5-2010

ARCHITECTURE OPTIMIZATION, TRAINING CONVERGENCE AND NETWORK ESTIMATION ROBUSTNESS OF A FULLY CONNECTED RECURRENT NEURAL NETWORK

Xiaoyu Wang
Clemson University, xiaoyuclemson@hotmail.com

ARCHITECTURE OPTIMIZATION, TRAINING CONVERGENCE AND NETWORK ESTIMATION ROBUSTNESS OF A FULLY CONNECTED RECURRENT NEURAL NETWORK

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Mechanical Engineering

---

by
Xiaoyu Wang
May 2010

Accepted by:
Dr. Yong Huang, Committee Chair
Dr. John Gowdy
Dr. Nader Jalili
Dr. Ardalan Vahidi

ABSTRACT

Recurrent neural networks (RNN) have been rapidly developed in recent years. Applications of RNN can be found in system identification, optimization, image processing, pattern reorganization, classification, clustering, memory association, etc.

In this study, an optimized RNN is proposed to model nonlinear dynamical systems. A fully connected RNN is developed first which is modified from a fully forward connected neural network (FFCNN) by accommodating recurrent connections among its hidden neurons. In addition, a destructive structure optimization algorithm is applied and the extended Kalman filter (EKF) is adopted as a network's training algorithm. These two algorithms can seamlessly work together to generate the optimized RNN. The enhancement of the modeling performance of the optimized network comes from three parts: 1) its prototype - the FFCNN has advantages over multilayer perceptron network (MLP), the most widely used network, in terms of modeling accuracy and generalization ability; 2) the recurrency in RNN network make it more capable of modeling non-linear dynamical systems; and 3) the structure optimization algorithm further improves RNN's modeling performance in generalization ability and robustness.

Performance studies of the proposed network are highlighted in training convergence and robustness. For the training convergence study, the Lyapunov method is used to adapt some training parameters to guarantee the training convergence, while the maximum likelihood method is used to estimate some other parameters to accelerate the training process. In addition, robustness analysis is conducted to develop a robustness measure considering uncertainties propagation through RNN via unscented transform.

Two case studies, the modeling of a benchmark non-linear dynamical system and a tool wear progression in hard turning, are carried out to testify the development in this dissertation.

The work detailed in this dissertation focuses on the creation of: (1) a new method to prove/guarantee the training convergence of RNN, and (2) a new method to quantify the robustness of RNN using uncertainty propagation analysis. With the proposed study, RNN and related algorithms are developed to model nonlinear dynamical system which can benefit modeling applications such as the condition monitoring studies in terms of robustness and accuracy in the future.

DEDICATION

This Dissertation is dedicated to my parents who have always been supportive in my life.

ACKNOWLEDGMENTS

I would like to thank Dr. Yong Huang, Dr. John Gowdy, Dr. Nader Jalili, and Dr. Ardalan Vahidi, for their support and guidance in my study.

I would like to thank Dr. Nhan Nguyen and Dr. Kalmanje Krishnakumar, NASA scientists from the Ames Research Center, for their help in my research.

I would like to thank my colleagues who helped me during my stay at Clemson: Yu Long, Lei Tang, Wei Wang, Yafu Lin, Yin Jun, Leigh Herran, Mason D. Morehead, and Kevin Foy.

I would like to acknowledge the financial support from the South Carolina Space Grant Consortium and the NASA Ames Research Center.

TABLE OF CONTENTS

Table of Contents (Continued)

Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

List of Figures (Continued)

Figure Page

CHAPTER ONE

INTRODUCTION

Background

Typical engineering systems have high-order, nonlinear, and dynamical features. These systems often include sensors and actuators which interact with the system itself and the environment. Many of these systems are defined by characteristic parameters indicating the complex relationship among their various physical characteristics, often exhibiting time dependency due to their inherent dynamical nature. These condition or characteristic parameters are often difficult, if not impossible, to measure directly. As a result, modeling, a process that can describe the behavior of such system parameters, is especially important in condition monitoring. This supervision can detect changes or drifts in process parameters which may indicate the inception and growth of fault modes in a system [Iser84] [Hofl96].

Several methods have been developed for modeling non-linear dynamical systems. Generally, they can be categorized into the following two classes:

1) Physical-driven methods are developed by looking into the underlying theory of systems and developing mathematical models to describe the relationship among variables interested. These models are often in the form of differential equations [Huan02]. Some of these equations can be analytically solved and result in explicit models, while the complicated ones are often solved by

numerical methods as the finite element method (FEM), which divides a system into numerous elements, numerically solving the equations [Xie05].

2) Data-driven methods are developed based on empirical observations by using the information obtained through experiments and developing equations to describe relationships of the system modeled. Regression models or parametric models (linear regression with nonlinear terms, polynomial regression, and nonlinear regression), select the form of model first, and then determine its parameters through regression [Ozel05]. Artificial intelligence (AI)-based methods using AI techniques such as neural networks (NN) [Liu99] [Kuo99] [Sche03] and fuzzy logic [Kuo98] can be also applied to model systems, while for this case the developed models often can't be written explicitly.

Theses modeling methods have advantages and disadvantages. Although analytical models provide better insight into a system's underlying physical mechanisms through physical-driven methods, they are sometimes less satisfactory due to over-simplifications and unrealistic assumptions in their development. On the other hand, the models solved by FEM can provide accurate results; however, it is time-consuming and not suitable for optimization using current computing technology. Time series and regression models are typically less accurate than the AI –based models. If both accuracy and speed are of interest instead of a system's underlying physical mechanisms, AI-based modeling approaches are favored for real-time applications.

Among the AI-based approaches, NN is extensively applied in system modeling applications because of the following advantages:

1) They can carry out arbitrary function approximation especially for non-linear systems
2) They do not require reprogramming and can be applied to different systems.
3) They are error-tolerant due to their parallel computation features.

These advantages make NN a viable, reliable, and attractive approach for modeling engineering systems [Chry90] [Das96]. Figure (1.1) shows various NN developed for different applications. This classification is based on the most frequent application of the network. For example, while multi-layer perceptron NN (MLP) also can be applied as classifications and clustering, here it is classified as estimation and modeling applications.

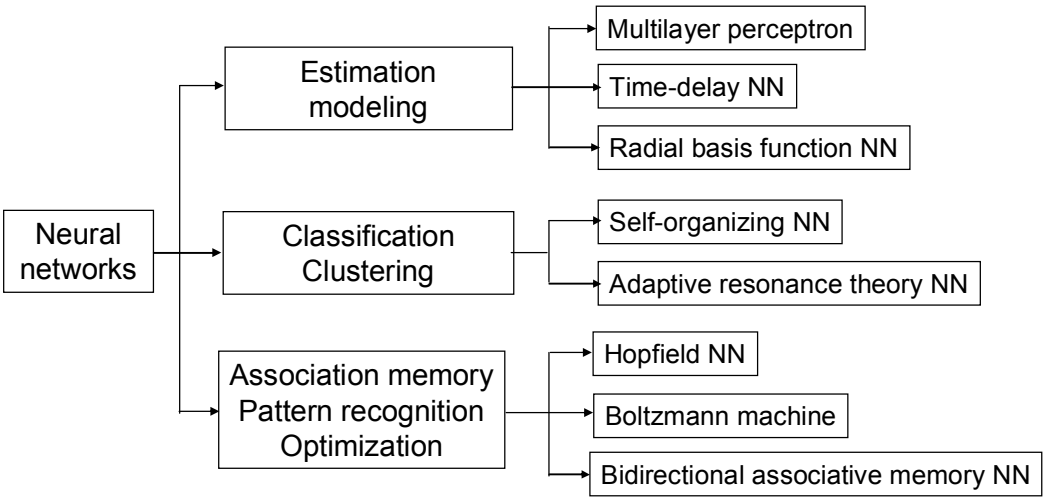


Figure 1.1: NN Developed for different applications

The research presented here focuses on applications in modeling and estimation. Its objective is to develop an NN with advantages over the currently applied ones and then study its performance. Among NN applied in system modeling, MLP is the one most frequently used. Previous research has found that a fully forward connected NN (FFCNN) exhibits better performance in terms of generalization ability, training accuracy, and structural robustness than an MLP [Wang08a]. In addition, an FFCNN can be modified to become a fully connected recurrent neural network (RNN), accommodating recurrent connections among neurons. Unlike an FFCNN, a RNN can store information from past states, making it more capable of modeling nonlinear dynamical phenomena. However, using a RNN involves such issues as divergence [Mand01], instability [Meds99], and a lack of robustness [Mand01]. The relationship among these three networks, including their advantages, can be seen in Figure (1.2).

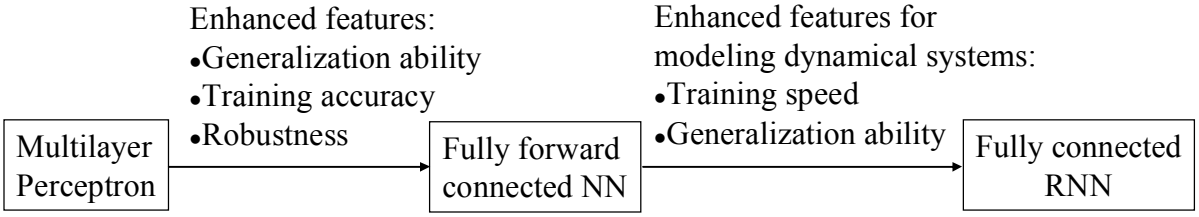


Figure 1.2: Networks with improved modeling performance

To apply RNN, several issues have to be carefully addressed.

1) Network architecture optimization

The determination of optimal network architecture is critical for the successful application of NN models because it can save calculation cost while at the same time maintaining modeling accuracy, generalization ability, and robustness [Alip02].

Optimization of network structure requires consideration of such features as the number of inputs, the number of outputs, the degree of complexity of the system, and the available training data for each application. Overly complicated networks tend to have an over-fitting problem, while architecture that is too simple results in poor training accuracy [Mood92].

2) Training convergence

The different training algorithms for RNN have advantages and disadvantages. To select an appropriate training algorithm in terms of training speed and accuracy is important. Furthermore, determining the training parameters is another major concern having a significant influence on network performance; specifically, training divergence can occur if these parameters are not selected properly [Luo97].

3) Robustness

Robustness studies on NN have primarily considered uncertainties in inputs and weights [Chiu93] [Alip01] [Alip04]. Once the structure of a network has been decided and the training process completed, a network is realized. The different architectures and configurations of NN training are realized in different network models. Robustness analysis of the realized networks is essential to eliminate those networks exhibiting poor robustness so that the best candidate is selected.

As this discussion indicates, it is important to develop a complete and reliable modeling technique for general non-linear dynamical systems. Network architecture optimization and training algorithm realization are the foundation, and performance studies including training convergence and network robustness can further enhance the

applications of RNN in non-linear dynamical system modeling. To investigate this area, the objectives of this study are to develop a fully connected RNN, to explore its capability for modeling dynamical systems and to evaluate its performance concerning training convergence and robustness.

Overview of This Study

Previous research on nonlinear system modeling applications has focused on applying RNN to model non-linear dynamical systems, but little has been conducted on the theoretical analysis. In this study, RNN with internal feedback connections are developed for modeling nonlinear dynamical systems with the following tasks:

1) A RNN is formed by accompanying recurrent connections in the hidden neuron section of an FFCNN. An extended Kalman filter algorithm (EKF) is applied to train the network.
2) Network architecture optimization is achieved using a destructive connectivity algorithm.
3) Performance analysis including a convergence study of the training process of RNN and a robustness analysis of the trained network are conducted, theoretically making the network substantially complete in theoretical proof and hence ensuring the quality of its performance.

This study was divided into the three parts shown in Figure (1.3). The first was RNN development including the development of the network architecture optimization and the training algorithm. A destructive optimization algorithm was applied to determine

network architecture and the extended Kalman filter algorithm to train the network. Performance studies were then applied to this resulting network. A convergence study was conducted to improve the network's training convergence performance and a robustness analysis conducted to assess its robustness to perturbations in the trained weights. Finally, a non-linear dynamical benchmark system and a tool wear propagation process were used to verify the algorithms applied.

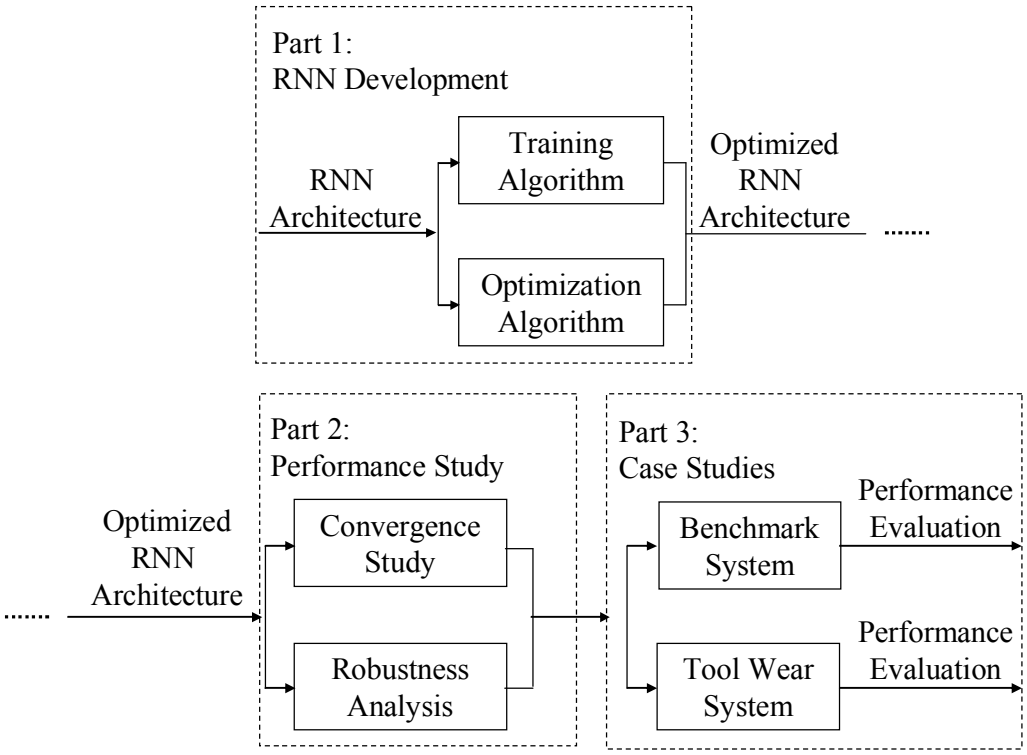


Figure 1.3: Layout of the study

Based on this study, the following conclusions were drawn:

1) The modeling capability of the proposed RNN is better than that of the commonly used MLP network.

2) Architecture optimization improves the modeling capability of the proposed RNN
3) The adaption law of training parameters improves the training convergence of the network.
4) The proposed robustness quantification method is effective and efficient.

The contributions of this study to the literature are summarized below:

1) The EKF training and destructive optimization algorithms can be applied to the proposed RNN
2) The Lyapunov method and the maximum likelihood method can be applied to tune the statistical matrices Q and R of the EKF to ensure the convergence of RNN training algorithm and at the same time to improve the convergence speed
3) The unscented transform method can be applied to quantify the robustness of RNN to uncertainties in the trained weights

Organization of This Study

The organization of the study is shown in Figure (1.4).

Chapter two provides the theoretical background of this study. Such topics as architectures, training algorithms, and topology optimization techniques of neural networks are introduced. The structures of networks, listed in Figure (1.2), applied in modeling applications are illustrated first. Three classes of network structure optimization methods (empirical methods, destructive or constructive methods, and other optimization

methods) are then reviewed. Training algorithms for RNN (back-propagation through time, real-time recurrent learning, and EKF training algorithm) are introduced. These algorithms are used to determine the structure and parameters of a RNN model. In addition to the development of networks, convergence and robustness studies of recurrent neural networks are reviewed as well. Convergence studies have been conducted on networks' states, outputs and training process and the last one is concerned in this study. According to different applications of RNN, robustness studies have different concerns. This study focuses on modeling applications and hence the estimation robustness is reviewed in details. Along with the background introduction, the motivations and concerns of this study are also discussed.

Chapter three proposes the development of an optimized RNN (Part 1 in Section 1.2) which has advantages in modeling non-linear dynamical systems over the commonly used MLP. First, the structure of a RNN is illustrated; the RNN is modified from an FFCNN by accommodating internal recurrency in its hidden neuron section. The EKF algorithm which used to train the network (determine the weights of the RNN) is then detailed in the following section. Finally a destructive optimization method is introduced, which optimizes the RNN network structure to form the optimized RNN (OptRNN).

Chapter four studies the training convergence and robustness of the proposed RNN (Part 2 in Section 1.2). The EKF training algorithm has divergence problem if its parameters are not selected properly. To solve the problem, Lyapunov method is applied to develop an adaption law on a training parameter, the covariance of measurement noise. Furthermore, the convergence speed is accelerated by an adaption law on another training

parameter, the covariance of process noise, using the maximum likelihood method. In addition to the training convergence, another important issue for the successful implementation of RNN, the robustness is studied by conducting an uncertainty propagation analysis using the unscented transform.

Chapter five and Chapter six verifies the studies in Chapter three and Chapter four using two case studies, a non-linear dynamical benchmark system and a tool wear progression process (Part 3 in Section 1.3).

The final chapter of this dissertation presents the conclusions of this study.

The appendix lists the flow charts of matlab programs for RNN training, RNN optimization, and RNN training with R and Q adaption law applied.

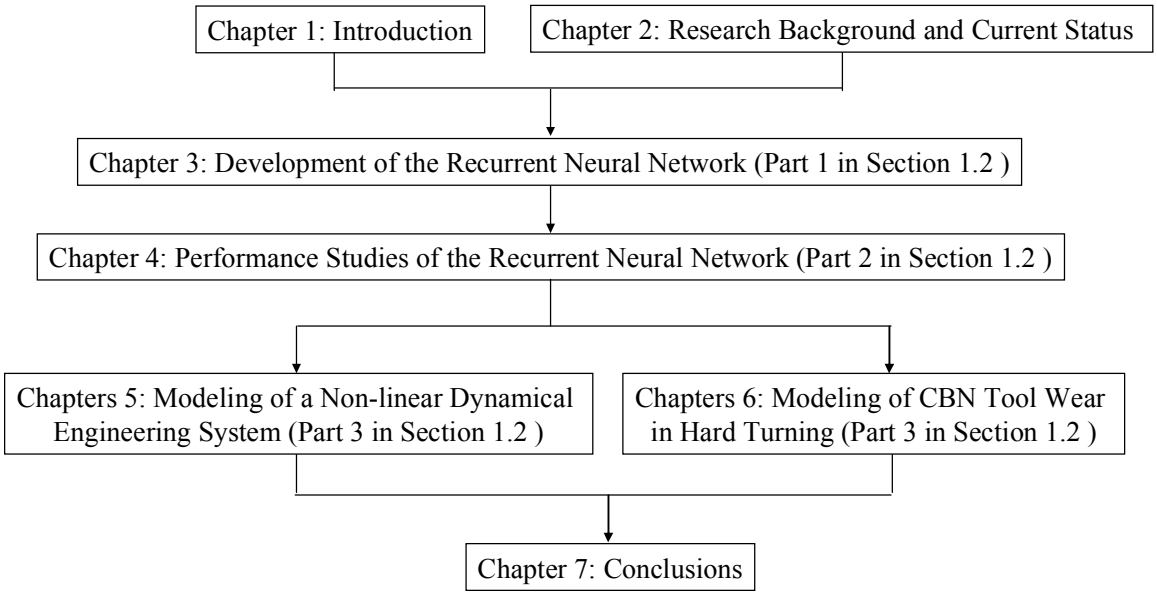


Figure 1.4: Organization of the study

CHAPTER TWO

RESEARCH BACKGROUND AND CURRENT STATUS

Abstract

This chapter introduces the theoretical background of this study. Several topics are covered on neural network architecture, topology optimization, training algorithms, convergence study, and robustness study of RNN. The structure of the proposed RNN is modified from an FFCNN which has advantages over the widely used MLP in training accuracy and generalization ability. Network topology optimization techniques are applied to optimize the structure of a network. Among these approaches, pruning approach can generate simple robust and efficient optimized structure. Training algorithm is applied to tune the weights of the developed structure. For RNN training, there are three major algorithms developed, namely, back-propagation through time (BPTT) algorithm, real-time recurrent learning (RTRL) algorithm and EKF training algorithm. Among them, EKF is proved to be fast and accurate. Convergence studies of RNN include three brunches - state convergence, output convergence and training convergence. Training convergence concerns the stability of weight update during training process and it is the focus of this study. Finally, robustness of NN for modeling applications, called estimation robustness, is reviewed in this chapter. Basic concepts and the up to date developments in these areas are introduced for each topic. Based on the background review, the techniques applied in this study are also briefly introduced.

Nomenclature

Multilayer Perceptron Network	
Symbol	Definition
$b_{j,i}$	Bias of the *ith* neuron in layer *j*
$f_{j,i}(\cdot)$	Activation function of the *ith* neuron in layer *j*
$net_{j,i}$	Net input of the *ith* neuron in layer *j*
$w_{j,i,k}$	Weight of the connection from neuron *k* in layer *j-1* to neuron *i* in layer *j*
$y^{o}_{j,i}$	Output of the *ith* neuron in layer *j*

Recurrent Multilayer Perceptron Network	
Symbol	Definition
$f_{j,i}(\cdot)$	Activation function of the *ith* neuron in layer *j*
$net_{j,i}(n)$	Net input of the *ith* neuron in layer *j* at time step n
n_j	Number of neurons in layer *j*
$w^{f}_{j,i,k}$	Feedforward weight from neuron *k* in layer (*j-1*) to the *ith* neuron in layer *j*
$w^{r}_{j,i,k}$	Feedback weight from neuron *k* in layer (*j+1*) to the *ith* neuron in layer *j*
$y^{o}_{j,i}(n)$	Output of the *ith* neuron in layer *j* at time step n

EKF Training Algorithm	
Symbol	Definition
$H(k)$	Jacobian matrix at training step k
$K(k)$	Kalman gain at training step k
$P(k)$	Covariance matrix of weight estimation at training step k
$Q(k)$	Covariance matrix of process noise at training step k
$R(k)$	Covariance matrix of measurement noise at training step k
$\vec{w}(k)$	Estimation of weight vector $\vec{w}^*$ at training step k
$\vec{y}^*(k)$	Desired output at training step k
$\vec{y}(k)$	Output of neural network at training step k

Neural Networks Architecture

Artificial neural network, often abbreviated as neural network, was invented by Warren S. McCulloch and Walter Pitts in 1943 [Mccu43], which simulates the operations of biological neural network. It is composed of a number of highly interconnected processing elements (neurons) working in parallel to solve specific problems.

As shown in Figure (2.1), biological neurons are the core component of a human brain, which are responsive cells that transmit and process signals. A neuron cell is generally comprised of the cell body, axon, and dendrites. It receives signals from other neurons through dendrites. In addition, it also sends out spikes of electrical activity through an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon.

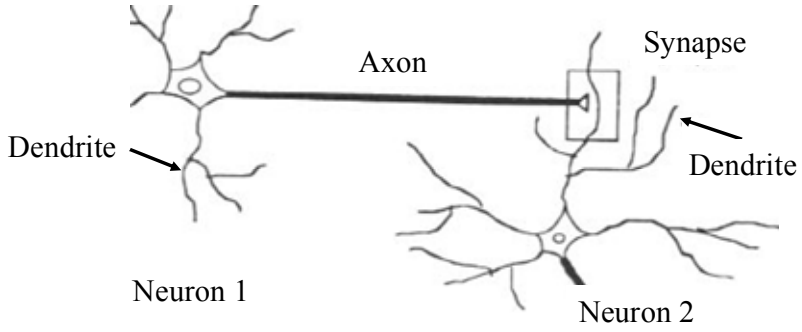


Figure 2.1: Two connected neuron cells

Neuron network is built to imitate a human's neuron system. Figure (2.2) shows a typical artificial neuron in an NN. Such a neuron unit is also called a perceptron [Rose58] which can be viewed as a simplest NN – a single layer neural network with one neuron. Typically an artificial neuron is composed of weights, a summation operator and an activation function. The perceptron can be used to form a mapping function from its inputs ($x_1, x_2, ...x_m$) to its output (y). A weight simulates the function of a dendrite in Figure (2.1). The summation of weighted inputs is called net input which feeds into the activation function to form output of the neuron. The mapping function is described as follows:

$$net = \sum_{k=1}^{m} w_k x_k \tag{2.1}$$

$$y = f(net) \tag{2.2}$$

where net is the net input, m is the number of inputs, w_k is the weight for input k, y is the output, and $f(\cdot)$ is the activation function

The activation function represents the function of a cell body in Figure (2.1). Here it is a step function – if the net input is less than 0, the output is -1, and otherwise it will be 1. A lot of functions can be selected as activation function as shown in Figure (2.3). Among them the most popular one is the sigmoid function. For an NN with connected neurons, the output of a neuron can propagates through its axon to other neurons.

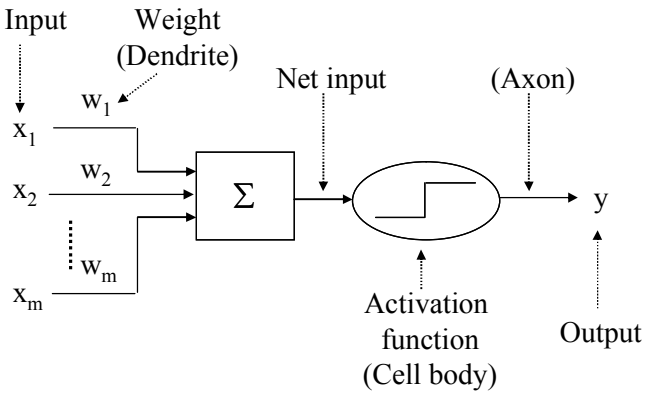


Figure 2.2: A neuron unit in an NN

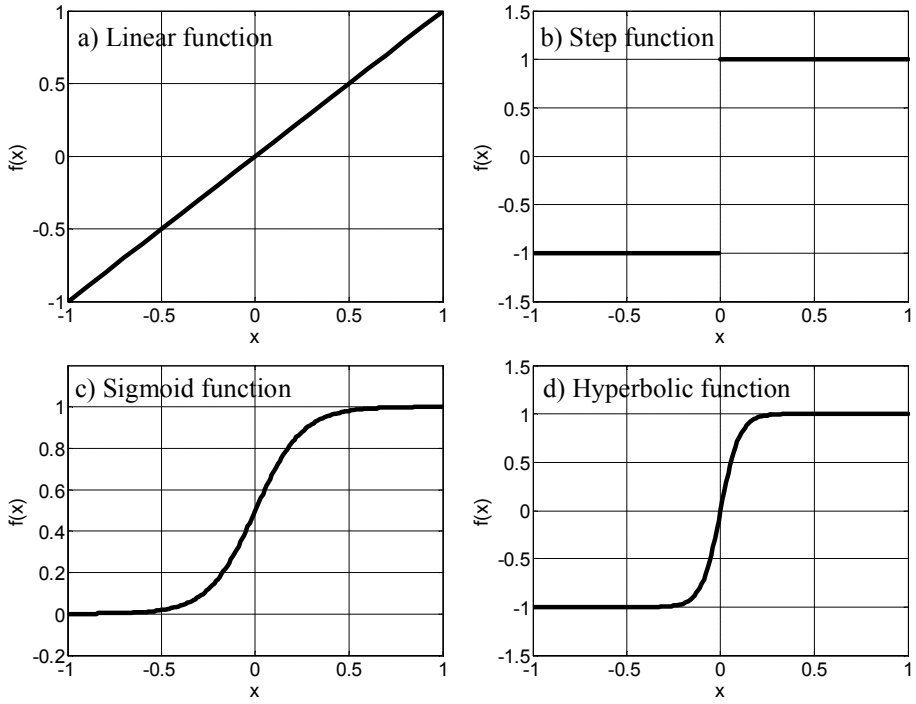


Figure 2.3: Different types of activation functions

A neuron network is often composed of a lot of connected neurons. There are many types of neural networks with different structures and operation mechanism, each of which has different strengths particular to some certain applications. Some well-known

NN are MLP NN - an NN model with more than three layers of neurons often using sigmoid activation function that maps sets of input data onto a set of output, radial basis function (RBF) NN - an NN similar to MLP but uses radial basis functions as its activation function, self-organizing map (SOM) NN – an NN that often used to produce a low-dimensional (typically two dimensional) representation of the input high-dimensional data, Hopfield NN – an NN applied in content-addressable memory application that the network can converge to a "remembered" state if it is given a distorted input, Boltzmann machine – a type of stochastic NN which is used to model the a system's statistical behavior, adaptive resonance theory (ART) NN – an NN used to achieve a self-organized stable pattern recognition capability in real time by using the adaptive resonance theory, and neural fuzzy NN – an NN combining combination the fuzzy inference system in its body to incorporate fuzzy IF-THEN rules to the network.

According to the direction of calculation flows within a network, neural networks can be divided into two classes, feedforward neural networks (FFNN) and recurrent neural networks (RNN). While a feedforward network, such as the MLP, only propagates data forwardly from input to output, a recurrent neural network also has feedback connections and it can propagate data from later processing elements to earlier elements which make it more suitable for modeling dynamical systems [Link96]. Some commonly used RNN are Elman network, Jordan network, and recurrent multilayer perceptron network. They are generally modified from the MLP NN.

MLP is the most popular and widely used neural network [Sama06]. As an example, Figure (2.4) shows an MLP neural network. Each circle represents a neuron

(perceptron) illustrated in Figure (2.2). The network is composed of three layers of neurons. The first layer is called input layer which take in inputs; the last layer is called output layer which generate output of network; the layers in between have no connection with the external world are called hidden layer. Information flows forwardly from layers in left to layers in right. Each neuron in one layer is connected to every neural on the next layer and there is no connection among neurons in the same layer. The network has 9 neurons to form a mapping function from its inputs (x_1, x_2, x_3, x_4) to its outputs (y_1, y_2). In Figure (2.4), two types of outputs need to be distinguished; output of a neuron is denoted as $y^o_{i,j}$ where the first subscript *i* denotes the layer number of the neuron and the second subscript *j* denotes the index of the neuron in layer *i*; on the other hand, output of the network ($\vec{y} = [y_1, y_2]$)is composed of the outputs of neurons in output layer. It is easy to see that $y^o_{3,1}$ and y_1 are two notations for the same output. It is proved that an MLP with at least one hidden layer can approximate any continuous function at any desired degree of accuracy with sufficiently many hidden neurons are available and hence MLP can be seen as a universal approximator [Horn89].

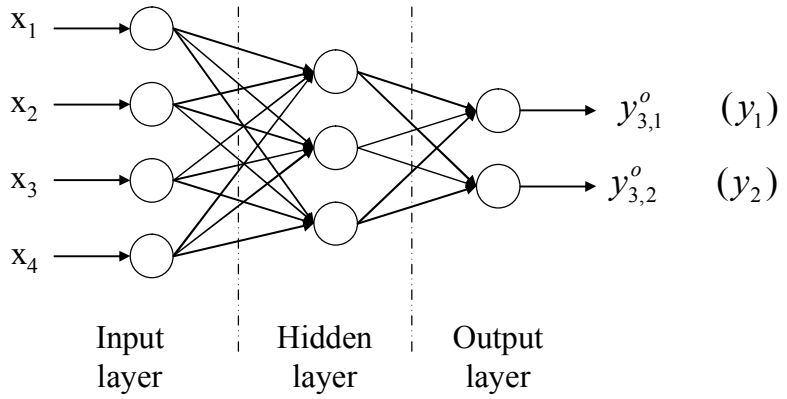


Figure 2.4: An MLP neural network

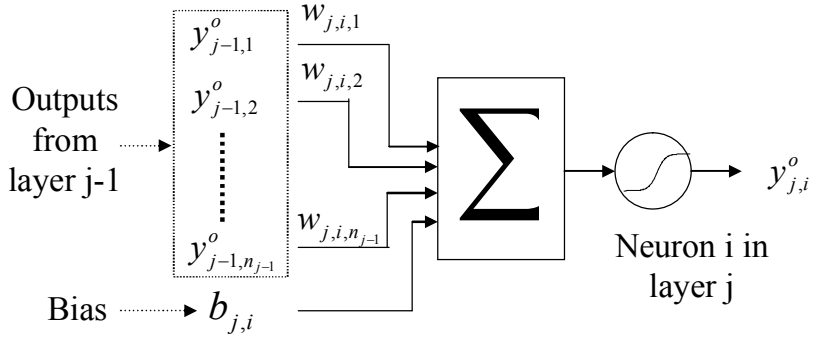


Figure 2.5: Schematic of Equations (2.3 and 2.4)

As shown in Figure (2.5), for the *ith* neuron in layer *j*, its output $y^o_{j,i}$ can be written by

$$y^o_{j,i} = f_{j,i}(net_{j,i}) \tag{2.3}$$

$$net_{j,i} = \sum_{k=1}^{n_{j-1}} w_{j,i,k} y^o_{j-1,k} + b_{j,i} \tag{2.4}$$

The output of the network can be written as:

$$y_i = y^o_{n,i} \tag{2.5}$$

where $net_{j,i}$ is the net input of the neuron, the weighted summation of outputs of neurons from the previous layer; $w_{j,i,k}$ is the weight for the connection from neuron *k* in layer *j-1* to neuron *i* in layer *j*; $b_{j,i}$ is the bias for the neuron; n is the layer number of output layer, and $f_{j,i}(\cdot)$ is the activation function of the neuron, which is often taken as a sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

Other than the MLP, FFCNN is also an FFNN. The FFCNN was proposed by Werbos [Werb90]. It can be viewed as a general version of MLP and is adopted as the prototype of the proposed RNN in this study.

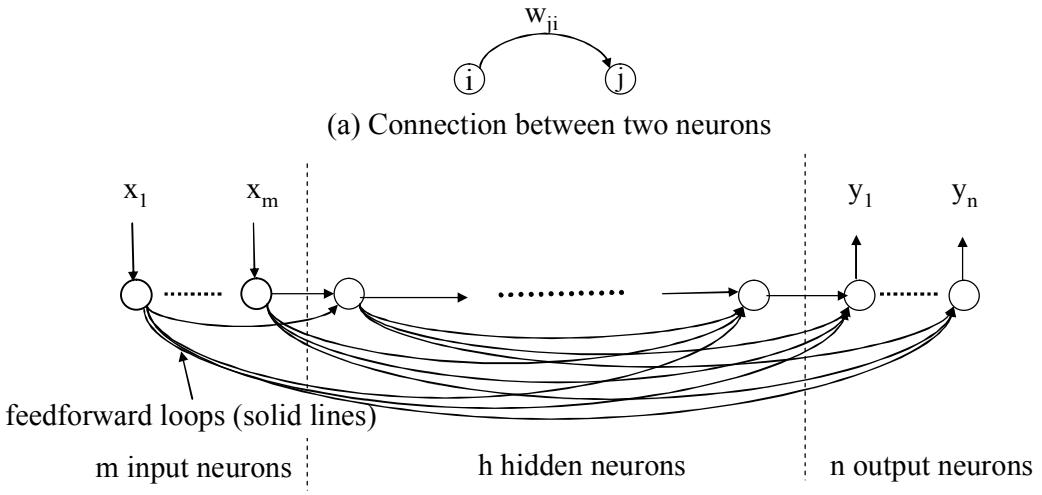


Figure 2.6: Architecture of a fully forward connected neural network

Recurrent Neural Networks

A representative FFCNN is shown in Figure (2.6). The network is composed of three sections, namely input neurons section, hidden neurons section, and output neurons section respectively. The network has m neurons in input section which receive the inputs of the network. h hidden neurons are used to relay the outputs of input neurons to output neurons. n output neurons generate outputs of the network. The network is used to model a system with m inputs n outputs. It is a forward network since there are no feedback connections and data only propagate from left to right. It is also a fully connected network because each neuron takes connections from every other neuron to the left of itself. [Werb90] [Kris93]. Figure (2.6(a)) shows the connection between two neurons. The weight w_{ji} represents the weight on the connection from neuron i to neuron j. This network architecture is used as the foundation for the proposed RNN. The equations (2.6-2.8) describe the mapping functions of the network.

As shown in Figure (2.7), for each neuron, its net input net_i is formed by summing the weighed outputs prior to it.

$$net_i = \sum_{j=1}^{i-1} w_{ij} y_j^o, \qquad 1 \le i \le m + h + n \tag{2.7}$$

where m, h, and n represent the number of the input neurons, hidden neurons, and output neurons respectively, net_i represents the net input to the neuron i, w_{ij} represents the weight on the connection from neuron j to the neuron i, and y_i^o represents the output of the neuron i

Each neuron i has an activation function $f_i(\cdot)$ which generates an output for its net input:

$$y_i^o = f_i(net_i), \qquad 1 \le i \le m+h+n \tag{2.8}$$

Neurons can have different activation functions in different sections. For neurons in the hidden section, a unipolar sigmoid activation function (shown in Figure (2.3)) is used as follows:

$$f_i(net_i) = \frac{1}{1+e^{-net_i}}, \qquad m < i \le m+h \tag{2.9}$$

For neurons not in the hidden section, the identity function is used as follows:

$$f_i(net_i) = net_i, \qquad 1 \le i \le m \text{ and } m+h < i \le m+h+n \tag{2.10}$$

The output of the network can be written as:

$$y_i = y_{i+m+h}^o \qquad i \le n \tag{2.11}$$

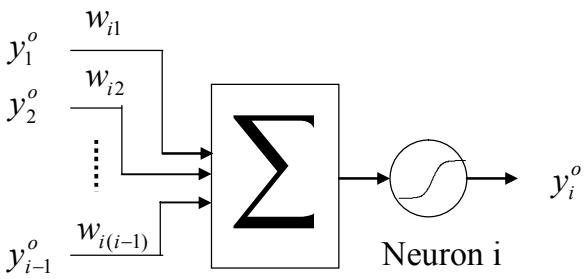
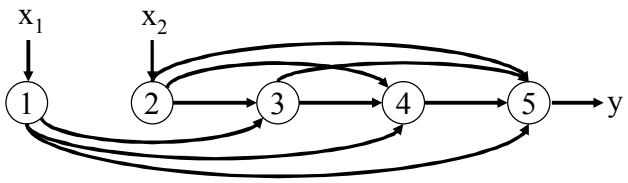


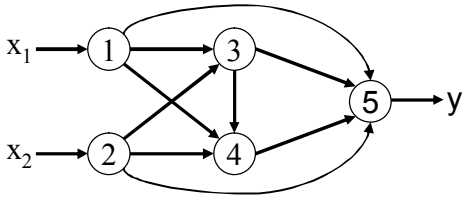
Figure 2.7: Illustration of Equations (2.7-2.9)

An FFCNN with a 2-2-1 structure is shown in Figure (2.8(a)) and it can be transformed into an MLP form in Figure (2.8(b)). In addition, a regular MLP (2-2-1) is shown in Figure (2.8 (c)) for comparison. It can be seen that, with the same structure, there are more connections and weights in the FFCNN (9) than in the MLP (6), and hence

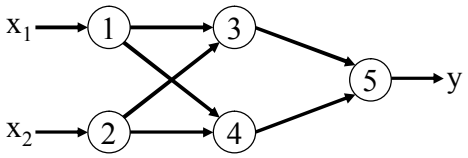
FFCNN has more parameters to tune in which sense it is said to be more general than an MLP.



a) An FFCNN



b) An equivalent MLP



c) A regular MLP

Figure 2.8: Comparison of FFCNN and MLP

Apart from the above FFNN, RNN are also applied in modeling applications and some of them are introduced in the following.

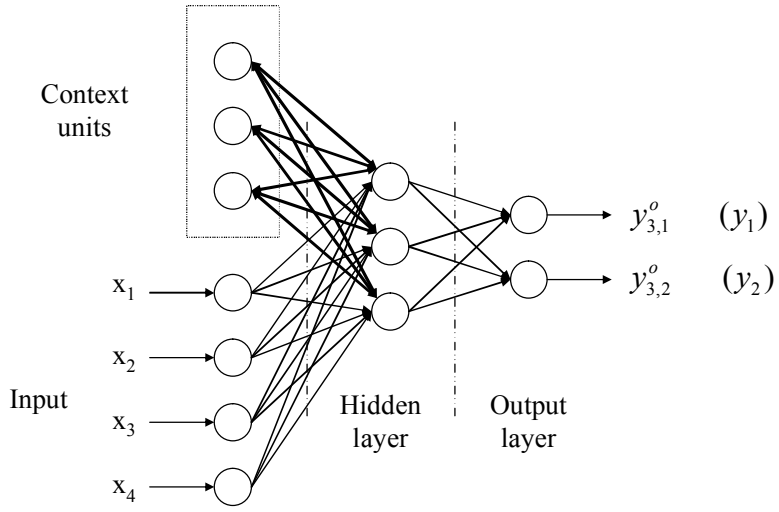


Figure 2.9: An Elman network

An Elman neural network [Elma90] is shown in Figure (2.9). The Elman Network has an extra set of input units, so-called context units. The context units contain a copy of the network's internal state (the outputs of neurons in the hidden layer) at the previous time step. The context units feed into the hidden layer just like the other input units do, so the network is able to compute a function that not only depends on the current input, but also on the network's internal state, which is determined by previous inputs. The network can be seen as an MLP with feedback connections from hidden layer to input layer.

Similar to an Elman network, as shown in Figure (2.10), a Jordan network [Jord86] can be viewed as an MLP with feedback connections from output layer to input layer. Its context units, working as an extra set of inputs, is a copy of the network's output at the previous time step.

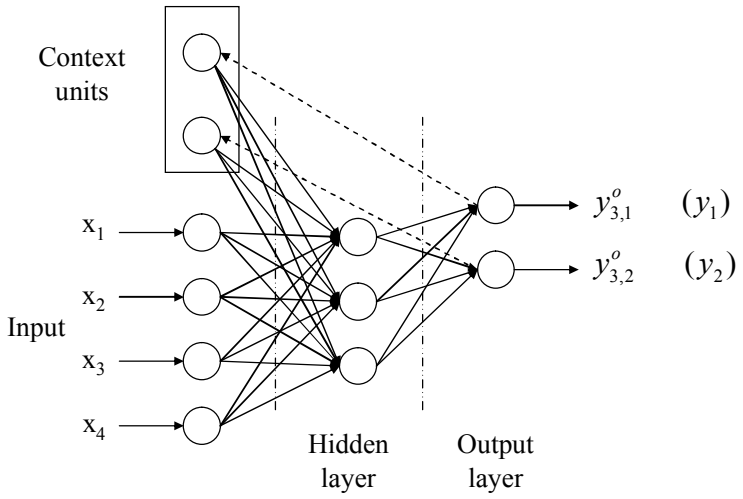


Figure 2.10: A Jordan network

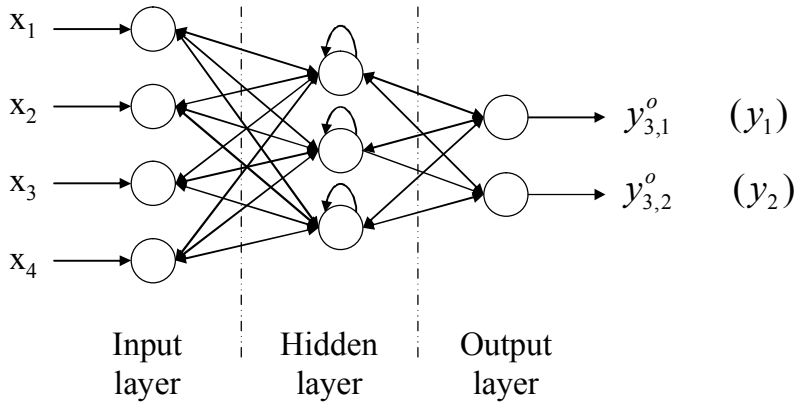


Figure 2.11: A RMLP network

As shown in Figure (2.11), a recurrent multilayer perceptron (RMLP) is modified from MLP by adopting feedback connections among the nodes of neighboring layers and feedback connections from neurons in hidden layer to themselves [Psal88]. The recurrent connections are delayed by one time step. In another point of view, the RMLP can be seen as a generalized version of an Elman network or a Jordan network.

For the *ith* neuron in layer *j*, its output can be described by

$$y^o_{j,i}(n) = f_{j,i}(net_{j,i}(n)) \tag{2.12}$$

$$net_{j,i}(n) = \sum_{k=1}^{n_{j-1}} w^f_{j,i,k}\, y^o_{-1,k}(n) + w^r_{j,i,i}\, y^o_{j,i}(n-1) + \sum_{k=1}^{n_{j+1}} w^r_{j+1,i,k}\, y^o_{j+1,k}(n-1) \tag{2.13}$$

where $f_{i,j}(\cdot)$ is the activation function, $net_{i,j}(n)$ is the net input at time step n, $w^f_{j,i,k}$ is the forward weight from neuron *k* in layer (*j-1*) to node *i* in layer *j*, $w^r_{j,i,k}$ is the feedback weight from neuron *k* in layer (*j+1*) to node *i* in layer *j*, and n_j is the number of neurons in layer *j*.

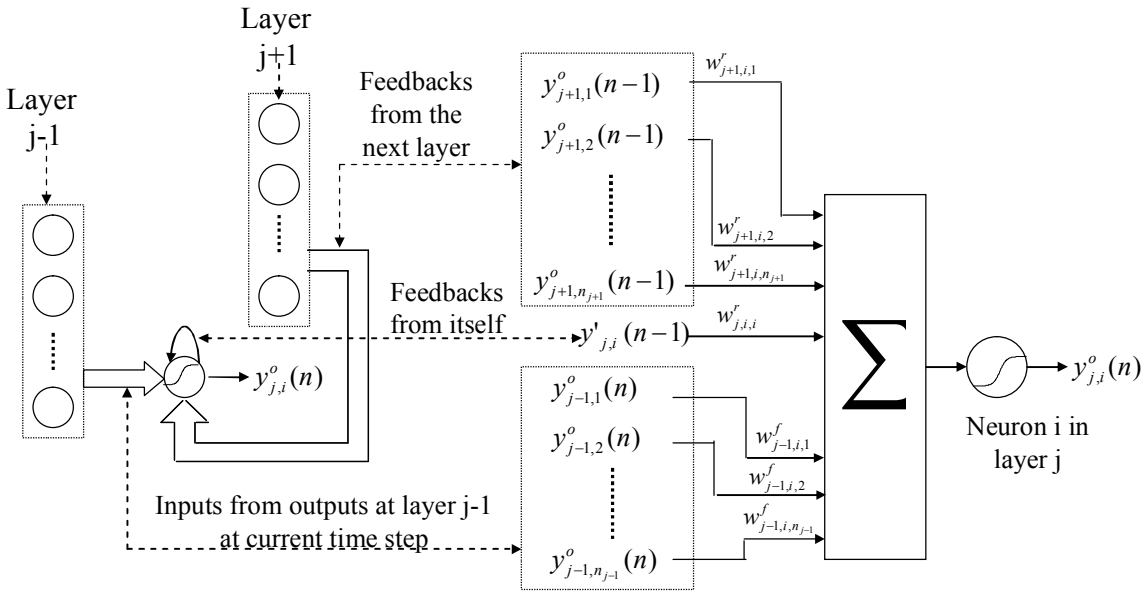


Figure 2.12: Schematic of equations (2.12 and 2.13)

In this study, a recurrent network modified from an FFCNN network is used to model non-linear dynamical systems. The specific structure of network is chosen because FFCNN has some advantages than MLP in terms of modeling accuracy and generalization ability. Hence the RNN modified is believed to have better performance in modeling.

Neural Network Optimization

Both MLP and RMLP are capable of modeling nonlinear dynamic systems [Lo94]. However, there are some problems with these fully connected neural networks:

1) They have a large parameters (weights and biases) space, which makes computation cost expensive; and

2) They are vulnerable to over fitting problem that networks tend to fit training data perfectly but poorly fit testing data.

Network architecture optimization can alleviate these problems. Optimal determination of network topology is indispensable to build an optimal NN modeling tool. Usually network topology is determined considering the following items:

1) How many hidden layers in the network;
2) How many neurons in each hidden layer; and
3) How neurons connect.

The function of hidden neurons is to model mapping function between network inputs and outputs. If insufficient number of hidden nodes is picked, it is not possible to form an accurate model for the training data (the data used to determine the weights of a network through a training process). On the other hand, if too many hidden nodes are used, the network may lose its ability to generalize. In addition, keeping the number of hidden layer nodes to a minimum can reduces the number of trainable weights, and hence can reduce the computational cost of training.

As shown in Figure (2.13), current network topology optimization techniques can be divided into three classes: empirical or trial and error method, destructive or constructive methods, and the applications of other optimization strategies to ANN [RAGG96].

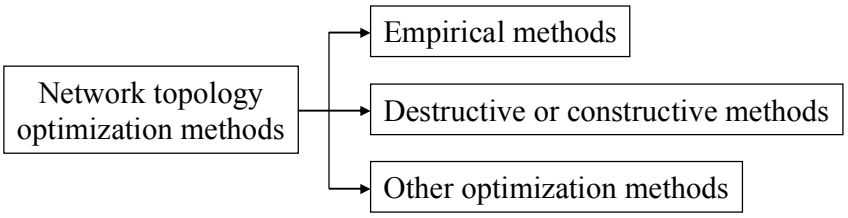


Figure 2.13: Classification of network topology optimization methods

The first class is trial and error which applied in most applications. Most researchers don't use a systematic approach, but test several sets of network topologies and compare the results. The best network structure (number of hidden layers, number of hidden neurons) is identified after comparison. The method is case-oriented and time consuming.

The second class is destructive or constructive methods. For the destructive methods, a network starts with an over-large structure and some of its neurons are eliminated until a minimum structure with acceptable modeling performance is achieved [Scha97]. Representative techniques of this class include magnitude based pruning (MBP) [Seti00], weight decay method [Chow94], and optimal brain surgeon (OBS) [Hass93]. For the constructive methods, a network is initialized with an over-simple structure and the topology gradually augments until the network performance is satisfactory. Cascade correlation [Fahl90] is the most popular one of the constructive algorithms.

The third class of methods uses other optimization techniques to determine the topology of networks. These methods include genetic algorithm-based techniques [Ezug95] [Diml00] [Habe03] and Bayesian regulation-based technique [Ozel05].

For the developed optimization methods, there are still some common problems existed:

1) The optimization methods are case dependent. Different samples of training data will generate different optimal networks for the same system to be modeled.
2) It is not possible to guarantee that the resultant network structure is optimal. Most times the final structures are suboptimal.

In this study, a destructive optimization is applied to RNN to form an optimized RNN. The method can generate simple and accurate network which can better avoid the over-fitting problem and the network is proved to be more robust [Kris99].

Training Algorithms of Recurrent Neural Networks

A neural network can be viewed as a parametric model with weights and biases as its parameters. Once its architecture is determined, the parametric structure of NN model is fixed. Furthermore, its parameters (weights) need to be tuned then, which is called the training process. In other words, training is the process to determine the weights of a network to make it model the system been studied.

There are three major training classes - supervised training, unsupervised training and reinforcement training, each of which applies to particular learning tasks.

For supervised training, the neural network is provided with a set of training patterns (inputs along with the corresponding desired outputs – targets), and training involves the algorithm comparing its current actual output with the correct or target

outputs, so that it knows what its error is, and update weights accordingly. Usually supervised training is applied in modeling, estimation, and classification.

For unsupervised training, the neural network is not told the target - for example, it is not trained on pairs consisting of an input and the desired output. Instead the network is given the input patterns and is left to find interesting patterns, regularities, or clusterings among them. Usually unsupervised training is applied in clustering, compression applications.

For reinforcement training, it can be considered as an intermediate form of the above two types of training. A network interacts with the environment and gets a feedback response from it. Based on the environmental response, the network adjusts its weights.

This study focuses on modeling applications, and the supervised training is introduced in more details. As shown in Figure (2.14), a neural network is to model a system $\vec{y}^* = f(\vec{x})$. The system is unknown but a set of training patterns (a set of input $\vec{x}$, and its corresponding target $\vec{y}^*$) are available. A neural network is to simulate the system based on the information from these training patterns. The network receives input $\vec{x}$ and generates its output $\vec{y}$. The objective of training is to tune the network adjustable parameters (weight $\vec{w}$) to make $\vec{y} \approx \vec{y}^*$, so that modeling error is small and the network can represent the system.

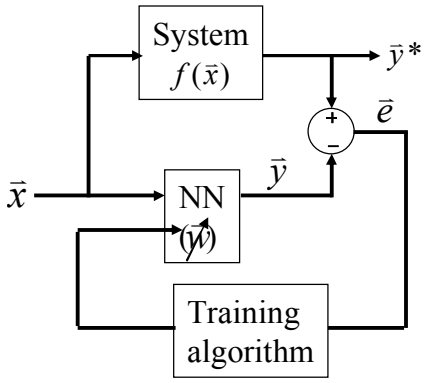


Figure 2.14: Illustration of the supervised training process

In training a neural network, the term epoch is used to describe a complete pass through all of the training patterns. The weights in the neural net may be updated after each pattern is presented to the net, or they may be updated just once at the end of the epoch.

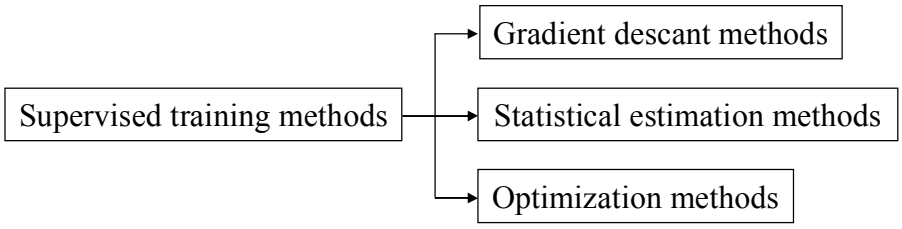


Figure 2.15: Classifications of the supervised training methods

There are several algorithms available to train neural networks. As shown in Figure (2.15), most of them can be viewed as applications of optimization theory and statistical estimation. Among them some popular training algorithms are:

1) Gradient descant methods such as the back-propagation (BP) method, which calculates the gradient of the modeling error of the network with respect to its

trainable weights and uses the gradient to guide the update of weights [Werb74],

2) Statistical estimation methods such as EKF based algorithm, which use EKF to estimate the weight update from training data [Sing89], and
3) Optimization methods such as genetic algorithm [Seif01] and simulated annealing [Boes93] which adopt optimization methods to minimize the cost function of training and to tune weights accordingly.

For RNN, as shown in Figure (2.16), there are three major training algorithms developed, namely, back-propagation through time (BPTT) algorithm, real-time recurrent learning (RTRL) algorithm and EKF training algorithm.

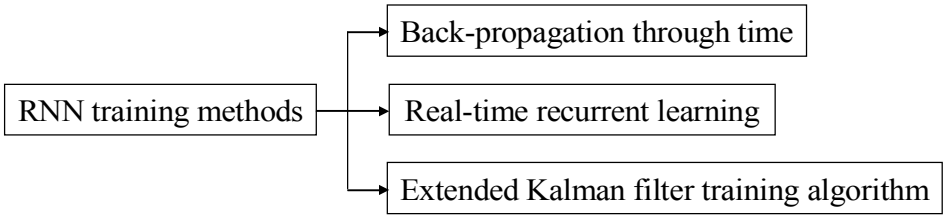
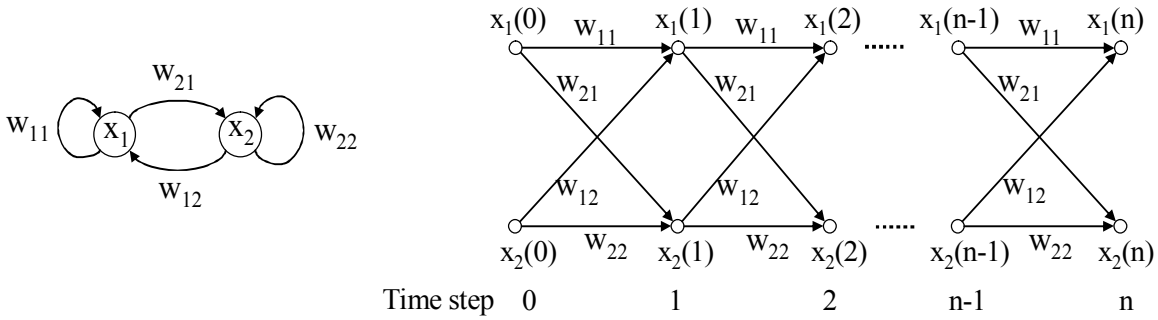


Figure 2.16: RNN training methods

The basic idea of BPTT is to unfold a RNN into a multilayer FFNN each time a sequence is processed [Rume86]. The resulting FFNN is then trained using the standard BP algorithm. An illustration of this process is shown in Figure (2.17). A recurrent network consists of two neurons and four weights. Since the layers have been obtained by replicating the RNN, the same weights in different layers should be the same. To achieve this, weights can only be updated at least after a complete forward step and a backward step to form a corresponding FFNN. The basic difference between it and the regular

back-propagation is that its desired responses ($x_1(i)$) are specified for neurons in layers of the network because the actual output layer is replicated many times when the temporal behavior of the network unfolded [Hayk99]. The unfolded network reflects the process where n represents the number of replication.



a) A recurrent network b) Signal flow graph of the unfolded network

Figure 2.17: An illustration of BPTT

RTRL computes the derivatives of states and outputs with respect to all weights as the network processes the sequence [Will89]. During the forward step, no unfolding is performed. Since derivatives of outputs are easily defined in terms of state derivatives, the trainable weights of RNN are updated after every time step in which output targets are available. This is one of the main advantages that RTRL can be used in online applications.

EKF algorithm was first introduced to train neural networks in [Sing89]. The network weights can be viewed as the states of the non-linear dynamical process that NN describes. The training of networks can be viewed as a parameter (trainable weights) estimation problem using state estimation methods such as the EKF algorithm. Comparing to the BPTT and RTRL algorithms, the EKF algorithm uses higher-order

information more efficiently. It is therefore much faster that the BPTT and RTRL algorithms, but at the expense of increment in computational complexity, which can be compensated by the rapid advancement in computing resources.

Due to its fast training speed and accuracy, the EKF training algorithm is adopted in this study as follows:

$$\vec{w}(k) = \vec{w}(k-1) - K(k)\left(\vec{y}(k) - \vec{y}^{*}(k)\right) \tag{2.14}$$

$$K(k) = P(k-1)H(k)[R(k) + H(k)^{T} P(k-1)H(k)]^{-1} \tag{2.15}$$

$$P(k) = P(k-1) - K(k)H(k)^{T} P(k-1) + Q(k) \tag{2.16}$$

where $\vec{w}(k)$ is the estimation of weight vector $\vec{w}^{*}$ at training step k, $\vec{y}$ is the output of neural network, $\vec{y}^{*}$ is the desired output. H is the Jacobian matrix which is comprised of $\frac{\partial \vec{y}}{\partial \vec{w}}$ - the derivative of output with respect to weight estimation, K is the Kalman gain calculated by Equation (2.15), Q is the approximate covariance matrix of process noise, and R is the approximate covariance matrix of measurement noise. The EKF algorithm requires, in addition to the estimate of the network's weight vector, the storing and updating of the approximate covariance matrix P, which is used to model the correlations or interactions between each pair of weights in the network.

In this study, EKF is used to train RNN own to its advantages in training speed and accuracy. However, for EKF application in RNN training, there are some problems, some of which will be studied in the following sections.

Convergence Studies of Recurrent Neural Networks

Generally speaking, convergence is defined as the property that a variable approaches toward a definite value, or a system approaches toward a fixed or equilibrium state as time goes on.

For non-linear dynamical systems, two major classes of methods have been applied in the study of convergence. Energy-based methods are based on the idea of passive energy, that is, if an energy function related to the state error (the difference of actual state and equilibrium state) is shown to be passive in some sense with respect to time, then the passivity implies the error will decay to zero in time, or in other words, the system is converge. Representative method in this class is the Lyapunov method. On the other hand, stability of equilibrium state which equivalents to the convergence can be also conducted using frequency domain analysis methods such as the circle criterion, the Popov criterion, and the describing function method.

There are many applications of RNN that relate to the network's convergence properties. Understanding the convergence properties of RNN is an initial and important step towards their applications [Yi06]. Generally speaking, convergence studies of RNN can be divided into three classes: state convergence, output convergence and training convergence.

State convergence is studied in applications such as content addressable memory when networks are required to have state convergence property [Cao03] [Lian01]. On the other hand, output convergence is of concerned in optimization applications [Li04] [Liu04].

Training convergence is concerned in modeling application where RNN are to be trained to map the relations among the systems. The training process is under studied to avoid the training divergence problem so that the developed RNN can model the system been studied.

For either case, convergence study of RNN desires to establish verifiable and sufficient conditions to guarantee convergence of the concerned process. Usually Lyapunov methods and energy functions method are adopted to conduct these studies.

Training Convergence of RNN

NN can be viewed as a multi-input and multi-output nonlinear system having a layered structure, and its weight learning/training algorithm can be regarded as parameter estimation for such a nonlinear system [Ligu92]. Two issues are of great importance in NN training: how to avoid training divergence and how to converge fast. Network training convergence is still a challenge in modeling input-output mapping relationships using NN, especially RNN. RNN training is still an open topic because network weight adjustments can affect the entire neural network state variables during the network evolution due to the inherent feedback and distributive parallel structure [Song08] and training is usually complex and might be divergent [Atiy00]. It should be pointed out that training convergence is different from the state or output convergence which is usually of concern in applying the trained RNN for associative memory applications [Tang07].

Among the most popular RNN training algorithms [Hayk99] such as back-propagation through time (BPTT), real time recurrent learning (RTRL), and extended

Kalman filter (EKF), EKF has been favored in terms of its training efficiency and accuracy [Matt90] [Leun03] [Liu06]. Unfortunately, training convergence of EKF-based RNN is still not well studied [Rubi07]. Up to present, only a few studies have been conducted on convergence of EKF-based neural network training [Ales03] [Rubi07] including RNN training [Rubi07]; unfortunately, they have introduced many assumptions to make these pioneering studies less generic and less efficient. For effective implementation of EKF-based RNN training, some theoretical studies must be performed and tested with some applications.

The objective of the training convergence study is to develop an effective EKF-based RNN training approach with a controllable training convergence. While EKF has been proved to be very useful in a wide variety of estimation or prediction applications, its effectiveness can be nullified by its divergence [Fitz71], which can be classified as follows [Schl67]: 1) apparent divergence, in which the associated errors don't approach infinity but the errors are too large to allow the estimates to be useful, and 2) true divergence, in which the mean square errors of estimation can actually approach infinity as training goes on, and this true divergence is of interest in this study.

There are several approaches have been proposed to deal with the divergence problem in EKF [Simo06]: 1) to increase the arithmetic precision, 2) to artificially add white noise to the (noiseless) process equation, 3) to use square root Kalman filters, 4) to make the state estimation error covariance matrix P symmetric, 5) to use a fading-memory Kalman filter, and 6) to adapt filter parameters. Evaluation of the effectiveness of the aforementioned approaches is often difficult and case-dependent. The first

approach is suitable for applications dealing with hardware implementation in which the high precision in hardware is often prohibitive. The second to fourth approaches aim to make the covariance matrix P nonnegative definite and/or symmetric. During the filtering process the covariance matrix P may fail to meet the nonnegative definite and/or symmetric requirements, resulting in the divergence problem. This can be alleviated by artificially adding a process noise (the second approach) and the magnitude of the additive noise is chosen to be large enough to ensure that the P matrix is nonnegative [Zarc01]. The square root Kalman filter is a more refined method to solve this divergence problem, and the covariance matrix is propagated in a square-root form by using the Cholesky factorization. However, the square root algorithm is computationally intensive which makes it less attractive in engineering applications [Harv89]. The fading-memory filter is another way of forcing the filter to forget measurements in the distant past and place more emphasis on recent measurements; however, it may result in the loss of optimality of the Kalman filter [Simo06]. The sixth approach is of interest in this study by adapting the covariance of measurement noise (R) and the covariance of process noise (Q) of Kalman filter. It is recognized that the poor statistics about R and Q may cause the divergence problem in estimation using the Kalman filter [Jwo07], so this study will investigate the EKF training algorithm stability in RNN training by adaptively adjusting the two noise covariance matrices R and Q.

Estimation Robustness of RNN

Most dynamic systems have a capacity, which is generally called robustness, to tolerate various system variations without exceeding predetermined tolerance bounds in the vicinity of nominal dynamic behaviors. Robustness analysis is usually studied to estimate the perturbation-induced performance variation or to quantify the system's resilience to any possible perturbations.

Analysis of NN robustness has been of great interest since the network robustness information allows the researchers to have a global and synthetic understanding of the network behavior under uncertainties. A robust network is expected to be fault tolerant and noise immune; if the inputs or the parameters (weights and others) of a network are contaminated with noise, or faults occur, the network response should differ only slightly with respect to the ideal performance [Eick07].

NN robustness has been studied for different applications including associative memory, classification, and modeling. For associative memory applications, the robustness is usually studied by establishing sufficient conditions for valid memory functions under uncertainties of network parameters such as weight and bias [Liu93] [Liu96] [Feng99] [Arik03] [Liu06]. For classification applications, the robustness is conducted by investigating the relationship between permissible variations of inputs and the associated network classification performance [Pier06]. On the other hand, for modeling applications, the robustness is characterized by studying the effects of perturbations in weights [Yee91] [Kris99] [Alip02] [Alip04] or inputs [Eick07] on network outputs.

The NN robustness in modeling applications has been of great interest. The goal for these applications is to reduce the sensitivity of modeling capacity to uncertainties in parameters, or to make the network fault tolerance.

Dynamic systems can be modeled using different approaches including the data-driven NN method [Hayk99]. When a system is represented by a NN-based model, it is naturally expected that the NN should have certain robustness to various perturbations [Chiu93] [Alip01] [Alip04]. For example, NN should still accurately describe system behaviors even its weights are altered due to different reasons:

1) Hardware drifting over a period of time [Chiu93]
2) Hardware implementation of analog and/or digital circuitry of NN in current technologies such as quantization and environmental noise [Dund95] [Raza00] [Wido02] [Alip04] [Eic07]
3) Software perturbations [Asso04], and
4) Neural network faults which includes disconnection or saturation of weights and lost of neurons [Phat95]

While the effect of input uncertainties on the NN robustness has been studied [Pier06], the effect of weight alternation is usually of great interest in characterizing the NN robustness [Alip04]. As aforementioned the network weights are easy to be altered during various NN implementation scenarios, and robustness analysis on the effect of network weight perturbations has an immediate impact on NN physical realization [Alip04].

The NN robustness in modeling applications has been of great interest, and it has been studied mainly using the performance loss-based approach [Chiu93] [Alip04] [Eick07] and the sensitivity matrix-based approach [Yee91] [Kris99]. While these approaches have been developed for FFNN, they can also be extended to RNN.

The performance loss-based approach is usually realized by computing the network modeling capability degradation due to any perturbation in its parameters such as weights. The performance loss is characterized in terms of the mean square error (MSE) over available measurement data sets [Chiu93] by introducing certain perturbations in trained network parameters such as weights. Perturbations can be introduced using a constant scaling factor which linearly changes the value of parameters of interest [Chiu93] [Alip04] and using a certain probability distribution function such as the Gaussian distribution [Eick07] and uniform distribution [Dund95] [Alipp04]. The upper boundary of performance loss for all the input data indicates the network robustness. Unfortunately, this approach is implemented using the measurement data and requires a large amount of measurement data, which are usually limited in real applications.

The general procedures to conduct performance loss-based robustness quantification are concluded as follows:

1) Introduce certain perturbation in a trained NN's parameters (i.e. weight) to form a series of NN,

2) Feed available data pairs, inputs and their corresponding measurements (target response), to each of NN formed in Step (1) and compute corresponding MSE as the performance loss using NN model,

3) Use the maximum of the performance loss for all the input data to represent the network's robustness.

The aforementioned MSE in Step (2) is computed by:

$$MSE(x_i, \Delta\theta) = \frac{1}{n_p} \sum_{i=1}^{n_p} \left(t(x_i) - y(x_i, \Delta\theta) \right)^2 \tag{2.17}$$

where n_p is the number of available data set, x_i is the input, $t(x_i)$ is the target value for NN output $y(x_i)$, and $\Delta\theta$ is the perturbation in parameters.

For the sensitivity matrix-based approach, the robustness is studied using a differential analysis to compute the parameter-output sensitivity matrix of NN. The sensitivity matrix, often denoted as H, is the Jacobian matrix containing the derivatives of outputs with respect to parameters such as weights:

$$H = \begin{bmatrix} \frac{\partial y_1}{\partial w_1} & \frac{\partial y_1}{\partial w_2} & \cdots & \frac{\partial y_1}{\partial w_{n_t}} \\ \frac{\partial y_2}{\partial w_1} & \frac{\partial y_2}{\partial w_2} & \cdots & \frac{\partial y_2}{\partial w_{n_t}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_n}{\partial w_1} & \frac{\partial y_n}{\partial w_2} & \cdots & \frac{\partial y_n}{\partial w_{n_t}} \end{bmatrix} \tag{2.18}$$

where $\vec{w} = \left[w_1, w_2, ..., w_{n_t} \right]$ is the weight vector and $\vec{y} = \left[y_1, y_2, ... y_n \right]$ is the output vector

A norm of the sensitivity matrix, such as the 2-norm square $\|H\|_2 = \sum_{i,j} H_{ij}^2$ [Yee91] or the spectral norm $\|H\|_s = \sqrt{\lambda_{\max}\left[H^T H\right]}$ [Kris99], is used as the robustness index. Similar to the performance loss-based approach, this approach also needs plenty of measurement data sets to compute the sensitivity matrices so that the resultant robustness measure can cover the whole input space. Furthermore, each sensitivity matrix only reflects the sensitivity of an infinitesimal range centered around the nominal weight values.

The general procedures involved in this approach are listed as follows:

1) For each input data $\vec{x}(i)$, compute the sensitivity matrix H during the training process; calculate its norm, and
2) Use a statistic (average) of the norm values to indicate the network's robustness.

As the aforementioned two approaches are mainly limited by the available measurement data, this study aims to quantify the network robustness by computing weight perturbation-induced output uncertainties using an uncertainty propagation analysis.

Conclusions

This chapter reviews the theoretical background of this dissertation. The architectures of networks, topology optimization, and training algorithm are reviewed in

the first. Performance studies of NN such as the training convergence and estimation robustness are then introduced.

Neural networks include forward neural networks (MLP, FFCNN) and recurrent neural networks (Elman, Jordan, and RMLP) applied in modeling applications are introduced. The recurrent neural networks are modified from MLP which is the most widely used feedforward network. In this study another RNN is to be proposed based on the FFCNN which has advantages over MLP.

NN with different levels of complexity (in terms of numbers of neurons, layers and weights) can be applied for modeling non-linear dynamical systems. Usually a complicated network can generate small modeling error. However, the use of complicated networks is time consuming and often brings the over-fitting problem. To determine the optimized network for a specific application, generally there are three classes of methods: empirical method, destructive or constructive methods, and the applications of other optimization strategies. Usually a destructive method is preferred because the resulting optimized network is parsimonious and often has good extrapolation ability.

Training algorithm is used to determine the weights of a network. Three major training algorithms are developed for RNN, namely, back-propagation through time (BPTT) algorithm, real-time recurrent learning (RTRL) algorithm and EKF training algorithm. Among them, the EKF is the most accurate and fastest one which is applied in this study.

Two performance studies are also reviewed in this chapter. Training convergence studies the stability of weight update during training process and it is interested in this

study. Generally the Lyapunov method is applied to establish verifiable and sufficient conditions to guarantee convergence of the training process. To avoid training divergence, or fail of convergence, is still a major challenge in application of EKF based RNN training and the problem is to be addressed in this study by adapting some training parameters.

Finally, robustness of NN for modeling applications is reviewed in this chapter. The goal of the research is to reduce the sensitivity of a network's modeling capacity to uncertainties in its parameters. Two methods have been developed in quantifying robustness of FFNN, namely performance loss-based method and sensitivity matrix-based method. Both the methods are limited by the available measurement data. In this study an uncertainty propagation analysis based method is to be developed which is effective, efficient, and flexible to quantify robustness of RNN.

CHAPTER THREE

DEVELOPMENT OF THE RECURRENT NEURAL NETWORK

Abstract

In the study, a RNN and an optimized RNN are proposed to model non-linear dynamical systems. In this chapter, the development of RNN networks is introduced which include its structure, training algorithm, and architecture optimization algorithm. The network is modified from a fully forward connected network by the accommodation of one time step delayed internal recurrent connections in its hidden neuron section. The RNN EKF training algorithm is then introduced. The most time consuming part of the algorithm is to take the orderly derivative of network output with respect to trainable weights. The orderly derivative derivation is illustrated in three cases considering specific weight connections involved. The optimization of the network structure is achieved using a pruning approach which removes the insignificant connections. To conduct the structure optimization, first a connectivity coefficient is introduced to each connection through a connectivity function, then the coefficients are trained with weights simultaneously using EKF, and finally the unimportant connections are removed. With the techniques introduced in this chapter, an optimized RNN can be developed to model non-linear dynamical systems.

Nomenclature

Fully Forward Connected Neural Network	
Symbol	Definition
h	Number of the hidden neurons
m	Number of the input neurons
n	Number of the output neurons
$f_i(\cdot)$	Activation function of the *ith* neuron
net_i	Net input of the *ith* neuron
w_{ij}	Weight for the connection from neuron *j* to neuron *i*
y_i^o	Output of the *ith* neuron

Recurrent Neural Network	
Symbol	Definition
h	Number of the hidden neurons
m	Number of the input neurons
n	Number of the output neurons
$f_i(\cdot)$	Activation function of the *ith* neuron
$net_i(k)$	Net input of the *ith* neuron at time step *k*
w_{ij}	Weight for the connection from neuron *j* to neuron *i*
$y_i^o(k)$	Output of the *ith* neuron at time step *k*

Extend Kalman Filter Algorithm	
Symbol	Definition
$x^*(k)$	Actual state at time step k
$x(k)^-$	The a priori estimate of $x^*(k)$
$x(k)$ or $x(k)^+$	The a posteriori estimate of $x^*(k)$
$e(k)^-$	The a priori estimate error at time step k
$e(k)$	the a posteriori estimate error at time step k
$H(k)$	Jacobian matrix at time step k
$K(k)$	Kalman gain at time step k
$P(k)^-$	The a priori estimate error covariance at time step k
$P(k)$ or $P(k)^+$	The a posteriori estimate error covariance at time step k
$Q(k)$	Covariance matrix of process noise at time step k
$R(k)$	Covariance matrix of measurement noise at time step k
$\vec{w}(k)$	Estimation of weight vector $\vec{w}^*$ at training step k

Connectivity Optimization Algorithm for RNN	
Symbol	Definition
c_{ji}	Introduced connectivity coefficient for connection from neuron i to neuron j
$g(c_{ji})$	Connectivity function of c_{ji}

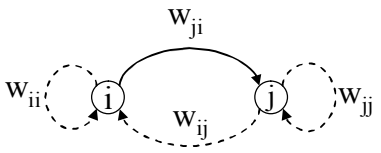
Architecture of the Proposed Neural Network

A fully connected recurrent neural network is proposed in this study. The network is developed from an FFCNN introduced in Chapter two.

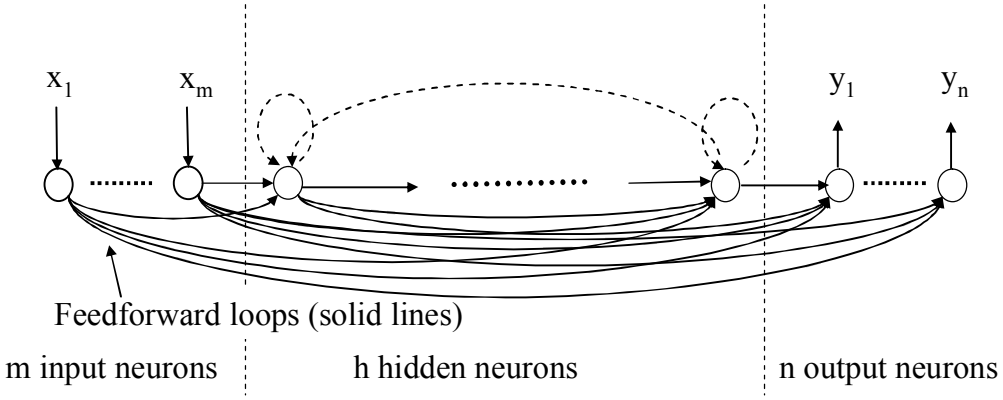
As shown in Figure (3.1(b)), the RNN is comprised of m neurons in its input section, h neurons in its hidden section and n neurons in its output section. In addition to the forward connections in FFCNN (shown in Figure (2.6)), each neuron in the hidden section also takes one time step delay feedback connections from the neurons right to it. Hence, the RNN is fundamentally different from an FFNN in the sense that it not only operates on an input space but also on an internal state space – a trace of what already has been processed by the network.

Figure (3.1(a)) shows the internal recurrency between a neuron *i* and a neuron *j* in the hidden section. w_{ij} represents the weight for the feedback connection between the two neurons, w_{ii} and w_{jj} represent the weights for the two neurons' self-feedback connections, while the w_{ji} represents the weight for the feed forward connection. Notice that the recurrency only exist in network's hidden section, and the other weights are for feedforward connections as in an FFNN.

The proposed RNN has intra-neuron internal recurrency (the dashed lines in Figure (3.1(b))) in its hidden section. Different from other RNN (Elman, Jordan, and RMLP) mentioned in Chapter 2 which are based on the MLP network, the proposed RNN is modified from the FFCNN. Because the FFCNN has advantages over the MLP in terms of modeling accuracy and robustness, the proposed RNN is believed to be better than those RNN.



(a) Connection between two neurons



(b) The proposed RNN structure

(Dash lines: state feedback loops. Solid lines: feed forward loops)

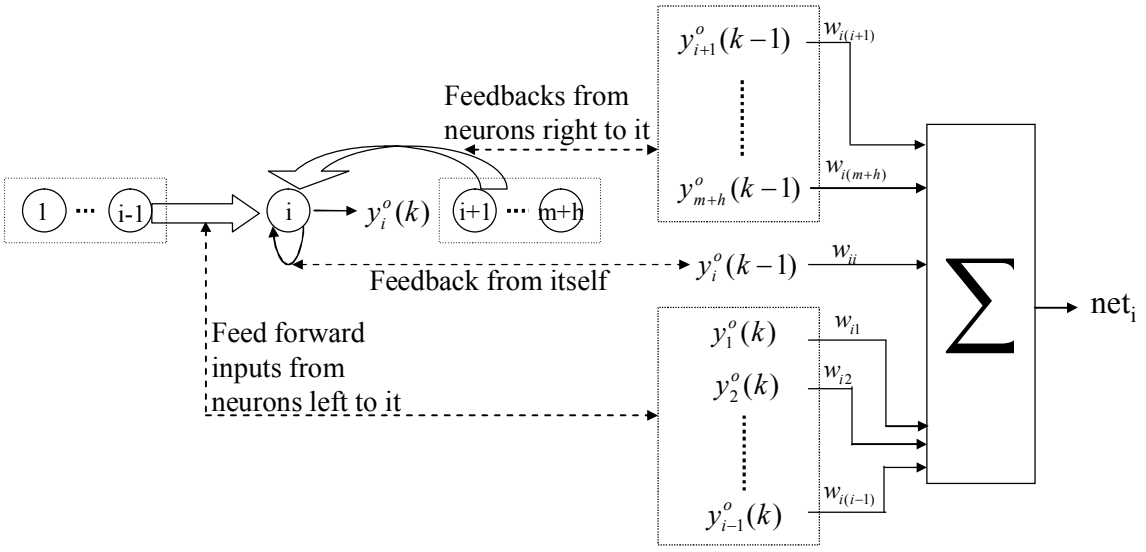
Figure 3.1: Architecture of RNN

For each neuron *i*, its output $y_i^o(k)$ at the time step *k*, is determined by the neuron's activation function $f_i(\cdot)$ and its net input $net_i(k)$:

$$y_i^o(k) = f_i(net_i(k)), \qquad 1 \le i \le m+h+n \tag{3.1}$$

Same to the FFCNN case (shown in Figure (2.7)), the net input for neurons in input neuron section and output neuron section is calculated as:

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij} y_j^o(k), \qquad i \le m \text{ or } i > m+h \tag{3.2}$$



a) Schematic of the connections b) Net_i composition

Figure 3.2: An illustration of the output generation of neuron *i* in hidden section

As shown in Figure (3.2), due to the feedback connections introduced in the hidden neuron section, the functions for hidden neurons are different from those in the previous FFCNN case. The net input of each neuron in the hidden section is comprised of two parts: the summation of outputs at the current step from the neurons left to it and the summation of outputs at the previous step of hidden neurons right to it.

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij}\, y_j^o(k) + \sum_{j=i}^{n_n} w_{ij}\, y_j^o(k-1), \qquad m < i \le m+h \tag{3.3}$$

where $n_n = m + h + n$ is the total number of neurons of network, activation function, and $y_j^o(k-1)$ is the output of neuron *j* at the previous time step (*k-1*).

Training Algorithm Development

As stated in Chapter 2, both BPTT and RTRL are based on first order gradient descendent and are heavy computationally and slow. GA is also computation demanding because the weight solution space of RNN is usually quite big. In this study an EKF algorithm is applied to train RNN. First a brief introduction of EKF is given in the following.

Introduction of Kalman Filter

Generally speaking, EKF [Mayb90] can be used to estimate states of a nonlinear system from its measurement history. The EKF is derived from Kalman filter (KF) which applies for linear systems. To better appreciate EKF, KF is introduced first as follows.

Kalman Filter

The objective of KF is to estimate the state x^* of a linear discrete-time system:

$$\begin{cases} x^*(k) = A(k-1)x^*(k-1) + G(k-1)u(k-1) + \gamma(k-1) \\ y^*(k) = H(k)x^*(k) + \xi(k) \end{cases} \tag{3.4}$$

where the first equation is called process equation, the second one is called measurement equation, x^* is the state of system at time step *k*, $u(k-1)$ is the control input at time step *k-1*, γ is the process noise, y is the measurement, ξ is the measurement noise, the matrices $A(k-1)$ and $G(k-1)$ relate the state and input at the previous time step *k-1* to the state of current time step *k*, $H(k)$ relates the state to measurement at the current time step, the noises are white, zero-mean, uncorrelated, and have known covariance matrices $Q(k)$ and $R(k)$ respectively:

$$\gamma(k) \sim (0, Q(k)) \tag{3.5}$$

$$\xi(k) \sim (0, R(k)) \tag{3.6}$$

$$\mathrm{E}\left[\gamma(k)\gamma(j)^T\right] = Q(k)\delta(k-j) \tag{3.7}$$

$$\mathrm{E}\left[\xi(k)\xi(j)^T\right] = R(k)\delta(k-j) \tag{3.8}$$

$$\mathrm{E}\left[\gamma(k)\xi(j)^T\right] = 0 \tag{3.9}$$

where $\delta(k-j)$ is the Kronecker delta function:

$$\delta(k-j) = \begin{cases} 1, & \text{if} \quad k = j \\ 0, & \text{if} \quad \mathrm{k} \neq \mathrm{j} \end{cases} \tag{3.10}$$

The goal of KF is to estimate the state $x^*(k)$ from the system dynamics (3.4) and the noisy measurement sequence $\{y^*(1), y^*(2),..., y^*(k)\}$. If all of the measurements before the current time step *k* are available, they can be used to form an a priori estimate of $x^*(k)$ as:

$$x(k)^- = \mathrm{E}\left[x^*(\mathrm{k}) \middle| y^*(1), y^*(2),..., y^*(k-1)\right] \tag{3.11}$$

The corresponding covariance of the estimation error is:

$$P(k)^- = \mathrm{E}\left[\left(x^*(\mathrm{k}) - x(k)^-\right)\left(x^*(\mathrm{k}) - x(k)^-\right)^T\right] \tag{3.12}$$

The state can be further estimated based on the availability of the measurement at current time step, to form an a posteriori estimate of $x*(k)$ as:

$$x(k)^+ = \mathrm{E}\left[\mathrm{x(k)} \middle| y^*(1), y^*(2),..., y^*(k)\right] \tag{3.13}$$

The corresponding covariance of the estimation error is:

$$P(k)^+ = \mathrm{E}\left[\left(x^*(\mathrm{k}) - x(k)^+\right)\left(x^*(\mathrm{k}) - x(k)^+\right)^T\right] \tag{3.14}$$

The KF algorithm is formed in a recursive way, to estimate $x^*(k)$ based on the estimation of $x^*(k-1)$. The a priori estimate ($x(k)^-$) and estimation covariance ($P(k)^-$) are calculated considering the process equation. The a posteriori estimate ($x(k)^+$) and estimation covariance ($P(k)^+$) are computed based on the knowledge of measurement (y^*) at time step *k*. The corresponding time update equations (3.15 and 3.16) and measurement equations (3.17-3.19) are listed as follows:

$$x(k)^- = A(k-1)x(k-1)^+ + G(k-1)u(k-1) \tag{3.15}$$

$$P(k)^- = A(k-1)P(k-1)^+ A(k-1)^T + Q(k-1) \tag{3.16}$$

$$K(k) = P(k)^- H(k)^T \left(H(k)P(k)^- H(k)^T + R(k)\right)^{-1} \tag{3.17}$$

$$x(k)^+ = x(k)^- + K(k)\left(y(k) - H(k)x(k)^-\right) \tag{3.18}$$

$$P(k)^+ = \left(I - K(k)H(k)\right)P(k)^- \tag{3.19}$$

To begin the recursive update, the estimation should be initialized as follows:

$$x(0)^+ = \mathrm{E}\left[x^*(0)\right] \tag{3.20}$$

$$P(0)^+ = \mathrm{E}\left[\left(x^*(0) - x(0)^+\right)\left(x^*(0) - x(0)^+\right)^T\right] \tag{3.21}$$

The KF is illustrated in Figure (3.3) as follows:

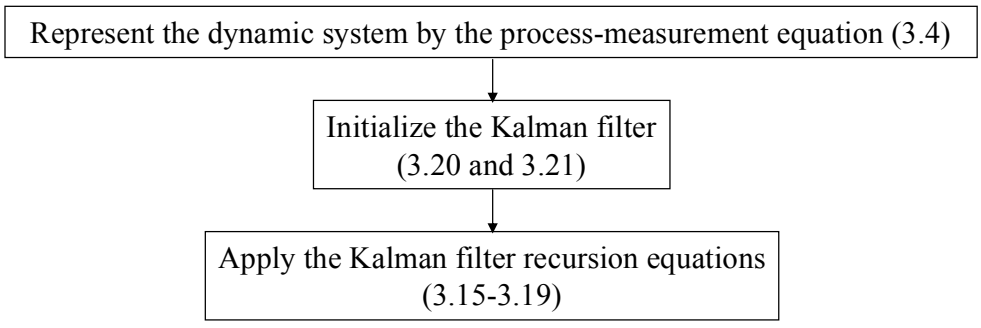


Figure 3.3: The Kalman filter algorithm

Introduction of Extended Kalman Filter

When the system being studied is a nonlinear system, the EKF is modified from the KF to make state estimation. The basic idea is to linearize the system first and then to apply KF on the linearized system. Suppose the original nonlinear system is governed by the equations:

$$\begin{cases} x^*(k) = f\left(x^*(k-1), u(k), \gamma(k-1)\right) \\ y^*(k) = h\left(x^*(k), \xi(k)\right) \end{cases} \tag{3.22}$$

where $x^*(k)$ and $y^*(k)$ are the actual state and measurement at current time step k, $f(\cdot)$ represents the nonlinear process equation, $h(.)$ represents the nonlinear measurement equation, u is the input of the system, and γ and ξ are the process noise and measurement noise which have the same properties as in Equations (3.5-3.9) :

To implement EKF, the system is first linearized at the approximate point $(x(k)^-, y(k)^-)$, or the a priori estimation point, which comes from the noise free system:

$$\begin{cases} x(k)^- = f\left(x*(k-1), u(k), 0\right) \\ y(k)^- = h\left(x(k)^-, 0\right) \end{cases} \tag{3.23}$$

$$\begin{cases} x^*(k) = x(k)^- + A(k-1)\left(x^*(k-1) - x(k-1)^-\right) + \lambda(k-1)\gamma(k-1) \\ y^*(k) = y(k)^- + H(k)\left(x^*(k) - x(k)^-\right) + V(k)\xi(k) \end{cases} \tag{3.24}$$

where $x(k)^-$ and $y(k)^-$ are the approximate state and measurement at time step k respectively, A is the Jacobian matrix of partial derivatives of f with respect to x^*, H is the Jacobian matrix of partial derivatives of h with respect to x^*, λ is the Jacobian

matrix of partial derivatives of f with respect to γ, and V is the Jacobian matrix of partial derivatives of h with respect to ξ.

$$A(k) = \frac{\partial f}{\partial x}\Big|_{x^*(k)} \tag{3.25}$$

$$\lambda(k) = \frac{\partial f}{\partial \gamma}\Big|_{0} \tag{3.26}$$

$$H(k) = \frac{\partial h}{\partial x}\Big|_{x^*(k)} \tag{3.27}$$

$$V(k) = \frac{\partial h}{\partial \xi}\Big|_{0} \tag{3.28}$$

Consider the estimation residual which is the difference between the actual state and the approximate state:

$$e_x^*(k) = x^*(k) - x(k)^- \tag{3.29}$$

The measurement residual is:

$$e_y^*(k) = y^*(k) - y(k)^- \tag{3.30}$$

Plug the above equations (3.29 and 3.30) into the linearized equation (3.24) to get the error model:

$$e_x^*(k) = A(k-1)e_x^*(k-1) + \varepsilon(k) \tag{3.31}$$

$$e_y^*(k) = H(k-1)e_x^*(k) + \eta(k) \tag{3.32}$$

Notice that the error equations are linear models hence the KF can be applied to compute the error estimate $e_x^*(k)$.The noises of the error system follows the following distributions:

$$\varepsilon(k) \sim \left(0, \lambda(k)Q(k)\lambda(k)^T\right) \tag{3.33}$$

$$\eta(k) \sim \left(0, V(k)R(k)V(k)^T\right) \tag{3.34}$$

After obtain the error estimate, the a posteriori estimate of $x^*(k)$ can be computed from the relation:

$$x(k)^+ = x(k)^- + e_x^*(k) \tag{3.35}$$

In conclusion, the EKF algorithm includes time update equations (3.36 and 3.37) and measurement equations (3.38-3.40):

$$x(k)^- = f\left(x^*(k-1), u(k), 0\right) \tag{3.36}$$

$$P(k)^- = A(k)P(k-1)A(k)^T + \lambda(k)Q(k-1)\lambda(k)^T \tag{3.37}$$

$$K(k) = P(k)^- H(k)^T \left(H(k)P(k)^- H(k)^T + V(k)R(k)V(k)^T\right)^{-1} \tag{3.38}$$

$$x(k)^+ = x(k)^- + K(k)\left(y^*(k) - h\left(x(k)^-, 0\right)\right) \tag{3.39}$$

$$P(k)^+ = \left(I - K(k)H(k)\right)P(k)^- \tag{3.40}$$

where $x(k)^-$ is the a priori estimate of $x^*(k)$ - the actual state to be estimated, $x(k)^+$ is the a posteriori estimate of $x^*(k)$, K is the Kalman gain, P^- is the a priori estimate error covariance $P^- = E\left[e^- e^{-T}\right]$, e^- is the a priori estimate error $e^- = x - x^-$, P^+ is the a posteriori estimate error covariance $P^+ = E\left[e\ e^T\right]$, and $e = x - x^+$ is the a posteriori estimate error. P^+ will be written as P for simplicity in the following sections.

The EKF algorithm is illustrated in Figure (3.4) as follows:

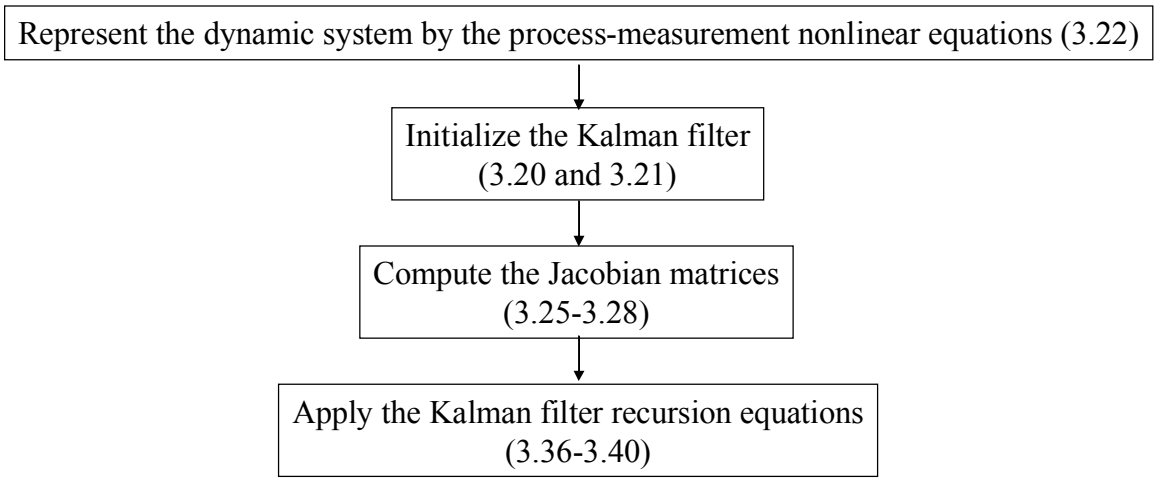


Figure 3.4: The extended Kalman filter algorithm

EKF RNN Training Algorithm

The EKF algorithm introduced above is for estimation of a scalar, but it can also apply to vectors as well. To apply EKF in RNN training, weights of a RNN are treated as the states of the network, and then EKF is used to estimate the states from network's output. Corresponding to Equation (3.22), the training model is represented by:

$$\begin{cases} \vec{w}^*(k) = \vec{w}^*(k-1) + \vec{\gamma}(k-1) \\ \vec{y}^*(k) = h\left(\vec{w}^*(k)\right) + \vec{\xi}(k) \end{cases} \tag{3.41}$$

where $\vec{w}^*(k)$ is the optimal weight vector $\vec{w}^*$ at time step k, $\vec{w}^*(k-1)$ is the weight vector at time step *k-1*, $\vec{y}^*$ is the measurement data or target data, and $h(\cdot)$ represents RNN mapping function which generates the output of network.

The optimal weight vector is the weight vector that minimize the difference of measurement $\vec{y}^*(k)$ and the output of neuron network $h\left(\vec{w}^*(k)\right)$. The EKF is used to estimate the constant vector $\vec{w}^*$ based on the measurements and the network function

$h(\cdot)$. Ideally $\vec{w}^*(k) = \vec{w}^*(k-1) = \vec{w}^*$ or $\vec{\gamma}(k-1) = \vec{0}$, but the non-zero process noise can introduce more flexibility in tuning the filter.

The objective of training process is to generate an estimate $\vec{w}(k)$ of $\vec{w}^*(k)$. The weights of a RNN are often represented in matrix form W; each of its element w_{ji} represents the weight on the connection from neuron *i* to neuron *j*. To apply the EKF, the elements of the matrix should be organized in the vector form $\vec{w}$. To achieve that, W is first written as a combination of row vectors: $W = \begin{bmatrix} \vec{W}_1 \\ \vec{W}_2 \\ \cdots \\ \vec{W}_{n_n} \end{bmatrix}$, $\vec{W}_i$ is the *ith* row of W, $n_n = m + h + n$ is the number of neurons of the network; $\vec{w}$ is then formed by combing the row vectors and taking transpose, $\vec{w} = [\vec{W}_1, \vec{W}_2, ..., \vec{W}_{n_n}]^T = \left[\underline{w}_1, \underline{w}_2, ..., \underline{w}_{n_t}\right]^T$; $\underline{w}_i$ is the *ith* element of the vector, n_t is the total number of trainable weights. For example, the weight matrix of the network in Figure (2.17) is $\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$; in this case,

$$\vec{w} = [\vec{W}_1, \vec{W}_2]^T = \begin{bmatrix} w_{11} & w_{12} & w_{21} & w_{22} \end{bmatrix}^T = \left[\underline{w}_1, \underline{w}_2, \underline{w}_3, \underline{w}_4\right]^T.$$

Corresponding to Equations (3.36-3.40), EKF for the weights update process can be written as:

$$\vec{w}(k)^- = \vec{w}(k-1) \tag{3.42}$$

$$P(k)^- = P(k-1) + Q(k-1) \tag{3.43}$$

$$K(k) = P(k)^- H(k)^T \left(H(k)P(k)^- H(k)^T + R(k)\right)^{-1} \tag{3.44}$$

$$\vec{w}(k) = \vec{w}(k)^{-} + K(k)\left(\vec{y}^{*}(k) - h\left(\vec{w}(k)^{-}\right)\right) \tag{3.45}$$

$$P(k) = \left(I - K(k)H(k)\right)P(k)^{-} \tag{3.46}$$

Combination of Equations (3.42-3.46) results in the simplified version of EKF training algorithm:

$$\vec{w}(k) = \vec{w}(k-1) - K(k)\left(\vec{y}(k) - \vec{y}*(k)\right) \tag{3.47}$$

$$K(k) = P(k-1)H(k)[R(k) + H(k)^{T} P(k-1)H(k)]^{-1} \tag{3.48}$$

$$P(k) = P(k-1) - K(k)H(k)^{T} P(k-1) + Q(k) \tag{3.49}$$

where $\vec{w}(k)$ is the estimate of weight vector $\vec{w}^{*}$ at time step *k*, $\vec{w}(k-1)$ is the estimation of weight vector at one time step before, $\vec{y}^{*}(k)$ is the target data, and $\vec{y}(k) = h\left(\vec{w}(k)^{-}\right)$ is the output of RNN which is supposed to be an estimate of $\vec{y}^{*}(k)$.

The flow chart of RNN training process is listed in Figure (3.5). The process includes four steps:

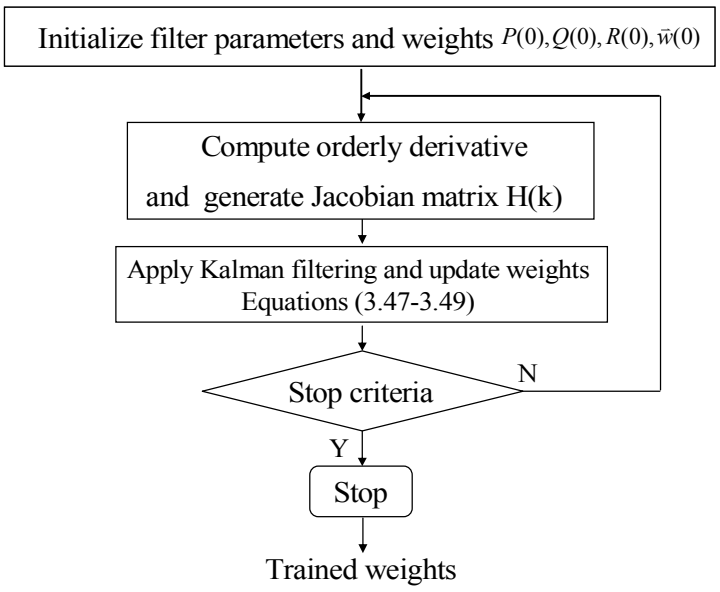


Figure 3.5: The flow chart of RNN training procedures

1) Training data preparation

NN is a data-driven modeling method. Suppose a system is to be modeled; the mathematical model of the system is unknown but its inputs-outputs data are available from experiments; an inputs-outputs pair is called a training pattern; a RNN is used to represent the system using these training patterns. Suppose the system to be modeled has m inputs and n targets, $\vec{y}^* = f(\vec{x})$, $\vec{x} = [x_1, x_2, ...x_m]$, $\vec{y}^* = [y_1^*, y_2^*, ...y_n^*]$, n_p sets of training patterns are generated as shown in Figure (3.6):

Training pattern 1: $x_1(1), x_2(1), ..., x_m(1) \longrightarrow y_1^*(1), y_2^*(1), ..., y_n^*(1)$
Training pattern 2: $x_1(2), x_2(2), ..., x_m(2) \longrightarrow y_1^*(2), y_2^*(2), ..., y_n^*(2)$
.....
Training pattern n_p: $x_1(n_p), x_2(n_p), ..., x_m(n_p) \longrightarrow y_1^*(n_p), y_2^*(n_p), ..., y_n^*(n_p)$

Figure 3.6: Training data set example

In order to avoid the saturation problem, all inputs and all outputs need to be normalized before they are fed into the network. The normalization is carried out by a linear mapping function as:

$$x_N = (x - x_{\min}) \frac{x_{N\max} - x_{N\min}}{x_{\max} - x_{\min}} + x_{N\min} \tag{3.50}$$

where x is the original variable in the range of $[x_{\min}, x_{\max}]$, $x_{\max}$ and $x_{\min}$ are the maximum and minimum values of the variable before normalization, x_N is the normalized variable, which is in the range of $[x_{N\min}, x_{N\max}]$, $x_{N\max}$ and $x_{N\min}$ are the maximum and minimum values of the normalized variable. Usually the variables are normalized into the range of [0, 1].

2) Training parameters initialization

To use the recursive equations of EKF, its parameters such as P, Q, R, and weights estimation $\hat{w}$ need to be initialized first. To avoid the saturation problem, each element of $\hat{w}_0$ is randomly selected from a uniform distribution in the range of [0, 1]. There is no general guide to initialize the other parameters. Usually they are arbitrarily selected and if the algorithm converge they can recursively converge to their desired values.

3) Training process

After that, training data are fed into the network to update weights. For a training pattern k, $\vec{x}(k) = [x_1(k), x_2(k), ..., x_m(k)]$ is the input to the network, and it generates output $\vec{y}(k) = [y_1(k), y_2(k), ..., y_n(k)]$ which is compared with the corresponding target value $\vec{y}^*(k) = [y_1^*(k), y_2^*(k), ...y_n^*(k)]$and the resulting error term is used to update weights as illustrated in Equation (3.47). To apply the EKF equations, the orderly derivatives (will be introduced in the following section) of network's output $\vec{y}$ with respect to weight vector $\vec{w}$ are calculated and form the Jacobian matrix $H = \begin{bmatrix} \frac{\partial^+ y_1}{\partial \underline{w}_1} & \frac{\partial^+ y_1}{\partial \underline{w}_2} & \cdots & \frac{\partial^+ y_1}{\partial \underline{w}_{n_t}} \\ \frac{\partial^+ y_2}{\partial \underline{w}_1} & \frac{\partial^+ y_2}{\partial \underline{w}_2} & \cdots & \frac{\partial^+ y_2}{\partial \underline{w}_{n_t}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^+ y_n}{\partial \underline{w}_1} & \frac{\partial^+ y_n}{\partial \underline{w}_1} & \cdots & \frac{\partial^+ y_n}{\partial \underline{w}_{n_t}} \end{bmatrix}$. The EKF equations (3.47-3.49) are then applied to train weights until the specified stop criteria are met, otherwise the process goes back to update

weights using training data for another epoch. A training epoch is used to describe a complete pass through all of the training patterns. For this case, the training process for training pattern 1 to pattern n_p is called a training epoch. Stop criteria are case dependent and may include the allowable maximum number of training epochs, the minimum training error, and the minimum amount of change in weights during training, etc. A normalized sum of square error (SSE) is used to represent the training error for a training epoch *j*:

$$e(j) = \sqrt{\left(\vec{y}(j) - \vec{y}^*(j)\right)^T \left(\vec{y}(j) - \vec{y}^*(j)\right) / \vec{y}^*(j)^T \vec{y}^*(j)} \times 100\% \tag{3.51}$$

where $\vec{y}^*(j)$ is the target data at epoch *j*, and $\vec{y}(j)$ is the output of RNN at epoch *j*.

Orderly Derivative

The most calculation consuming step in the above training flow chart is the step of calculating orderly derivative. RNN is an ordered system and the outputs of neurons need to be calculated orderly from the left to the right. Orderly derivatives are used for this case instead of the ordinary partial derivatives. For ordered systems where the values need to be orderly calculated in the order of $z_1, z_2, \ldots, z_n$ The orderly derivative of a target with respect of z_i can be written as [Werb90]:

$$\frac{\partial^+ \text{target}}{\partial z_i} = \frac{\partial \text{target}}{\partial z_i} + \sum_{j>i} \frac{\partial \text{target}}{\partial z_i} \frac{\partial z_j}{\partial z_i} \tag{3.52}$$

where the derivative with superscript represents orderly derivative and the derivative without superscript represents ordinary partial derivative. $\frac{\partial \text{target}}{\partial z_i}$, the ordinary partial,

derivative represents the direct impact of target on z_i, while $\frac{\partial^+ \text{target}}{\partial z_i}$, the orderly derivative, represents the total effect of target on z_i. A simple ordered system is illustrated for clarity as follows:

An ordered system is orderly governed by the following equations:

$$z_2 = 2z_1 \tag{3.53}$$

$$z_3 = z_1 + 2z_2 \tag{3.54}$$

From Equation (3.25), the ordinary partial derivative $\frac{\partial z_3}{\partial z_1}$ is 1, while the orderly derivative $\frac{\partial^+ z_3}{\partial z_1}$ is 5. The difference comes from the indirect impact by z_2.

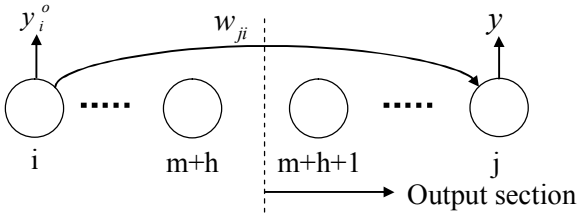
In RNN training, the orderly derivative matrix $H(k)$ is computed using the orderly chain rule considering all the possible connections contributing to the output. Each element $\frac{\partial^+ y_{i_1}}{\partial \underline{w}_{i_2}}$ is to be computed. The involved computation is introduced in the following. To better illustrate the process, the orderly derivative $\frac{\partial^+ y_{i_1}}{\partial \underline{w}_{i_2}}$ is denoted as $\frac{\partial^+ y}{\partial w_{ji}}$ where w_{ji} contains the connection information of $\underline{w}_{i_2}$. For simplicity, the subscript of $\hat{y}_{i_1}$ is omitted and w_{ji} is another notation for the weight $\underline{w}_{i_2}$. Generally the orderly derivative can be written as:

$$\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial^+ y}{\partial y_j^0} \frac{\partial y_j^o}{\partial net_j} \frac{\partial^+ net_j}{\partial w_{ji}} \tag{3.55}$$

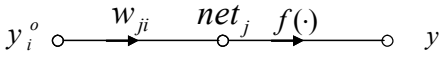
where y_j^o is the output of neuron j, net_j is the net input of neuron j.

The derivatives take different forms depending on the specific weight connections. The derivation is illustrated in the following:

Orderly Derivative Calculation in Case I



(a) Network connection involved in computing $\frac{\partial^+ y}{\partial w_{ji}}$



(b) Signal flow graph

Figure 3.7: Case I for calculation of the orderly derivative

Case I is the simplest case that only forward connections need to be considered to calculate the orderly derivative. In this case, for a weight w_{ji}, neuron j is in the output section and the neuron i is in the previous sections; only direct impacts act on y through the weight so that actually the ordinary derivative is considered here. Figure (3.7) shows an example in this case. Part (a) shows the connection from neuron i to neuron j. Part (b)

illustrates the signal flow graph from the output of neuron i $\left(y_i^o\right)$ to the output of network $\left(y\right)$, from which the orderly derivative can be written as:

$$\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial y}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial y}{\partial net_j} y_i^o \tag{3.56}$$

Orderly Derivative Calculation in Case II

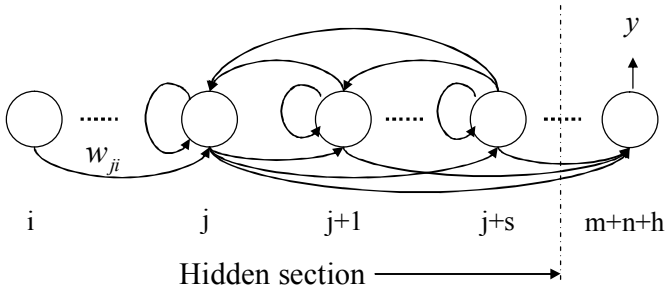
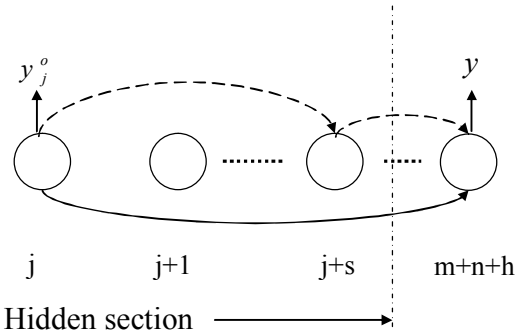
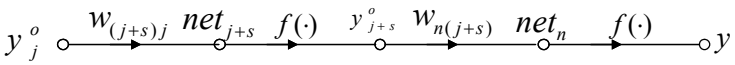


Figure 3.8: Case II for calculation of the orderly derivative

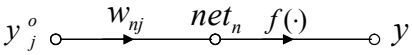
Case II deals with the situation that the weight w_{ji} is on a forward connection ($i<j$) and neuron j is in the hidden section. As shown in Figure (3.8), in this case, both forward loops and feedback loops need to be considered to derive the orderly derivative $\frac{\partial^+ y}{\partial w_{ji}}$.



(a) Illustration of direct impact (the solid line) and an indirect impacts (the dash lines) on $\frac{\partial^+ y}{\partial y_j^o}$



(b) The signal flow graph for the indirect impacts



(c) The signal flow graph for the direct impact

Figure 3.9: Illustration of calculation of $\frac{\partial^+ y}{\partial y_j^o}$ in case II

Figure (3.9) shows how to calculate the orderly derivative $\frac{\partial^+ y}{\partial y_j^o}$ in Equation (3.57). y_j^o, the output of neuron *j*, contributes to y not only directly through the weight $w_{n_n j}$ (the solid line in Figure (3.9 (a))), but also indirectly through other forward loops to the right of neuron *j* (the dash line in Figure (3.9 (a))).

$$\begin{aligned}\frac{\partial^+ y}{\partial y_j^o} &= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial y_{j+s}^o}\frac{\partial y_{j+s}^o}{\partial_j^o} \\ &= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial y_{j+s}^o}\frac{\partial y_{j+s}^o}{\partial net_{j+s}}\frac{\partial net_{j+s}}{\partial y_j^o} \\ &= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial y_{j+s}^o}\frac{\partial y_{j+s}^o}{\partial net_{j+s}} w_{(j+s)j}\end{aligned} \tag{3.57}$$

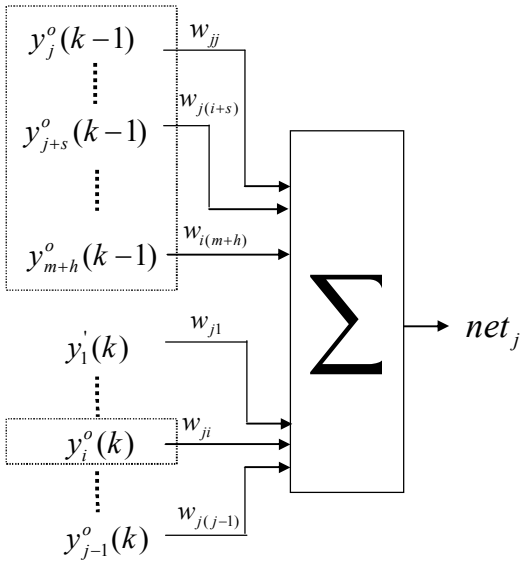


Figure 3.10: net_j decomposition (The items inside dash boxes contribute to the calculation of $\frac{\partial^+ net_j}{\partial w_{ji}}$)

Figure (3.10) shows the composition of net_j. It can be found that net_j has two components related to the orderly derivative $\frac{\partial^+ net_j}{\partial w_{ji}}$. One is the forward path from $y_i^o(k)$ through w_{ji} to net_j at the current time step *k*. The others are the paths from feedbacks $y_j^o(k-1), y_{j+1}^o(k-1), ..., y_{m+h}^o(k-1)$ which impact on $y_j^o(k-1)$ and hence indirectly influence on w_{ji}. *k* is used to denote time step because two consecutive time steps (current and the previous time step) need to be considered here. Figure (3.11) shows the effect of forward path at current time step *k* and the effect of a feedback path from $y_{j+s}^o(k-1)$ to $net_j(k)$.

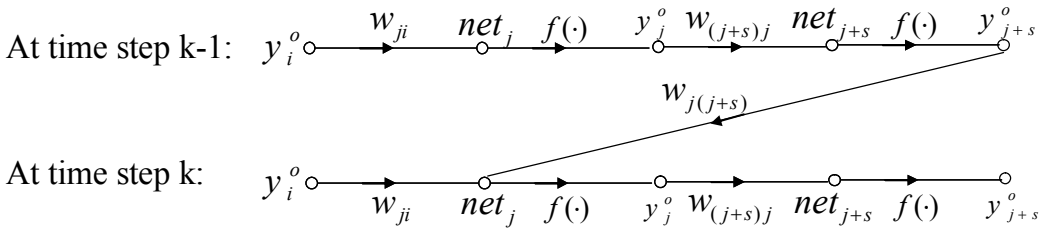


Figure 3.11: Signal flow graph to compute $\frac{\partial^+ net_j}{\partial w_{ji}}$

From Figure (3.11), the orderly derivative $\frac{\partial^+ net_j(k)}{\partial w_{ji}}$ can be computed as follows:

Consider the effect of forward path:

$$\frac{\partial^+ net_j(k)}{\partial w_{ji}} = y_i^o(k) \tag{3.58}$$

Consider the effect of feedback paths, as shown in the second item $\sum_{j=i}^{n} w_{ij} y_j^o(k-1)$ in Equation (3.3), the output of neurons at time step *k-1* also has impact on $net_j(k)$:

$$\begin{aligned}\frac{\partial^+ net_j(k)}{\partial w_{ji}} &= \frac{\partial^+ net_j(k)}{\partial y_j^o(k-1)} \frac{\partial y_j^o(k-1)}{\partial w_{ji}} \\ &= \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial y_{j+s}^o(k-1)} \frac{\partial^+ y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} \right) \frac{\partial y_j^o(k-1)}{\partial w_{ji}} \\ &= \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial y_{j+s}^o(k-1)} \frac{\partial^+ y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} \right) \times \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}}\end{aligned} \tag{3.59}$$

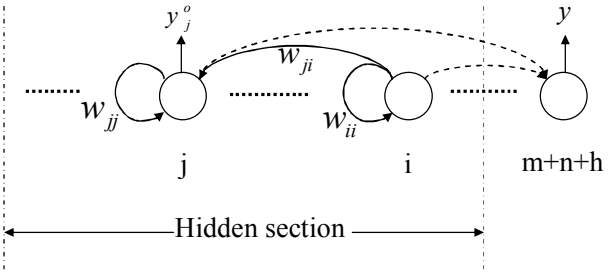
$$\frac{\partial^+ y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} = \frac{\partial y_{j+s}^o(k-1)}{\partial net_{j+s}(k-1)} \sum_{p=1}^{s} \frac{\partial net_{j+s}(k-1)}{\partial y_{j+s-p}^o(k-1)} \frac{\partial y_{j+s-p}^o(k-1)}{\partial y_j^o(k-1)} \tag{3.60}$$

Combine the above two equations, the orderly derivative is:

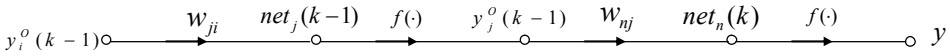
$$\frac{\partial^+ net_j(k)}{\partial w_{ji}} = y_i^o(k) + \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial y_{j+s}^o(k-1)} \frac{\partial^+ y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} \right) \times \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \tag{3.61}$$

Consider Equations (3.57, 3.61), Equation 3.55 can be written as:

$$\begin{aligned} \frac{\partial^+ y}{\partial w_{ji}} &= \frac{\partial^+ y}{\partial y_j^o} \frac{\partial y_j^o}{\partial net_j} \frac{\partial^+ net_j}{\partial w_{ji}} \\ &= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial y_{j+s}^o} \frac{\partial y_{j+s}^o}{\partial net_{j+s}} w_{(j+s)j} \frac{\partial y_j^o}{\partial net_j} \frac{\partial^+ net_j}{\partial w_{ji}} \\ &= \sum_{s=1}^{m+h-j} \frac{\partial^+ y}{\partial y_{j+s}^o} \frac{\partial y_{j+s}^o}{\partial net_{j+s}} w_{(j+s)j} \frac{\partial y_j^o}{\partial net_j} \\ &\quad \times \left(y_i^o + \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial y_{j+s}^o(k-1)} \frac{\partial y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} \right) \times \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \right) \end{aligned} \tag{3.62}$$



(a) Illustration of direct impact of w_{ji} on y . (Dash line: feedback connections. Solid line: forward connections)



(b) The signal flow graph for the impact

Figure 3.12: Case III for calculation of the orderly derivative

Orderly Derivative Calculation in Case III

Case III deals with the situation when the weight w_{ji} is on a feedback connection ($j<i$). In this case, both neuron *i* and neuron *j* are in the hidden section. As in Figure

(3.12), if weights are in the feedback loop ($j<i$), the orderly derivatives are computed as follows:

$$\begin{aligned}\frac{\partial^+ y}{\partial w_{ji}} &= \frac{\partial^+ y}{\partial y_j^o(k-1)} \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \\ &= \frac{\partial^+ y}{\partial y_j^o(k-1)} \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} y_i^o(k-1)\end{aligned} \tag{3.63}$$

Notice for this case the weight w_{ji} doesn't have impact on $y_i^o(k-1)$, because it would involve with calculations at 2 time steps before (k-2) and the network only considers one time step delayed recurrency.

For the above discussion, if don't mention otherwise, calculations are conducted for the current time step k. For example, net_j is the same as $net_j(k)$.

To better understand the classification of three cases, an example of trainable weights for each case is shown in Figure (3.13). The example network has a structure of 2-2-2. The rest weights are set to zero and don't need to be trained.

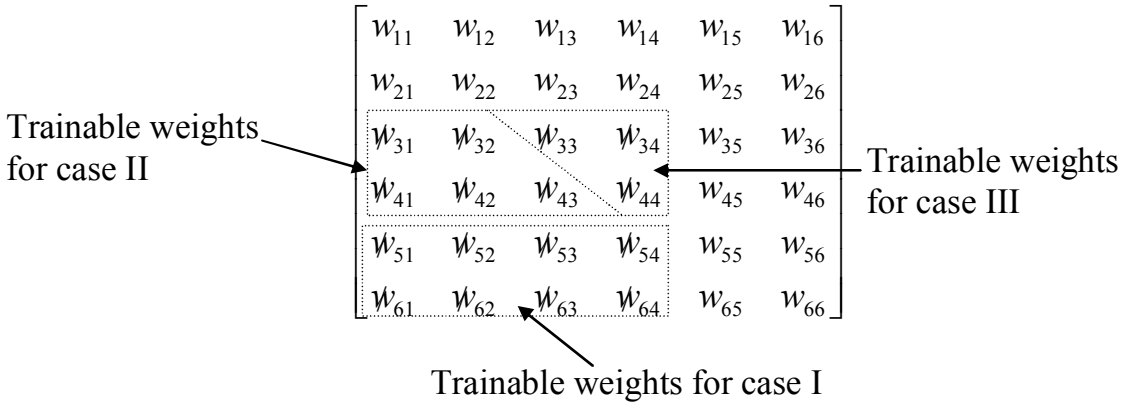


Figure 3.13: The trainable weights for the three cases

Connectivity Optimization Algorithm for RNN

In this study, a topology destructive optimization approach is utilized to optimize RNN. First, the number of hidden neurons is chosen based on a trial and error approach, then the network topology is optimized by disconnecting some weights among the network neurons using a method first proposed by KrishnaKumar [Kris93]. Such a pruned and optimized network has proved to be simpler, more accurate and robust [Kris99]. An example of the optimization process is illustrated in Figure (3.14), where the RNN originally has a structure of 1-3-1, and two connections (c_{31} and c_{24}) are disconnected after optimization.

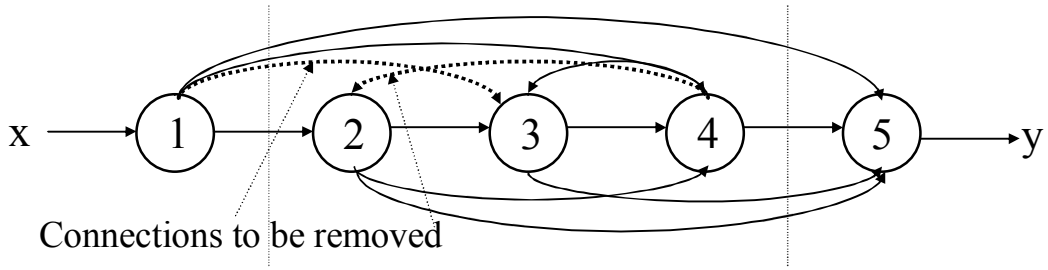


Figure 3.14: An illustration of connectivity optimization

To optimization the network connectivity, a function $g(c_{ij})$ as defined in Equation (3.64) is introduced to represent the availability of a connection between the neurons *j* and *i*. If $g(c_{ij})$ = 1.0, it implies there is a connection between the *ith* and *jth* neurons; while if $g(c_{ij})$ = 0, it implies there is no connection.

$$g(c_{ij}) = \frac{1}{1+e^{-c_{ij}}} \tag{3.64}$$

The procedures of optimization are listed as follows:

Derive the mapping functions of RNN considering the introduced connectivity coefficient c_{ji} for each connection $j \to i$, corresponding to Equations (3.1-3.3):

$$y_i^o(k) = f_i(net_i(k)), \qquad 1 \le i \le m+h+n \tag{3.65}$$

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij} y_j^o(k) g(c_{ji}), \qquad i \le m \text{ or } i > m+h \tag{3.66}$$

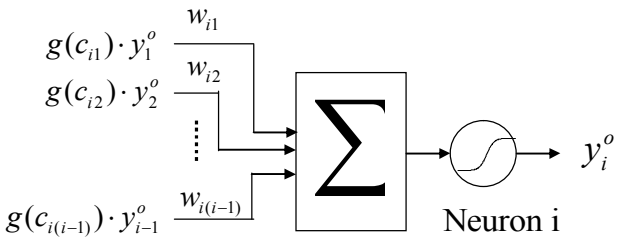


Figure 3.15: Illustration of Equations (3.65 and 3.66)

The equations (3.65 and 3.66) are illustrated in Figure (3.15). Compare to Figure (2.7), it can be seen the connectivity functions have been added into the net input of the concerned neuron. Similarly, the net input for hidden neurons can be computed as follows:

$$net_i(k) = \sum_{j=1}^{i-1} w_{ij} y_j^o(k) g(c_{ji}) + \sum_{j=i}^{nn} w_{ij} y_j^o(k-1) g(c_{ij}), m < i \le m+h \tag{3.67}$$

Calculate the orderly derivatives $\frac{\partial^+ y}{\partial w_{ij}}$ and $\frac{\partial^+ y}{\partial c_{ij}}$ accordingly.

The corresponding equations for computation of orderly derivatives are as follows:

For case I, corresponding to Equation (3.55):

$$\frac{\partial^+ y}{\partial w_{ij}} = \frac{\partial^+ y}{\partial y_j^o} \frac{\partial y_j^o}{\partial net_j} y_j^o g(c_{ji}) \tag{3.68}$$

Similarly, for case II, weights in forward loop $i<j$ and jth neuron is a hidden neuron:

$$\begin{aligned} \frac{\partial^+ y}{\partial w_{ji}} &= \frac{\partial^+ y}{\partial y_j^o(k)} \frac{\partial y_j^o(k)}{\partial net_j(k)} \\ &\times \left(y_j^o g(c_{ji}) + \left(\sum_{s=0}^{m+h-j} \frac{\partial net_j(k)}{\partial y_{j+s}^o(k-1)} \frac{\partial y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} \right) \times \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} \frac{\partial net_j(k-1)}{\partial w_{ji}} \right) \end{aligned} \tag{3.69}$$

$$\frac{\partial y_{j+s}^o(k-1)}{\partial y_j^o(k-1)} = \frac{\partial y_{j+s}^o(k-1)}{\partial net_{j+s}(k-1)} \sum_{p=1}^{s} \frac{\partial net_{j+s}(k-1)}{\partial y_{j+s-p}^o(k-1)} g(c_{(j+k)(j+k-p)}) \frac{\partial y_{j+s-p}^o(k-1)}{\partial y_j^o(k-1)} \tag{3.70}$$

For case III, weights in feedback loop $j<i$

$$\frac{\partial^+ y}{\partial w_{ji}} = \frac{\partial^+ y}{\partial y_j^o(k-1)} \frac{\partial y_j^o(k-1)}{\partial net_j(k-1)} y_i^o(k-1) g(c_{ji}) \tag{3.71}$$

Finally the orderly derivatives of RNN outputs to connectivity coefficients can be computed as:

$$\frac{\partial^+ y}{\partial c_{ji}} = \frac{\partial^+ y}{\partial w_{ji}} w_{ji} \frac{\partial g(c_{ji})}{\partial c_{ji}} / g(c_{ji}) \tag{3.72}$$

3) Then both the weights and connection coefficients are updated using the EKF algorithm. At the beginning of optimization, each c_{ij} is set as 0. When the stop criteria have been met, the connections with c_{ij} <0 will be disconnected by setting $g(c_{ij})$ = 0 and the others will be connected by setting $g(c_{ij})$ = 1.

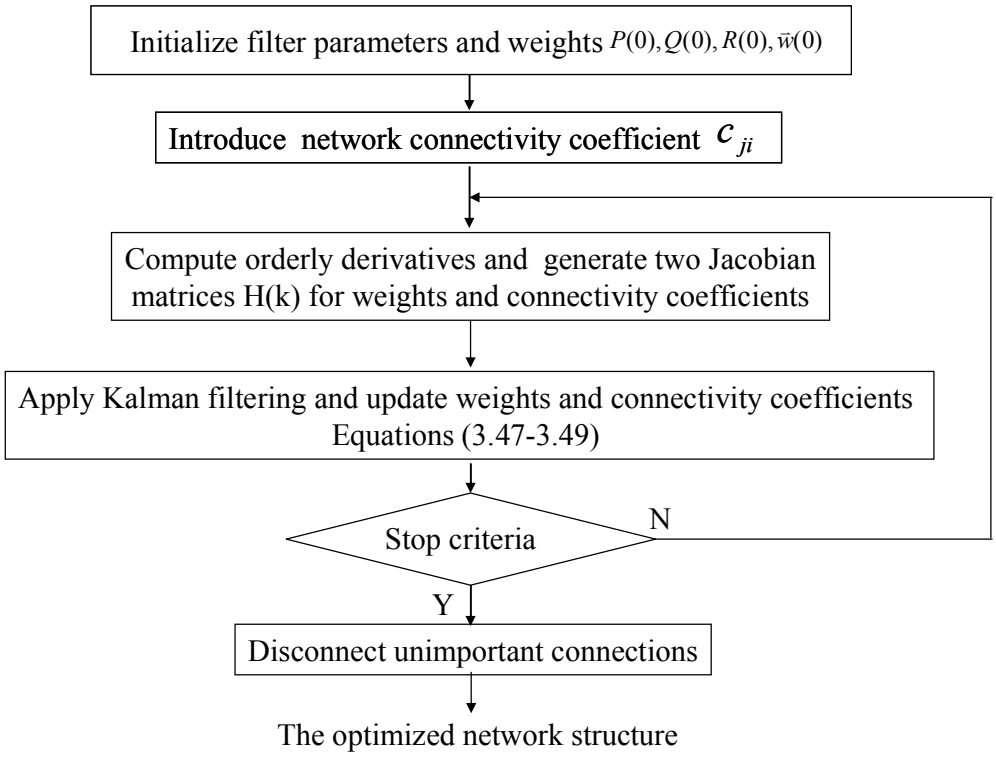


Figure 3.16: Network optimization process

As shown in Figure (3.16), an optimized RNN is formed through the optimization process. The training process of the optimized network is divided into two phases: the connectivity optimization process which forms an optimized RNN connectivity structure and then the weights of the optimized network are further refined using EKF and the same training data.

Conclusions

This chapter introduces the development of the proposed RNN.

The network is modified from FFCNN by importing internal recurrency into its hidden neuron section. The neurons receive not only the inputs at current time step from

neurons before it but also the inputs at the previous time step from neurons after it, Hence, RNN is fundamentally different from FFNN that it not only operates on the input space but also on its internal state space, which makes it more suitable for non-linear dynamical system modeling.

EKF algorithm is applied in training the network. To achieve that, the training process is first written in terms of process and measurement equations. The Jacobian matrix in the measurement equation is formed by taking the orderly derivatives of network's outputs with respect to its weights. The parameters of EKF are then initialized, and the EKF equations are applied to the system to estimate the weights update during training. The most time consuming procedure involved is the calculation of orderly derivatives.

The network optimization is embedded into the training process to optimize the network structure for any specific applications. A destructive method is applied. A network with redundant structure is selected first. The insignificant weights are removed gradually during the process. To apply the method, first a connectivity coefficient is introduced to each connection through a connectivity function, and then the coefficients are trained with weights using EKF, finally the connections having negative connectivity coefficients are considered as unimportant and are finally removed.

Through these development techniques, an optimized RNN can be formed to model non-linear dynamical systems.

CHPATER FOUR

PERFORMANCE STUDIES OF THE RECURRENT NEURAL NETWORK

Abstract

Training convergence and estimation robustness are important in successful implementation of RNN in modeling non-linear dynamical systems.

RNN has emerged as a promising tool in modeling of non-linear dynamical systems whereas the training convergence is still of concern. This study aims to develop an effective EKF-based RNN training approach with a controllable training convergence. The training convergence problem during extended Kalman filter-based RNN training has been proposed and studied by adapting two artificial training noise parameters: the covariance of measurement noise (R) and the covariance of process noise (Q) of Kalman filter. The R and Q adaption laws have been developed using the Lyapunov method and the maximum likelihood method, respectively.

Robustness study of recurrent neural network is critical to their successful implementations. The goal of robustness study is to reduce the sensitivity of modeling capacity to uncertainties in parameters, or to make the network fault tolerance. In this study, an uncertainty propagation analysis is conducted to quantify the robustness of a recurrent neural network output due to perturbations in its trained weights. Perturbations are added into the weights and the unscented transform is used to quantify the corresponding uncertainties in the network's output. A robustness measure has been proposed and compared with other two measures developed by sensitivity analysis and performance loss analysis.

Nomenclature

R Adaption Law Derivation	
Symbol	Definition
$H(k)$	Jacobian matrix at training step k
$K(k)$	Kalman gain at training step k
$P(k)$	Covariance matrix of weight estimation at training step k
$Q(k)$	Covariance matrix of process noise at training step k
$R(k)$	Covariance matrix of measurement noise at training step k
$r(k)$	Diagonal element of $R(k)$, $R(k)=r(k)\cdot I$
$\vec{w}(k)$	Estimation of weight vector $\vec{w}$ at training step k
$\vec{w}^*$	Optimal value of $\vec{w}$
$\vec{w}_e(k)$	Error of estimated weight vector $\vec{w}(k)$
$\vec{y}^*(k)$	Desired output at training step k
$\vec{y}(k)$	Output of neural network at training step k
$\vec{\xi}(k)$	Residual of first order approximation
$\vec{e}(k)$	Estimation error of RNN output
$B(k)$	Covariance matrix of network output
$V(k)$	Lyapuov function

The Robustness Measure Derivation	
Symbol	Definition
$\vec{w}^*$	Trained weight vector
$\vec{w}$	Perturbed weight vector, $\vec{w} \propto N(\vec{w}^*, \Sigma)$
Σ	Covariance matrix of $\vec{w}$
L	Perturbation level
$\vec{\chi}_i$	Sigma vector
$\vec{x}(j)$	The *jth* sample input vector
$\vec{y}(j)$	Output of $\vec{x}(j)$ from perturbed networks
$\bar{y}(j)$	mean of $\vec{y}(j)$
$\Sigma_{\vec{y}(j)}$	Covariance of $\vec{y}(j)$
$\sigma_{y_k(j)}$	Standard deviation of its kth element of $\vec{y}(j)$, $\sigma_{y_k(j)} = \sqrt{\Sigma_{\vec{y}(j)_{(k,k)}}}$

Convergence Study of RNN Training Algorithm

The stability of a dynamical system is usually evaluated using the Lyapunov theorems, which give a precise characterization of valid energy functions in the vicinity of equilibrium points [Sast99]. Lyapunov stability is concerned mainly with stability of equilibrium points, and a Lyapunov stable system is a system for which the states remain bounded for all time [Khal02]. In RNN training, the equilibrium points can be viewed as the optimal weight solutions that minimize the mean square error of the outputs of the neural network. The weight training process aims to find the optimal weights as the system's equilibrium points, and the Lyapunov indirect method is used here to study the convergence of training by adapting R.

While the training convergence is first guaranteed by adapting R, the process noise parameter Q is further to be estimated to accelerate the training process, which needs the simultaneous estimation of the noise statistics and the update of the Kalman filter gain. The noise covariance matrixes can be estimated through the Bayesian estimation [Alsp74], the correlation method [Mehr72], the covariance matching method [Mehr72], and the maximum likelihood method [Mayb82]. The maximum likelihood method is favored in this study because it is more efficient, consistent, and suitable for online applications. It should be pointed out that this method may generate biased estimates for small sample sizes. However, because the maximum likelihood estimates tend to have the true value of the estimated variable close to the center of their distributions, the bias is often negligible [Mayb82].

In the following sections, first the Lyapunove method and the maximum likelihood method are briefly introduced and then the convergence study is presented in detail.

Lyapunov Method

In this study, the direct Lyapunov method is applied to form an adaption law for the training parameter $R(k)$ in Equation (2.13) to guarantee the training process convergence. In this section, the direct Lyapunov method, also called the second Lyapunov method is reviewed.

This method is used to determine the stability of an autonomous system without explicitly integrating its differential equation. The idea behind Lyapunov's direct method is to establish properties of the equilibrium point by studying how a carefully selected scalar function of the state (Lyapunov function) evolve as the system state evolves [Khal02].

Consider a non-linear dynamical system is represented by a differential equation:

$$\dot{x}(t) = f(x(t)) \tag{4.1}$$

Suppose origin is the equilibrium state:

$$f(0) = 0 \tag{4.2}$$

To prove the system is Lyapunov stable at the origin, a candidate Lyapunov function $V(x)$ needs to be proposed:

$$\begin{cases} V(0) = 0, & x = 0 \\ V(x) > 0, & x \neq 0 \end{cases} \tag{4.3}$$

Such a $V(x)$ can be thought of as an energy function. Let $\dot{V}(x)$ denote the time derivative of $V(x)$ along any trajectory of the system, i.e. its rate of change as x(t) varies according to Equation (4.1). If this derivative is negative throughout the region (except at the origin):

$$\dot{V}(x) < 0 \qquad x \neq 0 \tag{4.4}$$

Equation (4.4) implies that the energy is strictly decreasing over time. In this case, because the energy is lower bounded by 0, the energy must go to 0 when t goes to infinity, which implies that all trajectories converge to the equilibrium state, zero state. Or in another words, the system is asymptotically stable at the origin.

For a discrete system, the key of the method is to find an appropriate Lyapunov function $V(k)$ for the concerned dynamic system so that $\Delta V(k) < 0$. In this study, a discrete Lyapunov function is chosen as

$$V(k) = \vec{w}_e(k)^T P(k)^{-1} \vec{w}_e(k) \tag{4.5}$$

where $\vec{w}_e(k) = \vec{w}(k) - \vec{w}^*$ is the error of estimated weight vector, $\vec{w}^*$ is the optimal weight and is a constant vector, $\vec{w}(k)$ is the estimate of $\vec{w}^*$ using EKF at time step *k*, and $P(k)$ is the approximate covariance matrix in Equations (3.48 and 3.49).

Maximum Likelihood Method

Once the training process is guaranteed to be convergent by using the Lypunov method mentioned in the previous section, another further question comes out, how to

make the convergence fast. To achieve that, the maximum likelihood method is adopted in this study to estimate the training parameter $Q(k)$ in Equation (3.49).

In lot of cases, a parameter estimation process is requisite to a modeling process. A model with its parameters is first developed to describe the observed data or measurements. The remaining task is to find the best estimation of the parameters that make the model best fit the data. Maximum likelihood estimation (MLE) is a popular statistical method used for parameter estimation [Kay93].

Another major parameter estimation method is the least squares estimation (LSE) which aims to determine the parameters to make the model most accurately fit the sample data. In general, results of MLE are different from those of LSE. In most cases MLE is preferable to LSE unless the likelihood function can't be easily formed. Generally MLE have desirable properties which makes MLE a desirable candidate to estimate the process noise $Q(k)$ in EKF Equation (3.49):

1) It is a sufficient estimator which contains complete information about the parameter of interest which is the covariance of process noise in RNN training in this study;
2) It is an unbiased estimator as the sample size increases;
3) It is a minimum variance estimator as the sample size increases; and
4) The likelihood function can be used to test hypotheses and construct confidence intervals for the model output which is the network output in this study.

To implement MLE, the first step is to develop the maximum likelihood function. A popular way to develop the function is described as follows. Suppose the variable to be modeled is a random variable and the functional form of the variable's probability density function (PDF) is known. A random sample $X=(X_1, X_2,..., X_n)$ for the variable is observed and the observation $(x_1, x_2,..., x_n)$ is used to estimate the parameters. A group of parametric models are established to describe the observation data, each model depends on a unknown parameter θ.

The PDF which accounts for the probability of random sample X given the parameter θ can be written as $f_\theta(X_1,...X_n \mid \theta)$. If individual samples X_i are independent of one another, the PDF can be written as a multiplication of each PDFs for all the observations:

$$f_\theta(X_1,...X_n \mid \theta) = f_\theta(X_1 \mid \theta) f_\theta(X_2 \mid \theta)...f_\theta(X_n \mid \theta) = \prod_{i=1}^{n} f_\theta(X_i \mid \theta) \tag{4. 6}$$

The likelihood function is defined by reversing the roles of sample X and parameter of θ in PDF, which represents the likelihood of parameter θ given the sample X:

$$L(\theta \mid X_1,...X_n) = f_\theta(X_1,...X_n \mid \theta) \tag{4.7}$$

Similar to (4.6), if the random samples X_i are independent with one another the likelihood function (4.7) can be simplified as:

$$L(\theta \mid X_1,...X_n) = \prod_{i=1}^{n} f_\theta(X_i \mid \theta) \tag{4.8}$$

For this case, the likelihood function is often written in logarithm format as follows:

$$L(\theta \mid X_1,...X_n) = \sum_{i=1}^{n} \log f_\theta (X_i \mid \theta) \tag{4.9}$$

In other words, the likelihood function represents how likely the parameter can be if the observed data x of X is given. The method of maximum likelihood estimation of θ that maximizes the likelihood function:

$$\hat{\theta} = \arg\max_{\theta} L(\theta \mid X_1,...X_n) \tag{4.10}$$

In summary, MLE begins with writing the likelihood function of the unknown parameter θ based on sample data. The parameter value that maximizes the likelihood function then provides MLE results – the maximum likelihood estimator of the parameter.

The Development of R Adaption Law

The covariance of measurement noise R describes the statistics of network modeling error, and this information is generally not available for a RNN training process. As seen from Equation (3.25), a small R value might lead to a large Kalman gain, which may make training divergent. For this training divergence problem, a dead-zone Kalman filter was developed to train a state space recurrent neural network to avoid divergence in training [Rubi07]. This study further extends the modified EKF work [Rubi07] in the following aspects:

1) The development here considers training process of general neural networks; the training model (3.18) is applicable to a general neural network.
2) The development here doesn't require the knowledge of upper bound of ξ_k, the residual of first order approximation, which is unavailable in most cases; and
3) The developed algorithm is more efficient because it doesn't have dead-zone regions.

To better illustrate the development process, weights update equations (3.47-3.49) are repeated as in the following equations (4.11-4.13):

$$\begin{cases} \vec{w}(k+1) = \vec{w}(k) - K(k)\left(\vec{y}(k) - \vec{y}^*(k)\right) \\ \vec{y}(k) = h\left(\vec{w}(k)\right) \end{cases} \tag{4.11}$$

$$\begin{aligned} K(k) &= P(k)H(k)[R(k) + H(k)^T P(k)H(k)]^{-1} \\ &\equiv P(k)H(k)B(k)^{-1} \end{aligned} \tag{4.12}$$

$$P(k+1) = Q(k) + \left[I - K(k)H(k)^T\right]P(k) \tag{4.13}$$

where $\vec{w}(k)$ is the estimate of weight vector $\vec{w}^*$ at time step k, $\vec{w}(k-1)$ is the estimation of weight vector at one time step before, K is the Kalman gain, $\vec{y}(k)$ is the output of RNN, $\vec{y}^*(k)$ is the target value for $\vec{y}(k)$, P is the approximate covariance matrices, H is the orderly derivatives matrix, and the covariance matrix of output ($R(k) + H(k)^{\mathrm{T}} P(k)H(k)$ or $Cov\left(\vec{y}(k)\right)$) is denoted as $B(k)$.

Expand output of RNN $\vec{y}(k)$ at the optimal weight vector $\vec{w}^*$:

$$\begin{aligned}\vec{y}(k) &= h\left(\vec{w}(k)\right) \\ &= h\left(\vec{w}^*\right) + \frac{\partial h}{\partial w}\left(\vec{w}(k) - \vec{w}^*\right) + \vec{\xi}(k) \\ &= \vec{y}^*(k) + H(k)^T \vec{w}_e(k) + \vec{\xi}(k)\end{aligned} \tag{4.14}$$

where $\vec{w}_e(k) = \vec{w}(k) - \vec{w}^*$ is the error of estimated weight vector, $\vec{w}^*$ is the optimal weight and is a constant vector, and $\vec{\xi}$ is the residual of first order approximation.

The estimation error of RNN output is represented as:

$$\vec{e}(k) = \vec{y}(k) - \vec{y}^*(k) = H(k)^T \vec{w}_e(k) + \vec{\xi}(k) \tag{4.15}$$

A Lyapunov function is selected as in [Rubi07]:

$$V(k) = \vec{w}_e(k)^T P(k)^{-1} \vec{w}_e(k) \tag{4.16}$$

$$\Delta V(k) = \vec{w}_e(k+1)^T P(k+1)^{-1} \vec{w}_e(k+1) - \vec{w}_e(k)^T P(k)^{-1} \vec{w}_e(k) \tag{4.17}$$

Plug Equations (4.12) and (4.15) into (4.11), the following equation is derived as:

$$\begin{aligned}\vec{w}_e(k+1) &= \vec{w}_e(k) - P(k)H(k)B(k)^{-1} e(k) \\ &= \vec{w}_e(k) - P(k)H(k)B(k)^{-1} H(k)^T \vec{w}_e(k) - P(k)H(k)B(k)^{-1} \vec{\xi}(k) \\ &= \left(I - P(k)H(k)B(k)^{-1} H(k)^T\right)\vec{w}_e(k) - P(k)H(k)B(k)^{-1} \vec{\xi}(k)\end{aligned} \tag{4.18}$$

The following equation is equivalent to Equation (4.13):

$$\left(P(k+1) - Q(k)\right) = \left(I - P(k)H(k)B(k)^{-1} H(k)^T\right)P(k) \tag{4.19}$$

Apply the matrix calculation formula on $\left(P(k)^{-1} + H(k)R(k)^{-1} H(k)^T\right)^{-1}$, according to the matrix inversion lemma:

$$\left(A + BCD\right)^{-1} = A^{-1} - A^{-1} B\left(DA^{-1} B + C^{-1}\right)^{-1} DA^{-1} \tag{4.20}$$

which leads to:

$$\left(P(k)^{-1} + H(k)R(k)^{-1} H(k)^T\right)^{-1} = \left(I - P(k)H(k)B(k)^{-1} H(k)^T\right)P(k) \tag{4.21}$$

Compare (4.19) and (4.21), it is found:

$$\left(P(k+1)-Q(k)\right)^{-1}=\left(P(k)^{-1}+H(k)R(k)^{-1}H(k)^{T}\right)>0 \tag{4.22}$$

From Equation (4.18), (4.21) and (4.22), it can be derived that:

$$\begin{aligned}&\left(P(k+1)-Q(k)\right)^{-1}\vec{w}_e(k+1)\\&=P(k)^{-1}\vec{w}_e(k)-\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\end{aligned} \tag{4.23}$$

$$\vec{w}_e(k+1)^{T}P(k+1)^{-1}\vec{w}_e(k+1)<\vec{w}_e(k+1)^{T}\left(P(k+1)-Q(k)\right)^{-1}\vec{w}_e(k+1) \tag{4.24}$$

Equations (4.23) and (4.24) give:

$$\begin{aligned}\Delta V(k)&=\vec{w}_e(k+1)^{T}P(k+1)^{-1}\vec{w}_e(k+1)-\vec{w}_e(k)^{T}P(k)^{-1}\vec{w}_e(k)\\&<\left[\vec{w}_e(k+1)-\vec{w}_e(k)\right]^{T}P(k)^{-1}\vec{w}_e(k)\\&\quad-\vec{w}_e(k+1)^{T}\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\end{aligned} \tag{4.25}$$

Plug Equation (4.18) into (4.25), also knowing $B(k)$ and $P(k)$ are symmetric:

$$\begin{aligned}\Delta V(k)<&-\vec{w}_e(k)^{T}H(k)B(k)^{-1}H(k)^{T}\vec{w}_e(k)-\vec{\xi}(k)^{T}B(k)^{-1}H(k)^{T}\vec{w}_e(k)\\&-\vec{w}_e(k+1)^{T}\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\end{aligned} \tag{4.26}$$

Consider the third part on the right side of Equation (4.26) gives (knowing $\Delta V(k)$ is a scalar, plug Equation (4.23) in):

$$\begin{aligned}&\vec{w}_e(k+1)^{T}\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\\&=\vec{\xi}(k)^{T}B(k)^{-1}H(k)^{T}P(k)\left(P(k+1)-Q(k)\right)^{-1}\vec{w}_e(k+1)\\&=\vec{\xi}(k)^{T}B(k)^{-1}H(k)^{T}P(k)[P(k)^{-1}\vec{w}_e(k)\\&\qquad\qquad-\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)]\\&=\vec{\xi}(k)^{T}B(k)^{-1}H(k)^{T}\vec{w}_e(k)\\&\quad-\vec{\xi}(k)^{T}B(k)^{-1}H(k)^{T}P(k)\left(P(k+1)-Q(k)\right)^{-1}P(k)H(k)B(k)^{-1}\vec{\xi}(k)\end{aligned} \tag{4.27}$$

Plug in Equation (4.27) into (4.26) and consider the relationships:

a) $R(k)+H(k)^{T}P(k)H(k)=B(k)$,

b) $R(k)^{-1} > B(k)^{-1}$, and

c) $H(k)^T P(k) H(k) B(k)^{-1} < I$

Equation (4.26) becomes:

$$\begin{aligned}
\Delta V(k) < & -\vec{w}_e(k)^T H(k) B(k)^{-1} H(k)^T \vec{w}_e(k) - \vec{\xi}(k)^T B(k)^{-1} H(k)^T \vec{w}_e(k) \\
& - \vec{\xi}(k)^T B(k)^{-1} H(k)^T \vec{w}_e(k) \\
& + \vec{\xi}(k)^T B(k)^{-1} H(k)^T P(k) \left(P(k+1) - Q(k)\right)^{-1} P(k) H(k) B(k)^{-1} \vec{\xi}(k) \\
= & -[\vec{w}_e(k)^T H(k) B(k)^{-1} H(k)^T \vec{w}_e(k) \\
& + 2\vec{\xi}(k)^T B(k)^{-1} H(k)^T \vec{w}_e(k) + \vec{\xi}(k)^T B(k)^{-1} \vec{\xi}(k)] \\
& + \vec{\xi}(k)^T [B(k)^{-1} + B(k)^{-1} H(k)^T P(k) \\
& \times \left(P(k)^{-1} + H(k) R(k)^{-1} H(k)^T\right) P(k) H(k) B(k)^{-1}] \vec{\xi}(k) \\
= & -\left[H(k)^T \vec{w}_e(k) + \vec{\xi}(k)\right]^T B(k)^{-1} \left[H(k)^T \vec{w}_e(k) + \vec{\xi}(k)\right] \\
& + \vec{\xi}(k)^T [B(k)^{-1} \left(I + H(k)^T P(k) H(k) B(k)^{-1}\right) \\
& + B(k)^{-1} H(k)^T P(k) H(k) R(k)^{-1} H(k)^T P(k) H(k) B(k)^{-1}] \vec{\xi}(k) \\
\le & -e(k)^T B(k)^{-1} e(k) + \vec{\xi}(k)^T \left[B(k)^{-1} (I + I) + R(k)^{-1}\right] \vec{\xi}(k) \\
\le & -e(k)^T B(k)^{-1} e(k) + 3\vec{\xi}(k)^T R(k)^{-1} \vec{\xi}(k)
\end{aligned} \tag{4.28}$$

Based on the matrix property, it is known that:

$$\lambda_m \left(B(k)^{-1}\right) \left\|\vec{e}(k)\right\|^2 \le \vec{e}(k)^T B(k)^{-1} \vec{e}(k) \le \lambda_M \left(B(k)^{-1}\right) \left\|\vec{e}(k)\right\|^2 \tag{4.29}$$

$$\lambda_m \left(R(k)^{-1}\right) \left\|\vec{\xi}(k)\right\|^2 \le \vec{\xi}(k)^T R(k)^{-1} \vec{\xi}(k) \le \lambda_M \left(R(k)^{-1}\right) \left\|\vec{\xi}(k)\right\|^2 \tag{4.30}$$

where $\lambda_m \left(B(k)^{-1}\right)$ and $\lambda_M \left(B(k)^{-1}\right)$ are the minimum and maximum eigenvalues of matrix $B(k)^{-1}$ respectively, and $\left\|\vec{e}(k)\right\|$ is the Euclidean norm of $\vec{e}(k)$ as $\sqrt{\vec{e}(k)^T \vec{e}(k)}$.

Plug Equations (4.29) and (4.30) into Equation (4.28), it can be seen:

$$\Delta V(k) < -\lambda_m \left(B(k)^{-1}\right) \left\|\vec{e}(k)\right\|^2 + 3\lambda_M \left(R(k)^{-1}\right) \left\|\vec{\xi}(k)\right\|^2 \tag{4.31}$$

Since there is no prior information about the measurement noise, this study simplifies the measurement noises for each output as uncorrelated but with the same variance $r(k)$ as follows:

$$R(k) = r(k)I \tag{4.32}$$

where $r(k)$ is a positive scalar, I is the $n \times n$ identity matrix, and n is the dimension of output vector. Then

$$\lambda_M\left(R(k)^{-1}\right) = \frac{1}{r(k)} \tag{4.33}$$

The minimum eigenvalue of $B(k)^{-1}$ is the inverse of maximum eigenvalue of $B(k)$:

$$\lambda_m\left(B(k)^{-1}\right) = \frac{1}{\lambda_M\left(B(k)\right)} \tag{4.34}$$

The maximum eigenvalue of $B(k)$ should follow the relationship:

$$\frac{1}{n}\sum_i \lambda_i\left(B(k)\right) \le \lambda_M\left(B(k)\right) \le \sum_i \lambda_i\left(B(k)\right) \tag{4.35}$$

where $\lambda_i\left(B(k)\right)$ is the ith eigenvalue of $B(k)$. The summation of eigenvalues of a matrix equals the trace of the matrix and it is known that the covariance matrix $B(k)$ is positive definite, which leads to:

$$\sum_i \lambda_i\left(B(k)\right) = Trace\left(B(k)\right) = Trace\left(\left[H(k)^{\mathrm{T}} P(k) H(k)\right]\right) + nr(k) \equiv h(k) + nr(k) \tag{4.36}$$

where $Trace\left(\left[H(k)^{\mathrm{T}} P(k) H(k)\right]\right)$ is the trace of $H(k)^{\mathrm{T}} P(k) H(k)$ and is denoted by $h(k)$.

Plug (4.36) into the left side of (4.35):

$$\frac{1}{n}h(k) + r(k) \le \lambda_M\left(B(k)\right) \tag{4.37}$$

Plug (4.37) into (4.34):

$$\lambda_m\left(B(k)^{-1}\right) \le \frac{1}{\frac{1}{n}h(k)+r(k)} = \frac{n}{h(k)+nr(k)} \tag{4.38}$$

Hence, the inequality (4.31) will be satisfied if

$$\Delta V(k) < -\frac{1}{\frac{1}{n}h(k)+r(k)}\left\|\vec{e}(k)\right\|^2 + 3\lambda_M\left(R(k)^{-1}\right)\left\|\vec{\xi}(k)\right\|^2 \tag{4.39}$$

Plug (4.34) into (4.39):

$$\Delta V(k) < -\frac{1}{\frac{1}{n}h(k)+r(k)}\left\|\vec{e}(k)\right\|^2 + \frac{3}{r(k)}\left\|\vec{\xi}(k)\right\|^2 \tag{4.40}$$

The training process is convergent if $\Delta V(k) < 0$, which leads to the following sufficient condition based on (4.40):

$$r(k) > \frac{\frac{3h(k)}{n}\cdot\left\|\vec{\xi}(k)\right\|^2}{\left\|\vec{e}(k)\right\|^2 - 3\left\|\vec{\xi}(k)\right\|^2} \tag{4.41}$$

Suppose $\left\|\vec{\xi}(k)\right\|$ is bounded by $\bar{\xi}(k)$ ($\bar{\xi}(k) \ge \left\|\vec{\xi}(k)\right\|$), consider the following two cases:

Case 1: The output estimation error $\left\|\vec{e}(k)\right\|^2$ is greater than $4\bar{\xi}(k)^2$:

$$\left\|\vec{e}(k)\right\|^2 > 4\bar{\xi}(k)^2 \tag{4.42}$$

then plug (4.42) in (4.41):

$$r(k) > \frac{\frac{3h(k)}{n}\cdot\left\|\vec{\xi}(k)\right\|^2}{4\left\|\vec{\xi}(k)\right\|^2 - 3\left\|\vec{\xi}(k)\right\|^2} = \frac{3h(k)}{n} \tag{4.43}$$

It can be seen that if Inequality (4.43) holds true then (4.41) holds true, and then $\Delta V(k) < 0$ as guided in (4.40), and the training process is convergent. $\bar{\xi}(k)$, the upper bound of the norm of $\vec{\xi}(k)$ in (4.42) should be known as a prior condition. As $\vec{\xi}(k)$ is the residual of liner model of $\vec{e}(k)$ as shown in (4.15), it follows the normal distribution: $\vec{\xi}(k) \sim N([\vec{0}]_{n\times 1}, r(k)I_{n\times n})$ based on the extended Kalman filter algorithm. Each element of $\vec{\xi}(k)$, $\xi_i(k)$, follows the normal distribution as $\xi_i(k) \sim N(0, r(k))$. Then the upper bound of $\xi_i(k)$ can be approximated as $\eta\sqrt{r(k)}$. If η is selected as 4, then at least 99.99% $\xi_i(k)$ values are bounded by $4\sqrt{r(k)}$. As an approximation, $\bar{\xi}(k)^2$ is taken as $(4\sqrt{r(k)})^2 n$, which is $16r(k)n$.

Thus, (4.42) becomes:

$$\left\|\vec{e}(k)\right\|^2 > 4\bar{\xi}(k)^2 = 64r(k)n \tag{4.44}$$

Combine Inequalities (4.43) and (4.44), it is found that

$$\frac{3h(k)}{n} < r(k) < \frac{\left\|\vec{e}(k)\right\|^2}{64n} \tag{4.45}$$

When (4.45) holds, both (4.42) and (4.44) hold which leads to $\Delta V(k) < 0$, and the training should be convergent as guaranteed by the Lyapunov method.

To implement (4.45), the training error information at each step, $\bar{e}(k)$, should be considered first. When $\frac{3h(k)}{n} < \frac{\|\bar{e}(k)\|^2}{64n}$, or $\|\bar{e}(k)\| > \sqrt{192h(k)}$, (41) is satisfied if $r(k)$ is set as the average of $\frac{3h(k)}{n}$ and $\frac{\|\bar{e}(k)\|^2}{64n}$ as follows:

$$r(k) = \frac{1}{2}\left(\frac{3h(k)}{n} + \frac{\|\bar{e}(k)\|^2}{64n}\right) \qquad \text{when } \|\bar{e}(k)\| > \sqrt{192h(k)} \tag{4.46}$$

Under this circumstance $\Delta V(k) < 0$, which means a convergent training process.

Case 2: The output estimation error $\|\bar{e}(k)\|^2$ is less than $4\bar{\xi}(k)^2$:

The training error is bounded and no adaption needs to be implemented at training step *k*.

Under this circumstance, $\Delta V(k) < 0$ can always be satisfied using Equation (4.46), which means a convergent training process. For some engineering applications where there is only one output (n=1), the *R* adaption law can be further simplified as follows:

$$r(k) = \frac{1}{2}\left(3H(k)^T P(k)H(k) + \frac{e(k)^2}{64}\right) \text{ when } |e(k)| > \sqrt{192H(k)^T P(k)H(k)} \tag{4.47}$$

It should be pointed out that the above condition (Equations (4.46) or (4.47)) is the sufficient condition for a convergent training process instead of as a necessary condition.

The Development of Q Adaption Law

In the previous section, a noise parameter R is adapted using Lyapunov method to guarantee the convergence of training process. Furthermore, another noise parameter Q is to be estimated to accelerate the training process. Estimation of Q falls in the region of adaptive filtering technologies, which simultaneously estimate the statistics of the noise and update the Kalman gain during the filtering process.

Generally four approaches are developed to estimate the noise covariance matrix:

1) Bayesian estimation was applied as in [Alsp74], which requires an a priori specification of a parameter density function, and sufficient statistical information to infer such density function is often not available in real applications. Also, usually prohibitive computation cost hurdle its wide application, especially for online estimations,
2) For the correlation method, autocorrelation functions of innovation sequence are constructed and the unknown covariance can be inferred by solving a set of equations [Mehr70]. However, to apply this method, system is assumed to be completely observable and controllable which is not valid for RNN training system model,
3) Covariance matching method makes the residuals consistent with their theoretical covariance and hereby solves the unknown matrices. The covariance usually does not match the actual one and the convergence of the method is therefore often doubtful. [Mehr72], and

4) The maximum likelihood method has been also used to adapt EKF. It can generate efficient, unbiased, and consistent estimate.

In this study, the maximum likelihood method is applied to estimate Q because it has the following advantages:

1) The maximum likelihood method can lead to an efficient estimate (an unbiased estimate with the lowest covariance);
2) It is consistent. The likelihood equation has a solution that converges to the true value of the variables as the number of sample grows to infinity; and
3) It is suitable for online application. On the other hand, the method may generate biased estimate for small sample size. However, because the maximum likelihood estimates tend to have the true value of the estimated variable close to the center to their distributions, the bias is often negligible [Mayb82].

To apply the maximum likelihood method, an appropriate likelihood function should be chosen at first.

The following conditional density function is selected because it exploits all a priori information available and can yield an effective and computationally feasible estimator [Mayb82]. Consider the conditional density function and using Baye's rule:

$$\begin{aligned} f_{\vec{w}(k),Y(k)|\vec{q}} &= f_{\vec{w}(k)|Y(k),\vec{q}} \cdot f_{Y(k)|\vec{q}} \\ &= f_{\vec{w}(k)|Y(k),\vec{q}} \cdot f_{\vec{y}(k),Y(k-1)|\vec{q}} \\ &= f_{\vec{w}(k)|Y(k),\vec{q}} \cdot f_{\vec{y}(k)|Y(k-1),\vec{q}} \cdot f_{Y(k-1)|\vec{q}} \\ &= f_{\vec{w}(k)|Y(k),\vec{q}} \prod_{j=1}^{k} f_{\vec{y}(j)|Y(j-1),\vec{q}} \end{aligned} \tag{4.48}$$

where $\vec{w}(k)$ is the state vector (weight vector) at time step k, $Y(k)$ is the measurement history $\{\vec{y}(1), \vec{y}(2), ..., \vec{y}(k)\}$, $\vec{q}$ is the vector composed of diagonal elements of matrix Q.

To simplify the calculation, instead of the whole measurement history $Y(k)$, only the n_w most resent measurement history are considered. Thus a fixed sample size n_w of measurement history $Y_N(k) = \{\vec{y}(k-n_w+1), \vec{y}(k-n_w+2), ..., \vec{y}(k-1), \vec{y}(k)\}$ is used in Equation (4.44), which leads to the new conditional density function:

$$\begin{aligned}
f_{\vec{w}(k),Y_N(k)|Y(k-n_w),\vec{q}} &= f_{\vec{w}(k)|Y_N(k),Y(k-n_w),\vec{q}} \cdot f_{Y_N(k)|Y(k-n_w),\vec{q}} \\
&= f_{\vec{w}(k)|Y_N(k),Y(k-n_w),\vec{q}} \frac{f_{Y_N(k),Y(k-n_w)|\vec{q}}}{f_{Y(k-n_w)|\vec{q}}} \\
&= f_{\vec{w}(k)|Y(k),\vec{q}} \frac{f_{Y(k)|\vec{q}}}{f_{Y(k-n_w)|\vec{q}}} \\
&= f_{\vec{w}(k)|Y(k),\vec{q}} \frac{f_{\vec{y}(k),Y(k-1)|\vec{q}}}{f_{Y(k-n_w)|\vec{q}}} \\
&= f_{\vec{w}(k)|Y(k),\vec{q}} \frac{f_{\vec{y}(k)|Y(k-1),\vec{q}} \cdot f_{Y(k-1)|\vec{q}}}{f_{Y(k-n_w)|\vec{q}}} \\
&= f_{\vec{w}(k)|Y(k),\vec{q}} \prod_{j=i-n_w+1}^{i} f_{\vec{y}(j)|Y(j-1),\vec{q}}
\end{aligned} \tag{4.49}$$

Each of the densities in Equation (4.49) can be assumed as a Gaussian density function:

$$\begin{aligned}
&f_{\vec{w}(k)|Y(k),\vec{q}}(\vec{\psi}(k) \mid \delta(k), \vec{\rho}) \\
&= \frac{1}{(2\pi)^{m/2}|P(k)|^{1/2}} \exp\left\{-\frac{1}{2}[\vec{\psi}(k) - \vec{w}(k)]^T P(k)^{-1}[\vec{\psi}(k) - \vec{w}(k)]\right\}
\end{aligned} \tag{4.50}$$

$$\begin{aligned}
&f_{\vec{y}(j)|Y(j-1),\vec{q}}(\vec{\gamma}(j) \mid \delta(j-1), \vec{\rho}) \\
&= \frac{1}{(2\pi)^{n/2}|B(j)|^{1/2}} \exp\left\{-\frac{1}{2}[\vec{\gamma}(j) - H(j)\vec{w}(j-1)]^T B(j)^{-1}[\vec{\gamma}(j) - H(j)\vec{w}(j-1)]\right\}
\end{aligned} \tag{4.51}$$

where $\vec{\psi}(k)$, $\delta(k)$, $\vec{\gamma}(j)$ and $\vec{\rho}$ are the realization of random variables $\vec{w}(k)$, $Y(k)$, $\vec{y}(j)$ and $\vec{q}$ respectively, and m and n are the dimensions of the states and output respectively, $P(k)$ is the approximate covariance matrix, $\vec{w}(j)$ is the estimate of $\vec{w}^*$, $H(j)$ is the orderly derivatives matrix, and $B(j) = R(j) + H(j)^{\mathrm{T}} P(j) H(j)$ is the covariance matrix of output.

The likelihood function is represented as $L = \ln\left(f_{\vec{w}(k), Y_N(k) | Y(k-n_w), \vec{q}}\right)$. To take derivative of the likelihood function with respect to $\vec{q}$ and make it zero will give the maximum likelihood equation as following:

$$tr\left\{P(k)^{-1}\frac{\partial P(k)}{\partial q_k}\right\} - 2\sum_{j=i-n_w+1}^{i}\frac{\vec{w}(j-1)}{\partial q_k}H(j)^T B(j)^{-1}\vec{r}(j) + \sum_{j=i-N_w+1}^{i} tr\left\{\left[A(j)^{-1} - A(j)^{-1}\vec{r}(j)\vec{r}(j)^T B(j)^{-1}\right]\right\}\Bigg|_{\vec{q}=\vec{q}^*(i)} = 0 \quad (4.52)$$

where $\vec{r}_j = \vec{y}_j - H(t_j)\vec{w}(j-1)$ is a notation for simplicity.

To enhance online applicability, some less sensitive terms in Equation (4.52) are neglected and thus form an approximated maximum likelihood equation as follows:

$$\sum_{j=k-N+1}^{k}\left[P(j-1) + Q(j-1) - P(j) - \Delta\vec{w}(j)\Delta\vec{w}(j)^T\right] = 0 \quad (4.53)$$

where $\Delta\vec{w}(j) = \vec{w}(j) - \vec{w}(j-1)$ is the difference of estimated weights at *k* and *k-1* time steps, and $Q(k)$ is assumed constant over the period $k - n_w + 1$ from to *k* [Mayb82]. Then the $Q(k)$ matrix can be estimated as follows [Mayb82]:

$$\hat{Q}(k) = \frac{1}{n_w}\left\{\sum_{j=k-n_w+1}^{k}\left\{\Delta\vec{w}(j)\Delta\vec{w}(j)^T\right\} + P(k) - P(k-n_w)\right\} \quad (4.54)$$

The *R* and *Q* adaption-based training is summarized in Figure (4.1). Training parameters such as the window size *N*, the noise parameters $Q(0)$ and $R(0)$, and the error

covariance $P(0)$ should be specified first. The adapted *R(k)* and *Q(k)* at each training step are fed into the EKF training algorithm to update the network weights. The training process iterates until the stop criteria are met, and the trained weights are obtained.

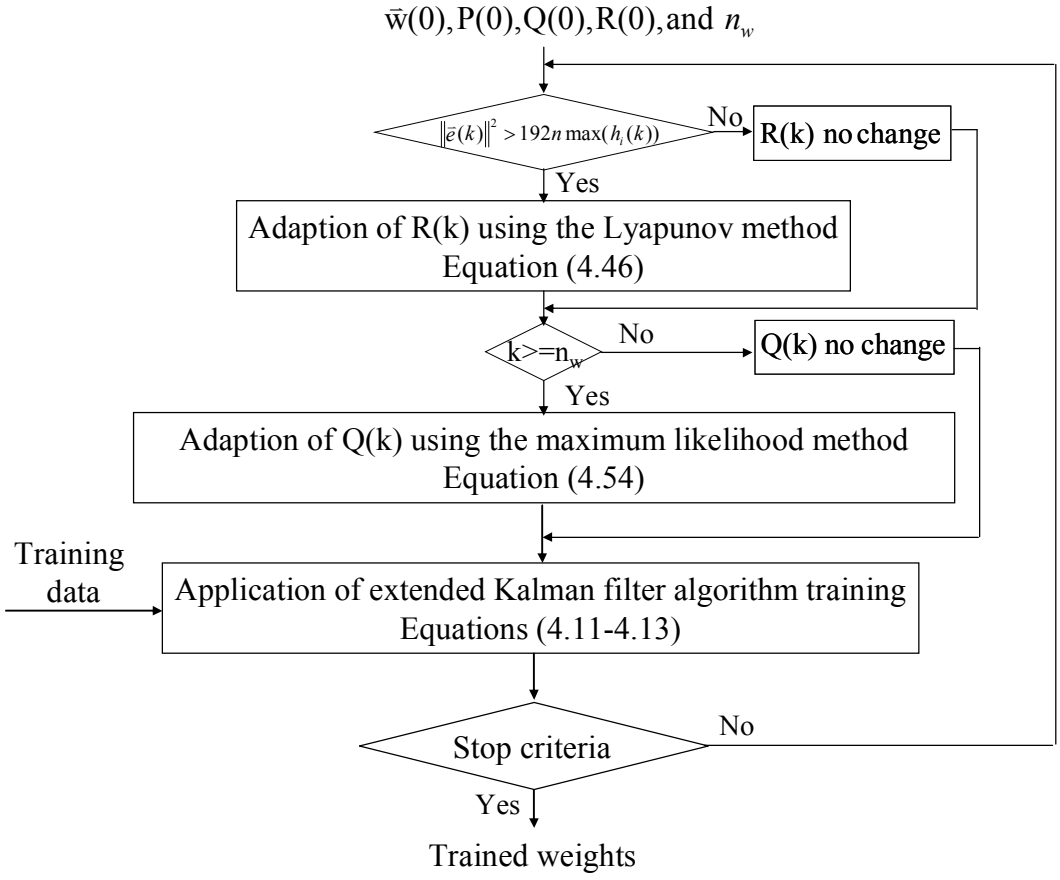


Figure 4.1: Flow chart of the convergence study

Robustness Analysis of RNN

Robustness Analysis

In addition to the training convergence study, network estimation robustness is also studied; a robustness measure is developed to quantify the robustness of a RNN network.

The major part of the proposed method is to take uncertainty propagation analysis. The two most commonly applied numerical approaches for uncertainty propagation analysis are Monte Carlo analysis and Monte Carlo with Latin Hypercube analysis. Monte Carlo analysis is numerical experimentation. Different from using closed form analytical expression to assess the propagation of uncertainty, Monte Carlo method repeatedly generates samples based on the probability distribution of the uncertain parameters to characterize the uncertainty in propagation [Driv00]. Application of Monte Carlo method in the NN robustness study often includes the following steps:

1) Define the domain of perturbation for each uncertain parameters (weight),
2) Draw a set of possible values of each of the uncertain parameters randomly from the domain,
3) Calculate the output of NN that corresponds to these particular values of the parameters
4) Repeat the above two steps and generate corresponding NN outputs, and
5) Aggregate the individual results from step (4) and get the statistics of NN outputs.

However, if the dimension of parameters (weights in this study) is large, the computation cost would make the method prohibitive to apply. To alleviate the problem, in this study a deterministic sampling method, namely unscented transform, is applied.

Instead of Monte-Carlo method, unscented transform is chosen as the computational method in this study due to its high efficiency that it can capture high order information about distributions using only a small number of samples [Juli97].

Using uncertainty propagation analysis, a new robustness measure is proposed here in two steps:

1) Input sample generation using the Latin hypercube sampling (LHS) method [Helt03]; and

2) Robustness quantification using the unscented transform.

This procedure is shown in Figure (4.2) and elaborated as follows.

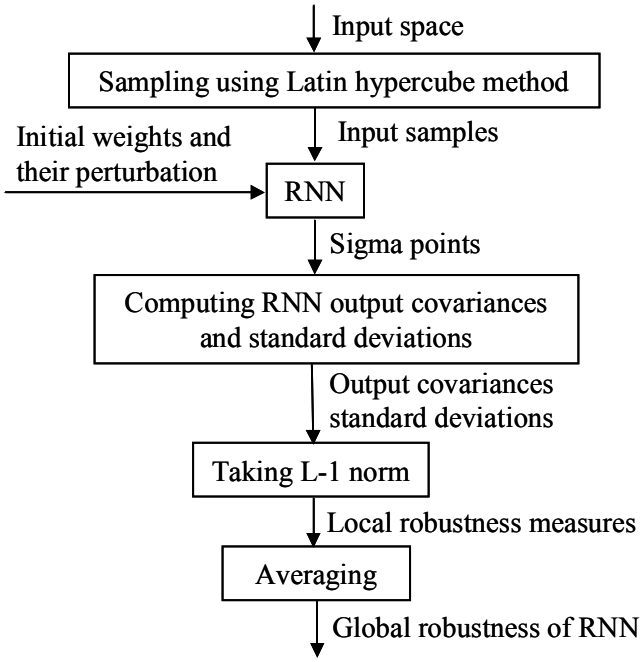


Figure 4.2: Proposed procedures for robustness quantification

The first step is to uniformly generate n samples $(\vec{x}(1), \vec{x}(2), ..., \vec{x}(n))$ from the whole input space. This is done by implementing an LHS method, which is a type of stratified Monte Carlo sampling methods [Loh96] and can be nearly five times more effective than other traditional sampling methods [Swid00]. During the first step n

samples are generated, and they are to be fed into the trained network, which may undergo certain weight perturbations.

The second step is to quantify the network robustness using the unscented transform method based on the generated n samples. The proposed unscented transform-based robustness quantification approach includes two measures: 1) local robustness for network robustness for a given input only; and 2) global robustness to evaluate the network robustness by collectively considering all possible inputs from the whole input space. As RNN is a nonlinear function which maps both inputs and weights to network outputs, outputs can be viewed as a function of weights for given inputs. As so, the local robustness can be interpreted as follows: for a specific input, how much do the outputs vary when the weights deviate from the trained value? It can be seen that the local robustness is input dependent. In this study, the local robustness for any input is defined as the L-1 norm of the output standard deviation vector.

During the second step, the distribution of perturbed weight vector should be determined first. The trained weights are assumed to be contaminated with zero mean finite variance multivariate normal distributed noises [Eick07], and the contaminated weight vector $\vec{w}$ follows the normal distribution as $\vec{w} \propto N(\vec{w}^*, \Sigma)$, where $\vec{w}^*$ is the trained weight vector, which is a column vector transformed from the trained weight matrix by cascading the rows of the matrix into a row vector and further taking transpose, and Σ is the covariance matrix of $\vec{w}$. The standard deviation of w_i, which is the *ith* element (weight) of $\vec{w}$, is determined as follows:

$$\sigma_{w_i} = Lw_i^*,\ i = 1,2,...,n_t \tag{4.55}$$

where σ_{w_i} is the square root of the *ith* diagonal element of Σ, n_t is the dimension of the weight vector, and L is the perturbation level which is a constant specified based on application needs.

The unscented transform is usually used to compute the statistics (mean and covariance) of a random vector which undergoes a nonlinear transformation. In this study the unscented transform is used to compute the statistics of RNN output due to perturbations introduced into the trained weights. With l as the dimension of the trained weight vector, $2l+1$ sigma vectors $\vec{\chi}_i$ $(l = 1,2,...,2l+1)$ are generated around $\vec{w}$ based on the mean ($\vec{w}^*$) and covariance (Σ) of the contaminated weight vectors:

$$\vec{\chi}_0 = \vec{w}^* \tag{4.56}$$

$$\vec{\chi}_i = \vec{w}^* + \left(\sqrt{(l+\lambda)\Sigma}\right)_i \qquad i = 1,2,...,l \tag{4.57}$$

$$\vec{\chi}_i = \vec{w}^* - \left(\sqrt{(l+\lambda)\Sigma}\right)_{i-l} \qquad i = l+1, l+2,...,2l \tag{4.58}$$

where $\lambda = \alpha^2(L+c) - l$ is a scaling parameter, α is a constant which determines the spread of the sigma vectors around $\vec{w}^*$ and it is set as 0.1 in this study [Wan01], c is a secondary scaling parameter and is set as 0 [Wan01], and $\left(\sqrt{(l+\lambda)\Sigma}\right)_i$ is the *ith* column of square root of matrix $(l+\lambda)\Sigma$. Accordingly, $2l+1$ new RNNs are formed based on the $2l+1$ sigma vectors.

To compute the local robustness measure for a *jth* sample input $\vec{x}(j)$, the sample is fed into these $2l+1$ networks respectively, and the corresponding outputs are obtained and called as the outputs of sigma vectors $\vec{\psi}_i(j)$ $(i = 1,2,...,2l+1)$.

$$\vec{\psi}_i(j) = f(\vec{\chi}_i) \qquad i = 0,...,2l \tag{4.59}$$

where $f(\cdot)$ is RNN mapping function.

The mean of network output $\bar{y}(j)$ can be obtained by weighting the outputs of sigma vectors:

$$\bar{y}(j) \approx \sum_{i=0}^{2l} w_i^{(m)} \vec{\psi}_i(j) \tag{4.60}$$

where $w_i^{(m)}$ is a weight used in the unscented transform. The covariance of $\bar{y}(j)$, $\Sigma_{\bar{y}(j)}$ is obtained by:

$$\Sigma_{\bar{y}(j)} = \sum_{i=0}^{2l} w_i^{(c)} (\vec{y}(j) - \vec{\psi}_i(j))(\vec{y}(j) - \vec{\psi}_i(j))^T \tag{4.61}$$

The weights in Equations (4.56) and (4.57) are given by

$$\begin{aligned} w_0^{(m)} &= \lambda/(l+\lambda) \\ w_0^{(c)} &= \lambda/(l+\lambda) + (1-\alpha^2+\beta) \\ w_i^{(m)} &= w_i^{(c)} = 1/\{2(l+\lambda)\}, \qquad i = 1,...,2l \end{aligned} \tag{4.62}$$

where β is a constant used to incorporate any prior knowledge of the distribution of $\vec{w}^*$ and is set as 2 for normal distributions [Wan01].

Suppose $\vec{y}(j)$ is RNN output vector for input sample $\vec{x}(j)$, the standard deviation of its kth element ($y_k(j)$) can be written as:

$$\sigma_{y_k(j)} = \sqrt{\Sigma_{\bar{y}(j)(k,k)}} \tag{4.63}$$

where $\Sigma_{\bar{y}(j)(k,k)}$ denotes the kth diagonal element of $\Sigma_{\bar{y}(j)}$.

A vector composed of the standard deviations of all the elements of output vector $\bar{y}(j)$ can be written as:

$$\vec{\sigma}_{\bar{y}(j)} = \left[\sigma_{y_1(j)}, \sigma_{y_2(j)}, ..., \sigma_{y_{n_y}(j)}\right] \tag{4.64}$$

where n_y is the dimension of $\bar{y}(j)$.

The L-1 vector norm, which computes the summation of absolute value of all elements of a vector, is used as the local robustness measure for the sample input $\bar{x}(j)$:

$$R(j) = \left\| \vec{\sigma}_{\bar{y}(j)} \right\|_1 \tag{4.65}$$

For finite dimensional vector spaces, all vector norms are equivalent [Horn90], and the L-1 norm is selected here due to its robustness to outliers and its easiness for implementation [Kwak08].

Finally, the global robustness measure which accounts for effects on all the input samples from input space is defined as the average of the local robustness measures as $R_1 = \frac{1}{n}\sum_{j=1}^{n} R(j)$, where n is the number of samples.

Conclusions

This chapter carries out two performance studies on the proposed network.

The study of training convergence is conducted by adapting the parameters of process noise and measurement noise. The Lyapunov method has been applied to adapt the covariance of measurement noise to guarantee training convergence. First, a candidate Lyapunov function is selected and its rate of change is computed which is a function of

the concerned parameter. The adaption law is then derived by making the function negative. In addition, the maximum likelihood method is applied to estimate the covariance of process noise to accelerate the training process. A likelihood function is formed by taking the joint probability density function of the weights estimation and outputs of network given the concerned parameter of the process noise. The MLE estimator of the parameter is derived by maximizing the likelihood function.

In addition to the training convergence, the study of estimation robustness of the network due to perturbations in trained weights is also carried out. Gaussian noise has been added into the trained weights which results in uncertainties in the network's output. An uncertainty propagation analysis is then conducted using the unscented transform to quantify the uncertainties in the network's output due to the perturbation. A robustness measure is developed in the study to be compared with the existing sensitivity-based measure and performance-loss based measure.

Both the performance studies are important in successful implementation of RNN in modeling non-linear dynamical systems.

CHAPTER FIVE

MODELING OF A NON-LINEAR DYNAMICAL BENCHMARK SYSTEM

Abstract

In this chapter, the techniques and algorithms developed in Chapter three and Chapter four are verified with a non-linear dynamical benchmark system. A RNN and an optimized RNN are developed to model the system. Modeling capability, training convergence, and robustness of the networks are investigated. Results show both networks are capable of modeling the system. In addition, RNN are better than two FFNN (a MLP and a FFCNN) in terms of training accuracy and speed. Furthermore, from the study of training convergence, it is found that the proposed R adaption law can guarantee the training convergence of the network and the proposed Q adaption law can accelerate the training convergence. Finally, the proposed robustness quantification method is also applied to the networks and is compared with another two methods. Results show that the proposed robustness quantification approach is more efficient, generic, and flexible to quantify the robustness of a recurrent neural network. All together, the results show that the developed optimized RNN has advantages over FFNN in modeling the non-linear dynamical benchmark system and the convergence study and robustness study can further improve the network's performance.

The Benchmark System

To validate the proposed adaption laws, a non-linear dynamical benchmark system [Nare92], which represents a single-input single-output (SISO) non-linear plant, has been modeled using RNN. Such a benchmark system (Equation (5.1)), as shown in Fig. (4), has been selected due to its generality as well as analytical tractability:

It can be seen from Figure (5.1) that the output sequence is a piecewise function which is composed of four regions: [1,250], [251-500], [501-750], and [751-1000].

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{5.1}$$

where $\lfloor u(k), y_p(k) \rfloor$ represents the input-output pair of the SISO plant at time step k,

$f(x_1, x_2, x_3, x_4, x_5) = \dfrac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_2^{\,2} + x_3^{\,2}}$, and

$$u(k) = \begin{cases} \sin(\pi k / 25), & 0 \le k < 250 \\ 1.0, & 250 \le k < 500 \\ -1.0, & 500 \le k < 750 \\ 0.3\sin(\pi k / 25) + 0.1\sin(\pi k / 32) + 0.6\sin(\pi k / 10), & 750 \le k < 1000 \end{cases}$$

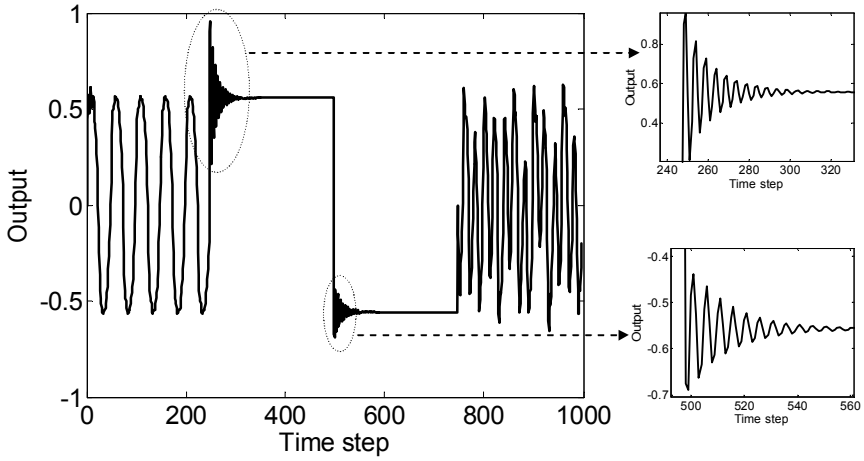


Figure 5.1: The output of the non-linear dynamical benchmark system

Recurrent Neural Network Implementation

A RNN and an optimized RNN are formed to model the system and their modeling result is compared with the measurements as well as the results of an FFCNN and an MLP approaches. The following tasks are conducted to train a RNN to model the benchmark system:

1) Training patterns preparation

In this study, a neural network is implemented to model the benchmark system, or in other words, to predict the output at next time step $y_p(k+1)$ based on six inputs such as current and previous outputs - $y_p(k), y_p(k-1), y_p(k-2)$, current and previous inputs - $u(k), u(k-1)$ and a constant bias 1.

1000 training patterns are formed using Equation (5.1) with the zero initial condition. Each training pattern contains one set of inputs and corresponding output. Usually training data need to be normalized to alleviate the risk of saturation. However, in this case the training data are already within the region of [0, 1], hence the normalization process is unnecessary here.

2) Training parameters configuration.

To train the RNN, some training parameters in the EKF algorithm such as P, Q, and R need to be initialized first. The training parameters configuration is referred from a previous study [Pusk94] without considering the training divergence issue. Without any specific note, in this study the error covariance matrix P, the covariance matrix of process noise Q, and the measurement noise covariance matrix R are all diagonal matrices. Each of the diagonal elements of P is initialized as 100. Each diagonal element

of the covariance matrix of process noise Q is initialized as 0.01 and this value descends linearly within 100,000 training patterns until Q reaches a minimum limit of 0.000001. Similarly, each diagonal element of the measurement noise covariance matrix R is initialized as 100 and it also descends linearly until it reaches a minimum value of 2. Both the settings of R and Q help the training error converge to a global minimum.

3) Training Process Configuration.

First, each weight of the network is randomly initialized in the region of [-1, 1]. Training parameters are initialized as mentioned before. Training patterns are then fed into the EKF training algorithm (Equations (3.47-3.49)) to train the network weights. Single pattern training is used here so that the weights are updated after each training pattern is presented. The training process stops when its stop criteria are satisfied. The stop criteria is used to guarantee the final training error is small so that the network is capable of modeling and at the same time not too small which leads to the over-fitting problem. In this study, the stop criteria are determined by trial-and-error: i) the number of training epochs (a complete pass through all of the training patterns) used should be less than 100 and the training process stops after 100 steps if no other stop criteria are met; or ii) if the training error is less than a predetermined case-dependent value (here is 3%) and the difference between the current error and the error of 20 epochs before is less than another predetermined case-dependent value (here is 0.03%).

4) Network structure determination

RNN network structure is first determined by setting the numbers of input neurons, hidden neurons and output neurons. Six network inputs are selected as follows:

three outputs at the previous steps $y_p(k)$, $y_p(k-1)$, and $y_p(k-2)$, two control actions $u(k)$ and $u(k-1)$, and a constant bias 1; and the network output is $y_p(k+1)$, which is the output of the benchmark system at the next time step (*k+1*). The number of hidden neurons is determined by a trial and error method, and the final training error results are listed in Table (5.1). According to a rule of thumb [Scha97], 13 hidden neurons are chosen first and the training error is found to be 3.2%. Afterwards, networks with fewer hidden neurons are selected and the corresponding training errors are investigated. It is found that RNN with more than 8 hidden neurons is able to adequately model the tool wear progression (the final training error is less than 5%). For example, the training error of the network (6-9-1) with 9 hidden neurons (3.7%) is quite close to that of the network (6-13-1) with 13 hidden neurons (3.2%). However, for the network with 8 hidden neurons, the training error becomes relatively large (5.7%). The training error results for selected networks are shown in Figure (5.2). A simple network structure is always preferred to reduce the risk of over-fitting, so the network with a 6-9-1 structure as shown in Figure (5.3) is selected in this study.

Table 5.1: Training error with different network structure

Network structure	Training error (%)
6-13-1	3.2
6-11-1	3.3
6-9-1	3.7
6-8-1	5.7

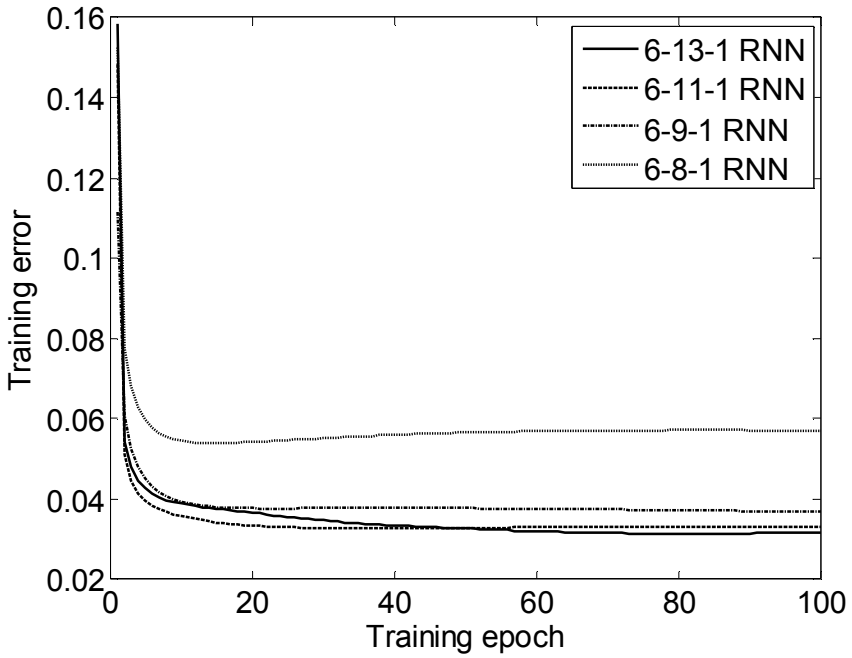


Figure 5.2: Training errors of RNN with typical structure configurations

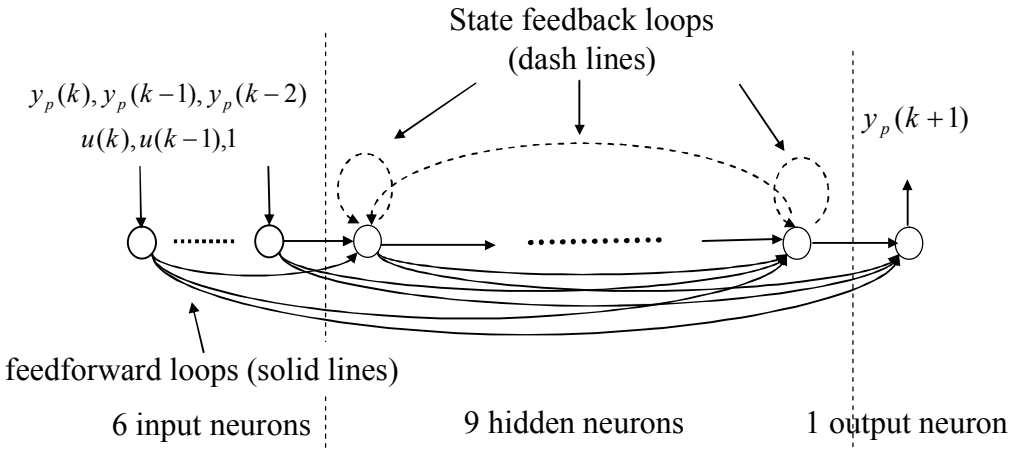


Figure 5.3: Modeling the bench mark system by a 6-9-1 RNN

Modeling Performance of the Recurrent Network

An MLP, an FFCNN and a RNN are implemented to model the system. The networks are trained using the training data which are assumed to be able to represent the overall characteristics of the system being studied. Therefore, through the training process, networks capable of modeling the training data are expected to represent the dynamics of the benchmark system.

During the training process, the appropriate network architecture is determined first as stated in the previous section. Using a similar trial and error approach, the MLP has been found to be 6-11-1 and the other networks (FFCNN and RNN) are 6-9-1. The same training data are used to train the networks. 400 training epochs are used since MLP converges much slower. The modeling results for MLP, FFCNN, and RNN are shown in Figures (5.4-5.6) respectively. Two discontinuous regions (250-300 and 500-550) are magnified to see more details of network's modeling performance no these complex part.

Table 5.2: Training error with different types of network

Network	Training error (%)
MLP	6.8
FFCNN	3.8
RNN	1.1

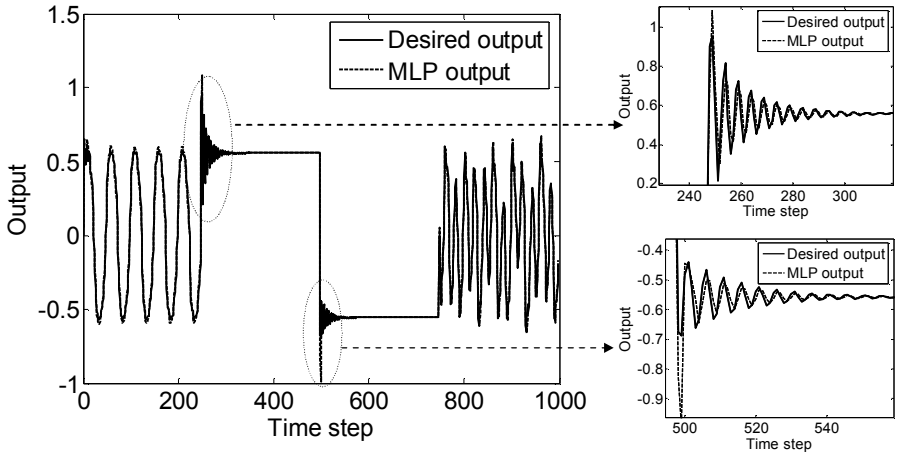


Figure 5.4: Training results of MLP

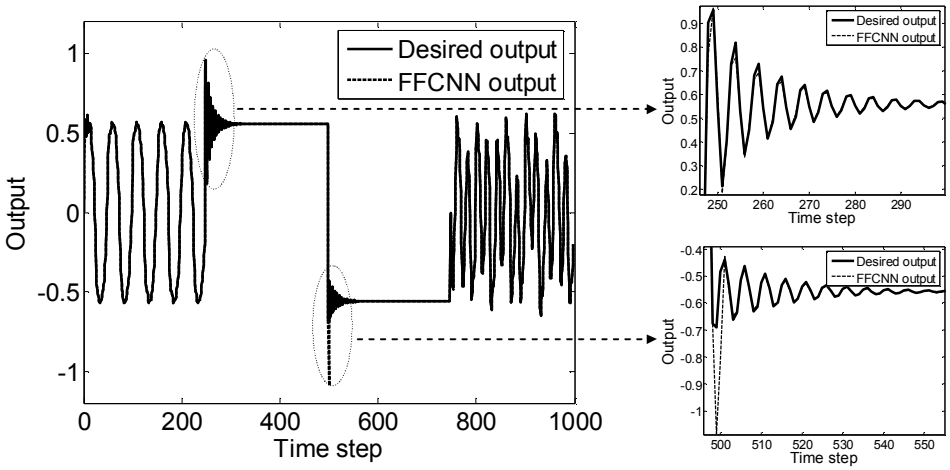


Figure 5.5: Training results of FFCNN

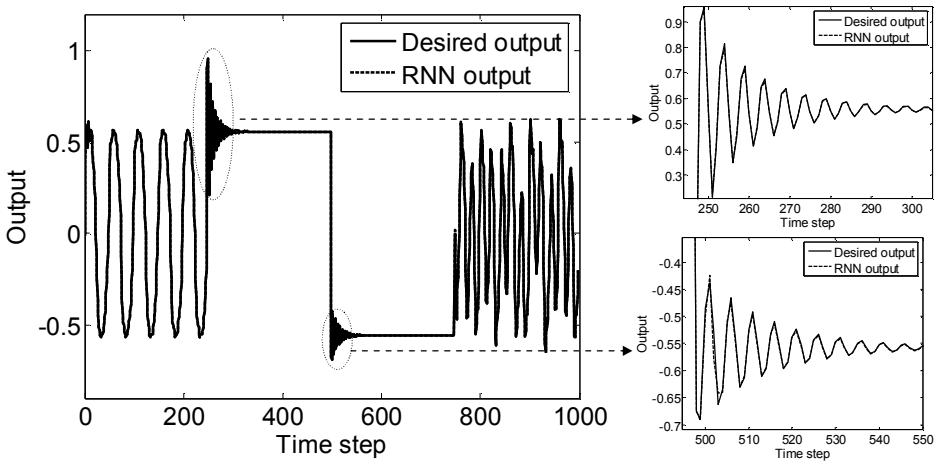


Figure 5.6: Training results of RNN

Moreover, to better compare these networks' modeling accuracy, the modeling errors are depicted in Figure (5.7). Training error and modeling error are two related but different concepts. The former one is defined by Equation (3.51) which accounts for the overall modeling error for all the training patterns of an epoch while the latter one accounts for the difference between target value (measurement) and the corresponding network output for a pattern. The sum of square errors in the four regions ([1-250] [251-500] [501-750] [751-1000]), and the overall region ([1-1000]) are listed in Table (5.3)

Table 5.3: Modeling errors of the networks

Network	Sum of square of modeling error				
	1-250	251-500	501-750	751-1000	1-1000
MLP	0.3381	0.3173	0.0722	0.3654	1.0929
FFCNN	0.0435	0.2792	0.0093	0.0065	0.3385
RNN	0.0053	0.0021	0.0055	0.0024	0.0154

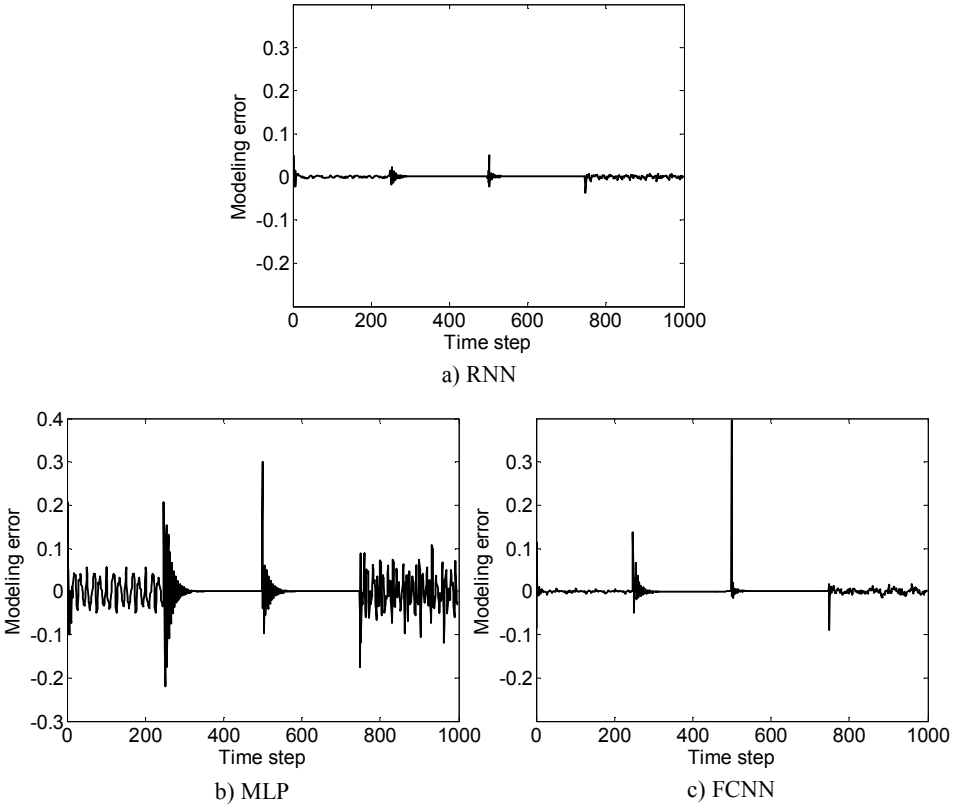


Figure 5.7: Modeling errors of the networks

Some observations can be drawn from the results:

1) The largest modeling errors are distributed around the discontinuous regions around time steps 250, 500, and 750 because theses regions contain abrupt changes and are most difficult to be modeled.
2) For MLP, the error distribution is relatively even and there are large steady errors (±0.1) in the first and last regions.
3) For FFCNN, the modeling error at time step 500 is the largest among all the results, although its average modeling error is smaller than the MLP.

4) The distribution of modeling errors of RNN are similar to that of FFCNN within the four regions, while in the discontinuous regions the modeling errors of RNN are much smaller which indicates that RNN works better in modeling non-linear dynamical systems.

Comparing all the modeling results, RNN can most accurately model the benchmark system.

Training Convergence Study of Recurrent Neural Network

Apart from the modeling accuracy, some other aspects of RNN are also studied. Training convergence and estimation robustness are studied and results are shown in the following sections. Training convergence studies are conducted for two objectives:

1) To make the training process convergent, and
2) To make the training process converges faster.

In this study, the first objective is achieved by adapting the covariance matrix of process noise R of the EKF training algorithm, while the second one is achieved by adapting the covariance matrix of measurement noise Q of the EKF.

R Adaption Law for Convergence Guarantee

The R adaption law is verified with the case study first. As mentioned in Chapter four, training divergence of RNN may occur due to improper choice of training noise parameters. Figure (5.8) shows the result of a modeling scenario as r is first initialized as 5 and it then decreases linearly to 0.5 at –4.5×10-5 per training pattern during first

100,000 training patterns. The corresponding modeling error is also shown in Figure (5.8), which diverges right after 50 patterns even before finishing a training epoch. At that time, r has been linearly reduced from 5 to 4.9978 as in Figure (5.9). The modeling error for a certain training pattern is defined as the difference between its desired output and RNN output.

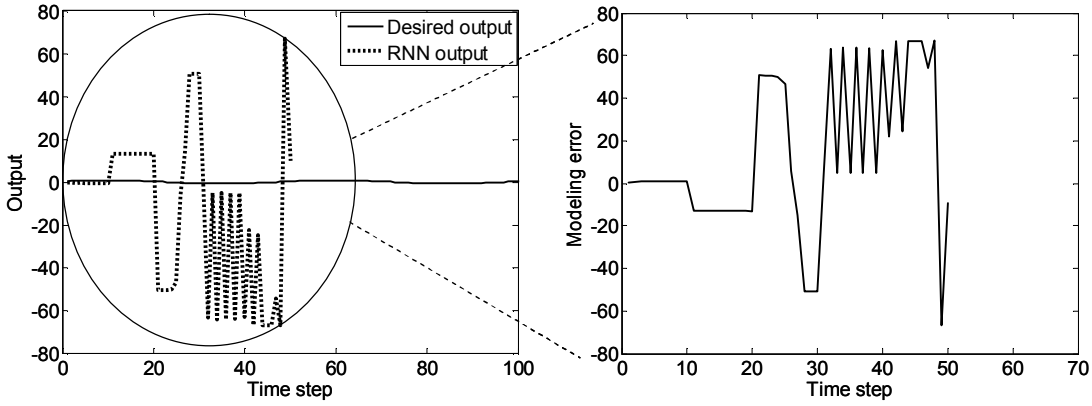


Figure 5.8: Training result and modeling error without R adaption law

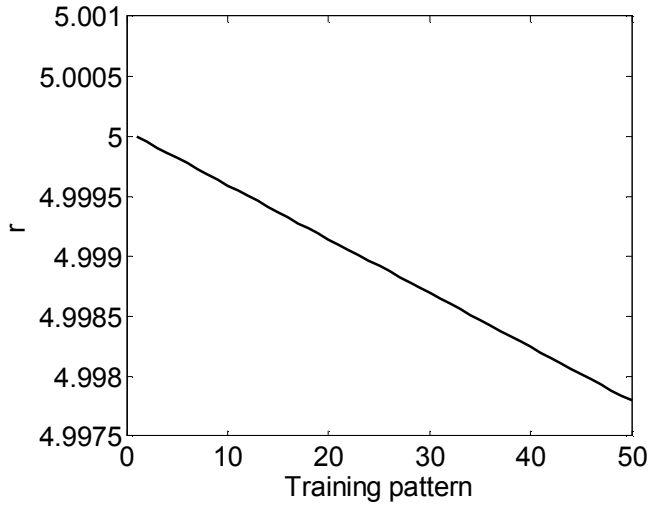


Figure 5.9: r values during training without R adaption law

To verify the effectiveness of the R adaption law in stabilizing the training process, RNN is further trained by applying the proposed R adaption law. As shown in Equation (4.43), the R adaption law takes effect when the modeling error is beyond a certain boundary ($|e(k)| > \sqrt{192 H(k)^T P(k) H(k)}$). As in the aforementioned divergent case, the r is first initialized as 5 and gradually reduced whereas the R adaption law is concurrently implemented. Corresponding to Figure (5.8), Figure (5.10) shows the convergent training results. Using the R adaption law, the training process is finally convergent and the modeling errors are much smaller and its final training error =3.6% which is comparable to the stable RNN training error, 3.7% in the previous case shown in Table (5.1).

Figure (5.11) shows the magnitude of modeling errors gradually decreases from 2 to 0.015 during the training process. Corresponding to Figure (5.9), the r values during training for this convergent case are shown in Figure (5.12). It can be seen that when the precondition in R adaption law (Equation (4.43)) is met, R adaption law takes effect – the r values deviate away from the straight line as shown in Figure (5.9). The oscillations in Figures (5.11) and (5.12) are the outcomes of the R adaption law. From Figure (5.11), it can be seen that relatively big modeling errors (about ±2) are generated in the oscillation region, while with the R adaption law the error gradually decreases during the training process and finally reach a stable small value 0.015. From these results it is clear that the R adaption law make the previously divergent training process become convergent.

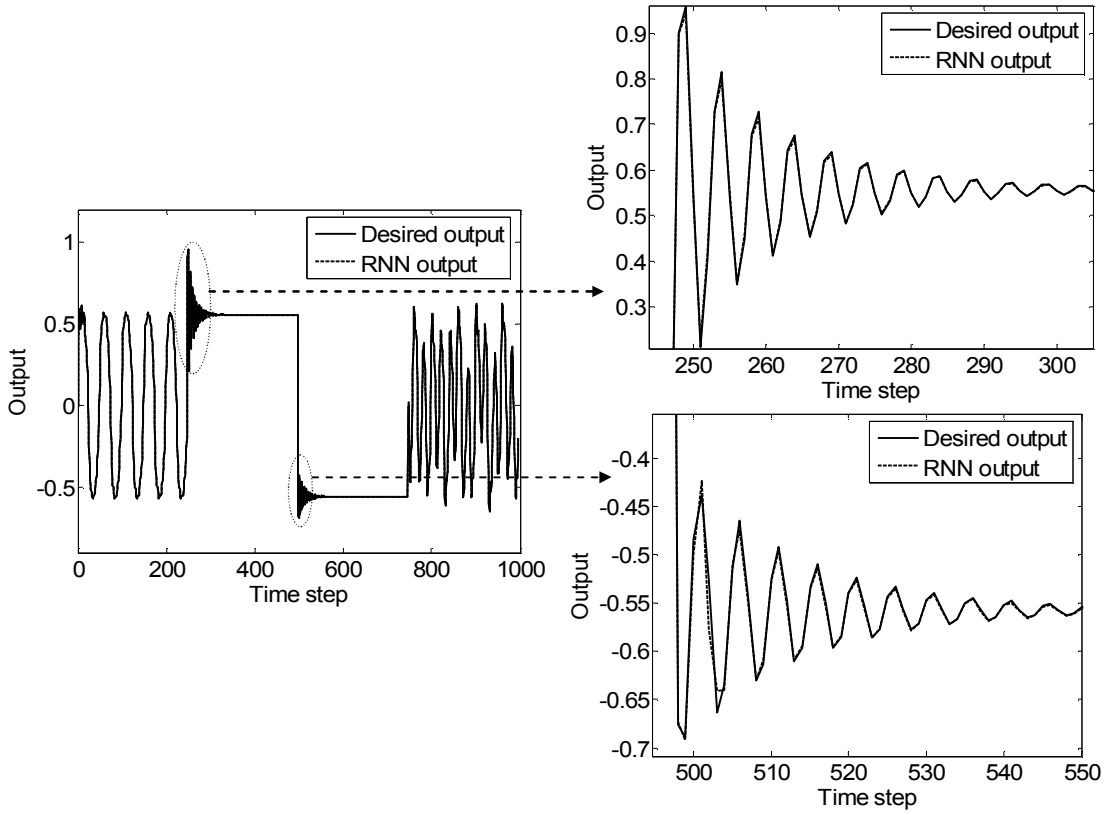


Figure 5.10: Training results with R adaption law

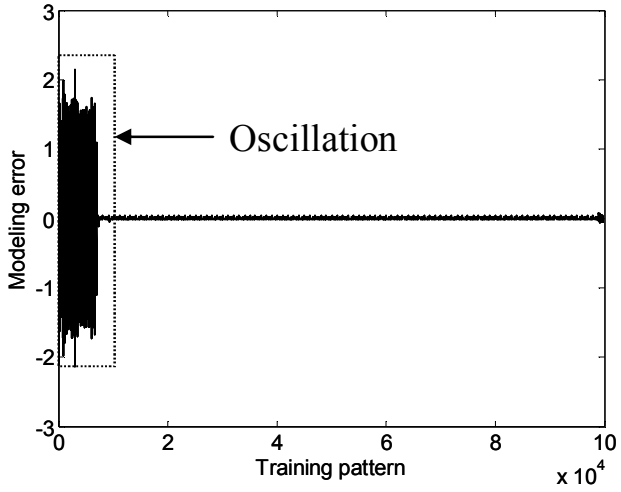


Figure 5.11: Modeling error during training with R adaption law

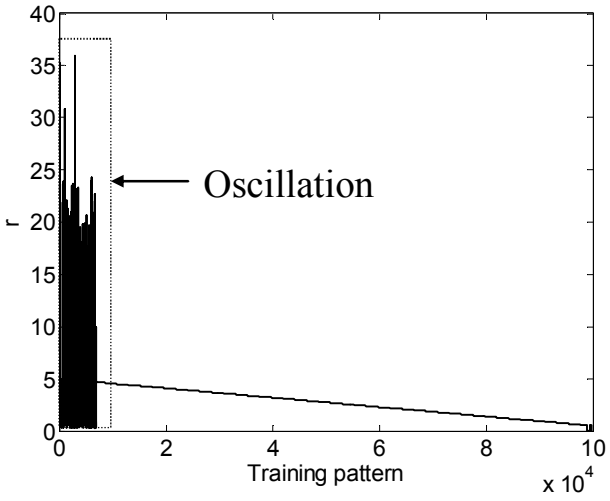


Figure 5.12: r values during training with R adaption law

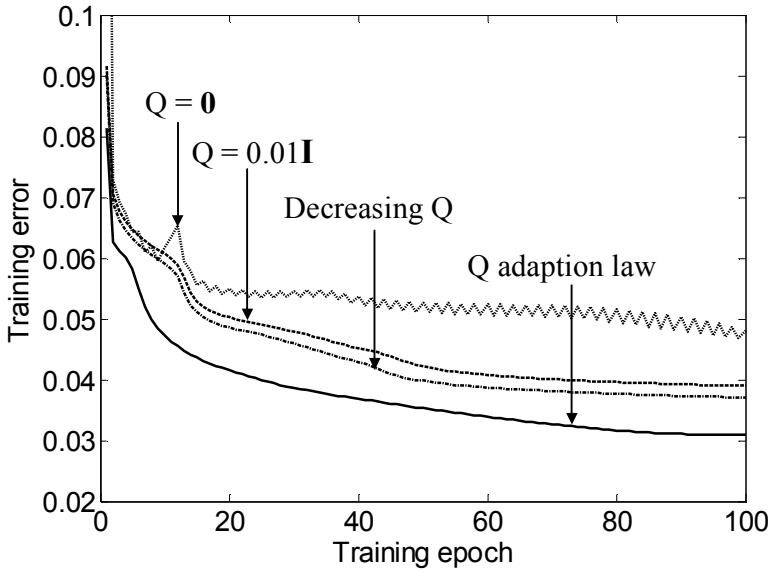
Q Adaption Law for Convergence Speed

In addition to the R adaption law, a Q adaption law derived from the maximum likelihood estimation method is applied to speed up the training convergence process. Four simulation scenarios are studied to appreciate the importance of the Q adaption law:

1) The Q matrix is set as a null matrix, which means a zero covariance matrix of process noise or the process noise is removed from the EKF training algorithm;
2) RNN is trained using constant Q (Q = 0.01I);
3) RNN is trained as follows: each diagonal element of Q is initialized as 0.01 and this value decreases linearly during 100,000 training patterns until it reaches a limit of 0.000001; and

4) The proposed Q adaption law is implemented during the training process. In all the cases r is initialized as 100 and it is reduced linearly until r reaches a limit of 2 as in a previous study [Pusk94]. It should be pointed out that the diagonal elements of Q under the third scenario are always the same whereas they might be different under the fourth scenario.

For the above four scenarios, the training process is found always stable and Figure (5.13) illustrates the effecting of introducing the Q adaption law during training. The final training error after the training process is listed in Table (5.4). As seen from Figire (5.13), the Q adaption law has helped achieve the best convergence performance with a minimum final training error (3.2%) and fastest training speed, followed by the decreasing Q setting (Scenario 3 with a 3.7% error), the constant Q setting (Scenario 2 with a 3.9% error), and the zero Q setting (Scenario 1 with a 4.8% error). Different from the other cases, in Scenario 1, the training error fluctuates during the training process. The fluctuation is due to that the update of weight is only driven by the measurement noise and the modeling error fluctuates during the training process.



. Figure 5.13: Comparison of RNN training with different Q settings

Similarly, to show the effectiveness of the proposed Q adaption law, same sets of simulations have been conducted for the optimized RNN network. The training error results are listed in Table (5.4). As shown in Figure (5.14), the results are similar to those in Figure (5.13) except that the training errors decrease much faster owning to the connectivity optimization process mentioned in Chapter three. Again, the Q adaption law has helped achieve the best convergence performance with a minimum final training error (3.0%) and fastest training speed, followed by the decreasing Q setting (Scenario 3 with a 3.6% error), the constant Q setting (Scenario 2 with a 3.8% error), and the zero Q setting (Scenario 1 with a 4.3% error).

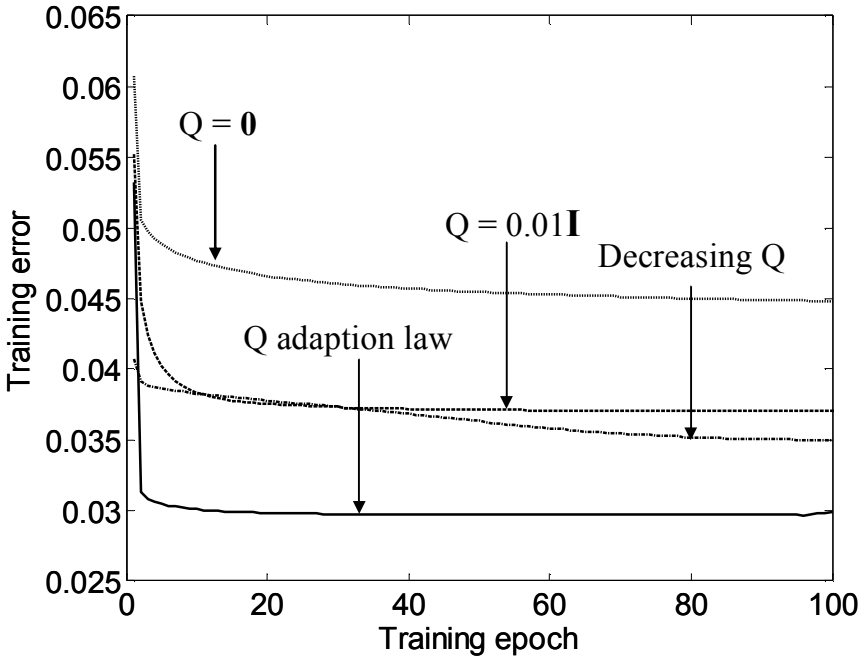


Figure 5.14: Comparison of optimized RNN training with different Q settings

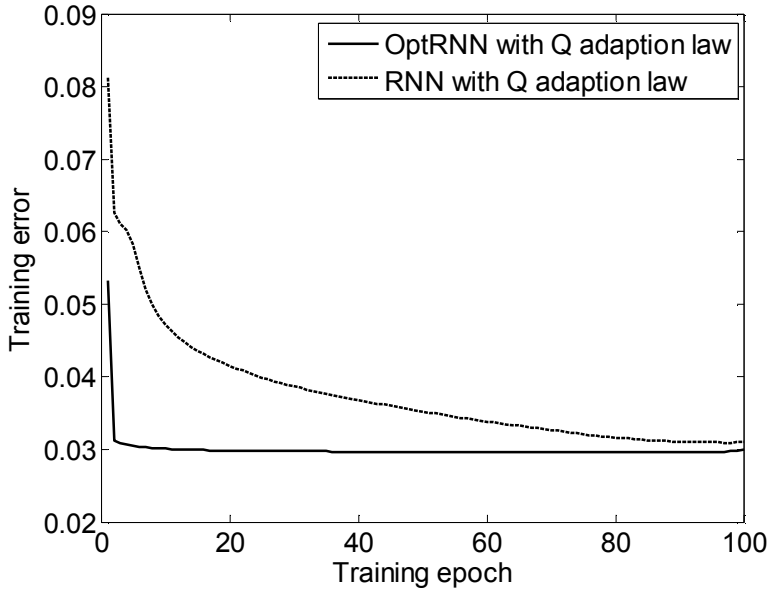


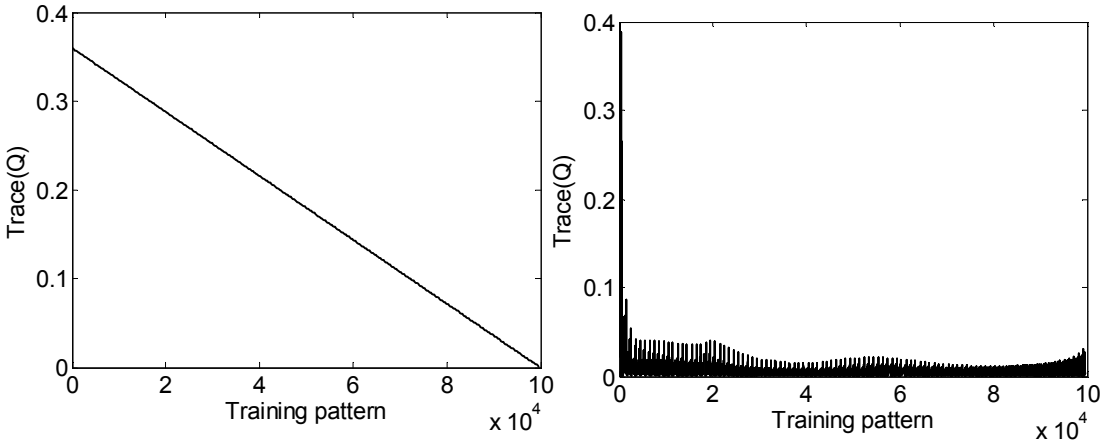
Figure 5.15: Comparison of training process of RNN and optimized RNN with Q adaption law

Figure (5.15) is drawn to better illustrate the effectiveness of Q adaption law on RNN and corresponding optimized RNN. Although the training speed of optimized RNN is much faster, the final training errors of both cases are comparable (3.1% for RNN and 3.0% for the optimized RNN).

Table 5.4: Final training errors of different Q settings

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
RNN	3.1%	3.9%	3.7%	4.8%
OptRNN	3.0%	3.8%	3.6%	4.3%

Figure (5.16) shows the trace of Q during training for Scenario 1 and Scenario 3. Trace of Q is drawn versus training patterns. For Scenario 3, the diagonal elements of Q linearly decrease during training process while for Scenario 1, the trace value reduce dramatically first and then staying on a periodical-like feature. To watch the details of Part (b), the region of training patterns [5001, 7000] is depicted in Figure (5.17). The training patterns [5001, 6000] account for training epoch 5 and the training patterns [6001, 7000] account for training epoch 6. It can be seen that in each training epoch, trace of Q is relatively high in the discontinuous regions, such as regions close to training pattern 5000, 5250, 5500, 5750 etc.



(a) Trace of Q during training for Scenario 3 (b) Trace of Q during training for Scenario 1

Figure 5.16: Trace of Q during training processes of Scenarios 1 and 3

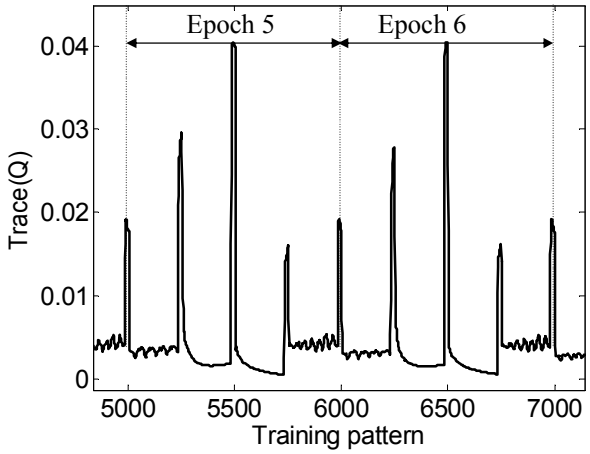
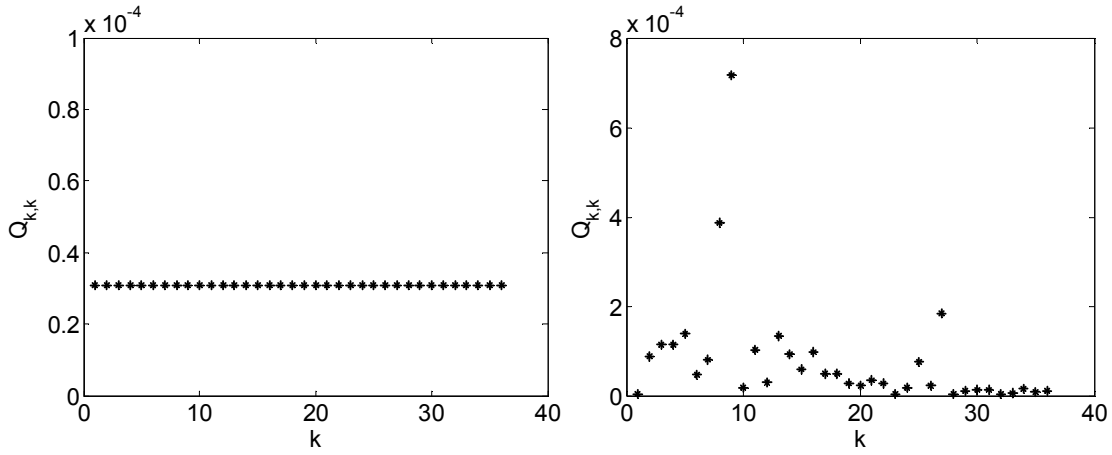


Figure 5.17: Trace of Q in training epochs (5 and 6)

Q is diagonal matrix. The diagonal element of Q after the training process is shown in Figure (5.18). In the training Scenario 3 the diagonal elements are equal while in the training Scenario 1 the diagonal elements are varied. It is obvious that training Scenario 1, which uses the proposed Q adaption law, has more freedom in setting Q.



(a) Diagonal elements of Q after training for Scenario 3

(b) Diagonal elements of Q after training for Scenario 1

Figure 5.18: Diagonal elements of Q after training scenarios 1 and 3

From the training convergence study, the following conclusion can be drawn as:

1) The R adaption law can stabilize the training process by increasing R values when modeling error in training is large. Generally small R value put more confidence to the measurement hence it is possible to make the training process converge fast. However, too small R value may make the training process diverge. Hence it is critical to use the R adaption law to guarantee the training convergence.
2) The Q adaption law can adapt Q during training which can accelerate the training convergence. It is further found that when the modeling error is relatively large, the Q values are enlarged to drive the training process more efficiently; when the modeling error is very small, the Q values are adjusted to be small, which means there is no need to change the weights too much.

Altogether, the weight update is driven by the two noises; the developed adaption laws can adapt these noises and hence stabilize NN training process, increase its training accuracy, and accelerate the training process.

Robustness Study of the Recurrent Neural Network

In a pervious section, A RNN and an optimized RNN are applied to model the non-linear dynamical benchmark system. The architectures of these two networks are both 6-9-1. Six inputs are the current and previous outputs $y_p(k), y_p(k-1)$, and , $y_p(k-2)$, two control actions $u(k), u(k-1)$ and a constant bias 1; the output of the network is $y_p(k+1)$.

In this section, the proposed robustness quantification method is applied to quantify the robustness of the trained RNN and the optimized RNN. Here two robustness measures are considered: the local robustness measure, which is input-dependent, for any specific input sample based on Equations (4.61) and the global robustness measure for overall network robustness based on the average of local robustness measures. Generally, the perturbation level should be determined based on experimental observations as the hardware/software might have during the implementation of ANN. Here the perturbation level L in Equation (4.51) has been taken as 1% for simplicity and 100 input samples have been used if not mentioned otherwise. Based on the proposed approach a smaller robustness value means higher system robustness to external perturbations.

Figure (5.19) shows the varying local robustness measures of NN using 100 sample inputs. Each point represents a local robustness measure for the nth input sample.

The global robustness measure is found to be 0.0136 by averaging the local robustness measures. Similar varying local robustness measure tendency has been observed with optimized RNN, and its global robustness measure is found to be 0.0067, which is smaller than that of NN.

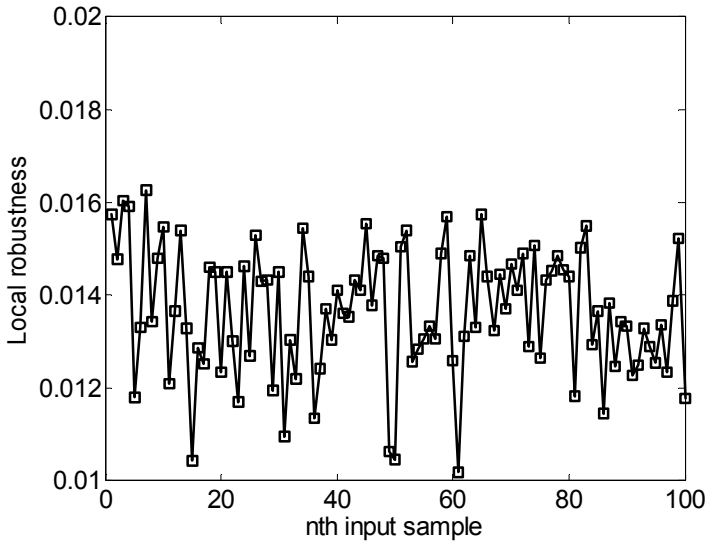


Figure 5.19: Local robustness measures for RNN using 100 input samples

The proposed global robustness measure is dependent on the two factors: the number of input samples and the perturbation level applied. The following sections study the effects of the two factors on the global robustness value.

Effect of Number of Input Samples on Global Robustness Measure

In general, the more input samples are used, the more reliable the global robustness measure represents the system robustness performance over the whole input space. Unfortunately, it is impossible to compute the global measure based on an exhaustive way by sampling all possible inputs. As so, a minimum amount of input

samples, which are needed for global robustness quantification, should be determined first. To find this minimum amount in this study, different numbers of input samples have been selected and their corresponding global robustness measures for both RNN and the optimized RNN are computed and shown in Figure (5.20).

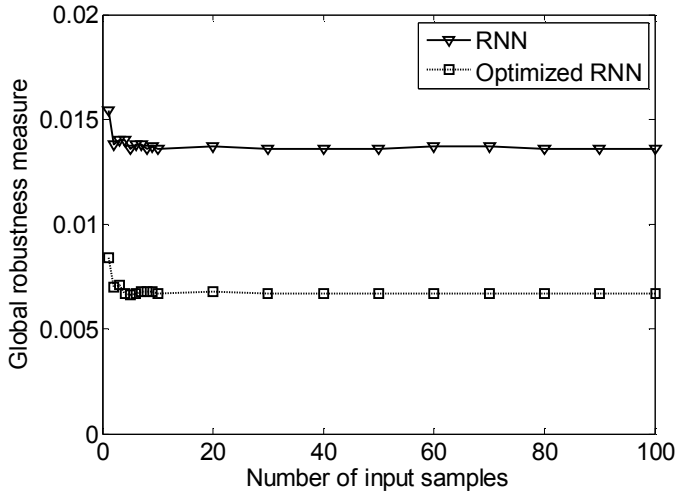


Figure 5.20: Robustness of RNN and optimized RNN (perturbation level = 1%)

It can be seen that for each network the robustness measures converge to a steady value quickly after more than 10 uniformly generated input samples are used. Based on a conservative consideration, 100 input samples are used here and in the following sections. Based on the 100 input samples, optimized RNN has a global robustness value of 0.0067, which is smaller than that of RNN (0.0136), implying that optimized NN is more robust than the regular NN [Kris93].

Effect of the Perturbation Level on Global Robustness Measure

The perturbation levels ranging from 1% to 20% have been applied to the trained network weights to study the network robustness under perturbed weights; and the results are shown in Figure (5.21). It can be seen from Figure (5.21) that the optimized RNN is more robust than the regular RNN for all the perturbation levels, which indicates the connectivity optimization process has improved the network robustness as observed before [Kris93].

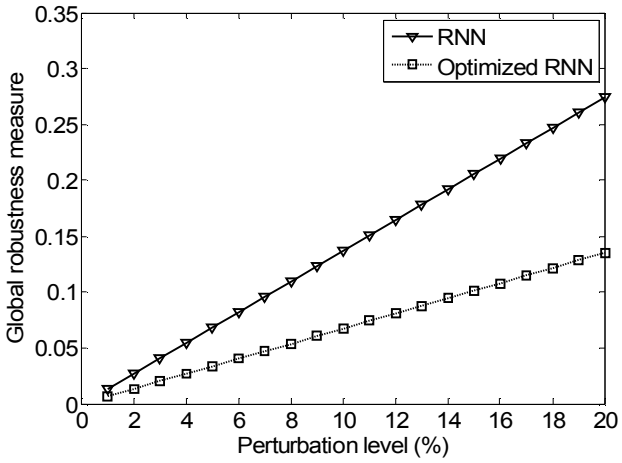


Figure 5.21: Network robustness values under different perturbation levels

It can also be seen that the relationship between the robustness measure and the perturbation level can be approximated as linear. This linear pattern is attributed to the following reason. All the input and output neurons use a linear activation function, only hidden neurons which adopt a sigmoid activation function that may generate nonlinearity. However, most hidden neurons work in the linear region of their activation functions

under small perturbations, which may lead to a linear mapping between the perturbation and the robustness measure.

Comparison among Robustness Measures

The proposed robustness quantification approach is further compared with the performance loss-based and sensitivity matrix-based approaches. The performance loss-based and sensitivity matrix-based measures are computed based on the training data mentioned in the previous section. To fairly compare the three approaches, the same training data are also used to compute the proposed robustness measure, and the LHS input sampling process is not applied here to generate input samples for the proposed approach.

For both the performance loss-based and the proposed approaches, the 1% perturbation level is used. For the performance loss-based approach, 10,000 networks are generated based on the perturbed weights, and the network output is then compared with the corresponding desired outputs to compute MSE. The resulting maximum MSE is taken as the performance loss-based measure. For the sensitivity matrix-based approach, the sensitivity matrices for all the training inputs are obtained during the training process and the spectral norm [Kris99] for each matrix is computed as the local robustness measure. It should be emphasized that the output of the non-linear dynamical benchmark system is a scalar, so the sensitivity matrix is actually a vector. It is known that the matrix norm corresponding to the Euclidean vector norm is the spectral norm [Baks80],

therefore the spectral norm of a vector is the same as its Euclidean norm. The average of these norms is used as the sensitivity matrix-based measure.

It should be pointed out that the three robustness measures cannot be directly compared against each other because they are computed using different criteria. Instead, the ratio of robustness measures between RNN and optimized RNN is studied to indicate the effectiveness of any quantification approach. A larger robustness ratio means that this quantification approach is more sensitive in quantifying the robustness difference.

Table 5.5: Comparison of robustness quantification approaches

Robustness quantification approach	RNN robustness (r1)	Optimized RNN robustness (r2)	Robustness ratio (r1/r2)
Performance loss-based	0.0395	0.0342	1.16
Sensitivity matrix-based	2.7163	1.9988	1.36
Proposed	0.0138	0.007	1.97

Table (5.5) lists the comparison results. For all the three quantification approaches, the optimized RNN is found to be more robust than RNN. It is found that the proposed approach (1.97) has the largest robustness ratio than those of the performance loss-based approach (1.16) and the sensitivity matrix-based approach (1.36). As the largest robustness ratio value is associated with the most sensitive quantification approach, it is concluded that the proposed robustness quantification approach is the most effective one among these three approaches.

Relationship between Proposed and Sensitivity Matrix-based Approaches

Under a small perturbation level the uncertainty propagation analysis used in this proposed approach can also be related to the sensitivity matrix–based approach. Each

element ($H_{ij} = \frac{\partial y_i}{\partial w_j}$) of the weight–output Jacobian sensitivity matrix H represents the derivative of an output (y_i) with respect to a weight (w_j). Using the uncertainty propagation analysis under a small weight perturbation such as 0.01%, this H_{ij} can be approximated by the standard deviation ratio $S_{ij} = \frac{std(y_i^{(j)})}{std(w_j)}$, where std(·) is a standard deviation operator and $std(y_i^{(j)})$ represents the standard deviation of output y_i under a perturbation with weight w_j. This standard deviation ratio S_{ij} describes the dependence of the output variation on the weight variation. It should be pointed out that different from the proposed robustness quantification approach where perturbations are added to all the weights simultaneously to compute $\bar{\sigma}_{\bar{y}(j)}$, here each time perturbation is only added into a specific weight w_j to compute $std(y_i^{(j)})$ while all the other weights remain the same.

Table (5.6) lists the sensitivity matrix-based robustness measures averaged based on their spectral norms [Kris99], which are computed using the traditional method (H_{ij}) and the uncertainty propagation analysis (S_{ij}), respectively. It is found that the two results quite match each other; therefore the proposed analysis can also be used to compute the sensitivity matrix-based measure.

Table 5.6: Sensitivity matrix-based robustness of RNN and optimized RNN

	RNN robustness	Optimized RNN robustness
Robustness (*H* matrix based)	2.7163	1.9988
Robustness (*S* matrix based)	2.7274	1.9946

Efficiency of the Proposed Robustness Quantification Method

The unscented transform is efficient to quantify the uncertainties in RNN output; to verify that, another uncertainty propagation method using Monte Carlo analysis is carried out. A set of weight vectors are randomly generated based on the Gaussian distribution of the perturbed weight vector used before, and each of them form a RNN. The inputs are fed into these RNN, and the standard deviation in networks' output are used as the local robustness measure, which further forms the global robustness measure by averaging. Different numbers of RNN are generated and the computation times are recorded in Table (5.7). It is found that to reach the same robustness result (0.0138), 2000 RNN need to be generated and the computation time is about 6 times of the proposed method, whose corresponding results are shown in Table (5.8). The experiment is carried out on a computer with the configuration of Intel(R) Core(TM) 2 Duo CUP @ 2.8GHz and 3.0 G RAM.

Although the proposed robustness measure is developed based on the assumptions that same level of perturbation is introduced to all the weights, in real applications the perturbation level for weights can be specified based on measurements and the same procedure can be applied.

Table 5.7: Robustness results of RNN from a Monte Carlo method

Number of RNN generated	Robustness measure	Computation time (second)
10	0.0099	9.21
100	0.0132	95.82
500	0.0133	464.28
1000	0.0137	960.07
2000	0.0138	1895.31

Table 5.8: Robustness results of RNN from the proposed UT-based method

Number of RNN generated	Robustness measure	Computation time (second)
301	0.0138	284.91

From these results in robustness study, it can be seen that:

1) The proposed robustness quantification method is flexible and viable. It can study robustness of a network under different levels of perturbation level. It does not need the training data, instead it is an uncertainty propagation method and only a few amounts (100) of input samples are required to quantify network's robustness.
2) As an uncertainty propagation based method, because of the application of the unscented transform, it is more efficient than the Monte Carlo simulation based method.
3) The proposed robustness quantification is more effective compared with the other two methods.
4) The optimized RNN is more robust than RNN.

Conclusions

From this case study, some conclusions can be drawn that:

1) RNN network is capable of modeling the non-linear dynamical benchmark system.
2) The modeling capability of RNN is enhanced through the connectivity optimization process and the optimized RNN excels RNN in training speed and modeling accuracy.
3) The proposed R and Q adaption laws can further improve a RNN's training convergence performance - to stabilize and accelerate its training process.
4) The developed uncertainty propagation analysis based robustness measure is more flexible and effective than the other two methods.

All of the results prove that the developed RNN modeling approach is powerful in terms of accuracy, speed, and stability.

CHAPTER SIX

MODELING OF CBN TOOL WEAR IN HARD TURNING

Abstract

In addition to the benchmark system, another application is used to test performance of the developed RNN modeling tool. Hard turning with Cubic Boron Nitride (CBN) tools has been proved to be more effective and efficient in turning hardened steels than traditional grinding operations. However, rapid tool wear is always a problem which hurdles the wide implementation of hard turning in industry. Therefore, a better understanding of the CBN tool wear progression will help optimize cutting conditions and tool geometry to reduce tool wear, which may make hard turning a viable technology. The goal of this case study is to use the optimized RNN to model the tool wear progression and further investigate the network's performance in training convergence and robustness. The results show that the developed optimized RNN have advantages over FFNN in modeling the tool wear progression in hard turning and the convergence study and robustness study can further improve the network's performance.

CBN Tool Flank Wear

Based on a typical CBN tool wear observation [Daws02], CBN tool flank wear length or wearland (VB), as shown in Figure (6.1), is generally regarded as the tool life criterion or an important index to evaluate the tool performance in hard turning [Taka83] [Abra95] [Dewe96]. The tool wear rate is assumed uniform across the width of cut as shown in Figure (6.1).

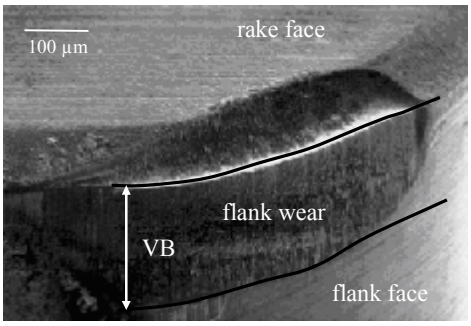


Figure 6.1: Typical tool wear picture in CBN hard turning

Figure (6.2) shows a typical tool wear progression process which is to be modeled by the proposed RNN and the optimized RNN. Usually many factors would affect the process. For a given tool and workpiece combination, the capability to estimate the tool wear as a function of cutting conditions as cutting speed, feed rate, and depth of cut, is critical to the overall optimization of a hard turning process. The objective of this case study is to model the tool wear progression process using proposed RNN networks.

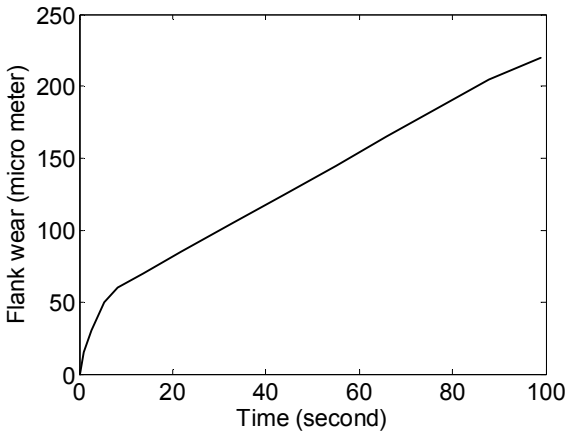


Figure 6.2: A Typical tool wear progression in hard turning

Experimental Setup

The Data are collected from an experiment [Huan04]. In that experiment, hardened AISI 52100 bearing steel with a hardness 62 HRc was machined on a horizontal Hardinge lathe using a low CBN content tool insert (Kennametal KD050) with a -20º and 0.1 mm wide edge chamfer and a 0.8 mm nose radius. The ISO DCLNR-164D tool holder was used, which introduced a negative 5º rake angle. No cutting fluid was applied. Flank wear length was measured using an optical microscope (Zygo NewView 200). The experiment was stopped when a sudden force jump was observed signaling a chipping or broken tool condition.

Table 6.1: Cutting conditions of the experiments [Huan04]

Condition index	Speed (m/s)	Feed (mm/re)	Depth of cut (mm)
1	3.05	0.152	0.203
2	1.52	0.152	0.203
3	3.05	0.076	0.203
4	2.29	0.114	0.203
5	1.52	0.076	0.203
6	3.36	0.114	0.203
7	2.29	0.114	0.203
8	2.29	0.061	0.203
9	2.29	0.168	0.203
10	1.21	0.114	0.203
11	2.29	0.114	0.203
A	1.52	0.076	0.102
B	1.52	0.076	0.152

Machining test was performed based on a standard central composite design test matrix with an alpha value of 1.414. The center point (0,0) was determined based on the tool manufacturer's recommendation [Huan04]. A typical depth of cut was suggested as 0.203 mm, which was used in the test matrix. To further investigate the effect of depth of cut on tool wear, experiments with various depths of cut were also studied. Ten different cutting conditions [Huan04], namely conditions 1-5, 8-10, a, and b are listed in Table (6.1). Conditions 7 and 11 are not utilized here since they are the same as condition 4, and condition 6 (cutting speed = 3.36 m/s) is also not used since the break-in period accounted for a large portion of tool flank wear and microchipping was a dominant factor of tool life under such an aggressive cutting speed. Uncertainty characterization is not offered here due to the size of the experimental data set.

Recurrent Neural Network Implementation

In this study, a RNN and an optimized RNN are formed to model the CBN tool wear progression based on the data from Huang et al's study [Huan04] and their modeling performance is compared with the measurements as well as that of FFNN approaches from previous studies [Wang08a] [Wang08b]. The following tasks need to be conducted to train a RNN modeling the tool wear progression:

1) Training and testing data preparation

As shown in Table (6.1), there are total 10 groups of data available from the hard turning experiment. Among them data of conditions 1, 5, 9, 10, and a are used for network training. The training data contain 48 training patterns. The rest of the data (44 patterns) are used to test the generalization ability of the proposed RNN model.

2) Training parameters configuration

To train RNN, some training parameters such as P, Q, and R need to be initialized first. The training parameters configuration is referred from a previous study [Pusk94] without considering the training divergence issue. Without any specific note, in this study, the error covariance matrix P is initialized as a diagonal matrix and each of its diagonal elements is initialized as 100. Each diagonal element of the covariance matrix of process noise Q is initialized as 0.01 and this value descends linearly within 100,000 training cycles until Q reaches a minimum limit of 0.000001. Similarly, each diagonal element of the measurement noise covariance matrix R is initialized as 100 and it also descends linearly until it reaches a minimum boundary of 2. Both the settings of R and Q help the training error converge to a global minimum.

3) Training Process Configuration

First, each weight of the network is randomly initialized in the region of [-1, 1]. Training parameters are initialized as mentioned before. Training data are then fed into the EKF training algorithm (Equations (3.47-3.49)) to train the network weights. During the training process, the training data are used for each training epoch and the weights are updated accordingly. The procedure of training using all the training patterns once is called a training step or epoch. The training process stops when the stop criteria are satisfied. The stop criteria are determined by trial-and-error: i) the number of training step should be less than 500 and the training process stops after 500 steps if no other stop criteria are met; or ii) if the training error is less than 3% and the difference between the current error and the error of 20 epochs before is less than 0.03% [Wang09].

4) Network structure determination

RNN network structure is first determined by setting the numbers of input neurons, hidden neurons and output neurons. Four independent variables - cutting speed, feed rate, depth of cut and machining time and a constant bias 1 are used as the inputs. The output of the network is the tool flank wear length. The number of hidden neurons is determined by a trial and error method, and the training error results are listed in Table (6.2). According to a rule of thumb [Scha97], 12 hidden neurons are chosen first and the training error is found to be 4.2%. Afterwards, networks with fewer hidden neurons are selected and the corresponding training errors are investigated. It is found that RNN with more than 1 hidden neuron is able to adequately model the tool wear progression. For example, the training error of the network (5-2-1) with 2 hidden neurons (4.5%) is quite

close to that of the network (5-12-1) with 12 hidden neurons (4.2%). However, for the network with 1 hidden neuron, the training error becomes relatively large (12.8%). The training results for typical networks are shown in Figure (6.3). The simple network structure would reduce the risk of over-fitting, so the network with a 5-2-1 structure as shown in Figure (6.4) is selected in this study.

Table 6.2: Training error with different network structure

Network structure	Training error (%)
5-12-1	4.2
5-2-1	4.5
5-1-1	12.8

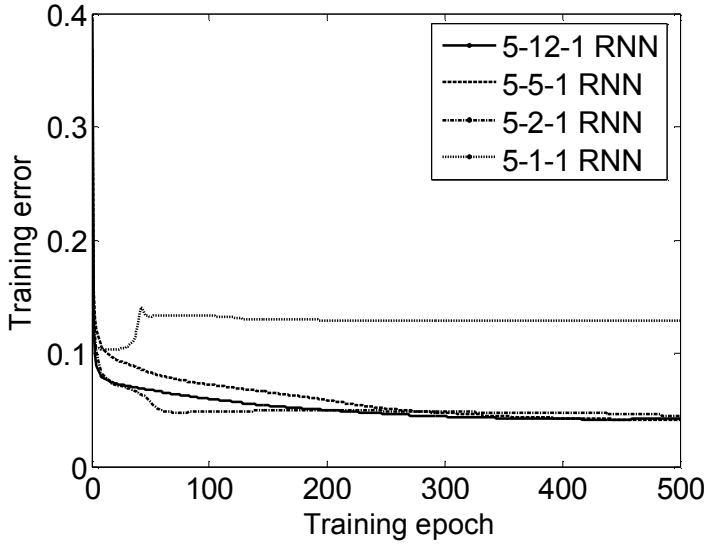


Figure 6.3: Training errors of RNN with typical structure configurations

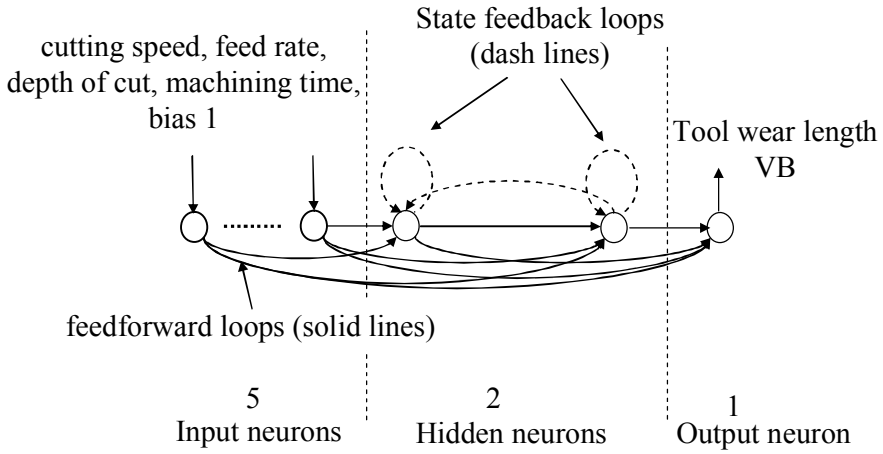


Figure 6.4: Modeling the tool wear progression by a 5-2-1 RNN

Modeling Performance of the Recurrent Network

An MLP, an FFCNN, an optimized FFCNN (OptFFCNN), a RNN, and an optimized RNN (OptRNN) are applied to model the process. Both the training performance and testing performance are studied.

Training results indicate the fitness of the network model in modeling the training data. Training data are assumed to be able to represent the overall characteristics of the system being studied. Therefore, from the training process, a network capable of modeling the training data is expected to represent the system dynamics.

During the training process, the appropriate network architecture should be determined first as stated in a previous section. The MLP has been found to be 5-5-1 and the other networks (RNN and OptRNN) are 5-2-1. The same training data (conditions 1, 4, 5, 9, and a) have been used to train these networks. 500 training epochs are used for RNN and OptRNN training while 100000 training epochs are used for MLP training

since MLP converges much slower. The final training error results are listed in Table (6.3).

Table 6.3: Training error with different types of networks

Network	Training error (%)
MLP	4.8
RNN	4.5
OptRNN	4.4

From the results, all the training errors are smaller than 5% (MLP: 4.8%, RNN: 4.5%, and OptRNN: 4.4%). Figure (6.5) shows some representative training result comparisons. It can be seen that the modeling performance of these investigated NN are close in modeling the training data and all the networks are able to accurately represent the training data and model the tool wear progression.

From Figure (6.5), it should be pointed out that the training results of conditions 10 and a are more accurate than those of conditions 1 and 5. It is because that during this pattern learning process the network is trained orderly from condition 1 to condition a. As a result, more training effort has been put to the most recent training patterns. While the overall training error of the OptRNN is generally smaller than that of MLP, the OptRNN may have large errors for some specific training patterns.

The trained networks are further tested for their generalization ability. Conditions 2, 3, 4, 8, and b have been used as the testing cases, which are unseen in the above network development process.

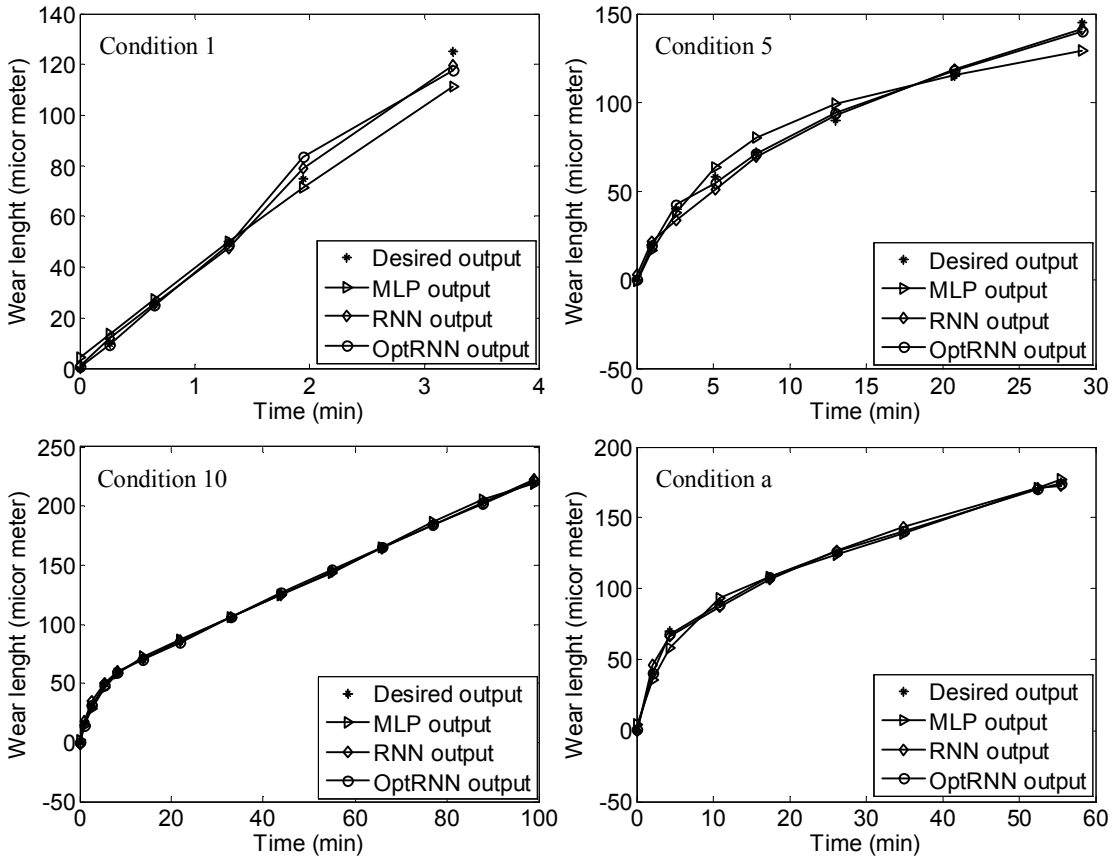


Figure 6.5: Training results for training cases

Figure (6.6) shows the testing results from the MLP, RNN, and OptRNN. It can be seen that:

1) Except for a few testing patterns in condition 4, RNN is more capable of accurately modeling this non-stationary and dynamical tool wear progression than the MLP;
2) For most cases, the discrepancy between the network prediction and its desired output (the experimental measurement) increases with time;

3) RNN and OptRNN have the similar modeling performance in this tool wear study mainly due to their inherent recurrent architectures; and
4) The MLP tends to over estimate the tool wear length for all the testing cases which implies its limitation in modeling this non-stationary and dynamical system.

The modeling performance is further compared with those of an FFCNN approach and an OptFFCNN approach [Wang08a]. Table (6.4) shows the testing errors for these networks which better illustrate their overall modeling capability. Some observations can be drawn as follows:

1) The average testing errors of optimized networks (the optimized FFCNN and the optimized RNN) are smaller than those of their corresponding regular networks (FFCNN and RNN);
2) The average testing errors of recurrent networks (RNN and the optimized RNN) are smaller than those of purely forward networks (MLP and FFCNN); and
3) From the variance of errors, the MLP has the largest variation which means it is the least robust network while the optimized networks have smaller variances (8.0 and 6.1) which indicates the optimization process can improve network's robustness as well as their modeling accuracy.

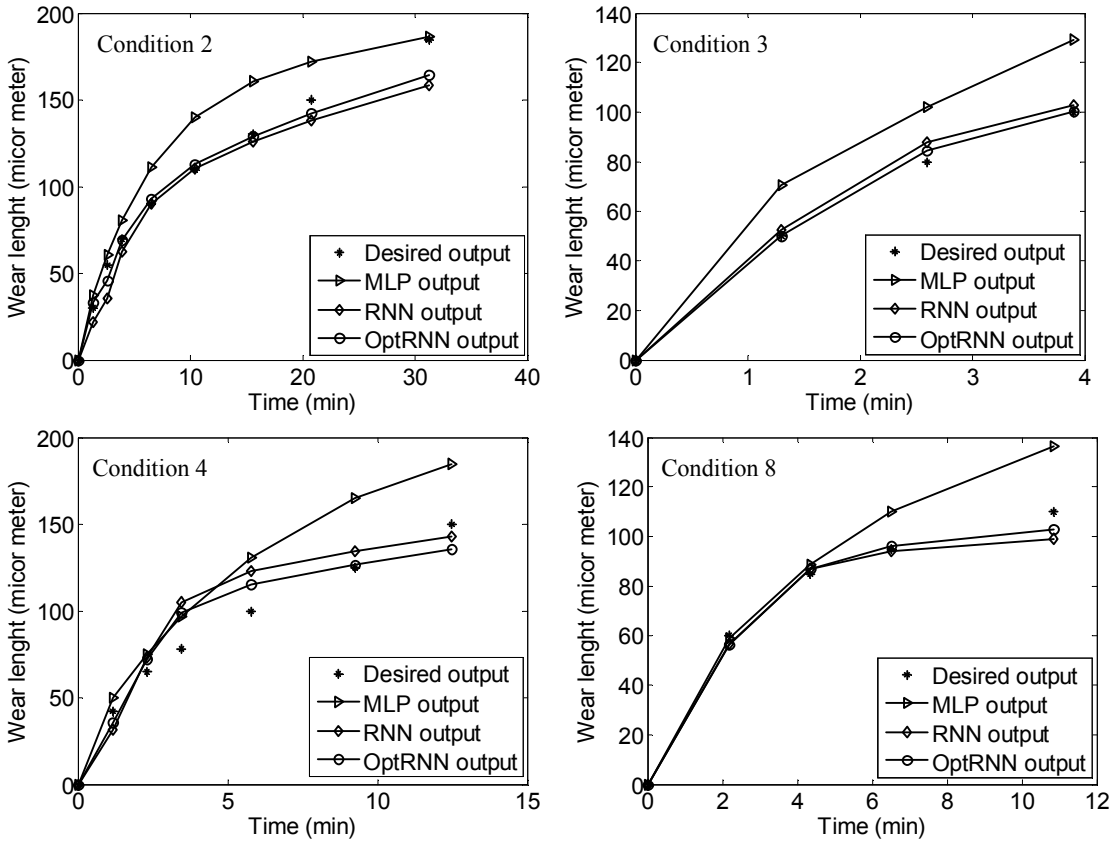


Figure 6.6: Modeling results for testing cases

Table 6.4: Modeling error for testing cases

Condition Index	MLP (%)	FFCNN (%)	Optimized FFCNN (%)	RNN (%)	Optimized RNN (%)
2	8.37	10.65	10.29	10.63	7.65
3	39.45	26.32	12.77	16.42	10.27
4	17.94	22.03	11.51	16.09	12.62
8	15.57	15.27	13.71	9.35	8.65
B	5.96	6.50	5.60	5.84	5.29
Avg. of error	17.46	16.15	10.78	11.67	8.90
Var. of error	140.4	52.4	8.0	16.5	6.1

Training Convergence Study of RNN

In the previous study, parameter setting of EKF training algorithm is borrowed from [Pusk94]. The parameter R represents the confidence of noised measurement, the smaller the value the more confidence would put on the measurements. From Equation (3.48), small R also leads to high learning rate. However, too small R may lead to excessively large learning rate which cause the training instability problem. On the other hand, the covariance matrix of measurement noise, R, also indicates the amount of noise added in the measurements. In the beginning stage of training process, the network output is far from the desired output. The network output can be viewed as the estimation result of a measurement with large noise. Hence, the diagonal element of R is supposed to set as a large number (100) at the beginning stage. During the training process, the modeling error become smaller and the network output can be viewed as the estimation of a measurement with small noise. Follow this intuition, the R setting in [Pusk94] provides an empirical guide for R configuration. However, the R setting can't guarantee the convergence of a training process. In contrast the R adaption law is proposed in this study and the results are presented in the following section.

R Adaption Law for Convergence Guarantee

The R adaption law is first verified in this case study. Similarly as in the benchmark system case study, a divergent case is first illustrated and the R adaption law is applied to make the process convergent. A divergence training case is shown in Figure (6.7). The r in Equation (4.32) is initialized as 0.45 and linearly reduced to 0.1 during

training with 100000 training patterns and the modeling error goes to infinite after 922 patterns.

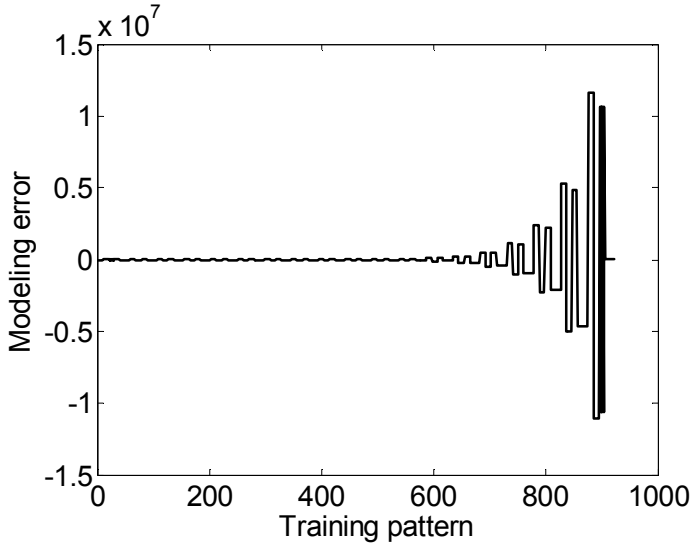


Figure 6.7: A divergent training processes

For the above case, its modeling results have big errors. The final modeling performance (trained with 922 training patterns) of case b) is shown in Figure (6.8). Large modeling errors (more than 1000 times of the magnitude of the measurements) are observed in Figure (6.8).

Correspondingly, the r values during the training process are shown in Figure (6.9) as below. This R setting leads to the divergence problem indicated in Figure (6.8).

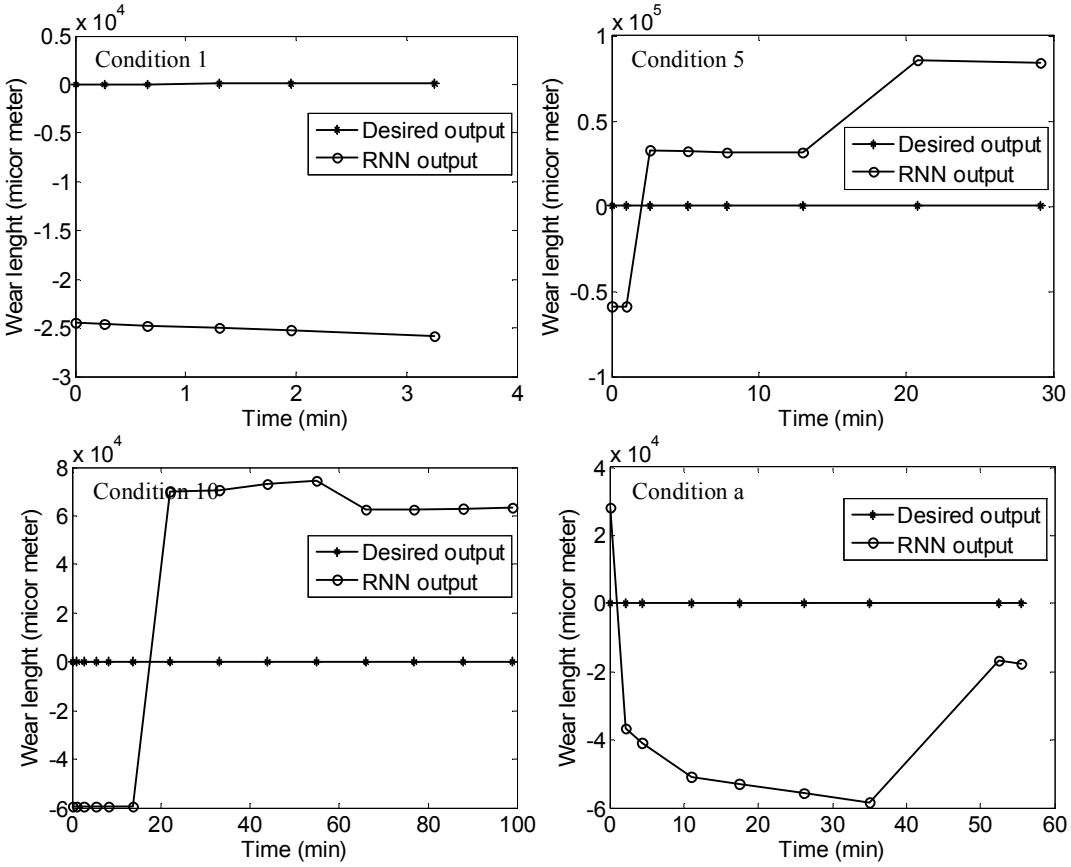


Figure 6.8: Modeling performance for tool wear progression without R adaption law

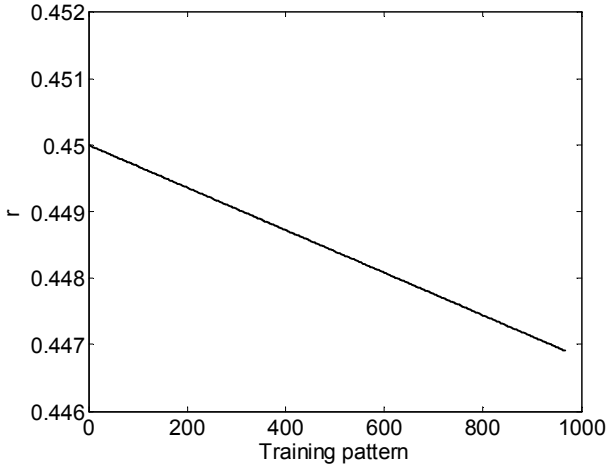


Figure 6.9: r values during training without R adaption law

RNN is also trained using the proposed R adaption law. The r value is initialized as in the aforementioned divergent case, and the R adaption law takes effect when the modeling error is beyond the certain threshold as specified by Equation (4.43). Under the R adaption law, the training process became convergent as shown in the modeling error during training plot, Figure (6.10). It is found that two groups of big oscillations occur in the beginning region of the plot. But the training error hasn't blow up to infinity during the training process and finally it converge to a small value, 4.1%.

In addition to modeling errors, the training results under adaption are also shown in Figure (6.11). Comparing to the divergent case in Figure (6.7), the modeling errors here are relatively small.

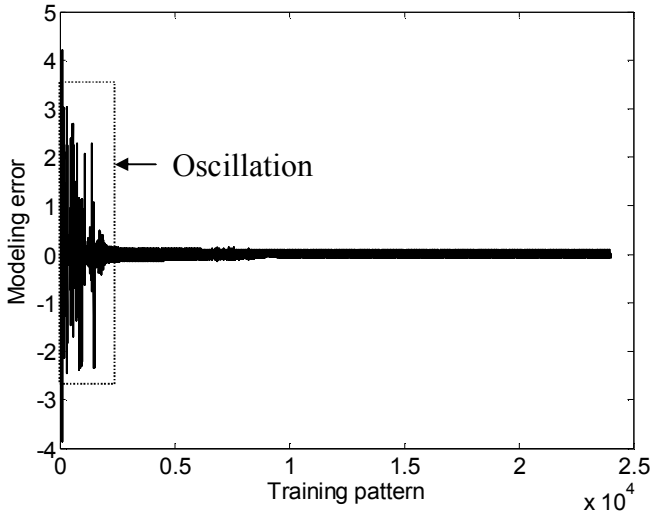


Figure 6.10: Modeling error during training with R adaption law

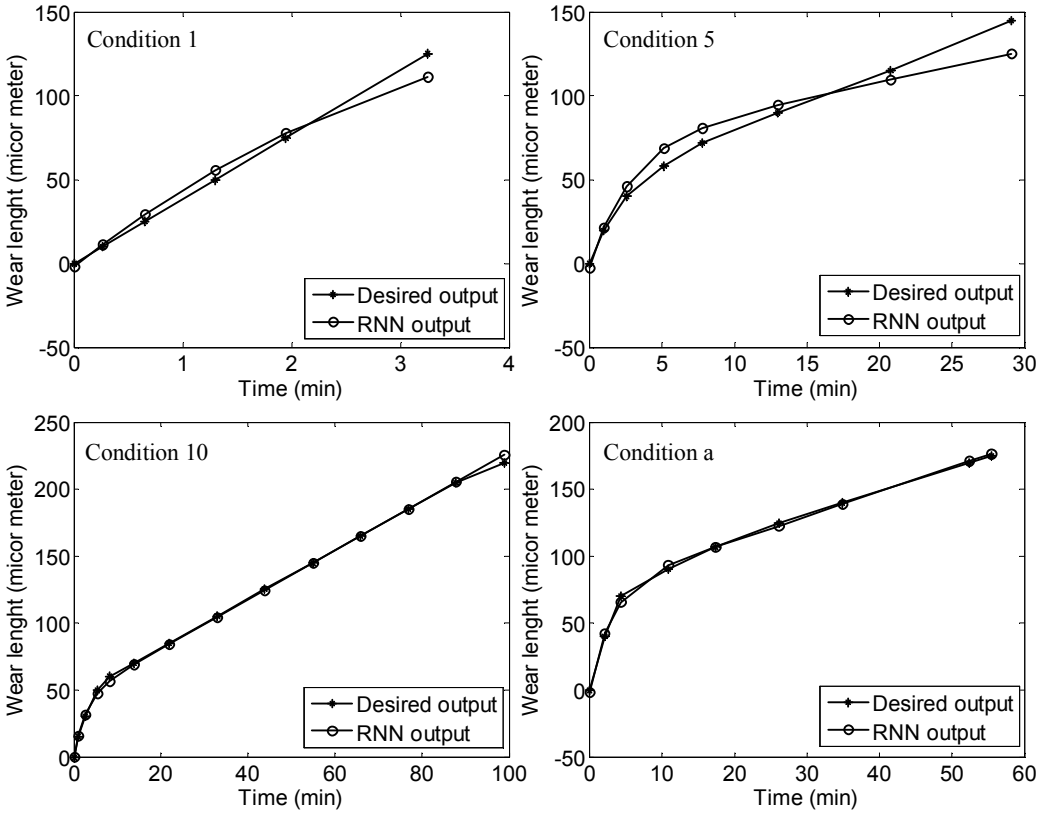


Figure 6.11: Training results for tool wear progression with R adaption law

To show the effect of the adaption law on R, the adapted r during the training process is drawn in Figure (6.12). Again, the big variations account for the effect of R adaption law. It can be also seen that the oscillations in Figure (6.12) are coherent with oscillations in the modeling error plot Figure (6.10). When the modeling error is beyond the boundary in Equation (4.43) ($|e(k)| > \sqrt{192 H(k)^T P(k-1) H(k)}$), the adaption law takes effect to generate a larger r to draw the training process to be convergent. From these results of training in Figure (6.7-6.12), a conclusion can be drawn that the R adaption law can draw the training process from divergence to convergence.

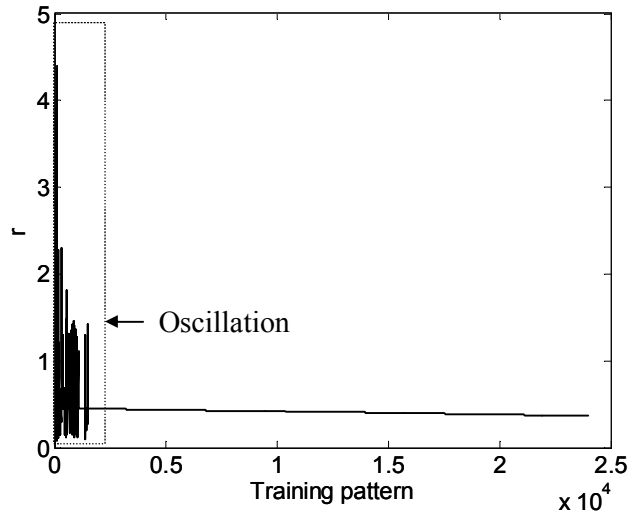


Figure 6.12: r values during training with R adaption law

Q Adaption Law for Convergence Speed

In addition to the R adaption law, the Q adaption law (Equation (4.43)) developed in Chapter four is applied in this case study as well. The effect of the Q adaption law is also tested in tool wear modeling under the same representative cutting conditions. As in the benchmark validation chapter, the four simulation scenarios are studied to appreciate the importance of the Q adaption law:

1) The Q matrix is set as a null matrix, which means a zero covariance matrix of process noise or the process noise is removed from the EKF training algorithm;
2) RNN is trained using constant Q (Q = 0.01I);

3) RNN is trained as follows: each diagonal element of Q is initialized as 0.01 and this value decreases linearly during 100,000 training patterns until it reaches a limit of 0.000001; and

4) The proposed Q adaption law is implemented during the training process. In all the cases r is initialized as 100 and it is reduced linearly until r reaches a limit of 2 as in a previous study [Pusk94]. It should be pointed out that the diagonal elements of Q under the third scenario are always the same whereas they might be different under the fourth scenario.

The results for RNN training are shown in Figure (6.13) and the final training errors are listed in Table (6.5). It can be seen that the Q law is most effective to minimize training error and accelerate the training process. The training process with Q=0 has the largest training error. For the other two scenarios, they are overlap in some regions and it is difficult to compare their effectiveness.

Table 6.5: Final training errors of different Q settings

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
RNN	3.8%	4.4%	4.5%	5.4%
OptRNN	3.6%	4.2%	4.0%	4.5%

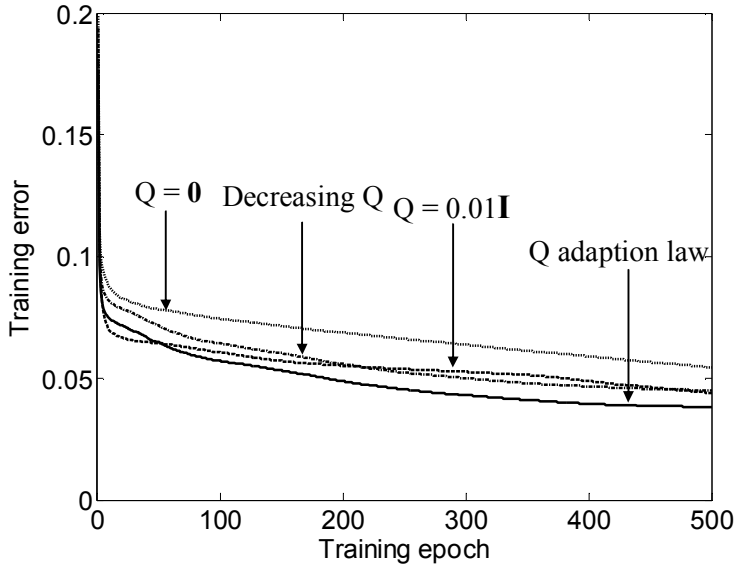
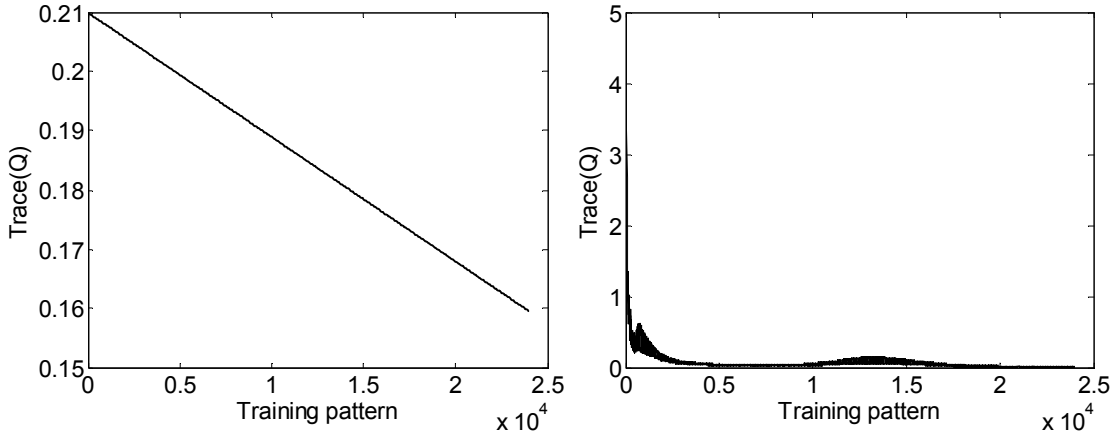


Figure 6.13: Comparison of RNN training errors with different Q settings

To investigate how the Q law affects the training process, Figure (6.14) shows the trace of Q during training processes under Scenarios 3 and 1. It is clear that the trace of Q decrease linearly in Scenario 3, while the trace of Q in Scenario 1 decreases more dramatically in the beginning region. A sub-section of Figure (6.14) is detailed in Figure (6.15) which reveals the periodicity feature for training patterns. It can be seen that the Q's are specified for training patterns in each training epoch. Further comparing this figure to the training patterns, it is found that the high Q's correspond to big changes in training patterns. It means the adaption law generates big Q's for high variation in the training patterns. High Q represents high Kalman gain hence the law put more effort to model these patterns. This is the first factor that contributes to the improvement in modeling performance in Scenario 3.



(a) Trace of Q during training for Scenario 3 (b) Trace of Q during training for Scenario 1

Figure 6.14: Trace of Q for Scenarios 1 and 3 during training process

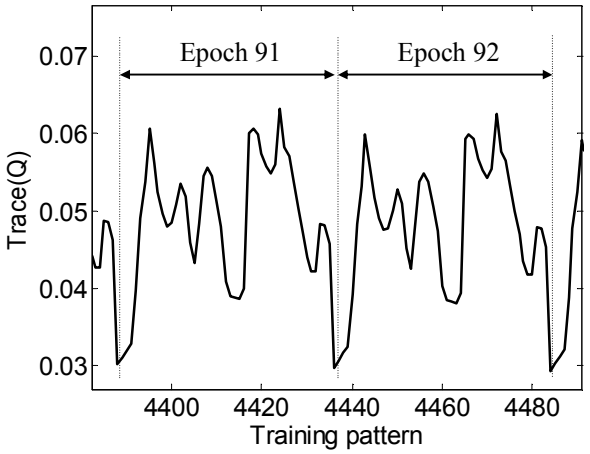
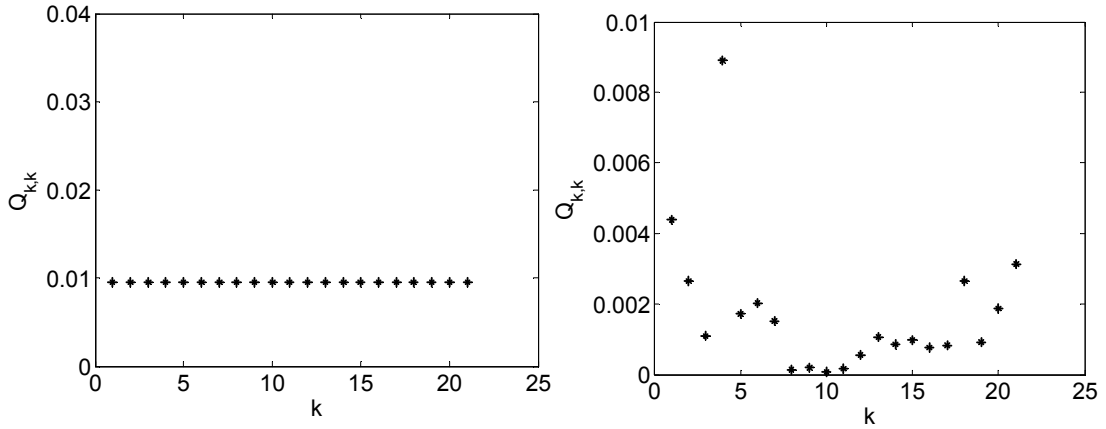


Figure 6.15: Trace of Q in training epochs (91) and (92)

Figure (6.16) shows the diagonal elements of Q after training under Scenario 3 and 1. For Scenario 3, the elements have the same values while for Scenario 1 different values are assigned to the elements of Q and hence different training strengths are to be

put on different weights based on needs. This is the second factor that contributes to the higher modeling capability of Scenario 3.



(a) Diagonal elements of Q after training for Scenario 3

(b) Diagonal elements of Q after training for Scenario 1

Figure 6.16: Diagonal elements of Q after training for Scenarios 3 and 1

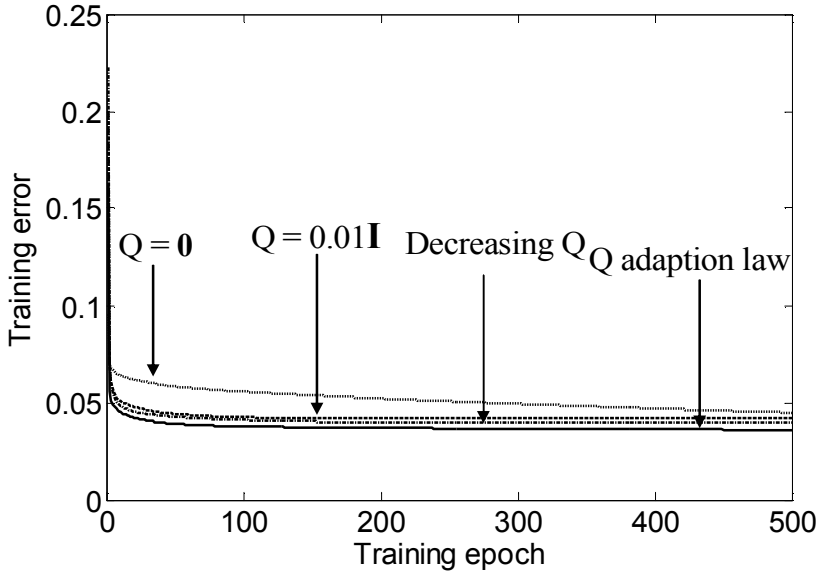


Figure 6.17: Comparison of OptRNN training errors with different Q settings

The four Q configurations are also applied on the optimized RNN. Figure (6.17) shows the results. From these results, the proposed Q adaption law generates the smallest training error compared to other Q control schemes.

Robustness of Neural Networks

In a pervious section, A RNN and an optimized RNN are applied to model the non-linear dynamical benchmark system. The architectures of these two networks are both 5-2-1. Five inputs are cutting speed, feed rate, depth of cut and machining time and a constant bias 1 are used as the inputs. The output of the network is the tool flank wear length.

In this section, the proposed robustness quantification method is applied to quantify the robustness of the trained RNN and the optimized RNN. First local robustness measure for input samples are computed based on Equations (4.61) and the global robustness measure for overall network robustness based on the average of local robustness measures. Generally the perturbation level should be determined based on experimental observations as the hardware/software might have during the implementation of ANN. Here the perturbation level L in Equation (4.51) has been taken as 1% for simplicity and 100 input samples have been used if not mentioned otherwise. Based on the proposed approach a smaller robustness value implies higher system robustness to external perturbations. In the following sections, similar tasks have been conducted as the robustness study of RNN in modeling the benchmark system in Chapter 5.

Corresponding to Figure (5.19), Figure (6.18) shows the varying local robustness measures using 100 sample inputs. Each point represents a local robustness measure for the nth input sample. The global robustness measure is found to be 0.0412 by averaging the local robustness measures. Similar varying local robustness measure tendency has been observed with the optimized RNN, and its global robustness measure is found to be 0.0232, which is smaller than that of RNN.

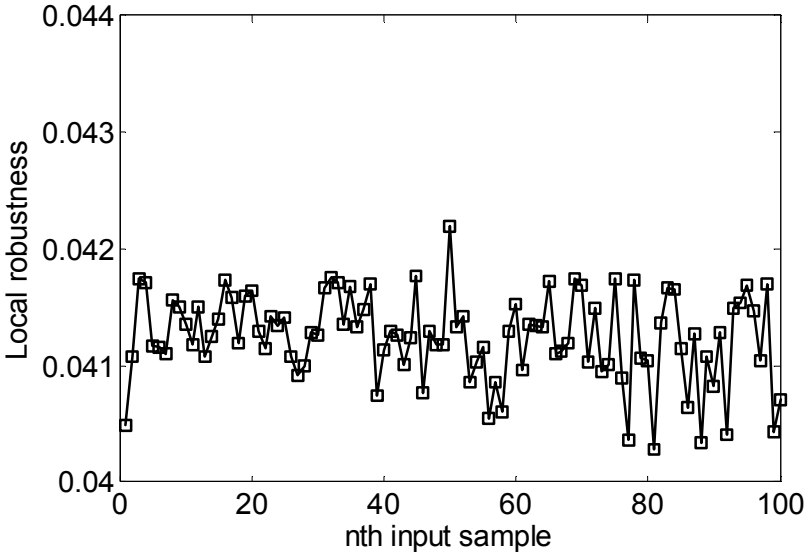


Figure 6.18: Local robustness measures for RNN using 100 input samples

The effects of the number of input samples and the perturbation level to the global robustness measure are studied as follows.

Effect of Number of Input Samples on Global Robustness Measure

In general, the more input samples are used, the more reliable the global robustness measure represents the system robustness performance over the whole input

space. Unfortunately, it is impossible to compute the global measure based on an exhaustive way by sampling all possible inputs. As so, a minimum amount of input samples, which are needed for global robustness quantification, should be determined first. To find this minimum amount in this study, different numbers of input samples have been selected and their corresponding global robustness measures for both RNN and optimized RNN are computed and shown in Figure (6.19).

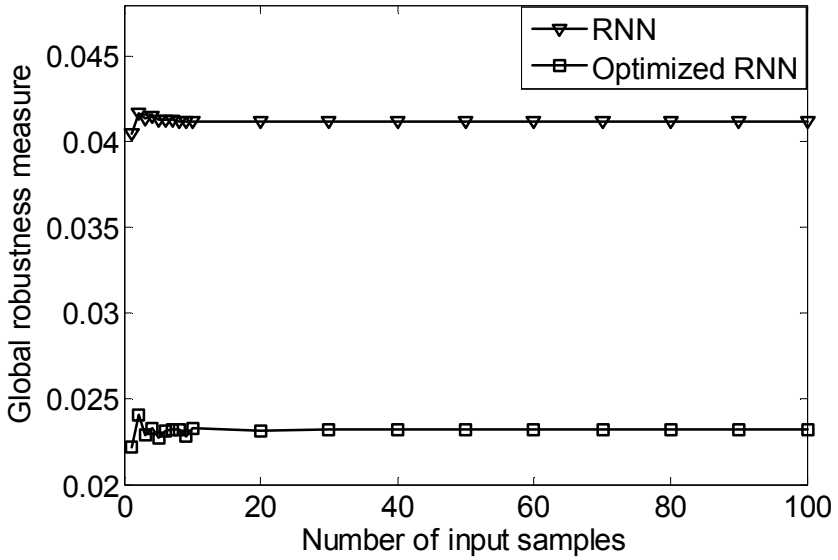


Figure 6.19: Robustness of RNN and optimized RNN (perturbation level = 1%)

It can be seen that for each network the robustness measures converge to a steady value quickly after more than 10 uniformly generated input samples are used. Based on a conservative consideration, 100 input samples are used here and in the following sections. Based on the 100 input samples, optimized RNN has a global robustness value of 0.0232, which is smaller than that of RNN (0.0412), implying that optimized NN is more robust than the regular NN [Kris93].

Effect of the Perturbation Level on Global Robustness Measure

Similar to the robustness study in Chapter five, the perturbation levels ranging from 1% to 20% have been applied to the trained network weights to study the network robustness under different level of perturbed weights; and the results are shown in Figure (6.20). It can be seen from Figure (6.20) that the optimized RNN is more robust than the regular RNN for all the perturbation levels, which indicates the connectivity optimization process has improved the network robustness as observed before [Kris93].

It can also be seen that the relationship between the robustness measure and the perturbation level can be approximated as linear. This linear pattern is again attributed to the network mapping feature discussed in Chapter five.

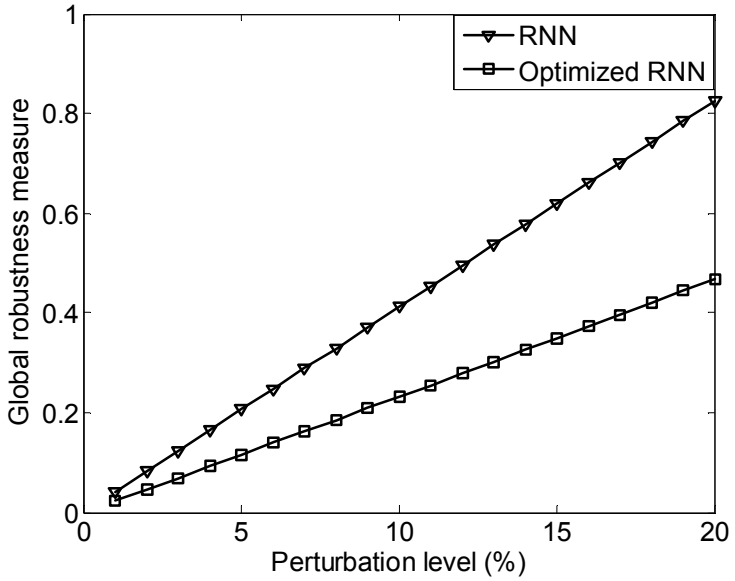


Figure 6.20: Network robustness values under different perturbation levels

Comparison among Robustness Measures

The proposed robustness quantification approach is further compared with the performance loss-based and sensitivity matrix-based approaches. The performance loss-based and sensitivity matrix-based measures are computed based on the training data mentioned in the previous section. To fairly compare the three approaches, the same training data (condition 1, 5, 9, 10, and a) are also used to compute the proposed robustness measure, and the LHS input sampling process is not applied here to generate input samples for the proposed approach.

For both the performance loss-based and the proposed approaches, the 1% perturbation level is used. For the performance loss-based approach, 10,000 networks are generated based on the perturbed weights, and the network output is then compared with the corresponding desired outputs to compute MSE. The resulting maximum MSE is taken as the performance loss-based measure. For the sensitivity matrix-based approach, the sensitivity matrices for all the training inputs are obtained during the training process and the spectral norm [Kris99] for each matrix is computed as the local robustness measure. The average of these norms is used as the sensitivity matrix-based measure.

It should be pointed out that the three robustness measures cannot be directly compared against each other because they are computed using different criteria. Instead, the ratio of robustness measures between RNN and optimized RNN is studied to indicate the effectiveness of any quantification approach. A larger robustness ratio means that this quantification approach is more sensitive in quantifying the robustness difference.

Table 6.6: Comparison of robustness quantification approaches

Robustness quantification approach	RNN robustness (r1)	Optimized RNN robustness (r2)	Robustness ratio (r1/r2)
Performance loss-based	0.1118	0.0954	1.1719
Sensitivity matrix-based	2.3821	2.1164	1.1255
Proposed	0.0412	0.0232	1.7835

Table (6.6) lists the comparison results. For all the three quantification approaches, optimized RNN is found to be more robust than RNN. It is found that the proposed approach (1.78) has the largest robustness ratio than those of the performance loss-based approach (1.17) and the sensitivity matrix-based approach (1.12). As the most sensitive quantification approach is associated with largest robustness ratio value, it is concluded that the proposed robustness quantification approach is the most effective one among these three approaches.

Relationship between Proposed and Sensitivity Matrix-based Approaches

Under a small perturbation level the uncertainty propagation analysis used in this proposed approach can also be related to the sensitivity matrix–based approach. Each element ($H_{ij} = \frac{\partial y_i}{\partial w_j}$) of the weight–output Jacobian sensitivity matrix H represents the derivative of an output (y_i) with respect to a weight (w_j). Using the uncertainty propagation analysis under a small weight perturbation such as 0.01%, this H_{ij} can be approximated by the standard deviation ratio $S_{ij} = \frac{std(y_i^{(j)})}{std(w_j)}$, where std(·) is a standard deviation operator and $std(y_i^{(j)})$ represents the standard deviation of output y_i under a

perturbation with weight w_j. This standard deviation ratio S_{ij} describes the dependence of the output variation on the weight variation. It should be pointed out that different from the proposed robustness quantification approach where perturbations are added to all the weights simultaneously to compute $\vec{\sigma}_{\bar{y}(j)}$, here each time perturbation is only added into a specific weight to compute $std(y_i^{(j)})$ while all the other weights remain the same.

Table (6.7) lists the sensitivity matrix-based robustness measures averaged based on their spectral norms [Kris99], which are computed using the traditional method (H_{ij}) and the uncertainty propagation analysis (S_{ij}), respectively. It is found that the two results quite match each other; therefore the proposed analysis can also be used to compute the sensitivity matrix-based measure.

Table 6.7: Sensitivity matrix-based robustness of RNN and optimized RNN

	RNN robustness	Optimized RNN robustness
Robustness (*H* matrix based)	2.3821	2.1164
Robustness (*S* matrix based)	2.3799	2.1105

Efficiency of the Proposed Robustness Quantification Method

The unscented transform is efficient to quantify the uncertainties in RNN output; to verify that, another uncertainty propagation method using Monte Carlo method is carried out. A set of weight vectors are randomly generated based on the Gaussian distribution of the perturbed weight vector used before, and each of them form a RNN. The inputs are fed into these RNN, and the standard deviation in networks' output are used as the local robustness measure, which further forms the global robustness measure

by averaging. Different numbers of RNN are generated and the computation times are recorded in Table (6.8). It is found that to reach the same robustness result (0.0422), 1500 RNN need to be generated and the computation time is about 32 times of the proposed method, whose corresponding results are shown in Table (6.9). The experiment is carried out on a computer with the configuration of Intel(R) Core(TM) 2 Duo CUP @ 2.8GHz and 3.0 G RAM.

Table 6.8: Robustness results of RNN from a Monte Carlo method

Number of RNN generated	Robustness measure	Computation time (second)
10	0.0413	0.30
100	0.0429	2.65
500	0.0426	13.25
1000	0.0425	26.15
1500	0.0422	38.16

Table 6.9: Robustness results of RNN from the proposed UT-based method

Number of RNN generated	Robustness measure	Computation time (second)
43	0.0422	1.16

It should be pointed out that the robustness measure 0.0422 in Table (6.9) is slightly different from the result 0.0412 in Table (6.6). The difference is due to that different input sets are applied in these two computation processes. In Table (6.6), 100 uniformly generated input samples are used while in Table (6.9) the training input data are applied.

Similar results have been obtained in this tool wear modeling application. The following conclusions can be drawn based on this robustness study:

1) The proposed robustness quantification method is flexible and viable.

2) The proposed method is more efficient than another Monte Carlo simulation based uncertainty propagation analysis based method.
3) The proposed robustness quantification is more effective compared with the other two methods, the performance loss-based and the sensitivity matrix-based approaches.
4) All the three methods show that the optimized RNN is more robust than RNN.

Conclusions

In this chapter, modeling performance, training convergence, and robustness of the developed RNN are studied using a tool wear progression process. The modeling capability for an MLP, a RNN, and an optimized RNN are compared using the training data. It can be seen that all the networks can accurately model the training data. Furthermore, the generalization capability of RNN, the optimized RNN, MLP, FFCNN, and the optimized FFCNN are studied. It is found that the generalization ability of the optimized RNN is the best among these networks. The training convergence is also studied in this case, and the R adaption law is able to make the training process convergent while the Q adaption law can accelerate the training convergence speed. Finally, the robustness of the RNN and the optimized RNN are studied. It is found that the proposed uncertainty propagation based method is more effective, flexible, viable and efficient than other two existing methods in quantifying RNN's robustness.

Some conclusions can be drawn from the case study:

1) The applied RNN structure, training algorithm, and optimization method can make the developed RNN modeling approach more accurate, robust and fast.
2) The proposed adaptive training algorithm can make the training process of RNN more stable and faster.
3) The proposed robustness measures are effective and efficient to quantify the robustness of RNN.

All of the results prove that the developed RNN modeling approach is powerful in terms of accuracy, speed, and stability.

CHAPTER SEVEN

CONCLUSIONS

In this study a RNN is developed based on an FFNN by adding recurrences in its hidden neuron section. The added recurrences are found to be beneficial in modeling non-linear dynamics and hence can improve the network's modeling performance.

Challenges Addressed

This study endeavors to address the following challenges:

Network Structure Determination and Training

How to form an optimal network structure and how to efficiently train networks are common problems in neural network study. A network is defined from two sides, its structure and values of its weights. A structured network is actually a parametric model. The procedure of determining network structure is to find the functional form of the model and the procedure of training is to determine its parameters.

As for the structure, the prototype of RNN is an FFCNN which has three sections of neurons (input neuron section, hidden neuron section and output neuron section), and information flows strictly feedforward in only one direction, from input units to output units. RNN network can be viewed as the summation of a FFCNN and recurrent connections added in its hidden neuron section. The recurrent connections can introduce state feedback into the network structure and improve the network's modeling performance. A network with optimal structure is desirable in training speed, training

accuracy, generalization ability, and robustness. However, the optimal topology of a RNN is case dependent. A pruning method is seamlessly embedded into the training process to optimize the network structure. The optimized RNN is found to be most capable of modeling non-linear dynamical systems.

As for the network training, the EKF is an efficient and powerful state estimation algorithm and applied in the study. The training process can be viewed as a state (weights) estimation problem and the EKF is used to estimate the state from the training patterns. The most time consuming part in EKF training implementation is the formation of the orderly derivatives of network's outputs with respect to weights considering the effect of all contributive connections.

Training Convergence

Training divergence can occur if training parameters are not selected properly. Furthermore, RNN are more vulnerable to training divergence than FFNN due to their recurrent connections. To solve the problem, a parameter R of EKF training algorithm is adapted to guarantee the training convergence while another parameter Q is adapted to accelerate the training. Q is estimated by maximum likelihood method to accelerate training speed and R is adapted by Lyapunov method to ensure training convergence.

Network Robustness Quantification

In addition to training convergence, robustness is another important issue of RNN for its successful implementation. The robustness study considering perturbations in

trained weights is vital for network's implementation. Various networks can be developed for an application. But to select the best fault tolerant network is important for successful application of a RNN. A uncertainty propagation analysis based robustness measures using the unscented transform is proposed.

Methodology Validation and Performance Evaluation

For methodology validation and performance evaluation, two non-linear dynamical systems, a benchmark system and a tool wear progression process in CBN hard turning, are used to verify the modeling performance of the network. From the two case studies, some observations are found as follows:

1) The developed RNN is better than FFNN such as FFCNN and MLP in training accuracy, generalization ability, and training speed.
2) The connectivity optimization can improve a network's performance in terms of training speed, computation cost, and network robustness.
3) The developed R adaption law can guarantee convergence of the training process, while the Q adaption law can speed up RNN training process.
4) The proposed robustness measures have advantages over the other two existing measures developed from the differential analysis and performance loss analysis, respectively.

Contributions

This research can contribute to the current research state of the art as follows:

Develop and Apply RNN in Modeling Applications

A RNN is developed which is better than the commonly used MLP network in modeling applications. This study investigates RNN's modeling performance and compares it with FFNN including FFCNN and MLP. In addition, structure optimization is applied to the networks and forms corresponding optimized networks. Training speed, training accuracy and generalization ability are studied for each network. The results show that RNN surpass FFNN and the connectivity optimization process further improves a network's performance.

Training Convergence Study of RNN

When NN is applied in modeling, training convergence is seldom studied and researchers just train networks without justifying the convergence property first. Training convergence is carried out in this study. Improperly selection of training parameters would result in training divergence, so that the outputs of the network turn to out of bounds. The training convergence include two levels, how to guarantee training convergence and how to accelerate training process. To solve the problem, Lyapunov method is applied to form an R adaption law to guarantee the convergence of training, while maximum likelihood method is used to adapt Q to speed up the training process.

Robustness Study of RNN

An unscented transform based robustness quantification measure for RNN has been developed in this study. The unscented transform is applied in the uncertainty propagation analysis owning to its advantages in efficiency and computation cost saving. Uncertainty propagation analysis is conducted to assess the robustness of the network considering perturbations in network's trained weights. The proposed quantification method is found to have advantages over two other methods.

To summarize, the study proposes the developments of:

1) RNN architecture implementation,
2) the EKF-based training algorithm for RNN,
3) the connectivity optimization algorithm for RNN,
4) the convergence study of developed RNN, and
5) the robustness analysis of RNN.

Future Work

While a thorough study has been carried out for RNN to model non-linear dynamical systems, some work still needs to be carried out which can further improve the study in the future.

To Study the Coupled Effects of Developed Adaption Laws

It should be pointed out that the R adaption law may also affect the training speed. To further optimize the training convergence speed, R needs to be adapted to guarantee

convergence as well as to accelerate the training speed. Future work should mathematically investigate the coupled effects of the two noise parameters on EKF training and further improve the training convergence performance.

To Study the Robustness of RNN in Training Process

In this study the robustness of trained networks is investigated and a method to quantify robustness is proposed. However, evaluation of the robustness of RNN during training is more useful. Combining the robustness study into the training process needs to be carried out in the future. The study includes how to select a training technology and how to select training parameters to improve the network's robustness. To accomplish that, the cost function of robustness for a network which relates robustness to training parameters should be proposed and further studied.

To Study the Robustness of RNN Due to Architecture Variation

Current study only considers robustness of network due to perturbation in trained weights, although the developed method can also apply to assessing robustness due to perturbation in inputs. However, consideration of perturbation in architecture level, for example missing/fault of neuron and connections is also another important aspect of robustness study which needs to be conducted in the future. This problem is correlated with the above one, because the network's structure is determined through the connectivity optimization process.

REFFERENCES

A. M. Abrao, M. L. H. Wise, and D. K. Aspinwall, "Tool Life and Workpiece Surface Integrity Evaluations when Machining Hardened AISI 52100 Steels with Conventional Ceramic and PCBN Tool Materials," *SME Technical Paper*, MR95-159, pp.1-9, 1995.

A. Alessandri, M. Cuneo, S. Pagnan, and M. Sanguineti, "On the Convergence of EKF-based Parameters Optimization for Neural Networks," *42nd IEEE Conference on Decision and Control*, pp. 6181–6186, 2003.

C. Alippi and M. Milena, "A Poly-time Analysis of Robustness in Feedforward Neural Networks," *IEEE International Workshop on Virtual and Intelligent Measurement Systmes*, pp. 76-80, Budapest, Hungary, May 19-20, 2001.

C. Alippi, "Selecting Accurate, Robust, and Minimal Feedforward Neural Networks," *IEEE Trans on Circuits and Systems -I: Fundamental Theory and Applications*, Vol. 49(12), pp. 1799-1810, 2002.

C. Alippi, D. Sam, and F. Scotti, "A Training-time Analysis of Robustness in Feed-Forward Neural Networks," *2004 IEEE International Joint Conference on Neural Networks*, Vol. 4, pp. 2853- 2858, 2004.

D. L. Alspach, "A Parallel Filtering Algorithm for Linear Systems with Unknown Time Varying Statistics," IEEE *Trans. Automatic. Control.*, Vol. AC-19, pp. 552-556, 1974.

S. Arik, "Global Robust Stability of Delayed Neural Networks," *IEEE Trans. on Circuits and Systems—I: Fundamental Theory and Applications*, Vol. 50(1), pp. 156-160, 2003.

A. Assoum, M. Geagea, and R. Velazco, "Influence on ANNs Fault Tolerance of Binary Errors Introduced during Training," *International Conference on Information and Communication Technologies: From Theory to Applications*, pp. 435-436, 2004.

A. F. Atiya and A. G. Parlos, "New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence," *IEEE Transactions on Neural Networks,* Vol. 11(3), pp. 697-709, 2000.

J. K. Baksalary and R. Kala, "A New Bound for the Euclidean Norm of the Difference between the Least Squares and the Best Linear Unbiased Estimators," *The Annals of Statistics*, Vol. 3, pp. 679-681, 1980.

K.D. Boese and A.B. Kahng, “Simulated Annealing of Neural Networks: The "Cooling" Strategy Reconsidered,” *IEEE International Symposium on Circuits and Systems*, Vol. 4, pp. 2572-2575, 1993.

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 Part I*, Vol. 50, pp. 34-44, 2003.

C. Chiu, K. Mehrotra, C. K. Mohan, and S. Rankat, “Robustness of Feedforward Neural Networks,” *IEEE International Conference on Neural Networks,* Vol.2, pp. 783-788, 1993.

M. Chow and J. Teeter, “Analysis of Weight Decay as A Methodology of Reducing Three-Layer Feedforward Artificial Neural Networks for Classification Problems,” *IEEE International Conference on Neural Networks - Conference Proceedings, Part 1 (of 7)*, pp. 600-605, Orlando, FL, 27th-29th June, 1994.

G. Chryssoluouris and M. Guillot, “A Comparison of Statistical and AI Approaches to the Selection of Process Parameters in Intelligent Machining,” *ASME, Journal of Engineering for Industry*, Vol. 112, pp. 122-131, 1990.

S. Das, A. B. Chattopadhyay and A. S. R. Murthy, “Force Parameters for On-line Tool Wear Estimation: A Neural Network Approach,” *Neural Networks*, Vol. 9, pp. 1639-1645, 1996.

T. Dawson, *Machining Hardened Steel with Polycrystalline Cubic Boron Nitride Cutting Tools*, Ph.D. Thesis, GIT, Atlanta, GA, 2002.

R. C. Dewes and D. K. Aspinwall, “The Use of High Speed Machining for the Manufacture of Hardened Steel Dies,” *Trans. of NAMRI*, Vol. 24, pp. 21-26, 1996.

D. E. Sr. Dimla and P. M. Lister, “On-Line Metal Cutting Tool Condition Monitoring. II: Tool-State Classification Using Multi-layer Perceptron Neural Networks,” *International Journal of Machine Tools & Manufacture*, Vol. 40, pp. 769-781, 2000.

J. Driver, S. R. Baker, and D. McCallum, *Residential Exposure Assessment: A Sourcebook,* Springer, 2000.

G. Dundar and K. Rose, “The Effects of Quantization on Multilayer Neural Networks,” *IEEE Trans on Neural Networks*, Vol. 6(6), pp. 1446-1451.

R. Eickhoff and U. Ruckert, “Robustness of Radial Basis Functions,” *Neurocomputing*, Vol. 70, pp. 2758–2767, 2007.

J. L. Elman, "Finding Structure in Time," *Cognitive Science*, Vol. 14(2), pp. 179-211, 1990.

E. O. Ezugwu, S. J. Arthur, and E. L. Hines, "Tool-wear Prediction Using Artificial Neural Networks," *Journal of Materials Processing Technology*, Vol. 49, pp. 255-264, 1995.

S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," Advances *in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), San Mateo, CA: Morgan Kaufmann, pp. 524- 532, 1990.

Zhaoshu Feng and Anthony N. Michel, "Robustness Analysis of a Class of Discrete-Time Recurrent Neural Networks under Perturbations," *IEEE Trans on Circuits and Systems-I: Fundamental Theory and Applications,* Vol. 46, No. 46, pp. 1482-1486, 1999.

R. Fitzgerald, "Divergence of the Kalman Filter," *IEEE Trans on Automatic Control,* Vol. 16(6), pp. 736-747, 1971.

R. E. Haber, A. Alique, "Intelligent Process Supervision for Predicting Tool Wear in Machining Processes," *Mechatronics*, Vol. 13, pp. 825-849, 2003.

A. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*, Cambridge University Press, New York, 1989.

B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal Brain Surgeon and General Network Pruning," *IEEE International Conference on Neural Networks*, San Francisco, CA, Mar 28-April 1, 1993, pp. 293-299, 1993.

S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Prentice-Hall, Upper Saddle River, NJ, 1999.

J. C. Helton and F. J. Davis, "Latin Hypercube Sampling and the Propagation of Uncertainty in Analyses of Complex Systems," *Rliab. Eng. Syst. Safety*, Vol. 81(1), pp.23-69, 2003.

T. Hofling and R. Isermann, "Fault Detection Based on Adaptive Parity Equations and Single-Parameter Tracking," *Control Engineering Practice*, Vol. 4, pp. 1361–1369, 1996.

K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, pp. 359-366, 1989.

R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge Univ. Press, 1990.

Y. Huang, *Predictive Modeling of Tool Wear Rate with Application to CBN Hard Turning*, Ph.D. Thesis, Georgia Institute of Technology, GA, 2002.

Y. Huang and S. Y. Liang, "Modeling of CBN tool flank wear progression in finish hard turning," *ASME J. of Manufacturing Science and Engineering*, Vol. 126, pp. 98-106, 2004.

R. Iserman, "Process Fault Detection Based on Modeling and Estimation Methods — A Survey," *Automatica*, Vol. 20, pp. 387–404, 1984.

M. I. Jordan, "Attractor Dynamics and Parallelism in A Connectionist Sequential Machine," *Proceeding of the 1986 Connitive Science Conference*, pp. 531-546, 1986.

S. Julier and J. K. Uhlmann, 1997, "A New Extension of the Kalman Filter to Non-linear Systems", *Proc of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Control*, Florida, 1997.

D. Jwo and T. Cho, "A Practical Note on Evaluating Kalman Filter Performance Optimality and Degradation," *Applied Mathematics and Computation*, Vol. 193(2), pp. 482-505, 2007.

S. M. Kay, Fundamentals *of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993.

N. Kwak, "Principal Component Analysis Based on L1-Norm Maximization," *IEEE Trans on Pattern Analysis and Machine Intelligence*, Vol. 30(9), pp. 1672-1680, 2008.

H. K. Khalil, *Nonlinear Systems*, Prentice Hell, New Jersey, 2002.

K. KrishnaKumar, "Optimization of the Neural Net Connectivity Pattern Using A Backpropagation Algorithm," *Neurocomputing*, Vol. 5, pp. 273-286, 1993.

K. KrishnaKumar and K. Nishta, "Robustness Analysis of Nueral Networks with An Application to System Identification," *Journal of Guidance, Control and Dynamics*, Vol. 22, pp. 695-701, 1999.

R. J. Kuo and P. H. Cohen, "Intelligent Tool Wear Estimation System through Artificial Neural Networks and Fuzzy Modeling," *Artificial Intelligence in Engineering*, Vol. 12, pp. 229-242, 1998.

R. J. Kuo and P. H. Cohen, "Multi-sensor Integration for On-line Tool Wear Estimation through Radial Basis Function Networks and Fuzzy Neural Network," *Neural Networks*, Vol. 12, pp. 355-370, 1999.

C. S. Leung and L. W. Chan, “Dual extended Kalman filtering in recurrent neural networks source,” *Neural Networks*, Vol. 16(2), pp. 223–239, 2003.

X. Li, L. Huang, and J. Wu, “A New Method of Lyapunov Functionals for Delayed Cellular Neural Networks,” IEEE *Trans. Circuits Syst. I,* Vol. 51(11), pp. 2263–2270, 2004.

X. B. Liang and J. Wang, “An Additive Diagonal Stability Condition for Absolute Exponential Stability of a General Class of Neural Networks,” *IEEE Trans on Circuits and Systems Part I,* Vol. 48, pp. 1308-1317, 2001.

Y. Liguni, H. Sakai, and H. Tokumaru, “A Real-time Learning Algorithm for a Multilayered Neural Network Based on the Extended Kalman Filter,” *IEEE Trans. Signal Process*, Vol. 40(4), pp. 59–966, 1992.

D. Linkens and Y. Nyongesa, “Learning Systems in Intelligent Control: An Appraisal of Fuzzy, Neural and Genetic Algorithm Control Applications”, *IEE Proc. Control Theory App*., Vol. 134 (4), pp. 367-385, 1996.

D. Liu and A. N. Michel, “Robustness Analysis of a Class of Neural Networks,” Circuits and Systems, 1993., *Proceedings of the 36th Midwest Symposium*, pp. 1077-1080, 1993.

D. Liu and A. N. Michel, “Robustness Analysis and Design of a Class of Neural Networks with Sparse Interconnecting Structure,” *Neurocomputing*, Vol. 12, pp. 59-76, 1996.

Q. Liu and Y. Altintas, “On-line Monitoring of Flank Wear in Turning with Multilayered Feed-forward Neural Network,” *International Journal of Machine Tools & Manufacture*, Vol. 39, pp.1945-1959, 1999.

D. Liu, S. Hu, and J. Wang, “Global Output Convergence of a Class of Continuous-time Recurrent Neural Networks with Time-varying Thresholds,” *IEEE Trans. Circuits Syst. II*, Vol. 51(4), pp. 161–167, 2004.

W. Liu, L. Yang, and L. Hanzo, “Recurrent Neural Network based Narrowband Channel Prediction,” *IEEE 63rd Vehicular Technology Conference*, Vol. 5, pp. 2173-2177, 2006.

J. T. Lo, “Synthetic Approach to Optimal Filtering,” *IEEE Trans on Neural Networks,* Vol. 5(5), pp. 803-811, 1994.

W. L. Loh, "On Latin Hypercube Sampling," *Annals of Statistics*, Vol. 24(5), pp. 2058-2080, 1996.

L. Luo, C. Guo, G. Ma, and A. Ji, “Choice of Optimum Model Parameters in Artificial Neural Networks and Application to X-ray Fluorescence Analysis,” *X-Ray Spectrometry*, Vol. 26, pp. 15-22, 1997.

D. Mandic and J. Chambers, *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*, Wiley, 2001.

M. B. Matthews, “Neural Network Nonlinear Adaptive Filtering Using the Extended Kalman Filter Algorithm,” *in Proceedings of the International Neural Networks Conference*, Vol. 1, pp. 115-119, Paris, 1990.

P. S. Maybeck, *Stochastic Models, Estimation, and Control*, Vol. 2, Academic Press, 1982.

P. S. Maybeck, *The Kalman Filter: An Introduction to Concepts in Autonomous Robot Vehicles*, I. J. Cox, G. T. Wilfong (eds), Springer-Verlag, 1990.

W. S. McCulloch and P. Walter, 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* Vol 5, pp 115-133, 1943.

L. Medsker and L. C. Jain, *Recurrent Neural Networks: Design and Applications*, CRC, 1999.

R. K. Mehra, “On the Identification of Variance and Adaptive Kalman Filtering,” *IEEE Trans. On Automatic Control*, Vol. 15(2), pp. 175-184, 1970.

R. K. Mehra, “Approaches to Adaptive Filtering,” *IEEE Trans. Automatic. Control*, Vol. 17(5), pp. 693-698, 1972.

J. E. Moody, S. J. Hanson, and R. P. Lippmann, "The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems," *Advances in Neural Information Processing Systems*, Vol. 4, pp. 847-854, 1992.

K. S. Narendra, *Adaptive Control of Dynamical System Using Neural Networks, Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches,* New York, NY: Van Nostrand Reinhold, 1992.

T. Ozel and Y. Karpat, “Predictive Modeling of Surface Roughness and Tool Wear in Hard Turning Using Regression and Neural Networks,” *International Journal of Machine Tools & Manufacture*, Vol. 45, pp. 467-479, 2005.

D. S. Phatak and I. Koren, “Complete and Partial Fault Tolerance of Feedforward Neural Nets,” *IEEE Trans. Neural Networks*, Vol.6 (2), pp. 446–456, 1995.

S. G. Pierce, Y. B. Haim, K. Worden, and G. Manson, "Evaluation of Neural Network Robust Reliability Using Information-Gap Theory," *IEEE Trans on Neural Neteorks*, Vol. 17(6), pp. 1349-1361, 2006.

D. Psaltis, A. Sideris, and A. Yamamura, "A Multilayered Neural Network Controller," *IEEE Control Systems Magazine*, Vol. 8(2), pp. 17-21, 1988.

G. V. Puskorius and L.A. Feldkamp, "Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter Trained Recurrent Networks," *IEEE Trans. on Neural Networks*, Vol. 5, pp. 279-297, 1994.

T. Ragg, H. Braun, and H. Landsberg, "A Comparative Study of Neural Network Optimization Techniques," *13th International Conference on Machine Learning: Workshop Proceedings on Evolutionary Computing and Machine Learning*, pp. 111-118, 1996.

B. Razavi, *Design of Analog CMOS Integrated Circuits*, McGraw-Hill, New York, 2000.

F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, Vol. 65(6), pp. 386-408, 1958.

J. D. J. Rubio and W. Yu, "Nonlinear System Identification with Recurrent Neural Networks and Dead-zone Kalman Filter Algorithm," *Neurocomputing*, Vol. 70, pp. 2460–2466, 2007.

D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations, Cambridge, MA: MIT Press, 1986.

S. Samarasinghe, *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*, Auerbach Publications, 2006.

J. Santos and R. J. Duro, "Evolutionary generation and training of recurrent artificial neural networks," *Proceedings of the First IEEE Conference on Evolutionary Computation*, Vol. 2, pp. 759-763, 1994.

S. Sastry, *Nonlinear Systems: Analysis, Stability, and Control*, Springer, 1999.

R. J. Schalkoff, *Artificial Neural Networks*, McGraw-Hill Inc., New York, 1997.

C. Scheffer, H. Kratz, P. S. Heyns, and F. Klocke, "Development of A Tool Wear-monitoring System for Hard Turning," *International Journal of Machine Tools & Manufacture*, Vol. 43, pp. 973-985, 2003.

F. H. Schlee, C. J. Standish, and N. F. Toda, "Divergence in the Kalman Filter," *AIAA Journal*, Vol. 5, pp. 1l14-1120, 1967.

U. Seiffert, "Multiple Layer Perceptron Training Using Genetic Algorithms," *European Symposium on Artificial Neural Networks ESANN*, pp. 159-164, 2001.

R. Setiono and W. K. Leow, "FERNN: an Algorithm for Fast Extraction of Rules from Neural Networks, *Applied Intelligence*," Vol. 12, pp. 15-25, 2001.

D. Simon, *Optimal State Estimation*, John Wiley & Sons, New Jersey, 2006.

S. Singhal and L. Wu, "Training Feed forward Networks with Extended Kalman Filter Algorithm," *Proceedings - ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1187-1190, Glasgow, Scotland, 23th-26th May, 1989.

Q. Song, Y. Wu, and Y. C. Soh, "Robust Adaptive Gradient-Descent Training Algorithm for Recurrent Neural Networks in Discrete Time Domain," *IEEE Trans on Neural Networks*, Vol. 19(11), pp. 1841-1853, 2008.

J. F. Swidzinski and C. Kai, "Nonlinear Statistical Modeling and Yield Estimation Technique for Use in Monte Carlo Simulations," *IEEE Trans on Microwave Theory and Techniques*, Vol. 48(12), pp.2316-2324, 2000.

S. Takatsu, H. Shimoda, and K. Otani, "Effect of CBN Content on the Cutting Performance of Polycrystalline CBN Tools," *International Journal of Refract. Hard Mat.*, Vol. 2(4), pp. 175-178, 1983.

H. Tang, K. C. Tan, and Z. Yi, *Neural Networks: Computational Models and Applications*, Springer, 2007.

E. A. Wan and R. V. D. Merwe, *Kalman Filtering and Neural Networks, Chapter 7 : The Unscented Kalman Filter*, Wiley Publishing, Eds. S. Haykin, 2001.

X. Wang, W. Wang, Y. Huang, N. Nguyen, and K. Krishnakumar, "Design of Neural Network-based Estimator for Tool Wear Modeling in Hard Turning," *Journal of Intelligent Manufacturing*, Vol. 19(4), pp. 383-396, 2008.

X. Wang, Y. Huang, N. Nguyen, and K. Krishnakumar, "CBN Tool Flank Wear Modeling Using Hybrid Neural Network," *International Journal of Mechatronics and Manufacturing Systems*, Vol. 1(1), pp. 83-102, 2008.

X. Wang and Y. Huang, "Optimized Recurrent Neural Network-Based Tool Wear Modeling in Hard Turning," *Transactions of NAMRI/SME*, Vol. 37, pp. 213-220, 2009.

P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in The Behavioral Sciences*, Harvard University, Unpublished doctoral dissertation, 1974.

P. J. Werbos, "Back propagation through time: what it does and how to do it," *Proc. of the IEEE*, Vol. 78(10), pp. 1550-1560, 1990.

B. Widrow and J. Kolla, *Quantization Noise*, Prentice-Hall, NJ, USA, 2002.

R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running fully Recurrent Neural Networks," *Neural Computation*, Vol. 1 , pp. 270-280, 1989.

L. J. Xie, J. Schmidt, J. C. Schmidt, and F. Biesinger, "2D FEM Estimate of Tool Wear in Turning Operation," *Wear*, Vol. 258, pp. 1479-1490.

S. O. Yee and M. Y. Chow, "Robustness of an Induction Motor Incipient Fault Detector Neural Network Subject to Small Input Perturbations," *IEEE Proceedings of Southeastcon '91*, Vol. 1, pp. 365-369, 1991.

Z. Yi, C. Jian, and L. Zhang, 2006, "Output Convergence Analysis for a Class of Delayed Recurrent Neural Networks with Time-Varying Inputs," *IEEE Trans on Systems, Man, and Cybernetics-Part B: Cybernetics*, Vol. 36(1), pp. 87-95, 2006.

P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering: A Practical Approach*, AIAA, 2001.

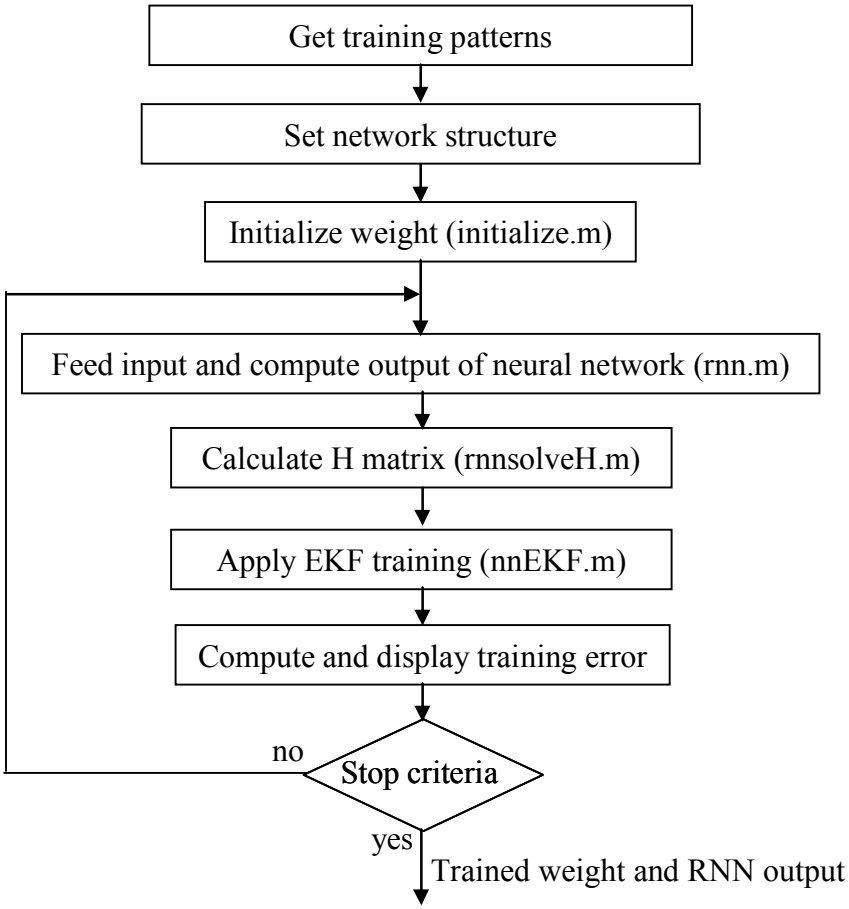
L. Zhang, "Neural Network-based Market Clearing Price Prediction and Confidence Interval Estimation with an Improved Extended Kalman Filter Method," *IEEE Trans on Power Systems*, Vol. 20 (1), pp. 59-66, 2005.

Z. Zhang, J. C. Lv, and L. Zhang, "Output Convergence Analysis for a Class of Delayed Recurrent Neural Networks with Time-varying Inputs," *IEEE Trans on Systems, Man, and Cybernetics—Part B: Cybernetics*, Vol. 36(1), pp. 87-95, 2006.

APPENDICES

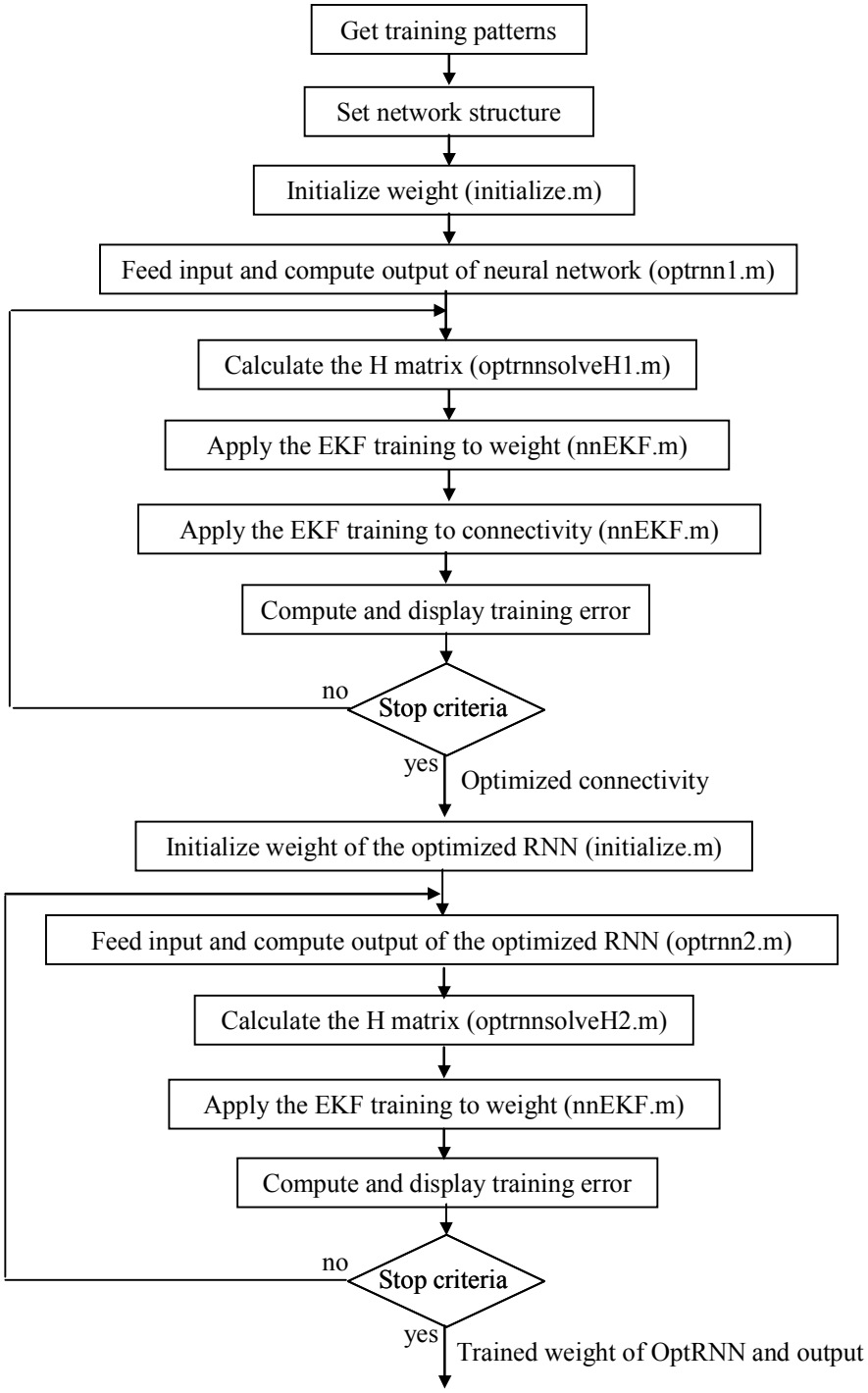
Appendix A

Program to train RNN - trainrnn.m



Appendix B

Program to optimize RNN and train OptRNN - trainoptrnn.m



Appendix C

Program to apply R and Q adaption laws in training RNN – trainrnn_adrq.m