Training algorithm design and weight convergence analysis for discrete-time recurrent neural networks

Xu, Zhao

2013

Xu, Z. (2013). Training algorithm design and weight convergence analysis for discrete-time recurrent neural networks. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/53454>

<https://doi.org/10.32657/10356/53454>

**TRAINING ALGORITHM DESIGN AND
WEIGHT CONVERGENCE ANALYSIS FOR
DISCRETE-TIME RECURRENT NEURAL
NETWORKS**

XU ZHAO

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

2013

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

.....

XU ZHAO

Acknowledgement

I want to take this opportunity to thank those who supported me in any aspect during the course of my Ph.D. studies.

First and foremost, I am heartily thankful to my supervisors, Prof. Song Qing and Prof. Wang Danwei, whose guidance, encouragement and support from the initial to the final level enabled me to develop an understanding of the subject. I admire them for their broad minds, creative thinking and deep insight in intelligent computational area. This thesis would not have been possible without their enthusiastic help.

I must also express my deep gratitude to Dr. Luo Ming (Singapore Institute of Manufacturing Technology). I have learned from her deep understanding and good problem-solving skills with respect to engineering projects.

I must also thank the postgraduates, staff members and visiting scholars at Intelligent Robotics Lab, Nanyang Technological University.

Last but not least, my thanks would go to my loved parents, my sister and my husband. Their unwavering love and support have been the important factors in the realization of this thesis.

Abstract

Recurrent neural networks (RNNs) have become an important study subject in the field of neural networks due to the remarkable developments in both theoretical research and practical applications. RNNs contain feedback loops in the structures which make them much more powerful in dynamical modeling of complex systems as compared with other neural network architectures. This thesis focuses on the design of robust training algorithms for RNNs based on the popular real time recurrent learning (RTRL) concept.

As a starting point, an efficient robust gradient descent training algorithm for multi-input multi-output (MIMO) discrete-time RNNs is proposed which can provide an optimal or suboptimal tradeoff between RNNs training accuracy and weight convergence speed. We design a multivariate robust adaptive gradient-descent (MRAGD) training algorithm for MIMO RNNs. The weight convergence of MRAGD during training is proven in the sense of Lyapunov function. To test the efficiency of the proposed algorithm, RNN based system identifications for both open and close loop conditions are developed. The RNNs are trained by the MRAGD method and the weight convergence conditions are proven.

Secondly, we propose a robust recurrent simultaneous perturbation stochastic approximation (RRSPSA) algorithm under the framework of deterministic system with guaranteed weight convergence. RRSPSA is inspired by the excellent properties of simultaneous perturbation stochastic approximation (SPSA) algorithm which is a well-known recursive procedure for finding roots of equations in the presence of noisy measurements. SPSA has the potential to be significantly more computationally efficient than the usual algorithms of Kiefer-Wolfowitz/Blum type that are based on

standard finite-difference gradient. We show that RRSPSA has the same form as SPSA, and only two objective function measurements are used at each iteration, which maintains the efficiency of SPSA.

Next, we propose a recurrent kernel online learning (RKOL) algorithm which integrates kernel and RTRL learning algorithms. The novel RKOL algorithm achieves guaranteed weight convergence with a sparsification procedure which is explained from system stability point of view to reduce the computational complexity. It can automatically eliminate the respective kernel according to the weight convergence and stability condition.

Finally, in order to further reduce the computational time for RKOL, we propose an improved recurrent kernel online learning algorithm (IRKOL) with a coherence-based sparsification rule to reduce the computational complexity. Furthermore, we present some closed formulas for the sparsification scheme that can be derived from the weight convergence of the RBF-like recurrent network as integral expressions. By focusing on the Gram matrix embedded in the weight convergence proof, we provide explicit formulas suited for the design of the robust recurrent training algorithm.

Symbols and Acronyms

Algebraic Operators

$\text{tr}(A)$	Summation of all the diagonal elements of matrix A
$\text{diag}(A_1, \dots, A_n)$	Block diagonal matrix with main block diagonal matrices
$\ \cdot\ $	L_2 norm for vector or the induced norm for matrix
$\ \cdot\ _\infty$	L_∞ norm for vector or the induced norm for matrix
$\ \cdot\ _F$	Frobenius norm for matrix

Sets

$\mathbb{R}$	Set of real numbers
$\mathbb{C}$	Set of complex numbers

Acronyms

RTRL	- Real Time Recurrent Learning
MISO	- Multi-input Single-output
MIMO	- Multi-input Multi-output
MSE	- Mean Squared Error
SPSA	- Simultaneous Perturbation Stochastic Approximation
RKHS	- Reproducing Kernel Hilbert Space

Contents

Acknowledgements	i
Abstract	iii
Symbols and Acronyms	v
List of Contents	vii
List of Figures	xi
1 Introduction	1
1.1 Motivations and Objectives	1
1.2 Organization and Main Contributions of the Thesis	4
2 Literature Review	7
2.1 Basics of Neural Networks	7
2.1.1 Typical Model of a Neuron	7
2.1.2 Architectures of NNs	8
2.2 RNNs Training Algorithms and Stability Analysis	13
2.2.1 RNN Training Algorithms	13
2.2.2 Stability of RNNs	21
2.3 Generalization of RNNs	25
2.4 Mathematical Preliminaries	26
2.4.1 Normed Spaces and Inner Products	26
2.4.2 Reproducing Kernel Hilbert Space	29

2.4.3	Lyapunov Stability Theorems	30
3	Multivariate RNNs with Robust Adaptive Gradient-Descent Training Algorithm	33
3.1	Basics of the RNN	34
3.2	Output Layer Training and Stability Analysis	37
3.3	Hidden Layer Training and Stability Analysis	43
3.4	Application of MRAGD in Tracking Control and Stability Analysis .	48
3.5	Experimental Verification	53
3.6	Summary	55
4	RNN Training Algorithm Based on SPSA	59
4.1	Basics of the RNN and SPSA	60
4.2	Output Layer Training and Stability Analysis	64
4.3	Hidden Layer Training and Stability Analysis	75
4.4	Experimental Verification	85
4.4.1	Time Series Prediction	85
4.4.2	Michaelis-Menten equation	85
4.4.3	Robot Control	87
4.5	Summary	91
5	Online Prediction of Time Series Data with Recurrent Kernels	93
5.1	The RKOL Algorithm	94
5.2	Weight Convergence and Online Sparicification	98
5.3	Experimental Verification	102
5.3.1	Mackey-Glass Time Sequential Prediction	102
5.3.2	Chaotic Laser Time-series Prediction	104
5.4	Summary	106
6	Recurrent Kernel Online Learning	111
6.1	Introduction	111
6.2	Motivation and the IRKOL algorithm	112

6.2.1	The Existing ALD and Coherence Sparsification Schemes . . .	112
6.2.2	The IRKOL Algorithm	113
6.3	Weight Convergence and Sparsification - an Integrated Theoretical Analysis	116
6.4	Simulations	121
6.4.1	Mackey-Glass time-sequential prediction	122
6.4.2	Chaotic laser time-series prediction	122
6.5	Summary	127
7	Conclusions and Future Work	129
7.1	Conclusions	129
7.2	Future Work	131
	Author's Publications	135
	Bibliography	137

List of Figures

2.1	Structure of a single neuron.	8
2.2	Architecture of the single layer network.	9
2.3	Architecture of the multi-layer network.	10
2.4	Architecture of the RBF network.	10
2.5	Architecture of a Jordan network.	11
2.6	Architecture of an Elman network.	12
3.1	Structure of control scheme.	50
3.2	Output $y^d(k)$ and reference signal $y(k)$ using RTRL and MRAGD and BP based neural controller.	56
3.3	Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ (initial re- sponse in Figure 3.2) from the step 1 to 200 using RTRL and MRAGD and BP training algorithms.	57
3.4	Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ from the step 200 to 2000 using RTRL and MRAGD and BP training algorithms. . .	57
3.5	Full traces of normalization factors $\rho^v(k)$ and $\rho^w(k)$	58
4.1	Block diagram of RRSPSA training for the output feedback based RNN.	62
4.2	Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RRSPSA training algorithms for time series prediction. . .	86

4.3	Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RRSPSA training algorithms for simulation of Michaelis-Menten equation.	88
4.4	Output $y^d(k)$ and reference signal $y(k)$ using RRSPSA and MRAGD based neural controller.	89
4.5	Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ (initial response in Figure 3.2) from the step 1 to 200 using RRSPSA and MRAGD training algorithms.	90
4.6	Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ from the step 200 to 2000 using RRSPSA and MRAGD training algorithms.	90
5.1	Comparison of desired and predicted testing data of Mackey-Glass time-series by NORMA, KNLMS and RKOL.	105
5.2	Learning curve for RKOL on testing data of Mackey-Glass time-series.	106
5.3	Full trace of L_2 norms of weight for Mackey-Glass time-series.	107
5.4	Full trace of normalization factor ρ_t	107
5.5	Comparison of desired and predicted testing data of laser time-series by NORMA, KNLMS and RKOL.	108
5.6	Learning curve for RKOL on testing data of laser time-series.	109
5.7	Learning curve for RKOL on testing data of laser time-series.	109
5.8	Full trace of normalization factor ρ_t for laser time-series.	110
6.1	A comparison of the desired and predicted testing data for the Mackey-Glass time series trained using NORMA, KNLMS and IRKOL.	123
6.2	The learning curve for KNMLS, NORMA, RKOL and IRKOL applied to the testing data of the Mackey-Glass time series.	124
6.3	The full trace of the L_2 norms of the weights for the Mackey-Glass time series.	125
6.4	The full trace of the normalization factor ρ_t	125
6.5	A comparison of the desired and predicted testing data for the laser time series trained using NORMA, KNLMS and IRKOL.	126

6.6	The learning curve for IRKOL applied to the testing data of the laser time series.	127
6.7	The full trace of the L_2 norms of the weights for the laser time series.	128
6.8	The full trace of the normalization factor ρ_t for the laser time series.	128

Chapter 1

Introduction

1.1 Motivations and Objectives

The study of robust training algorithm and weight convergence in recurrent neural networks (RNNs) has remarkable development in both theoretical research and practical applications. RNNs contain feedback loops in the network structures which make them more powerful in dynamical modeling of complex systems compared with other neural network structures. The approximation capability also renders RNNs as highly competitive candidate for learning of input-output dynamic models. RNNs have already been applied to a large number of engineering problems involving nonlinear system identification, time-series prediction and control applications [1–9]. Robust training algorithm and weight convergence for online learning are critical issues that make RNNs work for practical applications. So far the gradient-based training algorithm for RNNs is one of the most well developed and widely used methods due to the ease of implementation and high adaptability. Most importantly, it can be utilized in case of training data set which has non-stationary statistics. In the robust training algorithm considerations of RNNs, we are interested in whether a particular training algorithm can achieve the system stability, weight convergence and computational efficiency under the presence of disturbances. Essentially, the study of weight convergence and system stability of a robust training algorithm is to derive conditions under which the weight of RNNs will be convergent. Simple

and relaxed convergent conditions that can be easily verified are favorable for practical use. However, due to heavy computational complexity of RNNs, the weight convergence and stability properties of RNNs learning algorithms are difficult to obtain. In many cases, RNN training is designed and tuned by empirical method without any theoretical guidance. Conventional training algorithm of RNNs such as backpropagation through time (BPTT) [10, 11] is an offline learning algorithm which calculates the exact model derivatives by pursuing all steps back into the past. Real time recurrent learning (RTRL) [12, 13] algorithm is an online training algorithm which updates the synaptic weights of a fully connected RNN in real time. It is more efficient than BPTT since it doesn't unfold the model into the past, but instead of assuming that the model parameters do not change during a specific time interval. However, the RTRL is not guaranteed to follow the negative gradient of the error function with respect to the weight matrices. The approximation of the true error gradient by RTRL may result in instability in the system. This effect can be avoided by using small learning rate. Furthermore, realistic processes usually suffer from external disturbance, and there may also imperfectly known inputs. In such cases, robustness has to be taken into account in the design of the RNN training algorithms. In literature, most of robust stability studies of RNNs have focused on the case of fixed weights with zero or constant input with the objective to show either the origin is globally absolutely stable, or to provide the respective local conditions for multi-stable points with many equilibria [14–32]. In contrast, the robustness of RNNs training is far less discussed compared to the former topic. Therefore, it is desirable to develop robust learning algorithms to obtain a tradeoff between the stability and fast convergence speed. We acknowledge that neural engineers have usually studied only the synaptic dynamic systems with weight updating or the neural dynamic systems with neuron state changing independently, which are equally important [33]. However, since the celebrated back-propagation training algorithm has been successfully applied to multi-layered RNNs like the Jordan and Elman networks taxonomy boundaries between studies of the two dynamic systems become somehow fuzzy [33].

On the other hand, when RNNs are trained with the training data, an important issue is how well it generalizes to patterns other than the training set. If the RNN simply memorizes the training patterns, it may perform well during training but may fail miserably when presented with different testing data. Therefore, the goal of network training is not only to learn an exact representation of the training patterns, but also to build a model of the process. In particular for dynamic systems with the recurrent data modeling, theoretical studies [34,35] have 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 [36] and experiments [37,38] show that networks with fewer free parameters exhibit a better generalization performance. Lastly, from an implementation point of view, smaller networks requires fewer resources in any physical computational environment. To improve the generalization ability of RNNs, pruning or growing is an alternative way which can be incorporated into the training algorithms to optimize the size of RNNs.

Motivated by these new trends and challenges of RNNs, this thesis mainly focuses on the robust training algorithm design of RNNs with the emphasis on the fast and stable learning ability based on RTRL architecture, which is particularly suitable for online training of the RNNs with the reduced computational complexity for RNNs training. This thesis also follows the previous research in [39] and [5] to extend the study of synaptic dynamic system into the RNNs training algorithm with the crucial modifications of adaptive parameters, in particular, the novel recurrent constrained parameter matrix. This study guarantees the weight convergence of RNNs in the sense of Lyapunov function and supremum of the estimate error. It should be pointed out that our study does not deal with the neural dynamic systems because the speed of weight convergence is usually much slower than that of neurons.

1.2 Organization and Main Contributions of the Thesis

This thesis consists of seven chapters. The whole thesis investigates the existing results in the literature and also provides answers to several open questions in this area. Specifically, the contents and contributions of each chapter are summarized as follows.

In Chapter 1, the motivation, objectives, organization and contributions of the thesis are provided.

In Chapter 2, a literature review for RNNs, the recent developments on the learning algorithms and stability analysis of RNNs are given.

In Chapter 3, the robust adaptive gradient-descent (RAGD) training algorithm [39] designed for multiple-input-single-output (MISO) RNNs is extended into multi-variate robust adaptive gradient-descent (MRAGD) training algorithm for MIMO RNNs. RNNs are highly nonlinear dynamic system, the weight convergence and stability analysis of MIMO RNNs is much more complex than that of MISO RNNs. The weight convergence and system stability of MRAGD are proven in the sense of Lyapunov function. To test the efficiency of the designed algorithm, an RNN based robust tracking controller is designed which is trained by MRAGD method and the closed-loop stability is proven. Simulation results are presented for a two-link robot tracking control problem. Note that the weight convergence study in RNNs results in automatically, in general, the L_2 stability as shown in Theorems 3.1 and 3.2 and we would explicitly explore this particular point in the following chapters to avoid duplicated theoretical analysis.

In Chapter 4, a robust recurrent simultaneous perturbation stochastic approximation (RRSPSA) algorithm is proposed. RRSPSA is inspired by the excellent properties of simultaneous perturbation stochastic approximation (SPSA) [40] algorithm which is a well-known recursive procedure for finding roots of equations in the presence of noisy measurements. SPSA is more economical in terms of loss function evaluation which is usually the most expensive part of an optimization problem.

We will show that RRSPSA has the same form as SPSA, and only two objective function measurements are used at each iteration, which maintains the efficiency of SPSA. The weight convergence proof and stability analysis are quite different from those in [40] under the framework of deterministic system. In addition to the simplicity and efficiency of RRSPSA, one powerful aspect of the developed RRSPSA algorithm is that it is able to use effectively large adaptive learning rate to accelerate weight convergence speed with guaranteed system stability. The recurrent connections are also automatically controlled based on the stability condition.

In Chapter 5, an RKOL algorithm is proposed for RNNs. The main contribution in this chapter is the establishment of an inherent link between the popular online kernel and recurrent learning via implementation of the proposed recurrent kernel online learning. The key idea of RKOL is to treat the kernel online training algorithm as a recurrent network, in which the output depends not only on the current external input to the network, but also on the current or previous outputs. A sparsification procedure is developed from the system stability point of view. Experimental verification is provided to show the effectiveness of the algorithm.

In Chapter 6, an IRKOL algorithm is proposed. The merit of this work is that it shows very good performances with respect to root mean square error (RMSE), computational time and number of neurons while weight convergence and system stability can be guaranteed. An adaptive recurrent hyper-parameter is used to control the recurrent feedback contribution for kernel function of the IRKOL algorithm in addition to classical adaptive learning rate and dead zone concepts. Furthermore, the hyper-parameter is defined in such a way that it is not only ensure the weight convergence but also provides an adaptive disk to link the weight convergence condition to the kernel sparsification conditions. This can automatically eliminate or stop growing of the respective kernel function of the recurrent learning to minimize the system structural risk in an online learning fashion. Experimental verification is provided to show the effectiveness of the algorithm.

In Chapter 7, conclusions and future work are addressed.

Chapter 2

Literature Review

2.1 Basics of Neural Networks

Neural networks (NNs) have been extensively studied with the aim of achieving human-like performance. They are systems that emulate the human brain in certain ways such as that they need to learn how to behave and they must be taught or trained based on a former knowledge of the environment. NNs are composed of a large number of highly interconnected processing elements which are called neurons working in parallel, which are inspired by the way biological nervous systems, such as the brain, process information to solve a specific problem. This section therefore focuses on the established results on NNs.

2.1.1 Typical Model of a Neuron

Many neuron models have been proposed and one of the widely used structures is shown in Figure 2.1, which can be described by the equation

$$y(k) = \Phi\left(\sum_{i=1}^n w_i(k)x_i(k) + b(k)\right) \quad (2.1)$$

where $x_i(k)$, $y(k)$ and $b(k)$ are neuron's external input, output and bias respectively, k is the number of input nodes, and $w_i(k)$ is the synaptic weight of neuron, and

Figure 2.1: Structure of a single neuron.

$\Phi(\cdot)$ denotes the activation function that defines the output of a neuron. There are several types of $\Phi(\cdot)$ that are frequently used in literature, such as threshold function, piecewise-linear function ect. In our work, we employ the sigmoidal function with the form

$$\Phi(x) = \frac{1}{1 + \exp(-\lambda x)} \quad (2.2)$$

in which λ is a positive number that represents the slop steepness of the transient range of the activation function.

The following properties of NNs make them great success in many applications: they are nonlinear systems; they learn by experience which is a ‘human-like’ learning behavior; they have the ability to generalize; they are parallel distributed architecture, allowing fast processing for large scale dynamical systems; they are robust in the presence of noise; they can realize multivariate systems. NNs have been applied in computer vision, signal/image processing, speech/character recognition, expert systems, medical image analysis, remote sensing, robotic processing, industrial inspection, and scientific exploration [41].

2.1.2 Architectures of NNs

Three fundamental neuron network architectures are presented in this section. In layered NNs, the neurons are organized in the form of layers. There are input layer,

Figure 2.2: Architecture of the single layer network.

hidden layers, and output layer. And the neurons in each layer are called input nodes, hidden nodes, and output nodes, respectively. Usually, the input layer is not counted because no computation occurs in this layer.

Feedforward Neural Networks (FNNs)

- Single-Layer FNNs

Single layer feedforward network (SLFN) has no hidden layer and the signal transmission is fixed in one direction. An illustration for the network with l input nodes and m output nodes is shown in Figure 2.2. n is time index.

- Multi-Layer Feedforward Networks

Multilayer feedforward network which is commonly referred to as Multilayer Perceptrons Neural Network (MLP) is an important type of feedforward neural networks (FNNs) which has at least one hidden layer. The presence of hidden layers in MLP aims to provide a more complicated network structure, which can manage higher-order statistics. An illustration for the MLP is shown in Figure 2.3. It is commonly used for learning the input-output mapping of an underlying system from the input-output data alone. MLP is good and simple to use when the underlying input-output mapping is static and the training data set is sufficiently large and representative of the normal operation of the

Figure 2.3: Architecture of the multi-layer network.

Figure 2.4: Architecture of the RBF network.

system under study [42, 43].

- Radial Basis Function (RBF) NNs

The RBF NN has a feedforward structure with three layers which play entirely different roles. Fig. 2.4 is the plot of a RBF neural network. The input layer connects the network to its environment. Note that the weights between input layer neurons and hidden layer neurons are all set to 1. The only hidden layer projects the input space to a high dimensional hidden space. The output layer is linear which supplies the response of the network to the activation signal applied to the input layer.

Figure 2.5: Architecture of a Jordan network.

Recurrent Neural Networks (RNNs)

RNNs distinguish themselves from FNNs in that they have at least one feedback loop in their structures. The architectures are inherently dynamic and the training of RNNs involves the recurrent feedback signal for weight updating. Therefore, RNNs with recurrent training law are more powerful to deal with dynamic systems as compared to FNNs.

Basically, there are two fundamental uses of RNNs [33]:

- Associative memories
- Input-output mapping networks

The best-known RNN useful as associative memories is the Hopfield network which consists of a set of neurons and a corresponding set of unit delays, forming a multiple-loop feedback system. It has fixed symmetric synaptic connections and no hidden neurons in its structure. Hopfield publishes two fundamental papers on Hopfield network: the discrete model [44] and the continuous one [45]. The design of Hopfield networks mainly focuses on problems of existence of a fixed point, the conditions under which it is unique, the maximum number of fixed points that can be achieved in a given network and the conditions under which the equilibrium states of the RNNs are stable [14–32].

Figure 2.6: Architecture of an Elman network.

The other type of RNNs that is widely used as input-output mapping is with free synaptic weights, the operation of which relies on the availability of hidden neurons. It is much more powerful than Hopfield network on the dynamic representation. This type of RNNs has been found in many applications such as problems dealing with control systems, robotics and signal processing applications [1, 39, 46–54].

There have been many RNN architectures with free synaptic weights proposed by various groups. Jordan network [55] is a type of RNNs that the past output values of the network are fed back to the context nodes of the input layer. This structure pays more attention to the sequence of output values. Figure 2.5 is an example of a Jordan network architecture. The Elman network [56] is another type of RNNs in which the past values of the hidden units are fed back into themselves or to the context portion of the input layer. Figure 2.6 is an example of an Elman network architecture. This structure pays more attention to the sequence of input values. Modifications to Elman network are such as in [57], where self-connections are introduced in the context units. The Williams-Zipser network [12] can represent rich information of past output which it allows any neuron to be connected to any other neuron in the network. It is the most general type of network.

2.2 RNNs Training Algorithms and Stability Analysis

The most fundamental characteristic that enables an RNN to serve as a powerful computational device is its learning ability. RNNs possessing the same structure can exhibit different dynamic behavior, due to the use of distinct learning algorithms. The added complexity of learning in recurrent networks has given rise to a variety of techniques and associated research projects. A goal is to design efficient algorithms by adjusting the weight parameters to implement certain functions or make the RNN output best fit into certain curve with the given data, and meanwhile the cost function of the modeling error is minimized. However, feedback is like a two-edged sword, in that when it is applied improperly it can produce instability. Therefore, stability must also be considered. In this section, an overview of the training algorithms and their stability analysis for RNNs is presented.

2.2.1 RNN Training Algorithms

There are two general concepts of recurrent structures training: fixed point learning and nonfixed point learning. Fixed point learning is aimed at making the network reach the prescribed stable fixed points or perform steady-state matching. The only requirement on the transients is that they die out. There are several fixed point training algorithms [58–60]. The fixed point learning is unable to learn nonfixed point attractors or to produce desired temporal behavior over a bounded interval, or even to learn to reach their fixed point quickly [61]. Hence, nonfixed point learning is of major concern.

There are basically two modes of training: batch mode and sequential mode. Batch learning is a common strategy for offline training. The idea is to adapt the weights once the whole training set has been presented to an adaptive system. On the other hand, sequential mode is for online training and the weights are adjusted after the presentation of each pattern in the training set.

The algorithm that only uses the first order derivative is called gradient-descent method which is the most widespread supervised learning mechanism for NNs. Many specific training algorithms are tailored by gradient-descent method which also provide estimates of the convergence behavior under these algorithms. A popular manifestation of gradient descent is backpropagation (BP) introduced by Rumelhart [34]. Gradient methods, and in particular backpropagation learning have been extended to train RNNs. Various attempts to extend backpropagation learning to recurrent networks is summarized in Pearlmutter [62]. Werbos introduced the backpropagation through time (BPTT) approach [10, 11], by unfolding the temporal operation of RNNs into a FNN, the topology of which grows by one layer at every time step. BPTT algorithm calculates the exact model derivatives by pursuing all k steps back into the past which makes it an offline training method. Since the number of training data samples is generally quite large, the computational effort and memory requirement of BPTT are usually unacceptable in practice. To the application of real-time situation, [33] employs a Truncated BPTT (T-BPTT) method. It is an approximation version of error gradient by going back to only few steps. In such case, a tradeoff exists between the computational effort and the accuracy of the derivative calculation. In any events, the truncation depth must be large enough to produce derivative that closely approximate the actual values. Another algorithm proposed by Williams *et al.* to overcome the shortcoming of BPTT is real time recurrent learning (RTRL) [12, 13]. RTRL updates the synaptic weights of a fully connected RNN in real time. It is more efficient since it doesn't unfold the model into the past, but instead assumes that the model parameters do not change during a specific time interval, i.e.,

$$V(k) = V(k-1) = \dots = V(k-d)$$

$$W(k) = W(k-1) = \dots = W(k-d)$$

where d is the window size. This assumption is exactly satisfied for batch learning mode, where the weight parameters are updated only when all the training data is presented. It can also applied for sequential training mode when the learning rate

is small enough such that the weight changes during some steps are negligible, i.e.,

$$V(k) \approx V(k-1) \approx \dots \approx V(k-d)$$

$$W(k) \approx W(k-1) \approx \dots \approx W(k-d)$$

The RTRL is not guaranteed to follow the negative gradient of the error function with respect to the weight matrices. The approximation of the true error gradient by RTRL may result in instability in the system. This effect can be avoided by using small learning rate. RTRL is simpler and faster compared to BPTT, which requires less memory and its complexity does not depend on the size of training duration. However, both BPTT and RTRL suffer from slow convergence when dealing with statistically non-stationary input. Mandic and Chambers [6] propose a normalized RTRL (N-RTRL) method that is similar to the idea of normalized least mean square (NLMS) [63] to train RNNs. It introduces a normalization factor in the output layer training which magnitude varies in time with the power of the first order derivative of activation function. In [64], an adaptive learning rate is developed based on N-RTRL to improve the robustness of the gradient training. Song proposes a robust adaptive gradient-descent training (RAGD) algorithm for RNNs with an optimal or suboptimal tradeoff between RNNs training accuracy and weight convergence [39]. RAGD uses a specifically designed error derivative, similar to RTRL training, to approximate the recurrent gradient for real-time learning, which provides an improved training speed over the standard RTRL algorithm and guaranteed weight convergence.

Gradient-descent learning using first-order (gradient) information is conceptually very simple. However, in practical implementations, it may lead to several problems: 1) the results may be trapped at local minima of the error function; 2) the convergence in backpropagation learning is not guaranteed; 3) slow convergence. To overcome these shortcomings, some novel training methods are proposed. Some researchers turn to use the second-order derivative information such as Newton method and Quasi-Newton learning algorithm. Conjugate gradient is used to ac-

celerate the typically slow rate of convergence while avoiding the computational requirement associated with the evaluation, storage, and inversion of the Hessian matrix in Newton's method. It is proven to be remarkably effective in dealing with general objective functions and is considered among the best general purpose optimization methods presently available. There are some variations of conjugate gradient method for RNNs training [65–67]. Extended Kalman filter (EKF) algorithm is also used to train RNN to overcome the slow converge problem of gradient-descent method [68]. A main drawback of the EKF-based training is that theoretical analysis requires the uncertainty of neural modeling to be Gaussian process. Certainly, the use of higher order information about the error surface can be an effective way of generating improved solutions at each step of the process. But, when dealing with highly nonlinear large-scale problems, practical aspects related with excessive computational burden, numerical errors in matrix operations, and the need for large matrix storage can make such a kind of second-order methods unfeasible to be implemented. In such a situation, even a first-order method is a better choice. In general, these aspects are frequently present in the supervised neural network learning, where the search space is highly nonlinear and of large dimension. These are the main reasons for the widespread use of the first order algorithm, in spite of its slowness and oscillatory behavior.

The difficulty in training RNNs often results in approximations. Spall proposes an efficient parameter approximation method through simultaneous perturbation of the weights via a stochastic approach [40], whereas in [4] the weights are trained by approximating the gradient by simultaneous perturbation stochastic approximation method under the framework of a deterministic system. Sundareshan presents two alternate learning procedures that do not require gradient evaluations [50]. They describe two distinct approaches. One uses concepts from the theory of learning automata and the other is based on the classical simplex optimization approach. The complexity of this task reveals the unique capability of RNNs for approximating temporal dynamics. In [69], a stable bounding ellipsoid algorithm is proposed and stability analysis of identification error with the BE algorithm is given by a

Lyapunov-like technique.

Kernel learning based on the use of Mercer kernel functions is very attractive. Kernel-based learning first appears in the form of support vector machines which is an offline method. The basic idea behind kernel methods is that a Mercer kernel function, which is applied to pairs of input vectors, can be interpreted as an inner product in a high-dimensional Hilbert space or the feature space, thus allowing inner products in the feature space to be computed without making direct reference to feature vectors [70]. This idea, which is commonly known as the kernel trick [71], has been used extensively in recent years. Existing kernel online training algorithms include kernel least mean squares (KLMS) [72], kernel recursive least squares (KRLS) [70], kernel affine projection algorithms (KAPA) [73], quantized kernel least mean square algorithm [74] and extended kernel recursive least squares (EX-KRLS) [75]. These methods generalize naturally the classical linear least square algorithm in reproducing kernel Hilbert spaces (RKHS) and are very appealing for nonlinear adaptive systems. However, the bottleneck of the online kernel learning algorithm is that its computational complexity increases with the growing number of training samples. When training samples are very large, the kernel method without sparse representation will be very slow and need a large memory space to store the sample information. In order to make the kernel methods suitable for online learning, sparse representation should be developed to select a compact dictionary (support vectors set) to represent the whole training data in order to reduce the number of kernel functions, which can reduce the computational cost and generalize the algorithm well. Consider time-series prediction problem where a set of input-output data arising from some unknown source is given as

$$S^t = \{(x_1, y_1), \dots, (x_t, y_t)\}.$$

The goal of the prediction problem is to find the best predictor $\hat{y}_t$ for y_t based on $S^{t-1} \cup x_t$. Kernel methods present a nonparametric alternative for the conventional

parametric methods. Solutions of the kernel methods are in a linear model:

$$\hat{y}(\cdot) = \sum_{j=1}^{m_t} \alpha_j k(\cdot, c_j) \quad (2.3)$$

where $k(\cdot, c_j)$ is the kernel function and c_j is the j th center and α_j is the j th coefficient. Denote $C(t) = \{c_j\}_{j=1}^{m_t}$ as the center set (or dictionary). When a new example (x_t, y_t) is presented, the sparsification rule needs to decide if x_t is accepted as a new center.

There are mainly two existing sparsification criteria in the literature of online kernel learning: the novelty criterion (NC) introduced by Platt [76] and the approximate linear dependency (ALD) test proposed by Engel [70]. NC evaluate the distance of x_t to the present dictionary

$$\delta_1 = \min_{c_j \in C(i)} \|x_t - c_j\|. \quad (2.4)$$

If $\delta_1 < \nu_0$, x_t will not be accepted as a new kernel center. Otherwise, it further computes the prediction error $e_t = y_t - \hat{y}(x_t)$. Only if $|e_t| > \nu_1$, x_t will be added into the dictionary as a new kernel center. ν_0 and ν_1 are two user-specified parameters [77]. In the ALD based sparsification process, a new input sample was inserted into the dictionary $C(t) = \{c_j\}_{j=1}^{m_t}$ only if its feature space image could not be sufficiently well approximated by the previously selected dictionary members. The ALD condition is to test whether the input feature is approximately linearly dependent on the dictionary vectors:

$$\delta_2 = \min_{\alpha} \left\| \sum_{j=1}^{m_t} \alpha_j k(c_j, \cdot) - k(x_t, \cdot) \right\|$$

By straightforward calculus, it turns out that

$$\delta_2^2 = k(x_t, x_t) - \mathbf{h}(i)^T \mathbf{G}^{-1}(i) \mathbf{h}(i) \quad (2.5)$$

where

$$\mathbf{h}(i) = [k(x_t, c_1), \dots, k(x_t, c_{m_t})] \quad (2.6)$$

$$\mathbf{G}(i) = \begin{bmatrix} k(c_1, c_1) & \dots & k(c_{m_t}, c_1) \\ \vdots & \ddots & \vdots \\ k(c_1, c_{m_t}) & \dots & k(c_{m_t}, c_{m_t}) \end{bmatrix} \quad (2.7)$$

If δ_2^2 in (2.5) is smaller than a predefined threshold ν , $k(x_t, \cdot)$ can be approximated by linear combination of current dictionary members and it is not necessary to insert x_t into the dictionary. ALD sparsification process has been incorporated into the popular recursive least square algorithm as KRLS algorithm [70]. ALD leads to elaborate and costly operations which scales quadratically with the size of the dictionary. A simplified sparsification rule is to use the “nearest” center in the dictionary to estimate the overall distance [77], i.e.,

$$\delta_3 = \min_{\forall \alpha, \forall c_j \in C(i)} \|k(x_t, \cdot) - \alpha k(c_j, \cdot)\|$$

By straightforward calculus, one has

$$\delta_3^2 = \min_{\forall c_j \in C(i)} \left[k(x_t, x_t) - \frac{k^2(x_t, c_j)}{k(c_j, c_j)} \right] \quad (2.8)$$

When the kernel is unit-norm, that is $k(x, x) = 1$ for all x , this distance measure is equivalent to the coherence measure. A coherence-based sparsification rule is proposed in [78] to further reduce the computational complexity compared to ALD. Coherence is a fundamental parameter to characterize a dictionary in linear sparse approximation problems [78, 79]. In [78], coherence-based sparsification rule is developed. Consider the Babel function given by

$$\mu_1(m_t) = \max_i \max_{j \in \mathcal{J}, i \neq j} \sum_{j=1}^{m_t} |k(c_i, c_j)| \quad (2.9)$$

where $\mathcal{J}$ is a set of m_t indices. Function $\mu_1(m_t)$ is defined as the maximum total coherence between a fixed kernel function $k(\cdot, c_i)$ and a subset of m_t other functions $k(\cdot, c_j)$ of the dictionary. The sufficient condition for a dictionary of kernel functions to be linearly independent is presented.

Proposition 1. [78] *Let $k(\cdot, c_1), \dots, k(\cdot, c_{m_t})$ be an arbitrary set of m kernel functions from a dictionary, and let $\mu_1(m_t)$ be the babel function evaluated for this set. If $\mu_1(m_{t-1}) < 1$, then this set is linearly independent.*

The computation of $\mu_1(m_{t-1})$ is expensive, so the simpler condition with low computational cost is proposed as follows

$$\max_{j \in \mathcal{J}_{t-1}} |k(x_t, c_j)| \leq \mu_0 \quad (2.10)$$

where μ_0 is a parameter in $[0,1]$ determining both the level of sparsity and the coherence of the dictionary. It also shows that under a reasonable condition on the input space, the dimension of the dictionary determined under rule (2.10) remains finite as t goes to infinity.

Proposition 2. [78] *Let $\mathcal{U}$ be a compact subspace of a Banach space, and $k : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ be a Mercer kernel. Then, the dimension of the dictionary determined under the sparsification rule (2.10) with $0 \leq \mu_0 < 1$ is finite for any sequence $\{c_i\}_{i=1}^\infty$.*

The above proposition implies that the computational cost per time-step of algorithms implementing the strategy (2.10) becomes independent of time after a transient period. After such period, the computational cost depends only on the cardinality m_t of the final dictionary, which is a function of the threshold μ_0 [78]. The sparsification rule (2.10) is an alternative to the ALD condition. The connection between the ALD condition and rule (2.10) is given in the following proposition.

Proposition 3. [78] *Let $k(\cdot, c_1), \dots, k(\cdot, c_{m_t})$ be m_t kernel functions selected by the coherence-based rule (2.10). If $(m_t - 1)\mu_0 < 1$, then the norm of the projection of $k(\cdot, c_{m_t})$ onto the span of the other $m_t - 1$ kernel functions is less than or equal to $\sqrt{(m_t - 1)\mu_0^2 / (1 - (m_t - 1)\mu_0)}$.*

It is concluded that the coherence-based rule (2.10) implicitly specifies a lower bound on the squared error which is explicitly governed by the threshold parameter in the ALD. Both approaches can generate linearly independent sets of kernel functions,

a constraint that will be ignored in what follows [78]. The coherence-based rule is simpler and less time consuming than ALD.

The common characteristic of above-mentioned methods is that they are all based on the characteristics of input feature, while the reliability of the system output is not taken into consideration. In a different perspective, an information theoretic approach based on the surprise measure is proposed for different kinds of sparse kernel adaptive filters [77]. The surprise measure is defined as the negative log likelihood of the posterior joint distribution of the system input and desired output. In general cases, when the distribution of the input features is not available or assumed as uniform in the process of evaluating the surprise measure, the surprise measure mainly takes the posterior distribution of system output into consideration and the characteristics of the input features play a relative less important role.

The kernel methods of recurrent version are also promising tools for nonlinear signal processing especially when they are applied for non-stationary signal processing. The recurrent input-output model could be extended to the kernel methods and an online kernel-based learning method of recurrent version can be derived. One preliminary work is [80]. The paper introduces the temporal-kernel recurrent neural network (TKRNN), which is a variant of the RNN by representing the input string with a stable state of its hidden units. TKRNN can cope with long-term dependencies much more easily than a standard RNN. It also shows that the TKRNN developed short-term memory which can successfully solve the serial recall task.

2.2.2 Stability of RNNs

RNNs can be viewed as nonlinear dynamical systems. Thus, a particular concern is about the system stability. For the case that RNNs are used as associative memory, the stability analysis mainly focuses on establishing conditions that ensure the global asymptotic stability of equilibrium points of an RNN. Recently, the equilibria and stability properties of RNNs are studied and various sufficient conditions for the uniqueness and stability of the equilibrium point for RNNs are also established. Liu and Michel establish the results for the globally asymptotical stability of the equi-

librium of a discrete-time RNN with state saturations using the Lyapunov function method which can be used to solve a problem of the n th-order fixed-point digital filters. It is also pointed out that the global stability of the NNs guarantees the nonexistence of limit cycles in such filters [81]. Matsuoka gets sufficient conditions on the asymmetric weight matrix of a type of RNNs to insure the absolute stability using a new Lyapunov function [82]. Forti *et al.* obtain the necessary and sufficient conditions for absolute stability of Hopfield neural networks with symmetric weight matrix. The conclusion is that the negative semidefiniteness of the weight matrix is a necessary and sufficient condition for absolute stability and that the class of neural circuits with a negative semidefinite weight matrix is the largest class of circuits that can be employed for embedding and solving optimization problems without the risk of spurious responses [83]. An analysis of the absolute stability for a general class of discrete-time RNNs is presented in [84] and some sufficient conditions for the absolute stability are derived using Ostrowskis theorem and the similarity transformation approach. Later, Arik and Tavsanoğlu derive two conditions concerning absolute stability of nonsymmetric RNNs. Each condition guarantees the uniqueness and global asymptotic stability of the equilibrium point for different classes of activation functions and under different constraint conditions imposed on the weight matrix [85]. Mandić *et al.* discuss the relationship between the number of neurons in the structure, the slope in the activation function and a measure of the weight matrix that guarantee convergence of a relaxation process realized by fully connected RNNs [6]. Barabanov *et al.* address the problem of global Lyapunov stability of discrete-time RNNs in the unforced (unperturbed) setting and obtains sufficient conditions on weight matrices and biases which guarantee that the states of the RNNs converge to a unique equilibrium point [86]. Hu *et al.* present new results on global asymptotic stability (GAS) and global exponential stability (GES) of a general class of continuous-time RNNs with Lipschitz continuous and monotone nondecreasing activation functions [87]. Hu *et al.* further extend and improve the existing absolute stability and the absolute exponential stability (AEST) in the literature to present a new result on AEST of a class of continuous-time RNNs with

locally Lipschitz continuous and monotone nondecreasing activation functions. The additively diagonally stable connection weight matrices are proven to be able to guarantee AEST of the RNNs [23]. In [88], both the existence of an equilibrium point and the AEST of RNNs with generalized activation functions are addressed. Wang *et al.* present a set of sufficient and necessary conditions for GES of a class of generic discrete-time RNNs. By means of the uncovered conditions, GES and convergence properties of the RNNs are analyzed quantitatively. It is shown that exact equivalences exist among the GES property of the RNNs, the contractiveness of the deduced nonlinear operators, and the GAS of the RNNs plus the spectral radius of Jacobian matrix of the RNNs at the unique equilibrium point less than one. When the RNNs have small state feedback coefficients, it is shown further that the infimum of exponential bounds of the trajectories of the neural networks equals exactly the spectral radius of Jacobian matrix of the neural networks at the unique equilibrium point [89]. The difference between GES and GAS of the discrete-time RNNs is also clarified. Wang *et al.* further reveals two important characterizations of GES of a generic class of continuous-time RNNs [90]. Cao and Zeng work on the global asymptotic and exponential stability of RNNs with time delays [15, 18, 22, 91–94]. Unfortunately, most of the existing results can only check the stability of RNNs after the network weights are fixed to some values which is after training of RNNs. If some of the equilibrium points are unstable, the RNNs must be re-trained by using a small enough learning rate, which is very time-consuming. Thus, there is a demand to develop a systematic way to analyze the stability of dynamic systems during dynamic learning processes, which is the major concern of this thesis. Literatures about stability analysis of RNNs during training is rather scarce. Ku and Lee propose a model of diagonal recurrent neural network (DRNN) which is applied to dynamic system control problem. Online dynamic backpropagation algorithm (DBp) is developed and used to train both DRNC and DRNN. Guaranteed convergence in the sense of Lyapunov function is achieved by using adaptive learning rates [95]. Probabilistic convergence of a backpropagation algorithm for RNNs containing either output or hidden layer recurrence is obtained in [96]. In [97], Liang *et al.* develop the

multiplier and constrained learning rate algorithms to ensure the stability of the RNN in the sense of Lyapunov function during a dynamic weight learning process which is applied for associative memory. In [98], the stability condition for robust local stability of multilayer RNN with two hidden layers is derived which follows from linking theory about linearization, robustness analysis of linear systems under nonlinear perturbation and matrix inequalities. Modified dynamic backpropagation constrained by the new stability condition is also presented. In [6], techniques for convergence analysis for an online stochastic gradient descent algorithm for RNNs are presented based on the contraction mapping properties of nonlinear neurons. Xu *et al.* use an offline gradient-descent algorithm to train the diagonal RNNs in the case that only a small number of training samples are available and prove some convergence results under deterministic nature [99]. Patan introduces a class of locally recurrent globally feedforward networks and the stability analysis is solved as a constrained optimization task based on gradient projection and minimum distance projection methods. These two methods are also introduced into the existing learning algorithm to develop convergence conditions [100]. In [101], an extended Kalman filter-based RNN training approach is proposed based on the Lyapunov method and the maximum likelihood method with a controllable training convergence. In [39], a robust adaptive gradient-descent (RAGD) training algorithm of RNN is proposed based on a novel RNN hybrid training concept. The weight convergence and L_2 -stability of the algorithm are derived via the conic sector theorem. The key characteristic of the RAGD algorithm is the use of three adaptive parameters to adjust the effective adaptive learning rate and to provide guaranteed weight convergence and system stability for the RNN training. The optimized adaptive learning maximizes the training speed of RNNs for each weight update without violating the stability and convergence criteria. Song develops a robust extended Elman BP (eEBP) training algorithm [5] for Elman neural networks with a new adaptive dead zone scheme based on eEBP training concepts. The convergence of the weights with the adaptive learning rate and adaptive dead zone estimates is proven in the sense of Lyapunov functions.

2.3 Generalization of RNNs

During the training of an RNN, we typically use as many of the training examples as possible into the network to update the synaptic weights of an RNN, hoping that the RNN so designed will generalize. However, when a neural network learns too many input-output examples, the network may end up with memorizing the training data which is called overfitting. Therefore, the goal of network training is not only to learn an exact representation of the training pattern, but to build a model of the process. In particular, theoretical studies [34, 35] have shown that when the number of hidden units equals the number of training examples (minus one), the backpropagation error surface is guaranteed to have no local minima. On the other hand, both theory [36] and experiments [37, 38] show that networks with fewer free parameters exhibit a better generalization performance. Lastly, from an implementation point of view, smaller networks require fewer resources in any physical computational environment.

There are two alternative ways that can be taken for the issue of generalization [33]:

1. Fix the architecture of the network and adjust the size of the training data set for a good generalization.
2. The set of the training data is fixed or unknown, and the issue of interest is to determine the best network architecture for a good generalization .

However, when the training is online, the training data set cannot be repeatedly accessed and whether the data are for training or generalization cannot be determined. The only solution is the optimization of the network architecture. To solve the problem of optimizing the size of NNs, two different incremental approaches are often used. The first one starts with a small initial network and gradually adds a new neuron or a new layer of hidden neurons when the desired specification cannot be met [102, 103], which is named as growing technique. The second starts with a relatively large network and then prune it by weakening or eliminating certain synaptic weights in a selective and orderly fashion. This approach combines the advantages of training large networks and those of running small ones.

2.4 Mathematical Preliminaries

2.4.1 Normed Spaces and Inner Products

Definition 1. (*Vector Space*) A linear space V over a field F is a set whose elements are called vectors and where two operations, addition and scalar multiplication, are defined:

1. the addition $+$ is defined on V and whenever $x, y, z \in V$ is satisfied

$$(a) \ x + y = y + x$$

$$(b) \ (x + y) + z = x + (y + z)$$

$$(c) \ (V, +) \text{ is a group, with neutral element denoted by } 0 \text{ and inverse denoted by } -, \ x + 0 = 0 + x = x \text{ and } x + (-x) = x - x = 0$$

2. scalar multiplication of $x \in V$ by elements $a \in F$, denoted by $ax \in V$, and the following properties are satisfied for all $a, b \in F$

$$(a) \ a(x + y) = ax + ay$$

$$(b) \ (a + b)x = ax + bx$$

$$(c) \ 1 \cdot x = x \text{ where } 1 \text{ denotes the unit in } F$$

Definition 2. (*Metric Space*) A metric space is an ordered pair (S, d) where S is a set and d is a metric on S , i.e., a function $d : S \times S \rightarrow R$ such that for any $x, y, z \in S$, the following holds

$$1. \ d(x, y) \geq 0 \text{ (non-negative)}$$

$$2. \ d(x, y) = 0 \text{ iff } x = y$$

$$3. \ d(x, y) = d(y, x)$$

$$4. \ d(x, z) \leq d(x, y) + d(y, z)$$

The function d is also called distance function or simply distance. Often, d is omitted and one just writes S for a metric space if it is clear from the context what metric is used [104].

Definition 3. (*Vector Norms*) A norm is a function that assigns a strictly positive length or size to all vectors in a vector space, other than the zero vector. The norm of a n -dimensional vector $x = (x_1, x_2, \dots, x_n)$ is usually denoted $\|x\|$. The definitions of some frequently used norms are

1. (L_1 norm) $\|x\|_1 = \sum_{i=1}^n |x_i|$
2. (L_2 norm) $\|x\|_2 = \sqrt{x_1^2 + \dots + x_n^2}$
3. (Maximum norm) $\|x\|_\infty = \max(|x_1|, \dots, |x_n|)$
4. (p -norm) $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$. Note that for $p = 1$ we get the L_1 norm, for $p = 2$ we get the (L_2 norm), and as p approaches infinity, the p -norm approaches the maximum norm.

Definition 4. (*Normed Vector Space*) A normed vector space $(V, \|\cdot\|)$ is a metric space by defining $d(x, y) = \|x - y\|$ which satisfies the following axioms for $x \in V$

1. $\|x\| \geq 0$, $\|x\| = 0 \Leftrightarrow x = 0$, $\forall x \in V$
2. $a\|x\| = |a| \|x\|$, $\forall a \in \mathbb{R}$ and $\forall x \in V$
3. $\|x + y\| \leq \|x\| + \|y\|$, $\forall x, y \in V$

Definition 5. (*Inner Product Space*) A inner product space is a vector space V over the field F (the field of real or complex numbers) together with an inner product, i.e., with a map $\langle \cdot, \cdot \rangle : V \times V \rightarrow F$ that satisfies the following three axioms for all vectors $x, y, z \in V$ and all scalars $a \in F$

1. (Conjugate symmetry) $\langle x, y \rangle = \overline{\langle y, x \rangle}$
2. (Linearity) $\langle ax, y \rangle = a \langle x, y \rangle$, $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
3. (Positive-definiteness) $\langle x, x \rangle \geq 0$ with equality only for $x = 0$

Definition 6. (*Cauchy Sequence*) Given a metric space (S, d) , a sequence $\{x_i\}_{i=1}^\infty$ is a Cauchy sequence, if for every positive real number $\mu > 0$ there is a positive

integer M such that for all natural numbers $m, n > M$, the distance $d(x_m, x_n) < \mu$. Roughly speaking, the terms of the sequence become arbitrarily close to each other as the sequence progresses.

Definition 7. (Convergence) Let $\{x_i\}_{i=1}^{\infty}$ be a sequence of elements on a normed space L . $\{x_i\}_{i=1}^{\infty}$ converges to $x_0 \in L$ if $\lim_{n \rightarrow \infty} \|x_n - x_0\| = 0$, i.e., if for all $\varepsilon > 0$, there exists an $N(\varepsilon)$ such that

$$\|x_i - x_0\| < \varepsilon, \forall i > N(\varepsilon)$$

the sequence is convergent to $x_0 \in L$.

Definition 8. (Banach Space) A normed vector space $(L, \|\cdot\|)$ is complete if every Cauchy sequence converges to an element of L . Such a complete space is called a Banach Space.

Definition 9. (Hilbert Space) An inner product space L which is complete with the norm $\|\cdot\| : L \times L \mapsto \mathbb{R}^+$ defined by $\|x\| = \langle x, x \rangle^{1/2}$ is called a Hilbert space. The distance between two points x, y in the Hilbert space H is defined in terms of the norm by $d(x, y) = \|x - y\| = \sqrt{\langle x - y, x - y \rangle}$.

Definition 10. (Matrix Norms) A matrix norm is simply a vector norm on the finite dimensional vector spaces. Let $\|\cdot\|_p$ a given vector norm on $\mathbb{C}^n$, the corresponding induced p -norm for any $A \in \mathbb{C}^{m \times n}$ on $\mathbb{C}^{m \times n}$ and $x \in \mathbb{C}^n$ is

$$\|\cdot\|_p = \max_{\|x\|_p=1} \|Ax\|_p = \max \frac{\|Ax\|_p}{\|x\|_p}$$

For $p = 1, 2, \infty$, the induced matrix norms can be derived as

1. $\|A\|_1 = \max_j \sum_{i=1} |a_{ij}|$ (column sums)
2. $\|A\|_2 = \bar{\sigma}$, the largest singular value of A
3. $\|A\|_{\infty} = \max_i \sum_{j=1} |a_{ij}|$ (row columns)

4. $\|A\|_{\max} = \max\{|a_{ij}|\}$ (*max norm*)
5. (*Frobenius Matrix Norm*) $\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 = \text{trace}(A^*A)$
6. (*Trace Norm*) $\|A\|_{tr} = \text{Trace}(\sqrt{A^*A}) = \sum_{i=1}^{\min\{m,n\}} \sigma_i$, where the function $\text{Trace}\{\cdot\}$ is the sum of the elements on the main diagonal of the variable matrix, e.g.

$$\text{Trace}\{A\} = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{i=1}^n a_{ii} \quad (2.11)$$

Lemma 2.1. Let $\|\cdot\|_p$ an induced matrix norm. For any $A, B \in \mathbb{C}^{n,n}$, $x \in \mathbb{C}^n$ we have $\|AB\|_p \leq \|A\|_p \|B\|_p$ and $\|Ax\|_p \leq \|A\|_p \|x\|_p$. More generally, all norms with this property are called compatible, multiplicative, or sub-multiplicative matrix norms.

2.4.2 Reproducing Kernel Hilbert Space

Definition 11. (*Reproducing Kernel*) Given a set χ , a function $\kappa(x, y)$ is a reproducing kernel of a Hilbert space $\mathcal{H}$ if and only if the following two conditions are satisfied

1. $\kappa(x, y) \in \mathcal{H}$, $\forall y \in \chi$
2. $(f(x), \kappa(x, y)) = f(y)$

Some properties of reproducing kernels are:

1. If a reproducing kernel $\kappa(x, y)$ exists, then it is unique
2. A reproducing kernel exists if and only if $|f(y)| \leq c(y) \|f\| \forall f \in \mathcal{H}$ where $c(y) = \|\kappa(\cdot, y)\|$.

Definition 12. (*Reproducing Kernel Hilbert Space*) A Reproducing Kernel Hilbert Space (RKHS) is a Hilbert space $\mathcal{H}$ with a reproducing kernel whose span is dense in $\mathcal{H}$.

Definition 13. (*Finitely positive semi-definite functions*) Given a space X , a function $\kappa : X \times X \rightarrow \mathbb{R}$ satisfies the finitely positive semi-definite property if it is a symmetric function for which the matrices formed by restriction to any finite subset of the space X are positive semi-definite.

Theorem 2.1. (*Mercer's Theorem*) [105] Let X be a compact subset of $\mathbb{R}^n$. Suppose κ is a continuous symmetric function such that the integral operator $T_\kappa : L_2(X) \rightarrow L_2(X)$

$$(T_\kappa f)(\cdot) = \int_X \kappa(\cdot, x) f(x) dx$$

is positive, that is

$$\int_{X \times X} \kappa(x, z) f(x) f(z) dx dz \geq 0$$

for all $f \in L_2(X)$. Then we can expand $\kappa(x, z)$ in a uniformly convergent series (on $X \times X$) in terms of functions ϕ_j , satisfying $\langle \phi_j, \phi_i \rangle = \delta_{ij}$

$$\kappa(x, z) = \sum_{j=1}^{\infty} \phi_j(x) \phi_j(z).$$

Furthermore, the series $\sum_{i=1}^{\infty} \|\phi_i(x)\|_{L_2(X)}^2$ is convergent.

Mercer's theorem is usually used to construct a feature space for a valid kernel.

2.4.3 Lyapunov Stability Theorems

Theorem 2.2. [106] (*Second Method of Lyapunov: For Continuous-Time Autonomous System*) . Consider an autonomous nonlinear dynamical system

$$\dot{x}(t) = f(x), x(t_0) = x_0 \in \mathbb{R}^n \quad (2.12)$$

where $x(t) \in \mathcal{D} \subseteq \mathbb{R}^n$ denotes the system state vector, $\mathcal{D}$ is an open set containing the origin, and $f : \mathcal{D} \rightarrow \mathbb{R}^n$ is continuous on $\mathcal{D}$. The autonomous system (2.12) is globally (over the entire $\mathcal{D}$) and asymptotically stable about its zero equilibrium, if there exists a scalar-valued function $V(x)$, defined on $\mathcal{D}$, such that

1. $V(0) = 0$.
2. $V(x) > 0$ for all $x \neq 0$ in $\mathcal{D}$.
3. $\dot{V}(x) < 0$ for all $x \neq 0$ in $\mathcal{D}$.

Theorem 2.3. [106] (*Second Method of Lyapunov: For Discrete-Time Autonomous System*) . Let $x^* = 0$ be an equilibrium of the discrete-time autonomous system

$$x_{k+1} = f(x_k). \quad (2.13)$$

where $f : \mathcal{D} \rightarrow \mathbb{R}^n$ is continuously differentiable in a neighborhood of the origin. Then the system is globally (over the entire domain $\mathcal{D}$) and asymptotically stable about this zero equilibrium if there exists a scalar-valued function $V(x_k)$, defined on $\mathcal{D}$ and continuous in x_k , such that

1. $V(0) = 0$.
2. $V(x_k) > 0$ for all $x_k \neq 0$ in $\mathcal{D}$.
3. $\Delta V(x_k) := V(x_k) - V(x_{k-1}) < 0$ for all $x \neq 0$ in $\mathcal{D}$.
4. $V(x) \rightarrow \infty$ as $\|x\| \rightarrow \infty$.

Note that the Lyapunov theorems stated above only offer sufficient conditions for asymptotic stability.

Chapter 3

Multivariate RNNs with Robust Adaptive Gradient-Descent Training Algorithm

Transient response is a critical issue for real-time signal processing applications. Conventional training algorithms of RNNs such as backpropagation through time (BPTT) [10, 11] and real time recurrent learning (RTRL) [12, 13] algorithms often suffer from drawbacks of slow convergence speed and heavy computational load. Although increasing the learning rate may help to improve the performance of the RNN, it can result in unstable training in terms of weight divergence. A robust adaptive gradient-descent (RAGD) training algorithm of RNN is developed in [39] based on a novel RNN hybrid training concept to solve these problems. It switches the training patterns between standard real-time online backpropagation (BP) and RTRL according to the derived convergence and stability conditions. However, the RAGD algorithm is only designed for MISO system. RNNs are highly nonlinear dynamic system, the stability analysis of MIMO RNNs is much more complex than that of MISO RNNs.

In this chapter, a multivariate robust adaptive gradient-descent (MRAGD) [107] training algorithm is developed to train the multivariate RNNs. The weight convergence and system stability are proven in the sense of Lyapunov function. In

addition, a RNN based robust tracking controller is designed for a class of MIMO discrete-time nonlinear systems and the closed-loop stability is proven. Simulation results are presented for a two-link robot tracking control problem.

3.1 Basics of the RNN

Consider a three-layer Jordan RNN. The output vector of the RNN at time instant k can be written as:

$$\hat{y}(k) = V(k)\Phi(W(k)x(k)) \in \mathbb{R}^m \quad (3.1)$$

where $V(k) \in \mathbb{R}^{m \times n_h}$ and $W(k) \in \mathbb{R}^{n_h \times n_I}$ are the matrix form of output and hidden layer weights respectively and n_h is the dimension of hidden layer. $x(k)$ is the input vector for the RNN which is made up of the external input vector and delayed RNN outputs

$$\begin{aligned} x(k) &= [u_1(k) \dots u_m(k) \dots \hat{y}^T(k-d) \dots \hat{y}^T(k-1)]^T \\ &= [x_1(k) \dots x_n(k)]^T \in \mathbb{R}^{n_I} \end{aligned} \quad (3.2)$$

where $n_I = n + m \times d$, n is the dimension of the external input vector and d is the maximum time delay factor of output recurrent feedback of the network. The output of hidden layer is

$$\Phi(W(k), x(k)) = [\Phi_1(k) \dots \Phi_m(k)]^T \quad (3.3)$$

where $\Phi_j(k), j = 1, \dots, n_h$ is the activation function defined as

$$\Phi_j(k) = \Phi(W_{j,:}(k)x(k)) = \frac{1}{1 + \exp(-4W_{j,:}(k)x(k))} \quad (3.4)$$

and $W_{j,:}(k) \in \mathbb{R}^{1 \times n_I}$ is the j th row vector of $W(k)$.

Define $y(k) \in \mathbb{R}^{m \times 1}$ as the desired output of the RNN, which can be presented as $y(k) = V^*\Phi(W^*, x(k))$ and $V^* \in \mathbb{R}^{m \times n_h}$ and $W^* \in \mathbb{R}^{n_h \times n_I}$ are the ideal output

layer and hidden layer weights respectively. Hence, the error is defined as

$$e(k) = y(k) - \hat{y}(k) + \varepsilon(k) \quad (3.5)$$

where $\varepsilon(k)$ is the disturbance.

The RNN training objective can be stated as updating the weights of each layer recursively to make the network output best fit the given data that minimizes $E(k)$ as

$$E(k) = \|e(k)\|^2 / 2.$$

Specifically, a gradient-type algorithm, such as the popular least mean square (LMS) minimization, can be used to recursively reduce the $E(k)$ by estimating the weight at each instant in time as follows:

$$V(k+1) = V(k) - \alpha \frac{\partial E(k)}{\partial V(k)} \quad (3.6)$$

$$W(k+1) = W(k) - \alpha \frac{\partial E(k)}{\partial W(k)} \quad (3.7)$$

where the step size α is the learning rate.

Many early works use feedforward neural network type of training to train RNNs, thus (3.6) and (3.7) can be written as

$$\begin{aligned} V(k+1) &= V(k) - \alpha \frac{\partial E(k)}{\partial V(k)} \\ &= V(k) - \frac{\alpha}{2} \frac{\partial (e^T(k)e(k))}{\partial V(k)} \end{aligned}$$

$$\begin{aligned} W(k+1) &= W(k) - \alpha \frac{\partial E(k)}{\partial W(k)} \\ &= W(k) - \frac{\alpha}{2} \frac{\partial (e^T(k)e(k))}{\partial W(k)} \end{aligned}$$

The above two equations only consider the relationship between the weight vectors and $\hat{y}(k)$. However, the input vector $x(k)$ (3.2) contains the delayed recurrent output components which is also a function of the weight vector $V(k)$ and $W(k)$. Simply

ignore this relationship in the calculation of the gradient will lose much information and result in a slow convergence speed. To differentiate RNNs from feed-forward neural networks, RTRL or BPTT training considers the feedback signal in the calculation of the gradient to facilitate the dynamic properties of RNNs. Thus (3.6) and (3.7) can be written as

$$\begin{aligned} V(k+1) &= V(k) - \alpha \frac{dE(k)}{dV(k)} \\ &= V(k) - \frac{\alpha}{2} \left[\frac{\partial (e^T(k)e(k))}{\partial V(k)} + \frac{\partial (e^T(k)e(k))}{\partial x^T(k)} \frac{\partial x^T(k)}{\partial V(k)} \right] \end{aligned} \quad (3.8)$$

$$\begin{aligned} W(k+1) &= W(k) - \alpha \frac{dE(k)}{dW(k)} \\ &= W(k) - \frac{\alpha}{2} \left[\frac{\partial (e^T(k)e(k))}{\partial W(k)} + \frac{\partial (e^T(k)e(k))}{\partial x^T(k)} \frac{\partial x^T(k)}{\partial W(k)} \right] \end{aligned} \quad (3.9)$$

However, RTRL or BPTT with a fixed learning rate rule has several problems. First, the weight does not converge to an optimal or suboptimal value because of the possible disturbances in real-world applications. To solve this problem, we propose using a dead zone scheme [108] to estimate the noise level and an adaptive learning rate to guarantee the weight convergence. Second, the recurrent structure of the output signal $\hat{y}(k)$ provides feedback into the RNN input vector $x(k)$. Therefore, the recurrent training signals, such as $(\partial (e^T(k)e(k))/\partial x^T(k)) (\partial x^T(k)/\partial V(k))$ in the output layer training, which are important in the typical RTRL and BPTT schemes cause a further weight divergence due to the negative component contribution. Third, the fixed learning rate α tends to minimize the cost function only for the training data, this may not be the best way to obtain a small testing error with unknown testing data. In the following sections, we examine a MRAGD approach with an improved adaptive learning capability for modifying the standard RTRL training algorithm for RNNs found in (3.8) and (3.9). The MRAGD approach is based on analyzing the RNNs weight convergence and stability conditions, which explains why we need the dead zone parameter and an adaptive learning rates to replace the fixed learning rate and overcome the weight divergence problem in the RTRL training.

In a multilayered neural network, the output layer weight vector and hidden layer weight vector are normally updated separately using different learning rates as in the standard BP training algorithm [33]. During the updating of the output layer weights, the hidden layer weights are fixed and vice versa. We are now ready to propose the MRAGD algorithm to update the estimated weight vectors $V(k)$ and $W(k)$. Note that multivariate systems can operate in $\mathbb{R}^N$ and complex domain [109] or quaternion domain [110, 111]. In this thesis, we only discuss the case in $\mathbb{R}^N$.

3.2 Output Layer Training and Stability Analysis

To update the output layer weight $V(k)$ for MIMO RNN, the form of the MRAGD algorithm can be given as

$$V(k+1) = V(k) - \frac{\alpha^v(k+1)}{\rho^v(k+1)} \frac{dE(k)}{dV(k)} \quad (3.10)$$

where $\alpha^v(k+1)$ is adaptive learning rate based on the dead zone concept and $\rho^v(k)$ is adaptive normalization factor. The two parameters will be defined later.

$$\begin{aligned} \frac{dE(k)}{dV(k)} &= \frac{1}{2} \left[\frac{\partial (e^T(k)e(k))}{\partial V(k)} + \beta^v(k+1) \frac{\partial (e^T(k)e(k))}{\partial \Phi^T(k)} \frac{\partial \Phi^T(k-1)}{\partial V(k)} \right] \\ &= -e(k) [\Phi^T(k) + \beta^v(k+1) \bar{V}(k) D_v(k) W^T(k) \Phi'(k)] \\ &= -e(k) [\Phi^T(k) + \beta^v(k+1) A(k)]. \end{aligned} \quad (3.11)$$

The second partial derivative on the right side of (3.11) reveals the recurrent nature of RNNs training in that the history of past outputs is incorporated into the MRAGD learning algorithm, which is similar to a classical RTRL algorithm in terms that the past derivative is involved in the RNNs training. However, it creates new problems on weight convergence and stability of the system which may lead to slow transient speed and/or even instability. To overcome these difficulties, the adaptive learning rate, dead zone and recurrent hybrid parameter concepts are proposed in the follows

to speed up the RNNs learning procedure with guaranteed weight convergence.

The key notations appeared in (3.10) and (3.11) are as follows.

1. The extended recurrent gradient is defined as

$$A(k) = \bar{V}(k)D_v(k)W^T(k)\Phi'(k) \in \mathbb{R}^{1 \times n_h}. \quad (3.12)$$

2. The long vector of $V(k-1)$ is

$$\bar{V}(k) = [V_{1,1}(k) \dots V_{h,1}(k) \dots V_{1,m}(k) \dots V_{h,m}(k)] \in \mathbb{R}^{1 \times (n_h \times n_I)}. \quad (3.13)$$

3. Jacobian Matrix $D_v(k)$ is defined as

$$\begin{aligned} & D_v(k-1) \\ = & \begin{bmatrix} \frac{u_1}{\partial \bar{V}(k-1)} & \dots & \frac{u_m}{\partial \bar{V}(k-1)} & \frac{\partial y^T(k-d)}{\partial \bar{V}(k-1)} & \dots & \frac{\partial y^T(k-2)}{\partial \bar{V}(k-1)} \end{bmatrix} \\ \approx & \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0} & \frac{\partial y^T(k-d)}{\partial \bar{V}(k-d)} & \dots & \frac{\partial y^T(k-2)}{\partial \bar{V}(k-2)} \end{bmatrix} \\ \in & \mathbb{R}^{(n_h \times m) \times n_I}. \end{aligned} \quad (3.14)$$

4. The diagonal matrix $\Phi'(k-1)$ is

$$\begin{aligned} \Phi'(k) &= \text{diag}[\Phi'_1(k) \Phi'_2(k) \dots \Phi'_m(k)] \\ &= \text{diag}[h'|_{W_{1,:}(k)x(k)} \ h'|_{W_{2,:}(k)x(k)} \dots h'|_{W_{m,:}(k)x(k)}]. \end{aligned} \quad (3.15)$$

5. The recurrent hybrid adaptive parameter is determined by

$$\begin{cases} \beta^v(k+1) = 1, & \text{if } A(k)[\delta I + \Phi(k)\Phi^T(k)]^{-1}\Phi(k) \geq 0 \\ \beta^v(k+1) = 0, & \text{if } A(k)[\delta I + \Phi(k)\Phi^T(k)]^{-1}\Phi(k) < 0. \end{cases} \quad (3.16)$$

6. $\alpha^v(k+1)$ is an adaptive learning rate similar to the dead zone approach in the

classical neural network and adaptive control systems, given by

$$\begin{cases} \alpha^v(k+1) = 1, & \text{if } \|e(k)\| \geq \varepsilon_{\max}^V / \sqrt{1 - \|\Phi^T(k) + \beta^v(k)A(k)\|^2 / \rho^v(k+1)} \\ \alpha^v(k+1) = 0, & \text{if } \|e(k)\| < \varepsilon_{\max}^V / \sqrt{1 - \|\Phi^T(k) + \beta^v(k)A(k)\|^2 / \rho^v(k+1)} \end{cases} \quad (3.17)$$

where $\varepsilon_{\max}^V$ is the maximum value of the bounded equivalent disturbance of the output layer defined in equation (3.20).

7. $\rho^v(k)$ is the normalization factor defined by

$$\rho^v(k+1) = v\rho^v(k) + \max\{\bar{\rho}^v, 1 + \|\Phi^T(k) + \beta^V(k+1)A(k)\|^2\} \quad (3.18)$$

where $0 < v < 1$ and $\bar{\rho}^v$ are positive constants.

Remark 1. *Historically, RNNs are complex nonlinear systems, the evaluated study must be take proper assumptions which is different from the linear system approach. Though MRAGD only gets approximate gradient for realtime learning, we use three adaptive parameters defined in (3.16), (3.17) and (3.18) to guarantee the weight convergence and L_2 -stability of the learning as shown in Theorem 3.1 later such that the the estimated error is bounded. When the MIMO extension is involved, the output layer weights must be extended to a long vector as (3.13) as required by the partial derivative rules : $\frac{\partial \hat{y}^T(k)}{\partial \Phi^T(k)} = \bar{V}(k)$ and $\frac{\partial \Phi^T(k)}{\partial V(k)} = D_v(k)W^T(k)\Phi'(k)$. The approximation of the extended recurrent gradient $D_v(k)$ in (3.14) is inspired from the idea of RTRL training concept by assuming that the weights do not vary during a specific time interval. Thus, the partial derivatives between the output and the weights can be obtained in a recursive fashion which provides an affordable computation load and is suitable for online adaptive control system.*

For the stability and convergence analysis, approximation error of the RNN as de-

defined in (3.5) will be extended as

$$\begin{aligned}
e(k) &= y(k) - \hat{y}(k) + \varepsilon(k) \\
&= V^* \Phi^*(k) - V(k) \Phi(k) + \varepsilon(k) \\
&= V^* \Phi^*(k) - V^* \Phi(k) - [V(k) \Phi(k) - V^* \Phi(k)] + \varepsilon(k) \\
&= \varepsilon^V(k) - e_V(k)
\end{aligned} \tag{3.19}$$

where the bounded equivalent disturbance of the output layer is defined as

$$\varepsilon^V(k) = [\varepsilon_1^V(k) \dots \varepsilon_h^V(k)]^T = \varepsilon(k) + V^* \Phi^*(k) - V^* \Phi(k) \tag{3.20}$$

and the posterior estimation error is defined as

$$e_V(k) = V(k) \Phi(k) - V^* \Phi(k) = \tilde{V}(k) \Phi(k) \tag{3.21}$$

and $\tilde{V}(k) = V(k) - V^*$.

Theorem 3.1. *If the output layer of the MIMO RNN is trained by the adaptive normalized gradient algorithm (3.11), the weight $V(k)$ is guaranteed to be convergent in the sense of Lyapunov function $\left(\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \right) \leq 0 \forall k$ with $\tilde{V}(k) = V(k) - V^*$ and the training error will be L_2 -stable in the sense of $e_V \in L_2$ and $\lim_{k \rightarrow \infty} \sup \|e(k)\| \leq \varepsilon_{\max}^V / \sqrt{1 - \Delta_{vmax}} = \Delta_v$, where*

$$\Delta_{vmax} = \max \left(\left\| \Phi^T(k) + \beta^v(k+1) A(k) \right\|^2 / \rho^v(k) \right) < 1 \forall k.$$

Proof. For simplicity of notations, we ignore the time subscripts of $\alpha^v(k+1)$, $\beta^v(k+1)$ and $\rho^v(k+1)$. According to the MRAGD training algorithm (3.10) of the output layer and using the matrix trace property $tr \{AB\} = tr \{BA\}$, if B^T and A have the same dimension,

$$\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2$$

$$\begin{aligned}
&= \text{tr}\{2\alpha^v \rho^v e(k) (\Phi^T(k) + \beta^v A(k)) \tilde{V}^T(k) \\
&\quad + (\alpha^v)^2 \|\Phi^T(k) + \beta^v A(k)\|^2 \|e(k)\|^2\} / (\rho^v)^2 \\
&= \text{tr}\{2\alpha^v \rho^v (\Phi^T(k) + \beta^v A(k)) \\
&\quad \times (\Phi(k)\Phi^T(k))^\dagger \Phi(k)\Phi^T(k) \tilde{V}^T(k) e(k) \\
&\quad + (\alpha^v)^2 \|\Phi^T(k) + \beta^v A(k)\|^2 \|e(k)\|^2\} / (\rho^v)^2 \tag{3.22} \\
&= [2\alpha^v \rho^v e^T(k) \tilde{V}(k) \Phi(k) (1 + \beta^v A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k)) \\
&\quad + \|\Phi^T(k) + \beta^v A(k)\|^2 (\alpha^v)^2 \|e(k)\|^2] / (\rho^v)^2 \\
&= [2\alpha^v \rho^v e^T(k) e_V(k) (1 + \beta^v A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k)) \\
&\quad + \|\Phi^T(k) + \beta^v A(k)\|^2 (\alpha^v)^2 \|e(k)\|^2] / (\rho^v)^2.
\end{aligned}$$

Note that $\tilde{V}(k)\Phi(k)$ is replaced with $e_V(k)$ as defined in equation (3.21). $(\Phi(k)\Phi^T(k))^\dagger$ is the pseudoinverse of $\Phi(k)\Phi^T(k)$. Multiply both sides of (3.19) by $e^T(k)$ and replace $e^T(k)e_V(k)$ with $e^T(k)e_V(k) = e^T(k)\varepsilon^V(k) - \|e(k)\|^2$, we can rewrite equation (3.22) as

$$\begin{aligned}
&\|\tilde{V}(k+1)\|_F^2 - \|\tilde{V}(k)\|_F^2 \\
&= [2\alpha^v \rho^v (e^T(k)\varepsilon^V(k) - \|e(k)\|^2) \\
&\quad \times (1 + \beta^v A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k)) \\
&\quad + \|\Phi^T(k) + \beta^v A(k)\|^2 (\alpha^v)^2 \|e(k)\|^2] / (\rho^v)^2 \\
&\leq [\alpha^v \rho^v \{(\|\varepsilon^V(k)\|^2 + \|e(k)\|^2) \\
&\quad \times (1 + \beta^v A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k)) \\
&\quad - 2\|e(k)\|^2 (1 + \beta^v A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k))\} \\
&\quad + \|\Phi^T(k) + \beta^v A(k)\|^2 (\alpha^v)^2 \|e(k)\|^2] / (\rho^v)^2. \tag{3.23}
\end{aligned}$$

The above inequality is based on $2e^T(k)\varepsilon^V(k) \leq \|\varepsilon^V(k)\|^2 + \|e(k)\|^2$. By the definitions of the hybrid adaptive learning rate $\beta^v(k)$ in (3.16) and adaptive learning rate

$\alpha^v(k)$ in (3.17), we finally have

$$\begin{aligned}
& \left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \\
& \leq - \left(1 + \beta^v A(k) \left(\Phi(k) \Phi^T(k) \right)^\dagger \Phi(k) \right) \frac{\alpha^v}{\rho^v} \\
& \quad \times \left(- \left\| \varepsilon^V \right\|^2 + \left(1 - \frac{\left\| \Phi^T(k) + \beta^v A(k) \right\|^2}{\rho^v} \right) \left\| e(k) \right\|^2 \right) \\
& \leq 0.
\end{aligned} \tag{3.24}$$

Since $\beta^v A(k) \left(\Phi(k) \Phi^T(k) \right)^\dagger \Phi(k) \geq 0$ due to the definition in (3.16), it can be ignored from (3.24). Note also that

$$\rho^v > \left\| \Phi^T(k) + \beta^v A(k) \right\|^2$$

as defined in (3.18), we have

$$\left\| \Phi^T(k) + \beta^v A(k) \right\|^2 / \rho^v < 1.$$

The adaptive learning rate defined in (3.17) implies that the recurrent training of MRAGD will be active only if the output error of RNN is bigger than the predefined value, which guarantees $\left(\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \right)$ is bounded below zero and is not increasing. The limit of $\left(\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \right)$ therefore exists and

$$\left(- \left\| \varepsilon^V(k) \right\|^2 + \left(1 - \frac{\left\| \Phi^T(k) + \beta^v A(k) \right\|^2}{\rho^v} \right) \left\| e(k) \right\|^2 \right)$$

will go to zero, then we can get

$$\lim_{k \rightarrow \infty} \sup \|e(k)\| \leq \varepsilon_{\max}^V / \sqrt{1 - \Delta_{vmax}} = \Delta_v. \tag{3.25}$$

This completes the proof. ■

Remark 2. According to the theoretical analysis, the three parameters $\beta^v(k+1)$,

$\alpha^v(k+1)$ and $\rho^v(k+1)$ play a key role for the proof of weight convergence and stability analysis. They are designed to guarantee the negativeness of Lyapunov function as in (3.24). The hybrid learning rate $\beta^v(k+1)$ defined in equation (3.16) can be interpreted as activating the recurrent learning model in case of $A(k) (\Phi(k)\Phi^T(k))^\dagger \Phi(k) \geq 0$. Otherwise, the training procedure will be the same as a static BP algorithm. Detailed explanations for the functions of the three parameters are presented in [39]. From the closed-loop control point of view, the use of the adaptive learning rate $\alpha^v(k+1)$ is critical as stated in [112]. If a fixed learning rate $\alpha^v \ll 1$ is used, the network output may converge to a mapping point, which will be very different from the desired mapping $y(k)$.

3.3 Hidden Layer Training and Stability Analysis

For hidden layer training, we have

$$W(k+1) = W(k) - \frac{\alpha^w(k+1)}{\rho^w(k+1)} \frac{dE(k)}{dW(k)}. \quad (3.26)$$

The gradient function of the hidden layer is

$$\begin{aligned} \nabla E(k) &= \frac{dE(k)}{dW(k)} \\ &= \frac{1}{2} \left[\frac{\partial(e^T(k)e(k))}{\partial W(k)} + \beta^w(k+1) \frac{\partial(e^T(k)e(k))}{\partial x(k)} \frac{\partial x(k)}{\partial W(k)} \right] \\ &= -\Phi'(k)V^T(k)e(k)[x^T(k) + \beta^w(k+1)\bar{W}(k)D_w(k)] \\ &= -\Phi'(k)V^T(k)e(k)[x^T(k) + \beta^w(k+1)B(k)]. \end{aligned} \quad (3.27)$$

The key notions are as follows.

1. The partial derivative term $B(k)$ is

$$B(k) = \bar{W}(k)D_w(k) = [B_1(k) \dots B_n(k)] \in \mathbb{R}^{n_I}. \quad (3.28)$$

2. The long vector of $W(k-1)$ is

$$\bar{W}(k) = [W_{1,:}(k) \ W_{2,:}(k) \ \dots \ W_{m,:}(k)] \in \mathbb{R}^{1 \times (n_h \times n_I)}. \quad (3.29)$$

3. The recurrent gradient (Jacobian matrix) for the hidden layer is

$$D_w(k-1) = [D_w^1(k-1) \ D_w^2(k-1) \ \dots \ D_w^m(k-1)]^T \in \mathbb{R}^{(n_h \times n_I) \times n_I} \quad (3.30)$$

where

$$\begin{aligned} & D_w^j(k-1) \\ &= \begin{bmatrix} \frac{\partial(y^d(k-d))^T}{\partial W_{j,:}(k-1)} & \cdots & \frac{\partial(y^d(k-1))^T}{\partial W_{j,:}(k-1)} & \frac{\partial y^T(k-d)}{\partial W_{j,:}(k-1)} & \cdots & \frac{\partial y^T(k-2)}{\partial W_{j,:}(k-1)} \end{bmatrix} \\ &\approx \begin{bmatrix} \frac{\partial(y^d(k-d))^T}{\partial W_{j,:}(k-1)} & \cdots & \frac{\partial(y^d(k-1))^T}{\partial W_{j,:}(k-1)} & \frac{\partial y^T(k-d)}{\partial W_{j,:}(k-d)} & \cdots & \frac{\partial y^T(k-2)}{\partial W_{j,:}(k-2)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial x_1(k-1)}{\partial W_{j,:}(k-1)} & \cdots & \frac{\partial x_{h \times d}(k-1)}{\partial W_{j,:}(k-1)} & \frac{\partial x_{h \times d+1}(k-1)}{\partial W_{j,:}(k-d)} & \cdots & \frac{\partial x_n(k-1)}{\partial W_{j,:}(k-2)} \end{bmatrix} \\ &\in \mathbb{R}^{n_I \times n_I}, \ 1 \leq j \leq m. \end{aligned} \quad (3.31)$$

4. The hybrid adaptive learning rate of the hidden layer $\beta^w(k)$ is defined as

$$\begin{cases} \beta^w(k+1) = 1, & \text{if } B(k)(\delta I + x(k)x^T(k))^{-1}x(k) \geq 0 \\ \beta^w(k+1) = 0, & \text{if } B(k)(\delta I + x(k)x^T(k))^{-1}x(k) < 0. \end{cases} \quad (3.32)$$

5. The adaptive dead zone learning rate of hidden layer is defined as

$$\begin{cases} \alpha^w(k+1) = 1, & \text{if } \|e(k)\| \geq \varepsilon_{\max}^W / \varpi(k) \\ \alpha^w(k+1) = 0, & \text{if } \|e(k)\| < \varepsilon_{\max}^W / \varpi(k) \end{cases} \quad (3.33)$$

where $\varpi(k) = \sqrt{1 - \frac{\|\Phi'(k)V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2}{\rho^w(k)h'_{\min}(k)}}$ and $\varepsilon_{\max}^W$ is the maximum value of the bounded equivalent disturbance of the hidden layer defined in equation

(3.37) and $h'_{\min}(k)$ is the smallest value of the derivatives of the nonlinear activation function.

6. The normalization factor is given by

$$\begin{aligned} \rho^w(k+1) &= v\rho^w(k) \\ &+ \max\{\bar{\rho}^w, 1 + \frac{\|\Phi'(k)V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2}{h'_{\min}(k)}\} \end{aligned} \quad (3.34)$$

where $\bar{\rho}^w$ is a positive constant.

For the stability and convergence analysis, the approximation error can be extended as

$$\begin{aligned} e(k) &= y(k) - \hat{y}(k) + \varepsilon(k) \\ &= V^* \Phi^*(k) - V(k) \Phi(k) + \varepsilon(k) \\ &= V^* \Phi^*(k) - V(k) \Phi(W^* x(k-1)) \\ &\quad + V(k) \Phi(W^* x(k)) - V(k) \Phi(W(k)x(k)) + \varepsilon(k) \\ &= V(k) \Phi(W^* x(k)) - V(k) \Phi(W(k)x(k)) + \varepsilon^W(k) \\ &= -e_W(k) + \varepsilon^W(k) \end{aligned} \quad (3.35)$$

where $e_W(k)$ and $\varepsilon^W(k)$ are posterior error and bounded equivalent disturbance of the hidden layer respectively. According to the mean value theory, the posterior error of the hidden layer $e_W(k)$ can be extended as

$$\begin{aligned} e_W(k) &= V(k) \Phi(W(k)x(k)) - V(k) \Phi(W^* x(k)) \\ &= V_{:,1}(k) \tilde{W}_{1,:}(k) \mu_1(k) x(k) + V_{:,2}(k) \tilde{W}_{2,:}(k) \\ &\quad \times \mu_2(k) x(k) + \cdots + V_{:,m}(k) \tilde{W}_{m,:}(k) \mu_m(k) x(k) \\ &= \sum_{j=1}^m V_{:,j}(k) \tilde{W}_{j,:}(k) \mu_j(k) x(k) \end{aligned} \quad (3.36)$$

with $\tilde{W}_{j,:}(k) = W_{j,:}(k) - W_{j,:}^*(k) \in \mathbb{R}^{1 \times n}$, which is the vector difference between the j th row of the estimated $W(k)$ and ideal weight W^* , and $0 < \mu_j(k) \leq 1 \ \forall k$ being

the mean value of the j th nonlinear activation function, and

$$\varepsilon^W(k) = V^* \Phi^*(k) - V(k) \Phi(W^* x(k)) + \varepsilon(k) \quad (3.37)$$

$$\|e(k)\|^2 = -e^T(k) e_W(k) + e^T(k) \varepsilon^W(k). \quad (3.38)$$

Theorem 3.2. *The weight matrix $W(k)$ is guaranteed to be convergent in the sense of Lyapunov function ($\|\tilde{W}(k+1)\|_F^2 - \|\tilde{W}(k)\|_F^2 \leq 0$) and the hidden layer training of MRAGD will be L_2 -stable in the sense of $e_W \in L_2$ with $\tilde{W}(k) = W(k) - W^*$, where W^* represents an ideal weight matrix of a local minimum point of the gradient and $\lim_{k \rightarrow \infty} \sup \|e(k)\| \leq \varepsilon_{\max}^W / \sqrt{1 - \Delta_{w\max}} = \Delta_w$, where $\Delta_{w\max} = \max \left(\frac{\|\Phi'(k) V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2}{\rho^w(k) h'_{\min}(k)} \right) \forall k$.*

Proof. For simplicity of notations, we ignore the time subscripts of $\alpha^w(k+1)$, $\beta^w(k+1)$ and $\rho^w(k+1)$.

$$\begin{aligned} & - \text{tr}\{\nabla E(k) \tilde{W}^T(k)\} \\ &= \text{tr}\left\{ \Phi'(k) V^T(k) e(k) \left[x^T(k) + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) x^T(k) \right] \tilde{W}^T(k) \right\} \\ &= \text{tr}\left\{ \left[x^T(k) + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) x^T(k) \right] \tilde{W}^T(k) \Phi'(k) V^T(k) e(k) \right\} \\ &= x^T(k) \tilde{W}^T(k) \Phi'(k) V^T(k) e(k) \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \\ &= \left\{ \sum_{j=1}^m [\Phi'_j(k) x^T(k) \tilde{W}_{j,:}^T(k) V_{:,j}^T(k) e(k)] \right\} \\ & \quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \\ &= \left\{ \sum_{j=1}^m \frac{\Phi'_j(k)}{\mu_j(k)} [x^T(k) \tilde{W}_{j,:}^T(k) \mu_j(k) V_{:,j}^T(k) e(k)] \right\} \\ & \quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \\ &= \left\{ \sum_{j=1}^m \frac{\Phi'_j(k)}{\mu_j(k)} [x^T(k) \tilde{W}_{j,:}^T(k) \mu_j(k) V_{:,j}^T(k) e(k)] \right\} \\ & \quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \end{aligned}$$

$$\begin{aligned}
&\leq \left\{ \frac{h'_{\min}(k)}{\mu_{\max}(k)} \sum_{j=1}^m [x^T(k) \tilde{W}_{j,:}^T(k) \mu_j(k) V_{:,j}^T(k) e(k)] \right\} \\
&\quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \quad (3.39) \\
&= \left\{ \frac{h'_{\min}(k)}{\mu_{\max}(k)} [e_W^T(k) e(k)] \right\} \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right)
\end{aligned}$$

where $\mu_{\max}(k) = 1 \forall k$ is the maximum value of the activation function. The inequality in equation (3.39) is due to the fact

$$\begin{aligned}
-\nabla E_j(k) \tilde{W}_{j,:}^T(k) &= V_{:,j}^T(k) e(k) \Phi'_j(k) x^T(k) \tilde{W}_{j,:}^T(k) \\
&\quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \leq 0 \forall k
\end{aligned}$$

for the attractor basin of a local minimum point [39].

Substituting W^* and then squaring both sides of equation (3.26) and using the trace property, we get

$$\begin{aligned}
&\left\| \tilde{W}(k+1) \right\|_F^2 - \left\| \tilde{W}(k) \right\|_F^2 \\
&= [2 \text{tr} \{ \nabla E(k) \tilde{W}^T(k) \} \alpha^w \rho^w \\
&\quad + \left\| \Phi'(k) V^T(k) e(k) (x^T(k) + \beta^w B(k)) \right\|_F^2 (\alpha^w)^2] / (\rho^w)^2 \\
&\leq [2 \left\{ h'_{\min}(k) [e_W^T(k) e(k)] \right\} \\
&\quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \alpha^w \rho^w \\
&\quad + \left\| \Phi'(k) V^T(k) \right\|^2 \left\| x^T(k) + \beta^w B(k) \right\|^2 \|e(k)\|^2 (\alpha^w)^2] / (\rho^w)^2 \\
&= [2 \left\{ h'_{\min}(k-1) [e^T(k) \varepsilon^W(k) - \|e(k)\|^2] \right\} \\
&\quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \alpha^w \rho^w \\
&\quad + \left\| \Phi'(k) V^T(k) \right\|^2 \left\| x^T(k) + \beta^w B(k) \right\|^2 \|e(k)\|^2 (\alpha^w)^2] / (\rho^w)^2 \\
&\leq \left\{ h'_{\min}(k-1) [\|\varepsilon^W(k)\|^2 - \|e(k)\|^2] \right\} \\
&\quad \times \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right) \alpha^w \rho^w \\
&\quad + \left\| \Phi'(k) V^T(k) \right\|^2 \left\| x^T(k) + \beta^w B(k) \right\|^2 \|e(k)\|^2 (\alpha^w)^2] / (\rho^w)^2 \\
&\leq \frac{\alpha^w}{\rho^w} h'_{\min}(k) \left(1 + \beta^w B(k) (x(k) x^T(k))^{\dagger} x(k) \right)
\end{aligned}$$

$$\begin{aligned} & \times \left(\|\varepsilon^W\|^2 - \left(1 - \frac{\|\Phi'(k)V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2}{\rho^w h'_{\min}(k)}\right) \|e(k)\|^2 \right) \\ & \leq 0. \end{aligned} \quad (3.40)$$

Since $\left(1 + \beta^w B(k) (x(k)x^T(k))^{\dagger} x(k)\right) > 0$ due to the definition of β^w in (3.32), it can be ignored from (3.40). The definition of ρ^w in (3.34) is to guarantee

$$\frac{\|\Phi'(k)V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2 / h'_{\min}(k)}{\rho^w} < 1.$$

α^w defined in (3.33) implies that the recurrent training of MRAGD will be active only if the output error of RNN is bigger than the predefined value, which guarantees $(\|\tilde{V}(k+1)\|_F^2 - \|\tilde{V}(k)\|_F^2)$ is bounded below zero and is not increasing. The limit of $(\|\tilde{V}(k+1)\|_F^2 - \|\tilde{V}(k)\|_F^2)$ therefore exists and

$$(\|\varepsilon^W(k)\|^2 - \left(1 - \frac{\|\Phi'(k)V^T(k)\|^2 \|x^T(k) + \beta^w B(k)\|^2}{\rho^w h'_{\min}(k)}\right) \|e(k)\|^2)$$

will go to zero. Then we have

$$\lim_{k \rightarrow \infty} \sup \|e(k)\| \leq \varepsilon_{\max}^W / \sqrt{1 - \Delta_{wmax}} = \Delta_w. \quad (3.41)$$

This completes the proof. ■

3.4 Application of MRAGD in Tracking Control and Stability Analysis

Consider a class of dynamical nonlinear plants, which can be represented as an I/O form as follows:

$$y^d(k) = f^*(k-1) + u(k-1) + \varepsilon(k) \quad (3.42)$$

where $y^d(k) \in \mathbb{R}^m$ is the system output vector, $u(k-1) \in \mathbb{R}^m$ is the control signal vector and $f^*(k-1) \in \mathbb{R}^m$ is the inherent nonlinear part of the dynamical plants. $\varepsilon(k) \in \mathbb{R}^m$ denotes a noise vector of the nonlinear plant with the bound $\|\varepsilon(k)\| \leq \varepsilon_m$. Given a desired trajectory $y(k) \in \mathbb{R}^m$, a control scheme is proposed as shown in Figure 3.1. The tracking error for the control system can be defined as

$$s(k) = y^d(k) - y(k). \quad (3.43)$$

Define the control signal $u(k-1)$ as

$$u(k-1) = -\hat{y}(k-1) + y(k) + k_v s(k-1) \quad (3.44)$$

where k_v is the gain parameter of the fixed proportional controller. For the RNN, $\hat{y}(k-1) = [\hat{y}_1(k-1) \dots \hat{y}_h(k-1)]^T \in \mathbb{R}^m$ is the estimated output vector which is used to approximate the nonlinear function $f^*(k-1)$ and the input vector for the RNN is written as

$$\begin{aligned} x(k-1) &= [(y^d(k-d))^T \dots (y^d(k-1))^T \dots \hat{y}^T(k-d) \dots \hat{y}^T(k-2)]^T \\ &= [x_1(k-1) \dots x_n(k-1)]^T \in \mathbb{R}^{n_I} \end{aligned} \quad (3.45)$$

where $n_I = 2 \times m \times d$ and d is the maximum time delay factor of output recurrent feedback of the network.

Then, the estimation error is given by

$$\bar{e}(k) = y(k-1) - \hat{y}(k-1) + \tau(k) = e(k-1) \quad (3.46)$$

where $\tau(k)$ is a bounded reconstruction error vector that includes all the uncertainties during estimation [113]. In the closed-loop control system, this estimation error may not be directly measurable, however, it can be generated by combining (3.42), (3.43), (3.44) and (3.46), so that

$$\bar{e}(k) = (1 - z^{-1}k_v)s(k) - \zeta(k) \quad (3.47)$$

For a neural network controller as shown in the inner loop of Figure 3.1, $\hat{y}(k-1)$ in the last estimation and its tapped delays are fed back into the network. Thus, the estimate error $\bar{e}(k)$ and its derivatives are not only dependent on the current feedback signal and weight, but also the past ones. Obviously, the recurrent time dependence is essential to be considered in the error derivative of RNNs learning algorithm to capture system dynamics. However, most of existing works of the so-called RNNs based nonlinear control employ only the standard backpropagation training method or its minor variations to train NNs [1, 46, 49, 51, 53, 54], in which the tapped delay feedback signal is indeed fed back into the NN and it looks like RNNs based training but the second partial derivative term in (3.11) was never considered. Ignorance of the inherent recurrent property of nonlinear dynamic systems in the RNNs weight updating procedure similar to (3.11) in most of the theoretical researches of NNs based controllers is partially due to the fact that it is often too complicated to prove weight convergence and stability of the closed-loop control system. The proposed MRAGD algorithm (3.10) uses the specifically designed derivative (3.11) which is a recursive training algorithm suitable for online adaptive control.

Corollary 3.1. *Assume that the maximum singular value $k_{v \max}$ of k_v is smaller than one, which implies that the estimate error $\bar{e}(k)$ can be generated from a fixed stable filter from the tracking error $s(k)$ as in (3.47), the tracking error $s(k)$ will converge to a compact set, i.e. $\lim_{k \rightarrow \infty} \|s(k)\| \leq \Delta_s$, where*

$$\Delta_s > 2 \frac{\|k_v\| (\Delta_m + \zeta_m)}{(1 - k_{v \max}^2)} \quad (3.48)$$

is a positive constant with $\Delta_m = \min[\Delta_v, \Delta_w]$ and $\zeta_m = \max(\|\zeta(k)\|)$.

Proof: Define the Lyapunov function $\|s(k)\|^2$ and use (3.47) we have

$$\begin{aligned} & s^T(k+1)s(k+1) - s^T(k)s(k) \\ &= -s^T(k)[1 - k_v^T k_v]s(k) + \|\bar{e}(k)\|^2 + \|\zeta(k)\|^2 \\ & \quad + 2\zeta^T(k)\bar{e}(k) + 2s^T(k)k_v\bar{e}(k) + 2s^T(k)k_v\zeta(k) \\ &\leq -(1 - k_{v \max}^2)\|s(k)\|^2 + \Delta_m^2 + \zeta_m^2 + 2\zeta_m\Delta_m \\ & \quad + 2\|k_v\|\|s(k)\|\Delta_m + 2\|k_v\|\|s(k)\|\zeta_m \\ &= -\left[(1 - k_{v \max}^2)\left(\|s(k)\| - \frac{\|k_v\|(\Delta_m + \zeta_m)}{(1 - k_{v \max}^2)}\right)^2\right. \\ & \quad \left.- ((1 - k_{v \max}^2) + \|k_v\|^2) \frac{(\Delta_m + \zeta_m)^2}{(1 - k_{v \max}^2)}\right]. \end{aligned} \quad (3.49)$$

If we select $k_{v \max} < 1$, the L.H.S of (3.49) will be no-increasing, whenever

$$(1 - k_{v \max}^2) \left(\|s(k)\| - \frac{\|k_v\|(\Delta_m + \zeta_m)}{(1 - k_{v \max}^2)} \right)^2 \geq ((1 - k_{v \max}^2) + \|k_v\|^2) \frac{(\Delta_m + \zeta_m)^2}{(1 - k_{v \max}^2)}. \quad (3.50)$$

and (3.50) state that the control tracking error $s(k)$ is bounded as soon as it falls into a ball around zero. This is similar to the process in [114]. To make this point

clear, we relax (3.50) to have a weaker bound:

$$\begin{aligned} \|s(k)\|^2 - 2\|s(k)\| \frac{\|k_v\| (\Delta_m + \zeta_m)}{(1 - k_{vmax}^2)} + \frac{\|k_v\|^2 (\Delta_m + \zeta_m)^2}{(1 - k_{vmax}^2)^2} \\ \geq \frac{\|k_v\|^2 (\Delta_m + \zeta_m)^2}{(1 - k_{vmax}^2)^2} \Rightarrow \|s(k)\| \geq 2 \frac{\|k_v\| (\Delta_m + \zeta_m)}{(1 - k_{vmax}^2)}. \end{aligned} \quad (3.51)$$

The conclusion from (3.49) and (3.51) is that, given any

$$\Delta_s > 2 \frac{\|k_v\| (\Delta_m + \zeta_m)}{(1 - k_{vmax}^2)} \quad (3.52)$$

$\lim_{k \rightarrow \infty} \sup \|s(k)\| \leq \Delta_s$ will be a positive invariant set. This completes the proof.

Summary of the MRAGD algorithm for neural controller:

1. Initialize the weights for the RNN;
2. Form new input $x(k-1)$ as defined in equation (3.45);
3. Perform forward computation and calculate the output $\hat{y}(k-1)$ of the neural network: use the input state $x(k-1)$ and the updated weights of the network;
4. Calculate the control input $u(k-1)$ based on equation (3.44);
5. The new measurement of the system dynamics is taken and the measurable tracking error $s(k)$ is used to get the training error $\bar{e}(k)$ through (3.47);
6. Calculates the weights $V(k-1)$ and $W(k-1)$ via the learning law (3.10) and (3.26) for the output and hidden layers, respectively, for the next iteration;
7. Go back to step 2 to continue the iteration until $s(k)$ reached to the allowable level.

3.5 Experimental Verification

A two-link direct drive robot model with its I/O discrete-time version as shown in (3.53) is obtained from the SCARA robot model [115] based on Euler's method.

$$\begin{bmatrix} q(k+1) \\ \dot{q}(k+1) \end{bmatrix} = \begin{bmatrix} q(k) + T\dot{q}(k) \\ \dot{q}(k) - TM^{-1}(q(k)) [V(q(k), \dot{q}(k)) + F(\dot{q}(k))] \end{bmatrix} + \begin{bmatrix} 0 \\ u(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \tau(k+1) \end{bmatrix} \quad (3.53)$$

where $q(k+1) = [q_1(k+1) \ q_2(k+1)]^T$ and $\dot{q}(k+1) = [\dot{q}_1(k+1) \ \dot{q}_2(k+1)]^T$ are two-dimensional joint angle and velocity vectors, respectively. T is the sampling period, $u(k)$ is the control signal, $\tau(k+1)$ is normally distributed random numbers with zero mean and variance is 0.1 and the configuration dependent inertia matrix, the centrifugal and coriolis effect, and the Coulomb friction are, respectively, given as shown in the equation below:

$$M(q(k)) = \begin{bmatrix} 3.32 + 0.32\cos(q_2(k)) & 0.12 + 0.16\cos(q_2(k)) \\ 0.12 + 0.16\cos(q_2(k)) & 0.12 \end{bmatrix}$$

$$V(q(k), \dot{q}(k)) = \begin{bmatrix} -\dot{q}_2(k)(2\dot{q}_1(k) + \dot{q}_2(k))0.16\sin(q_2(k)) \\ 0.16\dot{q}_1^2\sin(q_2(k)) \end{bmatrix}$$

$$F(\dot{q}(k)) = \begin{bmatrix} 5.3\text{sgn}(\dot{q}_1(k)) \\ 1.1\text{sgn}(\dot{q}_2(k)) \end{bmatrix}.$$

The robot output vector for the tracking control system in Figure 3.1 is defined as $y^d(k) = \dot{q}(k)$. The MIMO nonlinear function for the RNN identification is $f^*(k) = V^*\Phi(W^*, x(k)) = \dot{q}(k) - TM^{-1}(q(k)) [V(q(k), \dot{q}(k)) + F(\dot{q}(k))]$ as defined in (3.42) and the joint angle $q(k) = [q_1(k) \ q_2(k)]^T$ can be obtained from the simulation model (3.53) by replacing $f^*(k)$ with the RNN output $y(k) = V(k)\Phi(W(k), x(k))$ as defined in (3.1).

A three-layered RNN defined in Section 3.4 is used for the simulation, which has 20

hidden layer neurons and two output neurons. We compare the proposed MRAGD algorithm to both the standard Backpropagation (BP) and RTRL based RNNs control systems, respectively, all based on the same control structure as shown in Figure 3.1. We choose a small fixed learning rate 0.01 for the RTRL and BP based controllers to guarantee stability in the presence of noise. The desired joint angle velocity trajectories for the two joints are square waveforms and the sampling period is $T = 0.002s$. The proportional gain parameter of the fixed controller is given as $k_v = \text{diag}[0.5, 0.5]$ for all the controllers.

We perform the Monte Carlo simulations for 100 times with the same initial conditions for different algorithms and each batch consists of 2000 steps. Figure 3.2 shows the tracking control results using BP, RTRL and MRAGD algorithms which are compared with the desired tracking trajectories. The squared instant tracking errors of the initial responses of the two joints using the three neural controllers are presented in Figure 3.3, which are plotted inside the ellipse circles in Figure 3.2. As shown in Figure 3.3, it takes about 40 steps and 200 steps for the tracking error of the two joints to reach steady state using RTRL and BP controller respectively. However, it only takes about 40 steps for joint one and 20 steps for joint two to reach steady state using MRAGD controller. Therefore, MRAGD controller has better transient response and is faster than BP and RTRL controllers in terms of the transient response. The different transient response is because that BP and RTRL do not include the second partial derivatives similar to that of (3.11) as in MRAGD. Furthermore, BP and RTRL cannot guarantee weight convergence during training, so the learning rate must be small enough to avoid divergence, which result in the slow transient response. However, MRAGD use three adaptive learning rate to guarantee weight convergence during training, weights are only updated when the convergence conditions are satisfied. Therefore, MRAGD can use much larger learning rate. Percentages of 1 and 0 in the four adaptive parameters $\alpha^v(k)$, $\alpha^w(k)$, $\beta^v(k)$, $\beta^w(k)$ over the 2000 training steps are listed in Table 3.1 to show effectiveness of the proposed adaptive learning rates. To provide a clear comparison among the steady state responses using the three algorithms, the squared instant tracking

Table 3.1: Percentages of 1 and 0 in the Adaptive Parameters over the 2000 Training Steps.

	$\alpha^v(k+1)$	$\beta^v(k)$	$\alpha^w(k)$	$\beta^w(k)$
1	0.9857	0.6643	0.9861	0.2283
0	0.0143	0.3357	0.0139	0.7717

errors from the step 200 to 2000 of the two joints are given in Figure 3.4. The traces of normalization factors $\rho^v(k)$ and $\rho^w(k)$ of MRAGD are shown in Figure 3.5. Table 3.2 shows the mean squared error (MSE) and the standard deviation (Std) for each dimension of the tracking results, which are respectively defined as:

$$MSE_i \triangleq \frac{1}{n} \sum_{k=1}^n (s_i(k))^2$$

$$Std_i \triangleq \left(\frac{1}{n} \sum_{k=1}^n (s_i(k) - \bar{s}_i)^2 \right)^{1/2}$$

where $\bar{s}_i \triangleq \frac{1}{n} \sum_{k=1}^n s_i(k)$ and $i = 1, 2$ is the dimension index. We can see that the steady state tracking error with MRAGD controller is the smallest for both of the joints as compared to that of RTRL and BP.

Table 3.2: Comparations of the Tracking Statistics of Steady-state between MRAGD and RTRL and BP.

	MRAGD	RTRL	BP
MSE_1	$1.9616e^{-4}$	$2.5329e^{-4}$	$2.1e^{-3}$
MSE_2	$2.5597e^{-4}$	$6.4334e^{-4}$	$8.5e^{-3}$
Std_1	$1.4e^{-2}$	$1.58e^{-2}$	$3.97e^{-2}$
Std_2	$1.58e^{-2}$	$2.48e^{-2}$	$7.22e^{-2}$

3.6 Summary

In this chapter, the MRAGD training method for RNN is proposed and the conditions for weight convergence and stability are performed. The RNN is used in the closed-loop system to estimate online unknown nonlinear system function. A multivariate robust adaptive gradient-descent training algorithm is developed to train

(a) Output $y^d(k)$ and reference signal $y(k)$ using RTRL based neural controller.

(b) Output $y^d(k)$ and reference signal $y(k)$ using BP based neural controller.

(c) Output $y^d(k)$ and reference signal $y(k)$ using MRAGD based neural controller.

Figure 3.2: Output $y^d(k)$ and reference signal $y(k)$ using RTRL and MRAGD and BP based neural controller.

Figure 3.3: Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ (initial response in Figure 3.2) from the step 1 to 200 using RTRL and MRAGD and BP training algorithms.

Figure 3.4: Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ from the step 200 to 2000 using RTRL and MRAGD and BP training algorithms.

Figure 3.5: Full traces of normalization factors $\rho^v(k)$ and $\rho^w(k)$.

the RNN. The weight convergence and system stability are proven in the sense of Lyapunov function. A complete stability analysis is performed and it is proved that the tracking error will be bounded under the neural controller in the presence of disturbance. Simulation results are presented for a two-link robot tracking control problem which show that the proposed MRAGD based neural controller performed better than a neural controller based on the standard BP algorithm and RTRL algorithm.

Chapter 4

RNN Training Algorithm Based on SPSA

To maximize potentials and capabilities of the RNNs, one of the preliminary tasks is to design an efficient training algorithm with fast weight convergence speed and low computational complexity. Gradient-descent optimization techniques are often used for NNs which consist of adjustable weights along the negative direction of error gradient. However, due to the dynamic structure of RNNs, the obtained error gradient tends to be very small. From computational efficiency and weight convergence points of view, Spall proposed a simultaneous perturbation stochastic approximation (SPSA) method to find roots and minima of neural network functions which are contaminated with noise in [40]. The name of “simultaneous perturbation” arises from the fact that all elements of unknown weight parameters are being varied simultaneously. SPSA has the potential to be significantly more efficient than the usual p -dimensional algorithms (of Kiefer-Wolfowitz/Blum type) that are based on standard finite-difference gradient approximations and it is shown in [40] that approximately the same level of estimation accuracy can typically be achieved with only $1/p$ th the amount of data needed in the standard approach. The main advantage of SPSA is its simplicity and excellent weight convergence property, in which only two objective function measurements are used. Thus, SPSA is more economical in terms of loss function evaluation which is usually the most expensive

part of an optimization problem. SPSA has attracted great attention in application areas such as neural networks training, adaptive control and model parameter estimation [4, 116–120].

In this chapter, a robust recurrent simultaneous perturbation stochastic approximation (RRSPSA) method is applied to approximate the gradient of the cost function following deterministic approach for RNNs training. And the convergence property and stability of the weight estimate of the proposed RRSPSA training algorithm for RNNs are also addressed. We do not require the weights to converge to the ideal values exactly and we use the fixed dead zone concept as the MRAGD training algorithm as shown in Chapter 3.

4.1 Basics of the RNN and SPSA

To simplify notation of this paper, we consider a three-layer m -external input and m -output RNN with the output feedback structure as shown in Fig. 4.1, which can be easily extended into a RNN with arbitrary numbers of inputs and outputs. Output vector of the RNN is $\hat{y}(k)$ which can be presented as

$$\hat{y}(k) = H(\bar{W}(k), x(k)) \bar{V}(k) = [\hat{y}_1(k), \dots, \hat{y}_m(k)]^T \in \mathbb{R}^m \quad (4.1)$$

where $\bar{V}(k) \in \mathbb{R}^{p^v}$ ($p^v = m \times n_h$ and n_h being the dimension of hidden layer) is the long vector of the output layer weight matrix $\hat{V}(k) \in \mathbb{R}^{n_h \times m}$ defined by

$$\bar{V}(k) = [\hat{V}_{1,1}(k), \dots, \hat{V}_{n_h,1}(k), \dots, \hat{V}_{1,m}(k), \dots, \hat{V}_{n_h,m}(k)]^T \quad (4.2)$$

and $x(k) \in \mathbb{R}^{n_I}$ ($n_I = (l+1) \times m$) is the input vector of the RNN defined by

$$\begin{aligned} x(k) &= [x_1(k), x_2(k), \dots, x_{n_I}(k)]^T \\ &= [\hat{y}^T(k-1), \dots, \hat{y}^T(k-l), u_1(k), \dots, u_m(k)]^T \end{aligned} \quad (4.3)$$

and $\bar{W}(k) \in \mathbb{R}^{p^w}$ ($p^w = n_I \times n_h$) is the long vector of the hidden layer weight matrix $\hat{W}(k) \in \mathbb{R}^{n_h \times n_I}$ defined by

$$\bar{W}(k) = [\hat{W}_{1,:}(k), \hat{W}_{2,:}(k), \dots, \hat{W}_{n_h,:}(k)]. \quad (4.4)$$

$H(\bar{W}(k), x(k)) \in \mathbb{R}^{m \times p^v}$ is the nonlinear activation block-diagonal matrix given by

$$H(k) = H(\bar{W}(k), x(k)) = \begin{bmatrix} h_1(k), \dots, h_{n_h}(k), & 0, \dots & 0, \dots \\ 0, \dots & \ddots & \vdots \\ \vdots & & \\ 0, \dots & 0, \dots & h_1(k), \dots, h_{n_h}(k) \end{bmatrix} \quad (4.5)$$

where $h_j(k)$ with $1 \leq j \leq n_h$ is one of the most popular nonlinear activation functions,

$$h_j(k) = h(\hat{W}_{j,:}(k)x(k)) = \frac{1}{1 + \exp(-4\lambda\hat{W}_{j,:}(k)x(k))} \quad (4.6)$$

with $\hat{W}_{j,:}(k) \in \mathbb{R}^{n_I}$ being the j th row vector of $\hat{W}(k)$ and $\lambda > 0$ is the gain parameter of the threshold function that is defined specifically for ease of use later when deriving the sector condition of the hidden layer.

The output vector $\hat{y}(k)$ is an approximation of the ideal nonlinear function $y(k) = H(\bar{W}^*, x(k))\bar{V}^*$ with $\bar{V}^* \in \mathbb{R}^{p^v}$ and $\bar{W}^* \in \mathbb{R}^{p^w}$ being the ideal output layer and hidden layer weight respectively.

SPSA algorithm is to solve the problem of finding an optimal weight θ^* of the gradient equation of neural networks

$$\hat{g}(\hat{\theta}(k)) = \frac{\partial L(\hat{\theta}(k))}{\partial \hat{\theta}(k)} = 0$$

for some differentiable loss function $L : \mathbb{R}^p \rightarrow \mathbb{R}^1$, where $\hat{\theta}(k)$ is the estimated weight vector of neural networks. The basic idea is to vary all elements of the vector $\hat{\theta}(k)$ simultaneously and approximate the gradient function using only two measurements of the loss function (4.7) and (4.8) without requiring exact derivatives

Figure 4.1: Block diagram of RRSPSA training for the output feedback based RNN.

or a large number of function evaluation which provides significant improvement in efficiency over the standard gradient decent algorithms. Define

$$\hat{g}^+(\hat{\theta}(k)) = L(\hat{\theta}(k) + c(k)\Delta(k)) + \epsilon^+(k) \quad (4.7)$$

$$\hat{g}^-(\hat{\theta}(k)) = L(\hat{\theta}(k) - c(k)\Delta(k)) + \epsilon^-(k) \quad (4.8)$$

where $\epsilon^+(k)$ and $\epsilon^-(k)$ represent measurement noise terms, $c(k)$ is a scalar and $\Delta(k) = [\Delta_1(k), \dots, \Delta_p(k)]$ is the perturbation vector. Then, the form of the estimation of $g(\hat{\theta})$ at the k th iteration is

$$\hat{g}(\hat{\theta}(k)) = \left[\frac{\hat{g}^+(\hat{\theta}(k)) - \hat{g}^-(\hat{\theta}(k))}{2c(k)\Delta_1(k)} \quad \dots \quad \frac{\hat{g}^+(\hat{\theta}(k)) - \hat{g}^-(\hat{\theta}(k))}{2c(k)\Delta_p(k)} \right]^T \quad (4.9)$$

Theory and numerical experience in [40] indicate that SPSA can be significantly more efficient than the standard finite difference-based algorithms in large-dimensional problems.

Inspired by the excellent economical and convergence properties of SPSA, we propose the RRSPSA algorithm under the framework of deterministic system based on the idea of simultaneous perturbation with guaranteed weight convergence. The loss

function for RRSPSA is defined as

$$L(\hat{\theta}(k)) = \frac{1}{2} \|e(k)\|^2 \quad (4.10)$$

$$e(k) = y(k) - \hat{y}(k) + \varepsilon(k) \quad (4.11)$$

where $y(k)$ is the ideal output and a function of θ^* , $\hat{y}(k)$ is a function of $\hat{\theta}(k)$, and $\varepsilon(k)$ is a disturbance term. Similar to SPSA, the RRSPSA algorithm seeks to find an optimal weight vector θ^* of gradient equation, i.e., the weight vector that minimized differentiable $L(\hat{\theta}(k))$. That is, the proposed RRSPSA algorithm for updating $\hat{\theta}(k) \in \mathbb{R}^{(p^w+p^v)}$ as an estimation of the ideal weight vector θ^* is of the form

$$\hat{\theta}(k+1) = \hat{\theta}(k) - \alpha(k+1)\hat{g}(\hat{\theta}(k)) \quad (4.12)$$

where $\alpha(k+1)$ is an adaptive learning rate and $\hat{g}(\hat{\theta}(k))$ is an approximation of the gradient of the loss function. This approximated gradient function (normalized) is of the SPSA form

$$\begin{aligned} & \hat{g}(\hat{\theta}(k)) \\ &= \frac{L\left(\hat{\theta}(k) + c(k+1)\Delta(k+1)\right) - L\left(\hat{\theta}(k) - c(k+1)\Delta(k+1)\right) + \delta(k)}{2c(k+1)\rho(k+1)} r(k+1) \end{aligned} \quad (4.13)$$

where $\delta(k)$ is an equivalent approximation error of the loss function, which is also called the measurement noise, and $\rho(+1)$ is the normalization factor. $\Delta(k+1)$, $r(k+1)$, and $c(k+1) > 0$ are the controlling parameters of the algorithm defined as

1. The perturbation vector

$$\Delta(k+1) = [\Delta_1(k+1), \dots, \Delta_{p^w+p^v}(k+1)]^T \in \mathbb{R}^{(p^w+p^v)} \quad (4.14)$$

is a bounded random directional vector that is used to perturb the weight

vector simultaneously and can be generated randomly.

2. The sequence of $r(k+1)$ is defined as

$$r(k+1) = \left[\frac{1}{\Delta_1(k+1)}, \dots, \frac{1}{\Delta_{p^w+p^v}(k+1)} \right]^T \in \mathbb{R}^{(p^w+p^v)}. \quad (4.15)$$

3. $c(k+1) > 0$ is a sequence of positive numbers [40, 118, 119]. $c(k+1)$ can be chosen as a small constant number for a slow time-varying system as pointed in [121] and it controls the size of perturbations.

Note that for simplicity of notations, we ignore the time subscripts of several parameters where no confusion is caused such as $\Delta(k+1)$, $r(k+1)$, $c(k+1)$, $\delta(k)$ and $\alpha(k+1)$.

4.2 Output Layer Training and Stability Analysis

We are now ready to propose the RRSPSA algorithm to update the estimated weight vector $\bar{V}(k)$ of output layer. That is,

$$\bar{V}(k+1) = \bar{V}(k) - \alpha^v(k+1) \hat{g}^v(\bar{V}(k)) \quad (4.16)$$

where $\hat{g}^v(\bar{V}(k)) \in \mathbb{R}^{p^v}$ is the normalized gradient approximation that uses simultaneous perturbation vectors $\Delta^v(k+1) \in \mathbb{R}^{p^v}$ and $r^v(k+1) \in \mathbb{R}^{p^v}$ to stimulate weight. $\Delta^v(k+1)$ and $r^v(k+1)$ are, respectively, the first p^v components of perturbation vectors defined in (4.14) and (4.15).

$$\hat{g}^v(\bar{V}(k)) = -e^T(k)H(k)\Delta^v r^v (\rho^v)^{-1} - \beta^v(k+1)e^T(k)A(k)r^v (2c\rho^v(k+1))^{-1} \quad (4.17)$$

where $A(k) \in \mathbb{R}^m$ is the extended recurrent gradient defined as

$$\begin{aligned} A(k) = & H\left(\bar{W}(k), \hat{D}^v(k) (\bar{V}(k) + c\Delta^v)\right) \bar{V}(k) \\ & - H\left(\bar{W}(k), \hat{D}^v(k) (\bar{V}(k) - c\Delta^v)\right) \bar{V}(k) \end{aligned} \quad (4.18)$$

$$\hat{D}^v(k) = \left[H^T(k-1), \dots, H^T(k-l), 0, \dots, 0 \right]^T. \quad (4.19)$$

For simplicity of notations, we ignore the time subscripts of $\Delta^v(k+1)$, $r^v(k+1)$, $\beta^v(k+1)$, $\rho^v(k+1)$ and $\alpha^v(k+1)$ in the equations.

Remark 3. *Our purpose is to find the desired output layer weight vector $\bar{V}^*$ of the gradient equation*

$$\hat{g}(\bar{V}(k)) = \frac{\partial L(\bar{V}(k))}{\partial \bar{V}(k)} = 0.$$

Inspired by the excellent properties of SPSSA, we obtain $\hat{g}(\bar{V}(k))$ by varying all elements of unknown weight parameters simultaneously as shown in the proof of Lemma 4.1 under the framework of deterministic system. We will show that RRSPSSA has the same form as SPSSA, and only two objective function measurements are used at each iteration, which maintains the efficiency of SPSSA. However, the weight convergence proof and stability analysis are quite different from those in [40] under the framework of deterministic system as shown in Theorem 4.1 and 4.2. For RNNs, the outputs depend on not only the weights at current time steps, but also those at the previous time steps since the current input vector contains the delayed outputs. The first term on the right side of equation in (4.17) is the result of perturbation of the feedforward signals in RNNs and $A(k)$ in (4.18) is the result of perturbation of the recurrent signals which makes the training complicated and the system easy to be unstable. In addition to the simplicity and efficiency of RRSPSSA, one powerful aspect of the developed RRSPSSA algorithm is that it is able to use effectively large adaptive learning rate $\alpha^v(k+1)$ to accelerate weight convergence speed with guaranteed system stability. To avoid weight divergence at each time step, an adaptive learning rate $\beta^v(k+1)$ is used in (4.17) to control the recurrent signals and switch off the recurrent connections whenever the convergence condition is violated as shown in (4.41). The normalization factor $\rho^v(k+1)$ is also used to bound the signals in learning algorithms as traditionally did in adaptive control system [4, 112]. $\alpha^v(k+1)$ is to guarantee the weight convergence during training, which means the weights are only updated when the convergence and stability requirements are met and it does

not need to be small as in RTRL training algorithm. The sufficient conditions for the three key parameters are shown in (4.37), (4.40), and (4.41). $\hat{D}^v(k)$ is obtained during the mathematical deduction in the proof of Lemma 4.1.

Lemma 4.1. *The normalized gradient approximation in (4.17) is an equivalent presentation of the standard SPSA algorithm in equation (4.13).*

Proof. We will prove that (4.17) is obtained by perturbing all the weights simultaneously as SPSA did. Rewrite the loss function defined in (4.10) as

$$L(\hat{\theta}(k)) = L(\bar{V}(k), \bar{W}(k)) = \frac{1}{2} \|y(k) - \hat{y}(k) + \varepsilon(k)\|^2. \quad (4.20)$$

Further, we use $\hat{y}(k) = H(k)\bar{V}(k)$ in (4.1) to define

$$\begin{aligned} & \hat{y}^{v+}(\bar{V}(k)) \\ &= H(\bar{V}(k) + c\Delta^v) + H(\bar{W}, (x(k) + \Delta x(k+1))) \bar{V}(k) \\ &= H(\bar{V}(k) + c\Delta^v) \\ & \quad + H\left(\bar{W}, \left([\hat{y}^T(k-1), \dots, \hat{y}^T(k-l), 0, \dots, 0]^T + \Delta x(k+1)\right)\right) \bar{V}(k) \\ &= H(\bar{V}(k) + c\Delta^v) + H\left(\bar{W}, \left([\bar{V}^T(k-1)H^T(k-1), \dots, \right. \right. \\ & \quad \left. \left. \bar{V}^T(k-l)H^T(k-l), 0, \dots, 0]^T + \Delta x(k+1)\right)\right) \bar{V}(k) \\ &= H(\bar{V}(k) + c\Delta^v) + H\left(\bar{W}, \left[(\bar{V}(k-1) + c\Delta^v)^T H^T(k-1), \right. \right. \\ & \quad \left. \left. \dots, (\bar{V}(k-l) + c\Delta^v)^T H^T(k-l), 0, \dots, 0]^T\right)\right) \bar{V}(k). \end{aligned} \quad (4.21)$$

$\hat{y}^{v+}(\bar{V}(k))$ has the same motivation as $g^+(\hat{\theta}(k))$ of SPSA shown as in (4.7) which is to perturb all the weights simultaneously. Different from FNN training which only perturb the current weight, the input vector must be perturbed as well for RNN training because of the time dependence in RNNs. The disturbance $\Delta x(k+1)$ is to perturb the weights at previous time steps which are contained in $x(k)$ as seen in (4.3). However, the last equation of (4.21) is difficult to implement. Here

we assume the model parameters do not change apparently between each iteration, i.e. $\bar{V}(k) \approx \bar{V}(k-1) \cdots \approx \bar{V}(k-l)$. We can derive a similar approach as RTRL introduced by Williams and Zipser [12] makes the proposed algorithm real time and adaptive in a recursive fashion. Under the approximation,

$$\begin{aligned}
& \hat{y}^{v+}(\bar{V}(k)) \\
& \approx H(k)(\bar{V}(k) + c\Delta^v) + H\left(\bar{W}(k), \left[(\bar{V}(k) + c\Delta^v)^T H^T(k-1), \right. \right. \\
& \quad \left. \left. \cdots, (\bar{V}(k) + c\Delta^v)^T H^T(k-l), 0, \cdots, 0\right]^T\right) \bar{V}(k) \\
& = H(k)(\bar{V}(k) + c\Delta^v) + H\left(\bar{W}(k), \left[H^T(k-1), \cdots, H^T(k-l), 0, \cdots, 0\right]^T \right. \\
& \quad \left. \times (\bar{V}(k) + c\Delta^v)\right) \bar{V}(k) \\
& = H(k)(\bar{V}(k) + c\Delta^v) + H\left(\bar{W}(k), \hat{D}^v(k)(\bar{V}(k) + c\Delta^v)\right) \bar{V}(k)
\end{aligned} \tag{4.22}$$

where $\hat{D}^v(k)$ is defined in (4.19). By the approximation $\bar{V}(k) \approx \bar{V}(k-1) \cdots \approx \bar{V}(k-l)$, the recurrent term $H^T(k-1), \cdots, H^T(k-l)$ of (4.19) can be iteratively obtained from the previous steps to speed up the calculation of RRSPSA (normally, the input $u(k)$ is independent from weight so that the last entries of $\hat{D}^v(k)$ are zero vectors). Weight convergence and system stability can be guaranteed by using the three adaptive learning rates which will be explained detailly later.

Similar to (4.21), we can get

$$\begin{aligned}
\hat{y}^{v-}(\bar{V}(k)) &= H(k)(\bar{V}(k) - c\Delta^v) + H(\bar{W}(k), (x(k) - \Delta x(k+1))) \bar{V}(k) \\
&\approx H(k)(\bar{V}(k) - c\Delta^v) + H\left(\bar{W}(k), \hat{D}^v(k)(\bar{V}(k) - c\Delta^v)\right) \bar{V}(k)
\end{aligned} \tag{4.23}$$

so that

$$\hat{y}^{v+}(\bar{V}(k)) - \hat{y}^{v-}(\bar{V}(k)) \approx (\bar{V}(k)) 2H(k)c\Delta^v + A(k) \tag{4.24}$$

where the extended recurrent gradient $A(k)$ is defined as (4.18). $A(k)$ in (4.24) is the result of recurrent connections. To guarantee weight convergence and system

stability during training, we insert an adaptive recurrent hybrid parameter $\beta^v(k+1)$ (see Fig. 4.1) to control the recurrent training signal. Thus, (4.24) can be rewritten as follows

$$\hat{y}^{v+}(\bar{V}(k)) - \hat{y}^{v-}(\bar{V}(k)) = 2H(k)c\Delta^v + \beta^v(k+1)A(k). \quad (4.25)$$

Note that we change the approximation in equation (4.24) to equal is because we use the formula on the right hand side of the equation to update the weight during learning.

According to (4.13), we can get

$$\begin{aligned} & \hat{g}^v(\bar{V}(k)) \\ &= \frac{L(\bar{W}(k), \bar{V}(k) + c\Delta^v) - L(\bar{W}(k), \bar{V}(k) - c\Delta^v) + \delta^v(k)}{2c\rho^v} r^v \\ &= \frac{\|y(k) - \hat{y}^{v+}(\bar{V}(k))\|^2 - \|y(k) - \hat{y}^{v-}(\bar{V}(k))\|^2 + 2\delta^v(k)}{4c\rho^v} r^v \\ &= \left\{ (y(k) - \hat{y}^{v+}(\bar{V}(k)) + y(k) - \hat{y}^{v-}(\bar{V}(k)))^T \right. \\ & \quad \times (\hat{y}^{v-}(\bar{V}(k)) - \hat{y}^{v+}(\bar{V}(k))) + 2\delta^v(k) \Big\} r^v (4c\rho^v)^{-1} \\ &= (y(k) - \hat{y}(k) + \varepsilon(k))^T (\hat{y}^{v-}(\bar{V}(k)) - \hat{y}^{v+}(\bar{V}(k))) r^v (2c\rho^v)^{-1} \\ &= e^T(k) (\hat{y}^{v-}(\bar{V}(k)) - \hat{y}^{v+}(\bar{V}(k))) r^v (2c\rho^v)^{-1} \\ &= -e^T(k)H(k)\Delta^v r^v (\rho^v)^{-1} - \beta^v e^T(k)A(k)r^v (2c\rho^v)^{-1}. \end{aligned} \quad (4.26)$$

$e(k)$ in the fifth equality of (4.26) comes from (4.11). The fourth equality of (4.26) reveals the relationship between gradient approximation error $\delta^v(k)$ and system disturbance $\varepsilon(k)$, i.e.,

$$\begin{aligned} \delta^v(k) &= (-2\hat{y}(k) + \hat{y}^{v+}(\bar{V}(k)) + \hat{y}^{v-}(\bar{V}(k)) + \varepsilon(k))^T \\ & \quad \times (\hat{y}^{v-}(\bar{V}(k)) - \hat{y}^{v+}(\bar{V}(k))) \\ &= \varepsilon^T(k) (\hat{y}^{v-}(\bar{V}(k)) - \hat{y}^{v+}(\bar{V}(k))). \end{aligned} \quad (4.27)$$

In the fifth equality of (4.26), the gradient approximation presentation appears exactly the same as the original one in [40] (see [40, eq. (2.2)]). Only two objective function measurements are used at each step which maintains the efficiency of SPSA.

■

The training error $e(k)$ comes from the weight estimation error and the disturbance. For stability analysis, $e(k)$ will be linked to the weight estimating error of the output layer and the disturbance. Define the weight estimating errors of the output layer and hidden layer as

$$\tilde{\bar{V}}(k) = \bar{V}^* - \bar{V}(k) \quad (4.28)$$

$$\tilde{\bar{W}}(k) = \bar{W}^* - \bar{W}(k) \quad (4.29)$$

where $\bar{W}^*$ and $\bar{V}^*$ are the ideal (optimal) weight vectors of $\bar{W}(k)$ and $\bar{V}(k)$. According to (4.11), the training error can be rewritten as

$$\begin{aligned} e(k) &= y(k) - \hat{y}(k) + \varepsilon(k) \\ &= H(\bar{W}^*, x(k)) \bar{V}^* - H(k) \bar{V}(k) + \varepsilon(k) \\ &= H(\bar{W}^*, x(k)) \bar{V}^* - H(k) \bar{V}^* + H(k) \bar{V}^* - H(k) \bar{V}(k) + \varepsilon(k). \end{aligned} \quad (4.30)$$

Because the term $H(\bar{W}^*, x(k)) \bar{V}^* - H(k) \bar{V}^*$ is temporarily bounded during output layer training (hidden layer weights are fixed), we define

$$\begin{aligned} \tilde{e}^v(k) &= H(\bar{W}^*, x(k)) \bar{V}^* - H(k) \bar{V}^* + \varepsilon(k) \\ &= \tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) \bar{V}^* + \varepsilon(k) \end{aligned} \quad (4.31)$$

with $\tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) = H(\bar{W}^*, x(k)) - H(k)$. The training error caused by the weight estimation error of the output layer is defined as

$$\phi^v(k) = -H(k) \tilde{\bar{V}}(k). \quad (4.32)$$

$$e(k) = -\phi^v(k) + \tilde{e}^v(k). \quad (4.33)$$

Hence, the training error is directly linked to the weight estimating error of the output layer contained in $\phi^v(k)$ and the disturbance $\tilde{e}^v(k)$. This is critical for the convergence proof.

Theorem 4.1. *If the output layer of the RNN is trained by RRSPSA algorithm (4.16), $\bar{V}(k)$ is guaranteed to be convergent in the sense of Lyapunov function $\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \leq 0$, $\forall k$ with $\tilde{V}(k) = \bar{V}^* - \bar{V}(k)$ and the training error will be L_2 -stable in the sense of $\tilde{e}^v(k) \in L_2$ and $\phi^v(k) \in L_2$.*

Proof. According to the training algorithm in (4.16), we can get

$$\tilde{V}(k+1) = \tilde{V}(k) + \alpha^v \hat{g}^v(\bar{V}(k)) \quad (4.34)$$

where $\hat{g}^v(\bar{V}(k))$ is defined in (4.26), then we have

$$\begin{aligned} & \left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \\ &= 2\alpha^v \hat{g}^v(\bar{V}(k))^T \tilde{V}(k) + \left\| \alpha^v \hat{g}^v(\bar{V}(k)) \right\|^2 \\ &= -\alpha^v \frac{e^T(k) (2H(k) c \Delta^v + \beta^v A(k))}{c\rho^v} (r^v)^T \tilde{V}(k) + \left\| \alpha^v \hat{g}^v(\bar{V}(k)) \right\|^2 \\ &= -\alpha^v \frac{2c \left\{ e^T(k) H(k) \Delta^v (r^v)^T \tilde{V}(k) \right\} + \left\{ \beta^v e^T(k) A(k) (r^v)^T \tilde{V}(k) \right\}}{c\rho^v} \\ & \quad + \left\| \alpha^v \hat{g}^v(\bar{V}(k)) \right\|^2 \\ &= -2\alpha^v \left\{ e^T(k) H(k) \tilde{V}(k) \right\} \left\{ \tilde{V}^T(k) \left(\tilde{V}(k) \tilde{V}^T(k) \right)^\dagger \Delta^v (r^v)^T \tilde{V}(k) \right\} (\rho^v)^{-1} \\ & \quad - \alpha^v \left\{ \beta^v e^T(k) H(k) \tilde{V}(k) \left(H(k) \tilde{V}(k) \right)^T \left(H(k) \tilde{V}(k) \left(H(k) \tilde{V}(k) \right)^T \right)^\dagger \right. \\ & \quad \times \left. A(k) (r^v)^T \left(H^T(k) H(k) \right)^\dagger H^T(k) H(k) \tilde{V}(k) \right\} (c\rho^v)^{-1} + \left\| \alpha^v \hat{g}^v(\bar{V}(k)) \right\|^2 \\ &= 2\alpha^v p^v \left\{ e^T(k) \phi^v(k) \right\} (\rho^v)^{-1} \\ & \quad + \alpha^v \beta^v \left\{ e^T(k) \phi^v(k) \right\} \text{trace} \left\{ \left(H(k) \tilde{V}(k) \right)^T \left(H(k) \tilde{V}(k) \left(H(k) \tilde{V}(k) \right)^T \right)^\dagger A(k) (r^v)^T \right. \\ & \quad \times \left. \left(H^T(k) H(k) \right)^\dagger H^T(k) H(k) \tilde{V}(k) \right\} (c\rho^v)^{-1} + \left\| \alpha^v \hat{g}^v(\bar{V}(k)) \right\|^2 \end{aligned}$$

$$\begin{aligned}
&= 2\alpha^v p^v \{e^T(k)\phi^v(k)\} (\rho^v)^{-1} + \alpha^v \beta^v \{e^T(k)\phi^v(k)\} \\
&\quad \times \text{trace} \left\{ A(k) (r^v)^T (H^T(k)H(k))^\dagger H^T(k)H(k)\tilde{V}(k) \left(H(k)\tilde{V}(k) \right)^T \right. \\
&\quad \times \left. \left(H(k)\tilde{V}(k) \left(H(k)\tilde{V}(k) \right)^T \right)^\dagger \right\} (c\rho^v)^{-1} + \|\alpha^v \hat{g}^v(\bar{V}(k))\|^2 \\
&= 2\alpha^v p^v \{e^T(k)\phi^v(k)\} (\rho^v)^{-1} + \alpha^v \{e^T(k)\phi^v(k)\} \beta^v (r^v)^T \\
&\quad \times (H^T(k)H(k))^\dagger H^T(k)A(k) (c\rho^v)^{-1} + \|\alpha^v \hat{g}^v(\bar{V}(k))\|^2
\end{aligned} \tag{4.35}$$

According to (4.33), $\phi^v(k)$ in the last equation of (4.35) can be replaced with $\tilde{e}^v(k) - e(k)$.

$$\begin{aligned}
&\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \\
&\leq 2\alpha^v p^v \{e^T(k) (\tilde{e}^v(k) - e(k))\} (\rho^v)^{-1} + \alpha^v \{e^T(k) (\tilde{e}^v(k) - e(k))\} \beta^v (r^v)^T \\
&\quad \times (H^T(k)H(k))^\dagger H^T(k)A(k) (c\rho^v)^{-1} \\
&\quad + \|e(k)\|^2 \left\| \alpha^v (2H(k)c\Delta^v + \beta^v A(k)) \right\|^2 \|r^v\|^2 (2c\rho^v)^2 \\
&\leq \alpha^v p^v (\rho^v)^{-1} \{ \|\tilde{e}^v(k)\|^2 - \|e(k)\|^2 \} + \alpha^v \{ \|\tilde{e}^v(k)\|^2 - \|e(k)\|^2 \} \beta^v (r^v)^T \\
&\quad \times (H^T(k)H(k))^\dagger H^T(k)A(k) (2c\rho^v)^{-1} \\
&\quad + \|e(k)\|^2 \left\| \alpha^v (2H(k)c\Delta^v + \beta^v A(k)) \right\|^2 \frac{\|r^v\|^2}{(2c\rho^v)^2} \\
&= \alpha^v (\rho^v)^{-1} \{p^v + 0.5\beta^v Y(k)\} \|\tilde{e}^v(k)\|^2 \\
&\quad - \alpha^v (\rho^v)^{-1} \left\{ p^v + 0.5\beta^v Y(k) - \alpha^v (\rho^v)^{-1} Z(k) \right\} \|e(k)\|^2 \\
&\leq \alpha^v (\rho^v)^{-1} \{p^v + 0.5\beta^v Y(k)\} \left\{ \|\tilde{e}^v(k)\|^2 - \left(1 - \frac{\alpha^v \rho(k+1)^{-1} Z(k)}{p^v + 0.5\beta^v Y(k)} \right) \|e(k)\|^2 \right\}
\end{aligned} \tag{4.36}$$

where $Y(k)$ and $Z(k)$ are defined in (4.38) and (4.39), respectively. Note also that the second inequality in (4.36) is from the fact that $2e^T(k)\tilde{e}^v(k) \leq \|\tilde{e}^v(k)\|^2 + \|e(k)\|^2$. p^v is the dimension of the output layer weight vector which is positive. $\beta^v Y(k) \geq 0$ due to the definition of recurrent adaptive learning rate $\beta^v(k+1)$ in (4.41) and $\rho^v > \frac{\alpha^v Z(k)}{p^v + 0.5\beta^v Y(k)}$ due to definition of the normalization factor in (4.37). The adaptive learning rate $\alpha^v(k+1)$ in (4.40) implies that recurrent training of RRSPSA will be

α^v (positive scalar) only if the output error of RNN is bigger than a predefined value otherwise it will be zero, which guarantees $\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2$ is bounded below zero and is not increasing. The limit of $\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2$ therefore exists and $\left\{ \left\| \tilde{e}^v(k) \right\|^2 - \left(1 - \frac{\alpha^v \rho(k+1)^{-1} Z(k)}{p^v + 0.5 Y(k)} \right) \left\| e(k) \right\|^2 \right\}$ will go to zero. Then, we finally have

$$\left\| \tilde{V}(k+1) \right\|_F^2 - \left\| \tilde{V}(k) \right\|_F^2 \leq 0.$$

■

The conditions for the three adaptive learning rates in (4.16) and (4.17) are as follows. The detailed procedure to obtain these conditions is shown in the above proof of Theorem 4.1.

1. $\rho^v(k+1)$ is the normalization factor defined by

$$\rho^v(k+1) = \tau \rho^v(k) + \max \left\{ \frac{\alpha^v Z(k)}{p^v + 0.5 \beta^v Y(k)} + 1, \bar{\rho} \right\} \quad (4.37)$$

where $0 < \tau < 1$, $0 < \alpha^v \leq 1$, and $\bar{\rho}$ are positive constants and

$$Y(k) = \frac{(r^v)^T (H^T(k) H(k))^{\dagger} H^T(k) A(k)}{c} \quad (4.38)$$

where η is a small perturbation parameter.

$$Z(k) = \|(2H(k)c\Delta^v + \beta^v A(k))\|^2 \|r^v\|^2 (4(c)^2)^{-1}. \quad (4.39)$$

2. $\alpha^v(k+1)$ is an adaptive learning rate similar to dead zone approach in the classical neural network and adaptive control

$$\begin{cases} \alpha^v(k+1) = \alpha^v, & \text{if } \|e(k)\| \geq \Lambda(k) \\ \alpha^v(k+1) = 0, & \text{if } \|e(k)\| < \Lambda(k) \end{cases} \quad (4.40)$$

with $\Lambda(k) = \tilde{e}_m^v(k) / \sqrt{1 - \frac{\alpha^v (\rho^v)^{-1} Z(k)}{p^v + 0.5 \beta^v Y(k)}}$ and $\tilde{e}_m^v(k)$ being the maximum value of

$\tilde{e}^v(k)$ defined in (4.31).

3. $\beta^v(k+1)$ is the recurrent hybrid adaptive parameter determined by

$$\begin{cases} \beta^v(k+1) = 1, & \text{if } Y(k) \geq 0 \\ \beta^v(k+1) = 0, & \text{if } Y(k) < 0. \end{cases} \quad (4.41)$$

Remark 4. According to the theoretical analysis, three parameters $\beta^v(k+1)$, $\alpha^v(k+1)$ and $\rho^v(k+1)$ play a key role in the proof of weight convergence and stability analysis. They are designed to guarantee the negativeness of Lyapunov function. The function of $\beta^v(k+1)$ is to automatically switch off the important recurrent training signal in the lower left corner of Fig. 4.1 when the convergence and stability requirements are violated resulted from recurrent training signals. The normalization factor $\rho^v(k+1)$ is also used to bound the signals in learning algorithms as traditionally did in adaptive control system [4, 112]. $\alpha^v(k+1)$ is to guarantee weight convergence during training, which means that the weights are only updated when convergence and stability requirements are met. Besides, $\alpha^v(k+1)$ does not need to be small as in RTRL training algorithm which will make the training by RRSPSA faster than that by RTRL. From the definition of $\alpha^v(k+1)$ in (4.40), it can be seen that the weights will be updated only if the training error is bigger than the dead zone value which means the learning will be stopped if the training error is lower than the dead zone level. This is to provide good generalization other than to learn the system noise. In this paper, we will choose a suitable value for $\tilde{e}_m^v(k)$ according to the noise level.

Remark 5. Perturbation size $c(k+1)$ is a critical variable as discussed in the original SPSSA papers [4, 40, 118]. A natural question is whether $c(k+1)$ will affect the learning procedure in the recurrent learning algorithm. The answer is yes. It is not obvious to see from the updating formula of $\hat{g}^v(\bar{V}(k))$ in (4.17). However, this can be easily seen by expanding the second term $A(k)$ in (4.17) to be explicit in $c(k)$. According to the mean value theory and that activation function in (4.6) is a non-decreasing function, it can be readily shown that there exist unique positive mean

values $\mu_j(k)$ such that

$$h\left(\hat{W}_{j,:}(k)x_1(k)\right) - h\left(\hat{W}_{j,:}(k)x_2(k)\right) = \mu_j(k)\hat{W}_{j,:}(k)(x_1(k) - x_2(k)) \quad (4.42)$$

where $\hat{W}_{j,:}(k)$ is the estimated weight vector linked to the j th hidden layer neurons. From (4.42) and the matrix in (4.5), $A(k)$ in (4.18) can be further extended as

$$A(k) = 2c\Omega(k) \quad (4.43)$$

where $\Omega(k) \in R^m$ is defined as

$$\Omega(k) = \begin{bmatrix} \Omega_1(k), \dots, \Omega_{n_h}(k), & 0, \dots & 0, \dots \\ 0, \dots & \ddots & \vdots \\ \vdots & & \\ 0, \dots & 0, \dots & \Omega_1(k), \dots, \Omega_{n_h}(k) \end{bmatrix} \bar{V}(k) \quad (4.44)$$

and $\Omega_j(k) = \mu_j(k)\hat{W}_{j,:}(k)\hat{D}^v(k)\Delta^v$, $j = 1, \dots, n_h$. By such expanding of $A(k)$, $c(k)$ is explicit as shown in (4.43). To reveal the effect of $c(k)$, we further expand (4.25) as

$$\hat{y}^{v+}(\bar{V}(k)) - \hat{y}^{v-}(\bar{V}(k)) = 2H(k)c\Delta^v + 2\beta^v(k+1)c\Omega(k). \quad (4.45)$$

Using (4.25) and (4.27), we have the presentation of the system disturbance in terms of the measurement noise $\delta^v(k)$ of the RRSPSA algorithm as

$$\begin{aligned} \varepsilon(k) = & - \left\{ (H(k)\Delta^v + \beta^v\Omega(k))((H(k)\Delta^v + \beta^v\Omega(k)))^T \right\}^{-1} \\ & \times (H(k)\Delta^v + \beta^v\Omega(k))\delta^v(k)(2c)^{-1}. \end{aligned} \quad (4.46)$$

Furthermore, according to the equivalent disturbance (4.31), we have

$$\begin{aligned}\tilde{e}^v(k) &= \tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) \bar{V}^* + \varepsilon(k) \\ &= \tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) \bar{V}^* - \left\{ (H(k) \Delta^v + \beta^v \Omega(k)) \right. \\ &\quad \left. \times ((H(k) \Delta^v + \beta^v \Omega(k)))^T \right\}^{-1} \frac{(H(k) \Delta^v + \beta^v \Omega(k)) \delta^v(k)}{2c}.\end{aligned}\tag{4.47}$$

The equivalent disturbance $\tilde{e}^v(k)$ is important since its maximum value $\tilde{e}_m^v(k)$ decides the range of dead zone for $\alpha^v(k+1)$ in (4.40). Note also that the measurement noise $\delta^v(k)$ is defined the same way as in [40], which measures difference between true gradient and gradient approximation as derived in (4.26). Because the system disturbance $\varepsilon(k)$ is decided only by external conditions, a relatively smaller gain parameter c will imply a smaller measurement noise $\delta^v(k)$ according to (4.46). This allows us to choose the dead zone range mainly according to the external system disturbance $\varepsilon(k)$. Furthermore, as shown in (4.40), α^v will be nonzero whenever the norm of $e(k)$ is bigger than a dead zone value which is related to $\tilde{e}_m^v(k)$. If the external disturbance is also small, we can choose a small dead zone range, which ensures that the adaptive learning rate α^v has the nonzero value for a prolong period of training time.

4.3 Hidden Layer Training and Stability Analysis

The algorithm for updating estimated weight vector $\bar{W}(k)$ of the hidden layer is

$$\bar{W}(k+1) = \bar{W}(k) - \alpha^w(k+1) \hat{g}^w(\bar{W}(k))\tag{4.48}$$

where $\hat{g}^w(\bar{W}(k)) \in \mathbb{R}^{p^w}$ is the normalized gradient approximation that uses simultaneous perturbation vectors $\Delta^w(k+1) \in \mathbb{R}^{p^w}$ and $r^w(k+1) \in \mathbb{R}^{p^w}$ to stimulate weight of the hidden layer. $\Delta^w(k+1)$ and $r^w(k+1)$ are, respectively, the last p^w

components of perturbation vectors defined in (4.14) and (4.15).

$$\hat{g}^w(\bar{W}(k)) = -\frac{e^T(k) \left(B^+(k) - B^-(k) + \beta^w(k+1) \hat{D}^w(k) \right)}{2c\rho^w(k+1)} r^w(k+1) \quad (4.49)$$

where

$$B^+(k) = H(\bar{W}(k) + c\Delta^w(k+1), x(k)) \bar{V}(k) \in \mathbb{R}^m \quad (4.50)$$

$$B^-(k) = H(\bar{W}(k) - c\Delta^w(k+1), x(k)) \bar{V}(k) \in \mathbb{R}^m \quad (4.51)$$

$$\begin{aligned} & \hat{D}^w(k) \\ &= \left(H\left(\bar{W}(k), \left[(B^+(k-1))^T, \dots, (B^+(k-l))^T, 0, \dots, 0 \right]^T \right) \right. \\ & \quad \left. - H\left(\bar{W}(k), \left[(B^-(k-1))^T, \dots, (B^-(k-l))^T, 0, \dots, 0 \right]^T \right) \right) \bar{V}(k) \in \mathbb{R}^m. \end{aligned} \quad (4.52)$$

For simplicity of notations, we ignore the time subscripts of $\Delta^w(k+1)$, $r^w(k+1)$, $\beta^w(k+1)$, $\rho^w(k+1)$ and $\alpha^w(k+1)$ in the following part of the paper.

Remark 6. *The same as the output layer, our purpose is to find the desired hidden layer weight vector $\bar{W}^*$ of the gradient equation*

$$\hat{g}(\bar{W}(k)) = \frac{\partial L(\bar{W}(k))}{\partial \bar{W}(k)} = 0.$$

The detailed mathematical deduction is given in the proof of Lemma 4.2. $B^+(k) - B^-(k)$ is the result of the perturbations of the feedforward signals and $\hat{D}^w(k)$ is the result of the perturbations of the recurrent signals in RNNs. The function of β^w is to automatically switch off recurrent contribution when the convergence and stability requirements are violated. The conditions for the key parameters appeared in (4.48) and (4.49) are shown in (4.71), (4.74), and (4.75).

Lemma 4.2. *The normalized gradient approximation in (4.49) is an equivalent presentation of the standard SPSSA algorithm in equation (4.13).*

Proof. Similar to the definitions of $\hat{g}^{v+}(\bar{V}(k))$ and $\hat{g}^{v-}(\bar{V}(k))$ in (4.21) and (4.23),

we define two perturbation functions $\hat{g}^{w+}(\bar{V}(k))$ and $\hat{g}^{w-}(\bar{V}(k))$, i.e.,

$$\hat{g}^{w+}(\bar{W}(k)) = H(\bar{W}(k) + c\Delta^w, x(k)) \bar{V}(k) + H(\bar{W}(k), (x(k) + \Delta x(k+1))) \bar{V}(k). \quad (4.53)$$

From the definition of $B^+(k)$ in (4.50),

$$\begin{aligned} & \hat{g}^{w+}(\bar{W}(k)) \\ &= H(\bar{W}(k) + c\Delta^w, x(k)) \bar{V}(k) + H(\bar{W}(k), (x(k) + \Delta x(k+1))) \bar{V}(k) \\ &= B^+(k) + H\left(\bar{W}(k), \left([\hat{y}^T(k-1), \dots, \hat{y}^T(k-l), 0, \dots, 0]^T + \Delta x(k+1)\right)\right) \bar{V}(k) \\ &= B^+(k) + H\left(\bar{W}(k), \left[\bar{V}^T(k-1)H^T(\bar{W}(k-1) + c\Delta^w, x(k-1)), \dots \right.\right. \\ & \quad \left.\left. \bar{V}^T(k-l)H^T(\bar{W}(k-l) + c\Delta^w, x(k-l)), 0, \dots, 0\right]^T\right) \bar{V}(k) \\ &\approx B^+(k) + H\left(\bar{W}(k), \left[(B^+(k-1))^T, \dots, (B^+(k-l))^T, 0, \dots, 0\right]^T\right) \bar{V}(k). \end{aligned} \quad (4.54)$$

From the definition of $B^-(k)$ in (4.51), and similar to (4.54), we get

$$\begin{aligned} & \hat{g}^{w-}(\bar{W}(k)) \\ &\approx B^-(k) + H\left(\bar{W}(k), \left[(B^-(k-1))^T, \dots, (B^-(k-l))^T, 0, \dots, 0\right]^T\right) \bar{V}(k). \end{aligned} \quad (4.55)$$

Then, we have

$$\hat{g}^{w+}(\bar{W}(k)) - \hat{g}^{w-}(\bar{W}(k)) \approx B^+(k) - B^-(k) + \hat{D}^w(k). \quad (4.56)$$

$\hat{D}^w(k)$ in (4.56) is the result of recurrent training signal as shown in Fig. 4.1. To guarantee weight convergence and system stability during training, we add an adaptive recurrent hybrid parameter $\beta^w(k+1)$ to control the recurrent learning

signal path. That is,

$$\hat{g}^{w+}(\bar{W}(k)) - \hat{g}^{w-}(\bar{W}(k)) = B^+(k) - B^-(k) + \beta^w(k+1)\hat{D}^w(k). \quad (4.57)$$

According to (4.13), we can get

$$\begin{aligned} \hat{g}^w(\bar{W}(k)) &= \frac{L(\bar{W}(k) + c\Delta^w, \bar{V}(k)) - L(\bar{W}(k) - c\Delta^w, \bar{V}(k)) + \delta^w(k)}{2c\rho^w} r^w \\ &= \frac{\|y(k) - \hat{g}^{w+}(k)\|^2 - \|y(k) - \hat{g}^{w-}(k)\|^2 + 2\delta^w(k)}{4c\rho^w} r^w \\ &= \frac{(y(k) - \hat{g}^{w+}(k) + y(k) - \hat{g}^{w-}(k))^T (\hat{g}^{w-}(k) - \hat{g}^{w+}(k)) + 2\delta^w(k)}{4c\rho^w} r^w \\ &= \left(2y(k) - \hat{g}^{w+}(k) - \hat{g}^{w-}(k) + \hat{g}^{w+}(k) + \hat{g}^{w-}(k) - 2\hat{y}(k) + 2\varepsilon(k) \right)^T \\ &\quad \times \left((\hat{g}^{w-}(k) - \hat{g}^{w+}(k)) + 2\delta^w(k) \right) r^w (4c\rho^w)^{-1} \\ &= (y(k) - \hat{y}(k) + \varepsilon(k))^T (\hat{g}^{w-}(k) - \hat{g}^{w+}(k)) r^w (2c\rho^w)^{-1} \\ &= e^T(k) (\hat{g}^{w-}(k) - \hat{g}^{w+}(k)) r^w (2c\rho^w)^{-1} \\ &= -e^T(k) \left(B^+(k) - B^-(k) + \beta^w \hat{D}^w(k) \right) r^w (2c\rho^w)^{-1} \end{aligned} \quad (4.58)$$

where the equivalent disturbance $\delta^w(k)$ is defined as

$$\begin{aligned} \delta^w(k) &= (\hat{g}^{w-}(\bar{W}(k)) - \hat{g}^{w+}(\bar{W}(k)))^T \varepsilon(k) + \frac{1}{2} (\hat{g}^{w-}(\bar{W}(k)) - \hat{g}^{w+}(\bar{W}(k)))^T \\ &\quad \times (\hat{g}^{w+}(\bar{W}(k)) + \hat{g}^{w-}(\bar{W}(k)) - 2\hat{y}(k)). \end{aligned} \quad (4.59)$$

The fifth equality in (4.58) reveals the relationship between $\delta^w(k)$ and $\varepsilon(k)$ as in (4.59). ■

For stability analysis, the training error will be linked to the parameter estimating error of the hidden layer and the disturbance. According to mean value theorem and that activation function is a nondecreasing function, it can be readily shown

that there exist unique positive mean values $\tilde{\mu}_j(k)$ such that

$$h(W_{j,:}^* x(k)) - h(\hat{W}_{j,:}(k)x(k)) = \tilde{\mu}_j(k) (W_{j,:}^* - \hat{W}_{j,:}(k)) x(k) = \tilde{\mu}_j(k) \tilde{\tilde{W}}_{j,:}(k)x(k) \quad (4.60)$$

where $\hat{W}_{j,:}(k)$, $W_{j,:}^*$, $\tilde{\tilde{W}}_{j,:}(k)$ are the estimated weight, ideal weight, and weight estimate error vectors linked to the j th hidden layer neurons, respectively. From (4.60) and the matrix in (4.5), it can be shown that

$$\tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) \bar{V}(k) = (H(\bar{W}^*, x(k)) - H(k)) \bar{V}(k) = \tilde{\Omega}(k) \tilde{\tilde{W}}(k) \quad (4.61)$$

where $\tilde{\Omega}(k) \in \mathbb{R}^{m \times p^w}$ is defined as

$$\begin{aligned} \tilde{\Omega}(k) &= \begin{bmatrix} \tilde{\mu}_1(k) \bar{V}_1(k) x^T(k) & \cdots & \tilde{\mu}_{n_h}(k) \bar{V}_{n_h}(k) x^T(k) \\ \tilde{\mu}_1(k) \bar{V}_{n_h+1}(k) x^T(k) & \cdots & \tilde{\mu}_{n_h}(k) \bar{V}_{2n_h}(k) x^T(k) \\ \vdots & & \vdots \\ \tilde{\mu}_1(k) \bar{V}_{(m-1)n_h+1}(k) x^T(k) & \cdots & \tilde{\mu}_{n_h}(k) \bar{V}_{p^w}(k) x^T(k) \end{bmatrix} \\ &= [\tilde{\mu}_1(k) \hat{V}_{1,:}^T(k) x^T(k) \cdots \tilde{\mu}_{n_h}(k) \hat{V}_{n_h,:}^T(k) x^T(k)] \end{aligned} \quad (4.62)$$

and $\hat{V}_{j,:}^T(k)$ is inverse of j th row of the estimated weight matrix $\hat{V}(k)$ of output layer. From (4.6), the maximum value of the derivative of $h_j(k)$,

$$\dot{h}_j(k) = (\partial h(\hat{W}_{j,:}(k) x(k))) / (\partial(\hat{W}_{j,:}(k) x(k)))$$

is λ , and hence $0 \leq \tilde{\mu}_j(k) \leq \lambda$ ($1 \leq i \leq n_h$).

We can rewrite the estimation error as

$$\begin{aligned} e(k) &= y(k) - \hat{y}(k) + \varepsilon(k) \\ &= H(\bar{W}^*, x(k)) \bar{V}^* - H(\bar{W}^*, x(k)) \bar{V}(k) + H(\bar{W}^*, x(k)) \bar{V} - H(k) \bar{V}(k) + \varepsilon(k) \\ &= -\phi^w(k) + \tilde{e}^w(k) \end{aligned} \quad (4.63)$$

where

$$\phi^w(k) = -\tilde{H}(\bar{W}(k), \bar{W}^*, x(k)) \bar{V}(k) = -\tilde{\Omega}(k) \tilde{W}(k) \quad (4.64)$$

and the bounded equivalent disturbance $\tilde{e}^w(k)$ of the hidden layer is defined as

$$\tilde{e}^w(k) = H(\bar{W}^*, x(k)) \tilde{V}(k) + \varepsilon(k). \quad (4.65)$$

Note that $\tilde{e}^w(k)$ is a bounded signal because $\tilde{V}(k)$ has been proven to be bounded in Section III.

According to the definition of $B^+(k)$ and $B^-(k)$ defined in (4.50) and (4.51), we can get

$$(B^+(k) - B^-(k)) (2c)^{-1} = \hat{\Omega}(k) \Delta^w \quad (4.66)$$

where $\hat{\Omega}(k) \in \mathbb{R}^{m \times p^w}$ is similar to $\tilde{\Omega}(k)$ in (4.62) except with different mean values.

Theorem 4.2. *If the hidden layer of the RNN is trained by the adaptive RRSPSA algorithm (4.48), weight $\bar{W}(k)$ is guaranteed to be convergent in the sense of Lyapunov function*

$$\left\| \tilde{W}(k+1) \right\|^2 - \left\| \tilde{W}(k) \right\|^2 \leq 0, \quad \forall k$$

with $\tilde{W}(k) = \bar{W}^* - \bar{W}(k)$ and the training error will be L_2 -stable in the sense of $\tilde{e}^w(k) \in L_2$ and $\phi^w(k) \in L_2$.

Proof. Consider that SPSA is an approximation of the gradient algorithm [40], and using the property of local minimum points of the gradient [112]

$$-(\partial(e^T(k)e(k))) / (\partial(\bar{W}(k))^T) \tilde{W}(k) \geq 0$$

we have

$$\begin{aligned} 0 &\leq -\frac{\partial(e^T(k)e(k))}{\partial(\bar{W}(k))^T} \tilde{W}(k) \\ &= \sum_{i=1}^{n_h} \left\{ \dot{h}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right\} \end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{n_h} \left\{ \frac{\dot{h}_i(k)}{\tilde{\mu}_i(k)} \left(\tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right) \right\} \\
&\leq \frac{\lambda}{\lambda_{\min}} \sum_{i=1}^{n_h} \left\{ \tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right\} \\
&= \frac{\lambda}{\lambda_{\min}} e^T(k) \left(\tilde{\Omega}(k) \tilde{W}(k) \right) \\
&= - \frac{\lambda}{\lambda_{\min}} e^T(k) \phi^w(k)
\end{aligned} \tag{4.67}$$

where λ is the maximum value of derivative $\dot{h}_j(k)$ of activation function in (4.6), and $\lambda_{\min} \neq 0$ is the minimum nonzero value of mean value $\tilde{\mu}_j(k)$ defined in (4.72). (Note that the inequality is always true if $\lambda_{\min} = 0$, which implies $\dot{h}_j(k) = \tilde{\mu}_j(k) = 0$.) According to the training algorithm for hidden layer in (4.48), we get

$$\tilde{W}(k+1) = \tilde{W}(k) + \alpha^w \hat{g}^w(\bar{W}(k)) \tag{4.68}$$

where $\hat{g}^w(\bar{W}(k))$ is defined in (4.58). Using (4.57), (4.68) and (4.67), by the definition of $\bar{\Omega}(k)$ in (4.73), we get

$$\begin{aligned}
&\left\| \tilde{W}(k+1) \right\|^2 - \left\| \tilde{W}(k) \right\|^2 \\
&= 2\alpha^w \hat{g}^w(\bar{W}(k)) \tilde{W}(k) + \left\| \alpha^w \hat{g}^w(\bar{W}(k)) \right\|^2 \\
&= -\alpha^w e^T(k) \frac{(B^+(k) - B^-(k)) + \beta^w \hat{D}^w(k)}{c\Delta^w} (r^w)^T \tilde{W}(k) + \left\| \alpha^w \hat{g}^w(\bar{W}(k)) \right\|^2 \\
&= \sum_{i=1}^{n_h} \left\{ -\mu_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right\} \left\{ \tilde{W}^T(k) \left(\tilde{W}(k) \tilde{W}^T(k) \right)^\dagger \Delta^w (r^w)^T \tilde{W}(k) \right\} \\
&\quad \times \frac{2\alpha^w}{\Delta^w} - \alpha^w e^T(k) \left(\bar{\Omega}(k) \tilde{W}(k) \right) \left(\bar{\Omega}(k) \tilde{W}(k) \right)^T \left[\left(\bar{\Omega}(k) \tilde{W}(k) \right) \left(\bar{\Omega}(k) \tilde{W}(k) \right)^T \right]^\dagger \\
&\quad \times \hat{D}^w(k) (r^w)^T \tilde{W}(k) \beta^w (c\Delta^w)^{-1} + \left\| \alpha^w \hat{g}^w(\bar{W}(k)) \right\|^2 \\
&= \sum_{i=1}^{n_h} \left\{ \frac{\mu_i(k)}{\tilde{\mu}_i(k)} \left(-\tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right) \right\} 2p^w \alpha^w (\Delta^w)^{-1} \\
&\quad + \sum_{i=1}^{n_h} \left\{ \frac{1}{\tilde{\mu}_i(k)} \left(-\tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right) \right\} \alpha^w (c\Delta^w)^{-1} \\
&\quad \times \beta^w (r^w)^T \left(\bar{\Omega}^T(k) \bar{\Omega}(k) \right)^\dagger \bar{\Omega}^T(k) \bar{\Omega}(k) \tilde{W}(k) \left(\bar{\Omega}(k) \tilde{W}(k) \right)^T
\end{aligned}$$

$$\begin{aligned}
& \times \left[\left(\bar{\Omega}(k) \tilde{W}(k) \right) \left(\bar{\Omega}(k) \tilde{W}(k) \right)^T \right]^\dagger \hat{D}^w(k) + \|\alpha^w \hat{g}^w(\bar{W}(k))\|^2 \\
& = \sum_{i=1}^{n_h} \left\{ \frac{\mu_i(k)}{\tilde{\mu}_i(k)} \left(-\tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right) \right\} 2p^w \alpha^w (\Delta^w)^{-1} \\
& \quad + \sum_{i=1}^{n_h} \left\{ \frac{1}{\tilde{\mu}_i(k)} \left(-\tilde{\mu}_i(k) e^T(k) \hat{V}_{i,:}^T(k) x^T(k) \tilde{W}_{i,:}(k) \right) \right\} \alpha^w (c\Delta^w)^{-1} \\
& \quad \times \beta^w (r^w)^T \left(\bar{\Omega}^T(k) \bar{\Omega}(k) \right)^\dagger \bar{\Omega}^T(k) \hat{D}^w(k) + \|\alpha^w \hat{g}^w(\bar{W}(k))\|^2 \tag{4.69} \\
& \leq 2\alpha^w \frac{\lambda_{\min}}{\lambda} \frac{p^w}{\Delta^w} e^T(k) \phi^w(k) \\
& \quad + \alpha^w \frac{1}{\lambda} \frac{\beta^w}{c\Delta^w} e^T(k) \phi^w(k) (r^w)^T \left(\bar{\Omega}^T(k) \bar{\Omega}(k) \right)^\dagger \bar{\Omega}^T(k) \hat{D}^w(k) \\
& \quad + \|e(k)\|^2 \left\| \frac{B^+(k) - B^-(k) + \beta^w \hat{D}^w(k)}{2c} \right\|^2 \|r^w\|^2 \left(\frac{\alpha^w}{\Delta^w} \right)^2.
\end{aligned}$$

Similar to the output layer case, $\phi^w(k)$ is replaced with $(\tilde{e}^w(k) - e(k))$, we have the following Lyapunov function for convergence analysis:

$$\begin{aligned}
& \left\| \tilde{W}(k+1) \right\|^2 - \left\| \tilde{W}(k) \right\|^2 \\
& \leq \frac{\alpha^w}{\Delta^w} \left[\frac{\lambda_{\min}}{\lambda} p^w + \frac{\beta^w}{2\lambda c} (r^w)^T \left(\bar{\Omega}^T(k) \bar{\Omega}(k) \right)^\dagger \bar{\Omega}^T(k) \hat{D}^w(k) \right] \\
& \quad \times \left\{ \left\| \tilde{e}^w(k) \right\|^2 - \left(1 - \frac{\alpha^w}{\Delta^w} \left\| \frac{B^+(k) - B^-(k) + \beta^w \hat{D}^w(k)}{2c} \right\|^2 \|r^w\|^2 \right. \right. \tag{4.70} \\
& \quad \left. \left. \times \left[\frac{\lambda_{\min}}{\lambda} p^w + \frac{\beta^w}{2\lambda c} (r^w)^T \left(\bar{\Omega}^T(k) \bar{\Omega}(k) \right)^\dagger \bar{\Omega}^T(k) \hat{D}^w(k) \right] \right)^{-1} \|e(k)\|^2 \right\}
\end{aligned}$$

The three adaptive parameters defined in (4.71), (4.74) and (4.75) guarantee the negativeness of Lyapunov function. That is

$$\left\| \tilde{W}(k+1) \right\|^2 - \left\| \tilde{W}(k) \right\|^2 \leq 0, \forall k$$

■

The three adaptive learning rates appeared in (4.48) and (4.49) are as follows.

1. The normalization factor is defined as

$$\rho^w(k+1) = \tau \rho^w(k) + \max \left\{ 1 + \alpha^w M(k) \|r^w\|^2 \Upsilon^{-1}(k), \bar{\rho} \right\} \quad (4.71)$$

where

$$M(k) = \left\| \frac{B^+(k) - B^-(k) + \beta^w \hat{D}^w(k)}{2c} \right\|^2$$

$$\Upsilon(k) = \frac{\lambda_{\min}}{\lambda} p^w + \frac{1}{2\lambda c} \beta^w (r^w)^T (\bar{\Omega}(k) \bar{\Omega}(k))^{\dagger} \bar{\Omega}^T(k) \hat{D}^w(k)$$

and $0 < \alpha^w \leq 1$ being a positive constant, λ being the maximum value of the derivative $\dot{h}_j(k)$ of the activation function in (4.6), and $\lambda_{\min} \neq 0$ being the minimum nonzero value of the mean value $\tilde{\mu}_j(k)$ defined as

$$0 \leq \tilde{\mu}_j(k) \leq \lambda \quad (1 \leq i \leq n_h) \quad (4.72)$$

and

$$\bar{\Omega}(k) = \begin{bmatrix} \bar{V}_1(k)x^T(k) & \cdots & \bar{V}_{n_h}(k)x^T(k) \\ \bar{V}_{n_h+1}(k)x^T(k) & \cdots & \bar{V}_{2n_h}(k)x^T(k) \\ \vdots & & \vdots \\ \bar{V}_{(m-1)n_h+1}(k)x^T(k) & \cdots & \bar{V}_{p^v}(k)x^T(k) \end{bmatrix} \quad (4.73)$$

$$= [\hat{V}_{1,:}^T(k)x^T(k) \cdots \hat{V}_{n_h,:}^T(k)x^T(k)] \in R^{m \times p^w}$$

where $\bar{V}_j(k)$ is the j th element of $\bar{V}(k)$, $1 \leq j \leq p^v$.

2. The adaptive dead zone learning rate of hidden layer is defined as

$$\begin{cases} \alpha^w(k+1) = \alpha^w, & \text{if } \|e(k)\| \geq \Gamma(k) \\ \alpha^w(k+1) = 0, & \text{if } \|e(k)\| < \Gamma(k) \end{cases} \quad (4.74)$$

with $\Gamma = \tilde{e}_m^w(k) / \sqrt{\left(1 - \frac{\alpha^w \|r^w\|^2}{\rho^w \Upsilon(k)} M(k)\right)}$ and $\tilde{e}_m^w(k)$ being the maximum value of $\tilde{e}^w(k)$ defined in (4.65).

3. The recurrent hybrid adaptive parameter of hidden layer β^w is defined as

$$\begin{cases} \beta^w(k+1) = 1, & \text{if } (r^w)^T \chi(k) \geq 0 \\ \beta^w(k+1) = 0, & \text{if } (r^w)^T \chi(k) < 0 \end{cases} \quad (4.75)$$

with $\chi(k) = (\bar{\Omega}(k)\bar{\Omega}(k))^{\dagger} \bar{\Omega}^T(k) \hat{D}^w(k)$. $\beta^w(k+1)$ can automatically cut off the recurrent training signal as in the lower right corner of Fig. 4.1 whenever weight convergence and stability conditions of the RNN are violated as shown in Theorem 4.2.

Remark 7. *Summary of the RRSPSA algorithm for RNNs.*

1. *Set a maximum training step as the stop criterion;*
2. *Initialize the weights for the RNN;*
3. *Form new input $x(k)$ as defined in equation (4.3);*
4. *Perform forward computation and calculate the output $\hat{y}(k)$ and error $e(k)$ with the input and current estimated weights;*
5. *Calculate $\rho^v(k+1)$, $\alpha^v(k+1)$, and $\beta^v(k+1)$ using (4.37), (4.40), and (4.41), respectively;*
6. *Update the output layer weights $\bar{V}(k+1)$ via the learning law (4.16) for the next iteration;*
7. *Calculate $\rho^w(k+1)$, $\alpha^w(k+1)$, and $\beta^w(k+1)$ using (4.71), (4.74), and (4.75), respectively;*
8. *Update the hidden layer weights $\bar{W}(k+1)$ via the learning law (4.48) for the next iteration;*
9. *Go back to step 3 to continue the iteration until the stop criterion is reached.*

4.4 Experimental Verification

4.4.1 Time Series Prediction

Sunspots are dark blotches on the sun which are caused by magnetic storms on the surface of the sun. The underlying mechanism for sunspot appearances is not exactly known. The sunspots data is a classical example of a combination of periodic and chaotic phenomena which has served as a benchmark in the statistics literature of time series and much work has been done in trying to analyze the sunspot data using neural networks [122, 123]. One set of the high quality America Sunspot Numbers, monthly sunspot numbers from December 1949 to October 2009 in the database, is used as an example to minimize unnecessary human measurement errors. The total America sunspot number in the monthly database is 778 points. The output sequence is scaled by dividing by 100, to get into a range suitable for the network. The hidden neuron number of the RNN is ten. The selected learning rates for BP and RTRL are 0.09 and 0.01, respectively. Perturbation step size for RRSPSA is $c = 0.01$. Fig. 4.2 shows outputs of the plant $y^*(k)$ and estimated outputs $y(k)$ of the RNN trained by the three different methods. Table 4.1 gives comparison results of the three methods in terms of MSE and STD.

Table 4.1: Comparison of Mean Squared Error and Standard Deviation of the Training Error for Sunspot time-series.

Squared Error	RRSPSA	RTRL	BP
MSE	$6.01e^{-4}$	$1.30e^{-3}$	$4.30e^{-3}$
STD	$2.45e^{-2}$	$3.61e^{-2}$	$6.57e^{-2}$

4.4.2 Michaelis-Menten equation

Consider a continue time Michaelis-Menten equation [124]

$$\dot{y}(t) = 0.45 \frac{y(t)}{2.27 + y(t)} - y(t) + u(t) \quad (4.76)$$

Figure 4.2: Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RRPSA training algorithms for time series prediction.

Given an unit impulse as input $u(t)$, a fourth-order Runge-Kutta algorithm is used to simulate this model with integral step size $\Delta t = 0.01$, and 1000 equi-spaced samples are obtained from input and output with a sampling interval of $T = 0.02$ time units. The measured noise is white Gaussian noise with mean 0 and variance 0.05. An RNN is utilized to emulate the dynamics of the given system and approximate the system output in (4.76). Gradient function of this model has many local minima and the global minimum has a very narrow domain of attraction [124]. We compare performance of the RRSPSA to that of both BP and RTRL training algorithms. The hidden neuron number of RNNs in this example is selected as ten. The fixed learning rates for BP and RTRL are 0.03 and 0.01, respectively. The perturbation size $c = 0.01$ is selected for RRSPSA. Fig. 4.3 shows outputs of the plant $y^*(k)$ and estimated outputs $y(k)$ of the RNN trained by the three different methods. Table 4.2 shows the comparison results of the three methods in terms of MSE and STD.

Table 4.2: Comparison of Mean Squared Error and Standard Deviation of the Error for Michaelis-Menten equation.

	RRSPSA	RTRL	BP
MSE	$4.2e^{-3}$	$1.64e^{-2}$	$2.98e^{-2}$
STD	$6.46e^{-2}$	$1.28e^{-1}$	$1.72e^{-1}$

4.4.3 Robot Control

To compare the performance of RRSPSA with MRAGD in Chapter 3, we test RRSPSA on the same robot control problem as in Section 3.5. The system description and experimental setup are the same as those in Section 3.5. The perturbation size $c = 0.03$ is selected for RRSPSA. Figure 4.4 shows the tracking control results using RRSPSA and MRAGD algorithms which are compared with the desired tracking trajectories. The squared instant tracking errors of the initial responses of the two joints using the two neural controllers are presented in Figure 4.5, which are plotted inside the ellipse circles in Figure 4.4. To provide a clear comparison among the steady state responses using the three algorithms, the squared instant tracking errors from the step 200 to 2000 of the two joints are given in Figure 4.6. As shown

Figure 4.3: Output $y(k)$ and reference signal $y^*(k)$ and error $e(k)$ using BP and RTRL and RSPSA training algorithms for simulation of Michaelis-Menten equation.

(a) Output $y^d(k)$ and reference signal $y(k)$ using RRSPSA based neural controller.

(b) Output $y^d(k)$ and reference signal $y(k)$ using MRAGD based neural controller.

Figure 4.4: Output $y^d(k)$ and reference signal $y(k)$ using RRSPSA and MRAGD based neural controller.

in Figure 4.5 and Figure 4.6, RRSPSA and MRAGD can obtain comparable results. They arrive at the stable state at similar steps and the steady state responses are also similar. Table 4.3 shows the mean squared error (MSE) and the standard deviation (Std) for each dimension of the tracking results which are defined the same way as those in Chapter 3.

Figure 4.5: Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ (initial response in Figure 3.2) from the step 1 to 200 using RRSPSA and MRAGD training algorithms.

Figure 4.6: Squared instantaneous tracking errors $s_1^2(k)$ and $s_2^2(k)$ from the step 200 to 2000 using RRSPSA and MRAGD training algorithms.

Table 4.3: Comparations of the Tracking Statistics of Steady-state between RRSPSA and MRAGD.

	RRSPSA	MRAGD
MSE_1	$1.444e^{-4}$	$1.9616e^{-4}$
MSE_2	$1.5737e^{-4}$	$2.5597e^{-4}$
Std_1	$1.2e^{-2}$	$1.4e^{-2}$
Std_2	$1.25e^{-2}$	$1.58e^{-2}$

4.5 Summary

We propose a robust recurrent simultaneous perturbation stochastic approximation (RRSPSA) algorithm under the framework of deterministic system with guaranteed weight convergence. The basic idea is to vary all elements of unknown weight parameters simultaneously to solve the optimal or suboptimal weight from the gradient function. Only two objective function measurements are used at each iteration, which maintain the efficiency of SPSA. And the convergence property and stability of the weight estimate of the proposed SPSA training algorithm for RNNs are also addressed. It utilizes the specifically designed three adaptive parameters to maximize training speed for recurrent training signal while exhibiting certain weight convergence properties with only two objective function measurements as a standard SPSA algorithm. Robust stability and weight convergence proofs of RRSPSA algorithm are provided based on Lyapunov function. Two experiments are carried out to verify the derived theoretical results, which show clearly that the learning performance of RRSPSA was improved in terms of convergence speed and mean squared error as compared to that of conventional RNN algorithms.

Chapter 5

Online Prediction of Time Series Data with Recurrent Kernels

Kernel methods present an alternative to the linear parametric approach. The basic idea underlying kernel methods is that a Mercer kernel function, which is applied to pairs of input vectors, can be interpreted as an inner product in a high-dimensional Hilbert space, the feature space; this formulation enables the computation of inner products in the feature space without explicitly using the feature vectors. This idea, which is commonly known as the kernel trick, has been used extensively in recent years [70–72, 75, 125–127]. In online kernel learning applications, sparsification is performed at each time step using a new training sample, and a simple decision has to be made: either add the sample into the representation or discard it.

Let H denotes the RKHS of real-valued functions on a compact set $U \subset R^l$, and let $\langle \cdot, \cdot \rangle_H$ be the inner product on H . For every input pattern x_i , there exists a feature vector $\phi(x_i)$ that can map the original vector x into H ; the feature vector satisfies

$$\phi(x_i) = \langle \phi(\cdot), k(\cdot, x_i) \rangle_H \quad (5.1)$$

i.e.,

$$k(x_i, x_j) = \langle k(\cdot, x_i), k(\cdot, x_j) \rangle_H \quad (5.2)$$

The integration of kernel and RTRL learning algorithms has historically attracted re-

search interest because both types of algorithms are powerful tools for the implementation of adaptive nonlinear time series prediction and filtering algorithms [33] [128]. This integrated approach requires through research to simultaneously address two important issues in a unified manner, i.e., the sparsification of the kernel-based method and a weight convergence of the RTRL-based approach [75] [6] [128]. Previous research has addressed the two problems separately; for example, different kernel sparsification algorithms were developed in [70], [72], and [75]. Novel adaptive nonlinear RTRL learning algorithms were proposed in [5, 39] for adaptive online learning. A quantized kernel least-mean-squares algorithm subjected to a type of feedforward network weight convergence analysis is presented [75]. Despite that Kivinen et al [127] proposed an initial study in the weight convergence linked to a simple kernel learning method, there is little research focused on a unified approach that considers the two important aspects of kernel sparsification and weight convergence analysis for recurrent kernel online learning algorithms. The second issue is more important for the kernel recurrent learning algorithm compared with a normal feedforward network.

In this chapter, we establish an inherent link between popular online kernel and RTRL learning methods via implementation of the proposed recurrent kernel online learning (RKOL) algorithm. The method not only ensures weight convergence but also yields kernel sparsification conditions that can automatically eliminate or stop the growth of the kernel function.

5.1 The RKOL Algorithm

Define the discrete-time RNN network input vector as

$$x_t = [u_t, \dots, u_{t-l}, \dots, \hat{y}_{t-1}, \dots, \hat{y}_{t-d}] \in \mathbb{R}^{n_I} \quad (5.3)$$

where u_t is the external input. d is the discrete time delay step of the recurrent output feedback of Jordan network [129] and l is the input time delay step.

The output of the proposed kernel function can typically be presented as a reducing Radial Basis Function (RBF) or RBF-like network of size m_t at the discrete time step t :

$$\hat{y}_t = \alpha_t K_t = \sum_{i=1}^{m_t} \alpha_{t,i} k(x_t, c_{t,i}) \quad (5.4)$$

K_t is defined as follows:

$$K_t = [k(x_t, c_{t,1}), \dots, k(x_t, c_{t,m_t})]^T \in \mathbb{R}^{m_t} \quad (5.5)$$

where $k(x_t, c_{t,i})$ is the classical Gaussian kernel, $k(x_t, c_{t,i}) = \exp(-||x_t - c_{t,i}||^2 / 2\beta^2)$, $i = 1, \dots, m_t$, $\beta > 0$ is the kernel width, the variable m_t is the kernel network size at the discrete time step t and $c_{t,i}$ with $1 \leq i \leq m_t$ is the center of the Gaussian kernel function.

The coefficient vector is expressed as

$$\alpha_t = [\alpha_{t,1}, \dots, \alpha_{t,m}] \in \mathbb{R}^{m_t}. \quad (5.6)$$

From (5.3), it is shown that the delayed outputs are included in the input vector, therefore, it is inherently a RNN structure.

Define the conventional loss function

$$E_t = \frac{1}{2}(e_t)^2 \quad (5.7)$$

and error is defined as

$$e_t = y_t - \hat{y}_t + \varepsilon_t \quad (5.8)$$

where y_t is the desired output and ε_t is the disturbance.

In a normal least mean square algorithm, the parameter weight vector α_t can be easily estimated through a conventional iterative equation. The general form of the

update rule for the weight is

$$\begin{aligned}\alpha_{t+1} &= \alpha_t - \eta \frac{\partial E_t}{\partial \alpha_t} \\ &= \alpha_t + \eta e_t K_t^T\end{aligned}\tag{5.9}$$

where η is a constant learning rate.

However, there are two problems in the iterative equation (5.9). First the classical estimate algorithm is sensitive to disturbance ε_t based on a small constant learning rate η while the learning processing can be very slow if the learning rate is too small. The second problem is if we set up a similar iterative equation based on a nonlinear gradient of E_t as in (5.9) to estimate the kernel function center $c_{t,i}$, which may be estimated in a similar manner as (5.9) and it will lead to a nonlinear recursive estimation problem without the capability to control the size of the network [33, 39]. For the reduced data dictionary and the integration of kernel and RTRL learning algorithm, our loss function differential is defined as

$$\begin{aligned}& \frac{dE_t}{d\alpha_t} \\ &= \frac{1}{2} \left[\frac{\partial (e_t^2)}{\partial \alpha_t} + \frac{\partial (e_t^2)}{\partial K_t} \frac{\partial K_t}{\partial \alpha_t} \right] \\ &= -e_t K_t^T \\ &\quad - e_t \alpha_t \left[k(x_t, c_{t,1}) \left(-\frac{x_t - c_{t,1}}{\beta^2} \right) \dots k(x_t, c_{t,m}) \left(-\frac{x_t - c_{t,m}}{\beta^2} \right) \right]^T \frac{\partial x_t}{\partial \alpha_t} \\ &= -e_t (K_t^T + \alpha_t \bar{K}_t D_t^\alpha)\end{aligned}\tag{5.10}$$

For RNNs, the weight is not only depend on the output at the current time, but also those at previous steps, which can be seen from the input vector in (5.3). It will lose much information and may result in ineffective learning if the recurrent dependence is simply ignored while calculating the gradient. That is why the second term is

included in (5.10). Thus, (5.9) is rewritten as

$$\begin{aligned}\alpha_{t+1} &= \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + \alpha_t \bar{K}_t D_t^\alpha) \\ &= \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + A_t)\end{aligned}\tag{5.11}$$

where

$$A_t = \alpha_t \bar{K}_t D_t^\alpha \in \mathbb{R}^{m_t} \tag{5.12}$$

$$\bar{K}_t = [k(x_t, c_{t,1})(-\frac{x_t - c_{t,1}}{\beta^2}) \dots k(x_t, c_{t,m_t})(-\frac{x_t - c_{t,m_t}}{\beta^2})]^T \in R^{m_t \times n_I} \tag{5.13}$$

$$\begin{aligned}D_t^\alpha &= \left[\frac{\partial u_t}{\partial \alpha_t}, \dots, \frac{\partial u_{t-l}}{\partial \alpha_t}, \frac{\partial y_{t-1}}{\partial \alpha_t}, \dots, \frac{\partial y_{t-d}}{\partial \alpha_t} \right]^T \\ &\approx \begin{bmatrix} \left(\frac{\partial u_t}{\partial \alpha_t} \right)^T \\ \dots \\ \dots \\ \left(\frac{\partial u_{t-l}}{\partial \alpha_t} \right)^T \\ \left(\frac{\partial y_{t-1}}{\partial \alpha_{t-1}} \right)^T \\ \dots \\ \dots \\ \left(\frac{\partial y_{t-d}}{\partial \alpha_{t-d}} \right)^T \end{bmatrix}^T \in R^{n_I \times m_t}\end{aligned}\tag{5.14}$$

The definitions for η_t and ρ_t is shown in (5.15) and (5.16) respectively which are both come from the weight convergence proof. The adaptive normalization factor is defined to overcome the vanished cone problem in a classical adaptive training scheme like the RAGD [39]. The classical adaptive learning rate is defined to gainst the weight drift in the iteration [130].

$$\begin{cases} \eta_t = 1 & \text{if } e_t \geq \sqrt{\left(1 - \frac{\|K_t^T + A_t\|^2}{\rho_t(1 + \Gamma_t A_t^T)}\right)} \tilde{\varepsilon}_m^\alpha \\ \eta_t = 0 & \text{if } e_t < \sqrt{\left(1 - \frac{\|K_t^T + A_t\|^2}{\rho_t(1 + \Gamma_t A_t^T)}\right)} \tilde{\varepsilon}_m^\alpha \end{cases}\tag{5.15}$$

$$\rho_t = \nu \rho_{t-1} + \max\{\bar{\rho}_t, \frac{\|K_t^T + A_t\|^2}{1 + \Gamma_t A_t^T}\}\tag{5.16}$$

where $\Gamma_t = K_t^T (K_t K_t^T)^{\dagger}$, $\nu > 0$, $\bar{\rho}_t > 0$ and $\tilde{\varepsilon}_m^\alpha > 0$.

The second term in the last equation of (5.10) comes from the recurrent signal and the definition of D_t^α shows clearly the derivative of the delayed outputs with respect to the weight at the current step. Similar to RTRL introduced by Williams and Zipser [12, 13], we assume the model parameters have small change between each iteration, i.e. $\alpha(k) \approx \alpha(k-1) \cdots \approx \alpha(k-l)$. Moreover, such approximation makes the proposed algorithm real time and adaptive in a recursive fashion. The partial derivatives in each column of D_t^α can be iteratively obtained from the previous steps to speed up the calculation.

5.2 Weight Convergence and Online Sparicification

For stability analysis, the training error $e(k)$ will be linked to the parameter estimating error of the output layer and the disturbance.

$$\begin{aligned}
 e_t &= y_t - \hat{y}_t + \varepsilon_t \\
 &= \alpha_t^* K_t^* - \alpha_t K_t + \varepsilon_t \\
 &= \alpha_t^* K_t^* - \alpha_t^* K_t - (\alpha_t K_t - \alpha_t^* K_t) + \varepsilon_t \\
 &= \tilde{\varepsilon}_t^\alpha - e_t^\alpha
 \end{aligned} \tag{5.17}$$

$\alpha^* \in \mathbb{R}^{m_t}$ is the true weight vector of the kernel function with m_t ideal (optimal or suboptimal true value) components and K_t^* is the desired value for K_t .

$$\tilde{\varepsilon}_t^\alpha = \alpha_t^* (K_t)^* - \alpha_t^* K_t + \varepsilon_t. \tag{5.18}$$

$$e_t^\alpha = \alpha_t K_t - \alpha_t^* K_t = \tilde{\alpha}_t K_t. \tag{5.19}$$

$$\tilde{\alpha}_t = \alpha_t - \alpha^* \in \mathbb{R}^{m_t} \tag{5.20}$$

is the prediction error of the weight at step t . The e_t^α , $e(t)$ and ε_t^α correspond to the weight estimation error, the RNN output error, and the disturbance, respectively. Subtracting α^* and then squaring both sides of equation (5.11), we obtain

$$\begin{aligned}
& \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\
&= \left[2\eta_t \rho_t e_t \tilde{\alpha}_t (K_t^T + A_t)^T + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right] / \rho_t^2 \\
&= \left[2\eta_t \rho_t e_t \tilde{\alpha}_t K_t \left((1 + K_t^T (K_t K_t^T)^\dagger A_t^T) + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right) \right] / \rho_t^2 \\
&= \left[2\eta_t \rho_t e_t e_t^\alpha \left(1 + K_t^T (K_t K_t^T)^\dagger A_t^T \right) + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right] / \rho_t^2 \\
&= \left[2\eta_t \rho_t e_t e_t^\alpha \left(1 + K_t^T (K_t K_t^T)^\dagger A_t^T \right) + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right] / \rho_t^2
\end{aligned} \tag{5.21}$$

Multiply both sides of (5.18) by e_t and replacing $e_t e_t^\alpha$, we can rewrite (5.21) as

$$\begin{aligned}
& \left[2\eta_t \rho_t (\tilde{\varepsilon}_t^\alpha e_t - e_t^2) \left(1 + K_t^T (K_t K_t^T)^\dagger A_t^T + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right) \right] / \rho_t^2 \\
&\leq \eta_t \rho_t \left[\left((\tilde{\varepsilon}_t^\alpha)^2 + e_t^2 \right) \left(1 + K_t^T (K_t K_t^T)^\dagger A_t^T \right) \right. \\
&\quad \left. - 2e_t^2 \left(1 + K_t^T (K_t K_t^T)^\dagger A_t^T \right) + \eta_t^2 e_t^2 \|K_t^T + A_t\|^2 \right] / \rho_t^2 \\
&\leq -\frac{\eta_t}{\rho_t} (1 + \Gamma_t A_t^T) \left[-(\tilde{\varepsilon}_m^\alpha)^2 + \left(1 - \frac{\|K_t^T + A_t\|^2}{\rho_t (1 + \Gamma_t A_t^T)} \right) e_t^2 \right]
\end{aligned} \tag{5.22}$$

From the definitions of η_t and ρ_t in (5.15) and (5.16), we can obtain

$$\frac{\eta_t}{\rho_t} \left[-(\tilde{\varepsilon}_m^\alpha)^2 + \left(1 - \frac{\|K_t^T + A_t\|^2}{\rho_t (1 + \Gamma_t A_t^T)} \right) e_t^2 \right] > 0$$

To make the $\|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \leq 0$, $1 + \Gamma_t A_t^T$ in the last inequality of (5.22) should not be negative. The non-negative of the formula $1 + \Gamma_t A_t^T$ is realized by our proposed sparsification scheme.

We show that the proposed RKOL algorithm can be naturally derived from the weight convergence condition and yields a similar sparsification condition that is

connected to the ALD condition.

Define the Gram kernel matrix G_t as

$$G_t = \begin{bmatrix} k(c_{t,1}, c_{t,1}) & \cdots & k(c_{t,1}, c_{t,m_t}) \\ k(c_{t,1}, c_{t,2}) & & \vdots \\ \vdots & \ddots & k(c_{t,m_t-1}, c_{t,m_t}) \\ k(c_{t,1}, c_{t,m_t}) & \cdots & k(c_{t,m_t}, c_{t,m_t}) \end{bmatrix} \quad (5.23)$$

Rewrite the formula

$$\begin{aligned} 1 + \Gamma_t A_t^T &= 1 + K_t^T (K_t K_t^T)^\dagger A_t^T \\ &= 1 + K_t^T \left[\psi(x_t) \psi(c_{t-1}) \psi(c_{t-1})^T \psi(x_t)^T \right]^\dagger A_t^T \\ &= 1 + K_t^T \left[\psi(x_t) G_{t-1} \psi(x_t)^T \right]^\dagger A_t^T \\ &= 1 + K_t^T \left[\phi(x_t) G_{t-1} \phi(x_t) \right]^\dagger A_t^T \\ &= 1 + K_t^T (G_{t-1})^\dagger A_t^T \\ &= 1 - K_t^T (G_{t-1})^\dagger K_t + \Delta_t \end{aligned} \quad (5.24)$$

where $\psi(x_t) = \text{diag}[\phi(x_t), \dots, \phi(x_t)]$ $\psi(c_{t-1}) = [\phi(c_{t-1,1}), \dots, \phi(c_{t-1,m})]^T$, and

$$\Delta_t = K_t^T (G_{t-1})^\dagger [K_t + A_t^T]. \quad (5.25)$$

From the weight convergence analysis, we propose a sparsification condition as follows:

1. Pruning if $\Delta_t < \Delta_{tl}$: Delete the datum corresponding to the smallest item in Δ_t until $1 + \Gamma_t A_t^T \geq 0$;
2. Add a new datum into dictionary if $\Delta_t > \Delta_{tup}$.

Compare equation (III.4) in [70] and equation (14) in [72] with the last inequality of (5.24), they are of the form of ALD condition. The sparsification procedure dynamically eliminates the kernel functions in the network feedforward path which

contributes the negative value in a recurrent stability gain parameter specifically derived from the weight convergence condition until $1 + \Gamma_t A_t^T \geq 0$. From (5.4), we can see that the output is the linear combination of the kernel functions. The contribution of each kernel to Δ_t can be evaluated separately by separating Δ_t into two parts. One is $K_t^T(G_{t-1})^\dagger \in R^{m_t}$, the other is $(K_t + A_t^T) \in R^{m_t}$. It is obvious that $(K_t + A_t^T)$ evaluate the contribution of each kernel function. Consider $K_t^T(G_{t-1})^\dagger$ as the coefficients of the kernel functions as the weight vector α_t in (5.4). The elimination of the respective kernel function in the linear parameter updating processing is realized by deleting the kernels corresponding to the smallest item $\min \{ (K_t^T(G_{t-1})^\dagger)_i (K_t + A_t^T)_i, i = 1, \dots, m_t \}$ in Δ_t until $1 + \Gamma_t A_t^T \geq 0$. Furthermore, we will add an complementary growing procedure to select the new centers that carry main characteristics of the past time series information for future data prediction. The RKOL algorithm with the sparsification scheme is as follows:

-
- At time $t = 1$, start with m_t kernels, the weight and kernel centers are randomly initialized.
 - At $t = t + 1$, if $\Delta_t > \Delta_{tup}$

- $D_t^\alpha = [D_t^\alpha \text{ Col}[0]]$
- $\alpha_t = [\alpha_t \ 0]$
- $m_t = m_t + 1$

else if $\Delta_t < \Delta_{tl}$

- Find index i that is the smallest item in $(K_t^T(G_{t-1})^\dagger)_i (K_t + A_t^T)_i, i = 1, \dots, m_t$
- Delete α_i
- Delete the i th column of D_t^α
- $m_t = m_t - 1$

else, set $m_t = m_t$ and ignore the current input datum as in the dictionary.

- Update $K_t, K_t'', D_t^\alpha, A_t$.
- Update

$$\alpha_{t+1} = \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + A_t)$$

and then return to the second step.

5.3 Experimental Verification

In this section, we experimentally test the RKOL algorithm for two benchmark problems. The performance comparison of RKOL with two other on-line learning algorithms (KNLMS [78] and NORMA [127]) are also presented. The learning accuracy measurement is the root mean square error (RMSE) which is defined as $\text{RMSE} = \sqrt{\frac{\sum_{t=1}^n (y_t - \hat{y}_t)^2}{n}}$.

5.3.1 Mackey-Glass Time Sequential Prediction

The Mackey-Glass time series is a chaotic time sequential data. It is widely recognized as one of the benchmark time series problems, which is generated from the following time delay ordinary differential equation:

$$\frac{dx(t)}{dt} = -bx(t) + \frac{ax(t-c)}{1+x(t-c)^{10}} \quad (5.26)$$

with $b = 0.1$, $a = 0.2$, $c = 30$. The time series is discretized at a sampling period of 6 seconds. The prediction problem uses the previous ten data to predict the present one, therefore the input vector can be formulated as $x_t = [y_{t-1}, \dots, y_{t-10}]$. A segment of 2000 samples is used as training data and the subsequent 200 points as testing data.

For RKOL, we start from 50 neurons and the dictionary centers were randomly initialized. If the training step is less than 50, the input vector at each training step was considered as a new center and did sparsification at the same time according to the given condition. If the training step is bigger than 50, the input vector was considered as a new center only if the given condition was satisfied and sparsification was also performed at the same time. The optimal parameters are found as shown in Table 5.1.

The same criterion as RKOL was followed to initialize and test NORMA and KNLMS. The preliminary experiments conducted led to the parameter settings displayed in Table 5.1, where the same notations as in [78, 127] are used. The Gaussian kernel was considered which was parameterized by its width hyperparameter $\sigma = 1$. Table 5.1 also shows the dictionary size and the testing RMSE. It is clear that RKOL outperforms NORMA and KNLMS in terms of testing RMSE. NORMA performs worst because it takes the t nearest samples instead of the most informative ones to predict the present point as in RKOL. NORMA evaluates the cumulative loss as shown in equation (6) in [127]. However, RKOL evaluate the instant error as shown in (5.7). KNLMS can get comparable prediction performance as RKOL. However, RKOL needs more CUP time because of the slow sparsification procedure and arrives at more number of neurons. The merit of this work is that our RKOL provide mathematical proof of weight convergence. From the convergence condition (5.15) derived from (5.22), we can see that the weight will be updated only if the convergence condition is satisfied.

Fig. 5.1 shows the comparison of desired and predicted testing data of Mackey-Glass time-series trained by NORMA, KNLMS and RKOL respectively. Fig. 5.2 compares RMSE at each training step by using the three training methods. Fig. 5.3 shows the trace of L_2 norms of the weights at each step. It is seen that the L_2 norms of the weights of NORMA still oscillate after 2000 training steps, but L_2 norms of the weights of RKOL is stable after much less training steps. The percentage of η_t that is not zero over the 2000 training step is 0.7385. This agrees with our theoretical analysis that the weight is only updated when the convergence condition is satisfied.

The trace of normalization factor ρ is displayed in Fig. 5.4.

Table 5.1: Experimental Setup and Performance of Independent Test Sequences for Mackey-Glass Time-series.

Algorithms	Computational Time(s)	RMSE of testing error	Parameter setting	No. of Neurons
RKOL	3.4480	0.0958	$\Delta_{tup} = 1.12, \Delta_{tl} = -0.05$ $\eta = 0.3$	34
NORMA	2.0572	0.2575	$\lambda = 0.01, \eta = 0.72$	30
KNLMS	1.7985	0.1120	$\mu_0 = 0.8, \eta = 0.05, \epsilon = 0.16$	18

5.3.2 Chaotic Laser Time-series Prediction

The data comes from the Santa Fe time-series competition which is particularly difficult to predict, due both to its chaotic dynamics and to the fact that only three “intensity collapse” events occur in the training set [131]. The training data consists of 750 samples, and the testing data being the subsequent 100 sample. We normalized the series so that their value lined in the range $[0,1]$. The free parameters of the algorithm were tuned the same way as that in Section 5.3.1.

The optimal parameters are listed in 5.2. The preliminary experiments conducted led to the parameter settings displayed in Table 5.2, where the same notations as in [78,127] are used. Table 5.2 also shows the dictionary size and the testing RMSE. It is shown that RKOL can get smallest testing error with least number of neurons compared with KNLMS and NORMA.

Fig. 5.5 shows the comparison of desired and predicted testing data of laser time-series trained by NORMA, KNLMS and RKOL respectively. It can be seen that RKOL performs the best. Fig. 5.6 compares RMSE at each training step by using the three training method. Fig. 5.7 shows the trace of L_2 norms of the weights at each step. The percentages of η_t that is not zero over the 1000 training step is 0.8530. The trace of normalization factor ρ is displayed in Fig. 5.8.

(a) NORMA

(b) KNLMS

(c) RKOL

Figure 5.1: Comparison of desired and predicted testing data of Mackey-Glass time-series by NORMA, KNLMS and RKOL.

Figure 5.2: Learning curve for RKOL on testing data of Mackey-Glass time-series.

Table 5.2: Experimental Setup and Performance of Independent Test Sequences for Laser Time-series.

Algorithms	Computational Time(s)	RMSE of testing error	Parameter setting	No. of Neurons
RKOL	1.0920	0.0352	$\Delta_{tup} = 1.275, \Delta_{tl} = 0.695$ $\eta = 1.2$	25
NORMA	0.5716	0.1015	$\lambda = 0.008, \eta = 0.9$	28
KNLMS	0.6062	0.0749	$\mu_0 = 0.9, \eta = 0.03, \epsilon = 0.01$	28

5.4 Summary

The main contribution in this chapter is the establishment in the recurrent kernel online learning algorithms of its kind, an inherent link between the popular online kernel and recurrent learning via implementation of the proposed recurrent kernel online learning. The key idea of RKOL is to treat the kernel online training algorithm as a recurrent network, in which the output depends not only on the current external input to the network, but also on the current or previous outputs. For sparsification purpose, we proposed a sparsification procedure which is developed from system stability point of view. It can automatically eliminate the respective kernel

Figure 5.3: Full trace of L_2 norms of weight for Mackey-Glass time-series.

Figure 5.4: Full trace of normalization factor ρ_t .

(a) NORMA

(b) KNLMS

(c) RKOL

Figure 5.5: Comparison of desired and predicted testing data of laser time-series by NORMA, KNLMS and RKOL.

Figure 5.6: Learning curve for RKOL on testing data of laser time-series.

Figure 5.7: Learning curve for RKOL on testing data of laser time-series.

Figure 5.8: Full trace of normalization factor ρ_t for laser time-series.

function according to the weight convergence and stability condition. The RKOL is compared with KNLMS and NORMA. Simulation results shows that RKOL outperforms KNLMS and NORMA in terms of testing RMSE but needs more CPU time and neuron numbers.

Chapter 6

Recurrent Kernel Online Learning

6.1 Introduction

In Chapter 5, we propose a RKOL algorithm based on the celebrated RTRL that exploits the kernel trick in a recurrent online training manner. The merit of the RKOL is that it not only ensures weight convergence but also yields kernel sparsification conditions that can automatically eliminate or stop the growth of the kernel function. However, even though it can achieve smaller RMSE compared with other algorithms, it needs relatively longer computational time which is due to the time-consuming sparsification procedure. In this chapter, we propose an improved recurrent kernel online learning algorithm (IRKOL) with a coherence-based sparsification rule to reduce the computational complexity of RKOL. Furthermore, we integrate the recurrent weight convergence condition with a greedy algorithm to eliminate the need of the inverse of the kernel Gram matrix as shown in (5.25) to further improve the computational time.

Note that the same notations and definitions as those in Chapter 5 will not repeat in this chapter and please refer to the previous chapter if necessary.

6.2 Motivation and the IRKOL algorithm

6.2.1 The Existing ALD and Coherence Sparsification Schemes

Online kernel learning requires normally a regularized optimization to control the size of the network. In particular, the bottleneck problem of kernel-based online learning algorithms such as adaptive filters and nonlinear time series prediction, which refers to how the structure (complexity) increases with each new datum, must be solved. Active data selection (i.e., retaining important data and removing redundant data) is a natural approach to control the size of the network. In addition to the recently developed information theoretic approach, which can be viewed as an extension of approximate linear dependency (ALD) [72], two primary existing criteria in the online kernel learning literature are: the novelty criterion (NC) proposed by Platt in resource-allocating networks and the ALD test introduced by Engel for KRLS [70]. The prediction variance criterion is similar to ALD related sparsification condition, where a new kernel function $k(., x_t)$ at step t is inserted into the dictionary if the following condition is satisfied [70, 72]

$$1 - K_t^T G_t^{-1} K_t \geq \zeta \quad (6.1)$$

where ζ is a positive number that determines the level of sparsity of the model, and G_t is the kernel Gram matrix.

Because evaluation of the ALD sparsification condition, especially for the inverse of G_t , is very expensive in (6.1), Richard et al [78] proposes a simplified coherence criterion that is an approximation to ALD related sparsification condition (6.1); the coherence sparsification criterion is given in the following equation.

$$\max |k(c_{t,i}, x_t)| \leq \mu_0 \quad (6.2)$$

where $1 \leq i \leq m_t$, $\mu_0 \in (0, 1)$ is a positive parameter that determines both the level of sparsity and the coherence of the dictionary.

Based on ALD or coherence related sparsification conditions, the selected feature

sequence $\{\phi(c_{t,i})\} 1 \leq i \leq m_t$ forms the dictionary with the linearly independent centers in the feature space for popular kernel online learning algorithms. A major advantage of the coherence-based rule is that it is simpler and far less time consuming than ALD-based rule [78].

6.2.2 The IRKOL Algorithm

We present a new recurrent kernel online learning algorithms to naturally translate kernel online learning into recurrent learning in an RTRL manner with less computational time compared with RKOL. We first discuss the case in which the structural risk function is determined using recurrent constraint learning and focus on a cost function that determines the empirical risk function regulated by a recurrent sparsification scheme. Note that we base our method on the popular kernel least-squares growing algorithm [72,75,127], in which the number of kernel functions increases by one at each iteration of the proposed IRKOL algorithm if the sparsification and weight convergence conditions (specified below) are met simultaneously. This requires that we limit the complexity of the network or the number of kernel functions to reduce the computational cost of the algorithm and, more importantly, to improve the generalization performance to avoid the over-fitting problem.

For the reduced data dictionary, the regularized loss function differential is

$$\begin{aligned} & \left. \frac{dE_t}{d\alpha_t} \right|_{\lambda_t} \\ &= \frac{1}{2} \left[\frac{\partial (e_t^2)}{\partial \alpha_t} + \lambda_t \frac{\partial (e_t^2)}{\partial K_t} \frac{\partial K_t}{\partial \alpha_t} \right] \\ &= -e_t (K_t^T + \lambda_t \alpha_t \bar{K}_t D_t^\alpha) \end{aligned} \tag{6.3}$$

where $\bar{K}_t$ and D_t^α is defined in (5.13) and (5.14). We design an adaptive recurrent hyper-parameter λ_t with switch on-off condition for the recurrent training signal such that the method not only ensures weight convergence but also yields kernel sparsification conditions that can automatically eliminate or stop the growth of the kernel function, minimizing the system structural risk in online learning manner.

$\lambda_t = \lambda_t^o K_t^T K_t$ is defined according to the weight convergence and sparsification conditions as we show below in Section 6.3 with

$$\begin{cases} \lambda_t^o = \frac{1}{\lambda_0 + \Gamma_t} & \text{if } \mu_0(m_t - 1) < 1 \\ \lambda_t^o = 0 & \text{if } \mu_0(m_t - 1) \geq 1 \end{cases} \quad (6.4)$$

Note that the switch on-off condition $\mu_0(m_t - 1)$ is selected such that an optimal or suboptimal compromise between the weight convergence and approximation linear dependent sparsification like the coherence condition (6.2) is achieved. λ_0 is a positive constant and $\Gamma_t = \frac{\|K_t\|_1 \|G_t\|_{1,1} \|A_t\|_{1,1}}{1 - \mu_0(m_t - 1)}$, where we use $\|\cdot\|_1$ to denote the l_1 vector norm and $\|\cdot\|_{1,1}$ to denote the l_1 matrix norm. Note also that $\|\cdot\|$ denotes the l_2 norm by default in the rest of this chapter.

The IRKOL weight iterative equation is

$$\begin{aligned} \alpha_{t+1} &= \alpha_t - \frac{\eta_t}{\rho_t} \frac{dE_t}{d\alpha_t} \\ &= \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + \lambda_t \alpha_t \bar{K}_t D_t^\alpha) \\ &= \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + \lambda_t A_t) \end{aligned} \quad (6.5)$$

where $A(t)$ is defined in (5.12).

The adaptive normalization factor is defined to overcome the vanished cone problem in a classical adaptive training scheme like the RAGD [39]

$$\rho_t = \nu \rho_{t-1} + \max\{\bar{\rho}, \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\varsigma}\} \quad (6.6)$$

with $\nu > 0$, $\bar{\rho} > 0$ and $0 < \varsigma \leq (1 + \lambda_t^o K_t^T K_t A_t)$.

The classical adaptive learning rate is defined to gainst the weight drift in the iteration [130]

$$\begin{cases} \eta_t = \eta & \text{if } e_t \geq \tilde{\varepsilon}_m^\alpha / \sqrt{\gamma} \\ \eta_t = 0 & \text{if } e_t < \tilde{\varepsilon}_m^\alpha / \sqrt{\gamma} \end{cases} \quad (6.7)$$

in which the constants satisfy $\eta > 0$, $\tilde{\varepsilon}_m^\alpha > 0$ and $\gamma > 0$.

The IRKOL algorithm is similar to the popular kernel-based online training algorithms that increase the number of kernel functions by one in each step if the new data are not associated with the old data in terms of the ALD condition [72, 127]. We use the coherence-based kernel sparsification method, which is computationally more efficient compared with the standard ALD-based algorithm [72]. The IRKOL algorithm guarantees weight convergence and a lower computational time requirement compared with other kernel-based algorithms. The IRKOL algorithm with the coherence based sparsification scheme is as follows:

- At time $t = 1$, take the first pattern x_1 and initialize $c_{1,1}$, K_1 , D_1^α , $\bar{K}_1$, with $m_t = 1$.
- At $t = t + 1$, temporarily set $m_t = m_t + 1$ input a new online datum x_t . If $\max |k(c_{t,i}, x_t)| \leq \mu_0$ with $1 \leq i \leq m_t$, then take x_t into the dictionary as a future kernel center for the next growing step and perform the following assignments:

$$- D_t^\alpha = [D_t^\alpha \text{ Col}[0]]$$

$$- \alpha_t = [\alpha_t \ 0]$$

else, set $m_t = m_t$ and ignore the current input datum as in the dictionary.

- Update $K_t, \bar{K}_t, D_t^\alpha, A_t$.
- Update

$$\alpha_{t+1} = \alpha_t + \frac{\eta_t}{\rho_t} e_t (K_t^T + \lambda_t A_t) \quad (6.8)$$

and then return to the second step.

6.3 Weight Convergence and Sparsification - an Integrated Theoretical Analysis

We show that the proposed IRKOL algorithm can be naturally derived from the weight convergence condition and yields a similar sparsification condition that is connected to the coherence-based sparsification condition. Furthermore, we present some closed formulas for the sparsification scheme that can be derived from the weight convergence of the RBF-like recurrent network as integral expressions. By focusing on the Gram matrix embedded in the weight convergence proof, we provide explicit formulas suited for the design of the robust recurrent training algorithm.

To start the theoretical proof and analysis, we still follow the definitions of $e(t)$ in (5.17), $\tilde{\varepsilon}_t^\alpha$ in (5.18), e_t^α in (5.19) and $\tilde{\alpha}_t$ in (5.20).

Proposition 1: The weight of the kernel iterative equation (6.5) of the IRKOL algorithm converges if the inequalities

$$1 + \lambda_t^\alpha K_t G_t^{-1} A_t^T > 0 \quad (6.9)$$

$$\left[-(\tilde{\varepsilon}_m^\alpha)^2 + \left(1 - \frac{\|K_t^T + \lambda_t^\alpha A_t\|^2}{\rho_t(1 + \lambda_t^\alpha K_t^T G_t^{-1} A_t^T)} \right) e_t^2 \right] \geq 0 \quad (6.10)$$

are true.

Proof. By subtracting $\tilde{\alpha}_t^*$, squaring both sides of equation (6.5), we obtain

$$\begin{aligned} & \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\ &= \left[2\eta_t \rho_t e_t \tilde{\alpha}_t (K_t + \lambda_t A_t^T) + \eta_t^2 e_t^2 \|K_t^T + \lambda_t A_t\|^2 \right] / \rho_t^2 \end{aligned} \quad (6.11)$$

The key difference between the recurrent and feedforward neural network training algorithms is the recurrent training term A_t in equation (6.5), which is derived from the second term of the chain rule for the loss function (6.3). We can rewrite the

original weight convergence condition (6.11)

$$\begin{aligned}
 & \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\
 &= \left[2\eta_t \rho_t e_t \tilde{\alpha}_t (K_t + \lambda_t^\circ K_t K_t^\top A_t^\top) + \eta_t^2 e_t^2 \|K_t^\top + \lambda_t^\circ K_t^\top K_t A_t\|^2 \right] / \rho_t^2 \\
 &= \left[2\eta_t \rho_t e_t e_t^\alpha (1 + \lambda_t^\circ (K_t^\top A_t^\top) + \eta_t^2 e_t^2 \|K_t^\top + \lambda_t^\circ K_t^\top K_t A_t\|^2) \right] / \rho_t^2
 \end{aligned} \tag{6.12}$$

Note that the last equality of the above equation is derived from (5.19).

We have $e_t e_t^\alpha = \tilde{\varepsilon}_t^\alpha e_t - e_t^2$, and replacing $e_t e_t^\alpha$ with $\tilde{\varepsilon}_t^\alpha e_t - e_t^2$, we can rewrite (6.12) as

$$\begin{aligned}
 & \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\
 &= \left[2\eta_t \rho_t (\tilde{\varepsilon}_t^\alpha e_t - e_t^2) (1 + \lambda_t^\circ K_t^\top A_t^\top) + \eta_t^2 e_t^2 \|K_t^\top + \lambda_t^\circ K_t^\top K_t A_t\|^2 \right] / \rho_t^2 \\
 &\leq \left[\eta_t \rho_t ((\tilde{\varepsilon}_t^\alpha)^2 + e_t^2) (1 + \lambda_t^\circ K_t^\top A_t^\top) - 2\eta_t \rho_t e_t^2 (1 + \lambda_t^\circ K_t^\top A_t^\top) \right. \\
 &\quad \left. + \eta_t^2 e_t^2 \|K_t^\top + \lambda_t^\circ K_t^\top K_t A_t\|^2 \right] / \rho_t^2 \\
 &\leq -\frac{\eta_t}{\rho_t} (1 + \lambda_t^\circ K_t^\top A_t^\top) \left[-(\tilde{\varepsilon}_t^\alpha)^2 + \left(1 - \frac{\|K_t^\top + \lambda_t^\circ K_t^\top K_t A_t\|^2}{\rho_t (1 + \lambda_t^\circ K_t^\top A_t^\top)} \right) e_t^2 \right] \\
 &\leq 0
 \end{aligned} \tag{6.13}$$

where we assume that $\|\tilde{\varepsilon}_t^\alpha\| < \tilde{\varepsilon}_m^\alpha$ is a bounded uncertainty as introduced in (5.8) ■

The inverse of kernel Gram matrix G_t^{-1} plays an important role in the ALD and coherence related sparsification algorithms. Therefore, we can modify the first convergence condition $1 + \lambda_t^\circ K_t^\top A_t^\top > 0$ in Proposition 1 such that the inverse kernel matrix can be embedded into the equation and further linked to the coherence based sparsification condition (6.2). Note that this implies that we embed the kernel Gram matrix G_t and its inverse implicitly into the recurrent hyper-parameter $\lambda_t = \lambda_t^\circ K_t^\top A_t^\top$ such that the regularized loss function (6.3) can deal with the sparsification requirement.

Proposition 2: The first convergence condition

$$1 + \lambda_t^o K_t^T A_t^T = 1 + \lambda_t^o K_t^T G_t G_t^{-1} A_t^T > 0$$

in Proposition 1 is always true if the recurrent hyper-parameter λ_t^o can be defined as (6.4) and linked to the coherence parameter μ_0 .

Proof. According to the definition of λ_t^o in (6.4) $1 + \lambda_t^o K_t G_t^{-1} A_t^T > 0$ is true if $\mu_0(m_t - 1) \geq 1$, i.e. $\lambda_t^o = 0$ so we need only proof the case of $\mu_0(m_t - 1) < 1$ as the following.

The Gram kernel matrix G_t uses the Gaussian kernel with unit diagonal. We first rewrite the Gram kernel matrix G_t using an off-diagonal component $\bar{G}_t$ that satisfies

$$G_t = I_{m_t} + \bar{G}_t \quad (6.14)$$

By the definition of the Babel function $\mu_1(m_t - 1)$ and l_1 norm [78], [79] and let $\mu_0 \in (0, 1)$ be a positive number as defined in (6.2),

$$\mu_1(m_t - 1) = \|\bar{G}_t\|_{1,1} = \max_j \sum_{j \neq i} |k(c_{t,i}, c_{t,j})| < \mu_0(m_t - 1) \quad (6.15)$$

Note that the Babel function measures the maximum total coherence between a fixed center and a collection of others. However, due to the calculation complexity of $\mu_1(m_t - 1) < 1$ we may also use a more restrictive condition $(m_t - 1)\mu_0 < 1$ to replace it to reduce the computational cost [78]. Furthermore, it was argued in [78] that the m_t kernel functions $k(\cdot, c_{t,1}), \dots, k(\cdot, c_{t,m_t})$ from a dictionary become linearly independent if the Babel function $\mu_1(m_t - 1) < 1$. We need, however, only a more restrictive condition $(m_t - 1)\mu_0 < 1$ to define the recurrent hyper-parameter λ_t to ensure the weight convergence condition because the coherence sparsification condition like the ALD sparsification condition in (6.1) is just an approximation linear independent condition which may need to eliminate more input data than the true linear independent condition for an effective sparsification.

As we have $\lambda_t = 1$, i.e. $(m_t - 1)\mu_0 < 1 \Rightarrow \|\bar{G}_t\|_{1,1} < 1$ as shown in (6.15), then, we

can further extend the inverse of the kernel Gram matrix as a von Neumann series and use Banach algebra methods to estimate its norm. Note that the von Neumann series $\sum_{k=0}^{\infty} (-\bar{G}_t)^k$ under the inequality condition (6.15) converges to the inverse $[I_{m_t} + \bar{G}_t]^{-1}$ [79, 132]. Thus,

$$\begin{aligned}
 & \|G_t^{-1}\|_{1,1} \\
 &= \|[I_{m_t} + \bar{G}_t]^{-1}\|_{1,1} \\
 &= \left\| \sum_{k=0}^{\infty} (-\bar{G}_t)^k \right\|_{1,1} \\
 &\leq \sum_{k=0}^{\infty} \|\bar{G}_t\|_{1,1}^k = \frac{1}{1 - \|\bar{G}_t\|_{1,1}} \\
 &\leq \frac{1}{1 - \mu_0(m_t - 1)}
 \end{aligned} \tag{6.16}$$

According to (6.16) we can further develop

$$\begin{aligned}
 |\lambda_t^o K_t^T A_t^T| &= |\lambda_t^o K_t^T G_t G_t^{-1} A_t^T| \\
 &\leq \frac{\lambda_t^o \|K_t\|_1 \|G_t\|_{1,1} \|A_t\|_{1,1}}{(1 - \mu_0(m_t - 1))} \\
 &= \frac{\|K_t\|_1 \|G_t\|_{1,1} \|A_t\|_{1,1} / (1 - \mu_0(m_t - 1))}{\lambda_0 + \|K_t\|_1 \|G_t\|_{1,1} \|A_t\|_{1,1} / (1 - \mu_0(m_t - 1))} < 1 \\
 &\Rightarrow 1 + \lambda_t^o K_t^T A_t^T > 0
 \end{aligned} \tag{6.17}$$

■

Using Proposition 2, if the equivalent of ALD related sparsification condition, i.e. the coherence sparsification criterion (6.2) is true, the first convergence condition (6.9) in Proposition 1 is satisfied. Up to proof of (6.9), we can rewrite the original proof in Proposition 1 to meet the second convergence condition (6.10).

Proposition 3: We can rewrite the original weight convergence proof in (6.13) of Proposition 1 as:

$$\begin{aligned}
 & \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\
 & \leq -\frac{\eta_t}{\rho_t} (1 + \lambda_t^o K_t^T A_t^T) \left[-(\tilde{\varepsilon}_m^\alpha)^2 + \gamma e_t^2 \right] \leq 0
 \end{aligned} \tag{6.18}$$

Proof. According (6.17) we can find positive constants $\varsigma > 0$ and $1 > \gamma > 0$ to have

$$0 < \varsigma < (1 + \lambda_t^o K_t^T A_t^T) \quad (6.19)$$

Then

$$\begin{aligned} & - \left(1 - \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\rho_t(1 + \lambda_t^o K_t^T A_t^T)} \right) \\ & < - \left(1 - \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\varsigma \rho_t} \right) \\ & = - \left(1 - \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\varsigma(\nu \rho_{t-1} + \max[\bar{\rho}, \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\varsigma}])} \right) \leq -\gamma \end{aligned} \quad (6.20)$$

We can use the sparsification scheme and the definitions of the specific learning rate η_t in (6.7) and the normalization factor ρ_t in (6.6) to show that the weight convergence proof (6.13) in Proposition 1 becomes

$$\begin{aligned} & \|\tilde{\alpha}_{t+1}\|^2 - \|\tilde{\alpha}_t\|^2 \\ & \leq -\frac{\eta_t}{\rho_t}(1 + \lambda_t^o K_t^T A_t^T) \\ & \quad \times \left[-(\tilde{\varepsilon}_m^\alpha)^2 + \left(1 - \frac{\|K_t^T + \lambda_t^o K_t^T K_t A_t\|^2}{\rho_t \varsigma} \right) e_t^2 \right] \\ & \leq -\frac{\eta_t}{\rho_t}(1 + \lambda_t^o K_t^T A_t^T) \left[-(\tilde{\varepsilon}_m^\alpha)^2 + \gamma e_t^2 \right] \leq 0 \end{aligned} \quad (6.21)$$

Note that the last inequality is from (6.20).

Furthermore, using (6.18) we can conclude that the estimate error is bounded by [39]

$$\limsup_{t \rightarrow \infty} \|e_t\| \leq \tilde{\varepsilon}_m^\alpha / \sqrt{\gamma} \quad (6.22)$$

■

Remark 8. Through the proof of Proposition 3, now we can justify the link between the important recurrent hyper-parameter λ_t^o defined in (6.4) with the selection of

the more restrictive condition for its switch function to cut the recurrent training signal for some situation, i.e. set $\lambda_t^o = 0$ in case of $\mu_0(m_t - 1) \geq 1$, i.e. turn the structural risk function (6.3) temporarily into an empirical risk function to minimize the estimate error in case of $\lambda_t^o = 0$. As we mentioned in the proof of Proposition 2, the Babel function $\mu_1(m_t - 1) < 1$ was proved in [78] as a sufficient condition for the linear independence of the kernel functions $k(\cdot, c_{t,1}), \dots, k(\cdot, c_{t,m_t})$. However, what we need is a more restrictive sparsification scheme like the coherence and ALD related conditions. Therefore, in addition to the computational complexity problem as mentioned in the proof of Proposition 2, we can further show that if one chooses the Babel function $\mu_1(m_t - 1)$ as defined in (6.15) for the switch on-off conditions of the recurrent hyper-parameter λ_t^o in (6.4), since $\mu_1(m_t - 1) < \mu_0(m_t - 1)$ as in (6.15), which will lead to a smaller ς in (6.19) and a larger γ in (6.20). This results in a relatively smaller estimate error bound $\tilde{\varepsilon}_m^\alpha / \sqrt{\gamma}$ as from (6.22) of the Proposition 3. This implies that we should select a relatively restrictive sparsification scheme like the ALD or coherence related condition rather than the pure linear independent condition like the Babel function $\mu_1(m_t - 1) < 1$, which incidently coincides with the requirement of reduced computational complexity as mentioned in Proposition 2 and [78]. Therefore, according to the design of the adaptive recurrent hyper-parameter λ_t^o in (6.4) the IRKOL algorithm automatically weights the regularized second term in the structural risk minimization function (6.3) in terms that we can minimize not only the estimate error e_t but also improve the generalization performance via sparsification. This theoretical claim can further proved in the following simulation section.

6.4 Simulations

In this section, we experimentally test the IRKOL algorithm. A performance comparison of IRKOL with KNLMS, NORMA and RKOL is also presented. The learning accuracy is quantified using the root-mean-square error (RMSE), which is defined as $\text{RMSE} = \sqrt{\frac{\sum_{t=1}^n (d_t - y_t)^2}{n}}$.

Table 6.1: Experiment A: Experimental Setup and Performance of Independent Test Sequences for the Mackey-Glass Time Series.

Algorithms	Computational Time(s)	RMSE of testing error	Parameter setting	No. of Neurons
IRKOL	1.6576	0.0883	$\mu_0 = 0.8, \eta = 0.3,$	18
RKOL	3.4480	0.0958	$\Delta_{tup} = 1.12, \Delta_{tl} = -0.05$ $\eta = 0.3$	34
NORMA	2.0572	0.2575	$\lambda = 0.01, \eta = 0.72$	30
KNLMS	1.7985	0.1120	$\mu_0 = 0.8, \eta = 0.05, \epsilon = 0.16$	18

6.4.1 Mackey-Glass time-sequential prediction

We test the IRKOL on the Mackey-Glass equation as used in Section 5.3.1. The parameter settings displayed in Table 6.1 were used, where the notation is the same as in [78,127]. From Table 6.1, we can see that IRKOL is much faster than RKOL due to the coherence-based sparsification rule. The analysis of the experiment follows the same idea as that in Section 5.3.1 and will not be repeated here. Fig. 6.1 shows a comparison of the desired and predicted testing data for the Mackey-Glass time series trained using NORMA, KNLMS and IRKOL, respectively. The comparison of KNLMS, NORMA and RKOL is shown in Fig. 5.1. Fig. 6.2 compares the RMSE values of the four training methods at each training step. Fig. 6.3 shows the trace of the L_2 norms of the weights at each step, which shows that IRKOL is superior in terms of smaller testing error (generalization performance) as compared with other three kernel learning algorithms. The fraction of non-zero adaptive learning rate over the 2000 training steps is 0.5819. The trace of normalization factor ρ_t is displayed in Fig. 6.4. As we can see, the fast weight convergence and good generalization property of IRKOL are achieved as compared with three other existing kernel learning algorithms.

6.4.2 Chaotic laser time-series prediction

We test the IRKOL on the chaotic laser time-series as used in Section 5.3.2. To compare IRKOL with NORMA and KNLMS, preliminary experiments were con-

(a) NORMA

(b) KNLMS

(c) IRKOL

Figure 6.1: A comparison of the desired and predicted testing data for the Mackey-Glass time series trained using NORMA, KNLMS and IRKOL.

Figure 6.2: The learning curve for KNMLS, NORMA, RKOL and IRKOL applied to the testing data of the Mackey-Glass time series.

Table 6.2: Experiment B: Experimental Setup and Performance for Independent Test Sequences for the Laser Time Series.

Algorithms	Computational Time(s)	RMSE of testing error	Parameter setting	No. of Neurons
IRKOL	0.5063	0.0553	$\mu_0 = 0.9, \eta = 0.317$	28
RKOL	1.0920	0.0352	$\Delta_{tup} = 1.275, \Delta_{tl} = 0.695$ $\eta = 1.2$	25
NORMA	0.5716	0.1015	$\lambda = 0.008, \eta = 0.9$	28
KNLMS	0.6062	0.0749	$\mu_0 = 0.9, \eta = 0.03, \epsilon = 0.01$	28

ducted to determine the parameter settings, which are given in Table 6.2, in which the same notation as in Table 6.2 also presents the dictionary size and the testing RMSE. The performance of RKOL is better than the other three algorithms but the computational time is much longer as discussed. IRKOL can get comparable RMSE with RKOL on this time-series but with much less computational complexity. Fig. 6.5 shows a comparison of the desired and predicted testing data for the laser time series by NORMA, KNLMS and IRKOL. Fig. 6.6 compares the RMSE values for the four training methods at each training step. Fig. 6.7 shows the trace of the l_2 norms of the weights at each step. The norms of the weights does not change significantly after step 625. The fraction of the 750 training steps for which $\eta_t = \eta$ is 0.5865. The trace of the normalization factor ρ_t is displayed in Fig. 6.8.

Figure 6.3: The full trace of the L_2 norms of the weights for the Mackey-Glass time series.

Figure 6.4: The full trace of the normalization factor ρ_t .

(a) NORMA

(b) KNLMS

(c) IRKOL

Figure 6.5: A comparison of the desired and predicted testing data for the laser time series trained using NORMA, KNLMS and IRKOL.

Figure 6.6: The learning curve for IRKOL applied to the testing data of the laser time series.

6.5 Summary

We proposed a new class of kernel learning algorithms named robust recurrent kernel online learning (IRKOL). This novel type of algorithm combines the advantages of both kernel and recurrent learning methods. A structural risk minimization cost function that includes a recurrent learning feedback term was developed. Furthermore, the weight convergence condition was proved and connected to the kernel sparsification scheme using an integrated approach. The fast weight convergence and good generalization property of IRKOL are achieved as compared with other three existing kernel learning algorithms in experimental results.

Figure 6.7: The full trace of the L_2 norms of the weights for the laser time series.

Figure 6.8: The full trace of the normalization factor ρ_t for the laser time series.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The training algorithm design and stability analysis are two major fields in the area of RNNs design. The major research results covered in this thesis can be summarized as follows.

- A MRAGD training method for RNNs was proposed. It was used in the design of robust RNNs tracking controller for a class of MIMO nonlinear dynamical systems with arbitrary square input-output dimensions. Our proposed MRAGD training algorithm for RNN based MIMO controller includes a specially designed recurrent training term in calculation of the gradient to explore the inherent dynamic structure of RNNs. Furthermore, the adaptive learning rate, dead zone and recurrent adaptive hybrid parameter concepts were used in the MRAGD training of MIMO system to achieve fast transient control system response, guaranteed stability and weight convergence. The conditions for weight convergence and stability were given in the sense of Lypunov function. A complete stability analysis was given and it was proven that the tracking error was bounded under the neural controller in the presence of disturbance. Simulation results showed that the proposed MRAGD based neural controller

performed better than the neural controllers based on the standard BP algorithm and RTRL algorithm.

- A new training algorithm of RNNs based on SPSA method was proposed following deterministic approach. The key characteristic of RRSPSA training algorithm was to use a recurrent hybrid adaptive parameter together with adaptive learning rate and dead zone concept. The recurrent hybrid adaptive parameter can automatically switch off recurrent training signal whenever the weight convergence and stability conditions are violated. There are several advantages to train RNNs based on RRSPSA algorithm. First, RRSPSA is relatively easy to implement as compared to other RNN training methods such as BPTT due to its RTRL type of training and simplicity. Second, RRSPSA provides excellent convergence property based on simultaneous perturbation of weight, which is similar to the standard SPSA. Finally, RRSPSA has capability to improve RNN training speed with guaranteed weight convergence over the standard RTRL algorithm by using adaptive learning method. RRSPSA is a powerful twin-engine SPSA type of RNN training algorithm. Robust stability and weight convergence proofs of RRSPSA algorithm were provided based on Lyapunov function. Two experiments were carried out to verify the derived theoretical results, which show clearly that the learning performance of RRSPSA is improved in terms of convergence speed and mean squared error as compared to that of the conventional RNN algorithms.
- An online recurrent kernel learning algorithm for RNNs was proposed. RKOL algorithm based on the celebrated RTRL that exploits the kernel trick in a recurrent online training manner. The merit of the RKOL is that it not only ensures weight convergence but also yields kernel sparsification conditions that can automatically eliminate or stop the growth of the kernel function. It can automatically eliminate the respective kernel function according to the weight convergence and stability condition to minimize the system structural risk in

an online fashion. We eliminated the respective kernel function in the linear parameter updating processing by deleting the kernels with negative contributions. Furthermore, we added an complementary growing procedure to select the centers that carries main characteristics of the past time series information for future data prediction. The RKOL was compared with KNLMS and NORMA. Simulation results showed that RKOL outperforms KNLMS and NORMA in terms of testing RMSE but needs more CPU time and neuron numbers.

- The IRKOL algorithm was proposed to reduce the computational time of the training algorithm while guaranteeing weight convergence. We first cover the case where the structural risk functional was formed with the recurrent constraint learning, and focused on the cost function which took into account the empirical risk function with recurrent sparsification scheme. The risk function that we define was a structural risk minimization which minimized the training error in the conventional cost function and was also integrated with the sparsification process. We showed that such a sparsification had naturally derived from the recurrent constrained learning and led to the similar sparsification condition linked to the ALD and coherence based conditions. Furthermore, we presented some closed formulas of the sparsification scheme that could be derived from the weight convergence of the RBF-like recurrent network in integral expressions. By focusing further on the Gram matrix embedded in the weight convergence proof, we provided explicit formulas suited for design of the robust recurrent training algorithm and preliminary experiments to illustrated the new approach.

7.2 Future Work

Based on the research work conducted in this thesis, I find several promising directions which are to be investigated in my future research study as follows:

- To improve the performance of our designed learning algorithms mentioned above, an optimal or suboptimal tradeoff is needed between training and testing errors. Therefore, we will design adaptive dead zone training algorithms for an MIMO system under different noisy environment. The adaptive dead zone scheme is similar to the adaptive filtering and prediction system, but with a new nonlinear system extension [133] which can make the RNN learning automatically switch on and off in an optimal or suboptimal manner compared to the fixed dead zone scheme. It will lead to an optimal compromise between training and testing errors, which particularly benefits the weight convergence and generalization performance of RNNs. In our thesis, we use predefined fixed deadzones which are not updated in adaptive way. Actually, there have been some preliminary results regarding adaptive deadzone gradient-descent based training of RNNs which have shown promising performance [130]. Next step, we will try to improve our algorithm by combining the adaptive dead zone scheme.
- The recent development of kernel based training algorithms combined with sparsification rule for RNNs provides a unified learning platform with widespread type of feature mappings and can be applied in regression and multi-class classification applications directly. In addition, simulation results showed that the kernel based training algorithm with adapting neural network structure is more efficient in terms of generalization ability and computational complexity than other computational intelligence techniques. However, a training algorithm cannot fit for all the data set. Therefore different sparsification rules should be tested to further verify the performance of the proposed algorithm, there still need a large amount of simulations with huge data sets. We have implemented two main approaches, one is by elimination as in Chapter 5 and the other is by construction which starts from an empty network and gradually adds centers as in Chapter 6. There are some newly developed alternatives of sparsification, such as the quantization approach in [74]. To further improve

the performance of kernel methods and curb the growth of the radial basis function structure, diverse schemes should be investigated.

- The works in this thesis are all based on Jordan network which has an output-layer feedback structure. In Elman networks (ENs), recurrent feedback takes place in the hidden layer. An EN can be viewed as a feedforward NN with an additional set of inputs from the context layer (feedback from the hidden layer). Standard BP training algorithm is used to train ENs, however, it is not the most suitable one. A low learning rate can improve the training of ENs but can also result in very slow convergence speeds and poor generalization performance, whereas a high learning rate can lead to unstable training in terms of weight divergence. Therefore, an optimal or suboptimal tradeoff between training speed and weight convergence with good generalization capability is desired for ENs. Moreover, the generalization ability is affected by the size of the neural network, fast and stable training algorithms with adapting structure need to be developed for ENs.

Author's Publications

Journal papers:

1. Z. Xu, Q. Song, and D. W. Wang, "Recurrent Neural Tracking Control Based on Multivariable Robust Adaptive Gradient-Descent Training Algorithm," *Neural Computing and Applications*, pp. 1-11, 2010.
2. Z. Xu, Q. Song, and D. W. Wang, "A Robust Recurrent Simultaneous Perturbation Stochastic Approximation Training Algorithm for Recurrent Neural Networks," *Submitted*.
3. Q. Song, Z. Xu, H. J. Fan and D. W. Wang, "Recurrent Kernel Online Learning," *Submitted*.

Conference papers:

1. Z. Xu, Q. Song, D. W. Wang and H. J. Fan, "A Training Algorithm and Stability Analysis for Recurrent Neural Networks ," in *Proceedings of 15th International Conference on Information Fusion* , Singapore, 2012.
2. Z. Xu, Q. Song, H. J. Fan and D. W. Wang, " Online Prediction of Time Series Data with Recurrent Kernels," in *Proceedings of International Joint Conference on Neural Networks* , Brisbane, Australia, 2012.
3. Z. Xu, Q. Song and D. W. Wang, "Recurrent Neural Network Based Tracking Control," in *Proceedings of 11th International Conference on Control, Automation, Robotics and Vision*, Singapore, pp. 2454-2459, 2010.

Bibliography

- [1] H. A. Talebi, K. Khorasani, and S. Tafazoli, "A recurrent neural-network-based sensor and actuator fault detection and isolation for nonlinear systems with application to the satellite's attitude control subsystem," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 45–60, 2009.
- [2] Z. G. Hou, M. M. Gupta, P. N. Nikiforuk, M. Tan, and L. Cheng, "A recurrent neural network for hierarchical control of interconnected dynamic systems," *IEEE Transactions on Neural Networks*, vol. 18, no. 2, pp. 466–481, 2007.
- [3] R. K. Al Seyab and Y. Cao, "Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation," *Journal of Process Control*, vol. 18, no. 6, pp. 568–581, 2008.
- [4] Q. Song, J. Spall, Y. C. Soh, and J. Ni, "Robust neural network tracking controller using simultaneous perturbation stochastic approximation," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 817–835, 2008.
- [5] Q. Song, "On the weight convergence of elman networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 3, pp. 463–480, 2010.
- [6] D. Mandic and J. Chambers, *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley, New York, USA, 2001.
- [7] J. S. Wang and Y. P. Chen, "A fully automated recurrent neural network for unknown dynamic system identification and control," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 6, pp. 1363–1372, 2006.
- [8] W. Yu, "Nonlinear system identification using discrete-time recurrent neural networks with stable learning algorithms," *Information Sciences*, vol. 158, pp. 131–147, 2004.
- [9] J. Connor, R. Martin, and L. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, 1994.
- [10] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

- [11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *MIT Press Computational Models Of Cognition And Perception Series*, vol. 1, pp. 318–362, 1986.
- [12] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [13] —, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Back-propagation: Theory, architectures and applications*, pp. 433–486, 1995.
- [14] H. Zhang, Z. Wang, and D. Liu, "Global asymptotic stability of recurrent neural networks with multiple time-varying delays," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 855–873, 2008.
- [15] J. Cao, K. Yuan, and H. Li, "Global asymptotical stability of recurrent neural networks with multiple discrete delays and distributed delays," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1646–1651, 2006.
- [16] Y. He, M. Wu, and J. She, "An improved global asymptotic stability criterion for delayed cellular neural networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 250–252, 2006.
- [17] X. Liao, J. Wang, and Z. Zeng, "Global asymptotic stability and global exponential stability of delayed cellular neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 7, pp. 403–409, 2005.
- [18] J. Cao and J. Wang, "Global exponential stability and periodicity of recurrent neural networks with time delays," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 5, pp. 920–931, 2005.
- [19] Q. Zhang, X. Wei, and J. Xu, "On global exponential stability of delayed cellular neural networks with time-varying delays," *Applied Mathematics and Computation*, vol. 162, no. 2, pp. 679–686, 2005.
- [20] H. Zhang, C. Ji, and D. Liu, "Robust stability analysis of a class of hopfield neural networks with multiple delays," *Advances in Neural Networks-ISNN*, pp. 209–214, 2005.
- [21] V. Singh, "A generalized lmi-based approach to the global asymptotic stability of delayed cellular neural networks," *IEEE Transactions on Neural Networks*, vol. 15, no. 1, pp. 223–225, 2004.
- [22] J. Cao and J. Wang, "Global asymptotic stability of a general class of recurrent neural networks with time-varying delays," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 1, pp. 34–44, 2003.

- [23] S. Hu and J. Wang, "Absolute exponential stability of a class of continuous-time recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, pp. 35–45, 2003.
- [24] X. Liao and J. Wang, "Algebraic criteria for global exponential stability of cellular neural networks with multiple time delays," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 2, pp. 268–274, 2003.
- [25] X. Liao, G. Chen, and E. Sanchez, "Lmi-based approach for asymptotically stability analysis of delayed neural networks," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 49, no. 7, pp. 1033–1039, 2002.
- [26] S. Arik, "Stability analysis of delayed neural networks," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 47, no. 7, pp. 1089–1092, 2000.
- [27] S. Hu and J. Wang, "Global stability of a class of discrete-time recurrent neural networks," *IEEE Transactions on Circuit and Systems I: Fundamental Theory and Applications*, vol. 49, no. 8, pp. 1104 – 1117, 2002.
- [28] S. Arik, "An analysis of global asymptotic stability of delayed cellular neural networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1239–1242, 2002.
- [29] T. Liao and F. Wang, "Global stability for cellular neural networks with time delay," *IEEE Transactions on Neural Networks*, vol. 11, no. 6, pp. 1481–1484, 2000.
- [30] J. Zhang and X. Jin, "Global stability analysis in delayed hopfield neural network models," *Neural Networks*, vol. 13, no. 7, pp. 745–753, 2000.
- [31] H. Chunhai and Q. Jixin, "Stability analysis for neural dynamics with time-varying delays," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 221–223, 1998.
- [32] T. Roska, C. Wu, M. Balsi, and L. Chua, "Stability and dynamics of delay-type general and cellular neural networks," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 6, pp. 487–490, 1992.
- [33] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall, New Jersey, USA, 1999.
- [34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

- [35] S. C. Huang and Y. F. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 47–55, 1991.
- [36] V. Cherkassky, X. Shao, F. M. Mulier, and V. N. Vapnik, "Model complexity control for regression using vc generalization bounds," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1075–1089, 1999.
- [37] Y. W. Lu, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, vol. 9, no. 2, pp. 461–478, 1997.
- [38] J. G. Polhill and M. K. Weir, "An approach to guaranteeing generalisation in neural networks," *Neural Networks*, vol. 14, no. 8, pp. 1035–1048, 2001.
- [39] Q. Song, Y. Wu, and Y. C. Soh, "Robust adaptive gradient-descent training algorithm for recurrent neural networks in discrete time domain," *IEEE Transactions on Neural Networks*, vol. 19, no. 11, pp. 1841–1853, 2008.
- [40] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992.
- [41] C. Johansson, "A study of interacting bayesian recurrent neural networks with incremental learning," *NADA*, p. 71, 2001.
- [42] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [43] F. Scarselli and A. Chung Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results," *Neural Networks*, vol. 11, no. 1, pp. 15–37, 1998.
- [44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America-biological sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [45] —, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 81, no. 10, pp. 3088–3092, 1984.
- [46] F. J. Lin, R. J. Wai, and C. M. Hong, "Recurrent neural network control for lcc-resonant ultrasonic motordrive," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 47, no. 3, pp. 737–749, 2000.
- [47] C. M. Lin and C. F. Hsu, "Recurrent-neural-network-based adaptive-backstepping control for induction servomotors," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 6, pp. 1677–1684, 2005.

- [48] T. W. S. Chow and Y. Fang, "A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics," *IEEE Transactions on Industrial Electronics*, vol. 45, no. 1, pp. 151–161, 1998.
- [49] K. P. Venugopal, A. S. Pandya, and R. Sudhakar, "A recurrent neural network controller and learning algorithm for the on-line learning control of autonomous underwater vehicles," *Neural Networks*, vol. 7, no. 5, pp. 833–846, 1994.
- [50] M. K. Sundareshan and T. A. Condarcure, "Recurrent neural-network training by a learning automation approach for trajectory learning and control system design," *IEEE Transactions on Neural Networks*, vol. 9, no. 3, pp. 354–368, 1998.
- [51] W. Yu, "Multiple recurrent neural networks for stable adaptive control," *Neurocomputing*, vol. 70, no. 1-3, pp. 430–444, 2006.
- [52] W. Jeen-Shing and C. Yen-Ping, "A fully automated recurrent neural network for unknown dynamic system identification and control," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 6, pp. 1363–1372, 2006.
- [53] L. Tian and C. Collins, "A dynamic recurrent neural network-based controller for a rigid-flexible manipulator system," *Mechatronics*, vol. 14, no. 5, pp. 471–490, 2004.
- [54] M. M. Abdelhameed and F. A. Tolbah, "A recurrent neural network-based sequential controller for manufacturing automated systems," *Mechatronics*, vol. 12, no. 4, pp. 617–633, 2002.
- [55] M. I. Jordan, "Generic constraints on underspecified target trajectories," in *Proceedings of International Joint Conference on Neural Networks, Washington, DC, USA*, 1989.
- [56] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.
- [57] D. T. Pham and X. Liu, "Identification of linear and nonlinear dynamic systems using recurrent neural networks," *Artificial Intelligence in Engineering*, vol. 8, no. 1, pp. 67–75, 1993.
- [58] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, no. 19, pp. 2229–2232, 1987.
- [59] G. E. Hinton, "Deterministic boltzmann learning performs steepest descent in weight-space," *Neural Computation*, vol. 1, no. 1, pp. 143–150, 1989.
- [60] P. Baldi and F. Pineda, "Contrastive learning and neural oscillations," *Neural Computation*, vol. 3, no. 4, pp. 526–545, 1991.

- [61] B. A. Pearlmutter, *Dynamic recurrent neural networks*. Carnegie Mellon University, Computer Science Department, 1990.
- [62] ———, “Gradient calculations for dynamic recurrent neural networks: A survey,” *IEEE Transactions on Neural Networks*, vol. 6, no. 5, pp. 1212–1228, 1995.
- [63] E. Soria-Olivas, J. Calpe-Maravilla, J. F. Guerrero-Martinez, M. Martinez-Sober, and J. Espi-Lopez, “An easy demonstration of the optimum value of the adaptation constant in the lms algorithm,” *IEEE Transactions on Education*, vol. 41, no. 1, p. 81, 1998.
- [64] Y. L. Wu, Q. Song, and S. Liu, “A normalized adaptive training of recurrent neural networks with augmented error gradient,” *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 351–356, 2008.
- [65] K. S. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.
- [66] W. F. Chang and M. W. Mak, “A conjugate gradient learning algorithm for recurrent neural networks,” *Neurocomputing*, vol. 24, no. 1-3, pp. 173–189, 1999.
- [67] N. N. Schraudolph, “Fast curvature matrix-vector products for second-order gradient descent,” *Neural Computation*, vol. 14, no. 7, pp. 1723–1738, 2002.
- [68] P. Trebatický, “Recurrent neural network training with the extended kalman filter,” in *IIT. SRC 2005: Student Research Conference*, 2004.
- [69] W. Yu and J. de Jesús Rubio, “Recurrent neural networks training with stable bounding ellipsoid algorithm,” *IEEE Transactions on Neural Networks*, vol. 20, no. 6, pp. 983–991, 2009.
- [70] Y. Engel, S. Mannor, and R. Meir, “The kernel recursive least-squares algorithm,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.
- [71] B. Scholkopf, “The kernel trick for distances,” *Advances in Neural Information Processing Systems*, pp. 301–307, 2001.
- [72] W. Liu, P. Pokharel, and J. Principe, “The kernel least-mean-square algorithm,” *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 543–554, 2008.
- [73] W. Liu *et al.*, “Kernel affine projection algorithms,” *EURASIP Journal on Advances in Signal Processing*, vol. 2008, 2008.

- [74] B. Chen, S. Zhao, P. Zhu, and J. Príncipe, “Quantized kernel least mean square algorithm,” *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 1, pp. 22–32, 2012.
- [75] W. Liu, I. Park, Y. Wang, and J. Príncipe, “Extended kernel recursive least squares algorithm,” *IEEE Transactions on Signal Processing*, vol. 57, no. 10, pp. 3801–3814, 2009.
- [76] J. Platt, “A resource-allocating network for function interpolation,” *Neural computation*, vol. 3, no. 2, pp. 213–225, 1991.
- [77] W. Liu, I. Park, and J. Príncipe, “An information theoretic approach of designing sparse kernel adaptive filters,” *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1950–1961, 2009.
- [78] C. Richard, J. Bermudez, and P. Honeine, “Online prediction of time series data with kernels,” *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1058–1067, 2009.
- [79] J. Tropp, “Greed is good: Algorithmic results for sparse approximation,” *Information Theory, IEEE Transactions on*, vol. 50, no. 10, pp. 2231–2242, 2004.
- [80] I. Sutskever and G. Hinton, “Temporal-kernel recurrent neural networks,” *Neural Networks*, vol. 23, no. 2, pp. 239–243, 2010.
- [81] D. Liu and A. N. Michel, “Asymptotic stability of discrete-time systems with saturation nonlinearities with applications to digital filters,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 39, no. 10, pp. 798–807, 1992.
- [82] K. Matsuoka, “Stability conditions for nonlinear continuous neural networks with asymmetric connection weights,” *Neural Networks*, vol. 5, no. 3, pp. 495–500, 1992.
- [83] M. Forti, S. Manetti, and M. Marini, “Necessary and sufficient condition for absolute stability of neural networks,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 41, no. 7, pp. 491–494, 1994.
- [84] L. Jin, P. N. Nikiforuk, and M. M. Gupta, “Absolute stability conditions for discrete-time recurrent neural networks,” *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 954–964, 1994.
- [85] S. Arik and V. Tavsanoglu, “Absolute stability of nonsymmetric neural networks,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 3, 1996, pp. 441–444.

- [86] N. E. Barabanov and D. V. Prokhorov, "Stability analysis of discrete-time recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 292–303, 2002.
- [87] S. Hu and J. Wang, "Global asymptotic stability and global exponential stability of continuous-time recurrent neural networks," *IEEE Transactions on Automatic Control*, vol. 47, no. 5, pp. 802–807, may 2002.
- [88] J. Xu, Y.-Y. Cao, Y. Sun, and J. Tang, "Absolute exponential stability of recurrent neural networks with generalized activation function," *IEEE Transactions on Neural Networks*, vol. 19, no. 6, pp. 1075–1089, june 2008.
- [89] L. Wang and Z. Xu, "Sufficient and necessary conditions for global exponential stability of discrete-time recurrent neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 6, pp. 1373–1380, 2006.
- [90] L. Wang, R. Zhang, Z. Xu, and J. Peng, "Some characterizations of global exponential stability of a generic class of continuous-time recurrent neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 3, pp. 763–772, 2009.
- [91] J. Cao and J. Wang, "Global exponential stability and periodicity of recurrent neural networks with time delays," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 5, pp. 920–931, 2005.
- [92] Z. Zeng and J. Wang, "Improved conditions for global exponential stability of recurrent neural networks with time-varying delays," *Neural Networks, IEEE Transactions on*, vol. 17, no. 3, pp. 623–635, 2006.
- [93] Z. Zeng, J. Wang, and X. Liao, "Global asymptotic stability and global exponential stability of neural networks with unbounded time-varying delays," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 52, no. 3, pp. 168–173, 2005.
- [94] —, "Global exponential stability of a general class of recurrent neural networks with time-varying delays," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 50, no. 10, pp. 1353–1358, 2003.
- [95] C.-C. Ku and K. Lee, "Diagonal recurrent neural networks for dynamic systems control," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 144–156, jan 1995.
- [96] C.-M. Kuan, "A convergence result for learning in recurrent neural networks," *Mechatronics*, vol. 6, no. 3, pp. 420–440, 1994.
- [97] L. Jin and M. Gupta, "Stable dynamic backpropagation learning in recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1321–1334, 1999.

- [98] J. Suykens, B. de Moor, and J. Vandewalle, "Robust local stability of multilayer recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 1, pp. 222–229, jan 2000.
- [99] D. Xu, Z. Li, W. Wu, X. Ding, and D. Qu, "Convergence of gradient descent algorithm for diagonal recurrent neural networks," in *Proceeding of Second International Conference on Bio-Inspired Computing: Theories and Applications, 2007*. IEEE, 2007, pp. 29–31.
- [100] K. Patan, "Stability analysis and the stabilization of a class of discrete-time dynamic neural networks," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 660–673, 2007.
- [101] X. Wang and Y. Huang, "Convergence study in extended kalman filter-based training of recurrent neural networks," *IEEE Transactions on Neural Networks*, no. 99, pp. 1–13, 2011.
- [102] N. B. Karayiannis and G. W. Mi, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE Transactions on Neural Networks*, vol. 8, no. 6, pp. 1492–1506, 1997.
- [103] G. B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.
- [104] [Online]. Available: http://en.wikipedia.org/wiki/Metric_space
- [105] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*. Cambridge Univ Pr, 2004.
- [106] [Online]. Available: <http://www.ee.cityu.edu.hk/~gchen/pdf/C-Encyclopedia04.pdf>
- [107] Z. Xu, Q. Song, and D. Wang, "Recurrent neural tracking control based on multivariable robust adaptive gradient-descent training algorithm," *Neural Computing & Applications*, pp. 1–11, 2011.
- [108] P. A. Ioannou and J. Sun, *Robust adaptive control*. Prentice-Hall, Inc., 1995.
- [109] D. Mandic *et al.*, *Complex valued nonlinear adaptive filters: noncircularity, widely linear and neural models*. Wiley, 2009, vol. 59.
- [110] B. Che Ujang, C. Took, and D. Mandic, "Quaternion-valued nonlinear adaptive filtering," *Neural Networks, IEEE Transactions on*, vol. 22, no. 8, pp. 1193–1206, 2011.

- [111] B. Che Ujang, C. Cheong Took, and D. Mandic, "On quaternion analyticity: Enabling quaternion-valued nonlinear adaptive filtering," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2117–2120.
- [112] Q. Song, J. Xiao, and Y. C. Soh, "Robust backpropagation training algorithm for multilayered neural tracking controller," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1133–1141, 1999.
- [113] S. Jagannathan and F. L. Lewis, "Robust implicit self-tuning regulator: Convergence and stability," *Automatica*, vol. 32, no. 12, pp. 1629 – 1644, 1996.
- [114] —, "Identification of nonlinear dynamical systems using multilayered neural networks," *Automatica*, vol. 32, no. 12, pp. 1707–1712, 1996.
- [115] F. Lewis, C. Abdallah, and D. Dawson, *Control of robot manipulators*. Macmillan, New York, USA, 1993.
- [116] Y. Maeda and M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its fpga implementation," *Neural Networks, IEEE Transactions on*, vol. 16, no. 6, pp. 1664–1672, 2005.
- [117] A. Trunov and M. Polycarpou, "Automated fault diagnosis in nonlinear multi-variable systems using a learning methodology," *IEEE Transactions on Neural Networks*, vol. 11, no. 1, pp. 91–101, 2000.
- [118] J. Spall and J. Cristion, "Model-free control of nonlinear stochastic systems with discrete-time measurements," *IEEE Transactions on Automatic Control*, vol. 43, no. 9, pp. 1198–1210, 1998.
- [119] J. C. Spall and J. A. Cristion, "A neural network controller for systems with unmodeled dynamics with applications to wastewater treatment," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 27, no. 3, pp. 369–375, 1997.
- [120] Y. Maeda and R. De Figueiredo, "Learning rules for neuro-controller via simultaneous perturbation," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1119–1130, 1997.
- [121] J. Spall and J. Cristion, "A neural network controller for systems with unmodeled dynamics with applications to wastewater treatment," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 27, no. 3, pp. 369–375, 1997.
- [122] T. Lin, C. Giles, B. Horne, and S. Kung, "A delay damage model selection algorithm for narx neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2719–2730, 1997.

- [123] Y. Park, T. Murray, and C. Chen, "Predicting sun spots using a layered perceptron neural network," *IEEE Transactions on Neural Networks*, vol. 7, no. 2, pp. 501–505, 1996.
- [124] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.
- [125] G. Li, C. Wen, and Z. Li, "A new online learning with kernels method in novelty detection," in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 2311–2316.
- [126] G. Li, C. Wen, D. Cui, and F. Yang, "A new method of online learning with kernels for regression," in *7th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2012*, july 2012, pp. 1291 –1296.
- [127] J. Kivinen, A. Smola, and R. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.
- [128] J. Suykens and J. Vandewalle, "Recurrent least squares support vector machines," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 47, no. 7, pp. 1109–1114, 2000.
- [129] M. I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in psychology*, vol. 121, pp. 471–495, 1997.
- [130] Q. Song, "Robust initialization of a jordan network with recurrent constrained learning," *Neural Networks, IEEE Transactions on*, vol. 22, no. 12, pp. 2460–2473, 2011.
- [131] Y. Engel, "Algorithms and representations for reinforcement learning," *Doktorarbeit, The Hebrew University of Jerusalem*, 2005.
- [132] E. Kreyszig, *Introductory functional analysis with applications*. Wiley, 1989, vol. 21, no. 3.
- [133] S. Koike, "A class of adaptive step-size control algorithms for adaptive filters," *IEEE Transactions on Signal Processing*, vol. 50, no. 6, pp. 1315–1326, 2002.