begin{bmatrix} -(x_{k+1,2} - x_{k,2})(2(x_{k+1,1} - x_{k,1}) + x_{k+1,2} - x_{k,2})0.16\sin(x_{k,2})/T^2 \\ 0.16(x_{k+1,1} - x_{k,1})^2\sin(x_{k,2})/T^2 \end{bmatrix}$$

and coulomb friction

$$F(x_k, x_{k+1}) = \begin{bmatrix} 5.3 \operatorname{sgn}((x_{k+1,1} - x_{k,1})/T) \\ 1.1 \operatorname{sgn}((x_{k+1,2} - x_{k,2})/T) \end{bmatrix}$$

with $T=0.002$ s being the sampling time.

The tracking error is defined as

$$r(k) = e_n(k) = x_2(k) - x_{2d}(k) \quad (5.56)$$

where $x_{2d}(k) = [x_{2d,1}(k), x_{2d,2}(k)]^T = [\sin(\pi/kT), \cos(\pi/kT)]^T$ is the reference trajectory signal, which is shown in Figure(5.1). The control signal is defined as

$$u(k) = x_{2d}(k) - \hat{f}(\bar{x})(k) + k_p r(k) \quad (5.57)$$

where $\hat{f}(\bar{x})(k)$ with $\bar{x}(k) = [1, h(x_1(k)), h(x_2(k))]^T$ is the output of an RBF network, with 80 initial hidden neurons to estimate $f(x(k))$.

The simulation is carried out in the following steps. First, $f(x(k))$ is estimated by the standard BP algorithm for 5000 steps with $k_p = \operatorname{diag}[0.5, 0.5]$. In the presence of the disturbance, it was observed that the disturbance caused the divergence of the RBF network as in Figure(5.2) in the sense that there is weight drifting in the hidden layer as shown in Figure(5.3) and (5.4). In contrast, the robust training and pruning algorithm provides the guaranteed convergence of the network in Figure(5.5) and smaller control tracking error shown in Figure(5.6), with decreasing the number of

Figure 5.1: Reference signals

hidden neurons (where L is set to 5), which, in turn, improved the generalization performance. And since the pruning is done only at the output layer, we observed the same weights in the hidden layer, it is shown (Figure(5.7) and (5.8)) that the weights shifting has been greatly reduced. From the implementation perspective, the value of the dead zone for both the output layer and hidden layer is decided by taking the maximum value of each related parameters, which all have an upper bound. i.e. $\varepsilon_m = d_{max} = 0.2, \tilde{H}_m = H_{max} = 1$.

And the next step, we set L to different length, such as 5, 10 and 20. When L is set to 5, the system output is shown in Figure(5.5) with the tracking error in Figure(5.6) and the number of hidden neurons is observed every 500 steps in the simulation and the number of the RBF network is pruned from 80 to about 15, which is shown in Figure(5.9). When L is set to 10 and 20, the result of the system output and number of hidden neurons are shown in Figure(5.10), (5.11), (5.12) and Figure(5.13),

Figure 5.2: System output $x_2(k)$ by using standard BP algorithm.

Figure 5.3: Hidden layer weight $w(1,1)$ by using standard BP algorithm.

Figure 5.4: Hidden layer weight $w(5,1)$ by using standard BP algorithm.

Figure 5.5: System output $x_2(k)$ by using pruning based robust BP training algorithm by setting $L=5$.

Figure 5.6: Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=5$.

Figure 5.7: Hidden layer weight $w(1,1)$ by using pruning based robust BP training algorithm by setting $L=5$.

Figure 5.8: Hidden layer weight $w(5,1)$ by using pruning based robust BP training algorithm by setting $L=5$.

(5.14),(5.15) respectively. From the above results, we can see that with the increase of the L , the output of neuron pruning-based controller is smoother. Moreover, the number of hidden neurons is pruned from 80 to around 20 and there are also more spikes with smaller L .

The absolute value for the instant tracking error(ITE) and the root mean squared error(RMSE) of the BP based training and pruning based training (PBT) with the same $L=10$, can be summarized as following:

	ITE	RMSE
BP	0.18	0.017
PBT	0.07	0.009

According to previous research and mentioned in Chapter 3, we can save certain computation on the neuron or weights updating since some of them are deleted

Figure 5.9: The curve for the neuron pruning by setting $L=5$.

Figure 5.10: System output $x_2(k)$ by using pruning based robust BP training algorithm by setting $L=10$.

Figure 5.11: Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=10$.

Figure 5.12: The curve for the neuron pruning by setting $L=10$.

Figure 5.13: System output $x_2(k)$ by using pruning based robust BP training algorithm by setting $L=20$.

Figure 5.14: Control tracking error of one link $x_{2,1}(k)$ by using pruning based robust BP training algorithm by setting $L=20$.

Figure 5.15: The curve for the neuron pruning by setting $L=20$.

during our pruning process. However, more research will be carried out to compare the overall computation complexity in the future work.

5.4 Concluding Remarks

The proposed RBF network tracking controller is trained and pruned online by the pruning based robust backpropagation training algorithm to give the guaranteed convergence. Both theoretical and simulation results prove that the estimation error converges to a compact set in the presence of disturbance and pruning. On the other hand, after doing the pruning on the networks, some of the weights updating are avoided and thus can reduce certain computation payload. The main benefit of the online pruning is that after the process, we may have a guideline that how many neurons is sufficient to achieve both tracking and generalization performance.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

This thesis provides an in-depth investigation into developing pruning based training algorithm for neural nonlinear tracking control systems. The convergence of the weights for the neural network and the stability of the controls system are analyzed. The conclusion from this study can be summarized below.

The SPSA algorithm is applied to a class of neural nonlinear tracking control system within the framework of a classical adaptive control system. The weight-shifting is removed and thus in turn the tracking control performance is improved. The conic-sector theory is introduced to design the neural tracking control system, which aims

at providing bounded conditions for both the weights of the neural network and the tracking error of the control system. Comparative studies have been carried out among SPSA algorithm, standard back-propagation algorithm and traditional PID controller. The results indicate that better performance is obtained using the SPSA algorithm, .

To improve the generalization ability of the neural system, a weight-pruning based training algorithm is proposed for the nonlinear tracking control system, and followed by a neuron-pruning based training algorithm. Completed convergence analyses have been made for both algorithms and the simulations show that the pruning-based training algorithm performs better in terms of generalization.

Conic sector theory is introduced to the design of the neural tracking control systems in Chapter 3 and 4, which aims to provide guaranteed boundedness for both the input-output signals and the weights of the neural network. One of the main advantages of the conic sector approach is that it provides a model free study. In Chapter 5, a dead zone scheme is integrated with the pruning based training algorithm and the convergence conditions are directly obtained by Lyapunov analysis.

6.2 Recommendations

Possible areas of future work that emerge from this thesis include:

6.2.1 Apply to Real Control Plant

The research presented in this thesis is mainly on the theoretical part and all the results are obtained from simulation. Some related research work on the real application, such as air handling unit [106], has been carried out in our group. We are considering applying our online pruning algorithm to the same system for further evaluation.

6.2.2 Incorporation of Growing Technique

Growing is a natural consideration of the extension to the pruning based training algorithm. Minimal resource allocation network (MRAN) [27, 54] was used in the growing technique for the RBF network, where one neuron will be added after satisfying certain conditions by judging the location of the RBF center. However, it is only applicable for off-line growing which implies the impact of the growing in the online control system stability is not investigated yet. So a direct extension of our pruning methods will apply for the growing neural network in the control system, where the convergence should be guaranteed.

6.2.3 Use of Online Support Vector Regression

As discussed in the last chapter, the RBF network has many advantages over standard multilayer perceptrons. However, as the performance of the RBF networks do depend on the number and position of the centers, both pruning and growing are of

heuristic methods which lack certain insight of the analytical solution.

In the last few years, there has been a surge of interest on support vector machine (SVM) and support vector regression (SVR) [7, 51, 107, 108]. SVM or SVR has shown good generalization ability on a wide variety of problems [7, 20]. From the view point of implementing the neural network, the SVR can be regarded as an algorithm which can decide the optimal number and positions of the neurons in the hidden layer for the RBF network. The main advantage of this approach is that the support vectors, i.e. hidden neurons, are obtained objectively by a constrained optimization for a given error bound.

Although the SVR can resolve the problem with global optimum, it is restricted to be off-line learning only which means that it is not yet suitable for the on line control applications. As for our applications, several compromises can be made to achieve better performance. For example, we can set a sliding window to handle the latest data and apply the SVR to find the least favorable data to delete it and move the window to handle the next data.

In conclusion, this thesis has addressed the problem of on line learning and structure optimization by using pruning based training algorithms and completed stability analysis has been provided for the control system. The improved performance can be summarized as better generalization ability and thus better tracking performance. The detailed results strongly support the proposed methods and algorithms in the simulation parts.

Appendix

Mathematical Preliminaries and Notations

This sections describes the mathematical fundamentals and notations used throughout this thesis. The main concern is with discrete signals which are infinite sequences of real numbers. Each signal may be considered a vector of infinite dimension and represents an element of a set known as a linear vector space.

Norms: Norms may be thought of as a measure of the size of a vector. Let $\mathbf{E}$ be the linear vector space. The zero vector in $\mathbf{E}$ is denoted by $\emptyset$. The function $\rho : \mathbf{E} \rightarrow \mathbf{R}_+$ (the set of positive real numbers) is a norm on $\mathbf{E}$ only if

- (a) $x \in E$ and $x \neq \emptyset$ implies $\rho(x) > 0$
- (b) $\rho(\alpha x) = |\alpha|\rho(x), \forall \alpha \in R, \forall x \in E$
- (c) $\rho(x + y) \leq \rho(x) + \rho(y), \forall x, y \in E$

Given the linear space $\mathbf{E}$ and a norm ρ on $\mathbf{E}$, the pair $(\mathbf{E}, \rho)$ is called a normed vector space.

L_2 -norm: Let $x = (x_1, x_2, \dots)$. The L_2 norm of x is defined as

$$\|x\|_2 = \left\{ \sum_{k=1}^{\infty} x_k^2 \right\}^{1/2}$$

The extension space, denoted by L_{2e} , is the space consisting of those elements x whose truncations lie in the space L_2 , e.g. x belongs to the extended space L_{2e} if

$$\|x\|_{2,T} = \left\{ \sum_{k=1}^T x_k^2 \right\}^{1/2} < \infty$$

$\forall T \in Z_+$ (the set of positive integers).

Frobenius norm: A matrix norm of an $m \times n$ matrix $\mathbf{A}$ defined as the square root of the sum of the absolute squares of its elements(i.e., $\|A\| = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$).

Operator: An operator is a mapping of normed vector spaces.

Passivity: Define the scalar inner product $\langle .|. \rangle$ of two infinite sequences x and y as

$$\langle x|y \rangle = \sum_{k=1}^{\infty} x_k y_k \quad (\text{A.1})$$

An operator $H : x \rightarrow y$ where $x, y \in L_{2e}$ is passive only if there exists some constant β such that

$$\langle y|x \rangle_T = \langle Hx|y \rangle_T \geq \beta \quad (\text{A.2})$$

Conic sector: An operator $H: x \rightarrow y$ where $x, y \in L_{2e}$ is:

(a) inside the cone (C,R) if

$$\langle y - (C - R)x | y - (C + R)x \rangle_T \leq 0, \forall T \in Z_+$$

(b) outside the cone (C,R) if

$$\langle y - (C - R)x | y - (C + R)x \rangle_T \geq 0, \forall T \in Z_+$$

(c) strictly inside the cone (C,R) if for some $\epsilon > 0$

$$\langle y - (C - R)x | y - (C + R)x \rangle_T \leq -\epsilon \|(x, y)\|_T^2, \forall T \in Z_+$$

(d) strictly outside the cone (C,R) if for some $\epsilon > 0$

$$\langle y - (C - R)x | y - (C + R)x \rangle_T \geq \epsilon \|(x, y)\|_T^2, \forall T \in Z_+$$

Dissipativeness: An operator $H : x \rightarrow y$ where $x, y \in L_{2e}$ is weakly (Q, S, R) dissipative only if there exists a constant β such that

$$\langle y|Qy \rangle_T + 2\langle y|Su \rangle_T + \langle u|Ru \rangle_T + \beta \geq 0 \quad (\text{A.3})$$

Figure 1: Extension to conic sector stability theory

$\forall T \in Z_+$ with $\beta = 0$, H is called dissipative.

The following theorem is a necessary extension to the conic sector stability of Safanov [109] for discrete-time adaptive control systems.

Theorem A.1 : Consider the following error feedback system 1

$$\begin{aligned} e(k) &= u(k) - x(k) \\ y(k) &= H_1 e(k) \\ x(k) &= H_2 y(k) \end{aligned} \tag{A.4}$$

with operators $H_1, H_2 : L_{2e} \rightarrow L_{2e}$ and discrete time signal $x(k), y(k), e(k) \in L_{2e}$ and $u(k) \in L_2$. If

$$\begin{aligned} (a) H_1 : e(k) \rightarrow y(k) \text{ satisfies} \\ \sum_{k=1}^N [y(k)^2 + \alpha e(k)y(k) + \beta e(k)^2] \geq -\gamma, \end{aligned} \tag{A.5}$$

$$\begin{aligned} (b) H_2 : y(k) \rightarrow x(k) \text{ satisfies} \\ \sum_{k=1}^N [\beta x(k)^2 - \alpha x(k)y(k) + y(k)^2] \leq -\eta \|(x(k), y(k))\|_N^2 \end{aligned} \tag{A.6}$$

for some $\alpha, \beta, \eta, \gamma$ and $\gamma, \eta > 0$, then the closed loop signals $x(k), y(k) \in L_2$.

Proof: From Eq.(A.4) and using $e(k) = u(k) - x(k)$

$$\sum_{k=1}^N [\beta x(k)^2 - \alpha x(k)y(k) + y(k)^2] + \sum_{k=1}^N [\alpha u(k)y(k) - 2\beta u(k)x(k) + \beta u(k)^2] \geq -\gamma \quad (\text{A.7})$$

Combining Eq.(A.6) and (A.7)

$$-\eta \|(x(k), y(k))\|_N^2 + \sum_{k=1}^N [\alpha u(k)y(k) - 2\beta u(k)x(k) + \beta u(k)^2] \geq -\gamma \quad (\text{A.8})$$

Using the Schwartz inequality

$$\eta \|(x(k), y(k))\|_N^2 - |\alpha| \cdot \|u(k)\| \cdot \|y(k)\|_N - 2|\beta| \cdot \|u(k)\| \cdot \|x(k)\|_N \leq \gamma + |\beta| \cdot \|u(k)\|^2 \quad (\text{A.9})$$

Assume $\|(x(k), y(k))\|_N^2 \rightarrow \infty$ as $N \rightarrow \infty$. Therefore from Eq.(A.9)

$$\eta \leq 0$$

This is a contradiction. Therefore $\|(x(k), y(k))\|_N^2$ is bounded.(i.e. $x(k), y(k) \in L_2$).

A corollary of this theorem is presented which will be useful in the analysis of certain adaptive control system

Corollary A.1: Consider the feedback system of Eq.(A.4). If

$$\begin{aligned} (a)H_1 &: e(k) \rightarrow y(k) \text{ satisfies} \\ &\sum_{k=1}^N [e(k)y(k) + \bar{\sigma}e(k)^2/2] \geq -\gamma, \end{aligned} \quad (\text{A.10})$$

$$\begin{aligned} (b)H_2 &: y(k) \rightarrow x(k) \text{ satisfies} \\ &\sum_{k=1}^N [\bar{\sigma}x(k)^2/2 - x(k)y(k)] \leq -\eta \|(x(k), y(k))\|_N^2 \end{aligned} \quad (\text{A.11})$$

for some $\bar{\sigma}, \gamma, \eta > 0$, then the closed loop signals $x(k), y(k) \in L_2$.

Proof: The proof follows the approach taken for Theorem A.1.

Author's Publications

Conference papers

- (1) Jie Ni, Qing Song and M.J. Grimble, "Robust Pruning of RBF Network for Neural Tracking Control Systems", *45th IEEE Conference on Decision and Control (CDC 2006)*, San Diego, California USA, April 2006.
- (2) Jie Ni and Qing Song, "Sequential Neuron Pruning Algorithm for RBF Network with Guaranteed Stability", *International Joint Conference on Neural Networks (IJCNN'2005)*, Montreal, Canada, July 2005.
- (3) Jie Ni and Qing Song, "Adaptive Simultaneous Perturbation Based Pruning Algorithm for Neural Control Systems", *European Symposium on Artificial Neural Networks (ESANN'2005)*, Bruges, Belgium, April 2005.
- (4) Jie Ni and Qing Song, "Pruning Neural Networks for a Two-Link Robot Control systems", *International Work-Conference on Artificial Neural Networks (IWANN'2005)*, Barcelona, Spain, June 2005.

Journal papers

- (5) Jie Ni and Qing Song, "Dynamic pruning algorithm for multilayer perceptron based neural control systems", In Press, *Neurocomput.*, June 2006.
- (6) Jie Ni and Qing Song, "Pruning Based Robust Backpropagation Training Algorithm for RBF Network Tracking Controller", accepted by *Journal of Intelligent and Robot System*, Springer, 2006.
- (7) Qing Song, J.C.Spall, Y.C.Soh, and Jie Ni, "Robust Neural Network Tracking Controller Using Simultaneous Perturbation Stochastic Approximation", In press, *IEEE Trans. On Neural Networks*, July 2006.
- (8) Jie Ni and Qing Song, "Pruning Radial Basis Function Neural Network for a Class of Nonlinear Control Systems", submitted to *IEEE Trans. On Neural Networks*, Dec. 2005.

Bibliography

- [1] D.S.Bayard, "A forward method for optimal stochastic nonlinear and adaptive control," *IEEE Trans. On Automatic Control*, vol. 36, pp. 1046–1053, 1991.
- [2] M.H.A.Davis and R.B.Vinter, *Stochastic modeling and control*. New York: Chapman and Hall, 1985.
- [3] *Adaptive methods for control system design*. New York: IEEE Reprint Series, 1986.
- [4] *Advances in adaptive control*. New York: IEEE Reprint Series, 1991.
- [5] T.L.Lai and G.Zhu, "Adaptive prediction in non-linear autoregressive models and control systems," *Statistica Sinica*, vol. 1, pp. 309–334, 1991.
- [6] A. Benveniste, M. Metivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*. New York: Springer-Verlag, 1990.
- [7] S. Haykin, *Neural Network –A Comprehensive Foundation*. Prentice Hall, 1999.
- [8] C. M.Bishop, *Neural Network for Pattern Recognition*. Oxford: Clarendon Press, 1996.
- [9] J.Mao and A.K.Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. On Neural Networks*, vol. 6, pp. 296–317, 1995.
- [10] C.Alippi and V.Piuri, "Experimental neural networks for prediction and identification," *IEEE Trans. On Instrumentation and Measurement*, vol. 45, pp. 670–676, 1996.
- [11] K.Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

- [12] F.Girosi and T.Poggio, "Networks and the best approximation property," *Biol. Cybern.*, vol. 63, pp. 169–176, 1990.
- [13] K.S.Narendra and K.Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. On Neural Networks*, vol. 1, pp. 4–27, 1990.
- [14] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural networks for control systems: A survey," *Automatica.*, vol. 28, pp. 1083–1112, 1992.
- [15] H. D.Pati, R.Carelli, and B.R.Kuchen, "Neural networks for advanced control of robot manipulators," *IEEE Trans. On Neural Networks*, vol. 13, pp. 343–354, 2002.
- [16] O. Adetona, E. Garcia, and L. H. Keel, "A new method for the control of discrete nonlinear dynamic systems using neural networks," *IEEE Trans. On Neural Networks*, vol. 11, pp. 102–112, 2000.
- [17] M.S.Ahmed, "Neural-net-based direct adaptive control for a class of nonlinear plants," *IEEE Trans. On Automatic Control*, vol. 45, pp. 119–124, 2000.
- [18] K.I.Funahashi, "On the approximate realization of continuous mapping by neural network," *Neural Nets.*, vol. 2, pp. 183–192, 1989.
- [19] K.Hornik, M.Stinchcombe, and H.White, "Multilayer feedforward network are universal approximators," *Neural Nets*, vol. 2, pp. 359–366, 1989.
- [20] V.Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
- [21] J. Polhill and M. K.Weir, "An approach to guaranteeing generalization in neural networks," *Neural Networks*, vol. 14, pp. 1035–1048, 2001.
- [22] G.N.Karystinos and D.A.Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Trans. On Neural Networks*, vol. 11, pp. 1050–1057, 2000.
- [23] C.Schittenkopf, G.Deco, and W.Brauer, "Two strategies to avoid overfitting in feedforward networks," *Neural Networks*, vol. 10, pp. 505–516, 1997.
- [24] D.E.Rumelhart, G.E.Hinton, and R.J.Williams, "Learning representations by backpropagation erros," *Nature*, vol. 323, pp. 533–536, 1986.

- [25] S.C.Huang and Y.F.Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Trans. On Neural Networks*, vol. 2, pp. 47–55, 1991.
- [26] V.Cherkassky, X.H.Shao, F.M.Mulier, and V.N.Vapnik, "Model complexity control for regression using vc generalizaiton bounds," *IEEE Transactions on Neural Networks*, vol. 10, pp. 1075–1089, 1999.
- [27] L.Yingwei, N.Sundararajan, and P.Saratchandran, "A sequential learning scheme for function approximation by using minimal radial basis function neural networks," *Neural Computation*, vol. 9, pp. 461–478, 1997.
- [28] J.G.Polhill and M.K.Weir, "An approach to guaranteeing generalization in neural networks," *Neural Networks*, vol. 14, pp. 1035–1048, 2001.
- [29] N.B.Karayiannis and G.W.Mi, "Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques," *EEE Trans. On Neural Networks*, vol. 11, pp. 306–314, 2000.
- [30] G.B.Huang, P.Saratchandran, and N.Sundararajan, "A generalized growing and pruning rbf neural network for function approximation," *IEEE Trans. Neural Networks*, vol. 16, pp. 57–67, 2005.
- [31] G.E.Hinton, "Connectionist learning procedures," *Artificial Intell.*, vol. 1, pp. 143–150, 1989.
- [32] G.Castllano, A.M.Fanelli, and M.Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 8, pp. 519–531, 1997.
- [33] C.-S. Leung, K.-W. Wong, P.-F. Sum, and L.-W. Chan, "A pruning method for the recursive least squared algorithm," *Neural Networks*, vol. 14, pp. 147–174, 2001.
- [34] P.V.S.Ponnapalli, K.C.Ho, and M.Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *IEEE Transactions on Neural Networks*, vol. 10, pp. 964–968, 1999.
- [35] B.Hassibi and D.G.Stork, "Second-order derivatives for network pruning : Optimal brain surgeon," *Advances in Neural Information Processing Systems*, pp. 164–171, 1993.

- [36] Y.L.Cun, J.S.Denker, and S.A.Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems 2*, pp. 598–605, 1993.
- [37] J.C.Spall, "Adaptive stochastic approximation by the simultaneous perturbation method," *IEEE Trans.On Automatic Control*, vol. 45, pp. 1839–1853, 2000.
- [38] X.Zhu and J.C.Spall, "A modified second-order spsa optimization algorithm for finite samples," *International Journal of Adaptive Control and Signal Processing*, vol. 16, pp. 397–409, 2002.
- [39] I.Aleksander and H.Morton, *An Introduction to Neural Computing*. London: Chapman and Hall, 1990.
- [40] D.J.Amit, *Modeling Brain Function: The World of Attractor Neural Networks*. New York: Cambridge University Press, 1989.
- [41] J.A.Anderson, *Introduction to Neural Networks*. Cambridge: MA:MIT Press, 1989.
- [42] K.A.Al-mashoug and I.S.Reed, "Including hints in training neural nets," *Neural Computation*, vol. 3, pp. 418–427, 1991.
- [43] S.Amari, "Topographic organization of nerve fields," *Bulletin of Mathematical Biology*, vol. 42, pp. 339–364, 1980.
- [44] —, "Neural theory of association and concept-formation," *Biological Cybernetics*, vol. 26, pp. 175–185, 1977.
- [45] Amari, S.K.Yoshida, and K.I.Kanatani, "A mathematical foundation for statistical neurodynamics," *SIAM Journal of Applied Mathematics*, vol. 33, pp. 95–126, 1977.
- [46] M.J.D.Powell, "The theory of radial basis function approximation," *Advances in Numerical Analysis*, vol. 2, pp. 105–210, 1992.
- [47] W.Light, "Ridge functions, sigmoidal functions and neural networks," *Approximation Theory VII*, pp. 163–206, 1992.
- [48] A.R.Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. On Information Theory*, vol. 39, pp. 930–945, 1993.

- [49] A.N.Tikhonov, "On solving incorrectly posed problems and method of regularization," *Doklady Akademii Nauk USSR*, vol. 151, pp. 501–504, 1963.
- [50] A.N.Tikhonov and V.Y.Arsenin, *Solutions of Ill-posed Problems*. Washington DC: W.H.Winston, 1977.
- [51] B.Schölkopf, C.Burges, and A.Smola, *Advances in Kernel Methods: Support Vector Machines*. Cambridge, MA: MIT Press, 1998.
- [52] R.Reed, "Pruning algorithms—a survey," *IEEE Transactions on Neural Networks*, vol. 4, pp. 740–747, 1993.
- [53] A.S.Weigend, D.E.Rumelhart, and B.A.Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems*, 1991, pp. 875–882.
- [54] J.Platt, "A resource allocating network for function interpolation," *Neural Computing*, vol. 3, pp. 213–225, 1991.
- [55] N. Y.W.L and P.Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, vol. 9, pp. 461–478, 1997.
- [56] D.E.Duckro, D.W.Quinn, and S.J.Gardner, "Neural network pruning with tukey-kramer multiple comparison procedure," *Neural Computation*, vol. 14, pp. 1149–1168, 2002.
- [57] D.C.Chin, "Comparative study of stochastic algorithms for system optimization based on gradient approximations," *IEEE Trans. On Systems, Man, and Cybernetics part B*, vol. 27, pp. 244–249, 1997.
- [58] H.J.Kushner and G.G.Yin, *Stochastic Approximation Algorithms and Applications*. New York: Springer-Verlag, 1997.
- [59] J.C.Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Trans. On Automatic Control*, vol. 37, pp. 332–341, 1992.
- [60] J.C.Spall and J.A.Cristion, "Model free control of nonlinear stochastic system with discrete-time measurements," *IEEE Trans. On Automatic Control*, vol. 43, pp. 1198–1210, 1998.

- [61] —, “A neural network controller for systems with unmodeled dynamics with applications to water plant treatment,” *IEEE Trans. On Systems, Man and Cybernetics, Part B*, vol. 27, pp. 369–375, 1997.
- [62] A.C.Rhodes, K.Johnson, J.LeMoigne, and I.Zavorin, “Multiresolution registration of remote sensing imagery by optimization of mutual information using a stochastic gradient,” *IEEE Trans. On Image Processing*, vol. 12, pp. 1495–1511, 2003.
- [63] Y.F.He, M.C, and S.I.Marcus, “Convergence of simultaneous perturbation stochastic approximation for nondifferentiable optimization,” *IEEE Trans. On Automatic Control*, vol. 48, pp. 1459–1463, 2003.
- [64] Y.Maeda and T.Toshiki, “Fpga implementation of a pulse density neural network with learning ability using simultaneous perturbation,” *IEEE Trans. On Neural Networks*, vol. 14, pp. 688–695, 2003.
- [65] Q.Song, J.C.Spall, and Y.C.Soh, “Robust neural network tracking controller using simultaneous perturbation stochastic approximation,” *Proceedings of the IEEE Conference on Decision and Control*, pp. 6194–6199, 2003.
- [66] M.M.Polycarpou and P.A.Ioannou, “Learning and convergence analysis of neural-type structured networks,” *IEEE Trans. On Neural Networks*, vol. 3, pp. 39–50, 1992.
- [67] F.Sun, Z.Sun, and P.Woo, “Stable neural-network-based adaptive control for sampled-data nonlinear systems,” *IEEE Trans. On Neural Networks*, vol. 9, pp. 956–968, 1998.
- [68] F.L.Lewis, C.T.Abadallah, and D.M.Dwason, *Control of Robot Manipulators*. McMillan, 1993.
- [69] F.C.Chen and H.K.Khalil, “Adaptive control of a class of nonlinear discrete-time systems using neural networks,” *IEEE Trans. On Automatic Control*, vol. 40, pp. 791–801, 1995.
- [70] Q.Song, J.Xiao, and Y.C.Soh, “Robust back-propagation training algorithm for multilayered neural tracking controller,” *IEEE Trans. On Neural Networks*, vol. 10, pp. 1133–1141, 1999.
- [71] S.Jagannathan and F.L.Lewis, “Discrete-time neural net controller for a class of nonlinear dynamic systems,” *IEEE Trans. On Automatic Control*, vol. 41, pp. 1693–1699, 1996.

- [72] Q.Song, R.Katebi, and M.J.Grimble, "Robust multivariable implicit adaptive control," *IMA Journal of Mathematical Control and Information*, vol. 10, pp. 49–70, 1993.
- [73] V.R.Cluet, L.Shah, and G.Fisher, "Robustness analysis of discrete-time adaptive control systems using input-output stability theory: a tutorial," in *IEEE Proceedings, Part-D*, 1988, pp. 133–141.
- [74] J.F.D.Canete, A.Barrero, A.G.Cerezo, and I.G.Moral, "An input-output based stabilization criterion for neural network control of nonlinear systems," *IEEE Trans. On Neural Networks*, vol. 12, pp. 1491–1497, 2001.
- [75] J.Ni and Q.Song, "Dynamic pruning algorithm for multilayer perceptron based neural control systems," *Neurocomputing*, vol. In press, 2006.
- [76] A. LA, "A nonlinear dynamical approach to system identification," *IEEE Circuits System Soc. Newslett.*, vol. 1, pp. 10–23, 2000.
- [77] J.Sjoberg, Q.Zhang, L.Ljung, A.Benveniste, and B.Delyon, "Nonlinear black-box modelling in system identification: a unified overview," *Automatica*, vol. 31, pp. 1691–1724, 1995.
- [78] T.Söderström and P.Stoica, *System Identification*. Prentice Hall International Series In System and Control Engineering, 1989.
- [79] S.A.Billings, "Identification of nonlinear systems—a survey," *IEE Proc.D*, pp. 272–285, 1980.
- [80] S.A.Billings, H.B.Jamaluddin, and S.Chen, "Properties of neural networks with applications to modelling nonlinear dynamical systems," *International Journal of Control*, vol. Vol.55, No.1, pp. 193–224, 1992.
- [81] I.J.Leontaritis and S. A. Billings, "Input-output parametric models for nonlinear systems part i: Deterministic nonlinear system," *Int.J.Contr*, vol. 41, pp. 303–328, 1985.
- [82] Q.Song, W.J.Hu, and T.N.Zhao, "Robust neural controller for variable airflow volume system," in *IEEE Pro.-Control Theory Appli*, 2003, pp. 112–118.
- [83] I.J.Wang and E.K.P.Chong, "A deterministic analysis of stochastic approximation with randomized directions," *IEEE Trans. On Automatic Control*, vol. 43, pp. 1745–1749, 1998.

- [84] R.Ortega, L.Praly, and I.D.Laudau, “Robustness of discrete-time direct adaptive controllers,” *IEEE Trans. On Automatic Control*, pp. 1179–1187, 30.
- [85] S.Jagannathan and F.L.Lewis, “Discrete-time neural net controller for a class of nonlinear dynamic systems,” *IEEE Trans.On Automatic Control*, vol. 4, pp. 1693–1699, 1996.
- [86] M.Zhihong, H.R.Wu, and M.Palaniswami, “An adaptive tracking controller using neural networks for a class of nonlinear systems,” *IEEE Trans.On Neural Networks*, vol. 9, pp. 947–955, 1998.
- [87] E.B.Baum and D.Haussler, “What size gives valide generalization,” *Neural Computation*, vol. 1, pp. 151–160, 1989.
- [88] J.Denker, D.Schwartz, B.Wittner, S.Solla, and R.Howard, “Large automatic learning, rule extraction, and generalization,” *Complex System*, vol. 1, pp. 877–922, 1987.
- [89] Y.L.Cun, “Generalization and network disign strategies,” *Connectionism in Perspective*, pp. 143–155, 1989.
- [90] J. E. Moody, “Note on generalization, regularization and architecture selection in nonlinear learning systems,” in *Proc. 1st IEEE-SP Wkshp. Neural Networks Signal Processing*, 1991, pp. 1–10.
- [91] R. J.W.Park and G.K.Venayagamoorthy, “Comparision of mlp and rbf neural networks using deviation signals for on-line identification of a synchronous generator,” *IEEE Trans. On Neural Networks*, pp. 274–279, 2002.
- [92] M.A.Abido and Y.L.Abdel-Magid, “Online identification of synchronous machines using radial basis function neural network,” *IEEE Trans. On Power Systems*, vol. 12, pp. 1500–1506, 1997.
- [93] S.Elanayar and Y.C.Shin, “Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems,” *IEEE Trans. On Neural Networks*, vol. 5, pp. 594–603, 1994.
- [94] G.Ramakrishna and O.P.Malik, “Rbf identifier and pole-shifting controller for pss application,” in *Proc.1999 IEEE International Electric Machines and Drives Conference*, 1999, pp. 589–591.
- [95] S. Lee and R. Kil, “A gaussian potential function network with hierarchically self-organizing learning,” *Neural Networks*, vol. 4, pp. 207–224, 1991.

- [96] A. G. Bors and M. Gabbouj, "Minimal topology for a radial basis functions neural network for pattern classification," *Digital Signal Processing: A Review Journal*, vol. 4, pp. 173–188, 1994.
- [97] T.P.Chen and H.Chen, "Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks," *IEEE Trans. On Neural Networks*, vol. 6, pp. 904–910, 1995.
- [98] C.-C.Lee, P.-C.Chung, J.-R.Tsai, and C.-I.Chang, "Robust radial basis function neural networks," *IEEE Trans. Syst., Man, Cybern.B*, vol. 29, pp. 674–685, 1999.
- [99] J.B.Gomm and D.L.Yu, "Selecting radial basis function network centers with recursive orthogonal least square training," *IEEE Trans. On Neural Networks*, vol. 11, pp. 306–314, 2000.
- [100] S.Geva and K.Malmstrom, "Local cluster neural net: architecture, training and applications," *Neuralcomput.*, vol. 20, pp. 36–56, 1998.
- [101] V.Kadirkamanathan and M.Niranjan, "Function estimation approach to sequential learning with neural network," *Neural Computing*, vol. 5, pp. 954–975, 1993.
- [102] G.Goodwin and K.S.Sin, *Adaptive Filtering, Prediction and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
- [103] M.Heiss, "Error-minimizing deadzone for basis function networks," *IEEE Trans. On Neural Networks*, vol. 7, pp. 39–50, 1996.
- [104] J.-J.E.Slotine and R.M.Sanner, "Neural networks for adaptive control and recursive identification: A theoretical framework," *Essays on Control: Perspectives in the Theory and its Application*, vol. 11, pp. 381–436, 1993.
- [105] A.Stahlberger and M.Riedmiller, "Fast network pruning and feature extraction by using the unit-obs algorithm," in *Advances in Neural Information Processing Systems*, 1997, pp. 655–661.
- [106] C.y.Guo, Q.Song, and W. Cai, "A neural network assisted cascade control system for air handling unit," *In press, IEEE on Industrial Electronic*, 2006.
- [107] A.Smola, "A tutorial on support vector regression," *NeuroCOLT Tech.Rep.No.NC-TR-98-030*, London: Royal Holloway College, University of London.

-
- [108] C.J.C.Burges, “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, vol. 2, pp. 955–974, 1998.
- [109] M.G.Safanov, *Stability and robustness of multivariable feedback systems*. MIT Press, 1980.