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.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

Optimization Approaches to the Training of Neural Networks with RF/Microwave Applications

by

Changgeng Xi, B. Eng., M. Eng.,

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

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Electronics
Carleton University
Ottawa, Ontario K1S 5B6
Canada

© 1999, Changgeng Xi

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.

0-612-48465-3

Canada

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

**Optimization Approaches to the Training of Neural Networks with RF/Microwave
Applications**

submitted by Changgeng Xi, B. Eng., M. Eng.,
in partial fulfillment of the requirements for
the degree of Master of Engineering

A handwritten signature in black ink, appearing to read 'Q.J. Zhang', written over a horizontal line.

Q.J. Zhang, Thesis Supervisor

A handwritten signature in black ink, appearing to read 'J.S. Wight', written over a horizontal line.

J.S. Wight, Chair, Department of Electronics

Abstract

This work is motivated by the increasing recognition of the neural networks as an emerging modeling technique in microwave area. Appropriate training algorithms are very important in developing neural network models for microwave applications. They decide the amount of training data required, the accuracy that could possibly be achieved, and more importantly the developmental cost of neural models. In this thesis a comprehensive set of training methods suitable for microwave applications has been developed through optimization approaches. These training methods, including quasi Newton method, Huber quasi Newton method, Simplex method, Simulated Annealing algorithm and Genetic algorithm are developed for different training tasks or problems. Accordingly the advantages of the methods upon applications are demonstrated through microwave examples with various neural network structures such as standard feedforward networks, the structures with prior knowledge and wavelet networks.

To especially address the problem of inevitable gross errors in microwave training data, a robust and efficient training algorithm, namely Huber quasi Newton method (HQN) is proposed in this thesis. This algorithm is formulated by combining the Huber concept with the quasi Newton method. Huber concept has been verified to be robust against gross errors and quasi Newton method is considered as the most effective

nonlinear optimization method for general problems. The robustness and efficiency of the proposed training algorithm are confirmed by examples in two types of training applications. The first application shows its robustness against gross errors in the training data. The second application uses HQN to learn the smooth behavior of the training data so as to isolate the sharp variation by subtracting the trained model from the training data.

Acknowledgement

Firstly I would like to take this opportunity to thank my supervisor Professor Qi-Jun Zhang for his constant supervision and assistance in the completion of this thesis and the degree program. My grateful thanks also go to Professor Kaj Madsen of the Technical University of Denmark and Professor John Bandler of the McMaster University, for their invaluable information on the research work.

I would like to thank my colleagues Vijaya Kumar Devabhaktuni and Dr. Fang Wang for their suggestion and cooperation. Especially for Fang, her excellent programming has created an admirable supporting system for me to implement and test my training methods. Thanks as well to all the staff and fellow graduate students in the department of Electronics for their assistance and friendship.

Finally my deepest appreciation goes to my mother. Without her endless support and encouragement this thesis would not be possible.

Contents

Chapter 1	Introduction	1
1.1	Motivation of the Research	1
1.2	Organization of the Thesis.....	3
Chapter 2	Overview	6
2.1	Problem Statement.....	6
2.2	Standard Feedforward Neural Networks	8
2.2.1	Multilayer Perceptrons (MLP)	12
2.2.2	Radial Basis Function Networks (RBF).....	13
2.2.3	Knowledge-Based Neural Networks (KBNN).....	15
2.2.4	Wavelet Neural Networks	20
2.3	Neuron Sorting Algorithm.....	21
2.3.1	Topological Sorting.....	22
2.3.2	Loop Check	23
2.3.3	Neural Network Structure Identification.....	24
2.4	Training Algorithms	24
2.4.1	Training Objective.....	24
2.4.2	Review of Backpropagation Algorithm	25

2.4.3 Gradient-based Optimization Methods	27
2.4.3.1 Newton's Method.....	30
2.4.3.2 Conjugate Gradient Method	31
2.4.3.3 Levenberg-Marquardt and Gauss-Newton Training Algorithms	32
2.4.3.4 Line Search	34
Chapter 3 Quasi Newton Method	36
3.1 Introduction	36
3.2 Mathematical Formulation	37
3.2.1 Update Formula of Quasi Newton Methods	37
3.2.2 Steps of Quasi Newton Methods.....	41
3.2.3 Hereditary Positive Definiteness Condition.....	41
3.3 Examples	45
Chapter 4 Huber Quasi Newton Method	49
4.1 Introduction	49
4.2 Formulation of the Huber Training Method.....	51
4.3 Examples	54
4.3.1 A Quadratic Model.....	55
4.3.2 A Stripline Model.....	58
4.3.3 Heterojunction Bipolar Transistor (HBT) Model.....	62
4.3.4 MESFET Example	66
4.3.5 Microstrip Filter	70
Chapter 5 Simplex Method	75
5.1 Mathematical Description	75

5.2	Flow Diagram of Simplex Training Method	78
5.3	Example	81
Chapter 6	Simulated Annealing Algorithm	83
6.1	Introduction	83
6.2	Description of Simulated Annealing Algorithm.....	84
6.3	Examples	90
Chapter 7	Genetic Algorithm	98
7.1	Description of Genetic Algorithm	98
7.1.1	Major Components of GA.....	99
7.1.2	Flow Diagram of Genetic Algorithm	102
7.2	Examples	106
Chapter 8	Summary and Future Work	109
8.1	Comparison of Different Training Algorithms.....	109
8.2	Conclusions	110
8.3	Future Work.....	111
References		113

List of Figures

2.1	The representation of i^{th} neuron in a general feed-forward structure	9
2.2	Knowledge Based Neural Network (KBNN) structure	19
3.1	Three conductor microstrip line.....	46
4.1	The Huber, l_1 and l_2 functions.....	52
4.2	Comparison between neural model and training data of quadratic example: (a) Case 1:Neural model trained by the l_2 methods. Model is affected by the bad data points. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method. Model is not affected by the bad data points.	57
4.3	Comparison between neural model and training data of Stripline example, data samples with negative inductance values are bad samples: (a) Case 1: Neural model trained by l_2 methods. Model is affected by the bad data samples. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method. Model is not affected by the bad data samples.....	60

4.4	Model display comparison of Stripline example: (a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.....	61
4.5	Comparison between neural model and training data of HBT example: The training data has large errors at the beginning of every frequency sweep. (a) Case 1: Neural model trained by standard l_2 method is affected by large errors. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method is not affected by large errors.	64
4.6	Model display comparison of HBT example:(a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.....	65
4.7	Comparison between neural model and training data of MESFET example: The training data has a few large/gross errors. (a) Case 1: Neural model trained by standard l_2 method is affected by large errors. (b) Case 2: Neural model trained by the proposed HQN technique is not affected by large errors.	68
4.8	Model display comparison of MESFET example:(a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.....	69
4.9	A comparison between neural model and training data for two out of the thirty-six microstrip filters after the first stage training using HQN method. (a) and (b) filter with normalized values of w_d , g_{12} , and g_{23} as	

0.5, 2.0 and 2.0 respectively; (c) and (d) filter with normalized values of w_d , g_{12} , and g_{23} as 1.0, 1.5 and 2.0 respectively.....	73
4.10 A comparison between final neural model and training data for two out of the thirty-six microstrip filters modeled by the proposed IMS approach. (a) and (b) filter with normalized values of w_d , g_{12} , and g_{23} as 0.5, 2.0 and 2.0 respectively; (c) and (d) filter with normalized values of w_d , g_{12} , and g_{23} as 1.0, 1.5 and 2.0 respectively.....	74
5.1 Flow diagram of simplex method	80
5.2 Comparison between neural model and training data for the FET example	82
6.1 Flow diagram of the Simulated Annealing Algorithm	89
6.2 A two-dimensional illustration of the polynomial function.....	91
6.3 A one-dimensional plot of a quadratic function superimposed with an AM modulated sinusoidal function.....	92
6.4 A two-dimensional illustration of the nonlinear function.....	92
6.5 Training result from local training methods for the wavelet network example with one hidden neuron	95
6.6 Training curve of the Simulated Annealing Algorithm for the wavelet network example with one hidden neuron	95
6.7 Training result from the Simulated Annealing Algorithm for the wavelet network example with one hidden neuron	96
6.8 Training result from local training methods for the wavelet network example with two hidden neurons	96

6.9	Training curve of the Simulated Annealing Algorithm for the wavelet network example with two hidden neurons	97
6.10	Training result from the Simulated Annealing Algorithm for the wavelet network example with two hidden neurons	97
7.1	Roulette wheel chart	104
7.2	Crossover operators: (a) one-point crossover. (b) two-point crossover.....	104
7.3	Flow diagram of the Genetic Algorithm.....	105
7.4	Training result from the Genetic Algorithm for the wavelet network example with one hidden neuron	107
7.5	Training result from the Genetic Algorithm for the wavelet network example with two hidden neurons	107
7.6	Training curve of the Genetic Algorithm for the wavelet network example with one hidden neuron	108
7.7	Training curve of the Genetic Algorithm for the wavelet network example with two hidden neuron.....	108

List of Tables

3.1	Comparison of Various Training Algorithms for Microstrip Line Example.	46
3.2	Comparison of Various Training Algorithms for MESFET Example..	48
4.1	Strategy of Comparison between Huber quasi-Newton Method and l_2 Methods	54
4.2	Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for Quadratic Example	56
4.3	Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Method for Stripline Example.....	59
4.4	Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for HBT Example	63
4.5	Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for MESFET Example.....	67
4.6	Stage-Wise Training Results for S_{11} of Microstrip Filter Example Using the Proposed IMS Training Approach.	72
5.1	Comparison of Different Training Algorithms for the FET Example..	82

Chapter 1

Introduction

1.1 Motivation of the Research

The drive for manufacturability-oriented design and reduced time-to-market in the microwave industry require design tools that are accurate and fast. Yield analysis and optimization with detailed physics/EM models of active/passive components can be an important step towards a design for first-pass success, but it is computationally intensive. In the recent years a novel CAD approach based on neural network technology has been introduced in the microwave community, for the modeling of passive and active microwave components [1][2][3][4][5], and microwave circuit design [2][4][6][7]. A neural network model for a device/circuit can be developed by learning and abstracting from measured/simulated microwave data, through a process called training. Once trained, the neural network model can be used during microwave design to provide instant answers to the task it learnt [1]. Recent work by microwave researchers demonstrated the ability of neural networks to accurately model a variety of microwave components, such as microstrip interconnects [1][3][8], vias [3][9], spiral inductors [5][10], FET devices [1][11][12][13], power transistors and power amplifiers [14],

coplanar waveguide (CPW) circuit components [4], packaging and interconnects [15] etc. Neural networks have been used in circuit simulation and optimization [2][13][16], signal integrity analysis and optimization of VLSI interconnects [8][15], microstrip circuit design [17], microwave filter design [18], IC modeling [11], process design [19], synthesis [6], Smith Chart representation [7] and microwave impedance matching [20]. The neural network technologies have been applied to microwave circuit optimization and statistical design with neural network models at both device and circuit levels [2][12]. These pioneering work helped to establish the framework of neural modeling technology in microwave applications. Neural models are much faster than original detailed EM/physics models [1][2], more accurate than polynomial and empirical models [21], allow more dimensions than table lookup models [22] and are easier to develop when a new device/technology is introduced [16].

Training algorithms are an integral part of neural network model development. An appropriate structure may still fail to give a better model, unless trained by a suitable training algorithm. A good training algorithm will shorten the training time, while achieving a better accuracy. In the microwave applications most of the modeling problems are regression or function approximation problems, where the relationship is static. Neural networks with feedforward structure are known to have the ability of learning such relationship and the training of such neural network can be formulated as an unconstrained nonlinear optimization problem. However in microwave applications some specific challenges in the training exist, e.g., the error space is highly nonlinear and with numerous local minima, the model is of high dimensions, training data contains catastrophic errors. In this thesis through a generic formulation of the feedforward neural

network structure the optimization approaches are explored in developing suitable training algorithms to address these difficulties with RF/microwave applications. In neural network area the prevailing training methods are Backpropagation or its variants. The contribution of this thesis is to apply a variety of optimization algorithms to the training of neural networks. Several training methods derived from the optimization methods such as second order optimization methods, simplex method and global optimization methods are developed to tackle the difficult training problems in different aspects. In the microwave applications large errors are often introduced in the training data due to simulation/measurement errors. To specifically address this practical issue a novel and robust Huber quasi Newton method (HQN) is proposed in this thesis.

1.2 Organization of the Thesis

The thesis is organized as follows.

In Chapter 2, an overview of neural network structures, training methods and relevant issues is presented. Firstly the problem statement of the neural network modeling is described. Then a brief review of various feedforward neural network structures such as Multilayer Perceptrons (MLP), Radial Basis Function Networks (RBF), Knowledge Based Neural Networks (KBNN) and Wavelet Neural Networks (WNN) is given. Following this a review of different training methods is presented. This chapter also addresses relevant issues about how to compute the derivatives of the error function and how to determine the neurons' processing sequence in the feedforward neural network.

In Chapter 3, quasi Newton training method is described. Some important issues in the implementation are discussed, such as the hereditary positive definiteness

condition and how to preserve the positive definiteness property of Hessian matrix. The advantages of the quasi Newton method are demonstrated by comparison with other training methods in microwave application examples.

Chapter 4 proposes a novel Huber quasi Newton method which combines the Huber concept with the second-order optimization algorithm. The effectiveness of the proposed method in dealing with gross errors in the training data has been demonstrated through four examples by comparison with the conventional l_2 methods. Its application in Iterative Multi-Stage training (IMS) is also introduced.

In Chapter 5, the Simplex method is introduced in the training of neural networks. The standard operations in the simplex method, namely reflection, expansion and contraction are explained. A DC characteristic modeling example of the Field Effect Transistor is used to compare the simplex method with other training methods.

Two global training methods, namely Simulated Annealing Algorithm and Genetic Algorithm, are described in the subsequent Chapters 6 and 7. Chapter 6 explains the Simulated Annealing Algorithm (SA) and some implementation consideration. The simplex method is used as a random movement generator to improve the searching efficiency of the SA. The global searching ability is demonstrated through four examples, including two wavelet neural network examples.

In Chapter 7, Genetic Algorithm (GA) is investigated and applied to the optimization of neural networks. To apply it in the continuous domain for the neural

network training a couple of conversion schemes are discussed. The same four examples as used in testing SA are also used here to test the Genetic Algorithm.

Finally, Chapter 8 presents the summary and the final conclusion of the thesis. A brief comparison of various training algorithms in different aspects is presented. Future directions and challenges in this area are also discussed.

Chapter 2

Overview

2.1 Problem Statement

Let $\mathbf{x}$ be a N_x -vector containing parameters of a given device or a circuit, e.g., gate length and gate width of a FET transistor; or geometrical and physical parameters of transmission lines. Let $\mathbf{y}$ be a N_y -vector containing the responses of the device or the circuit under consideration, e.g., drain current of a FET; or S-parameters of transmission line. The relationship between $\mathbf{x}$ and $\mathbf{y}$ may be multidimensional and nonlinear. In the original EM/circuit problems, this relationship is represented by

$$\mathbf{y} = \mathbf{f}(\mathbf{x}). \quad (2.1)$$

Such a relationship can be modeled by a neural network, by training it through a set of $\mathbf{x} - \mathbf{y}$ sample pairs given by

$$\{(\mathbf{x}_p, \mathbf{d}_p), p=1,2,\dots,N_p\}, \quad (2.2)$$

where $\mathbf{x}_p$ and $\mathbf{d}_p$ are N_x - and N_y -dimensional vectors representing the p^{th} sample of $\mathbf{x}$ and $\mathbf{y}$ respectively. This sample data called the training data is generated from original

EM simulations or measurement. Let the neural network model for the relationship in (2.1) be represented by

$$y = y(\mathbf{x}, \mathbf{w}), \quad (2.3)$$

where $\mathbf{w}$ is the parameters of the neural network model, which is also called the weight vector in neural network literature, and $\mathbf{x}$ and y are called the inputs and outputs of the neural model. The definition of $\mathbf{w}$, and how y is computed through $\mathbf{x}$ and $\mathbf{w}$ determine the structure of the neural network. The neural model of (2.3) will not represent the original problem of (2.1), unless the neural model is trained by data in (2.2). A basic description of the training problem is to determine $\mathbf{w}$ such that the difference between the neural model outputs y and desired outputs d from simulation/measurement,

$$e_p(\mathbf{x}_p, \mathbf{w}) = \sqrt{\sum_{k=1}^{N_p} (y_{pk}(\mathbf{x}_p, \mathbf{w}) - d_{pk})^2} \quad (2.4)$$

and

$$E(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{N_p} e_p^2 \quad (2.5)$$

is minimized. In (2.4) d_{pk} is the k^{th} element of vector $\mathbf{d}_p$, $y_{pk}(\mathbf{x}_p, \mathbf{w})$ is the k^{th} output of the neural network when the input presented to the network is $\mathbf{x}_p$. Once trained, the neural network model can be used for predicting the output values given only the values of the input variables. In the model testing stage, an independent set of input-output samples, called the testing data is used to test the accuracy of the neural model. Normally, the testing data should lie within the same input range as the training data. The ability of neural models to predict y when presented with input parameter values $\mathbf{x}$, never seen during training is called the generalization ability. A trained and tested neural model can then be used online during microwave design stage providing fast model evaluation

replacing original slow EM/Device simulators. The benefit of the neural model approach is especially significant when the model is highly repetitively used in design processes such as, optimization, Monte Carlo analysis and yield maximization.

When the outputs of neural network are continuous functions of the inputs, the modeling problem is known as regression or function approximation, which is the most common case in microwave design area. In the next section, a brief review of neural network structures used for this purpose is presented.

2.2 Standard Feedforward Neural Networks

This section describes the architectures, error measurement and its derivatives of the feed-forward neural networks. A unifying framework that covers most kinds of feed-forward neural network paradigms with supervised learning capabilities is described.

Neural network is a network structure consisting of a number of nodes connected through directional links. Each node represents a process unit, and the links between nodes specify the causal relationship between the connected nodes. Neural networks are generally classified into two categories on the basis of the type of connections: feed-forward and recurrent. In feed-forward networks there is no feedback link that forms a circular path in the network. From the viewpoint of graph theory, a feed-forward network is represented by an acyclic directed graph which contains no directed cycles, while a recurrent network contains at least one directed cycle. A feed-forward neural network is actually a static mapping between its input and output spaces and thus capable of approximating generic classes of functions, including continuous and integrable ones.

For the purpose of function approximation, the error is defined in (2.4) which measures the discrepancy between the network's actual outputs and the desired outputs. In order to calculate this error and its derivatives, a general framework is formulated for all the feed-forward neural networks, such as MLP, RBF, and Wavelet Neural Networks. The representation of a typical neuron in the general structure is shown in Figure 2.1 [23].

Figure 2.1: The representation of i^{th} neuron in a general feed-forward structure.

In the following discussion, we shall assume that each node $Z^{(i)}$ performs a static mapping from its input(s) to output(s), i.e., the output only depends on its current inputs. Moreover to facilitate the development of derivative of the error function, we assume all node functions are differentiable. In general the network is heterogeneous and each node may have a specific node function different from the others. Links in the network are merely used to specify the propagation direction of the node outputs, generally there are

no weights or parameters associated with links. The relation between the inputs and outputs of i^{th} neuron $Z^{(i)}$ is given by

$$Z^{(i)} = f^{(i)}(\mathbf{x}^{(i)}, \mathbf{w}^{(i)}), \quad (2.6)$$

where $\mathbf{x}^{(i)}$ is the input vector, $\mathbf{w}^{(i)}$ is the parameter vector, and $f^{(i)}(\mathbf{x}^{(i)}, \mathbf{w}^{(i)})$ is the activation function of the i^{th} neuron $Z^{(i)}$. The relationship between $\mathbf{x}^{(i)}, \mathbf{w}^{(i)}$ and $\mathbf{x}, \mathbf{w}$ is given by

$$\mathbf{x} = \bigcup_{i \in \phi} \mathbf{x}^{(i)}, \quad \mathbf{w} = \bigcup_{i \in I} \mathbf{w}^{(i)}, \quad (2.7)$$

where ϕ is the index set of input neurons, I is the index set of all neurons. Other inputs $\mathbf{x}^{(j)}, j \notin \phi$ would be the outputs of hidden neurons.

Suppose we already label the nodes (as will be shown in Section 2.3) in an ordered sequence $1, 2, 3, \dots, N$, where N is the total number of neurons, such that there are no links from node i to node j whenever $i \geq j$. Let $p = 1, 2, \dots, N_p$ be the index of the samples and $k = 1, 2, \dots, N_y$ be the index of the outputs. Then $y_{pk}(\mathbf{x}_p, \mathbf{w})$ represents the k^{th} output of the neural network when the input presented to the network is $\mathbf{x}_p$. According to (2.4) (2.5) $y_{pk}(\mathbf{x}_p, \mathbf{w})$ needs to be calculated first before we could evaluate the training objective function $E(\mathbf{w})$. The calculation of $y_{pk}(\mathbf{x}_p, \mathbf{w})$ proceeds from the input neurons to the k^{th} output neuron in the ascending order of the ordered sequence, when the input of the neural network is presented with p^{th} sample $\mathbf{x}_p$. In this process the output of each neuron is calculated through function evaluation of its activation function. The inputs to every neuron are either from the outputs of the processed neurons or the inputs to the neural network, depending on where the input connection comes from. Therefore the processing sequence of the neurons would be important in order to have the inputs of the neuron ready before processing it. This is addressed in Section 2.3.

For an individual neuron, the derivatives of the outputs of i^{th} neuron $\mathbf{Z}^{(i)}$ in (2.6) with respect to the inputs and parameters can be calculated through the differentiation of the activation function, given by

$$\frac{\partial \mathbf{Z}^{(i)}}{\partial \mathbf{w}^{(i)}} = \frac{\partial \mathbf{f}^{(i)}}{\partial \mathbf{w}^{(i)}} \quad \text{and} \quad \frac{\partial \mathbf{Z}^{(i)}}{\partial \mathbf{x}^{(i)}} = \frac{\partial \mathbf{f}^{(i)}}{\partial \mathbf{x}^{(i)}}. \quad (2.8)$$

The derivatives of the overall error function E in (2.5) with respect to the trainable parameters $\mathbf{w}$ (weights of the neural network) can be computed through the back-propagation process. It starts from the output neurons and proceeds to the input neurons using the chain differentiation rule. The general formula can be expressed as

$$\frac{\partial E}{\partial w_j^{(i)}} = \sum_{p=1}^N e_p \frac{\partial e_p}{\partial w_j^{(i)}} \quad (2.9)$$

and

$$\frac{\partial e_p}{\partial w_j^{(i)}} = \sum_{(k,l) \in \Phi^{(i)}} \frac{\partial e_p}{\partial x_l^{(k)}} \frac{\partial Z_m^{(i)}}{\partial w_j^{(i)}}, \quad \frac{\partial e_p}{\partial x_n^{(i)}} = \sum_{(k,l) \in \Phi^{(i)}} \frac{\partial e_p}{\partial x_l^{(k)}} \frac{\partial Z_m^{(i)}}{\partial x_n^{(i)}}, \quad (2.10)$$

where $w_j^{(i)}$ is the j^{th} parameter of $\mathbf{Z}^{(i)}$, $x_n^{(i)}$ is the n^{th} input of $\mathbf{Z}^{(i)}$, and $\Phi^{(i)}$ is the index set of a pair of number (k, l) containing the link-to information of neuron $\mathbf{Z}^{(i)}$. Since the m^{th} output of i^{th} neuron $\mathbf{Z}^{(i)}$ is linked to the k^{th} neuron as the l^{th} input, i.e., $x_l^{(k)} = Z_m^{(i)}$, m can be determined by the neural network connection. If we consider (k, l) as a link from $\mathbf{Z}^{(i)}$, $\Phi^{(i)}$ includes all the links which start from $\mathbf{Z}^{(i)}$. The processing sequence in the calculation of derivatives is the reverse order of feed-forward sequence and therefore this process is called backward-propagation.

Different activation functions and different ways of connecting neurons in the general structure will lead to different feed-forward neural network structures.

2.2.1 Multilayer Perceptrons (MLP)

An important class of feed-forward neural networks is Multilayer Perceptrons (MLP). Typically, the MLP neural network consists of an input layer, one or more hidden layers and an output layer. The most commonly used function for hidden neurons $f(\mathbf{x}^{(i)}, \mathbf{w}^{(i)})$, is logistic sigmoid function given by

$$f(\mathbf{x}^{(i)}, \mathbf{w}^{(i)}) = \frac{1}{1 + \exp(-w_1^{(i)} - \sum_{k=2}^{nw^{(i)}} w_k^{(i)} x_{k-1}^{(i)})}, \quad (2.11)$$

where the $w_1^{(i)}$ is the first element of $\mathbf{w}^{(i)}$ and $nw^{(i)}$ is the total number of parameters in $\mathbf{w}^{(i)}$ and correspondingly the number of inputs of $\mathbf{Z}^{(i)}$ would be $nw^{(i)} - 1$. In general we define

$$t = w_1^{(i)} + \sum_{k=2}^{nw^{(i)}} w_k^{(i)} x_{k-1}^{(i)}. \quad (2.12)$$

The activation function can be redefined as $\sigma(t)$ and has the property of

$$\sigma(t) \longrightarrow \begin{cases} 1 & \text{as } t \rightarrow +\infty \\ 0 & \text{as } t \rightarrow -\infty \end{cases} \quad (2.13)$$

$\sigma(\cdot)$ is usually a monotone squashing function. Other possible candidates for $\sigma(\cdot)$ are the arctangent function given by

$$\sigma(t) = \left(\frac{2}{\pi} \right) * \arctan(t) \quad (2.14)$$

and the hyperbolic tangent function given by

$$\sigma(t) = \frac{(e^t - e^{-t})}{(e^t + e^{-t})}. \quad (2.15)$$

All these functions are bounded, continuous, monotonic and continuously differentiable.

The activation functions in output layer are linear functions $f(t) = t$.

It is well known that a two-layered MLP (no hidden layers) is not capable of approximating generic nonlinear continuous functions [24][25]. The universal approximation theorem [26][27] states that a 3-layer perceptron, with one hidden sigmoidal layer, is capable of modeling virtually any real function of interest to any desired degree of accuracy, provided sufficiently many hidden neurons are available. As such, failure to develop a good neural model can be attributed to inadequate learning, inadequate number of hidden neurons, or the presence of a stochastic rather than a deterministic relation between input and output [27].

However, in reality a neural network can only have finite number of hidden neurons. Usually, 3 or 4 layered perceptrons are used in neural modeling of microwave circuit components. Neural network performance can be evaluated based on generalization capability and mapping capability [28]. In the function approximation or regression area, generalization capability is a major concern. It is shown in [29] that four-layered perceptrons are not preferred in all but the most esoteric applications in terms of generalization capability. Intuitively, four-layered perceptrons would perform better in defining the decision boundaries in pattern classification tasks because of an additional nonlinear hidden layer resulting in hierarchical decision boundaries. This has been verified in [28] for the mapping capability of the network.

2.2.2 Radial Basis Function Networks (RBF)

Feedforward neural networks which have only one hidden layer, and use radial basis activation functions in the hidden layer, are called Radial Basis Function (RBF)

networks. Radial basis functions are derived from the regularization theory in the approximation of multivariate functions [30][31]. Park and Sandberg showed that RBF networks also have universal approximation ability [32][33]. Universal convergence of RBF nets in function estimation and classification has been proved by Krzyzak et al. [34].

The Radial Basis Network (RBF) has one input layer, one output layer and one hidden layer. The output neurons of RBF networks are also linear neurons. The activation function of hidden layer is a radial basis function. First we define

$$t = |\mathbf{x}^{(i)} - \boldsymbol{\theta}^{(i)}|, \quad (2.16)$$

where $\boldsymbol{\theta}^{(i)}$ is the center of radial basis function of the i^{th} hidden neuron. Some of the commonly used radial basis activation functions are [35]

$$f(\mathbf{x}^{(i)}, \mathbf{w}^{(i)}) = \sigma(t) = \exp\left[-\left(\frac{t}{\lambda}\right)^2\right] \quad (2.17)$$

and
$$f(\mathbf{x}^{(i)}, \mathbf{w}^{(i)}) = \sigma(t) = (c^2 + t^2)^\beta, \quad 0 < \beta < 1, \quad (2.18)$$

where (2.17) is the Gaussian function, (2.18) is the multiquadratic function and λ , c and β are the function parameters. Training parameters $\mathbf{w}^{(i)}$ includes $\boldsymbol{\theta}^{(i)}$ and λ / (c and β).

Although MLP and RBF are both feed-forward neural networks, the different nature of the hidden neuron activation functions makes them behave very differently. Firstly, the activation function of each hidden neuron in a MLP computes the inner product of the input vector and the synaptic weight vector of that neuron. On the other hand, the activation function of each hidden neuron in a RBF network computes the Euclidean norm between the input vector and the center of that neuron. Secondly, MLP networks construct global approximations to nonlinear input-output mapping. Consequently, they are capable of generalizing in those regions of the input space where little or no training

data is available. On the contrary, RBF networks use exponentially decaying localized nonlinearities (e.g., Gaussian functions) to construct local approximations to nonlinear input-output mapping. As a result RBF neural networks are capable of faster learning and exhibit reduced sensitivity to the order of presentation of training data [36]. Consequently, a hidden neuron influences the outputs of the network only for inputs near to its center, and an exponential number of hidden neurons are required to cover the entire domain. In [37], it is suggested that RBF networks are suited for problems with smaller number of inputs.

The universal approximation theorems for both MLP and RBF only state that there exists such a network to approximate virtually any nonlinear function. However, they did not specify how large a network should be for a particular problem complexity. Several algorithms have been proposed to find proper network size, e.g., constructive algorithms [38], network pruning [39]. Regularization [40] is also a technique used to match the model complexity with problem complexity. Rational functions have also been proved to universally approximate any real-valued functions. In [41], a network architecture that uses a rational function to construct a mapping neural network has been proposed. The complexity of the architecture is still considered as a major drawback of the rational function approach, although it requires fewer parameters than a polynomial function.

2.2.3 Knowledge-Based Neural Networks (KBNN)

Since MLP and RBF belong to the type of black box models structurally embedding no problem dependent information, the entire information about the application comes from training data. Consequently, large amount of training data is usually needed to

ensure model accuracy. In microwave applications, obtaining a relatively larger set of training data by either EM/physics simulation, or by measurement, is expensive and/or difficult. The underlying reason is that simulation/measurement may have to be performed for many combinations of geometrical/material/process parameters. On the contrary, if we try to reduce the training data, the resulting neural models may not be reliable. Neural network structures with prior knowledge address this problem.

In [1], a new microwave-oriented knowledge-based neural network (KBNN) was introduced. In this method the microwave knowledge in the form of empirical functions or analytical approximations is embedded into the neural network structure. This structure was inspired from the fact that practical empirical functions are usually valid only in a certain region of the parameter space. To build a neural model for the entire space, several empirical formulae and the mechanism to switch between them are needed. The switching mechanism expands the feature of Sigmoidal Radial Basis Function [42] into high dimensional space and with more generalized activation functions. This model retains the essence of neural networks in that the exact location of each switching boundary, and the scale and position of each knowledge function are initialized randomly and then determined eventually during training. The Knowledge Based Neural Network (KBNN) structure is a non-fully connected structure as shown in Figure 2.2. There are 6 layers in the structure, namely input layer X , knowledge layer S , boundary layer B , region layer R , normalized region layer $\bar{R}$ and output layer Y . The input layer X accepts parameters x from outside the model. The knowledge layer S is the place where microwave knowledge resides in the form of single or multidimensional functions

$\psi(\cdot)$. The output of the i^{th} knowledge neuron is given by

$$s^{(i)} = \psi^{(i)}(\mathbf{x}, \mathbf{w}^{(i)}) \quad i=1,2,\dots,N_s, \quad (2.19)$$

where $\mathbf{x}$ is a vector including neural network inputs x_i , $i=1,2,\dots,N_x$ and $\mathbf{w}^{(i)}$ is a vector of parameters in the knowledge formula. The knowledge function $\psi^{(i)}(\mathbf{x}, \mathbf{w}^{(i)})$ 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, gate width, channel thickness, doping density, gate voltage and drain voltage [43]. The boundary layer $\mathbf{B}$ 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 is calculated by

$$b^{(i)} = B^{(i)}(\mathbf{x}, \mathbf{v}^{(i)}) \quad i=1,2,\dots,N_b, \quad (2.20)$$

where $\mathbf{v}^{(i)}$ is a vector of parameters in $B^{(i)}$ defining an open or closed boundary in the input space $\mathbf{x}$.

Let $\sigma(\cdot)$ be a sigmoid function. The region layer $\mathbf{R}$ contains neurons to construct regions from boundary neurons

$$r^{(i)} = \prod_{j=1}^{N_b} \sigma(\alpha_{ij} b^{(j)} + \theta_{ij}), \quad i=1,2,\dots,N_r, \quad (2.21)$$

where α_{ij} and θ_{ij} are the scaling and bias parameters, respectively.

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

$$\bar{r}^{(i)} = \frac{r^{(i)}}{\sum_{j=1}^{N_r} r^{(j)}}, \quad i=1,2,\dots,N_{\bar{r}}, \quad N_{\bar{r}}=N_r. \quad (2.22)$$

The output layer Y contains second order neurons [44] combining knowledge neurons and normalized region neurons

$$y^{(j)} = \sum_{i=1}^{N_r} \beta_{ji} s^{(i)} \left(\sum_{k=1}^{N_{\bar{r}}} \rho_{jik} \bar{r}^{(k)} \right) + \beta_{j0}, \quad j=1,2,\dots,N_y, \quad (2.23)$$

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 1 indicating 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. Training parameters $\boldsymbol{w}$ for the entire KBNN model includes

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

The prior knowledge in KBNN gives it more information about the original microwave problem, beyond that in the training data. Therefore, such model has better reliability when training data is limited or when model is used beyond training range.

Figure 2.2: Knowledge Based Neural Network (KBNN) structure.

2.2.4 Wavelet Neural Networks

The idea of combining wavelet theory with neural networks has been recently proposed [45][46][47]. Though the wavelet theory has offered efficient algorithms for various purposes, their implementation is usually limited to wavelets of small dimension. It is known that neural networks are powerful tools for handling problems of large dimension. Combining wavelets and neural networks can hopefully remedy the weakness of each other, resulting in networks with efficient constructive methods and capable of handling problems of moderately large dimension. This resulted in a new type of neural networks, called wavelet networks which use wavelets as the hidden neuron activation functions. Wavelet networks are feedforward networks with one hidden layer. The hidden neurons are computed as

$$Z^{(i)} = \psi\left(\left\|\frac{(\mathbf{x}-\mathbf{T}_i)}{a_i}\right\|\right), \quad i = 1, \dots, N_1, \quad (2.25)$$

where $\psi(\cdot)$ is the radial type mother wavelet function, a_i are dilation parameters and $\mathbf{T}_i$ are translation vectors for the i^{th} hidden wavelet neuron. The output neurons of wavelet network are also linear neurons. The trainable parameters include a_i , $\mathbf{T}_i$ and the linear coefficients in the output neurons.

Due to the similarity between adaptive discretization of the wavelet decomposition and one-hidden-layer neural networks, there is an explicit link between the network parameters such as a_i and $\mathbf{T}_i$, and the decomposition formula. As such, the initial values of network parameters can be estimated from the training data using decomposition formulae. However, if the initialization uses regularly truncated wavelet frames [46], many useless wavelets on the wavelet lattice may be included in the network, resulting in

larger network size. Alternative algorithms for wavelet network construction were proposed in [45] to better handle problems of large dimension. The number of wavelets (hidden neurons) was considerably reduced by eliminating the wavelets whose supports do not contain any data points. Some regression techniques, such as stepwise selection by orthogonalization and backward elimination, were then used to further reduce the number of wavelets. The wavelet networks with radial wavelet functions can be considered as an RBF network, since both of them have the localized basis functions. The difference is that wavelet function is localized both in the input and frequency domains [47]. Besides retaining the advantage of faster training, wavelet networks have a guaranteed upper bound on the accuracy of approximation [48] with a multi-scale structure. Wavelet networks have been used in non-parametric regression estimation [45] and were trained based on noisy observation data to avoid the problem of undesirable local minima [48]. In [14], wavelet networks and stepwise selection by orthogonalization regression technique were used to build a neural network model for a 2.9 GHz microwave power amplifier. In this technique, a library of wavelets was built according to the training data and the wavelet that best fits the training data was selected. Later in an iterative manner, wavelets in the remainder of the library that best fits the data in combination with the previously selected wavelets were selected. For computational efficiency, later selected wavelets were orthonormalized to earlier selected ones.

2.3 Neuron Sorting Algorithm

The general feed-forward neural network structure can be viewed as a directed graph if we consider each neuron as a vertex and the connections as the directed edges of the graph. Since the general structure as we discussed here is only feed-forward, the

corresponding directed graph would be an acyclic directed graph which has no directed circuits.

2.3.1 Topological Sorting

The task of determining the processing sequence can be illustrated as following. We can label the vertices of an n -vertex acyclic directed graph G with integers from the set $\{1, 2, \dots, n\}$ such that the presence of the edge (i, j) in G implies that $i < j$. Note that the edge (i, j) is directed from vertex i to vertex j . Ordering the vertices in this manner is called topological sorting [49].

Before describing the method we explore one simple property of the acyclic directed graph: in an acyclic directed graph G there exist at least one vertex with zero in-degree and at least one with zero out-degree.

Proof: Let $P: v_1, v_2, \dots, v_p$ be a maximal directed path in G . We claim that the in-degree of v_1 and the out-degree of v_p are both equal to zero. If the in-degree of v_1 is not equal to zero, then there exists a vertex w such that the edge (w, v_1) is in G . Suppose $w \neq v_i$, for any i , $1 \leq i \leq p$, then there is a directed path $P': w, v_1, v_2, \dots, v_p$ that contains all the edges of P . This contradicts that P is a maximal directed path. Suppose $w = v_i$ for some i . Then in G there is a directed circuit $C: v_1, v_2, \dots, v_i, v_1$. This is again a contradiction since G has no directed circuits. Thus there is no vertex w such that the edge (w, v_1) is in G . In other words the in-degree of v_1 is zero. Following a similar reasoning we can show that the out-degree of v_p is zero.

The vertices with zero in-degree and zero out-degree are actually input and output neurons in the general neural network structure. Correspondingly in the process we would sort from the output neurons to the input neurons.

Select any vertex with zero out-degree. Label this vertex with the integer n . Now remove from G this vertex and the edges incident on it. Let G' be the reduced graph. G' should be acyclic as we do not introduce any new connections. Similarly we can now select a vertex whose out-degree in G' is zero. Label this with the integer $n-1$. Repeat this procedure until all the vertices are labeled. It is now easy to verify that this procedure results in a topological sorting of the vertices of G .

2.3.2 Loop Check

To find out the processing sequence, we should first check whether the corresponding directed graph is acyclic [49]. And if there is a loop, we should find the closed path so that the structure can be modified accordingly.

This could be done during the procedure of topological sorting. If the directed graph is acyclic, i.e., structure is feed-forward, we should be able to continue our procedure to the last vertex (one of the zero in-degree vertices). Otherwise the procedure would stop in middle and at that time there would be no zero out-degree neurons in the G' .

Then select any vertex v_1 in the remaining graph G' , and since the out-degree of v_1 is not zero, there exists a vertex v_2 such that (v_1, v_2) is in G' . Let $P: v_1, v_2$ be the directed path found so far. From v_2 through the same manner we could keep on growing the path since every vertex in G' has non-zero out-degree. Every time before adding the new vertex v_i into path P we should check whether this vertex already exists in the list of P . If that is so we actually already find the closed path with the starting and ending vertex as v_i . Since G' only has limited number of vertices, the closed path would always be found when we are going to exhaust the vertices.

2.3.3 Neural Network Structure Identification

This method could be further improved to identify the structure of neural network, e.g., determining whether the neural network structure is a layered structure [49]. MLP, RBF and Wavelet networks are all layered structures. The characteristic of layered structure is that all neurons in one layer are only connected to the neurons in another layer. The KBNN structure is not a layered structure since the input layer is connected to different layers.

To achieve this we need to modify the topological sorting procedure a little bit. Instead of sorting the neurons one by one we sort them layer by layer. Every time all the zero out-degree neurons are removed and labeled with the same number. Those neurons would then form a layer. Specifically if we know the topology of some structures we could identify them as well.

2.4 Training Algorithms

2.4.1 Training Objective

A neural network model can be developed through a process called training. Suppose the training data consists of N_p sample pairs, $\{(\mathbf{x}_p, \mathbf{d}_p), p=1,2,\dots,N_p\}$, where $\mathbf{x}_p$ and $\mathbf{d}_p$ are N_x - and N_y -dimensional vectors representing the inputs and the desired outputs of the neural network respectively. Let $\mathbf{w}$ be the weight vector containing all the N_w weights of the neural network. For example for KBNN $\mathbf{w}$ is given by (2.24).

The objective of training is to find $\mathbf{w}$ such that the error in (2.5) between the neural network predictions and the desired outputs are minimized,

$$\min_{\mathbf{w}} E(\mathbf{w}). \quad (2.26)$$

The objective function $E(\mathbf{w})$ is a nonlinear function w.r.t. the adjustable parameter $\mathbf{w}$. Due to the complexity of $E(\mathbf{w})$, iterative algorithms are often used to explore the parameter space efficiently[50].

2.4.2 Review of Backpropagation Algorithm

One of the most popular algorithms for neural network training is Back Propagation (BP) algorithm [51], proposed by Rumelhart, Hinton and Williams in 1986. The BP algorithm is a stochastic algorithm based on the steepest descent principle [52], wherein the weights of the neural network are updated along the negative gradient direction in the weight space. The update formulae are given by

$$\Delta \mathbf{w}_{now} = \mathbf{w}_{next} - \mathbf{w}_{now} = -\eta \left. \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w} = \mathbf{w}_{now}} \quad (2.27a)$$

and

$$\Delta \mathbf{w}_{now} = \mathbf{w}_{next} - \mathbf{w}_{now} = -\eta \left. \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w} = \mathbf{w}_{now}}, \quad (2.27b)$$

where η called learning rate controls the step size of weight update. Update formula (2.27b) is called sample-by-sample update, where the weights are updated after each sample is presented to the network. Update formula (2.27a) is called batch mode update, where the weights are updated after all training samples have been presented to the network.

The basic backpropagation, derived from the principles of steepest descent, suffers from slower convergence and possible weight oscillation. The addition of a momentum term to weight update formulae as proposed by [51], provided significant improvements to the basic backpropagation, reducing the weight oscillation.

$$\Delta \mathbf{w}_{now} = -\eta \left. \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{now}} + \alpha \Delta \mathbf{w}_{old} = -\eta \left. \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{now}} + \alpha (\mathbf{w}_{now} - \mathbf{w}_{old}) \quad (2.28a)$$

$$\Delta \mathbf{w}_{now} = -\eta \left. \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{now}} + \alpha \Delta \mathbf{w}_{old} = -\eta \left. \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{now}} + \alpha (\mathbf{w}_{now} - \mathbf{w}_{old}), \quad (2.28b)$$

where α is the momentum factor which controls the influence of the last weight update direction on the current weight update, and $\mathbf{w}_{old}$ represents the last point of $\mathbf{w}$. This technique is also known as the generalized delta-rule [36]. Other approaches to reduce weight oscillation have also been proposed, such as invoking a correction term that uses the difference of gradients [53], and the constrained optimization approach where constraints on weights are imposed to achieve better alignment between weight updates in different epochs [54].

An important way to improve efficiency of training by backpropagation is to use adaptation schemes that allow the learning rate and the momentum factor to be adaptive during learning [36], e.g., adaptation according to training errors [55]. One of the most interesting works in this area is the delta-bar-delta rule proposed by [56]. He developed an algorithm based on a set of heuristics in which the learning rate for different weights are defined separately and also adapted separately during the learning process. The adaptation is determined from two factors, one being the current derivative of the training error with respect to the weights, and the other being an exponentially weighted sum of the current and past derivatives of the training error. Various other adaptation techniques have also been proposed, for example, a scheme in which the learning rate was adapted in order to reduce the energy value of the gradient direction in a close-to-optimal way [57], an Enhanced Back Propagation algorithm [58] with a scheme to adapt the learning rate

according to values of weights in the neural net, and a learning algorithm inspired from the principle of “forced dynamics” for the total error function [59]. The algorithm in [59] updates the weights in the direction of steepest descent, but with the learning rate as a specific function of the error and the error gradient form.

In general, during neural network training, the weights are updated after each iteration by a certain step size along an updating direction. The standard backpropagation uses learning rate to adjust the step size, with the advantage that the method is very simple and does not require repetitive computation of the error functions. A different way to determine the step size, is to use one-dimensional line search methods [52], so that the training error is reduced or optimized along the given updating direction. The examples in this category are line search based on quadratic model [60], and line search based on linear interpolation [61][62]. One other way to improve training efficiency is the gradient reuse algorithm [63]. The basic idea of this method is that gradients which are computed during training are reused until the resulting weight updates no longer lead to a reduction in the training error.

2.4.3 Gradient-based Optimization Methods

The backpropagation based on steepest descent principle is relatively easy to implement. However, the error surface of neural network training usually contains planes with a gentle slope due to the squashing functions commonly used in neural networks. This gradient is too small for weights to move rapidly on these planes, thus reducing the rate of convergence. The rate of convergence could also be very slow when the steepest descent method encounters “narrow valley” in the error surface where the direction of gradient is close to the perpendicular direction of the valley. The update direction

oscillates back and forth along the local gradient.

Since supervised learning of neural networks can be viewed as a function optimization problem, higher order optimization methods using gradient information can be adopted in neural network training to improve the rate of convergence. Compared to the heuristic approach discussed in the earlier Backpropagation section, these methods have a sound theoretical basis and guaranteed convergence for most of the smooth functions. Some of the early work in this area was demonstrated in [64] with the development of second-order learning algorithms for neural networks. Papers [61][65][66] reviewed the first- and second-order optimization methods for learning in feedforward neural networks.

In neural network training a real-valued objective function $E(\mathbf{w})$ is defined on a N_w -dimensional input space $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_{N_w}]^T$. The purpose of training is to find a (possibly local) minimum point $\mathbf{w} = \mathbf{w}^*$ that minimizes $E(\mathbf{w})$. In general, a general objective function E may have a nonlinear form with respect to an adjustable parameter vector $\mathbf{w}$. Due to the complexity of E , we often use an iterative algorithm to explore the input space efficiently. In iterative descent methods, the next point $\mathbf{w}_{next}$ is determined by a step down from the current point $\mathbf{w}_{now}$ in a direction vector $\mathbf{h}$,

$$\mathbf{w}_{next} = \mathbf{w}_{now} + \eta \mathbf{h}, \quad (2.29)$$

where η is a positive step size regulating to what extent to proceed in that direction. In neural network literature, the term learning rate is used for the step size η . The next point $\mathbf{w}_{next}$ should satisfy the following inequality

$$E(\mathbf{w}_{next}) = E(\mathbf{w}_{now} + \eta \mathbf{h}) < E(\mathbf{w}_{now}). \quad (2.30)$$

The principal difference between various descent algorithms lies in the first procedure for determining successive directions. Once the direction is found, the optimal step size could be found by line search

$$\eta^* = \min_{\eta > 0} \phi(\eta), \text{ where } \phi(\eta) = E(\mathbf{w}_{now} + \eta \mathbf{h}). \quad (2.31)$$

When the downhill direction $\mathbf{h}$ is determined on the basis of the gradient ($\mathbf{g}$) of an objective function E , such descent methods are called gradient-based descent methods.

$$\mathbf{g}(\mathbf{w}) = \nabla E(\mathbf{w}) = \left[\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_1}, \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_2}, \dots, \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_{N_s}} \right]^T. \quad (2.32)$$

The procedure for finding a gradient vector in a network structure is generally similar to backpropagation [51] in the sense that the gradient vector is calculated in the direction opposite to the flow of output from each neuron. It has been discussed in Section 2.2.

In general, based on a given gradient, downhill direction adhere to the following condition for feasible descent directions [50],

$$\phi'(0) = \left. \frac{dE(\mathbf{w}_{now} + \eta \mathbf{h})}{d\eta} \right|_{\eta=0} = \mathbf{g}^T \mathbf{h} = |\mathbf{g}| |\mathbf{h}| \cos(\xi(\mathbf{w}_{now})) < 0, \quad (2.33)$$

where ξ signifies the angle between $\mathbf{g}$ and $\mathbf{h}$, and $\xi(\mathbf{w}_{now})$ denotes the angle between $\mathbf{g}_{now}$ and $\mathbf{h}$ at the current point $\mathbf{w}_{now}$. This can be verified by the Taylor series expansion of E ,

$$E(\mathbf{w}_{now} + \eta \mathbf{h}) = E(\mathbf{w}_{now}) + \eta \mathbf{g}^T \mathbf{h} + \mathbf{o}(\eta^2). \quad (2.34)$$

The second term on the right-hand side will dominate the third and other higher order terms of η when $\eta \rightarrow 0$. With such a small positive η , the inequality (2.30) clearly holds when $\mathbf{g}^T \mathbf{h} < 0$.

A class of gradient-based descent methods has the following fundamental form, in which feasible descent direction can be determined by deflecting the gradients through multiplication by G ,

$$\mathbf{w}_{next} = \mathbf{w}_{now} - \eta G \mathbf{g}, \quad (2.35)$$

with some positive step size η and some positive definite matrix G . Clearly, when $\mathbf{h} = -G\mathbf{g}$, the descent direction condition (2.33) holds since

$$\mathbf{g}^T \mathbf{h} = -\mathbf{g}^T G \mathbf{g} < 0. \quad (2.36)$$

Many other variants of gradient based methods (e.g., quasi-Newton method, Gauss Newton Method and the levenberg-Marquardt method) possess (2.35) to modify the negative gradient direction ($-\mathbf{g}$) for a better choice.

2.4.3.1 Newton's Method

Newton's method is a second-order optimization method where the descending direction $\mathbf{h}$ is determined by using the second-order derivatives of the objective function E . The objective function can be approximated by a Taylor series expansion up to second-order,

$$E(\mathbf{w} + \Delta \mathbf{w}) \approx E(\mathbf{w}) + \mathbf{g}(\mathbf{w})^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H}(\mathbf{w}) \Delta \mathbf{w}, \quad (2.37)$$

where $\mathbf{H}$ is the **Hessian** matrix, consisting of the second-order partial derivatives of $E(\mathbf{w})$. Since Equation (2.37) is a quadratic function of $\mathbf{w}$ we can simply find its minimum point $\mathbf{w}^*$ by differentiating (2.37) and set it to zero. This leads to

$$0 = \mathbf{g} + \mathbf{H} \Delta \mathbf{w} \quad \text{and} \quad \mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1} \mathbf{g}. \quad (2.38)$$

If $\mathbf{H}$ is positive definite and E is quadratic, then Newton's method would directly goes to the minimum in a single step. However the Hessian matrix of general nonlinear objective

function usually is not positive definite and would lead to a uphill direction which makes the optimization process unefficient. For this reason the original Newton's method is rarely used.

2.4.3.2 Conjugate Gradient Method

Like Newton's method the conjugate gradient methods are originally derived from quadratic minimization and the minimum of the objective function $E(\mathbf{w})$ can be efficiently found within N_w iterations, where N_w is the number of dimensions of $\mathbf{w}$. With

initial gradient $\mathbf{g}_{initial} = \left. \frac{\partial E}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{initial}}$, and direction vector $\mathbf{h}_{initial} = -\mathbf{g}_{initial}$, the conjugate

gradient method recursively constructs two vector sequences [62],

$$\mathbf{g}_{next} = \mathbf{g}_{now} + \lambda_{now} \mathbf{H} \mathbf{h}_{now}, \quad (2.39)$$

$$\mathbf{h}_{next} = -\mathbf{g}_{next} + \gamma_{now} \mathbf{h}_{now}, \quad (2.40)$$

$$\lambda_{now} = \frac{\mathbf{g}_{now}^T \mathbf{g}_{now}}{\mathbf{h}_{now}^T \mathbf{H} \mathbf{h}_{now}}, \quad (2.41)$$

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

or,

$$\gamma_{now} = \frac{(\mathbf{g}_{next} - \mathbf{g}_{now})^T \mathbf{g}_{next}}{\mathbf{g}_{now}^T \mathbf{g}_{now}}, \quad (2.43)$$

where $\mathbf{h}$ is called the conjugate direction and $\mathbf{H}$ is the Hessian matrix of the objective function E . Here, (2.42) is called the Fletcher-Reeves formula and (2.43) is called the Polak-Ribiere formula. To avoid the need of Hessian matrix to compute the conjugate direction, we proceed from $\mathbf{w}_{now}$ along the direction $\mathbf{h}_{now}$ to the local minimum of E at

$\mathbf{w}_{next}$ through line minimization, and then set $\mathbf{g}_{next} = \left. \frac{\partial E}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_{next}}$. This $\mathbf{g}_{next}$ can be used as

the vector of (2.39), and as such (2.41) is no longer needed. We make use of this line minimization concept to find conjugate direction in neural network training, thus avoiding intensive Hessian matrix computations. In this method, the descent direction is along the conjugate direction which can be accumulated without computations involving matrices. As such, conjugate gradient methods are very efficient and scale well with the neural network size.

Two critical issues have to be considered in applying conjugate gradient methods to neural network learning. Firstly, computation required during the exact one-dimensional optimization is expensive because every function evaluation involves a complete cycle of sample presentation and neural network feed-forward calculation. Therefore, efficient approximation in one-dimensional optimization has to be used. Secondly, since for neural network training, the error function is not quadratic w.r.t. the variable $\mathbf{w}$ as defined in (2.5), the convergence properties of the method are not assured a priori but depend on the degree to which a local quadratic approximation can be applied to the training error surface. In [57], inexact line search was proposed and a modified definition of the conjugate search direction was used to achieve this purpose. To further reduce computational complexities, a Scaled Conjugate Gradient (SCG) algorithm was introduced in [67] which avoids the line search per learning iteration by using Levenberg-Marquardt approach to scale the step size.

2.4.3.3 Levenberg-Marquardt and Gauss-Newton Training Algorithms

Neural network training is usually formulated as a nonlinear least squares problem. Methods dedicated to least-squares such as Gauss-Newton can be employed to estimate

the neural model parameters. The Gauss-Newton method is a linearization method. Let $\mathbf{r}$ be a vector containing the individual error terms in (2.4). Let $\mathbf{J}$ be the $(N_p \times N_y) \times N_w$ Jacobian matrix including the derivative of $\mathbf{r}$ w.r.t. $\mathbf{w}$. $\mathbf{J}$ has N_w columns and $N_p \times N_y$ rows, where N_p is the number of samples and N_y is the number of outputs. The Gauss-Newton update formula can be expressed as

$$\mathbf{w}_{next} = \mathbf{w}_{now} - (\mathbf{J}_{now}^T \mathbf{J}_{now})^{-1} \mathbf{J}_{now}^T \mathbf{r}_{now}. \quad (2.44)$$

In the preceding formula (2.44) $\mathbf{J}_{now}^T \mathbf{J}_{now}$ is positive definite unless $\mathbf{J}_{now}$ is rank deficient. The Levenberg-Marquardt [68] method can be applied when $\mathbf{J}_{now}$ is rank deficient and the corresponding weight update is given by

$$\mathbf{w}_{next} = \mathbf{w}_{now} - (\mathbf{J}_{now}^T \mathbf{J}_{now} + \mu \mathbf{I})^{-1} \mathbf{J}_{now}^T \mathbf{r}_{now}, \quad (2.45)$$

where μ is a non-negative number. A modified Levenberg-Marquardt training algorithm using a diagonal matrix instead of the identity matrix $\mathbf{I}$ in (2.45) was proposed [69] for efficient training of multilayer feedforward neural networks. In [70], to reduce the size of the Jacobian matrix, the training samples are divided into several groups called local-batches. The training is performed successively through these local batches. The computational requirement and memory complexity of Levenberg-Marquardt methods were reduced by utilizing the deficient Jacobian matrix [71]. A combined Levenberg-Marquardt and Quasi-Newton training technique was used in [1] to train the KBNN structure. When training parameters are far from local minimum of the error surface Levenberg-Marquardt algorithm was used, and when they are close to local minimum quasi-Newton was used for faster convergence.

It has been proved that the conjugate gradient method is equivalent to error backpropagation with momentum term [61]. The theoretical convergence rate and

practical performance of second-order gradient-based methods are generally superior to the first-order methods.

2.4.3.4 Line Search

Recall the formula of a class of gradient-based descent methods given by Equation (2.29), all the descent methods need to perform a line search along the descent direction. The line search, i.e., the sub-task of step size determination along search direction $\mathbf{h}$ at each iteration, is actually a one-dimensional minimization. The efficiency of the step size determination affects the entire minimization process.

There are two strategies commonly used for the line search, function comparison (accurate line search) and function approximation (inaccurate line search) [62]. Both methods need to initially bracket a minimum in an interval. A minimum is known to be bracketed only when there is a triplet of points, $a < b < c$, or $c < b < a$, such that $f(b)$ is less than both $f(a)$ and $f(c)$. In this case we know the minimum is in the interval (a, c) .

The most widely used function comparison strategy is golden section search. Given, at each stage, a bracketing triplet of points, the next point to be tried is that which is a fraction 0.38197 into the larger of the two intervals. Function comparison methods are reliable, but make no attempt to exploit the smoothness of function f . The second approach uses polynomial interpolation to locate the minimum of f . Typically the approximation is a parabolic or cubic polynomial. Parabolic interpolation requires three pieces of data obtained in bracketing. An example of polynomial interpolation algorithm is Brent's method which is widely used and highly regarded.

An alternative to the line search approach is the model trust region methods where the aim is to find the minimum in a trusted neighborhood around current point $\mathbf{w}_{now}$. The neighborhood is defined as the region around $\mathbf{w}_{now}$ in which the quadratic approximation, which is assumption of many second-order training methods is accurate enough. The trust region is usually defined as a region with a radius α_{now} . The Euclidean norm of the step should be less than α_{now} . Generally radius α_{now} would be updated upon whether the approximation model is accurate or not. Model trust region method is computationally efficient as it does not perform an exact one-dimensional optimization. This method is normally associated with the Levenberg-Marquardt algorithm where the search direction is more accurate and the computation effort is the major concern [68].

Chapter 3

Quasi Newton Method

3.1 Introduction

Quasi-Newton methods are the most effective nonlinear optimization methods for general problems. Quasi-Newton methods are named for the fact that successive estimates of the Hessian matrix are updated so as to retain a key Newton property. Only the objective function and its first-order partial derivatives are required for quasi-Newton methods.

Both Conjugate Gradient methods and quasi-Newton methods are derived from the optimization of quadratic function or second-order truncated Taylor series approximation of the objective function (2.37). Therefore both are second-order optimization techniques and have the property of quadratic termination, i.e., the minimum of a quadratic surface will be found in exact N_w line searches, where the function involves N_w variables. Besides this assumption, the Conjugate Gradient methods are also based on two other: (1) the initial line search was in the direction of steepest descent (negative gradient). (2) exact line searches were accomplished in each direction.

Quasi-Newton methods do not require a steepest descent start or exact line searches. They start with some positive definite estimate of the Hessian matrix and employ rank 1 or rank 2 updates to successively estimate the H^{-1} following every line search in a sequence of search directions. In quadratic surfaces, quasi-Newton methods that employ the exact line searches also possess the quadratic termination property.

3.2 Mathematical Formulation

Compared to the Newton's method, the quasi Newton method does not require second-order partial derivatives of $E(\mathbf{w})$, and the estimated inverse Hessian matrix is always positive definite so that each search direction in (2.35) is a descent direction [62]. In general, far from a minimum, we have no guarantee that the Hessian is positive definite. Taking the actual Newton's step with the real Hessian can move us to points where the function is increasing in value. The idea behind quasi Newton methods is to start with a positive definite, symmetric approximation of H or its inverse matrix B (usually the identity matrix) and build up the approximating H' or B' in such a way that it will remain positive definite and symmetric. Far from the minimum, this guarantees that we always move in a downhill direction. Close to the minimum, the approximation approaches the true Hessian matrix or its inverse and we would have the quadratic convergence speed of Newton's method.

3.2.1 Update Formula of Quasi Newton Methods

In Section 2.4.3.1 we try to find the minimum by using the Newton's method to search for a zero of the gradient of the function. Repeating (2.37) and (2.38) for a quadratic function near the point $\mathbf{w}^k$ we have

$$E(\mathbf{w}) = E(\mathbf{w}^k) + \mathbf{g}(\mathbf{w}^k)^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w}, \quad (3.1)$$

and
$$0 = \mathbf{g}^k + \mathbf{H} \Delta \mathbf{w}, \quad \mathbf{w}^* = \mathbf{w}^k - \mathbf{H}^{-1} \mathbf{g}^k \quad (3.2)$$

The Hessian matrix $\mathbf{H}$ is a constant for quadratic functions, so consideration of two different points $\mathbf{w}^k$ and $\mathbf{w}^{k+1}$ in (3.2) yields the linear mapping between changes of gradient and corresponding changes in position [50],

$$\mathbf{w}^{k+1} - \mathbf{w}^k = \mathbf{H}^{-1} [\mathbf{g}^{k+1} - \mathbf{g}^k]. \quad (3.3)$$

Now consider a nonquadratic function and define matrix $\mathbf{B}^k$ to be the estimate of the inverse Hessian matrix $\mathbf{H}^{-1}$, where k is the iteration number. Since $\Delta \mathbf{w}^k = \mathbf{w}^{k+1} - \mathbf{w}^k$ and $\Delta \mathbf{g}^k = \mathbf{g}^{k+1} - \mathbf{g}^k$ can be computed only after the line search (*Step 2* in Section 3.2.2), the estimated inverse Hessian matrix $\mathbf{B}^k$ does not represent this mapping correctly in the sense of (3.3). To force that, the *quasi-Newton condition* is defined,

$$\Delta \mathbf{w}^k = \mathbf{B}^{k+1} \Delta \mathbf{g}^k. \quad (3.4)$$

This quasi-Newton condition merits emphasis: it maintains the linear mapping between corresponding changes in gradient and position that underlies Newton's method.

A general definition of a matrix update of inversion of Hessian matrix is

$$\mathbf{B}^{k+1} = \mathbf{B}^k + \Delta \mathbf{B}^k, \quad \text{or} \quad \mathbf{B}^* = \mathbf{B} + \Delta \mathbf{B}. \quad (3.5)$$

The right-hand representation for $\mathbf{B}^*$ is used to avoid excessive notation, since the concern here is only with changes that occur during each iteration. Although the rank of the update term $\Delta \mathbf{B}$ can be as great as N_w , in practice, the updates are usually only of rank 1 or 2 [81].

A rank 1 update simply adds a scaled outer product of a vector $\mathbf{m}$ as defined by

$$\mathbf{B}^* = \mathbf{B} + q \mathbf{m} \mathbf{m}^T. \quad (3.6)$$

The outer product of the same vector $\mathbf{m}$ in update formula (3.6) would guarantee the hereditary symmetry of the approximation matrix $\mathbf{B}$. If the quasi-Newton condition in (3.3) is to be satisfied, then

$$\Delta \mathbf{w} = \mathbf{B} \Delta \mathbf{g} + q \mathbf{m} \mathbf{m}^T \Delta \mathbf{g}. \quad (3.7)$$

Note that $\mathbf{m}^T \Delta \mathbf{g}$ in the last term is a scalar, thus $\mathbf{m}$ must be proportional to $\Delta \mathbf{w} - \mathbf{B} \Delta \mathbf{g}$. A simple choice for $\mathbf{m}$ and q would be

$$\mathbf{m} = \Delta \mathbf{w} - \mathbf{B} \Delta \mathbf{g}, \quad (3.8)$$

and

$$q = \frac{1}{\mathbf{m}^T \Delta \mathbf{g}}. \quad (3.9)$$

Therefore the unique symmetric rank 1 update formula that satisfies the quasi-Newton condition in (3.3) is

$$\mathbf{B}^* = \mathbf{B} + \frac{(\Delta \mathbf{w} - \mathbf{B} \Delta \mathbf{g})(\Delta \mathbf{w} - \mathbf{B} \Delta \mathbf{g})^T}{(\Delta \mathbf{w} - \mathbf{B} \Delta \mathbf{g})^T \Delta \mathbf{g}}. \quad (3.10)$$

This rank 1 update formula (3.10) has two deficiencies. First it does not maintain positive definiteness. Second the denominator in (3.10) may approach zero.

Rank 2 updates can be written as the sum of two rank 1 updates,

$$\mathbf{B}^* = \mathbf{B} + q_1 \mathbf{m}_1 \mathbf{m}_1^T + q_2 \mathbf{m}_2 \mathbf{m}_2^T. \quad (3.11)$$

The quasi-Newton condition in (3.3) should also be satisfied in this case, leading to

$$\Delta \mathbf{w} = \mathbf{B} \Delta \mathbf{g} + q_1 \mathbf{m}_1 \mathbf{m}_1^T \Delta \mathbf{g} + q_2 \mathbf{m}_2 \mathbf{m}_2^T \Delta \mathbf{g}. \quad (3.12)$$

Now vectors $\mathbf{m}_1$ and $\mathbf{m}_2$ are not unique, but useful choices for these two vectors could be

$$\mathbf{m}_1 = \Delta \mathbf{w} \quad (3.13a)$$

and

$$\mathbf{m}_2 = \mathbf{B} \Delta \mathbf{g}. \quad (3.13b)$$

Use of these two choices in (3.12) results in the scalar equalities $q_1 \mathbf{m}_1^T \Delta \mathbf{g} = 1$ and $q_2 \mathbf{m}_2^T \Delta \mathbf{g} = -1$. This resulting rank 2 update formula is

$$\mathbf{B}_{DFP}^* = \mathbf{B} + \frac{\Delta \mathbf{w} \Delta \mathbf{w}^T}{\Delta \mathbf{w}^T \Delta \mathbf{g}} - \frac{(\mathbf{B} \Delta \mathbf{g})(\mathbf{B} \Delta \mathbf{g})^T}{(\mathbf{B} \Delta \mathbf{g})^T \Delta \mathbf{g}}. \quad (3.14)$$

This formula is the famous and first quasi Newton formula introduced by Davidon and simplified by Fletcher and Powell.

Adding another additional update term in (3.12) will lead to the BFGS formula [62],

$$\begin{aligned} \mathbf{B}^* &= \mathbf{B}_{DFP}^* + r \mathbf{m} \mathbf{m}^T \\ \mathbf{m} &= (\Delta \mathbf{g}^T \mathbf{B} \Delta \mathbf{g})^{1/2} \left[\frac{\Delta \mathbf{w}}{\Delta \mathbf{g}^T \Delta \mathbf{w}} - \frac{(\mathbf{B} \Delta \mathbf{g})}{(\mathbf{B} \Delta \mathbf{g})^T \Delta \mathbf{g}} \right], \quad r = 1. \end{aligned} \quad (3.15)$$

Since this additional term is a linear combination of $\Delta \mathbf{w}$ and $\Delta \mathbf{g}$, the update formula in (3.15) is still a rank-two update formula. It can be verified by direct substitution that the vector $\mathbf{m}$ is orthogonal to $\Delta \mathbf{g}$,

$$\mathbf{m}^T \Delta \mathbf{g} = (\Delta \mathbf{g}^T \mathbf{B} \Delta \mathbf{g})^{1/2} \left[\frac{\Delta \mathbf{w}^T \Delta \mathbf{g}}{\Delta \mathbf{g}^T \Delta \mathbf{w}} - \frac{(\mathbf{B} \Delta \mathbf{g})^T \Delta \mathbf{g}}{(\mathbf{B} \Delta \mathbf{g})^T \Delta \mathbf{g}} \right] = 0. \quad (3.16)$$

Hence any multiple of the rank 1 matrix $\mathbf{m} \mathbf{m}^T$ may be added to $\mathbf{B}^*$ without affecting the satisfaction of the quasi-Newton condition (3.3). Equation (3.15) with different r represents a set of quasi Newton formulas called Broyden family of inverse update formulas. The DFP and BFGS formulas anchor the ends of the Broyden family with $r = 0$ and $r = 1$ in (3.15). The name BFGS formula is commonly used to indicate that it was independently discovered by Broyden, Fletcher, Goldfarb, and Shanno, all in 1970.

Corresponding to (3.3) there are direct update formulas based on a matrix, say $\mathbf{A}$, that approximates the exact Hessian matrix $\mathbf{H}$. These formulas may be developed by simply interchanging the $\mathbf{A}$ for $\mathbf{B}$, $\Delta \mathbf{g}$ for $\Delta \mathbf{w}$, $\Delta \mathbf{w}$ and $\Delta \mathbf{g}$. Formulas related by interchanges of

variables are said to be *duals*. The rank 1 formula in (3.10) is self dual. The direct BFGS update formula is dual to the inverse DFP update formula and vice versa.

It was generally recognized that the DFP formula did not perform well with inexact line searches as the BFGS formula. Therefore in practice the BFGS update formula is usually implemented.

3.2.2 Steps of Quasi Newton Methods

The steps of the quasi Newton algorithm for minimizing a nonlinear function are described in this section. The steps are very similar to what we explained in Section 2.4.3 of the gradient-based optimization methods [50].

Step 1: Start with an initial point $\mathbf{w}^0$ of the nonlinear function. The initial estimate of the inverse Hessian matrix is set to identity matrix, i.e., $\mathbf{B}^0 = \mathbf{I}$. The $\mathbf{g}^0 = \nabla E(\mathbf{w}^0)$ is the related gradient vector. The iteration counter k is set to zero.

Step 2: Perform the one-dimensional line search $\mathbf{w}^{k+1} = \mathbf{w}^k + \lambda \mathbf{h}$ to find the optimal step size along the direction $\mathbf{h}$. The search direction $\mathbf{h}$ is determined by the equation $\mathbf{h} = -\mathbf{B}^k \mathbf{g}^k$.

Step 3: Terminate if the convergence criterion is met; otherwise go to **Step 4**

Step 4: Obtain a positive definite estimate $\mathbf{B}^{k+1}$ through a rank 1 or rank 2 update of the inverse Hessian matrix approximation $\mathbf{B}^k$

Step 5: Increment $k = k+1$ and go to **Step 2**.

3.2.3 Hereditary Positive Definiteness Condition

As stated in the beginning of Section 3.1, it is usually desirable for the approximating matrices of the inverse Hessian matrix ($\mathbf{B}^k$) or the Hessian matrix ($\mathbf{A}^k$) remaining to be

positive definite so that search direction would always be a descent direction. In other words, it is to require that the update formulae possess the property of *hereditary positive definiteness*, i.e., if $\mathbf{B}^k$ is positive definite, $\mathbf{B}^{k+1}$ is positive definite.

Here we would outline a proof of hereditary positive definiteness condition for the direct BFGS update formula [81]. Similarly this proof can be extended to other update formulae. The direct BFGS update formula is dual of the inverse DFP update formula (3.14), given by

$$\mathbf{A}_{BFGS}^* = \mathbf{A} + \frac{\Delta \mathbf{g} \Delta \mathbf{g}^T}{\Delta \mathbf{g}^T \Delta \mathbf{w}} - \frac{(\mathbf{A} \Delta \mathbf{w})(\mathbf{A} \Delta \mathbf{w})^T}{(\mathbf{A} \Delta \mathbf{w})^T \Delta \mathbf{w}}. \quad (3.17)$$

If $\mathbf{A}$ is positive definite, there must exist a non-singular matrix $\mathbf{R}$ that

$$\mathbf{A} = \mathbf{R}^T \mathbf{R}. \quad (3.18)$$

The formula (3.17) can be written as

$$\mathbf{A}^* = \mathbf{R}^T \mathbf{V} \mathbf{R}, \quad (3.19)$$

where the matrix $\mathbf{V}$ is given by

$$\mathbf{V} = \mathbf{I} + \frac{1}{\mathbf{s}^T \mathbf{r}} \mathbf{s} \mathbf{s}^T - \frac{1}{\mathbf{r}^T \mathbf{r}} \mathbf{r} \mathbf{r}^T, \quad (3.20)$$

with

$$\mathbf{r} = \mathbf{R} \Delta \mathbf{w}, \quad \mathbf{s} = (\mathbf{R}^T)^{-1} \Delta \mathbf{g}. \quad (3.21)$$

Equation (3.19) indicates that $\mathbf{A}^*$ will be positive definite if $\mathbf{V}$ is positive definite. Since $\mathbf{V}$ is a rank 2 modification of the identity matrix, it has $N_w - 2$ unit eigenvalues if $N_w \geq 3$. To verify this, let us consider an N_w dimensional space. Since $\mathbf{s}^T \mathbf{r} \neq 0$ (otherwise the denominator in (3.20) would be zero), $\mathbf{s}$ and $\mathbf{r}$ are linearly independent. Therefore we could find another $N_w - 2$ linearly independent vectors $\mathbf{u}_i$ ($i=3, 4, \dots, N_w$) which are

orthogonal to both s and r . These $N_w - 2$ vectors can be proved to be the eigenvectors of V corresponding to unit eigenvalue by multiplying u_i to (3.20),

$$Vu_i = u_i, \quad i = 3, \dots, N_w. \quad (3.22)$$

Let λ_1 and λ_2 denote the two remaining eigenvalues. Since the eigenvectors u_1 and u_2 corresponding to λ_1 and λ_2 are linearly independent to u_i , u_1 and u_2 must be linear combinations of s and r . We assume

$$u = ar + bs. \quad (3.23)$$

To obtain the a , b and the corresponding λ we use the relationship between eigenvalue and eigenvector,

$$Vu = aVr + bVs = \lambda u = a\lambda r + b\lambda s, \quad (3.24)$$

where

$$Vr = r + \frac{ss^T r}{s^T r} - \frac{rr^T r}{r^T r} = s \quad (3.25a)$$

and

$$Vs = s + \frac{ss^T s}{s^T r} - \frac{rr^T s}{r^T r} = s(1 + \frac{s^T s}{s^T r}) - r \frac{r^T s}{r^T r}. \quad (3.25b)$$

By matching the coefficients of vector r and s in (3.24) we have

$$a + b(1 + \frac{s^T s}{s^T r}) = b\lambda \quad (3.26a)$$

and

$$-b \frac{r^T s}{r^T r} = a\lambda. \quad (3.26b)$$

Let $k = \frac{a}{b}$, we have

$$k + (1 + \frac{s^T s}{s^T r}) = \lambda \quad (3.27a)$$

and
$$-\frac{\mathbf{r}^T \mathbf{s}}{\mathbf{r}^T \mathbf{r}} = k\lambda. \quad (3.27b)$$

By replacing λ in (3.27b) with (3.27a) we have

$$k^2 + (1 + \frac{\mathbf{s}^T \mathbf{s}}{\mathbf{s}^T \mathbf{r}})k + \frac{\mathbf{r}^T \mathbf{s}}{\mathbf{r}^T \mathbf{r}} = 0. \quad (3.28)$$

Using the relationship between roots and coefficients of the quadratic equation with one unknown, we have

$$k_1 + k_2 = -(1 + \frac{\mathbf{s}^T \mathbf{s}}{\mathbf{s}^T \mathbf{r}}) \quad (3.29a)$$

and
$$k_1 k_2 = \frac{\mathbf{r}^T \mathbf{s}}{\mathbf{r}^T \mathbf{r}}. \quad (3.29b)$$

Combining (3.27) with (3.29) we have

$$\lambda_1 + \lambda_2 = (1 + \frac{\mathbf{s}^T \mathbf{s}}{\mathbf{s}^T \mathbf{r}}) \quad (3.30a)$$

and
$$\lambda_1 \lambda_2 = \frac{\mathbf{r}^T \mathbf{s}}{\mathbf{r}^T \mathbf{r}}. \quad (3.30b)$$

Since $\mathbf{A}$ is positive definite, it holds that $\mathbf{r}^T \mathbf{r} = \Delta \mathbf{w}^T \mathbf{R}^T \mathbf{R} \Delta \mathbf{w} = \Delta \mathbf{w}^T \mathbf{A} \Delta \mathbf{w} > 0$, and consequently both λ_1 and λ_2 will be positive if $\mathbf{r}^T \mathbf{s}$ is positive. Since $\mathbf{r}^T \mathbf{s} = \Delta \mathbf{w}^T \mathbf{R}^T (\mathbf{R}^T)^{-1} \Delta \mathbf{g} = \Delta \mathbf{w}^T \Delta \mathbf{g}$, the update formula (3.17) would have the property of hereditary positive definiteness if and only if [81]

$$\Delta \mathbf{w}^T \Delta \mathbf{g} > 0. \quad (3.31)$$

The derivation of the hereditary positive definiteness condition can be formulated in a similar manner for other update formulae. Since condition (3.31) is self-dual, this condition would also hold for DFP update formula.

During the optimization process, condition (3.31) can not be always satisfied. Actually this condition roughly implies that the second-order derivatives of the real error function in the range should be positive definite, i.e., the function curve is concave (this is a necessary condition for the existence of a local minimum). However it is not possible to guarantee the function curve is always concave. Therefore the original quasi Newton method needs to be modified if condition (3.31) is not satisfied.

One way is to reset the approximation of the inverse Hessian matrix to identity matrix and that means we restart the approximation process. This method in most cases is not preferable because we actually restart from the steepest descent direction and consequently slow down the convergence of optimization. Another way is to reuse the approximation matrix B of last iteration in *Step 2*.

In the implementation the first method is used after some user-defined number of epochs, i.e., the approximation of the inverse Hessian matrix would restart from identity matrix after certain number of epochs (we call it Hessian refresh interval). The second method is used whenever the hereditary positive definiteness condition (3.31) is violated. This approach is proven to be very successful in the training.

3.3 Examples

In this section the quasi Newton method is compared with other popular training algorithms for standard feedforward neural networks such as adaptive backpropagation, conjugate gradient, and Levenberg-Marquardt methods. Two examples, i.e., three conductor microstrip line and physics-based MESFET, were used to illustrate the performance of these algorithms.

The first example is a microstrip line example. The model has 5 input neurons corresponding to conductor width (w), spacing between conductors (s_1, s_2), substrate height (h) and relative permittivity (ϵ_r) as shown in Figure 3.1. There are 6 output neurons corresponding to the self inductance of each conductor l_{11}, l_{22}, l_{33} and the mutual inductance between any two conductors l_{12}, l_{23}, l_{13} . There are totally 600 training samples and 640 test samples generated by LINPAR [85]. A three layer MLP structure with 28 hidden neurons is chosen as the sample structure. The training results are shown in Table 3.1. CPU time is given for Pentium (200 MHz).

Figure 3.1: Three conductor microstrip line.

Training algorithm	No of Epochs	Training Error (%)	Avg. Test Error (%)	CPU (in Sec)
Adaptive Backpropagation	10755	0.224	0.252	13724
Conjugate Gradient	2169	0.415	0.473	5511
Quasi-Newton	1007	0.227	0.242	2034
Levenberg-Marquardt	20	0.276	0.294	1453

Table 3.1: Comparison of Various Training Algorithms for Microstrip Line Example.

Quasi-Newton method achieved the best accuracy (0.242%) with about 34 minutes.

The testing error (0.294%) obtained by the Levenberg-Marquardt method (LM) is

slightly higher than quasi-Newton method, but the total CPU time used is relatively shorter (around 24 minutes). The adaptive backpropagation used many epochs and settled down to good accuracy of 0.252%, with around 4 hours of training time. The training time of Conjugate Gradient method is the longest (about 1.5 hours) among the second-order training methods and testing error is also higher. This example has comparatively small number of neurons (28 hidden neurons). Therefore Levenberg-Marquardt method has faster convergence. However for the large neural networks LM will be slow owing to the repetitive computation of LU decomposition. Therefore the quasi-Newton method would be preferable when the large neural model is needed and training time tends to be long. Since for smaller models the overall training time would not be excessively long, quasi-Newton method could also be considered as a general choice.

The second example is a physics-based transistor device model. Neural models learned from physics-based FET data can predict the scattering parameters from physical and electrical parameters of the device. This MESFET model is based on the Khatibzadeh and Trew model [83]. The training data is obtained by using OSA90[84] with this model. The inputs to neural model include frequency (f), channel thickness (a), gate-bias voltage (V_g), and drain-bias voltage (V_d). This particular MESFET has a fixed gate length of 0.55 *micron* and gate width of 1*mm*. The outputs include real and imaginary parts of S_{11} , S_{12} , S_{21} and S_{22} . The original simulated training data contains large/catastrophic errors. These erroneous data points have been removed manually before the data being used in the training. In

order to test the robustness of the Huber quasi-Newton method this example is also used in next chapter where the neural network is trained by the original data.

This example is relatively complicated compared to the microstrip line example. The neural network structure has 60 hidden neurons. The training results by the 4 training algorithms are shown in Table 3.2. The quasi-Newton achieved the best result in both aspects, i.e., CPU time and average testing error. This demonstrates that as the size of the neural network becomes large, Levenberg-Marquardt becomes slow as compared to quasi-Newton method.

Training algorithm	No of Epochs	Training Error (%)	Ave. Test Error (%)	CPU (in Sec)
Adaptive Backpropagation	15319	0.98	1.04	11245
Conjugate Gradient	1605	0.99	1.04	4391
Quasi-Newton	570	0.88	0.89	1574
Levenberg-Marquardt	12	0.97	1.03	4322

Table 3.2: Comparison of Various Training Algorithms for MESFET Example.

Chapter 4

Huber Quasi Newton Method

4.1 Introduction

Fast and efficient model of neural networks for microwave device can be developed by learning and abstracting from microwave data. Once trained, the neural networks can be reused many times during microwave design to provide instant answers. This saves the computational intensive EM simulation or the parameter extraction of the device [72][73]. The costs for developing neural models are mainly data collection and neural network training.

In microwave applications of neural networks training data is obtained by either simulation of the original EM or device/physics problem, or by measurement. In the process of data collection large (catastrophic) errors are often introduced, e.g., measurement performed at the limit of equipment frequency range, or data simulated at extreme parameter values leading to non-convergence or large roundoff/truncation errors. The conventional supervised training algorithms for neural networks are in least-squares (l_2) formulation, which use the l_2 norm of the difference between the network output and the training data as the error function.

Such methods are susceptible to the influence of gross errors. A few wild data points can alter the least-squares solution significantly. The l_1 method is robust against gross errors [74] but is not very effective for dealing with small errors, i.e., data within typical tolerance range which is actually the criterion of the neural networks training. In other words neither the l_1 nor the l_2 methods can discriminate and treat differently large (catastrophic) errors and small (smooth) errors during training. However both errors may exist in same data set.

To address this problem in this thesis the Huber function[75][76] is introduced to formulate the error function of the neural networks. Huber concept has been successfully applied in circuit modeling and optimization[77][78] and proved to be a unique numerical technique. Another similar approach in[79] proposes a weighted least squares training method to eliminate outlier data which may appear in the erroneous training data. Huber function is a combination of the l_1 and l_2 norms. The large errors are treated in the l_1 sense and the small errors are measured in terms of least squares. Consequently the Huber solution can provide a smooth error formulation of the neural networks from data which contains both small variations and gross errors. The second-order optimization algorithms can be applied to this training of the neural networks. Since the number of trainable variables (weights) of neural networks is very large, numerical methods involving LU factorization, such as Gauss-Newton method or Levenberg-Marquardt method, will be computational intensive and therefore impractical for this situation. In this thesis a new microwave-oriented training algorithm [80] is proposed, using the quasi-Newton methods (BFGS)[81] to optimize the neural networks where the Huber function is

incorporated for defining the error function. The advantages of this Huber quasi Newton method (HQN) in the presence of large and small errors is illustrated using a wide range of applications, including the modeling of Quadratic Function, Striplines, Heterojunction Bipolar Transistors (HBT), and MESFET. The application of HQN in Iterative Multi-Stage (IMS) training is also demonstrated through an example of the Microstrip Filter modeling.

4.2 Formulation of the Huber Training Method

Let us reintroduce the error between the desired outputs d_p and actual outputs y_p for each sample in (2.4),

$$e_p(\mathbf{x}_p, \mathbf{w}) = \sqrt{\sum_{k=1}^{N_p} (y_{pk}(\mathbf{x}_p, \mathbf{w}) - d_{pk})^2}. \quad (4.1)$$

The Huber-based error function is defined as [75]

$$\min_{\mathbf{w}} E(\mathbf{w}) = \sum_{p=1}^P \rho(e_p(\mathbf{x}_p, \mathbf{w})). \quad (4.2)$$

where $\mathbf{w}$ is the set of variables and $\rho(e)$ is the Huber function defined as

$$\rho(e) = \begin{cases} e^2 / 2, & \text{if } |e| \leq k \\ k|e| - k^2 / 2, & \text{if } |e| > k \end{cases} \quad (4.3)$$

where k is the Huber constant with positive value and e is the error function value for each sample.

The Huber function $\rho(e)$ has both least squares (l_2) (when $|e| \leq k$) norm and l_1 (when $|e| > k$) norm. As illustrated in Figure 4.1, the error (e) of each sample will be treated in the l_1 sense when the error is above the threshold k . Since in the supervised learning of neural model the error (e) is the difference between the

model output and the actual data, large errors (greater than the threshold) may represent the false data points which have been caused by measurement limit, convergence difficulty of simulator or file corruption. The l_1 is robust against gross errors in data [74] and therefore will not be sensible to those errors.

The Huber function defines a smooth transition between l_1 and l_2 at $|e| = k$, i.e., both function value and first-order derivative of the function at this point is continuous. The choice of k defines the threshold between the “large” and “small” errors. By varying the constant, the proportion of error functions to be treated in the l_1 or l_2 sense can be altered. If k is sufficiently large, this formulation (4.2) becomes the conventional least squares objective function. On the other hand, as the constant approaches zero, $\rho(e)/k$ will approach the l_1 error formulation.

Figure 4.1: The Huber, l_1 and l_2 functions.

In order to apply the gradient-based optimization method such as quasi-Newton method to minimize the Huber-based objective function of neural models the gradient of the objective function need to be derived.

The gradient of the Huber objective function (4.2) is

$$\nabla E(\mathbf{w}) = \sum_{p=1}^{N_p} v_p e_p', \quad (4.4)$$

where

$$v_p = \frac{\partial \rho(e_p(\mathbf{w}))}{\partial e_p(\mathbf{w})} = \begin{cases} e_p(\mathbf{w}) & \text{if } |e_p(\mathbf{w})| \leq k \\ \pm k & \text{if } |e_p(\mathbf{w})| > k \end{cases}, \quad (4.5)$$

$$e_p' = \left[\frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}_1} \quad \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}_2} \quad \dots \quad \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}_{N_w}} \right], \quad (4.6)$$

and N_p is the total number of samples.

The form of (4.4) is very similar to the gradient of least squares objective function [77], which is

$$\nabla E_{l_2}(\mathbf{w}) = \sum_{p=1}^P e_p e_p'. \quad (4.7)$$

The difference between (4.4) and (4.7) is the coefficients of the gradient of each sample. In the gradient of l_2 error function the coefficients are the function values which will give more weights to the large errors. In the Huber gradient when the function values are larger than the threshold, the coefficients become constant. When the function values are smaller than the threshold the coefficients are same as in l_2 formulation. Through this mechanism Huber function gives different emphasis to large errors and small errors.

For the neural network training problem, the error of each sample e_p and its derivatives with respect to $\mathbf{w}$ (4.6) could be obtained through the backpropagation process described in Section 2.2. And the Huber-based error function (4.2) and its derivatives (4.4) would be just the summation of these values. Then the quasi-Newton training method described in Chapter 3 could be applied for the training of neural networks, i.e., to minimize the error function $E(\mathbf{w})$.

4.3 Examples

The first four examples are used to demonstrate the robustness of HQN when the training data contains large errors. There are two sets of data in each example. Set A contains the large (catastrophic) and small errors. Set B is almost the same as set A except that the bad data points (large errors) have been removed or replaced. A strategy of comparison is designed and shown in Table 4.1.

Training Methods	Training Data	Testing Data
Case 1: l_2 Methods	Set A	Set B
Case 2: Huber quasi-Newton Method	Set A	Set B
Case 3: l_2 Methods	Set B	Set B
Case 4: Huber quasi-Newton Method	Set B	Set B

Set A – Original data with large errors, Set B – Training data without large errors

Table 4.1: Strategy of Comparison between Huber quasi-Newton Method and l_2 Methods.

From the analysis of HQN several observations could be expected from the comparison between different cases in Table 4.1 in terms of testing errors:

1. Comparison between Case 1 and Case 2: HQN is better than l_2 methods.
2. Comparison between Case 3 and Case 4: HQN is similar to l_2 methods.
3. Comparison between Case 2 and Case3/4: Case 2 should be as good as Case 3/4.
4. In case of more than enough neurons, HQN overlearns less than l_2 methods.

4.3.1 A Quadratic Model

This is a simple and illustrative example. The model has 1 input and 1 output with the relationship of $y=x^2$. Three layer perceptrons (MLP3: 1-5-1) were used to model this relationship. In data Set A, large errors were deliberately introduced at 2 sample points and small variations were added to the remaining data. Sample points in data set B only has small errors added.

Numerical comparison of four cases is listed in Table 4.2. The average and worst-case training errors of HQN in Case 2 is much larger than the training errors of l_2 methods in Case 1. This is because HQN did not follow the wrong data points and therefore has very big worst-case error. On the other hand the testing errors of HQN in Case 2 are much smaller than the corresponding errors of l_2 methods in Case 1 which means HQN yields correct model. When both training methods use data Set B as training data, i.e., Case3 and 4, the testing error of HQN (0.05%) are almost as good as that of l_2 methods (0.03%). Further, HQN model trained by data Set A (Case 2) can produce a model which is also very close to the model of Case 4, this can be more obviously observed from Figure 4.2.

Figure 4.2 shows the comparison between neural model and training data in Case 1 (a) and Case 2 (b). As expected, the l_2 solution suffers significantly from the presence of those 2 erroneous points. This could also be considered as an overlearning or overfitting [82]. With the same number of hidden neuron, HQN (b) is able to avoid overfitting in this example due to a different treatment of large errors. This shows the robustness of the HQN in another sense.

Training method	Test Data using Set A		Test Data using Set B	
	Average error (%)	Worst-case error (%)	Average error (%)	Worst-case error (%)
Case 1: l_2 methods	2.2	5.8	26	210
Case 2: Huber quasi-Newton	6.2	49.5	0.24	0.62
Case 3: l_2 methods	--	--	0.03	0.07
Case 4: Huber quasi-Newton	--	--	0.05	0.09

Table 4.2: Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for Quadratic Example.

(a)

(b)

Figure 4.2: Comparison between neural model and training data of quadratic example: (a) Case 1: Neural model trained by the l_2 methods. Model is affected by the bad data points. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method. Model is not affected by the bad data points.

4.3.2 A Stripline Model

Transmission line models are essential for delay and crosstalk analysis in high-speed VLSI interconnect design [16]. A 2 conductor stripline model is considered in this example. There are 3 inputs and 1 output. The 3 inputs are width of 1st conductor (w_1), width of 2nd conductor (w_2), and the separation between 2 conductors (s). The output is the mutual inductance between two conductors. MLP3 (3-12-1) was adopted in this example. Training data was obtained using LINPAR[85] simulator. To emulate the case of corrupt or contaminated data, some wrong data points with the inductance value of $-2e-8$ (the inductance value should always be nonnegative) were deliberately added to the original simulated data to make data Set A. The original simulated data is used as Set B.

Table 4.3 shows the numerical comparison of the same four cases. A similar conclusion can be drawn out of this table. Since in this example the bad data points were introduced by replacing the right values in Set B instead of removing, the worst-case error in Case 1 when tested using Set B is much larger (76%) than in Case 2 (0.7%). In the following 2 examples the bad data samples were removed from Set B and the difference of worst-case error in Case 1 and 2 may not be so obvious. Therefore in this example the overall performance of Case 2 is much better than Case 1 according to the testing error. Similarly the comparison between neural model and training data of Case 1 (a) and Case 2 (b) is shown in Figure 4.3.

For illustration of details of the difference among neural models of different cases, in Figure 4.4 the relationship of the mutual inductance L_{12} versus the width of the first conductor w_1 is displayed for Case 1, 2 and 3 in the place of wrong data points. Two curves are corresponding to different space s . The sudden falling-down-up in the display

of Case 1 (a) is caused by the data points with negative inductance value. Display in Case 2 is very similar to Case 3 and both models give nonnegative inductance values.

Training method	Test Data using Set A		Test Data using Set B	
	Average error (%)	Worst-case error (%)	Average error (%)	Worst-case error (%)
Case 1: l_2 methods	1.22	20	1.73	76
Case 2: Huber quasi-Newton	1.34	76	0.09	0.7
Case 3: l_2 methods	--	--	0.05	0.33
Case 4: Huber quasi-Newton	--	--	0.09	0.7

Table 4.3: Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Method for Stripline Example.

Figure 4.3: Comparison between neural model and training data of Stripline example, data samples with negative inductance values are bad samples: (a) Case 1: Neural model trained by l_2 methods. Model is affected by the bad data samples. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method. Model is not affected by the bad data samples.

Figure 4.4: Model display comparison of Stripline example: (a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.

4.3.3 Heterojunction Bipolar Transistor (HBT) Model

For the development of microwave circuit applications using heterojunction bipolar transistors (HBT's), it is essential to use an accurate HBT equivalent circuit model for simulating monolithic microwave integrated circuit (MMIC). In this example a HBT neural model is developed. The HBT model has three inputs namely, frequency (f), bias voltage (V_B) and bias current (I_B). The outputs of the model are the two-port scattering parameters S_{11} , S_{12} , S_{21} , and S_{22} . Magnitude and phase of S-parameters are measured using HP Network Analyzer. Since the phases of scattering parameters are discontinuous at some points, the measured data is converted to real and imaginary parts of S_{11} , S_{12} , S_{21} , S_{22} . The training data has small errors in most of the samples, and occasionally there are some large errors mostly due to extreme frequency samples. Two sets of data are used to train and test the neural model. Set A is the original data with small and large errors. Set B is obtained by pre-processing set A, i.e., samples with large errors in set A are manually identified and removed.

It can be seen from Table 4.4 and Figure 4.5 that HQN yields accurate neural models as compared to standard l_2 method. Figure 4.5 shows the model accuracy comparison of the 6th output (imaginary part of S_{21}) of both neural models.

In Table 4.4 it is noticed that Huber method gets much better result in terms of worst-case error when tested by Set B. That is due to the wrong measurement data points which happen at the beginning of the frequency sweep (at the beginning of each continuous curve). The training data is measured at the frequency range of (0.1GHZ, 40GHZ) and the bad points are at 0.1 GHZ. In Figure 4.5 it is observed

that the neural model response of Case 1 is distorted in these points. These wrong measurement data may be attributed to the equipment limit.

As in the Stripline example the input-output relationships of the trained neural models are compared. Figure 4.6 shows the frequency response of the imaginary part of S_{21} . Since the bad data points are in the lower frequency range, the performance of Case 1 at low frequency is significantly affected. Whereas the performance of Huber result in this range is almost same as in Case 3. Since Case 3 is trained by data without large error (Set B) we can expect this model will represent the correct model.

Training method	Test Data using Set A		Test Data using Set B	
	Average error (%)	Worst-case error (%)	Average error (%)	Worst-case error (%)
Case 1: l_2 methods	1.21	15	1.17	15
Case 2: Huber quasi-Newton	0.97	100	0.8	9
Case 3: l_2 methods	--	--	0.82	17
Case 4: Huber quasi-Newton	--	--	0.73	11

Table 4.4: Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for HBT Example.

(a)

(b)

Figure 4.5: Comparison between neural model and training data of HBT example: The training data has large errors at the beginning of every frequency sweep. (a) Case 1: Neural model trained by standard l_2 method is affected by large errors. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method is not affected by large errors. 200 samples means sweep at 50 frequency points at 4 different bias points.

Figure 4.6: Model display comparison of HBT example:(a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.

4.3.4 MESFET Example

This example is a physics-based MESFET model, which has been introduced in Section 3.3. The MESFET model has four inputs namely, frequency (f), drain voltage (V_D), gate voltage (V_G) and channel thickness (a). There are eight outputs, i.e., real and imaginary parts of two-port S-parameters S_{11} , S_{12} , S_{21} , and S_{22} . There are a few obvious large errors in data that can be attributed to non-convergence of the simulation due to input samples in extreme locations. MLP3 (4-30-8) was adopted in this example. This is a fairly large example, there are 30 hidden neurons in the hidden layer and the size of the optimization variables w is approximately $(8+4) * 30 = 360$.

From the model accuracy comparison of Case 1 and 2 in Figure 4.7 it can be observed that the original data contains 9 bad points. Those bad points may be originated from the convergence difficulty of the simulator because those abnormal points have similar value and show up periodically. The neural model in Case 2 was not biased by the bad points so much as in Case 1. Correspondingly in Table 4.5 the average testing error of Huber method using data without large error (Set B) is much better than l_2 methods. When there is no erroneous points in the training data (Case 3 and 4) the accuracy of neural models in both cases are comparable.

Figure 4.8 shows the display of input-output relationship of the trained neural models at wrong data points. The first 3 bad points are selected which occur when the frequency (f) and drain bias voltage (V_D) are constant ($f=14\text{Ghz}$, $V_D=1.5\text{volts}$). Therefore the channel thickness (a) is set as X-axis, and 2 curves correspond to different gate bias voltage (V_G). Analogously 2 curves in the display of Case 1 have jumps where the 3 bad points reside. However the displays in Case 2 and Case 3 are similarly smooth.

Training method	Test Data using Set A		Test Data using Set B	
	Average error (%)	Worst-case error (%)	Average error (%)	Worst-case error (%)
Case 1: l_2 methods	2.21	12	2.14	12
Case 2: Huber quasi-Newton	2.59	68	1.64	33
Case 3: l_2 methods	--	--	1.69	13
Case 4: Huber quasi-Newton	--	--	1.66	14

Table 4.5: Model Accuracy Comparison between Huber-quasi-Newton Method and l_2 Methods for MESFET Example.

Figure 4.7: Comparison between neural model and training data of MESFET example: The training data has a few large/gross errors. (a) Case 1: Neural model trained by standard l_2 method is affected by large errors. (b) Case 2: Neural model trained by the proposed Huber-quasi-Newton method is not affected by large errors.

Figure 4.8: Model display comparison of MESFET example:(a) Case 1: result of l_2 methods using Set A. (b) Case 2: result of Huber-quasi-Newton method using Set A. (c) Case 3: result of l_2 methods using Set B.

4.3.5 Microstrip Filter

In [86], a Multi-Stage training algorithm that incorporates Huber quasi-Newton (HQN) technique, was proposed to address highly non-linear and non-smooth modeling problems. The multi-stage algorithm decomposes the original difficult problem into simpler sub-tasks and solve them at different stages. To help the IMS approach in one of the training stages where only the smooth portion of the problem behavior needs to be modeled ignoring large/sudden variations, the HQN training method is used.

In this example, neural network model of a microstrip filter is developed. The model has four inputs namely, frequency (f), width of sections (w_d), and gaps between sections (g_{12} and g_{23}). There are two outputs namely, real and imaginary parts of the reflection coefficient (S_{11}). Data is generated from simulation using *Serenade* [87]. Thirty-six filters with different width and gaps are considered. For each filter, real and imaginary parts of S_{11} are simulated at 201 frequency points, resulting in a total of 7236 data samples. The outputs are, in general, smooth over the entire frequency range. However, each filter exhibits sharp/large variations at a couple of frequencies. Conventional one-shot training by standard l_2 methods to capture both smooth and sudden variations in just one stage has been tried, but it did not yield satisfactory models, because over-emphasis on large variations at resonant frequencies led to a poor fit at other frequencies. The multi-stage algorithm is applied. In the first stage, smoothly varying portions of the S-parameters are modeled. The neural network structure and training algorithm used in this stage are 3-layer MLP and HQN respectively. The HQN algorithm de-emphasizes large variations in training data and models only the smooth portions. The model after this

training stage is shown in Figure 4.9. In the second stage, a suitable neural network with knowledge neurons is chosen to model non-smooth portions of the filter response. The number of knowledge neurons is chosen to be 2, because each filter exhibits large variations twice in the frequency range. The knowledge neuron activation function is given by,

$$g(f, f_0, h, c) = \frac{c h}{(f - f_0)^2 + c} \quad (4.12)$$

where f represents the frequency, f_0 represents the dip frequency at which large variation of the filter response is observed, h represents the amplitude of variation or dip height, and c regulates the width of the dip. The parameters f_0 , h and c in the knowledge function of (4.12) supplied by the neuron outputs of previous layer for which the geometrical parameters g_{12} , g_{23} and w_d , are inputs. The sub neural model of second stage captured large/sudden variations in data. To model the highly non-linear training data in the third stage, a 4-layer MLP neural network was chosen. The sub neural models developed in the above stages are combined to produce the overall model. The overall model is observed to predict real and imaginary parts of S_{11} with a respectable accuracy as shown in Figure 4.10. The stage-wise training results are presented in Table 4.6. The sub model ($\tilde{Y}^m$) is the model obtained in every stage and the accumulated model ($\tilde{y}^m$) is the model accumulated from all the trained sub models up to current stage.

Stage Number m	Average Test Error of Accumulated Model $(\tilde{y}^m)$	Average Test Error of Sub Model $(\tilde{Y}^m)$	Neural Network Structure	Training Algorithm
1	2.16%	2.16%	MLP3	HQN
2	1.54%	2.29%	KBNN	l_2
3	0.34%	0.52%	MLP4	l_2

Table 4.6: Stage-Wise Training Results for S_{II} of Microstrip Filter Example Using the Proposed IMS Training Approach.

Figure 4.9: A comparison between neural model and training data for two out of the thirty-six microstrip filters after the first stage training using HQN method. (a) and (b) filter with normalized values of w_d , g_{12} , and g_{23} as 0.5, 2.0 and 2.0 respectively; (c) and (d) filter with normalized values of w_d , g_{12} , and g_{23} as 1.0, 1.5 and 2.0 respectively.

Figure 4.10: A comparison between final neural model and training data for two out of the thirty-six microstrip filters modeled by the proposed IMS approach. (a) and (b) filter with normalized values of w_d , g_{12} , and g_{23} as 0.5, 2.0 and 2.0 respectively; (c) and (d) filter with normalized values of w_d , g_{12} , and g_{23} as 1.0, 1.5 and 2.0 respectively.

Chapter 5

Simplex Method

5.1 Mathematical Description

The downhill simplex method was developed by Nelder and Mead [88]. This method requires only function evaluations. It is not efficient in terms of the number of function evaluations that it requires. However the downhill simplex method may still be a good choice to get something working when you have a problem whose analytical formula of derivatives is not available or too cumbersome to implement. In that situation all the local optimization methods which require the derivatives, such as back-propagation, Conjugate Gradient method, quasi Newton method and Levenberg-Marquardt method, would not be applicable or efficient (if derivatives are obtained through perturbation). Since this method explores the error space based on its geometrical tendency, it may not be effective when the problem is highly nonlinear or has higher dimensions. Since the simplex method does not require the derivative information it is used in next chapter as a random movement generator for the Simulated Annealing Algorithm.

A simplex is a geometric figure consisting of $N_w + 1$ points and all their interconnecting line segments, polygonal faces, etc, in an N_w -dimensional space [62].

Thus in two dimensions, a simplex is a triangle, and in three dimensions, a tetrahedron. In general we are only interested in simplexes that are nondegenerate, i.e., that enclose a finite inner N_w -dimensional volume. If any point of a nondegenerate simplex is taken as the origin, then the N_w other points define vector directions that span the N_w -dimensional vector space. If we start with an initial point w_0 , then we can take other N_w points w_i to be

$$w_i = w_0 + \lambda u_i \quad i = 1, 2, \dots, N_w \quad (5.1)$$

where u_i is the unit vector along the i^{th} coordinate axis.

The basic idea of simplex method is to compare the values of the objective function at the N_w+1 vertices of a general simplex and move the simplex gradually toward the optimum point during an iterative process. The movement of the simplex is achieved by using three operations, known as reflection, expansion and contraction [50].

Reflection:

If w_h is the vertex corresponding to the highest value of the objective function among the vertices of a simplex, we can expect the point w_r obtained by reflecting the point w_h in the opposite face to have the smallest value. If this is the case, we can construct a new simplex by rejecting the point w_h from the simplex and including the new point w_r . Since the direction of movement of the simplex is always away from the worst result, we will be moving in a favorable direction. If the objective function does not have steep valleys, repetitive application of the reflection process leads to a zigzag path in the general direction of the minimum. Mathematically, the reflected point w_r is given by

$$w_r = (1 + \alpha)w_m - \alpha w_h, \quad (5.2)$$

where w_h is the vertex corresponding to the maximum function value,

$$E(w_h) = \max_{i=0 \text{ to } N_w} E(w_i), \quad (5.3)$$

w_m is the center of all the points w_i except $i=h$,

$$w_m = \frac{1}{N_w} \sum_{\substack{i=0 \\ i \neq h}}^{N_w} w_i, \quad (5.4)$$

and $\alpha > 0$ is the reflection coefficient defined as

$$\alpha = \frac{\text{distance between } w_r \text{ and } w_m}{\text{distance between } w_h \text{ and } w_m}. \quad (5.5)$$

Thus w_r will lie on the line joining w_m and w_h , on the far side of w_m from w_h with $|w_r - w_m| = \alpha |w_h - w_m|$. If $E(w_r)$ lies between $E(w_h)$ and $E(w_l)$, where w_l is the vertex corresponding to the minimum function value,

$$E(w_l) = \min_{i=0 \text{ to } N_w} E(w_i), \quad (5.6)$$

then x_h is replaced by x_r and a new simplex is started.

Expansion:

If a reflection process gives a point w_r for which $E(w_r) < E(w_l)$, (i.e., if the reflection produces a new minimum), one can generally expect to decrease the function value further by moving along the direction pointing from w_m to w_r . Hence we expand w_r to w_e using the relation

$$w_e = (1 - \gamma)w_m + \gamma w_r, \quad (5.7)$$

where γ is called the expansion coefficient, defined as

$$\gamma = \frac{\text{distance between } w_e \text{ and } w_m}{\text{distance between } w_r \text{ and } w_m} > 1. \quad (5.8)$$

If $E(w_e) < E(w_h)$, we replace the point w_h by w_e and restart the process of reflection.

Contraction:

If the reflection process gives a point w_r for which $E(w_r) > E(w_i)$ for all i except $i = h$, i.e., the next-highest point in the simplex is better than w_r , we would perform the contraction process. We replace point w_h by w_r if $E(w_r) < E(w_h)$. The formula is given by

$$w_c = \beta w_h + (1 - \beta) w_m, \quad (5.9)$$

where β is called the contraction coefficient ($0 \leq \beta \leq 1$) and is defined as

$$\beta = \frac{\text{distance between } w_c \text{ and } w_m}{\text{distance between } w_h \text{ and } w_m}. \quad (5.10)$$

If the contraction process produces a point w_c for which $E(w_c) < E(w_h)$, we replace w_h by w_c and proceed with the reflection process again. On the other hand if $E(w_c) \geq E(w_h)$, the contraction process will be a failure, and in this case we replace all w_i by

$$w_i = (w_i + w_i) / 2 \quad (5.11)$$

and restart the reflection process.

5.2 Flow Diagram of Simplex Training Method

Unlike the gradient-based training methods, simplex method does not require the derivatives of the error function for either all samples as in the quasi-Newton method or individual sample as in HQN. This made the simplex method very attractive where the derivative information is either not available or not necessary. As we will see in next chapter the simplex method is modified to generate random movements in one of the global training algorithms, namely, Simulated Annealing Algorithm. There the derivative

information is not wanted as it may lead the training process to a local minimum. The simplex method is modified in such a way that the movement is more random in the early training stage and will eventually converge to a best point.

To apply the simplex method in the training of neural networks is relatively easy. The error function $E(\mathbf{w})$ and the trainable parameters $\mathbf{w}$ are already defined in Problem Statement Section in Chapter 2. The flow diagram of the simplex training method is shown in Figure 5.1. The simplex training method is implemented with the coefficients selected as $\alpha = 1$, $\beta = 0.5$, and $\gamma = 2$ respectively. These values are very standard values and can be easily understood geometrically following the definition of these parameters. The convergence criterion is set to be the pre-specified minimal difference between the highest and lowest error function values in the simplex.

Figure 5.1: Flow diagram of simplex training method.

5.3 Example

This example is a DC characteristic model of a Field Effect Transistor (FET). There are two inputs in the model namely the bias voltage between gate and source (V_{gs}) and bias voltage between drain and source (V_{ds}). The output of the model is the drain current (i_d). The model is trained by a three layer multilayer perceptrons (MLP) with 8 hidden neurons. The training data contains 78 samples. The training data is obtained by running a small simulator program using the empirical formula.

Table 5.1 shows the comparison of the CPU time and final accuracy of different training algorithms. The testing results are obtained using NeuroModeler on a Pentium 300MHz PC. The quasi Newton method achieves the best training error (0.41%) with least training CPU time (134 seconds). The simplex method achieve the second best accuracy (0.7%) after a lot of epochs (20,000). The conjugate gradient method takes shorter time than simplex method but finishes with a higher average training error. For the adaptive backpropagation, the training time is comparable to simplex method, while the average training error is much higher than the result of simplex method. From this example we can see the simplex method could also achieve very good result for problems of small dimensions. However the training process will take much more iterations than the second-order training methods.

Figure 5.2 shows the accuracy comparison between the model prediction and training data. The model response conforms to the training data very well.

Training algorithm	No of Epochs	Ave. Training Error (%)	Wst. Training Error (%)	CPU (in Sec)
Simplex method	20,000	0.70	2.70	540
Conjugate Gradient	626	0.94	3.96	186
Quasi-Newton	386	0.41	1.80	134
Adaptive Backpropagation	20,745	1.13	5.22	562

Table 5.1: Comparison of Different Training Algorithms for the FET Example.

Figure 5.2: Comparison between neural model and training data for the FET example.

Chapter 6

Simulated Annealing Algorithm

6.1 Introduction

All training techniques discussed so far are local optimization methods. The aim of these methods is just to locate a local optimum in the neighborhood of the starting point. Although fast in finding a minimum, they can be easily trapped in a local minimum and would require manual interference to escape from the trap. In many cases it is very difficult or impossible for the human to judge the locality of the minimum. The commonly used approach is to try many different starting points and select the best result. To overcome this there is another important class of training methods using random optimization techniques. These methods are characterized by a random search element in the training process allowing the algorithms to escape from local minima and converge to the global minimum of the objective function. Examples in this class are, e.g., simulated annealing which allows the optimization to jump out of a local minimum through an annealing process controlled by a parameter called temperature [89]; genetic algorithms which evolve the structure and weights of the neural network through generations in a manner that is similar to biological evolution [90]. Since the convergence of pure random

search techniques tends to be very slow, a more general method is the hybrid method which combines the local training algorithms with random optimization. The work in [60] introduced a hybrid algorithm combining the conjugate gradient method with line search, and the random optimization method to find the global minimum of the error function. During training with conjugate gradient method, if a flat error surface is encountered, the training algorithm switches to the random optimization method. After training escapes from the flat error surface, it once again switches back to conjugate gradient algorithm.

In this and the subsequent chapters, two global training methods, namely simulated annealing algorithm and genetic algorithm for the training of neural networks, will be discussed. A set of examples which have numerous local minima are presented to test the validity of these training methods.

6.2 Description of Simulated Annealing Algorithm

The method of simulated annealing is a technique that was first proposed [89] for its effectiveness in finding near optimal solutions for large scale combinatorial optimization problems, such as traveling salesperson problems (finding the shortest cyclical itinerary for a salesperson who must visit each of N cities in turn) and placement problems [91] (finding the arrangement of several hundred thousand circuit elements on a tiny silicon substrate that minimizes the interference among their connecting wires). Recent application of the simulated annealing algorithm and its variants [92] also demonstrates that this class of optimization approaches can be considered competitive with other approaches when there are continuous optimization problems to be solved.

Simulated annealing was derived from physical characteristics of spin glasses [89][93]. The principle behind simulated annealing is analogous to what happens when metals are cooled at a controlled rate. The slowly falling temperature allows the atoms in the molten metal to line themselves up and form a regular crystalline structure that has high density and low energy. But if the temperature goes down too quickly, the atoms do not have time to orient themselves into a regular structure and the result is a more amorphous material with higher energy.

In simulated annealing, the value of an objective function that we want to minimize is analogous to the energy in a thermodynamic system. At high temperatures, SA allows function evaluations at faraway points and it is likely to accept a new point with higher energy. This corresponds to the situation in which high-mobility atoms are trying to orient themselves with other nonlocal atoms and the energy state can occasionally go up. At low temperature, SA evaluates the objective only at local points and the likelihood of accepting a new point with higher energy is much lower. This is analogous to the situation in which the low-mobility atoms can only orient themselves with local atoms and the energy state is not likely to go up again.

The implementation of the SA is relatively simple. This general scheme of SA, of always moving in the downhill direction while sometimes taking an uphill direction is known as Metropolis algorithm [93]. This basic idea is also applicable to the optimization of problems with continuous N-dimensional control spaces. There are four elements required by the Metropolis procedure: system state, energy function, the control parameter T and a generator of random changes of the system state.

For the training of neural network, the first two required elements are correspondingly the trainable parameters $\mathbf{w}$, and the objective function $E(\mathbf{w})$. The control parameter T is a pseudo-temperature parameter with an annealing schedule by which it is gradually reduced. The generator of the change of the system state is problem dependent. As discussed in the introduction, the pure random optimization is very inefficient, especially when it encounters a narrow valley where almost all directions in the neighborhood are uphill directions. At the same time the gradient-based method is not preferable, as they would require the derivative information. As such we use a modified downhill simplex method to generate the random change of the system state.

This amounts to replacing the single point $\mathbf{w}$ as a description of the system state by a simplex of $N_w + 1$ points. The moves are the same as described in Chapter 5, namely reflections, expansions, and contractions of the simplex. The implementation of the Metropolis procedure is slightly subtle: We add a positive, logarithmically distributed random variable, proportional to the temperature T , to the stored function value associated with every vertex of the simplex, and we subtract a similar random variable from the function value of every new point that is tried as a replacement point. Like the ordinary Metropolis procedure, this method always accepts a true downhill step, but sometimes accepts an uphill one. When $T \rightarrow 0$, this algorithm reduces exactly to the downhill simplex method and converges to a local minimum.

The flow diagram is shown in Figure 6.1. The function value which has been imposed with a positive or negative random change is noted as E^+ or E^-

$$E^+(\mathbf{w}) = E(\mathbf{w}) + T * \log(rand), \quad E^-(\mathbf{w}) = E(\mathbf{w}) - T * \log(rand), \quad (6.1)$$

where T is the temperature and $rand$ is a uniformly distributed random number in $(0,1)$. There are several annealing schedules and the choice is by experiment. Specifically for neural network training, an exponential change is appropriate, i.e., reduce T to $a \cdot T$ ($a < 1$) after every $iter_t$ moves where $iter_t$ is the number of iterations for one temperature. A storage variable w_b records the best point ever encountered during the optimization process and this point is used to initialize the simplex at the beginning of every round for one temperature.

Figure 6.1: Flow diagram of the Simulated Annealing Algorithm.

6.3 Examples

Several examples are used to demonstrate the efficacy of this global training method. The first two are algebraic function with numerous local minima. The other two examples are wavelet neural networks for which all other local training methods failed to find the global optimum. The same set of examples will be used for the testing of the Genetic Algorithm in Chapter 7.

Example 1: Polynomial function with finite number of local minima

This is a polynomial function with finite number of local minima given by

$$F(\mathbf{x}) = \sum_{i=1}^N 0.1 * x_i^4 - 200 * (x_i - 5)^2 / 10^5 \quad (6.2)$$

where x_i is the element of the N-dimensional vector $\mathbf{x}$. For each dimension there will be two local minima at $x_i = -33.88, 28.74$. The two-dimensional illustration of the function is shown in Figure 6.2. We observe that there are 4 local minima the global optimum is at $(-33.88, -33.88)$. The SA is able to find the global optimum up to 5 dimensions.

Example 2: Nonlinear function with infinite number of local minima

Similar to example 1, the total function is the sum of the individual function, which is a quadratic function plus an AM modulated sinusoidal function. The function is given by

$$F(\mathbf{x}) = \sum_{i=1}^N x_i^2 - 100 * \cos(2 * x_i) - 100 * \cos(10 * x_i) \quad (6.3)$$

Due to the periodicity of sinusoidal function it would have infinite number of local minima. This can be noticed in the one-dimensional plot of the individual function in Figure 6.3. The two-dimensional illustration of the function is shown in Figure 6.4. The global optimum is $(0,0)$ where stands the minimum of both the quadratic function and the sinusoidal functions.

For this example the program takes more time to find the global optimum. This example demonstrates the power of global optimization method because for any local optimization method it would be impossible to jump out of so many deep valleys and find the global optimum.

Figure 6.2: A two-dimensional illustration of the polynomial function.

Figure 6.3: A one-dimensional plot of a quadratic function superimposed with an AM modulated sinusoidal function.

Figure 6.4: A two-dimensional illustration of the nonlinear function.

Example 3: A wavelet neural network with one hidden neuron

This example is a single input, single output wavelet neural network (see Section 2.2.4) with one hidden neuron. The wavelet function of the hidden neuron is an inverse Mexican Hat function given by

$$Z_j(\mathbf{x}) = \psi\left(\left\|\frac{\mathbf{x}_j - \mathbf{T}_j}{a_j}\right\|\right), \quad \psi(\gamma) = (\gamma^2 - N) \exp\left(-\frac{\gamma^2}{2}\right) \quad (6.4)$$

where $\mathbf{x}$ is the input vector and N is the dimensions of $\mathbf{x}$. $N=1$ in this example. The index j is dropped as there is only one hidden neuron. The training data is generated using the same wavelet function with translation parameter $T = 2$ and dilation parameter $a = 1$. The training result of the local training methods is shown in Figure 6.5. The model obtained is only trying to match the left wing of the wavelet. This situation corresponds to a local minimum and without exception all local methods obtain the same result. This would usually happen when the effective approximation region of the activation function, e.g., the central region of the wavelet, is highly nonlinear or non-smooth. As an interesting observation in popular neural network structure, such as MLP or RBF, the activation function is just a smooth switch function or Gaussian function.

The training curve and result of the SA are shown in Figure 6.6 and Figure 6.7 respectively. In Figure 6.6 the training error is the error of $\mathbf{w}_l$ which is the smallest among the perturbed function values. At higher temperature the amplitude of the variation of $\mathbf{w}_l$ among adjacent epochs is relative large corresponding to strong randomness of the training process. There is approximately a lower limit for the randomly changed $\mathbf{w}_l$ which is the best record $\mathbf{w}_b$. The $\mathbf{w}_b$ only changed several times in the training process as the lower limit moves down ladderlikely. And we can further observe that at higher

temperature the training can escape from the local minimum more easily. The training result in Figure 6.7 shows that the neural model matches the training data very well.

Example 4: A wavelet neural network with two hidden neurons

This example is also a wavelet network with the same wavelet function as in example 3. The only difference is that this model has two hidden neurons and therefore the neuron index $j = 1$ and 2 in (6.4). The training data is also generated through the sum of two wavelet functions (6.4) with translation parameters $T_1 = T_2 = 2$ and dilation parameters $a_1 = 2, a_2 = 0.5$ correspondingly. The function is given by

$$y = F(x) = 2 + \psi\left(\frac{x-2}{2}\right) - \psi\left(\frac{x-2}{0.5}\right) \quad (6.5)$$

The result from local training methods is shown in Figure 6.8. Similarly the local training methods try to match the wings on both sides using only one wavelet. We can imagine this should be a local minimum where one wavelet is utilized to approximate the model and the unused wavelet is far away. The training curve and result of the Simulated Annealing method are shown in Figure 6.9 and Figure 6.10. As observed in example 3, at higher temperature the best point w_b (the lower limit of the training error in the curve) could go down several times to escape from the local minimum.

Figure 6.5: Training result from local training methods for the wavelet network example with one hidden neuron.

Figure 6.6: Training curve of the Simulated Annealing Algorithm for the wavelet network example with one hidden neuron.

Figure 6.7: Training result from the Simulated Annealing Algorithm for the wavelet network example with one hidden neuron.

Figure 6.8: Training result from local training methods for the wavelet network example with two hidden neurons.

Figure 6.9: Training curve of the Simulated Annealing Algorithm for the wavelet network example with two hidden neurons.

Figure 6.10: Training result from the Simulated Annealing Algorithm for the wavelet network example with two hidden neurons.

Chapter 7

Genetic Algorithm

In this chapter the Genetic Algorithm is introduced in the training of neural networks. GA was first proposed by Holland [94] as an efficient search mechanism in artificially adaptive systems. This procedure was designed to incorporate specific operators such as natural selection, crossover and mutation, to simulate the process in the natural evolution. A description of the major components of the algorithm and the flow diagram is introduced first. Then the testing result of the examples is presented.

7.1 Description of Genetic Algorithm

Genetic Algorithm [95][96] is a stochastic optimization algorithm, derived from the concepts of biological evolutionary theory. The most influential theory on evolution was given by Charles Darwin. His ideas form the basis of most present-day evolutionary theories.

In Darwinian evolutionary theory, *natural selection* is seen as the creative force of evolution. When supplied with genetic variation it ensures that sexually reproducing species can adapt to changing environments. In the course of evolution, it preserves the

favorable part of the variation within a species. It often achieves this by letting the fitter individuals of a species produce more offsprings for the next generation. This provides a selective pressure that favors the fitter individuals of a population. The theory of natural selection is therefore often referred to as the survival of the fittest.

Genetic algorithms are based on a Darwinian-type survival of the fittest strategy, whereby potential solutions to a problem compete and mate with each other in order to produce increasingly stronger individuals. Each individual in the population represents a potential solution to the problem that is to be solved. The coding of a solution is described as a string of bits, similar to the way genetic information in organisms is coded onto chromosomes. In living cells, the information that determines their function is stored in a pair of chromosomes. A position on such a pair of chromosomes is called a *locus*. A locus holds two *genes* (one gene on each chromosome). The chromosome of a child is made up from genes from both parents and in this way the child would inherit some characteristics of their parents.

7.1.1 Major Components of GA

Genetic Algorithm encodes each point in a parameter (or solution) space into a binary bit string, and each point is associated with a "fitness" value, that is usually equal to the training objective function of the neural networks evaluated at the point. GA usually keeps a set of points as a population (or gene pool), which is then evolved repeatedly toward a better overall fitness value. In each generation, the GA constructs a new population using genetic operators such as crossover and mutation. Members with higher fitness values are more likely to survive and to participate in mating (crossover) operations. After a number of generations, the population contains members with better

fitness values. This is analogous to Darwinian models of evolution by random mutation and natural selection. Major components of GA [97] include encoding schemes, fitness evaluation, parent selection, crossover operators, and mutation operators. These will be explained next.

Encoding Schemes:

First we need to map the continuous variable space of NN parameters to the discrete bit-wise form of GA algorithm. Two encoding schemes are designed for this purpose.

In any digital computer the real number is represented by a sequence of binary bits. First strategy would directly use this binary representation in GA as the genes of one variable. For example floating number 43.1735, would be 0x422CB1A1 if we look at its internal binary representation. Concatenating all the variables in the NN structure would represent a chromosome (a solution of the problem). Since mostly float type is used to represent real number, the implementation uses the floating number format. However the first approach suffers the difficulty in convergence due to the difference between the exponential part and fractional part in a floating number. The mutation in the exponential part of a floating number would significantly change the value of the number.

Second approach is more reasonable. It will first fix the resolution (the number of bits) of a variable. For example suppose we want to change 5 decimal digits. Each floating number would be multiplied by 10^5 and rounded to integer number. The binary representation of this integer number would be selected and concatenated to form the chromosome of the variable. This scheme also covers a wide value range of w with the same storage space as in the first scheme. The integer number with 4 bytes has the range of $[-2147483648, 2147483648]$. Assuming the resolution is 5, the covered range would

be $[-21474.83648, 21474.83648]$. This range normally would be sufficient to accurately represent the trainable parameter ω . The minimal difference between two number will be 0.00001. For this scheme there would be an assumption that the effective values of ω should lie in the range $[-21474.83648, 21474.83648]$. Values outside this range would be truncated.

Fitness Evaluation:

The fitness of the each member in the population is a non-negative value representing its suitability. For our application we use the inverse of the training objective function $E(\omega)$. During the optimization processing every member is saved as real number. The encoding is only used when the genetic operation is required, e.g., mutation and crossover. The member that is selected as the parent or mutant would be converted to an intermediate binary format and after the mutation or crossover, the result would be converted back to real number and stored in the new population. In this way the fitness computation cost is just about the NN feed-forward evaluation and the variable conversions.

Parent Selection:

The selection operation determines which parents participate in producing offspring for the next generation, and it is analogous to survival of the fittest in natural selection. Usually members are selected for mating with a selection probability proportional to their fitness values. The most commonly used selection process is the roulette wheel selection [95]. We create a roulette wheel where each member is allocated a space proportional to its relative fitness values as shown in Figure 7.1. Therefore the probability that an individual is selected is equal to its relative fitness.

Crossover:

Crossover is usually applied to two parents selected through roulette wheel to create a pair of descendants. Crossover is usually conducted with a probability equal to a given crossover rate p_c (preset by the user and close to 1). With the probability $1 - p_c$ the new descendants only duplicate the parents. Various types of crossover are possible, of which the 1-point and 2-point types, depicted in Figure 7.2, are most frequently used. The crossover points are chosen randomly. The 2-point crossover is used in the implementation. The effect of crossover is similar to that of mating in the natural evolutionary process, in which parents pass segments of their own chromosomes to their children.

Mutation:

After generating the sufficient amount of population through the crossover or duplicate of the parental chromosomes in the old generation every bit of the resulting chromosomes is subject to possible mutation. Similarly this depends on a probabilistic chance p_m , the mutation rate, which is very close to zero. The mutation inverts the bit value of the binary genes.

7.1.2 Flow Diagram of Genetic Algorithm

The top-level flow diagram of the Genetic Algorithm is shown in Figure 7.3. An initial population is created by randomly selecting some points around the starting point. During the evolution process the fitness values of the members in the population are improved. In every generation all new members are generated through the genetic mechanism, i.e., select parents, crossover or copy the genes from parents, mutate the genes. Each member represents one w and the best member represents the point where

the training error is the lowest. Therefore through the parent selection the best point would have a much higher chance to be selected to pass their information to the new members.

Figure 7.1: Roulette wheel chart.

(a)

(b)

Figure 7.2: Crossover operators: (a) one-point crossover, (b) two-point crossover.

Figure 7.3: Flow diagram of the Genetic Algorithm.

7.2 Examples

For illustrative purpose the same four examples of last chapter are also used here. Theoretically the convergence speed of genetic algorithms is relatively much slower than other conventional training methods. But the concern of here is to find the global optimum or escape from the local optimum. In practice due to the high dimensions of the variables in the neural network, it is very difficult for the global training methods to converge to a very good accuracy even when it is very close to the global optimum. So these methods are considered either at the early stage of training or when the local methods stay at obviously high training error and could not go down.

The algebraic function examples are successfully solved by genetic algorithm. The second example demonstrates its robustness against the multiple minima. For the wavelet examples, the similar conclusion can be made about its capability of locating the global optimum. Figures 7.4, 7.5 and Figures 7.6, 7.7 are the training results and training curves of the genetic algorithm for the two wavelet examples. The training curve plots the best objective function values in the population throughout the generations.

Figure 7.4: Training result from the Genetic Algorithm for the wavelet network example with one hidden neuron.

Figure 7.5: Training result from the Genetic Algorithm for the wavelet network example with two hidden neurons.

Figure 7.6: Training curve of the Genetic Algorithm for the wavelet network example with one hidden neuron.

Figure 7.7: Training curve of the Genetic Algorithm for the wavelet network example with two hidden neuron.

Chapter 8

Summary and Future Work

This chapter summarizes the work of this thesis and discusses the direction of the future work. Some challenges of the future work are also viewed.

8.1 Comparison of Different Training Algorithms

There are some tradeoffs among the efficiency, memory requirement, and the ability of finding a global optimum. The global training algorithms GA and SA are very unefficient and require a lot of memory, but they have the ability of finding the global optimum. The second-order training methods are more efficient than other training methods but require the derivative information and may be trapped in a local minimum. Among them, LM and quasi Newton method would require more memory than Conjugate Gradient method, but have fast convergence speed. Since the LM method involves the LU decomposition, it will become slow when the problem size increases. Quasi Newton is suitable for most of the problems. In case the problem dimensions become very large Conjugate Gradient method and BP would be more appropriate as they require less memory and computation effort per epoch. The simplex method does not require derivatives and has similar performance as BP, but requires more memory.

8.2 Conclusions

This thesis was motivated by the rapidly increased application of the neural modeling technique in the microwave modeling, simulation and optimization. This work successfully applies the general optimization techniques in the neural network training. The specific contributions are as follows.

1. In microwave applications the training data is obtained through either simulation or measurement. Large (catastrophic) errors are often introduced in the process of data collection for various reasons, such as measurement performed at the limit of equipment frequency range, or data simulated at extreme parameter values leading to non-convergence or large roundoff /truncation errors. The conventional least-squares (l_2) training algorithms are susceptible to the influence of gross errors and the model can be significantly led astray at the neighborhood of the wrong data points. To address this problem Huber concept has been introduced in the training of neural networks. A new microwave-oriented training algorithm, which combines the Huber error formulation with the second-order optimization algorithm, has been developed in this thesis. The robustness and consistency of the proposed method against the gross errors in the training data have been demonstrated through examples by comparison with the conventional l_2 -based methods.

2. The training of the feed-forward neural networks can be formulated as an unconstrained nonlinear optimization problem and therefore various optimization algorithms can be applied. In this thesis through the optimization approaches a variety of training methods are implemented, including quasi Newton method, Huber quasi Newton method, Simplex method, Simulated Annealing algorithm and Genetic algorithm. The

advantages of these training methods are demonstrated with microwave examples. Instead of developing a special training method for a special neural network all training methods developed here can be applied to any feed-forward neural network structure through a general formulation. In this way various training methods and neural network structures can be independently developed. All these training algorithms are successfully incorporated into our research software NeuroModeler.

3. To apply the optimization algorithms in the training of neural networks for RF/Microwave applications, necessary modifications and adjustment have been made. In quasi Newton method two ways of preserving the positive definiteness property of the approximation matrix are used. In the Huber quasi Newton method, the single error in the Huber function is defined as the overall error of one sample rather than the error of each element of each sample. In this way the whole wrong data sample will be ignored in the training process because the sample would not be reliable anymore if large error has been detected in any of its output. The simplex method is used to generate random movement of SA. Two conversion schemes are exerted to generate the chromosomes for GA.

To facilitate a generic formulation of the feedforward neural networks a sorting program is developed, including the sorting of the neurons' sequence, loop check and the neural network structure identification.

8.3 Future Work

The work presented in this thesis has provided a very attractive prospect for the application of the emerging neural network technology in the modeling problem. Further mathematical investigation of the optimization techniques for the training process would be a good direction. In this approach some specific challenges associated with neural

network modeling problem need to be addressed: the high nonlinearity and the curse of high dimensions. High nonlinearity happens when the behavior of the model is become highly nonlinear. The curse of high dimensions exists because the size of training data increases exponentially as the number of model inputs and outputs increases. Some work has been done in [86] to address the high-nonlinearity challenge. Following the idea "divide and conquer" an iterative multi-stage training method is proposed to decompose the complex problem into sub task and solve it iteratively.

All the training methods developed here are only designed for solving the deterministic function approximation problem. And correspondingly the feed-forward neural network structure is used to learn the deterministic behavior of the model. However there are a lot of real-world problems with stochastic relationship where the recurrent network or time-varying network are appropriate. These problems are also very important in modeling domain and would be very profitable if a reliable model could be developed. Therefore developing training algorithms for this purpose would be another good direction.

Different training algorithms are appropriate for different training situation at different training stage. Therefore an intelligent scheme which can automatically switch between different algorithms would be necessary. However simple it may sound, this decision making scheme would not be easy to implement and could be another interesting topic in the neural network research.

References

- [1] F. Wang and Q.J. Zhang, "Knowledge based neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2333-2343, 1997.
- [2] A.H. Zaabab, Q.J. Zhang, and M.S. Nakhla, "Neural network modeling approach to circuit optimization and statistical design," *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1349-1358, 1995.
- [3] 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.
- [4] P.M. Watson and K.C. Gupta, "Design and optimization of CPW circuits using EM-ANN models for CPW components," *IEEE Trans. Microwave Theory and Tech.*, vol. 45, pp. 2515-2523, 1997.
- [5] G.L. Creech, B.J. Paul, C.D. Lesniak, T.J. Jenkins, and M.C. Calcaterra, "Artificial neural networks for accurate microwave CAD applications," *IEEE Intl. Microwave Symp. Digest*, pp. 733-736, San Francisco, 1996.
- [6] M. Vai, S. Wu, B. Li, and S. Prasad, "Creating neural network based microwave circuit models for analysis and synthesis," *Proc. Asia Pacific Microwave Conf.*, pp. 853-856, Hong Kong, Dec. 1997.

- [7] M. Vai and S. Prasad, "Microwave circuit analysis and design by a massively distributed computing network," *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1087-1094, May 1995.
- [8] Q. J. Zhang and M.S. Nakhla, "Signal integrity analysis and optimization of VLSI interconnects using neural network models," *IEEE Intl. Symp. Circuits Systems*, pp. 459-462, London, England, 1994.
- [9] P.M. Watson, K.C. Gupta, and R.L. Mahajan, "Development of knowledge based artificial neural network models for microwave components," *IEEE Intl. Microwave Symp. Digest*, pp. 9-12, Baltimore, MD, 1998.
- [10] 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.
- [11] V.B. Litovski, J.I. Radjenovic, Z.M. Mrcarica, and S.L. Milenkovic, "MOS transistor modeling using neural network," *Electronics Letters*, vol. 28, pp. 1766-1768, 1992.
- [12] A.H. Zaabab, Q.J. Zhang, and M.S. Nakhla, "Analysis and optimization of microwave circuits and devices using neural network models," *IEEE Intl. Microwave Symp. Digest*, pp. 393-396, San Diego, CA, May 1994.
- [13] G. Kothapalli, "Artificial neural networks as aids in circuit design," *Microelectronics Journal*, vol. 26, pp. 569-678, 1995.
- [14] Y. Harkouss, J. Rousset, H. Chehade, D. Barataud, and J.P. Teyssier, "Modeling microwave devices and circuits for telecommunications system

- design," *Proc. IEEE Intl. Conf. Neural Networks*, pp. 128-133, Anchorage, Alaska, May 1998.
- [15] 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.
- [16] Q.J. Zhang, F.Wang, and M.S. Nakhla, "Optimization of high-speed VLSI interconnects: A review," *Int. J. of Microwave and Millimeter-Wave CAD*, vol. 7, pp. 83-107, 1997.
- [17] T. Horng, C. Wang, and N.G. Alexopoulos, "Microstrip circuit design using neural networks," *IEEE Int. Microwave Symp. Digest*, pp. 413-416, Atlanta, Georgia, 1993.
- [18] P. Burrascano, M. Dionigi, C. Fancelli, and M. Mongiardo, "A neural network model for CAD and optimization of microwave filters," *IEEE Intl. Microwave Symp. Digest*, pp. 13-16, Baltimore, MD, 1998.
- [19] M.D. Baker, C.D. Himmel, and G.S. May, "In-Situ prediction of reactive ion etch endpoint using neural networks," *IEEE Trans. Components, Packaging, and Manufacturing Tech., part A*, vol. 18, pp.478-483, Sept. 1995.
- [20] M.Vai and S.Prasad, "Automatic impedance matching with a neural network," *IEEE Microwave Guided Wave Letter*, vol. 3, pp. 353-354, 1993.
- [21] R. Biernacki, J.W. Bandler, J. Song, and Q.J. Zhang, "Efficient quadratic approximation for statistical design," *IEEE Trans. Circuits Syst.*, vol. CAS-36, pp. 1449-1454, 1989.

- [22] P. Meijer, "Fast and smooth highly nonlinear multidimensional table models for device modelling," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 335-346, 1990.
- [23] F. Wang and Q. J. Zhang, "Incorporating functional knowledge into neural networks," In *Proc. Intl. Conf. Neural Networks*, pp. 266-269, Houston, TX, June 1997.
- [24] B. Widrow and M.A. Lehr, "30 years of adaptive neural networks: Perceptron, Madaline, and Backpropagation," *IEEE Proc.*, vol. 78, pp. 1415-1442, 1990.
- [25] M. Minsky and S. Papert, *Perceptrons*, Cambridge: MIT Press, 1969.
- [26] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Systems*, vol. 2, pp. 303-314, 1989.
- [27] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.
- [28] 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.
- [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] F. Girosi, "Regularization theory, radial basis functions and networks," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, V. Cherkassy, J.H. Firedman, and H. Wechsler, Eds. Berlin, NY: Springer-Verlag, 1992, pp. 166-187.

- [31] M.J.D. Powell, "Radial basis functions for multivariate interpolation: a review," in *Algorithms for Approximation*, J.C. Mason and M.G. Cox, Eds. Oxford, UK: Oxford University Press, 1987, pp. 143-167.
- [32] J. Park and I.W. Sandberg, "Universal approximation using Radial-Basis-Function networks," *Neural Computation*, vol. 3, pp. 246-257, 1991.
- [33] J. Park and I.W. Sandberg, "Approximation and Radial-Basis-Function networks," *Neural Computation*, vol. 5, pp. 305-316, 1991.
- [34] 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.
- [35] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks," *IEEE Trans. Neural Networks*, vol. 8, pp. 1177-1183, 1997.
- [36] S. Haykin, *Neural Networks: A Comprehensive Foundation*, New York, NY: IEEE Press, 1994.
- [37] J. Moody and C.J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281-294, 1989.
- [38] 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.
- [39] R. Reed, "Pruning algorithms - a survey," *IEEE Trans. Neural Networks*, vol. 4, pp. 740-747, Sept. 1993.

- [40] A. Krzyzak and T. Linder, "Radial basis function networks and complexity regularization in function learning," *IEEE Trans. Neural Networks*, vol. 9, pp. 247-256, 1998.
- [41] H. Leung and S. Haykin, "Rational function neural network," *Neural Computation*, vol. 5, pp. 928-938, 1993.
- [42] J. R. Tsai, P.C. Chung, and C.I. Chang, "A sigmoidal radial basis function neural network for function approximation," *Proc. IEEE Intl. Conf. Neural Networks*, pp. 496-501, Washington, DC, June 1996.
- [43] P.H. Ladbrooke, *MMIC Design: GaAs FET's and HEMTs*, Artech House, Boston, 1989.
- [44] 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.
- [45] Q.H. Zhang, "Using wavelet network in nonparametric estimation," *IEEE Trans. Neural Networks*, vol. 8, pp. 227-236, Mar. 1997.
- [46] Q.H. Zhang and A. Benveniste, "Wavelet networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 889-898, Nov. 1992.
- [47] B.R. Bakshi and G. Stephanopoulos, "Wave-Net: a multiresolution, hierarchical neural network with localized learning," *American Institute of Chemical Engineering Journal*, vol. 39, pp. 57-81, 1993.
- [48] B. Delyon, A. Juditsky, and A. Benveniste, "Accuracy analysis for wavelet approximations," *IEEE Trans. Neural Networks*, vol. 6, pp. 332-348, Nov. 1995.

- [49] K. Thulasiraman, M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley and Sons, Inc, 1992.
- [50] 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
- [51] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," In *Parallel Distributed Processing*, vol. I, D.E. Rumelhart and J.L. McClelland Eds. Cambridge, Massachusetts: MIT Press, 1986, pp. 318-362.
- [52] D.G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, Reading, Massachusetts, 1989.
- [53] 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.
- [54] S.J. Perantonis and D.A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, vol. 8, pp. 237-249, 1995.
- [55] *Neural Network Toolbox: For Use with Matlab*, The MathWorks Inc., Natick, Massachusetts, 1993.
- [56] R.A. Jacobs, "Increased rate of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.
- [57] R .Battiti, "Accelerated backpropagation learning: two optimization methods," *Complex Systems*, vol. 3, pp. 331-342, 1989.
- [58] 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.

- [59] 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.
- [60] 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.
- [61] P.P. Van-Der-Smagt, "Minimisation methods for training feedforward neural networks," *Neural Networks*, vol. 7, pp. 1-11, 1994.
- [62] 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.
- [63] D.R. Hush and J.M. Salas, "Improving the learning rate of back propagation with the gradient reuse algorithm," In *Proc. IEEE Int. Conf. Neural Networks*, vol. I, pp. 441-447, San Diego, California, 1988.
- [64] 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.
- [65] F. Wang, V. K. Devabhaktuni, C.G. Xi, and Q. J. Zhang, "Neural network structures and training algorithms for RF and Microwave applications," *Intl. Journal of RF and Microwave Computer-Aided Engineering: Special Issue on Applications of Artificial Neural Networks to RF and Microwave Design*, vol. 9, pp. 216-240, 1999.
- [66] R. Battiti, "First and second-order methods for learning: between steepest descent and Newton's method," *Neural Computation*, vol. 4, pp. 141-166, 1992.

- [67] M.F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol 6, pp. 525-533, 1993.
- [68] A.J. Shepherd, "Second-order optimization methods," In *Second-Order Methods for Neural Networks*, Berlin, NY: Springer-Verlag, 1997, pp. 43-72.
- [69] 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.
- [70] 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.
- [71] 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.
- [72] V. K. Devabhaktuni, C. G. Xi, and Q. J. Zhang, "A neural network approach to the modeling of heterojunction bipolar transistors from S-parameter data", *Proc. European Microwave Conf.*, vol. 1, pp. 306-311, Amsterdam, Netherlands, 1998.
- [73] C-J Wei and James C. M. Hwang, "Direct extraction of equivalent circuit parameters for heterojunction bipolar transistors", *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 2035-2039, 1995.
- [74] J. W. Bandler, W. Kellermann, and K. Madsen, "A nonlinear l_1 optimization algorithm for design, modeling and diagnosis of networks", *IEEE Trans. Circuit Syst.*, vol. CAS-34, pp. 174-181, 1987.
- [75] P. Huber, *Robust Statistics*, New York: Wiley, 1981.

- [76] G. Li and K. Madsen, "Robust nonlinear data fitting", *Numerical Analysis 1987*, D.F. Griffiths and G.A. Watson, Eds., Pitman Research Notes in Mathematics Series 170, pp.176-191, 1988.
- [77] J. W. Bandler, S.H. Chen, R.M. Biernacki, L.Gao, K.Madsen, and H. Yu, "Huber optimization of circuits: a robust approach", *IEEE Trans. Microwave Theory Tech.*, vol. 41, pp. 2279-2287, 1993.
- [78] J. W. Bandler, S. H. Chen, R. M. Biernacki, and K. Madsen, "The Huber concept in device modeling, circuit diagnosis and design centering", in *IEEE Int. Symp. Circuit Systems*, London England, 1994, pp. 129-132.
- [79] C. L. Philip Chen, "A rapid supervised learning neural network for function interpolation and approximation", *IEEE Trans. Neural Networks*, vol. 7, pp. 1220-1230, Sep. 1996.
- [80] C. G. Xi, F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "Huber optimization of neural networks: A robust training method", accepted by the *IEEE Intl. Conf. Neural Networks*, Washington DC, USA, July 1999.
- [81] T. R. Cuthbert Jr., "Quasi-Newton Methods and Constraints", In *Optimization using Personal Computers*, New York, NY: John Wiley & Sons, 1987, pp.233-314
- [82] S. Amari, N. Murata, K-R Muller, M. Finke, and H. H. Yang, "Asymptotic statistical theory of overtraining and cross-validation", *IEEE Trans. Neural Networks*, vol. 8, pp. 985-996, Sept. 1997.

- [83] M. A. Khatibzadeh and R. J. Trew, "A large-signal, analytical model for the GaAs MESFET", *IEEE Trans. Microwave Theory Tech.*, vol. 36, pp. 231-239, 1998.
- [84] *OSA90 Version 3.0*, Optimization Systems Associations Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, now HP Eesof, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403.
- [85] A. Djordjevic, R. F. Harrington, T. Sarkar and M. Bazdar, *Matrix Parameters for Multiconductor Transmission Lines: Software and User's Manual*, Artech House, Boston, 1989
- [86] V. K. Devabhaktuni, C. G. Xi, F. Wang and Q. J. Zhang, "Robust training of microwave neural models", Accepted for full-length presentation at the IEEE MTT-S International Symposium, Anaheim, CA, USA, June 1999.
- [87] *Serenade*, Ansoft Corporation, 669 River Drive, Suite #200, Elmwood Park, NJ 07407-1361.
- [88] J. A. Nelder and R. Mead. "A simplex method for function minimization", *Computer Journal*, Vol. 7, pp. 308-313, 1965.
- [89] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.
- [90] J. C. F. Pujol and R. Poli, "Evolving neural networks using a dual representation with a combined crossover operator," *Proc. IEEE Intl. Conf. Evol. Comp.*, pp. 416-421, Anchorage, Alaska, May 1998.
- [91] M.P. Vecchi, and S. Kirkpatrick, "Global wiring by Simulated Annealing", *IEEE Trans. Computer Aided Design*, vol. CAD-2, pp. 214-222, 1983.

- [92] L. Ingber and B.E. Rosen, "Genetic algorithms and very fast simulated annealing", *Mathematical and Computer Modeling*, vol. 16, pp. 87-100, 1992.
- [93] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equation of state calculations by fast computing machines", *Journal of Chemical Physics*, vol. 21, pp.1087-1092, 1953.
- [94] J. H. Holland, *Adaptation in natural and artificial systems*, the University of Michigan Press, Ann Arbor, Michigan, 1975.
- [95] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, Massachusetts, 1989.
- [96] D. B. Fogel, *Evolutionary Computation*, Institute of Electrical and Electronics Engineers, Inc., New York, 1995.
- [97] A. J. F. van Rooij, L. C. Jain, R. P. Johnson, *Neural Network Training Using Genetic Algorithms*, World Scientific, 1996.