INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

A Macromodeling Approach for Nonlinear Microwave/RF Circuits and Devices Based on Recurrent Neural Networks

by

Yonghua Fang, B. Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirement for the degree of

Master of Applied Science

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Electronics**

**Carleton University
Ottawa, Ontario, Canada K1S 5B6**

© Copyright 2001, Yonghua Fang

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-66889-4

Canada

The undersigned recommend to the Faculty of Graduate Studies and Research acceptance
of the thesis

**A Macromodeling Approach for Nonlinear
Microwave/RF Circuits and Devices Based on
Recurrent Neural Networks**

submitted by

Yonghua Fang, B. Eng.

in partial fulfillment of the requirements for the degree of Master of Applied Science

Professor Qi-Jun Zhang

Thesis Supervisor

Professor Michael Nakhla

Chair, Department of Electronics

Carleton University

Abstract

Recently, neural networks have been used in the microwave/RF area for fast and flexible modeling of devices/circuits. Macromodeling nonlinear devices and circuits is an important research topic in this area. In the thesis, a new macromodeling approach is developed in which Recurrent Neural Networks (RNNs) are used to model the time-domain input-output relationships. The task of constructing a macromodel is to determine the weights inside of RNN such that it will have a similar input-output relationship of the original nonlinear circuit. This process is accomplished by neural network training. Discrete input and output waveforms of the original nonlinear circuit are used as training data. A training scheme employing the gradient-based l_2 optimization technique based on Back Propagation Through Time (BPTT) is implemented to train the RNN.

The proposed technique is demonstrated through macromodeling three practical nonlinear microwave circuits and devices, namely, a power amplifier, a mixer and a MOSFET.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Qi-Jun Zhang, for his professional guidance, invaluable inspiration, continuous financial support and patience throughout the research work and the preparation of this thesis. His constant striving for rigorous research and profound knowledge will influence and benefit me for the rest of my life.

I am also grateful to Peggy, Lorena, Jacques and all other staff and faculty for providing the excellent lab facilities and friendly environment for study and research.

Jianjun, Vijay, Mustapha and other students in my group are thanked for the help, friendship and enjoyable collective working.

My special thank is given to Fang Wang and Xin Xu, who have generously given me numerous help and encourage in the life.

Finally, I would like to thank my parents and other family members back in China, their love, support and encourage have been and will be the source of strength for any difficulty and success in my life.

Contents

Chapter 1 Introduction.....	1
<i>1.1 Motivations</i>	<i>1</i>
<i>1.2 Thesis Contributions</i>	<i>3</i>
<i>1.3 Outline of the Thesis</i>	<i>4</i>
Chapter 2 Literature Review	6
<i>2.1 Application of Neural Network in RF/Microwave Area</i>	<i>6</i>
<i>2.2 Neural Network Structures</i>	<i>8</i>
2.2.1 Multilayer Perceptron Networks.....	8
2.2.2 RBF and Wavelet Neural Networks.....	10
2.2.3 Knowledge Based Neural Networks.....	14
<i>2.3 Training Algorithms.....</i>	<i>18</i>
2.3.1 Training Objective	18
2.3.2 Review of Back Propagation Algorithm.....	18
2.3.3 Gradient-Based Optimization Methods	21
2.3.4 Global Optimization Methods.....	29
<i>2.4 Review of Approaches of Macromodeling Nonlinear Microwave Circuits</i>	<i>30</i>
2.4.1 Behavioral Modeling Technique.....	30

2.4.2	Equivalent Circuit Based Approach.....	32
2.4.3	Time Domain S-parameter Technique.....	34
2.4.4	Krylov-subspace Based Technique.....	36
2.5	Conclusion	37
 Chapter 3 Proposed RNN Macromodeling Approach and Back Propagation		
	Through Time	38
3.1	<i>Formulation of Circuit Dynamics.....</i>	<i>38</i>
3.2	<i>Formulation of Macromodeling.....</i>	<i>40</i>
3.3	<i>Selection of RNN Structure and Training Scheme.....</i>	<i>41</i>
3.3.1	Structure: State Based vs. Input/output Based.....	41
3.3.2	Training Scheme: Series-Parallel vs. Parallel	43
3.4	<i>Detailed Formulation of The RNN Macromodel Structure</i>	<i>44</i>
3.5	<i>Model Development and RNN Training</i>	<i>48</i>
3.5.1	Objective of RNN Training.....	48
3.5.2	Back Propagation Through Time (BPTT)	49
3.6	<i>Summary and Discussion.....</i>	<i>51</i>
 Chapter 4 RNN Macromodeling Examples.....54		
4.1	<i>RFIC Power Amplifier Macromodeling Example.....</i>	<i>54</i>
4.2	<i>RF Mixer Macromodeling Example.....</i>	<i>61</i>
4.3	<i>MOSFET Time Domain Macromodeling Example.....</i>	<i>68</i>

4.4 Comparison between Standard Neural Network and The Proposed RNN Methods for Nonlinear Circuit Macromodeling.....	77
Chapter 5 Conclusions and Future Research.....	79
5.1 Conclusions.....	79
5.2 Suggestions for Future Directions	80
Appendix A Developing RNN Macromodels Using <i>NeuroModeler</i>[®] Software	
Package.....	82
A.1 The 'type' file	83
A.2 The 'structure' file	84
A.3 The 'data' file.....	86
A.4 An illustrating example.....	87
References	91

List of Figures

Figure 2.1: Knowledge Based Neural Network (KBNN) structure.....	17
Figure 2.2: Measurable amplifier parameters for behavioral modeling	31
Figure 2.3: Time Domain scattering functions model of a two port nonlinear circuit.....	35
Figure 3.1: Illustration of state-based RNN structure.....	42
Figure 3.2: Illustration of input/output based RNN structure.....	42
Figure 3.3: RNN training schemes, (a) Series-parallel scheme (b) Parallel scheme	45
Figure 3.4: The proposed RNN based macromodel structure.....	47
Figure 4.1: Power amplifier circuit to be represented by a RNN macromodel.	55
Figure 4.2: The structure of the RNN macromodel for power amplifier.....	56
Figure 4.3: Comparison between output waveforms from original amplifier (o) and that from a RNN macromodel with 3 buffers (-) trained in transient state. Good agreement is achieved even though these waveforms have never been used in training.....	59
Figure 4.4: Comparison between output waveforms from original amplifier (o) and that from a RNN macromodel with 3 buffers (-) trained in steady state. Good agreement is achieved even though these waveforms have never been used in training.....	60

Figure 4.5: Mixer circuit to be modeled by a RNN macromodel.....	62
Figure 4.6: The structure of the RNN macromodel for a mixer	63
Figure 4.7: Comparison between output waveforms from original mixer (o) and that from a RNN macromodel with 3 buffers (-) trained in transient state. Good agreement is achieved even though these waveforms have not been used in training. ($f_{RF} = 2.55$ GHz, $P_{RF} = -48$ dBm, $Z_{IF} = 50 \Omega$).....	65
Figure 4.8: Comparison between output waveforms from original mixer (o) and that from a RNN macromodel with 3 buffers (-) trained in steady state. Good agreement is achieved even though these waveforms have not been used in training. ($f_{RF} = 2.05$ GHz, $P_{RF} = -47$ dBm, $Z_{IF} = 35 \Omega$). This is one of the many test waveforms used.....	67
Figure 4.9: The circuit used to generate training data for a BSIM3-level49 transistor to be represented by a RNN macromodel.....	69
Figure 4.10: The structure of the RNN macromodel for a BSIM3-level49transistor.....	69
Figure 4.11: Effect of initial state estimation and comparison between output waveforms from original transistor simulation (o) and that from a RNN macromodel with 2 buffers (-) for $f = 0.9$ GHz, $V_{source} = 1.55$ Volt, V_g $= -2.125$ volts and $V_d = -3.25$ volts.....	72
Figure 4.12: Comparison between output waveforms of gate current from original transistor (o) and that from a RNN macromodel with 2 buffers (-) under various excitation with different parameters. Initial estimation by an initial RNN was used. The macromodel matches test data very well even though those various test waveforms have never been used in training.....	74

Figure 4.13: Comparison between output waveforms of drain current from original transistor (o) and that from a RNN macromodel with 2 buffers(-) under various excitation with different parameters. Initial estimation by an initial RNN was used. The macromodel matches test data very well even though those various test waveforms have never been used in training..... 76

List of Tables

Table 3.1 Comparison between FFNN technique and RNN technique	52
Table 4.1 Amplifier: Recurrent training and testing vs. different number of hidden neurons	57
Table 4.2 Amplifier: Comparison of recurrent model against different number of buffers.....	57
Table 4.3 Mixer: Recurrent training and testing vs. different number of hidden neurons	64
Table 4.4 Mixer: Comparison of recurrent model against different number of buffers.....	64
Table 4.5 Comparison of Test Errors Between Non-Recurrent and the Proposed Recurrent Models for Nonlinear Modeling	78
Table A.1: The format of the ‘type’ file for an input/output RNN	83
Table A.2: The format of the ‘structure’ file for an input/output RNN	85
Table A.3: Format of RNN ‘data’ file	87
Table A.4: An example RNN macromodel ‘structure’ file for power amplifier	88
Table A.5: An exmaple RNN macromodel structure for power amplifier (continued).....	89
Table A.6: An example RNN macromodel ‘type’ file for power amplifier	90

Chapter 1

Introduction

1.1 Motivations

The rapid growth of telecommunication and wireless industry in the recent years has resulted in a drive for continuous improvement of the performance of Radio Frequency (RF) and microwave circuits. Effective use of computer aided design (CAD) software has been a crucial step in designing RF and microwave circuits and systems.

The desire to have lower cost circuits and systems that offer more functionality led to shrinking of devices as well as increased complexities. Systems have larger number of components and the interactions between various components become more complicated. Numerically solving these circuits involves computationally intensive Newton-Raphson iterations and matrix inversions. The circuit simulation could be much slower and more inaccurate. On the other hand, the emphasis on time-to-market increased the use of virtual computer design instead of physical hardware prototyping. Using CAD tools, behaviors of the circuits and systems such as performance, manufacturability, electrical and physical reliability, can be predicted and optimized beforehand. However, these advanced design tools require highly repetitive simulations of the original nonlinear

circuit, for example statistical analysis and yield optimization. All these issues make the computer-aided design computationally prohibitive.

By nature, a large-scale system can usually be divided into several sub-modules. In most applications, these sub-modules are standard function blocks and the same sub-module could be reused many times in the system. Utilizing this fact, instead of putting all the components in the complicated system and performing simulation, a better practice is to first build models for each sub-module and subsequently simulate the overall system using these models. The process of macromodeling is to build models that can accurately characterize the external behaviors of the sub-modules. And the resulting macromodels are faster in evaluation as compared to original nonlinear circuit simulation. Equivalent circuit techniques, frequency domain input-output mapping techniques, and table look-up combined with curve fitting techniques have all been used to construct macromodels. For nonlinear microwave circuits, these techniques may not be sufficient to model the nonlinear dynamic characteristics. Trying to find an approach that can build fast and accurate macromodels for nonlinear microwave circuits is a primary motivation of this thesis.

Artificial neural networks (ANN) are information-processing networks inspired by the human brain's ability of learning and generalizing. Neural networks have been successfully applied to many complicated tasks in areas such as control, telecommunications, biomedicine, remote sensing, pattern recognition, manufacturing etc. In the microwave area, neural networks have been used to model a wide variety of circuit components and the developed neural models have been used in circuit design and optimization. Neural models can be developed through a computer-driven optimization

process called training. The neural models can learn and abstract the device/circuit characteristics from the measured/simulated training data (samples) without any prior knowledge of the application. Theoretically, neural networks are capable of mapping arbitrary continuous nonlinear relationships between inputs and outputs. On the other hand, neural models are very fast because the structure of neural network is simple and the evaluation only involves several basic arithmetic operations (multiplication and additions).

These advantages of neural networks have been exploited in many applications of RF/microwave design including passive components and active device modeling. However, all these applications of artificial neural networks only map static input/output relationships. In order to characterize the dynamic analog behavior of nonlinear circuits, where time domain information needs to be used, conventional feed forward neural networks are not sufficient. Examining suitable neural network structures and algorithms in order to extend the existing neural modeling techniques to dynamic nonlinear circuit modeling is also a key motivation of this thesis.

1.2 Thesis Contributions

The objective of this thesis is to develop a new time-domain macromodeling approach for nonlinear microwave circuits using neural network techniques. The following are the contributions from this thesis work:

- A new macromodeling approach for nonlinear RF/Microwave circuits is developed using recurrent neural networks (RNN). The validity of the proposed new approach is demonstrated by several practical circuit and device modeling examples [1].

- As an integral part of the model structure, the RNN is introduced. A detailed formulation of the RNN and its mechanism of modeling the dynamic behavior of nonlinear microwave circuits are described.
- To train RNN macromodels, Back Propagation Through Time (BPTT) scheme is implemented. The BPTT scheme incorporates the accumulation effect to each weight in the neural network that is caused by the feedback connections. Using the gradient information resulted from BPTT, l_2 based optimization methods can be applied to RNN training.
- An object-oriented software module programmed in Java and C++ is developed and integrated into *NeuroModeler*[®], software for building neural models for microwave applications. This module makes the utilities in *NeuroModeler*[®] applicable to RNN macromodels, such as training and testing.

1.3 Outline of the Thesis

The thesis is organized in the following way.

In chapter 2, a literature review is presented. First, a review of various neural network structures, training methods and their applications to RF/microwave design area are conducted. After that, existing macromodeling approaches of RF/Microwave nonlinear circuits and systems are reviewed.

In chapter 3, the problem of macromodeling nonlinear circuits and systems is firstly reformulated so that recurrent neural network can be applied. Exploration of different RNN structures is conducted and the determination of input-output RNN model structure is explained. As the new macromodeling approach based on RNN is presented, the input-

output RNN structure is described in detail. According to the feedback connections in the RNN structure, BPTT scheme is introduced and implemented to train the RNN macromodels.

In chapter 4, the proposed technique is applied to macromodel three practical nonlinear circuits and devices namely, a power amplifier, a mixer and a MOSFET. The validity of the proposed macromodeling approach is demonstrated.

In chapter 5, a conclusion and some suggestions for future research are presented.

Chapter 2

Literature Review

2.1 Application of Neural Network in RF/Microwave Area

Neural networks have been demonstrated to be a powerful modeling approach for both passive and active components in microwave area along these years [2][3][4][5][6]. Neural models can be developed purely empirically and directly from measurement/simulation data through training even when component formulas are not available. Neural models can learn and generalize the nonlinear relationship represented by the training data and give accurate and reliable answers to the task in the application instantly. Various components and circuits modeling applications of neural networks have been reported in literature such as microstrip interconnects [7][8][9][10][11], vias [12][13], bends [14], coplanar waveguide (CPW) circuit components [15][16], spiral inductors [17], FET devices [18][19][20], HEMT devices [21][22], packaging interconnects [23], embedded resistors [24], etc. The simple structure of neural models makes their calculation much faster than solving original physical models. The universal approximation theorem makes neural models more accurate than polynomial and

empirical models. They can handle more dimensions than table lookup models. What's more, neural models are easier to be applied to new devices and new technologies.

Two most important issues in neural model development are the determination of neural network structure and the training of constructed neural networks. Many neural networks are capable of modeling nonlinear, multidimensional problems, but different neural network structures may differ in efficiency when representing engineering problems. Appropriate structure may give better accuracy and require less training data and computation expense in training. For example, a Multi Layer Perceptron (MLP) network may use tens of hidden neurons and volumes of training data in predicting a nonlinear circuit waveform. But for knowledge based neural networks, in which prior knowledge is embedded in some particular neuron, less training data and hidden neurons are required as well as higher accuracy can be achieved [7]. Another aspect of the structure issue is to determine the number of hidden neurons. Theoretically, the more nonlinear the problem is, the more hidden neurons are needed. But too many hidden neurons will lead to overlearning, and too few hidden neurons will not represent the problem sufficiently, i.e. underlearning [2].

Training is the essential step in the model development after the neural network is initiated. A neural network can work only after being appropriately trained with sufficient training data. Training algorithm plays a very important role in the success of model development as training data does. A good training algorithm will speed up the training procedure with better accuracy. Back Propagation (BP) scheme was proposed to provide the derivative of output error to each weight in the MLP network in mid 1980's. Based on this scheme, basic weight updating and its variation methods were proposed firstly. After

that, second-order gradient-based optimization methods have been applied to it because the training process is essentially an optimization procedure. Recently, global optimization techniques such as Genetic Algorithm and Simulated Annealing were proposed to overcome the problem of local minimum in the gradient based training methods.

In the following two sections, a review of neural network structures and training algorithms are presented.

2.2 Neural Network Structures

In this section, different structures of neural networks, in terms of different activation functions in the hidden layer and different connection between neurons, that have been applied in RF/Microwave area in the past years are described.

2.2.1 Multilayer Perceptron Networks

A popularly used and basic type of neural networks that has been used in most of the applications is the Multilayer Perceptron (MLP) network [2]. The neurons in this kind of neural network are divided into layers. The neurons in the same layer have no connection to each other and the neurons between consecutive layers are fully connected. As a result, A MLP network is composed of an input layer, one or more hidden layers and an output layer. Suppose the total number of layers is L . The first layer is the input layer, the L th layer is the output layer, and layers 2 to $L-1$ are hidden layers. Suppose the number of neurons in l th layer is N_l , $l = 1, \dots, L$. Let w_{ij}^l represent the weight of the link between j th neuron of $(l-1)$ th hidden layer and i th neuron of l th hidden layer, w_{i0}^l and z_i^l be the bias

parameter and the output of i th neuron of l th hidden layer. So the vector of all trainable weights in a MLP is

$$\Phi = [w_{10}^2 \quad w_{11}^2 \quad w_{12}^2 \quad \dots \quad w_{N_L N_{L-1}}^L]^T \quad (2.1)$$

Let x_i represent the i th input parameter to the MLP. Then following equations stand:

$$z_i^1 = x_i, \quad i = 1, \dots, N_1 \quad (2.2)$$

$$z_i^l = \sigma \left(\sum_{j=1}^{N_{l-1}} w_{ij}^l z_j^{l-1} + w_{i0}^l \right), \quad i = 1, \dots, N_l, \quad l = 2, \dots, L-1 \quad (2.3)$$

where $\sigma(\cdot)$ is the activation function of hidden neurons. The outputs of MLP are produced by linear functions in the output neurons, and can be computed as

$$y_k = \sum_{i=1}^{N_{L-1}} w_{ki} z_i^{L-1} + w_{k0}^L, \quad k = 1, \dots, N_L \quad (2.4)$$

$\sigma(\cdot)$ is usually a monotone squashing function. The most commonly used one is the logistic sigmoid function given by

$$\sigma(\gamma) = \frac{1}{(1 + e^{-\gamma})} \quad (2.5)$$

It is a smooth switch from 0 to 1 as γ sweeps from negative infinity to positive infinity.

Other possible candidates for $\sigma(\cdot)$ could be arctangent function

$$\sigma(\gamma) = \left(\frac{2}{\pi} \right) \arctan(\gamma) \quad (2.6)$$

or hyperbolic tangent function

$$\sigma(\gamma) = \frac{(e^\gamma - e^{-\gamma})}{(e^\gamma + e^{-\gamma})} \quad (2.7)$$

All of them are bounded, continuous, monotonic and continuously differentiable.

The universal approximation theorem of MLP was proved by Cybenko [25] and Hornik et al. [26] in 1989. According to the universal approximation theorem, provided sufficient hidden neurons, a 3-layer perceptron network is virtually capable of approximating an arbitrary continuous, multi-dimensional real static nonlinear function to any desired accuracy. But in practical, how to choose the number of hidden neurons is still an open question. A definite answer is that, the number of hidden neurons depends on the degree of nonlinearity and the dimension of the original problem. More nonlinear and more dimensional problems need more neurons. But too many hidden neurons will result in lots of redundant free variables in Φ , thus lead to the problem of overlearning [2]. Several algorithms have been proposed to find a proper network size, e.g., constructive algorithm [27], network pruning [28].

In practice, 3-layer or 4-layer perceptron networks are commonly used for RF/Microwave applications. Intuitively, a 4-layer perceptron network would perform better for tasks that have certain common localized behavior in different regions of the problem space. For the same task, a 3-layer perceptron network needs lots of hidden neurons to repeat the same behavior in different input regions. In [29], it is pointed out that 3-layer perceptron networks are preferred in function approximation where generalization capability is a major concern. In [30], the author demonstrated that 4-layer perceptron networks would have better performance in boundary definition and are favored for pattern classification tasks.

2.2.2 RBF and Wavelet Neural Networks

Radial Basis Function (RBF) neural networks are feedforward neural networks that only have one hidden layer and the activation function of hidden neurons is a radial basis

function [2]. RBF neural networks have one input layer and one output layer. Neurons in consecutive layers are fully connected. For each connection between input and hidden neurons, there are two parameters, the center parameter and the standard deviation parameter. Suppose there are totally N_h hidden neurons, N_x inputs and N_y outputs. Let c_{ij} and λ_{ij} represent the center and standard deviation parameter of the connection between i th hidden neuron and j th input neuron, $i = 1, \dots, N_h$, $j = 1, \dots, N_x$. Let v_{ij} be the weights between i th output neuron and j th hidden neuron and v_{i0} be the bias of i th output neuron. So the vector of all trainable weights in the RBF is,

$$\Phi = [c_{11} \quad \lambda_{11} \quad c_{12} \quad \lambda_{12} \quad \dots \quad v_{10} \quad \dots \quad v_{N_y N_h}]^T \quad (2.8)$$

Let x_i represent the i th input parameter. Then the following equations stand:

$$\gamma_i = \sqrt{\sum_{j=1}^{N_x} \left(\frac{x_j - c_{ij}}{\lambda_{ij}} \right)^2}, \quad i = 1, \dots, N_h \quad (2.9)$$

$$z_i = \sigma(\gamma_i) \quad (2.10)$$

where $\sigma(\cdot)$ is a radial based function. The outputs of RBF neural networks are processed by linear functions in the output neurons, and can be computed as

$$y_k = \sum_{i=1}^{N_y} v_{ki} z_i + v_{k0}, \quad k = 1, \dots, N_y \quad (2.11)$$

The most commonly used function $\sigma(\cdot)$ for hidden neurons is the Gaussian function given by

$$\sigma(\gamma) = \exp(-\gamma^2) \quad (2.12)$$

Another possible candidate for $\sigma(\cdot)$ could be the multi-quadratic function

$$\sigma(\gamma) = \frac{1}{(\beta^2 + \gamma^2)^\alpha}, \quad \alpha > 0 \quad (2.13)$$

where β and α are constants.

In the feedforward calculation of each hidden neuron, it first calculates the Euclidean distance between input vector $x = [x_1 \dots x_{N_x}]^T$ and its center parameter $c_i = [c_{i1} \dots c_{iN_x}]^T$. The output of the hidden neuron to the output layer is exponentially decayed according to this distance by radial basis function. Consequently, each hidden neuron contributes the output of the total network only in the neighboring area around the center parameter. Because of the finite number of hidden neurons in a RBF neural network, the effective input space of the total neural network is also limited. To cover the space of input parameters, the number of hidden neurons may grow exponentially as the number of input parameter grows [31]. So RBF neural network is suggested for problems with a small number of inputs and whose definition domain is distributed sparsely in the input space [32].

Similar to MLP neural networks, RBF neural networks also have the ability of approximating any static continuous function. The universal approximation theorem for RBF neural networks was proved by Park and Sandberg [33][34]. Universal convergence of RBF net in function estimation and classification has been proved by Krzyzak et al. [35]. Choosing the number of hidden neurons and the parameter Φ for RBF neural networks is more difficult than for MLP networks because of the locality of radial basis functions. A good initialization of the parameters will speed up the training procedure and produce better accuracy.

Wavelet Neural Networks are the outcome of combining wavelet theory and neural network theory and have recently been proposed by Q. Zhang [36][37]. Wavelet neural networks have one input layer, one output layer and one hidden layer which uses wavelet

functions as the activation function. Similar to RBF neural networks, the connection between input layer and hidden layer in wavelet neural networks also has two parameters, i.e. the translation parameter and the dilation parameter. Let us use the notations used in RBF neural networks and rename c_{ij} and λ_{ij} as the translation parameter and dilation parameter. The trainable weights in wavelet neural networks are the same as weights in RBF neural networks. The output of the hidden neuron can be written as:

$$\gamma_i = \sqrt{\sum_{j=1}^{N_x} \left(\frac{x_j - c_{ij}}{\lambda_{ij}} \right)^2}, \quad i = 1, \dots, N_h \quad (2.14)$$

$$z_i = \psi(\gamma_i) \quad (2.15)$$

where $\psi(\cdot)$ is a radial type mother wavelet function. The outputs of Wavelet neural networks are processed by linear functions in the output neurons, and can be computed as

$$y_k = \sum_{i=1}^{N_y} v_{ki} z_i + v_{k0}, \quad k = 1, \dots, N_y \quad (2.16)$$

The most commonly used function $\psi(\cdot)$ for hidden neurons is the reverse-mexican-hat function given by

$$\psi(\gamma) = (\gamma^2 - N_x) \exp\left(-\frac{\gamma^2}{2}\right) \quad (2.17)$$

where N_x is the number of dimensions of input space.

Due to the characteristic of wavelet function in the hidden layer, there is an explicit relationship between the network parameters and the discrete wavelet decomposition of the original problem. As such, the initial values of network parameters can be estimated from the training data using wavelet decomposition formula [36]. The wavelet neural network with radial basis functions can also be considered as a RBF neural network due

to their localized activation function. But the wavelet function is both localized in the input and frequency domains [38]. Besides retaining the advantage of faster training, wavelet networks have a guaranteed upper bound on the accuracy of approximation with a multi-scale structure [39].

2.2.3 Knowledge Based Neural Networks

In the neural networks described in above sections, no problem relevant information is included in the structure. These neural networks that lump up massive hidden neurons in the hidden layer are very useful in nonparametric, purely black box modeling. To employ the abundant existing microwave knowledge into neural network modeling technique, a new structure should be devised. Knowledge Based Neural Network (KBNN) is such a newly proposed RF/Microwave oriented modeling technique that can embed known knowledge into neural network structure [7].

The structure of KBNN is shown in Figure 2.1. There are 6 layers in the structure that are not fully connected to each other. Namely, these 6 layers are input layer X , knowledge layer S , boundary layer B , region layer R , normalized layer $\bar{R}$ and output layer Y . The input layer X accepts parameters x from outside of the model. The knowledge layer S is the place where existing knowledge resides in the form of single or multidimensional functions $\psi(\cdot)$. For knowledge neuron i in the S layer, the output s is given by

$$s_i = \psi_i(x, w_i), \quad i = 1, \dots, N_s \quad (2.18)$$

where x is the vector including neural network inputs x_i ($i = 1, \dots, N_x$), N_s is the number of knowledge neurons, and w_i is the vector of all the parameters in the knowledge formula.

The knowledge function $\psi_i(x, w_i)$ is usually in the form of empirical or semi-analytical

functions. For example, the drain current of a FET is a function of its gate length, channel thickness, doping density, gate voltage and drain voltage [40]. The boundary layer B can incorporate knowledge in the form of problem dependent boundary functions $B(\cdot)$; or in the absence of boundary knowledge just as linear boundaries. Output of the i th neuron in this layer b_i is calculated by

$$b_i = B_i(\mathbf{x}, \mathbf{v}_i) , \quad i = 1, \dots, N_b \quad (2.19)$$

where $\mathbf{v}_i$ is a vector of all parameters in B_i defining an open or closed boundary in the input space $\mathbf{x}$, and N_b is the number of boundary neurons.

Let $\sigma(\cdot)$ be a sigmoid function. The region layer R contains neurons to construct regions from boundary neurons, whose output r can be calculated as

$$r_i = \prod_{j=1}^{N_b} \sigma(\alpha_{ij} b_j + \theta_{ij}), \quad i = 1, \dots, N_r \quad (2.20)$$

where α_{ij} and θ_{ij} are the scaling and bias parameters, respectively, and N_r is the number of region neurons.

The normalized region layer $\bar{R}$ contains rational function based neurons [41] to normalize the outputs of region layer,

$$\bar{r}_i = \frac{r_i}{\sum_{j=1}^{N_r} r_j}, \quad i=1, \dots, N_{\bar{r}}, \quad \text{where } N_{\bar{r}}=N_r \quad (2.21)$$

The output layer Y contains second order neurons [42] combining knowledge neurons and normalized region neurons

$$y_j = \sum_{i=1}^{N_x} \beta_{ji} s_i \left(\sum_{k=1}^{N_{\bar{r}}} \rho_{jik} \bar{r}_k \right) + \beta_{j0} , \quad j = 1, \dots, N_y \quad (2.22)$$

where β_{ji} reflects the contribution of the i th knowledge neuron to output neuron y_j and β_{j0} is the bias parameter. ρ_{jik} is a trainable region selector. If it is 1 indicates that region $\bar{r}_k$ is the effective region of the i th knowledge neuron contributing to the j th output. A total of $N_{\bar{r}}$ regions are shared by all the output neurons. As a special case, if we assume that each normalized region neuron selects a unique knowledge neuron for each output j , the function for output neurons can be simplified as

$$y_j = \sum_{i=1}^{N_s} \beta_{ji} s_i \bar{r}_i + \beta_{j0}, \quad j = 1, \dots, N_y \quad (2.23)$$

Training parameters Φ for the entire KBNN model includes

$$\Phi = [\mathbf{w}_i \ i=1, \dots, N_s; \ \mathbf{v}_i \ i=1, \dots, N_b; \ \alpha_{ij}, \theta_{ij} \ i=1, \dots, N_r, \ j=1, \dots, N_b; \\ \beta_{ji} \ j=1, \dots, N_y, \ i=0, \dots, N_s; \ \rho_{jik} \ j=1, \dots, N_y, \ i=1, \dots, N_s, \ k=1, \dots, N_{\bar{r}}] \quad (2.24)$$

Compared to pure black box neural network structures, the prior knowledge in KBNN gives neural network more information about the original microwave problem, besides the information included the training data. Consequently, KBNN models have better reliability when training data is limited or when the model is used beyond training range.

Figure 2.1: Knowledge Based Neural Network (KBNN) structure

2.3 Training Algorithms

2.3.1 Training Objective

A neural network model can be developed through a process called training. For microwave modeling purposes, the development of neural models is essentially a curve-fitting problem in a multi-dimensional input space. The training process is accordingly equivalent to finding surfaces in the multi-dimensional space that best fit the training data. Suppose the training data set has N_d sample pairs, $\{(x_i, d_i), i=1,2,\dots,N_d\}$, where x_i and d_i are N_x - and N_y -dimensional vectors representing the inputs and outputs of the training data respectively. Let $y = y(x, \Phi)$ represent the input-output relationship of the neural network.

The objective of training is to find variables Φ such that the cost function $E(\Phi)$ of the error between the neural network predictions and the training data is minimized,

$$\min_{\Phi} E(\Phi) = \sum_{i=1}^{N_d} e_i(\Phi) = \sum_{i=1}^{N_d} (y_i(\Phi) - d_i)^2 \quad (2.25)$$

where $e_i(\Phi)$ is the error value of i th sample and $y_i(\Phi)$ is the output of neural network under inputs x_i . The cost function $E(\Phi)$ is a nonlinear function with respect to the adjustable parameter Φ . Due to the complexity of $E(\Phi)$, iterative algorithms are often used to explore the parameter space efficiently [43].

2.3.2 Review of Back Propagation Algorithm

Back Propagation (BP) scheme and the corresponding basic neural network training algorithm were proposed by Rumelhart, Hinton and Williams in 1986 [44]. The training

algorithm is done step by step. In each step, the BP scheme is first done layer by layer to get the derivative information of cost function $E(\Phi)$ to weights Φ . The weights of the neural network Φ then are updated along the negative gradient direction in the weight space. The update formula is given by

$$\Delta\Phi_{now} = \Phi_{next} - \Phi_{now} = -\eta \left. \frac{\partial E(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} \quad (2.26a)$$

or

$$\Delta\Phi_{now} = \Phi_{next} - \Phi_{now} = -\eta \left. \frac{\partial e_i(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} \quad (2.26b)$$

where constant η called learning rate controls the step size of weight update. In formula (2.26a), the weights are updated after all the training samples have been used to teach the neural network. This mode is called batch mode. Another mode is called sample-by-sample mode and represented by formula (2.26b). In this mode, the weights are updated after each sample is presented to the network.

It is hard to determine the learning rate η in the basic BP algorithm described above. The smaller η is, the more iteration is needed for training. But if η is too large in order to speed up the learning process, the training may be unstable due to weight oscillation [31]. A simple method to improve this situation is the addition of a momentum term to weight update formulae as proposed by [44].

$$\Delta\Phi_{now} = -\eta \left. \frac{\partial E(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} + \alpha \Delta\Phi_{old} = -\eta \left. \frac{\partial E(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} + \alpha(\Phi_{now} - \Phi_{old}) \quad (2.27a)$$

$$\Delta\Phi_{now} = -\eta \left. \frac{\partial e_p(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} + \alpha \Delta\Phi_{old} = -\eta \left. \frac{\partial e_p(\Phi)}{\partial \Phi} \right|_{\Phi=\Phi_{now}} + \alpha(\Phi_{now} - \Phi_{old}) \quad (2.27b)$$

where constant α is the momentum factor which controls the influence of the last weight update direction on the current weight update, and Φ_{old} represents the last point of Φ . In [45], it is pointed out that the momentum term has not been sufficiently utilized for reducing oscillation when there is a ravine in the error surface. A correction term that uses the difference of gradients is proposed to reduce weight oscillation. Another idea of reducing the weight oscillation around ravine of error surface is proposed by [46]. An extra constraint is placed so that the alignments of successive weight updates are maximized. The training becomes a constrained optimization problem and nonlinear programming techniques are used as the solution.

An interesting way to accelerate the convergence of backpropagation is to use adaptation schemes that allow the learning rate and the momentum factor to be adaptive during training according to the training error [47]. An adaptive algorithm based on several heuristics is proposed by [48]. First, every weight in the neural network should have its own learning rate that is adaptive to the training process individually. Secondly, if a weight has kept changing in the same direction for several consecutive iterations, a larger step size needs to be applied. Otherwise if a weight is in an oscillation state for several iterations, a smaller step size should be used. In [49], an Enhanced Back Propagation algorithm is introduced. In this algorithm, the learning rate of a weight in the neural network will be increased if the weight is large and is going to be larger. A learning algorithm inspired from the principle of “forced dynamics” for the total error functional is proposed in [50]. In this algorithm, the weights are updated in the direction of steepest descent, but with the learning rate as a specific function of the error and the

norm of the error gradient. As a result, the total error of the neural network can be reduced more efficiently in the vicinity of a local minimum.

Each iteration of the training procedure can be considered as updating the weights in the neural network along a computed direction with a certain step size. The standard BP algorithm uses learning rate as the step size. A more efficient way is to choose an optimal step size using linear search methods along the computed update direction. Examples in this category are line search based on quadratic model [51], and line search based on linear interpolation [52][53].

2.3.3 Gradient-Based Optimization Methods

The standard BP and modified BP algorithms described in the above section are simple and relatively easy to implement. But there is no sound theory on choosing the values of training rate and momentum factor. The contradiction between slow convergence and oscillation around ravine area exists for most of the application. Heuristics are helpful to improve the situation. But implementing them in algorithms can be very difficult because there is no theory that can be used to determine the adaptation scheme. The algorithms using optimal step sizes have a theoretical way to find a good step size. But the update direction simply derived from BP is not efficient. The training is as slow as standard BP algorithm when ravine exists in the error function. The best way would be determining the update direction and step size optimally simultaneously.

Because supervised training of neural networks can be considered as a functional optimization procedure, nonlinear high-order optimization methods using gradient information can be used to improve the rate of convergence. Compared to the algorithms based on heuristics, these methods have a sound theoretical basis and guaranteed

convergence for most smooth functions. Some of the early work in this area was presented in [54] by developing second-order-training algorithms for neural networks. In [52] and [55], various first- and second-order optimization methods for feedforward neural networks training are reviewed.

As mentioned before, the cost function $E(\Phi)$ of training is defined as the summation of the square of difference between output of neural networks and training data. It is a parametric function depending on all the training parameter in the neural network. The purpose of training is to find a (possibly local) minimum point $\Phi = \Phi^*$ in the parameter space that minimizes $E(\Phi)$. Because of the nonlinear activation function and the massive hidden neurons in the hidden layer, $E(\Phi)$ is generally complicated and nonlinear to parameter Φ . Iterative descent algorithms are usually used in the training process. A general description can be as follows:

$$\Phi_{next} = \Phi_{now} + \eta h \quad (2.28)$$

where Φ_{next} is the next point of the parameters to be determined, Φ_{now} is the current point of the parameters, h is a direction vector and η is a positive step size. For the next point Φ_{next} , the following inequality should be satisfied

$$E(\Phi_{next}) = E(\Phi_{now} + \eta h) < E(\Phi_{now}) \quad (2.29)$$

Different optimization algorithms will have different ways to determine the direction vector h . The step size η is the optimal step size found by line search along h

$$\eta^* = \min_{\eta > 0} E(\Phi_{now} + \eta h) \quad (2.30)$$

If the direction vector h is determined based on the gradient (g) of the cost function $E(\Phi)$, the method is called a gradient-based descent method. Suppose there are totally

N_ϕ parameters in the neural network and can be formulated as $\Phi = [\Phi_1 \ \Phi_2 \ \dots \ \Phi_{N_\phi}]^T$. The gradient is formulated as

$$\mathbf{g}(\Phi) = \nabla E(\Phi) = \left[\frac{\partial E(\Phi)}{\partial \Phi_1}, \frac{\partial E(\Phi)}{\partial \Phi_2}, \dots, \frac{\partial E(\Phi)}{\partial \Phi_{N_\phi}} \right]^T \quad (2.31)$$

The gradient vector of a neural network can be derived by BP [44] procedure.

In general, a feasible descent direction vector $\mathbf{h}$ should satisfy the following condition [43],

$$\left. \frac{dE(\Phi_{now} + \eta \mathbf{h})}{d\eta} \right|_{\eta \rightarrow 0} = \mathbf{g}^T \mathbf{h} = |\mathbf{g}| |\mathbf{h}| \cos(\xi(\Phi_{now})) < 0 \quad (2.32)$$

where $\xi(\Phi_{now})$ denotes the angle between $\mathbf{h}$ and $\mathbf{g}_{now}$ at the current point Φ_{now} . This can be verified by the Taylor series expansion of $E(\Phi)$ respect to η ,

$$E(\Phi_{now} + \eta \mathbf{h}) - E(\Phi_{now}) \approx \eta \mathbf{g}^T \mathbf{h} \quad (2.33)$$

which is obtained by retaining the first term when $\eta \rightarrow 0$. Compare (2.33) and (2.29), it is clear that $\mathbf{g}^T \mathbf{h} < 0$ is the condition that has to be held.

A general and simple way to get a feasible descent direction is deflecting the gradient $\mathbf{g}$ by multiplication,

$$\mathbf{h} = -\mathbf{G}\mathbf{g}, \ \Phi_{next} = \Phi_{now} - \eta \mathbf{G}\mathbf{g} \quad (2.34)$$

where η is a positive step size and $\mathbf{G}$ is a positive definite matrix. The condition (2.32) is obviously held because

$$\mathbf{g}^T \mathbf{h} = -\mathbf{g}^T \mathbf{G}\mathbf{g} < 0 \quad (2.35)$$

The following described gradient-based methods (e.g., quasi-Newton method, Gauss Newton Method and the Levenberg-Marquardt method) also have a similar form of

(2.34). But they have better choices of the deflection matrix G or direction vector h with a theoretical basis.

A. Newton's Method

Newton's method is a second-order optimization method that uses the second-order derivatives of the cost function $E(\Phi)$ to determine the descending direction vector h [43]. First the cost function is approximated by a second-order Taylor series,

$$E(\Phi + \Delta\Phi) \approx E(\Phi) + g(\Phi)^T \Delta\Phi + \frac{1}{2} \Delta\Phi^T H(\Phi) \Delta\Phi \quad (2.36)$$

where H is the *Hessian* matrix, consisting of the second-order partial derivatives of $E(\Phi)$. The minimum point $\Phi^* = \Phi + \Delta\Phi$ can be simply found by differentiating (2.36) and setting it to zero.

$$0 = g + H\Delta\Phi \quad \text{and} \quad \Phi^* = \Phi - H^{-1}g \quad (2.37)$$

If H is positive definite and cost function $E(\Phi)$ is essentially quadratic, the Newton's method would directly go to the minimum in a single step. But $E(\Phi)$ is not quadratic according to the nonlinear activation functions. Iterative procedure is required in the training procedure. Calculating the inverse of Hessian matrix is also very computational expensive when the size of the network becomes large. This makes the training procedure inefficient and not scalable. For this reason the original Newton's method is rarely used in reality.

B. Conjugate Gradient Method

Conjugate gradient method is originally derived from quadratic minimization. The result direction vector h is the conjugate direction and is computed in the following way [53].

With an initial guess of weights $\Phi_{initial}$, the Conjugate Gradient method sets initial gradient $\mathbf{g}_{initial} = \frac{\partial E}{\partial \Phi} \Big|_{\Phi = \Phi_{initial}}$, and direction vector $\mathbf{h}_{initial} = -\mathbf{g}_{initial}$. Then vector

sequences $\mathbf{g}$ and $\mathbf{h}$ are constructed recursively by

$$\mathbf{g}_{next} = \mathbf{g}_{now} + \lambda_{now} \mathbf{H} \mathbf{h}_{now} \quad (2.38)$$

$$\mathbf{h}_{next} = -\mathbf{g}_{next} + \gamma_{now} \mathbf{h}_{now} \quad (2.39)$$

$$\lambda_{now} = \frac{\mathbf{g}_{now}^T \mathbf{g}_{now}}{\mathbf{h}_{now}^T \mathbf{H} \mathbf{h}_{now}} \quad (2.40)$$

$$\gamma_{now} = \frac{\mathbf{g}_{next}^T \mathbf{g}_{next}}{\mathbf{g}_{now}^T \mathbf{g}_{now}} \quad (2.41)$$

or,

$$\gamma_{now} = \frac{(\mathbf{g}_{next} - \mathbf{g}_{now})^T \mathbf{g}_{next}}{\mathbf{g}_{now}^T \mathbf{g}_{now}} \quad (2.42)$$

where $\mathbf{H}$ is the Hessian matrix of the objective function $E(\Phi)$. Equation (2.41) is called the Fletcher-Reeves formula and (2.42) is called the Polak-Ribiere formula [53].

An alternative way can be used to compute the conjugate direction. First compute Φ_{next} by proceeding Φ from Φ_{now} along the direction $\mathbf{h}_{now}$ to the local minimum through line minimization, and then set $\mathbf{g}_{next} = \frac{\partial E}{\partial \Phi} \Big|_{\Phi = \Phi_{next}}$. This $\mathbf{g}_{next}$ is then used as the result

vector of (2.38). By this way, only line minimization computation is needed to find the conjugate direction. The descent direction $\mathbf{h}$ i.e. the conjugate direction can be accumulated without involving computational-intensive matrix inversions. As a result, conjugate gradient methods are very efficient and scale well with the size of parameters in the neural network.

If the cost function $E(\Phi)$ is a quadratic and convex, the minimum of the cost function $E(\Phi)$ can be efficiently found within N_ϕ iterations, where N_ϕ is the parameters in the neural network. But as what is mentioned, $E(\Phi)$ is not quadratic with respect to the training variables Φ because of the nonlinear activation functions in the hidden layers. As a result, the convergence rate of the method depends on the degree that the cost function can be approximated by a local quadratic function.

Determining the optimal step size in each iteration is computational expensive because every function evaluation involves a complete cycle of sample presentation and neural network feed-forward calculation. A Scaled Conjugate Gradient (SCG) algorithm was introduced in [56] to avoid the line search per learning iteration by using Levenberg-Marquardt approach to scale the step size. It is proved that the conjugate gradient method is equivalent to error backpropagation with optimized momentum term [52].

C. Quasi-Newton Training Algorithms

Quasi-Newton method is also derived from quadratic objective function optimization. In this method, the descent direction h is got by deflecting the gradient vector g using a matrix A . Unlike Newton's method, the matrix A used here is an estimation of the inverse of Hessian matrix H^{-1} . The weights are updated as

$$\Phi_{next} - \Phi_{now} = -\eta h = -\eta A_{now} g_{now} \quad (2.43)$$

$$A_{now} = A_{old} + \Delta A_{now} \quad (2.44)$$

The A matrix is successively estimated employing rank 1 or rank 2 updates after each iteration [57]. There are two major rank 2 formulae to compute ΔA_{now} ,

$$\Delta A_{now} = \frac{\Delta \Phi \Delta \Phi^T}{\Delta \Phi^T \Delta g} - \frac{A_{old} \Delta g \Delta g^T A_{old}}{\Delta g^T A_{old} \Delta g} \quad (2.45)$$

or

$$\Delta A_{now} = (1 + \frac{\Delta g^T A_{old} \Delta g}{\Delta \Phi^T \Delta g}) \frac{\Delta \Phi \Delta \Phi^T}{\Delta \Phi^T \Delta g} - \frac{\Delta \Phi \Delta g^T A_{old} + A_{old} \Delta g \Delta \Phi^T}{\Delta \Phi^T \Delta g} \quad (2.46)$$

where

$$\Delta \Phi = \Phi_{now} - \Phi_{old}, \quad \Delta g = g_{now} - g_{old} \quad (2.47)$$

Equation (2.45) is called the DFP (Davidon-Fletcher-Powell) formula and equation (2.46) is called the BFGS (Broyden-Fletcher-Goldfarb-Shanno) formula [57].

Suppose there are total N_Φ weights in the neural network structure. For standard quasi-Newton methods, N_Φ^2 storage space and a line search are required to maintain an approximation of the inverse Hessian matrix and calculate a reasonably accurate step size. In Limited-memory (LM) or one-step BFGS method [58], the inverse Hessian approximation is reset to the identity matrix I after every iteration. The space of storing matrices is consequently saved. When N_Φ becomes large, the consumption of storage memory and computation on line search become the bottleneck for efficiency and scaling. In [59] a second-order learning algorithm is proposed so that the decent direction h is computed on the basis of a partial BFGS update with less memory. A reasonably accurate step size is efficiently calculated as the minimal point of a second-order approximation of the cost function. In [60], various approaches to the parallel implementation of second-order gradient-based MLP training algorithms, such as full and limited memory BFGS algorithms, were presented. Through the estimation of inverse Hessian matrix, quasi-Newton has faster convergence rate than conjugate gradient method.

D. Levenberg-Marquardt and Gauss-Newton Training Algorithms

Gauss-Newton and Levenberg-Marquardt optimization methods can be used for neural network training because of the similarity between neural network training and nonlinear least square curve fitting. They are derived by a linearized approximation of the Hessian matrix. Let N_d be the number of data samples and N_y be the number of outputs. Let e be a vector of length $N_d \times N_y$ containing all the individual errors of all data samples. Let J be the Jacobian matrix including the derivative of e with respect to Φ . J has N_ϕ columns and $N_d \times N_y$ rows.

The Gauss-Newton update formula can be expressed as [61]

$$\Phi_{next} = \Phi_{now} - (J_{now}^T J_{now})^{-1} J_{now}^T e_{now} \quad (2.48)$$

and The Levenberg-Marquardt [61] method is given by

$$\Phi_{next} = \Phi_{now} - (J_{now}^T J_{now} + \mu I)^{-1} J_{now}^T e_{now} \quad (2.49)$$

where μ is a non-negative number. An interesting point compared to Newton's method is that $J_{now}^T J_{now}$ is positive definite unless J_{now} is rank deficient. When J_{now} is rank deficient, a diagonal matrix μI is added in Levenberg-Marquardt method to compensate the deficiency.

If the training sample and neural network size become large, the size of Jacobian matrix grows intolerably. The inversion of the approximated Hessian matrix becomes computational prohibitive. In [62], a modified Levenberg-Marquardt training algorithm that uses a diagonal matrix instead of the identity matrix I in (2.49) was proposed for efficient training of multilayer feedforward neural networks. In [63], the training samples are divided into several groups called local-batches. The training is performed

successively through these local batches. In [64], an algorithm is proposed that utilizes the deficiency of the Jacobian matrix and reduces the computational and memory requirement of Levenberg-Marquardt method.

2.3.4 Global Optimization Methods

All gradient-based training methods share the defect of converging to a local minimum as the training result. To allow the training algorithms to escape from local minima and converge to the global minimum, some training methods using random optimization techniques have been proposed. These methods are characterized by a random search element in the training process. Two representative algorithms in this class are the simulated annealing algorithm and the genetic algorithm. In [65] is described the simulated annealing algorithm that analogs the annealing process of atoms inside metal. The random parameter in the optimization process is controlled by a parameter called temperature, which determines the search range a local minimum can jump out. In [66], the genetic training algorithm is introduced to train the neural network. The genetic algorithm simulates the biological evolution and the weights of the neural network are coded as chromosomes. The local convergence of the algorithm is achieved by crossover operation and selection procedure, and the global search is accomplished by the random mutation. In [67], based on a Langevin updating (LV) rule, noise is added to the weights during training. And in [68] a stochastic minimization algorithm is developed for neural networks training.

Because of the random search during the neural network training, the convergence of these training methods is slow. A hybrid method that combines the conjugate gradient method and random optimization is proposed in [51]. During the training with conjugate

gradient method, if a flat error surface is encountered, the training algorithm switches to the random optimization method. After the training escapes from the flat error surface, it switches back to the conjugate gradient algorithm.

2.4 Review of Approaches of Macromodeling Nonlinear Microwave Circuits

2.4.1 Behavioral Modeling Technique

A popularly used macromodeling approach of nonlinear microwave circuits is the Behavioral Modeling technique. In this approach, the input/output behaviors of the nonlinear circuit are characterized by a set of well-defined parameters. When new input signals are fed, the output signal then can be calculated from these parameters and inputs.

A simple version of this approach uses a set of simple parameters to describe different aspects of the relationship between input and output signals. For example, as illustrated in Figure 2.2, a nonlinear class-A power amplifier can be modeled with following parameters, small signal gain G_{ss} , compression coefficient K , saturated power P_{sat} , 1-dB compression point P_{1dB} , 3rd order intercept point P_{3rdICP} , third-order intermodulation IM3, DC power P_{DC} , power added efficiency PAE, and phase distortion AM-PM [69]. The task of macromodeling is to formulate the static mapping between these parameters and the input signals, namely, the DC biases, the input power P_{in} , the frequency f , and the phase ϕ_{in} . It can be accomplished by various curve-fitting techniques, such as linear regression, table-lookup (linear interpolation), logarithmic regression, power function regression, exponential regression, polynomial regression, and spline curve fitting, etc. In [69], a closed-form function in the form of

$$y(x) = A + Bx + \sum_{j=1}^J C_j \times \log_{10} \left[\frac{10^{\frac{x-x_j}{C_j}}}{1 + 10^{\frac{x-x_j}{C_j}}} \right] + \sum_{k=1}^K D_k \times \log_{10} \left[1 + 10^{\frac{x-y_k}{D_k}} \right] \quad (2.50)$$

is used to find the nonlinear relationship. The parameters A , B , C_j , x_j , D_k and y_k are determined by an optimization procedure based on a set of measurements. It is worthy to mention that feed forward neural networks can also easily do the function approximation task.

Figure 2.2: Measurable amplifier parameters for behavioral modeling

In a more systematic and generic behavioral modeling approach, the macromodels are built in frequency-domain. The spectral components of input and output signals at the ports in the circuit are used to build the model. The task of macromodeling is to generate functions to map the nonlinear relationship between all the input spectral components and all the output spectral components. Though from a mathematical point of view, the modeling procedure is a multi-dimensional function approximation that starts from

measurement or simulation data. Special attention should be paid to the huge input space, which includes all the spectral components of all the input ports. It is pointed out in [70] that several concepts are needed to make this approach successful in practice. The first one is the time-invariance concept, which means applying a frequency proportional phase shift to input spectral components will result in the same frequency proportional phase shift to all output spectral components [71]. The second one is applying the linearization concept to those relatively small spectral components so that the superposition principle can be applied [72]. This concept reduces the dimensionality of the input space to a manageable size. But even with these concepts, the function-approximation is not a small issue due to the high nonlinearity of the relationship.

To evaluate the overall system behavior, especially the time-domain behavior, only frequency-domain information is not enough. In [73][74], a Volterra-mapping based multi-order scattering parameter behavioral modeling technique is introduced. In this technique, a set of Volterra kernels/transfer functions are determined by the multi-order s -parameters using Fourier Transforms. With these Volterra kernels, the nonlinear time domain behavior of RF/Microwave nonlinear circuits and systems can be calculated. But in practice, it is difficult to extract accurate high-order Volterra kernels either from time-domain or frequency-domain measurements. This difficulty restricts the application of this technique only to those slightly nonlinear circuits and systems.

2.4.2 Equivalent Circuit Based Approach

Another popular macromodeling approach is the equivalent circuit based approach. Typically, the techniques of this approach result in a simpler circuit with lumped nonlinear components compared to the original nonlinear circuit. Several macromodeling

techniques based on equivalent circuit for large-scale nonlinear circuits and systems have been proposed. Early in 1974, techniques known as “circuit simplification” and “circuit built-up” were used to generate macromodels for operational amplifier [75]. Based on the understanding and experience, different parts of the original nonlinear circuit are either simplified by using smaller number of the same ideal elements or re-built by generating a new circuit configuration. The parameters and values of elements are determined by matching certain external specifications of the original circuit. Later, an automated macromodeling algorithm for analog circuits was introduced in [76]. It can be applied to general nonlinear circuits if the original circuit satisfies following conditions:

- All the components in the circuit can be modeled as independent current sources, resistors, capacitors, and voltage-controlled current sources.
- Resistors, capacitors, and controlled sources are not required to be linear, i.e., that they can be described by branch equations of the form $i_d = f_d(v_d)$ or $q_d = f_d(v_d)$, where i_d is the current flowing through the device, q_d is the charge on the capacitor, f_d is a function of class C^1 which depends on the device and v_d is the controlling branch voltage. Furthermore, it is reasonably assumed that there exists a constant $c_{min} > 0$ such that $(dq_d/dv_d) \geq c_{min}$ for all voltages v_d and all capacitors in the circuit.
- There are capacitors connecting the ground node to all other nodes of the circuit.

Besides the assumptions, a template, e.g. the topology of the equivalent circuit needs to be supplied. With the provided template, the dynamics of the macromodel can be formulated in time domain using general q - v - i form with outputs v^o as the solution:

$$\begin{cases} \dot{\mathbf{q}} = -\mathbf{f}(\mathbf{v}) + \mathbf{B}\mathbf{i} \\ \mathbf{v}^o = \mathbf{g}(\mathbf{q}) \end{cases} \quad (2.51)$$

where $\mathbf{f}$ and $\mathbf{g}$ are nonlinear C^1 functions, $\mathbf{B}$ is a matrix with 1 for nodes connected with current source $\mathbf{i}$ and 0 elsewhere.

The algorithm then determines the value of parameters in the equivalent circuit by minimizing the difference of solution $\mathbf{v}^o$ between the original circuit and the macromodel under the same excitation $\mathbf{i}$. Techniques used in optimal control and nonlinear programming are employed to do the minimization procedure. The resulting macromodel can be used for general-purpose circuit simulation.

Though with the merits of automation and generating general-purpose macromodels, this technique has several disadvantages. First, the requirements of the algorithm restrict the range of its application. Secondly, providing the template equivalent circuit of the macromodel is not a trivial task, which requires the good understanding of the original circuit and practical experience. Usually, a trial and error procedure is needed to get an appropriate equivalent circuit for a large-scale nonlinear circuit.

2.4.3 Time Domain S-parameter Technique

A macromodeling technique using time-domain measurements is newly proposed in [77]. In this technique, each sub-module in the system is modeled using time-domain nonlinear scattering functions. No equivalent circuits are used and the overall system is solved in time domain. Figure 2.3 illustrates a two-port time-domain scattering function model:

Figure 2.3: Time Domain scattering functions model of a two port nonlinear circuit

$$\begin{aligned} V_{1r}(t) &= S_{11}(V_{1i}(t)) + S_{12}(V_{2i}(t)) \\ V_{2r}(t) &= S_{21}(V_{1i}(t)) + S_{22}(V_{2i}(t)) \end{aligned} \quad (2.52)$$

where $V_{1i}(t)$ and $V_{2i}(t)$ are the incident voltage waveforms, $V_{1r}(t)$ and $V_{2r}(t)$ are the reflected voltage waveforms.

The incident, reflected and transmitted waves are measured to identify the time-domain scattering functions. A general form of the scattering function can be formulated in time domain as:

$$y(t) = \sum_{i=0}^n F_i(x(t-iT)) - \sum_{i=0}^n G_i(y(t-iT)) \quad (2.53)$$

where x and y are input and output waveforms respectively, T is the sampling time interval and F_i and G_i are nonlinear kernel functions in the form of $\alpha \tanh(\beta x)$. The number of kernel functions n and the parameters inside kernel functions F_i and G_i 's are to be identified by the modeling procedure.

This approach treats the nonlinear system under observation as a pure black box, and uses time-domain measurements directly to find the input-output model, thus it avoids the procedure of constructing an equivalent circuit for complicated nonlinear circuits. This could be very useful when the knowledge of the original circuit is not enough, or the original circuit is too complicated to generate an equivalent circuit. But in the optimization procedure of generating the scattering function, it is not easy to determine the number and the form of kernel functions G_i and F_i , and the parameters inside of the nonlinear kernel functions.

2.4.4 Krylov-subspace Based Technique

In the case that the dynamics of the original nonlinear circuits can be formulated and solved analytically, a technique based on Krylov-subspace is proposed in [78]. This technique can reduce the order of the original system to a user specified number q so that the first ' q ' derivatives of the time-response of the original system are retained.

In this technique, the nonlinear state-based dynamic equations of the original big nonlinear circuit are first formulated in time domain. Taylor expansion is then applied to time-related parameters in both sides of these equations with respect to time, which including system states and input signals. The Krylov subspace of the original nonlinear system is formulated as a matrix by putting together the first q Taylor coefficients of all the system states. Perform QR decomposition on the obtained matrix. The reduced system is composed by a set of new states, which is the result of performing a congruent transformation to the original system states using Q matrix resulted from QR decomposition. The new system has an order of q and they are theoretically identical to the first q orders of the original nonlinear system. By replacing the original nonlinear

circuit with the new system in circuit simulation, a significant speed improvement is found.

2.5 Conclusion

Neural networks have been extensively applied in the modeling of RF/Microwave components and circuits. Various kinds of feed-forward neural network structures have been successfully applied to find the nonlinear and multi-dimensional static input/output mapping. Different efficient training algorithms have been developed to make the modeling procedure convenient. However, no neural network structure has been introduced to modeling the dynamic time-domain nonlinear circuit behaviors. Four macromodeling techniques have been proposed in literature to build the macromodel for nonlinear circuits and systems. Based on the fact that Fourier/Laplace Transforms are not applicable to nonlinear case, nonlinear circuits are better modeled in time domain. The existing behavioral macromodeling approaches use frequency domain information either directly or indirectly, thus the result macromodel can not provide enough accurate overall system behavior. Those equivalent circuit based techniques require thorough understanding of the original circuit and experience to generate a condensed equivalent circuit. Because of the complexity of time-domain input/output relationship, lots of efforts are needed to generate the kernel function for time-domain scattering parameters. Krylov-subspace based technique can reduce the order of the system theoretically, but it requires the detailed formulation of the original system which may be unavailable when devices based on new technology are introduced. In the following part of the thesis, a new macromodeling approach is proposed to combine the merits of neural network technique and time-domain modeling approaches.

Chapter 3

Proposed RNN Macromodeling Approach and Back Propagation Through Time

3.1 Formulation of Circuit Dynamics

Let N_u and N_y be the total number of input and output signals in the nonlinear circuit respectively, and N_p be the total number of circuit parameters. Let $\mathbf{y}=[y_1 \ y_2 \ \dots y_{N_y}]^T$, $\mathbf{u}=[u_1 \ u_2 \ \dots u_{N_u}]^T$ and $\mathbf{p}=[p_1 \ p_2 \ \dots p_{N_p}]^T$ be vectors of the output signals, input signals and circuit parameters of the nonlinear circuit respectively, where T denotes transposition. The dynamics of the original nonlinear circuit can be generally described as a nonlinear system in state variable form as:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{\Gamma}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}) \\ \mathbf{y}(t) &= \mathbf{\Psi}(\mathbf{x}(t), \mathbf{u}(t))\end{aligned}\tag{3.1}$$

where $\mathbf{x}=[x_1 \ x_2 \ \dots x_{N_x}]^T$ is the vector of state variables and N_x is the number of states,

$\mathbf{\Gamma} \in \mathbf{R}^{N_x \times N_x \times N_p} \rightarrow \mathbf{R}^{N_x}$ and $\mathbf{\Psi} \in \mathbf{R}^{N_x \times N_u} \rightarrow \mathbf{R}^{N_y}$ are two independent static nonlinear functions. In a modified nodal formulation [79], the state vector $\mathbf{x}(t)$ includes nodal voltages, currents of inductors, currents of voltage sources and charge of capacitors.

When the nonlinear circuit becomes complicated, though the number of input parameter N_u and output parameter N_y may be small, the number of internal states N_x will be large. Solving these original nonlinear differential equations is computationally intensive. For higher level design and optimization, where this circuit is used as a sub-module and repetitive evaluations for different circuit inputs are needed, a more simplified and convenient computational form of the original circuit should be very useful.

Analogously reducing the order of nonlinear circuits in time domain has been conducted in [78]. But in the situations that the detailed formulation of the original nonlinear circuit is not available, what can be used to build macromodels will be the measurements or simulation data. Because of the discrete nature of measurements and simulation, it is better to model the nonlinear circuit in discrete time domain. With a small enough sampling rate, (3.1) can be narrated in normalized discrete time domain in input-state-output form as:

$$\begin{aligned} \chi(k+1) &= \varphi(\chi(k), u(k), p) \\ y(k+1) &= \psi(\chi(k+1), u(k+1)) \end{aligned} \quad (3.2)$$

where $\varphi \in R^{N_x \times N_x \times N_u \times N_p} \rightarrow R^{N_x}$ and $\psi \in R^{N_x \times N_u} \rightarrow R^{N_y}$ are two independent nonlinear functions.

It is assumed that the nonlinear circuit under modeling is stable, which means that the conceptual feedback loop gain of any part of the circuit is less than unit. Most of nonlinear circuits except oscillators satisfy this condition, such as amplifier, mixer, etc. Another assumption that can also be easily applied to most of the nonlinear circuit is that the nonlinear circuit can be linearized in a certain neighborhood of a bias. Every DC bias solution without any dynamic input signal is considered as an equilibrium state. In a

small enough region around a certain bias equilibrium state, it is assumed that the linearized circuit has a limited order and can be uniquely determined by an input sequence with sufficient length.

Under these assumptions that can be satisfied by most nonlinear RF/Microwave circuits and systems, according to the theory described in [80], the nonlinear circuit/system (3.2) can be expressed by an input-output formulation:

$$\mathbf{y}(k) = \mathbf{g}(\mathbf{y}(k-1), \dots, \mathbf{y}(k-M_y), u(k-1), \dots, u(k-M_u), \mathbf{p}) \quad (3.3)$$

where k is the time index in discrete time domain, M_y and M_u are total number of delays of $\mathbf{y}$ and u respectively, M_u usually equals M_y , and $\mathbf{g}$ is a set of nonlinear functions. M_y and M_u also represent the order of the original nonlinear circuit dynamics. This model can also be used as an alternative representation of the dynamics in the original circuit of Equation (3.1).

3.2 Formulation of Macromodeling

The purpose of macromodeling is to develop a model that has a similar input-output relationship as the original complex circuit within an acceptance error range. At the same time, the evaluation of the macromodel should be much faster than that of the original circuit. Suppose the candidate macromodel is represented by $M: u(t) \rightarrow \hat{\mathbf{y}}(t)$, which is a functional on the space of input waveforms $u(t)$. Suppose $[T_1, T_2]$ represents the time sampling range of interest for the input and output signals. Let u_{max} and u_{min} represent the upper and lower boundaries of input signal $u(t)$. For each input $u(t)$, $u_{min} \leq u(t) \leq u_{max}$, $t \in [T_1, T_2]$, the quality of the macromodel can be represented by the distance between the output of the macromodel and that of the original circuit in a l_2 norm:

$$\int_1^T \|\mathbf{y}(t) - \hat{\mathbf{y}}(t)\|^2 dt \quad (3.4)$$

3.3 Selection of RNN Structure and Training Scheme

In practice, it is often difficult to determine the Equation (3.2) and (3.3) analytically from measurements for a large-scale nonlinear circuit. Neural networks are well known to identify nonlinear relationships between input and output parameters, and have been successfully applied in solving practical modeling problems. But with conventional feedforward neural networks that have been used in RF/Microwave area, the self-dependence of outputs in time domain is hard to be formulated and trained. In this thesis, RNNs are employed to learn the dynamic characteristics of nonlinear circuits and construct the macromodels. RNNs are neural networks with feedback from outputs to the input. It has been used in areas such as signal processing, speech recognition [31], system identification and control [81][82][83][84][85].

3.3.1 Structure: State Based vs. Input/output Based

Corresponding to the formulation (3.2) and (3.3), there are two RNN structures that can be used for modeling purpose, i.e., the state-based structure and input/output-based structure as shown in Figure 3.1 and Figure 3.2. In this thesis, the input/output-based RNN structure is used based on the following practical reasons:

- Firstly, it is hard to determine an appropriate number of hidden states for a state-based RNN. Because the number of states can not usually be known beforehand when the circuit under modeling is treated as a black box. Comparatively, there is no hidden state in an input/output RNN structure.

Figure 3.1: Illustration of state-based RNN structure

Figure 3.2: Illustration of input/output based RNN structure

- Secondly, it is difficult to determine the values of hidden states h during training even if the number can be specified arbitrary, because the hidden states are inaccessible from outside. For input/output RNN structure, only input and output measurements are required, which can be easily done.
- The third reason is related to the model development. To build a state-based RNN macromodel, two cascade layer of nonlinear functions and lots of state variables are to be determined during training. These many varieties often require lots of training data. Comparatively, in the input/output RNN structure, there is only one layer of nonlinear functions, and there are no hidden states in the middle.

3.3.2 Training Scheme: Series-Parallel vs. Parallel

There are two schemes can be used to develop RNN macromodels. Namely, the Series-Parallel scheme and Parallel scheme as shown in Figure 3.3 (a) and (b) correspondingly. Both of these two schemes use the original nonlinear circuits as the reference and try to minimize the difference between the outputs of RNN macromodels and the original circuits under the same input. The key difference between the two schemes is the source of feedback during the development stage. In Series-Parallel scheme, the feedback to the RNN is the output of the reference circuits. While in Parallel scheme, the feedback comes from the output of RNN itself. The error between the outputs of the original nonlinear circuit and the RNN macromodel are used to adjust the weights inside the feed forward part between inputs and outputs. Conventional BP can be used for Series-Parallel scheme. But for Parallel scheme, an advanced BP scheme, BPTT is needed because of the self-dependence in the structure. Though the training of Parallel

scheme is more complicated than Series-Parallel scheme, it will be preferred if we take consideration of application after the RNN macromodels are developed:

- When RNN macromodels are applied in circuit simulation as the replacement of the original nonlinear circuit, there is no solution of the original nonlinear circuit available as the reference feedback. The feedback to the RNN macromodels has to be their own outputs.
- If a RNN macromodel is developed under Series-Parallel scheme, the output and the feedback are not correlated. The feedback is always the accurate solution from reference and the BP based training algorithm neglects the concatenated effect between outputs of consecutive time points. When the self-feedback is used in application, the problem of error propagation will appear. The accuracy of the macromodel decreases as time goes on and often ends with instability. Comparatively, macromodels developed under Parallel scheme are robust because self-feedback has already been considered during training.

Three-layer perceptron neural networks have been proved to be able to model arbitrary continuous nonlinear relationship by the universal approximation theorem [25][26]. Thus three-layer perceptron neural networks are used as the feed forward part in the input/output based RNN structure in this thesis. The proposed macromodeling approach and the RNN structure are illustrated in Figure 3.4.

3.4 Detailed Formulation of The RNN Macromodel Structure

The input of RNN includes time varying input part u and time-invariant part p and the

(a)

(b)

Figure 3.3: RNN training schemes, (a) Series-parallel scheme (b) Parallel scheme

output is the time varying signal y . For example, if the macromodel is used to represent an amplifier, then u will represent the amplifier input signal, p will represent the circuit parameters related to the amplifier, and y will be the output signal from amplifier. The first hidden layer of the RNN, labeled x layer, contains buffered history of circuit output signal y which is fed back from the output of RNN, buffered history of circuit input signal u , and circuit parameter p . Let time sample index be k . Let K_y and K_u be the number of buffers for output y and input parameter u , respectively. Then neurons in x layer are defined as:

$$x_i(k) = \begin{cases} y_j(k-n), & \text{for } i = (n-1)N_y + j, \quad 1 \leq j \leq N_y, 1 \leq n \leq K_y \\ u_j(k-n), & \text{for } i = K_y N_y + (n-1)N_u + j, \quad 1 \leq j \leq N_u, 1 \leq n \leq K_u \\ p_n, & \text{for } i = n + K_y N_y + K_u N_u, \quad 1 \leq n \leq N_p \end{cases} \quad (3.5)$$

Let N_x be the total number of neurons in x layer, we have $N_x = N_p + N_y K_y + N_u K_u$.

The next hidden layer is called the z layer. The neurons in this layer contain sigmoid activation functions. Let N_z be the total number of hidden neurons in the z layer. Let w_{ij} be the weight between i th neuron in z layer and j th neuron in x layer. The vector of weights between x layer and z layer is denoted as $\mathbf{w} = [w_{11} \ w_{12} \ \dots \ w_{N_z N_x}]^T$, and $\boldsymbol{\theta} = [\theta_1 \ \theta_2 \ \dots \ \theta_{N_z}]^T$ denotes the bias vector of neurons in z layer. For a given set of values in x layer, the outputs of hidden neurons in layer z can be computed from delayed input and output signals as:

$$\begin{aligned} \gamma_j(k) &= \sum_{i=1}^{K_y} \sum_{l=1}^{N_y} y_l(k-i) w_{j[(i-1)N_y+l]} + \sum_{i=1}^{K_u} \sum_{m=1}^{N_u} u_m(k-i) w_{j[K_y N_y + (i-1)N_u + m]} \\ &\quad + \sum_{i=1}^{N_p} p_i w_{j[K_y N_y + K_u N_u + i]} + \theta_j, \\ z_j(k) &= \phi(\gamma_j(k)) \quad j = 1, \dots, N_z \end{aligned} \quad (3.6)$$

Figure 3.4: The proposed RNN based macromodel structure.

where $\phi(\gamma)$ is a sigmoid function, $\phi(\gamma) = \frac{1}{1 + e^{-\gamma}}$.

The last layer is the output layer called y layer. The y layer outputs are linear functions of the responses of hidden neurons in the z -hidden layer. Let $\mathbf{v} = [v_{11} \ v_{12} \ \dots \ v_{N_y N_z}]^T$ be the vector of weights between z layer and y layer and $\boldsymbol{\eta} = [\eta_1 \ \eta_2 \ \dots \ \eta_{N_y}]^T$ be the bias vector of output neurons in y layer. The output of the overall macromodel is computed as:

$$y_i(k) = \sum_{j=1}^{N_z} z_j(k) v_{ij} + \eta_i \quad i = 1, \dots, N_y \quad (3.7)$$

where v_{ij} is the weight between the i th neuron in the y layer and the j th neuron in the z layer.

Let the parameters of the RNN be denoted as $\boldsymbol{\Phi}$, $\boldsymbol{\Phi} = [\mathbf{w}^T \ \boldsymbol{\theta}^T \ \mathbf{v}^T \ \boldsymbol{\eta}^T]^T$. The part in the macromodel structure from x layer, z layer, to y layer is a Feed Forward Neural Network (FFNN) denoted as $\mathbf{y}(\mathbf{x}, \boldsymbol{\Phi})$. The overall neural network realizes a nonlinear relationship:

$$\mathbf{y}(k) = f(\mathbf{y}(k-1), \dots, \mathbf{y}(k-K_y), \mathbf{u}(k-1), \dots, \mathbf{u}(k-K_u), \mathbf{p}, \boldsymbol{\Phi}) \quad (3.8)$$

Here the number of delay buffers K_y and K_u represent the order of dynamics in the RNN macromodel.

3.5 Model Development and RNN Training

3.5.1 Objective of RNN Training

The RNN macromodel will not represent the nonlinear microwave circuit behavior unless it is trained by training data. Following the requirement described in section 3.2, the macromodel should be trained to match the output of the original circuit under

various input excitations in the interval $[T_1, T_2]$. In the proposed approach, training data is a set of input and output waveforms of the original nonlinear circuit. They can be collected through simulations and/or measurements. A second set of waveform data, called testing data should also be generated from the original circuit for model verification. The excitations used for generating testing waveforms should be different from those for generating training waveforms.

Let $y(t)$ represent the output response of RNN, and $\mathbf{y}(t)$ represent the output waveform of original nonlinear circuit, i.e. the simulation and/or measurements. Let training data be represented by input-output waveforms $(u_q(t), \mathbf{y}_q(t))$, $T_1 \leq t \leq T_2$, $q = 1, \dots, N_W$, where $u_q(t)$ and $\mathbf{y}_q(t)$ are the q th input and output waveforms and N_W is the total number of waveforms.

Each dynamic response $y(t)$ from RNN macromodel over the time interval $[T_1, T_2]$ is also called a RNN output trajectory. The process of training is to minimize the difference between RNN trajectory $y(t)$ and circuit waveform data $\mathbf{y}(t)$ under various input signals $u(t)$. Let N_t be the number of time samples in $[T_1, T_2]$, and $y_{iq}(k)$ be the i th output signal at the k th time sample in the q th trajectory of RNN and $y_{iq}(k)$ be the corresponding waveform sample of the original nonlinear circuit. The objective of the training is:

$$\min_{\bullet} \frac{1}{2} \sum_{q=1}^{N_W} \sum_{i=1}^{N_y} \sum_{k=1}^{N_t} (y_{iq}(k) - y_{iq}(k))^2 \quad (3.9)$$

3.5.2 Back Propagation Through Time (BPTT)

In order to employ efficient gradient based optimization methods in the macromodel training, the derivatives of the objective error function with respect to each parameter in the RNN are required to form the Jacobian matrix. As shown in Equation (3.8), the

outputs of RNN macromodel are fed back to the input layer. Therefore, output y is not only a function of input u , but also the previous output of itself. The conventional error back propagation method is not applicable for the RNN training. An advanced macromodel training scheme based on BPTT [86] is implemented in the following paragraphs to derive the Jacobian Matrix:

For the k th training sample in the q th waveform ($u_q(k)$, $y_q(k)$), the l_2 error of the macromodel is:

$$E_q(k) = \sum_{i=1}^{N_r} \frac{1}{2} \{y_{iq}(k) - y_{iq}(k)\}^2 \quad (3.10)$$

To obtain derivatives of $E_q(k)$ with respect to weights Φ for training, we need $dy_{iq}(k)/d\Phi$ which is computed recursively from the history of $dy_{iq}(k)/d\Phi$ in the following way:

For $k = 1$, initialize $\frac{dy_{iq}(k)}{d\Phi} = \frac{\partial y_{iq}(k)}{\partial \Phi}$, where the left hand side is the derivative of the RNN macromodel and $\frac{\partial y_{iq}(k)}{\partial \Phi}$ in the right hand side is the derivative of the FFNN part in the macromodel between x and y , which can be computed by normal back propagation [31]. The subsequent derivative $\frac{dy_{iq}(k)}{d\Phi}$ for $k > 1$ can be computed by using

the history of $\frac{dy_{iq}(k)}{d\Phi}$ as:

$$\frac{dy_{iq}(k)}{d\Phi} = \frac{\partial y_{iq}(k)}{\partial \Phi} + \sum_{m=1}^{L_y} \sum_{j=1}^{N_r} \left\{ \frac{\partial y_{jq}(k)}{\partial x_{[j+(m-1)N_r]}} \frac{dy_{jq}(k-m)}{d\Phi} \right\} \quad (3.11)$$

where

$$L_y = \begin{cases} K_y & \text{if } k > K_y \\ k-1 & \text{otherwise} \end{cases} \quad (3.12)$$

The recurrent back propagation includes two parts. First, normal back propagation is done through the FFNN between x layer and y layer to get the partial derivative $\partial y_{iq}(k)/\partial \Phi$. This represents the sensitivity of circuit output waveform with respect to macromodel internal parameters. Secondly, $\partial y_{iq}(k)/\partial x(k)$ is obtained from further back propagation to FFNN input layer x . This derivative represents the dependency of circuit output waveform between adjacent time points and can be written as:

$$\frac{\partial y_{iq}(k)}{\partial x_{[j+(m-1)N_y]}} = \sum_{r=1}^{N_x} \frac{dy_{iq}(k)}{dz_r} \frac{dz_r}{dx_{[j+(m-1)N_y]}} = \sum_{r=1}^{N_x} v_{ir} z_r (1 - z_r) w_{r[j+(m-1)N_y]} \quad (3.13)$$

for $j = 1 \dots N_y, \quad m = 1 \dots K_y$

The result of $dy_{iq}(k)/d\Phi$ is stored and used recursively as the history for computing the derivative at $k+1$ step. Based on this backpropagation scheme, gradient based optimization algorithms, such as Levenberg-Marquardt and quasi Newton method are used to train the macromodel [87]. Once a RNN macromodel is trained, it can then represent the parametric and dynamic input-output relationship of the original nonlinear microwave circuit.

3.6 Summary and Discussion

In conclusion, because the time factor is taken into consideration, compared to the conventional FFNN technique, the RNN technique differs in aspects of network structure, training procedure, and training data. The comparison between FFNN technique and RNN technique is illustrated in Table 3.1.

	Feedback Connection	Delay in Inputs and Outputs	Training Method	Training Data
FFNN Technique	No	No	Standard BP	Static independent data samples
RNN Technique	Yes	Yes	BPTT (Back Propagation Through Time)	Training data are grouped in waveforms. Data samples inside of each waveform are related each other

Table 3.1 Comparison between RNN technique and FFNN technique

The number of delay buffers represents the order of the macromodel. Theoretically to identify the complete dynamic characteristics, the number of delay should at least exceed the number of states in the original circuit. For the purpose of model reduction and macromodeling, a system with a much lower order can predict the output of the original system successfully [76][78]. By choosing K_y and K_u to be smaller than the order of the original system, the proposed RNN model automatically achieves model reduction effects. Another factor in the RNN model is the number of hidden neurons. Different number of hidden neurons represents the extent of nonlinearity between FFNN inputs and outputs (x and y). Too few hidden neurons can not represent the nonlinearity sufficiently, while too many will lead to overlearning.

RNN can be used to learn the dynamics both in transient stage and in steady state stage, as to be shown in examples in Chapter 4. When using RNN macromodel, a good

estimation of the initial state is necessary. For models trained using transient waveform, the initial state can be considered to be zero or DC solution. If in the transient stage the signal has sudden changes in the beginning, a separate RNN model with smaller sampling interval can be trained for initial state estimation. For models trained using steady-state waveform data. The initial states are not simply zeroes or constants. A Time Delay Neural Network (TDNN) [88] can be trained using the information contained in training data and used to provide initial estimations.

Chapter 4

RNN Macromodeling Examples

4.1 RFIC Power Amplifier Macromodeling Example

This example demonstrates the macromodeling of a RFIC power amplifier shown in Figure 4.1 through the proposed RNN approach. The choice of power amplifier is due to the fact that it is a key circuit in wireless-communication systems. The amplifier contains 8 NPN BJT transistors modeled by two internal HP-ADS nonlinear models Q34 and Q37 [89]. Input parameters for the RNN are the voltage waveform of amplifier input and the sampling intervals. Output of the RNN is the voltage waveform of the amplifier output. The structure of RNN macromodel is shown in Figure 4.2.

The first macromodel is constructed to represent the transient characteristics of the amplifier. The sampling intervals are changed with frequency so that 50 points per cycle are ensured. The training waveform is gathered by exciting the circuit with a set of signal frequencies ($f = 0.8, 1.0, 1.2$ GHz) and amplitudes ($A = 0.2, 0.3 \dots 1.3$ V). Testing data is generated using frequencies (0.9, 1.1 GHz) and amplitudes (0.25, 0.35 ... 1.25 volts) which are different from those used in training.

Figure 4.1: Power amplifier circuit to be represented by a RNN macromodel.

Figure 4.2: The structure of the RNN macromodel for power amplifier

Number of Hidden Neurons in z layer	Recurrent Training Error (3 buffers)	Recurrent Testing Error (3 buffers)
30	1.35e-2	1.43e-2
40	1.08e-2	1.11e-2
50	1.06e-2	1.04e-2
60	1.12e-2	1.19e-2

Table 4.1 Amplifier: Recurrent training and testing vs. different number of hidden neurons

Number of Buffers (K)	Recurrent Training Error	Recurrent Testing Error
1	3.11e-2	3.00e-2
2	1.81e-2	1.83e-2
3	1.06e-2	1.04e-2
4	9.10e-3	9.33e-3

Table 4.2 Amplifier: Comparison of recurrent model against different number of buffers

Training results and testing results are listed in Table 4.1. Different numbers of delay buffers K ($K = K_u = K_y$) are tried in RNN and results are shown in Table 4.2.

A set of sample test waveforms is shown in Figure 4.3 for the RNN at frequencies 0.9 and 1.1 GHz, and amplitudes 0.55 and 1.15 volts. The RNN macromodel can reproduce amplifier output accurately even when amplifier input waveform is different from those used in training. Moreover, RNN macromodel can generate similar output as original simulation with a much faster speed. The evaluation of 900 different sets of input-output waveforms takes 10 seconds by the RNN macromodel and 177 seconds by the original simulation.

A second macromodel is trained to learn the large signal behavior of the amplifier in the steady state. The sampling cycle is also proportional to frequency so that 50 points per cycle are ensured. The training waveform is generated from HP-ADS with the following set of excitation signal: $f = 0.8, 0.9 \dots 1.2$ GHz and $A = 0.2, 0.3 \dots 1.2$ volts. Testing data is gathered for two cycles using a set of samples different from those used in training ($f = 0.85, 0.95 \dots 1.15$ GHz and $A = 0.25, 0.35 \dots 1.15$ V). The overall accuracy (average test error of all test waveforms) is 0.728%.

Figure 4.4 shows the comparison of RNN with a set of sample test waveforms (with three cycles) excited at sample frequencies 0.95 and 1.05 GHz, and amplitudes 0.45 and 1.15 volts. As expected, the RNN macromodel can reproduce amplifier output accurately even though these test waveforms were never used in training. Moreover, the training was done with one cycle of waveform data. In the testing of Figure 4.4, the RNN can reliably predict the signal well beyond one cycle.

Figure 4.3: Comparison between output waveforms from original amplifier (o) and that from a RNN macromodel with 3 buffers (-) trained in transient state. Good agreement is achieved even though these waveforms have never been used in training

Figure 4.4: Comparison between output waveforms from original amplifier (o) and that from a RNN macromodel with 3 buffers (-) trained in steady state. Good agreement is achieved even though these waveforms have never been used in training

4.2 RF Mixer Macromodeling Example

This example demonstrates the RNN method in macromodeling a mixer, which is a basic circuit in any transmission-reception process. The circuit is a Gilbert cell shown in Figure 4.5. The internal sub-circuit contains 14 NPN BJT transistors (modeled by five different HP-ADS nonlinear models M1 to M5) [89]. The RNN macromodel for the mixer has three inputs, namely, RF input waveform, Local Oscillator input waveform and IF load impedance. The sampling cycle is fixed proportionally to the highest frequency to be trained.

The first macromodel is constructed to model the dynamics of the mixer in transient state. The training data is gathered in such way: RF frequency and power level are changed from 1.8 to 3.0 GHz with a step size of 0.1GHz and from -60 dBm to -30 dBm with a step size of 5 dBm respectively. LO signal is fixed at 1.75 GHz and -5 dBm. IF Load impedance is sampled at 30, 40, 50, 60 and 70 Ω . The transient time range for training $[T_1, T_2]$ is $[0, 1\text{ns}]$. Test data is given using different set of samples from those used in training: RF frequencies (2.35, 2.55, 2.75 GHz), RF power levels (-38, -43, -47, -48, -53 dBm) and for a 50 Ω load IF impedance with transient time up to 2 ns. The RNN macromodel structure is shown in Figure 4.6.

Figure 4.5: Mixer circuit to be modeled by a RNN macromodel.

Figure 4.6: The structure of the RNN macromodel for a mixer

The training error and testing error are listed in Table 4.3. Results of the RNN with different number of buffers K ($K = K_u = K_y$) are shown in Table 4.4. Figure 4.7 shows an example test result of RNN at the RF frequency 2.55 GHz, RF power level -48 dBm, IF load impedance of 50Ω and transient time up to 6 ns. Again, using this macromodel, the analog dynamic behavior can be retained and the evaluation of such behavior is much faster than original circuit simulation.

Number of Hidden Neurons in z layer	Recurrent Training Error (3 buffers)	Recurrent Testing Error (3 buffers)
30	7.77e-3	5.69e-3
40	5.53e-3	3.60e-3
50	6.03e-3	4.17e-3
60	6.40e-3	5.40e-3

Table 4.3 Mixer: Recurrent training and testing vs. different number of hidden neurons

Number of Buffers (K)	Recurrent Training Error	Recurrent Testing Error
1	8.81e-2	7.85e-2
2	2.59e-2	2.71e-2
3	6.03e-3	4.17e-3
4	8.21e-3	7.89e-3

Table 4.4 Mixer: Comparison of recurrent model against different number of buffers

Figure 4.7: Comparison between output waveforms from original mixer (o) and that from a RNN macromodel with 3 buffers (-) trained in transient state. Good agreement is achieved even though these waveforms have not been used in training. ($f_{RF} = 2.55$ GHz, $P_{RF} = -48$ dBm, $Z_{IF} = 50 \Omega$)

A second macromodel is constructed to learn the behavior of the mixer in steady state using the same RNN structure as the macromodel in transient state. Training data is generated as follows: RF frequency and power level are changed from 1.8 to 3.0 GHz with a step size of 0.1 GHz and from -50 dBm to -30 dBm with a step size of 5 dBm. LO signal frequency is 1.75 GHz and power level is -5 dBm. IF Load impedance is sampled at 30, 40, 50, 60 and 70 Ω , and time range is up to 4 ns. Test data is given for a different set of RF frequencies (1.85, 1.95 ... 2.95 GHz), RF power levels (-47, -42, -37 and -32 dBm) and for a load IF impedance sampled at 35, 45, 55 and 65 Ω with time range up to 12 ns. The overall test error for all the waveforms is 0.541%, the RNN macromodel can predict the output waveforms accurately even beyond the time range used in training.

Figure 4.8 shows an example test result of RNN with RF frequency equals 2.05 GHz, RF power level -47 dBm and IF load impedance of 35 Ω . Again, using the macromodel, the original analog dynamic behavior can be retained and the evaluation of such behavior is much faster than original circuit simulation.

Figure 4.8: Comparison between output waveforms from original mixer (o) and that from a RNN macromodel with 3 buffers (-) trained in steady state. Good agreement is achieved even though these waveforms have not been used in training. ($f_{RF} = 2.05$ GHz, $P_{RF} = -47$ dBm, $Z_{IF} = 35 \Omega$). This is one of the many test waveforms used

4.3 MOSFET Time Domain Macromodeling Example

The last example is a transient device level example of a p-MOSFET transistor. The training data is collected by HSpice simulation using BSIM3 (level 49) model [90]. The physical parameters of this model are length $L = 0.4 \mu\text{m}$ and width $W = 170 \mu\text{m}$.

The RNN macromodel has two inputs, namely, gate and drain voltage waveforms. The two outputs of the RNN are gate and drain current waveforms (I_g and I_d). To generate the training and testing waveform data, the circuit shown in Figure 4.9 is used. The structure of RNN macromodel is shown in Figure 4.10. Sampling intervals are proportional to the frequency so 50 points per cycle are ensured. The training data is gathered by varying frequencies of excitation signal: $f = 0.8, 1.0, 1.2, 1.8, 2.0, 2.2 \text{ GHz}$; source amplitudes: $V_{\text{source}} = 0.8, 1.0, 1.2, 1.4, 1.5, 1.6 \text{ V}$; gate bias voltages: $V_{g\text{dc}} = -3.0, -2.75, \dots -2.0 \text{ V}$; and drain bias voltages: $V_{d\text{dc}} = -5.0, -4.5 \dots -3.0 \text{ V}$ with transient up to 2 cycles. Testing data are waveforms simulated in HSpice using a set of samples different from those in training (frequency = $0.9, 1.1, 1.9, 2.1 \text{ GHz}$; source amplitudes: $V_{\text{source}} = 0.9, 1.1, 1.3, 1.45, 1.55 \text{ V}$; gate bias: $V_{g\text{dc}} = -2.875, -2.625 \dots -2.125 \text{ V}$; drain bias: $V_{d\text{dc}} = -4.75, -4.25 \dots -3.25 \text{ V}$) with transient up to two cycles.

A separate RNN macromodel is trained to estimate the initial conditions for the above model. The training data for this initial RNN includes short segments of various training waveforms sampled at a higher sampling rate. The sampling interval of the training data for this initial macromodel is one fifth of that in the main RNN macromodel. Three delay buffers for inputs and outputs are used. This initial RNN helps to accommodate the

Figure 4.9: The circuit used to generate training data for a BSIM3-level49 transistor to be represented by a RNN macromodel

Figure 4.10: The structure of the RNN macromodel for a BSIM3-level49 transistor

sudden rise and fall of the signal in the beginning. Figure 4.11 shows the response of RNN under the same excitation with and without initial estimations.

The overall accuracy (test error) of the macromodel with initial estimation with respect to all test waveforms is 0.303%. The RNN macromodel can predict both gate and drain currents accurately even when the exciting gate and drain voltage waveforms are different from what are used in training. Figures 4.12 and 4.13 show the examples of different test results with different sets of variables (with varying source amplitude, frequency, gate bias voltage and drain bias voltage, respectively). The output trajectory from the proposed macromodel matches the test waveform from the original MOSFET very well under various excitations even though these waveforms have never been used in training.

(a) Gate current: without initial state estimation

(b) Drain current: without initial state estimation

(c) Gate current: with initial state estimation by an initial RNN

(d) Drain current: with initial state estimation by an initial RNN

Figure 4.11: Effect of initial state estimation and comparison between output waveforms from original transistor simulation (o) and that from a RNN macromodel with 2 buffers (-) for $f = 0.9$ GHz, $V_{source} = 1.55$ Volt, $V_g = -2.125$ volts and $V_d = -3.25$ volts

(a) Different V_{source} with $f = 1.1$ GHz, $V_g = -2.125$ V and $V_d = -3.25$ V

(b) Different frequencies with $V_g = -2.125$ V, $V_d = -3.25$ V and $V_{\text{source}} = 1.55$ V

(c) Different V_g with $f = 1.1$ GHz, $V_d = -3.25$ V and $V_{source} = 1.55$ V

(d) Different V_d with $f = 1.1$ GHz, $V_g = -2.125$ V and $V_{source} = 1.55$ V

Figure 4.12: Comparison between output waveforms of gate current from original transistor (o) and that from a RNN macromodel with 2 buffers (-) under various excitation with different parameters. Initial estimation by an initial RNN was used. The macromodel matches test data very well even though those various test waveforms have never been used in training

(a) Different V_{source} with $f = 1.1$ GHz, $V_g = -2.125$ V and $V_d = -3.25$ V

(b) Different frequencies with $V_g = -2.125$ V, $V_d = -3.25$ V and $V_{source} = 1.55$ V

(c) Different V_g with $f = 1.1$ GHz, $V_d = -3.25$ V and $V_{source} = 1.55$ V

(d) Different V_d with $f = 1.1$ GHz, $V_g = -2.125$ V and $V_{source} = 1.55$ V

Figure 4.13: Comparison between output waveforms of drain current from original transistor (o) and that from a RNN macromodel with 2 buffers (-) under various excitation with different parameters. Initial estimation by an initial RNN was used. The macromodel matches test data very well even though those various test waveforms have never been used in training

4.4 Comparison between Standard Neural Network and The Proposed RNN Methods for Nonlinear Circuit Macromodeling

In order to compare the proposed recurrent neural model with the conventional non-recurrent neural network model, the three examples in this section are also used to develop conventional Feed Forward Neural Network (FFNN), and Time Delay Neural Network (TDNN) models. In FFNN, the output is dependent only on the inputs at the same instantaneous time, while in TDNN, only the history of input signals are used as the inputs of the macromodel. Both FFNN and TDNN are non-recurrent models since they have no feedback from output to input. The test errors for FFNN, TDNN and RNN models are listed in Table 4.5.

As can be observed from the table, the proposed RNN method gives the best modeling results. The conventional FFNN model has poor accuracy since it can only represent a static input-output relationship and is not suitable for representing the dynamic behavior in the examples. The TDNN model is an improvement over FFNN because the history of inputs is used in training, representing a partial dynamic information. However the circuit output may be different even with the same input, because of the differences in the history of the circuit responses. In this case, the non-recurrent models (FFNN and TDNN) will try to learn the average between different outputs when the inputs are the same, therefore cannot give accurate modeling solutions. The RNN approach takes the history of outputs as additional inputs, thus feeding the

RNN with richer information identifying differences between different dynamic states of the circuit, and giving the best modeling accuracy among all the methods.

	Standard Feed Forward Neural Network Method (FFNN)	Standard Time Delay Neural Network Method (TDNN)	Proposed Recurrent Neural Network Method (RNN)
Amplifier Macromodel	1.56e-1	3.42e-2	1.04e-2
Mixer Macromodel	9.61e-2	8.12e-2	3.60e-3
MOSFET Macromodel	6.37e-3	4.32e-3	3.03e-3

Table 4.5 Comparison of Test Errors Between Non-Recurrent and the Proposed Recurrent Models for Nonlinear Modeling

Chapter 5

Conclusions and Future Research

5.1 Conclusions

This thesis is motivated by the need of building macromodels for nonlinear circuits and systems and the popularity of using neural network technique for modeling microwave devices and circuits. Aiming to bridge the gap between neural network technique and dynamic nonlinear time-domain behavior modeling technique, a new macromodeling approach based on recurrent neural networks for nonlinear microwave circuits has been proposed. Compared to the existing macromodeling techniques, the proposed approach has following characteristics:

- It retains the analog time-domain behavior of the original nonlinear circuit because the macromodel is developed in time domain.
- Only input and output waveform measurements/simulations are needed for developing macromodels. It makes this technique applicable to the situation where not enough knowledge of the original circuit is available.

- No Equivalent circuit is used by the proposed technique. It thus avoids the trial and error process of developing equivalent circuits, which may be time and energy consuming.
- It is fast and accurate. The model development process is an automated optimization procedure, which requires minimum human efforts. The resulting model gives an accurate answer of the original circuit fast.
- It is generic and flexible. It can be applied to most of nonlinear circuits, which satisfies three assumptions narrated in section 3.1. Adjustable number of delays and hidden neurons makes this technique applicable to a big range of nonlinear systems.

Besides the formulation of the proposed new technique, a software module is implemented inside *Neuromodeler*[®], using object-oriented design and programming. The software module includes model generation, training and various utilities. A description of using this module is presented in Appendix A.

5.2 Suggestions for Future Directions

Neural networks have been proved as a very powerful modeling technique for microwave devices and systems recently. It has a promising future for all stages of RF/microwave circuits design. Various applications have been conducted on the modeling of passive components, but few have been done for modeling time-domain behavior of nonlinear devices and circuits. This thesis is a small progressive step of applying recurrent neural network to this area. It can be expected that more research and applications will appear in the future.

An interesting topic after the idea in the thesis would be applying continuous recurrent neural networks to model the continuous behavior of nonlinear circuits. The neural networks structures and training scheme used in this thesis are formulated in discrete time-domain. Yet the real world works continuously in time. Compare to the continuous case, any discrete scheme will inevitably lose some information. Formulating various kinds of continuous recurrent neural networks and applying them to RF/Microwave circuits modeling are expected to have a splendid future.

Another interesting topic would be the idea of incorporating existing dynamic knowledge of the original nonlinear circuit into recurrent neural networks. What has been done in the thesis treats the nonlinear circuit under modeling as a totally black box. But any knowledge of the original circuit should help recurrent neural networks to achieve higher accuracy with less effort. Existing knowledge of nonlinear circuit can be collected in the form of states, initial values, or extra measurements, etc. How to formulate the new structure of recurrent neural network and develop the training schemes for them will be totally a new research topic.

Appendix A

Developing RNN Macromodels Using *NeuroModeler*[®] Software Package

For the convenience of using the proposed technique, along with the thesis, a complete software module has been implemented in *NeuroModeler*[®], which is recognized as a well-known software building neural networks models for RF/Microwave design. Input/output based RNN can be generated, trained, tested and outputted in this self-contained software. All the examples in this thesis were trained and tested using this software package. To be compatible with the convention of the software, the new module is programmed in C++ and Java applying object oriented concepts.

Feed forward neural networks has been extensively explored and implemented in *NeuroModeler*[®]. Because the part between input layer and output layer in the input/output based RNN is a feed forward neural network, what has been implemented in *NeuroModeler*[®] was effectively utilized for building RNN models. Which includes structure description, feed forward and backpropagation calculation, standard l_2 optimization algorithms, etc. For the time being, only the generally used MLP has been utilized as the feed forward part of RNN. In this appendix, materials needed to build an input/output RNN models in *NeuroModeler*[®] are described. Namely, they are the ‘type’

file, the 'structure' file and the 'data' file. For more information on how to use the software, please visit website <http://www.doe.carleton.ca/~qjz>.

A.1 The 'type' file

It is the 'type' file that distinguishes the RNNs from the existing feed forward neural networks. The 'type' file contains the abstract information of the RNN structures, which is used in formulating the detailed 'structure' file later. The file should be an ASCII text file. The format of a typical RNN 'type' file is shown in Table A.1.

Type of neural network
Number of parameters
Number of dynamic inputs
Number of static inputs
Number of hidden neurons
Number of delays of dynamic inputs
Number of delays of outputs
Number of outputs

Table A.1: The format of the 'type' file for an input/output RNN

The first line of the 'type' file indicates what type of neural network this one belongs to. The candidate type can be MLP3, KBNN, General or RNN. For recurrent neural networks, the keyword there should be RNN. The following line indicates how many

parameters are there in the ‘type’ file. Corresponding to the time-variant and time-invariant inputs to RNN, line three and four tell the total number of dynamic inputs and the total number of static inputs. Line five gives the number of hidden neurons in the hidden layer in the feed forward part. The number of delays for time-variant inputs and outputs are given in line six and seven respectively. The last line is the number of model outputs. This file can be generated automatically by creating a new RNN in *NeuroModeler*[®]. It can also be created manually by user.

Based on the information given in ‘type’ file, the feedback process can be done externally to the feed forward calculation and the Back Propagation Through Time scheme can be finished prior to l_2 optimization algorithms. These enable the overall procedure of RNN model development compatible to the existing procedure in *NeuroModeler*[®].

A.2 The ‘structure’ file

The ‘structure’ file contains the detailed structure information of the neural model. In order to keep the compatibility of RNN to the existing feed forward neural networks, the ‘structure’ file of RNN is similar to the one that is already used in *NeuroModeler*[®]. Except the number of inputs needs to be adjusted because of the delays and feedback. A template ‘structure’ file is shown in Table A.2. It also should be saved as an ASCII text file.

The first and second lines are lists of input and output neuron labels along with their total numbers. The total number of input neurons should be equal to the sum of static inputs to RNN, the number of time-variant inputs multiplied with the number of delays,

```

NumInputNeurons InputNeuron1_Label InputNeuron2_Label ...
NumOutputNeurons OutputNeuron1_Label OutputNeuron2_Label ...
NumTotalNeurons

Neuron1_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ...
NumOutConnections ToNueron1_Label ToNeuron2_Label ...
ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ...

Neuron2_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ...
NumOutConnections ToNueron1_Label ToNeuron2_Label ...
ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ...

.....

NeuronN_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ...
NumOutConnections ToNueron1_Label ToNeuron2_Label ...
ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ...

Neuron_Label Neuron_Label Neuron_Label ...

```

Table A.2: The format of the ‘structure’ file for an input/output RNN

and the number of outputs multiplied with the number of delays. Following them in the third line is the total number of neurons in RNN. The following are the descriptions of individual neurons. Each neuron is identified by its label and described by the interconnection with other neurons combined with the activation function and a set of parameters for that activation function. InConnections is the list of neurons whose outputs will be the input of this neuron, and OutConnections is the list of neurons that will take the output of this neuron. The last line of the file is the list of neuron labels describing the processing sequence of neurons during feed forward stage. The sequence of these neurons should be compatible to the directed graph that is composed by the neurons as nodes and the interconnections between neurons as edges. The 'structure' file can be generated automatically when a new RNN model is created in *NeuroModeler*[®].

A.3 The 'data' file

The training and testing 'data' file is the most commonly used data file formatted with columns in an ASCII file. But one thing needs to be mentioned is that the data used to train and test the RNN models are grouped in the concept of waveforms instead of individual data points. Data points inside one waveform are not independent to each other because they are related by time sequence. A special data line is used to separate different waveforms. It is characterized with 9999.99 for all the output data. Inside of each line, the first places are reserved for the feedback of previous outputs. For time-variant inputs, the sequence of delayed inputs in each line is required to be consistent among different waveforms.

The format of 'data' file is shown in Table A.3. Sign "\\" means the continuation of the lines before and after it. Sign "%" begins a line of comments.

```

%Beginning of a waveform
0 0 ... 0 0 0 ...0 InvariantInput_1 ... InvariantInput_J 9999.99 ... 9999.99

%Second line
Output1(0) Output2(0) ... 0 VariantInput_1(0) VariantInput_2(0) ...0 InvariantInput_1
\\... InvariantInput_J Output1(1) ... OutputN(1)
.....

Output1(t-1) Output2(t-1) ... OutputN(t-M) VariantInput_1(t-1) VariantInput_2(t-1)
\\...VariantInputP(t-Q) InvariantInput_1 ... InvariantInput_J Output1(t) ... OutputN(t)

%Beginning of a new waveform
0 0 ... 0 0 0 ...0 InvariantInput_1 ... InvariantInput_J 9999.99 ... 9999.99
.....

```

Table A.3: Format of RNN ‘data’ file

A.4 An illustrating example

The RNN macromodel for the power amplifier in Section 4.1 is used as an illustrating example for understanding. It has one time-variant input, one time-invariant input and one output. The model structure with 15 hidden neurons and 1 delay for both time-variant input and output will have the ‘structure’ file shown in Table A.4 and A.5. The ‘type’ file is shown in Table A.6.

```

3  1 2 3
1  19
19
1
1  -1
15  4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Relay 0
2
1  -2
15  4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Relay 0
3
1  -3
15  4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Relay 0
4
3  1 2 3
1  19
Sigmoid 4  1.08  -1.99  -0.05  2.55
5
3  1 2 3
1  19
Sigmoid 4  -0.91  -0.05  0.30  0.09
6
3  1 2 3
1  19
Sigmoid 4  0.23  -0.53  -0.80  0.25
7
3  1 2 3
1  19
Sigmoid 4  0.46  0.67  -0.14  2.28
8
3  1 2 3
1  19
Sigmoid 4  1.46  0.81  0.27  0.14
9
3  1 2 3
1  19
Sigmoid 4  0.01  1.00  0.28  0.07
10
3  1 2 3
1  19

```

Table A.4: An example RNN macromodel 'structure' file for power amplifier

RNN
6
1
1
15
1
1
1

Table A.6: An example RNN macromodel ‘type’ file for power amplifier

References

- [1] Y. Fang, M. C. E. Yagoub, F. Wang and Q. J. Zhang, "A New Macromodeling Approach for Nonlinear Microwave Circuits Based on Recurrent Neural Networks", *IEEE Trans. Microwave Theory Tech.*, vol. 48, pp. 2335-2344, 2000.
- [2] Q. J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*, Artech House, Boston, MA, 2000.
- [3] P. Burrascano and M. Mongiardo, "A review of artificial neural networks applications in microwave CAD," *Int. J. RF Microwave CAE, Special Issue on Appl. of ANN to RF and Microwave Design*, vol. 9, pp. 158-174, 1999.
- [4] K. C. Gupta, "Emerging trends in millimeter-wave CAD," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 747-755, 1998.
- [5] F. Wang, V. K. Devabhaktuni, C. G. Xi, and Q. J. Zhang, "Neural Network structures and training algorithms for RF and microwave applications," *International Journal of RF and Microwave Computer-Aided Engineering: Special Issue on Applications of Neural Networks to RF and Microwave Design*, vol. 9, 1999.
- [6] V. K. Devabhaktuni, M.C.E. Yagoub, Y. Fang, J. Xu and Q.J. Zhang, "Neural Networks for Microwave Modeling: Model Development Issues and Nonlinear

- Modeling Techniques," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 11, pp. 4-21, 2001 (invited).
- [7] F. Wang and Q.J. Zhang, "Knowledge based neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2333-2343, 1997.
 - [8] F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "A hierarchical neural network approach to the development of a library of neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 2391-2403, 1998.
 - [9] G. Antonini and A. Orlandi, "Gradient evaluation for neural networks based electromagnetic optimization procedures," *IEEE Trans. Microwave Theory Tech.*, vol. 48, pp. 874-876, 2000.
 - [10] Q. J. Zhang, F. Wang, and M. S. Nakhla, "Optimization of high-speed VLSI interconnects: A review," *Int. J. Microwave Millimeter-Wave CAE*, vol. 7, pp. 83-107, 1997
 - [11] T. Horng, C. Wang, and N. G. Alexopoulos, "Microstrip circuit design using neural networks," *IEEE MTT-S Int. Microwave Symp. Dig.*, Atlanta, GA, 1993, pp. 413-416.
 - [12] P. M. Watson and K. C. Gupta, "EM-ANN models for microstrip vias and interconnects in dataset circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 44, pp. 2495-2503, 1996.
 - [13] P. M. Watson, K. C. Gupta and R. L. Mahajan, "Applications of knowledge-based artificial neural network modeling to microwave components," *Int. J. RF. And Microwave CAE*, vol. 9, pp. 254-260, May 1999.

- [14] J. W. Bandler, M. A. Ismail, J. E. Rayas-Sanchez, and Q. J. Zhang, "Neuromodeling of microwave circuits exploiting space-mapping technology," *IEEE Trans. Microwave Theory Tech.*, vol. 47, pp. 2417-2427, December 1999.
- [15] P. M. Watson and K. C. Gupta, "Design and optimization of CPW circuits using EM-ANN models for CPW components," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2515-2523, 1997.
- [16] P. M. Watson, G. L. Creech and K. C. Gupta, "Knowledge based EM-ANN models for the design of wide bandwidth CPW patch/slot antennas," *IEEE APS Int. Symp. Dig.*, Orlando, FL, July 1999, pp. 2588-2591.
- [17] G. L. Creech, B. J. Paul, C. D. Lesniak, T. J. Jenkins, and M. C. Calcaterra, "Artificial neural networks for fast and accurate EM-CAD of microwave circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 794-802, 1997.
- [18] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "A neural network modeling approach to circuit optimization and statistical design," *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1349-1358, 1995.
- [19] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "Device and circuit-level modeling using neural networks with faster training based on network sparsity," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 1696-1704, 1997.
- [20] Y. Harkouss, J. Rousset, H. Chehade, D. Barataud, and J. P. Teyssier, "The use of artificial neural networks in nonlinear microwave devices and circuits modeling: an application to telecommunication system design," *Int. Journal*

- of RF and Microwave CAE*, Special Issue on Applications of ANN to RF and Microwave Design, vol. 9, pp. 198-215, 1999.
- [21] K. Shirakawa, M. Shimizu, N. Okubo and Y. Daido, "A large signal characterization of an HEMT using a multilayered neural network," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 1630-1633, 1997.
 - [22] K. Shirakawa, M. Shimizu, N. Okubo and Y. Daido, "Structural determination of multilayered large signal neural-network HEMT model," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 1367-1375, 1998.
 - [23] A. Veluswami, M. S. Nakhla and Q. J. Zhang, "The application of neural networks to EM-based simulation and optimization of interconnects in high speed VLSI circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 712-723, May 1997.
 - [24] S. Wang, F. Wang, V. K. Devabhaktuni and Q. J. Zhang, "A hybrid neural and circuit-based model structure for microwave modeling," *Proc. 30th European Microwave Conf.*, Munich, Germany, Oct. 1999, pp. 174-177.
 - [25] C. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Systems*, vol. 2, pp. 303-314, 1989.
 - [26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.
 - [27] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Networks*, vol. 8, pp. 630-645, 1997.

- [28] R. Reed, "Pruning algorithms – a survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740-747, September 1993.
- [29] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *IEEE Trans. Neural Networks*, vol. 4, pp. 136-141, 1992.
- [30] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: four layer versus three," *IEEE Trans. Neural Networks*, vol. 8, pp. 251-255, 1997.
- [31] S. Haykin, *Neural Networks: A comprehensive foundation*, second edition, Prentice Hall, Upper Saddle River, NJ, 1999.
- [32] S. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281-294, 1989.
- [33] J. Park and I. W. Sandberg, "Universal approximation using Radial-Basis-Function networks," *Neural Computation*, vol. 3, pp. 246-257, 1991.
- [34] J. Park and I. W. Sandberg, "Approximation and Radial-Basis-Function networks," *Neural Computation*, vol. 5, pp. 305-316, 1991.
- [35] A. Krzyzak, T. Linder, and G. Lugosi, "Nonparametric estimation and classification using radial basis function nets and empirical risk minimization," *IEEE Trans. Neural Networks*, vol. 7, pp. 475-487, Mar. 1996.
- [36] Q. H. Zhang, "Using wavelet network in nonparametric estimation," *IEEE Trans. Neural Networks*, vol. 8, pp. 227-236, 1997.
- [37] Q. H. Zhang and A. Benvensite, "Wavelet networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 889-898, 1992.

- [38] R. Bakshi and G. Stephanopoulos, "Wave-Net: a multiresolution, hierarchical neural network with localized learning," *America Institute of Chemical Engineering Journal*, vol. 39, pp. 57-81, 1993.
- [39] B. Delyon, A. Juditsky, and A. Benveniste, "Accuracy analysis for wavelet approximations," *IEEE Trans. Neural Networks*, vol. 6, pp. 332-348, Nov. 1995.
- [40] P. H. Ladbrooke, *MMIC Design: GaAs FET's and HEMTs*, Artech House, Boston, 1989.
- [41] H. Leung and S. Haykin, "Rational function neural network," *Neural Computation*, vol. 5, pp. 928-938, 1993.
- [42] A. Abdelbar and G. Tagliarini, "Honest: A new high order feedforward neural networks," *Proc. IEEE Intl. Conf. Neural Networks*, pp. 1257-1262, Washington DC, June 1996.
- [43] J. S. R. Jang, C. T. Sun, E. Mizutani, "Derivative-based Optimization", *Neuro-Fuzzy and Soft Computing, A Computational Approach to Learning and Machine Intelligence*, pp. 129-172, 1997.
- [44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume I, pp. 318-362, MIT Press, Cambridge, Massachusetts, 1986.
- [45] K. Ochiai, N. Toda, and S. Usui, "Kick-out learning algorithm to reduce the oscillation of weights," *Neural Networks*, vol. 7, pp. 797-807, 1994.

- [46] S. J. Perantonis and D. A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, vol. 8, pp. 237-249, 1995.
- [47] *Neural Network Toolbox, For Use with Matlab*, The MathWorks Inc., Natick, Massachusetts, 1993.
- [48] R. A. Jacobs, "Increased rate of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.
- [49] M. Arisawa and J. Watada, "Enhanced back-propagation learning and its application to business evaluation," *In Proc. IEEE Intl. Conf. Neural Networks*, vol. I, pp. 155-160, Orlando, Florida, July 1994.
- [50] A. G. Parlos, B. Fernandez, A. F. Atiya, J. Muthusami, and W. K. Tsai, "An accelerated learning algorithm for multilayer perceptron networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 493-497, May 1994.
- [51] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida, "A hybrid algorithm for finding the global minimum of error function of neural networks and its applications," *Neural Networks*, vol. 7, pp. 1253-1265, 1994.
- [52] P. P. Van-Der-Smagt, "Minimization methods for training feedforward neural networks," *Neural Networks*, vol. 7, pp. 1-11, 1994.
- [53] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge, Massachusetts: Cambridge University Press, 1986.

- [54] R. L. Watrous, "Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization," In *Proc. IEEE First Intl. Conf. Neural Networks*, vol. II, pp. 619-627, San Diego, California, 1987.
- [55] R. Battiti, "First and second-order methods for learning: between steepest descent and Newton's method," *Neural Computation*, vol. 4, pp. 141-166, 1992.
- [56] M. F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525-533, 1993.
- [57] T. R. Cuthbert(Jr.), *Optimization using Personal Computers, chapter Quasi-Newton Methods and Constraints*, pp. 233-314. John Wiley and Sons, New York, NY, 1987.
- [58] P. Gill, W. Murray and M. Wright, *Practical Optimization*, Academic Press, London, England, 1981.
- [59] K. R. Nakano, "Partial BFGS update and efficient step-length calculation for three layer neural network," *Neural Computation*, vol. 9, pp. 123-141, 1997.
- [60] S. Mcloone and G. W. Irwin, "Fast parallel off-line training of multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 8, pp. 646-653, 1997.
- [61] A. J. Shepherd, "Second-order optimization methods," *Second-order Methods for Neural Networks*, Berlin, NY: Springer-Verlag, 1997, pp. 43-72.
- [62] S. Kollias and D. Anastassiou, "An adaptive least squares algorithm for the efficient training of artificial neural networks," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 1092-1101, 1989.

- [63] M. G. Bello, "Enhanced training algorithms, and integrated training/ architecture selection for multilayer perceptron networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 864-875, Nov. 1992.
- [64] G. Zhou and J. Si, "Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency," *IEEE Trans. Neural Networks*, vol. 9, pp. 448-453, May 1998.
- [65] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [66] A. J. F. van Rooij, L. C. Jain, R. P. Johnson, *Neural Network Training Using Genetic Algorithms*, World Scientific, 1996.
- [67] T. Rognvaldsson, "On Langevin updating in multilayer perceptron," *Neural Computation*, vol. 6, pp. 916-926, 1994.
- [68] R. Brunelli, "Training neural nets through stochastic minimization," *Neural Networks*, vol. 7, pp. 1405-1412, 1994.
- [69] T. R. Turlington, *Behavioral Modeling of Nonlinear RF and Microwave Devices*, Boston, MA: Artech House, 2000.
- [70] J. Verspecht, F. Verbeyst, M. Vanden Bossche, and P. Van Esch, "System level simulation benefits from frequency domain behavioral models of mixers and amplifiers," in *Proc. 29th European Microwave Conf.*, Munich, vol. 2, pp. 29-32, 1999.
- [71] J. Verspecht, D. Schreurs, A. Barel and B. Nauwelaers, "Black box modeling of hard nonlinear behavior in the frequency domain," *Conference Record of*

the IEEE Microwave Theory and Tech. Symp. 1996, San Francisco, pp. 1735-1738, June 1996.

- [72] J. Verspecht and P. V. Esch, "Accurately characterizing hard nonlinear behavior of microwave components with the nonlinear network measurement system: introducing 'nonlinear scattering functions'", *Proceedings of the 5th International Workshop on Integrated Nonlinear Microwave and Millimeterwave Circuits*, pp. 17-26, Duisburg, Germany, October 1998.
- [73] T. H. Wang and T. J. Brazil, "A Volterra mapping-based S-parameter behavioral model for nonlinear RF and microwave circuits and systems," in *IEEE Int. Microwave Symp. Digest*, Anaheim, CA, pp. 783-786, 1999.
- [74] T. H. Wang and T. J. Brazil, "A mixed domain modeling method for microwave nonlinear systems and semiconductor devices in high frequency applications", *28th European Microwave Conference*, pp. 129-134, Amsterdam, Netherlands, 1998.
- [75] R. Boyle, B. M. Cohn, D. O. Pederson, and J. E. Solomon, "Macromodeling of integrated operational amplifiers," *IEEE J. Solid-State Circuits*, vol. 9, pp. 353-363, 1974.
- [76] G. Casinovi and A. Sangiovanni-Vincentelli, "A macromodeling algorithm for analog circuits," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 150-160, 1991.
- [77] M. I. Sobhy, E. A. Hosny, M. W. R. Ng, and E. A. Bakkar, "Nonlinear system and subsystem modeling in time domain," *IEEE Trans. Microwave Theory Tech.*, vol. 44, pp. 2571-2579, 1996.

- [78] P. K. Gunupudi and M. S. Nakhla, "Model-reduction of nonlinear circuits using Krylov-space techniques," in *Proc. IEEE Int. Design Automation Conf.*, New Orleans, Louisiana, pp. 13-16, June 1999.
- [79] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*, New York: Van Nostrand Reinhold, 1983.
- [80] I. J. Leontaritis and S. A. Billings, "Input-output parametric models for nonlinear systems, Part I: deterministic non-linear systems," *Int. J. Control*, vol. 41, NO. 2, pp. 303-328, 1985.
- [81] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4-27, 1990.
- [82] J. Aweya, Q. J. Zhang, and D. Y. Montuno, "A direct adaptive neural controller for flow control in computer networks," in *Proc. IEEE Int. Neural Networks Conf.*, Anchorage, Alaska, pp. 140-145, May 1998.
- [83] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks-part II: observability, identification, and control," *IEEE Trans. Neural Networks*, vol. 7, pp. 30-42, 1996.
- [84] T. Hrycej, *Neurocontrol: Towards An Industrial Control Methodology*, New York: Wiley & Sons, 1997.
- [85] S. Omatu, M. Khalid, and R. Yusof, *Neuro-Control and Its Applications*, London: Springer-Verlag, 1996.
- [86] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: a survey," *IEEE Trans. Neural Networks*, vol. 6, pp. 1212-1228, 1995.

- [87] *NeuroModeler Version 1.02*, Prof. Q.J. Zhang, Department of Electronics, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada.
- [88] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time delay neural networks", *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 37, pp. 328-339, 1989.
- [89] *HP-ADS Version 1.3*, Agilent Technologies, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403, U.S.A.
- [90] *HSPICE*, Avant! Corporation, Fremont, CA 94538, U.S.A.