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. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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.

UMI[®]

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

Knowledge Based Neural Networks for Microwave Modeling and Design

by

Fang Wang, 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
Doctor of Philosophy

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

©Copyright November 1998, Fang Wang

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.

0-612-37081-X

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

Knowledge Based Neural Networks for Microwave Modeling and Design

submitted by Fang Wang, B. Eng., M. Eng.,
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

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. Wight', written over a horizontal line.

J. Wight, Chair, Department of Electronics

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

External Examiner (Ph.D.)

Abstract

Neural networks have recently been introduced to the microwave area as a fast and flexible vehicle for microwave modeling, simulation and optimization. This thesis represents a new research direction in this area, i.e., incorporating microwave knowledge into neural network models. Two approaches utilizing functional and structural microwave knowledge are proposed in the thesis. They are used for efficient modeling of active and passive microwave components and for the development of libraries of microwave neural models.

In the first approach, a novel neural network structure, namely functional knowledge based neural network (KBNN), is proposed where microwave empirical or semi-analytical information is incorporated into the internal structure of neural networks. The microwave knowledge complements the capability of learning and generalization of neural networks by providing additional information which may not be adequately represented in a limited set of training data. Such knowledge becomes even more valuable when the neural model is used to extrapolate beyond the training-data region. A new training scheme employing the gradient-based l_2 optimization technique is developed to train the KBNN model. The proposed technique can be used to model passive and active microwave components with improved accuracy, reduced cost of model development and less need of training data over conventional neural models for microwave design.

The second approach addresses the development of libraries of neural models for passive and active components, a task with a potential significance for many microwave simulators. Developing libraries of neural models is very costly due to massive data generation and repeated neural network training. A new hierarchical neural

network approach is proposed, allowing both microwave functional knowledge and library inherent structural knowledge to be incorporated into neural models. The library models are developed through a set of base neural models, which capture the basic characteristics common to the entire library, and high-level neural modules which map the information from base models to the library model outputs. The proposed method substantially reduces the cost of library development through reduced need for data collection and shortened time of training. The technique is demonstrated through transmission line and FET library examples.

To my grandmother

Acknowledgements

At the outset, I would like to express my sincere thanks to my thesis supervisor Professor Qi-jun Zhang for his encouragement, continued assistance, expert guidance and motivation throughout the course of this work. I am highly indebted to him for having trained me into a full-time researcher with technical, computational and presentation skills, patience and professionalism. My grateful thanks also go to Professor Michel Nakhla of Carleton University and Professor Tet-hin Yeap of the University of Ottawa for their valuable advice and comments on the thesis.

My thanks are due to Dr. Greg Wilson of Nortel for his inputs on issues related to transmission line modeling problem. It is my pleasure to acknowledge inspiring discussions with my colleagues Vijaya Kumar Devabhaktuni and Changgeng Xi. Homayun Feyzbakhsh is thanked for training some of the MLP models. The friendliness of fellow graduate students in the department has cheered my life as a Ph.D. student. I also wish to extend my thanks to the departmental staff - Betty Zahalan, Peggy Piccolo, Lorena Duncan, Walter Fergusson, Nagui Mikhail and Jacques Lemieux, for providing an excellent supporting environment.

My close friend Xin Xu, and her mother are greatly thanked for sharing traditional Chinese food and giving me a good company on those lonely nights when I return home exhausted after a day's work on my thesis.

This thesis would not have been possible without years of support and encouragement from my parents and my sister. Father's guidance, mother's love and sister's admiration have been the precious treasures that I enjoyed through all the years of my study. This thesis is dedicated to my grandmother, who brought me up and was always proud of me but is no more here to feel happy about my achievements. Last

but not the least, I would like to thank Daguang for his support, tolerance and love first as a friend and then as a husband.

The financial assistance provided by the Department of Electronics through a Teaching Assistantship, the Ministry of Education through an Ontario Graduate Scholarship, and the Natural Sciences and Engineering Research Council of Canada through a Postgraduate Scholarship, is gratefully acknowledged.

Contents

1	Introduction	1
1.1	Motivations	1
1.2	Contributions	4
1.3	Outline of the Thesis	6
2	Literature Review	8
2.1	Neural Network Applications in Microwave Design	8
2.2	Neural Based Microwave Modeling: Problem Statement	10
2.3	Neural Network Structures	12
2.3.1	Standard Feedforward Neural Networks	12
2.3.2	Neural Network Structures with Prior Knowledge	17
2.3.3	Combining Neural Networks	20
2.3.4	Other Neural Network Structures	21
2.4	Neural Network Training Algorithms	24
2.4.1	Training Objective	24
2.4.2	Backpropagation Algorithm and Its Variants	25
2.4.3	Training Algorithms Using Gradient-based Optimization Tech- niques	28
2.4.4	Training Algorithms Utilizing Decomposed Optimization	34

2.4.5	Global Training Algorithms	36
2.5	Examples: Feedforward Neural Networks and Their Training	37
2.6	Application Example: Signal Integrity Optimization of High-speed VLSI Interconnects	40
2.6.1	Macromodeling Techniques for Interconnect Optimization . . .	42
2.6.2	Neural Network Models	44
2.6.3	Yield Optimization of a 3-Transmission Line Network Using Neural Models	49
2.7	Conclusions	50
3	Proposed Knowledge Based Neural Networks	52
3.1	Introduction	52
3.2	Proposed Knowledge Based Neural Network Structure	53
3.3	Discussions	56
3.4	Training Approach of Knowledge Based Neural Networks	58
3.5	An Illustrative Example	61
4	Application Examples of Knowledge Based Neural Networks	64
4.1	Circuit Waveform Modeling	64
4.2	Transmission Line Modeling	72
4.3	MESFET Modeling	83
5	Knowledge Based Hierarchical Framework for the Development of Library of Neural Models	95
5.1	Introduction	95
5.2	Problem Statement: Library Development	96
5.3	Base Models	97

5.4	Hierarchical Neural Network Structure	100
5.5	Algorithm for Overall Library Development	104
5.6	Discussions	106
6	Examples of Neural Model Library Development	109
6.1	Library of Stripline Models	109
6.2	Library of Microstrip Models	117
6.3	Library of MESFET Models	129
7	Conclusions and Future Work	136
7.1	Conclusions	136
7.2	Suggestions for Future Directions	138
Appendix A		
A General Neural Network Structure Description and POWERNET Software Package		141
A.1	General Neural Network Structure Description	142
A.2	POWERNET Software Package	144
A.2.1	The Components of POWERNET Software Package	144
A.2.2	The Usage of POWERNET Software Package	149
Bibliography		156

List of Figures

2.1	Multilayer Perceptron (MLP) structure. Typically, the network consists of an input layer, one or more hidden layers and an output layer.	14
2.2	Three conductor microstrip line.	38
2.3	Neural network model at the device level, an example of MESFET model after [3].	46
2.4	A 3-transmission line network with coupled transmission lines.	47
2.5	An example of neural network model at the circuit level for the circuit of Figure 2.4.	48
2.6	Histogram of the signal delay (in seconds) for V_{out2} of the 3-transmission line network obtained from exact circuit simulation of 500 random samples. Samples with delay value more than 1.72ns violate specifications. (a) before optimization (b) after optimization using neural models. . .	51
3.1	The proposed KBNN structure.	54
3.2	The decay sinusoidal example: (a) y versus x curve from MLP, (b) y versus x curve from KBNN.	63
4.1	The circuit for waveform modeling example with 3 transmission lines representing high-speed VLSI interconnects.	65

4.2	Model accuracy comparison with original HSPICE simulation for circuit modeling example when $r = 115\ ohm$: (a) MLP and (c) KBNN at node 3; (b) MLP and (d) KBNN at node 4. All models were trained with data from only 10 points per waveform.	68
4.3	The internal activities of KBNN: responses of individual neurons for each pair of resistor value and time points. This is the illustrative KBNN structure which only models one output waveform.	69
4.4	The internal activities of KBNN: responses of individual neurons for each pair of resistor value and time points. This is the actual KBNN structure used in this waveform modeling example, which models multiple output waveforms.	70
4.5	Model accuracy comparison of KBNN and MLP in terms of average testing error for the transmission line example. (a) Testing data sampled within the same range as training data (b) Testing data sampled around/beyond the boundary of training data. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).	77

4.6	Model accuracy comparison of KBNN and MLP in terms of worst case testing error for the transmission line example. (a) Testing data sampled within the same range as training data (b) Testing data sampled around/beyond the boundary of training data. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).	78
4.7	Scattering plot of mutual inductance L_{12} (a) from MLP and (b) from KBNN for the transmission line modeling example for 500 testing samples. Both models were trained with insufficient training data of only 100 samples.	79
4.8	Scattering plot of mutual inductance L_{12} (a) from MLP and (b) from KBNN for the transmission line modeling example for 500 testing samples. Both models were trained with training data of 300 samples. . .	80
4.9	Histograms of testing error of (a) MLP and (b) KBNN for the transmission line modeling example for 4069 testing samples around/beyond training data boundary. Both models were trained by only 100 training samples. Since concentration of errors is closer to 0% for KBNN than that of MLP, KBNN shows better accuracy than MLP.	81
4.10	Histograms of testing error of (a) MLP and (b) KBNN for the transmission line modeling example for 4069 testing samples around/beyond training data boundary. Both models were trained by 300 training samples.	82

4.11	Model accuracy comparison of KBNN and MLP in terms of average testing error for the MESFET example. (a) Test data set A (b) Test data set B. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).	89
4.12	Model accuracy comparison of KBNN and MLP in terms of worst case testing error for the MESFET example. (a) Test data set A (b) Test data set B. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).	90
4.13	An example of IV curves from (a) MLP and (b) KBNN for MESFET modeling example. Both models were trained with insufficient training data of only 100 samples. The 100 samples were generated by changing 6 FET parameters including gate width, length, channel thickness, doping density, V_G and V_D . KBNN is visibly better than standard MLP.	91
4.14	An example of IV curves from (a) MLP and (b) KBNN for MESFET modeling example. Both models were trained with reasonable size of training data of 300 samples.	92

4.15	Histograms of testing error of (a) MLP and (b) KBNN for MESFET modeling example with test data set B, i.e., extrapolation error. Both models were trained by only 100 training samples. Since concentration of errors is closer to 0% for KBNN than that of MLP, KBNN shows better accuracy than MLP.	93
4.16	Histograms of testing error of (a) MLP and (b) KBNN for MESFET modeling example with test data set B, i.e., extrapolation error. Both models were trained by 300 training samples.	94
5.1	A library of stripline models. The n th model in the library represents an n -conductor coupled stripline component.	98
5.2	The proposed hierarchical neural network structure. $\mathbf{X}$ and $\mathbf{Y}$ represent the inputs and outputs of the overall network. $\mathbf{L}_i$ is the i th low level module with an associated i th knowledge hub $\mathbf{U}_i(\cdot)$. $\mathbf{u}$ and $\mathbf{z}$ represent the inputs and outputs of low level modules. This structure can be used for each model in a library. For example, for the n th model, $\mathbf{Y} = \mathbf{Y}^n$, $\mathbf{H} = \mathbf{H}^n$, $\mathbf{z}_i = \mathbf{z}_i^n$, $\mathbf{L}_i = \mathbf{L}_i^n$, $\mathbf{u}_i = \mathbf{u}_i^n$, $\mathbf{U}_i = \mathbf{U}_i^n$ and $\mathbf{X} = \mathbf{X}^n$	101
5.3	A sequence of space mapping in the proposed hierarchical neural network structure. For example, for the n th model, $\mathbf{Y} = \mathbf{Y}^n$, $\mathbf{H} = \mathbf{H}^n$, $\mathbf{Z} = \mathbf{Z}^n$, $\mathbf{L}_i = \mathbf{L}_i^n$, $\mathbf{U}_i = \mathbf{U}_i^n$ and $\mathbf{X} = \mathbf{X}^n$	103
5.4	The flow chart of the algorithm for overall library development. . . .	107
6.1	Details of a typical N -conductor stripline component showing the physical and geometrical parameters.	110
6.2	The hierarchical neural model for the 3rd model in the stripline library, i.e., $n = 3$	114

6.3	Model accuracy comparison (average error on test data) between standard MLP and the proposed model for 3-Conductor stripline model. .	116
6.4	The microstrip library. The n th model in the library represents an n -conductor coupled microstrip model.	118
6.5	Details of a typical N -conductor microstrip component showing the physical and geometrical parameters.	119
6.6	The hierarchical neural model for the 3rd model in the microstrip library, i.e., $n = 3$	123
6.7	Model accuracy comparison (average error on test data) between standard MLP and the proposed model for 3-Conductor microstrip model.	125
6.8	The total amount of training data required for developing neural model library of microstrip lines versus the total number of models in the library. The overhead data of 400 required for the proposed approach due to base model training is represented by the nonzero value when $N_C = 0$. But the incremental amount of data needed for training each new model in the library is very small under the proposed approach. As the total number of models in the library increases, the total amount of training data required by the proposed approach becomes substantially less than that required by the standard MLP approach.	127

6.9	The total training time for developing neural model library of microstrip lines versus the total number of models in the library. The overhead training time of 14 minutes for the proposed approach due to base model training is represented by the nonzero value at $N_C = 0$. But the incremental training time for adding each new model to the library is very small under the proposed approach. As the total number of models in the library increases, the total training time required by the proposed approach becomes substantially less than that of the standard MLP approach.	128
6.10	Physics-based intrinsic MESFET device model following [151].	129
6.11	The hierarchical neural model for FET library model #5, i.e., $n = 5$	132
6.12	Model accuracy comparison (average error on test data) between standard MLP and the proposed model for the MESFET library model, $n = 5$, whose gate length equals $0.55 \mu m$	134

List of Tables

2.1	Model Accuracy Comparison Between MLP and RBF.	37
2.2	Comparison of Various Training Algorithms for Microstrip Line Example.	39
2.3	Comparison of Various Training Algorithms for MESFET Example. .	40
4.1	Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for Circuit Waveform Modeling Example. The Results Shown Are the Average of Three Different Trainings for Each Model.	66
4.2	Ranges of Training Data for Neural Model Input Parameters for the Transmission Line Modeling Example.	74
4.3	Model Accuracy Comparison Between MLP and KBNN for Transmission Line Modeling Example with Testing Data in the Same Region as Training Data. The Results Shown Are the Average of Three Different Trainings for Each Model.	75
4.4	Model Accuracy Comparison Between MLP and KBNN for Transmission Line Modeling Example with Testing Data around/beyond Training Data Boundary. The Results Shown Are the Average of Three Different Trainings for Each Model.	76

4.5	Training Data Ranges of Neural Model Input Parameters in MESFET Modeling Example. This is also the Parameter Range of Test Data Set A.	84
4.6	Ranges of Neural Model Input Parameters for Test Data Set B in MESFET Modeling Example.	84
4.7	Ranges of Neural Model Input Parameters for Test Data Set C in MESFET Modeling Example.	84
4.8	Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set A. The Results Shown Are the Average of Three Different Trainings for Each Model.	86
4.9	Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set B. The Results Shown Are the Average of Three Different Trainings for Each Model.	87
4.10	Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set C. The Results Shown Are the Average of Three Different Trainings for Each Model.	88
6.1	The Notations for Input and Output Parameters of Stripline Neural Models and the Effective Range of Their Input Parameters.	111
6.2	Stripline Library Models.	112
6.3	Base Models for Stripline Library.	113

6.4	Low Level Modules and Structural Knowledge Hubs for 3-Conductor Stripline, i.e., Library Model $n = 3$	113
6.5	Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for 3-Conductor Stripline Model.	115
6.6	Comparison of Number of Training Samples Needed and Library Model Accuracy for Stripline Library When Developed by Standard MLP and the Proposed Hierarchical Neural Network Structure, respectively. .	120
6.7	Microstrip Library Models.	121
6.8	Base Models for Microstrip Library.	122
6.9	Low Level Modules and Structural Knowledge Hubs for 3-Conductor Microstrip, i.e., Library Model $n = 3$	122
6.10	Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for 3-Conductor Microstrip Model.	124
6.11	Comparison of Number of Training Samples Needed and Training Time Used for Microstrip Library When Developed by Standard MLP and the Proposed Neural Network Structure, respectively.	126
6.12	Effective Ranges of Neural Model Input Parameters for MESFET Library.	130
6.13	Base Models for MESFET Library.	131
6.14	Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for Library Model, $n = 5$, of MESFET Library.	133
6.15	Comparison of Number of Training Samples Needed and Training Time Used for MESFET Library When Developed by Standard MLP and the Proposed Neural Network Structure, respectively.	135

A.1	Format of Neural Network Structure Description Files.	143
A.2	An Example MLP Structure Description File.	145
A.3	An Example KBNN Structure Description File for Mutual Inductance Modeling of Microstrip Lines.	146
A.4	An Example KBNN Structure Description File for Mutual Inductance Modeling of Microstrip Lines (continued).	147
A.5	The Classes within Each Components of POWERNET Software Pack- age.	149
A.6	Format of DEFAULT.scale File.	150
A.7	Format of DEFAULT.train File.	151
A.8	An Example DEFAULT.train File.	151
A.9	Format of Data Files.	152
A.10	Format of C++ Source File: user_func.cpp. This File Implements User Defined Functions.	154
A.11	Format of File: user_func.def. This File Lists User Defined Functions for POWERNET	155

List of Symbols

α_{ij} - Scaling parameter for the jth boundary neuron output in the activation function of the ith region neuron of KBNN

a - Channel thickness of a physics-based MESFET

$\mathbf{A}$ - Inverse of Hessian matrix $\mathbf{H}$ of the error function $E(\mathbf{w})$

$\mathbf{A}_{now}, \mathbf{A}_{old}$ - Approximation of the inverse of Hessian matrix in Quasi-newton training algorithms at the current and the previous weight parameters, respectively

$\Delta\mathbf{A}_{now}$ - Update of the approximation of the inverse of Hessian matrix at the current weight parameters from that at the previous weight parameters in Quasi-newton training algorithms

β_{j0} - Bias parameter in the activation function of the jth output neuron of KBNN

β_{ji} - Scaling parameter for the ith knowledge neuron output in the activation function of the jth output neuron of KBNN

β_k - Bias parameter of the kth output neuron of MLP

b_i - Output of the ith boundary neuron in KBNN

$B(.)$ - Activation function of boundary neurons in KBNN

ϵ_r - Relative dielectric constant of materials

η - Learning rate controlling the step size of weight update during neural network training

η^* - Optimal step size found by line search during neural network training

$\mathbf{e}$ - Error vector containing the error terms for individual outputs of individual samples

$\mathbf{e}_{now}$ - Error vector $\mathbf{e}$ at the current weight parameters for Gauss-Newton and Levenberg-Marquardt training algorithms

$E(\mathbf{w})$ - Error function representing the sum of least squares of the difference between neural model outputs $\mathbf{y}$ and desired outputs $\mathbf{d}$, for all training samples.

$E(\eta)$ - Objective function of line search during neural network training

$\phi^n(i)$ - Base model index of the i th low level module for the n th library model

Φ - Optimization variables in neural network training or interconnect optimization

$\mathbf{f}(\mathbf{x})$ - Relationship between inputs $\mathbf{x}$ and outputs $\mathbf{y}$ in the original component or circuit problems

$\bar{\mathbf{f}}(\mathbf{x}, \mathbf{w})$ - Neural network model for the relationship $\mathbf{f}(\mathbf{x})$

$\gamma_{now}, \lambda_{now}$ - Parameters in conjugate gradient training algorithms

Γ_i - Translation vector for the i th hidden wavelet neuron of wavelet networks

g_{b_i} - Derivative of ϵ w.r.t. the output of the i th boundary neuron of KBNN

g_{r_i} - Derivative of ϵ w.r.t. the output of the i th region neuron of KBNN
 $g_{r'_i}$ - Derivative of ϵ w.r.t. the output of the i th normalized region neuron of KBNN
 g_{x_i} - Derivative of ϵ w.r.t. the output of the i th input neuron of KBNN
 g_{y_j} - Derivative of ϵ w.r.t. the output of the j th output neuron of KBNN
 g_{z_i} - Derivative of ϵ w.r.t. the output of the i th knowledge neuron of KBNN
 $\mathbf{g}$ - Gradient vector of the error function $E(\mathbf{w})$
 $\Delta \mathbf{g}$ - Difference of gradient at the current and the previous weight parameters for Quasi-newton training algorithms
 $\mathbf{g}_{initial}, \mathbf{g}_{next}, \mathbf{g}_{now}, \mathbf{g}_{old}$ - Gradient for neural network training algorithms, at the initial, next, current and previous weight parameters, respectively
 h_c - Stripline conductor height above ground
 h_s - Substrate height for transmission lines
 $\mathbf{h}$ - Update direction vector for weight parameters during neural network training
 $\mathbf{h}_{initial}, \mathbf{h}_{next}, \mathbf{h}_{now}$ - Conjugate direction for conjugate gradient training algorithms, at the initial, next and current weight parameters, respectively
 $\mathbf{H}$ - Hessian matrix of the error function $E(\mathbf{w})$
 $\mathbf{H}^n$ - High level neural module for the n th library model
 i_d - DC drain-current of a physics-based MESFET
 $\mathbf{I}$ - Identity matrix

$\mathbf{J}$ - Jacobian matrix

$\mathbf{J}_{now}$ - Jacobian matrix at the current iteration

κ_i - Dilation parameter for the i th hidden wavelet neuron of wavelet networks

ϑ - Function parameter of Gaussian functions for RBF

l - Channel length of a physics-based MESFET

L_{ii} - Cross sectional per unit length self inductance of the i th conductor of transmission lines

L_{ij} - Cross sectional per unit length mutual inductance between the i th and the j th conductors of transmission lines

L_m - Cross sectional per unit length mutual inductance of any two conductors of transmission lines

L_s - Cross sectional per unit length self inductance of one conductor of transmission lines

$\mathbf{L}_i^n$ - i th low level neural module for the n th library model

μ - Positive definiteness correction parameter of the Jacobian matrix in Levenberg-Marquardt training algorithms

M^n - Total number of training samples for the n th library model

M_B^j - Total number of training samples for the j th base model

ν - Momentum factor for Backpropagation neural network training

N - Number of conductors in a transmission line

N_b - Number of neurons in the boundary layer **B** of KBNN

N_B - Total number of base models in the library

$N_{B_j}^n$ - Number of times the j th base model is reused in the low level neural modules of the n th library model

N_C - Total number of models in a library

N_h - Total number of hidden layers in feedforward neural networks

N_l - Number of neurons in hidden layer l of feedforward neural networks

N_L^n - Total number of low level neural modules in the n th library model

N_p - Number of samples for the neural network training

N_r - Number of neurons in the region layer **R** of KBNN

$N_{r'}$ - Number of neurons in the normalized region layer **R'** of KBNN

N_x - Number of parameters of a given device or a circuit

N_y - Number of response parameters of a given device or a circuit

N_z - Number of neurons in the knowledge layer **Z** of KBNN

ω - Angular frequency

$\psi(.)$ - Radial type mother wavelet function in wavelet networks

$\Psi(.)$ - Activation function of knowledge neurons in KBNN

ρ_d - Doping density of a physics-based MESFET

r - Resistance value

r_i - Output of the i th region neuron in KBNN

r'_i - Output of the i th normalized region neuron in KBNN

$\mathbf{R}$ - Region layer of KBNN

$\mathbf{R}'$ - Normalized region layer of KBNN

$\sigma(t)$ - Activation function of hidden neurons of feedforward neural networks

s_i - Spacing between the i th and the $i + 1$ th conductor of transmission lines

S_{ij} - S-parameter from port i to port j of a device or a circuit

θ_i - Center of radial basis function of the i th hidden neuron of RBF

θ_i^l - Bias parameter of i th neuron of l th hidden layer of MLP

θ_{ij} - Bias parameter for the j th boundary neuron output in the activation function of the i th region neuron of KBNN

t - Time

$\mathbf{u}$ - Input vector of low level neural modules in hierarchical neural network structure

$\mathbf{u}_i^n$ - Input vector of the i th low level neural module for the n th library model

$\mathbf{U}^n$ - Input vector of all the low level neural modules of the n th library model

$\mathbf{U}_i^n(.)$ - i th knowledge hub for the n th library model

v_{ki} - Weight of the link between i th neuron of the last hidden layer and k th neuron of output layer in feedforward neural networks

$\mathbf{v}$ - Weight parameters associated with the output layer of MLP

$\mathbf{v}_i$ - A vector of parameters in the i th boundary neuron activation function $B_i(\cdot)$ of KBNN

V_d, V_g - AC drain-source and gate-source voltages of a physics-based MESFET

V_D, V_G - DC drain-source and gate-source voltages of a physics-based MESFET

$\mathbf{V}^n$ - Weight vector of the high level neural module for the n th library model

w_{ij}^l - Weight of the link between j th neuron of $l - 1$ th hidden layer and i th neuron of l th hidden layer in feedforward neural networks

$\mathbf{w}$ - A N_w -vector containing the weight parameters of the neural network model

$\mathbf{w}_{initial}, \mathbf{w}_{next}, \mathbf{w}_{now}, \mathbf{w}_{old}$ - Weight parameters for neural network training algorithms, at the initial epoch, next epoch, current epoch and previous epoch, respectively

$\Delta\mathbf{w}_{now}, \Delta\mathbf{w}_{old}$ - Current and previous weight update during neural network training

$\mathbf{w}^l$ - Weight parameters associated with the l th hidden layer of MLP

$\mathbf{w}_i$ - A vector of parameters in the i th knowledge neuron of KBNN

$\mathcal{W}$ - Conductor width of transmission lines

$\mathcal{W}_i$ - Width of the i th conductor of transmission lines

$\mathcal{W}_c$ - Channel width of a physics-based MESFET

$\mathbf{W}_j$ - Weight vector for the jth base model

ξ_{jik} - Weight parameter which weights the modulation effect of the kth normalized region neuron output to the ith knowledge neuron output in the activation function of the jth output neuron of KBNN

$\mathbf{x}$ - A N_x -vector containing parameters of a given device or a circuit

$\mathbf{x}_p$ - A N_x -vector representing the pth sample of $\mathbf{x}$

$\mathbf{X}$ - Input layer of KBNN

$\mathbf{X}^n$ - Input vector of the nth library model

$\mathbf{X}^{n,k}$ - Input part of the kth training sample for the nth library model

$\mathbf{X}_B^j$ - Input vector of the jth base model

$\mathbf{X}_B^{j,k}$ - Input part of the kth training sample for the jth base model

y_{pk} - kth output of the neural network when the input presented to the network is $\mathbf{x}_p$

$\bar{y}_i^l$ - Output of ith neuron of lth layer of feedforward neural networks

$\mathbf{y}$ - A N_y -vector containing the responses of a given device or a circuit under consideration

$\mathbf{Y}$ - Output layer of KBNN

$\mathbf{Y}^n$ - Output vector of the nth library model

$\mathbf{Y}_B^j$ - Output vector of the jth base model

z_i - Output of the i th knowledge neuron in KBNN

$\mathbf{z}$ - Output vector of low level neural modules in hierarchical neural network structure

$\mathbf{z}_i^n$ - Output vector of the i th low level neural module for the n th library model

$\mathbf{Z}$ - Knowledge layer of KBNN

$\mathbf{Z}^n$ - Input vector of the high level neural module of the n th library model

$\mathbf{Z}^{n,k}$ - Input part of the k th training sample for the high level neural module of the n th library model

Chapter 1

Introduction

1.1 Motivations

The effective use of computer aided design tools in both electrical and physical design stages are important in designing RF and microwave circuits and systems with shrinking design margins and expanding system complexities. The need to reduce design iterations of such systems further demands that the tools be fast and reliable.

At high frequencies, conventional empirical and electrical models are not sufficient, especially for today's drive for first-pass design success. Detailed electromagnetic (EM) models and physics-based models of passive and active components become important. However, these models are computationally intensive because the model evaluation typically involves numerically solving partial differential equations. On the other hand, the desire to reduce time-to-market leads to the trend of increased use of upfront computer simulation instead of hardware prototyping, where many behaviors of the circuits and systems such as performance, manufacturability, electrical and physical reliabilities, can be predicted by computers before hardware implementation. Statistical analysis and yield optimization are among the key components in modern CAD tools in order to meet these requirements. Statistical design is a highly repet-

itive process in which component models and circuits need to be solved repetitively following known or assumed statistical distributions leading to more intensive computations. With the trend towards more functionalities within one system, the size and complexity of the circuits and systems that need to be solved tend to increase steadily. This results in not only analysis of large scale problems but also evaluation of large number of components. Statistical analysis coupled with the large scale and massive analysis of EM or physics-based models becomes a computationally prohibitive task. Addressing this challenge is one of the motivations of this thesis work. Fundamental to CAD tools is that all components are represented by mathematical models. The accuracy and efficiency of model evaluations will have a significant impact on many levels of circuit design including modeling, simulation, optimization, statistical and yield design. Empirical models, which assume simple mathematical relationships of the components, were developed to speed up model evaluations. Simpler models are fast but are often under limited assumptions or mismatch may occur between computer solutions and hardware measurements thus prolonging design cycle. This contradiction in the choice between reliability and efficiency of models exists in most standard modeling approaches.

Artificial neural networks are information processing systems inspired by the ability of human brain to learn from observations and to generalize by abstraction. The fact that neural networks can learn totally different things, led to their use in diverse fields such as pattern recognition, speech processing, control, medical applications and more. Since the introduction of neural networks into the microwave area in 1993, neural networks have gradually gained attention as a fast and flexible vehicle to circuit modeling, simulation and optimization. Neural networks enjoy the advantage of learning from given examples, and have the ability to generalize. Once trained by

corresponding microwave data which are obtained through solving original EM and physics problems or through measurement, neural networks can be used for rapid estimation of microwave component and circuit performances in highly repetitive design processes such as optimization and statistical design. By combining the fast evaluation capability of neural networks and the detailed EM/physics effects in the original problem, the new approach breaks the aforementioned contradictions and offers both speed and accuracy. How to fully explore this new area and to further develop efficient neural model methodology becomes another motivation.

By nature, neural networks are very fast in model evaluation because of simple function evaluations and easy weighted sum operations used inside neural networks. They are usually considered as black box models structurally embedding no problem-dependent information. The capability of neural networks to model EM/physics effects comes from its training by microwave data samples. Large amount of training data is usually needed to ensure adequate modeling of EM/physics effects. However, in microwave applications, obtaining a relatively large set of training data by either EM/physics simulation, or by measurement, can be expensive and/or difficult. The underlying reason is that simulation/measurement may have to be performed for many combinations of geometrical/material/process parameters. Therefore, there is a serious need to develop neural network models in the presence of limited training data. Using standard neural network structures, the model accuracy would be poor due to limited information available to learn from. On the other hand, engineers in the microwave area have rich empirical experience with components and circuits. Especially, many empirical models exist which can provide estimates of component and circuit behavior without training data. The gap between these empirical experience and the learning ability of neural networks motivates us in the search for a new

technique that has the potential to remedy the weakness of both empirical models and normal neural network models.

As the applications of neural networks in the electronics and microwave system design grow, more and more components will need to be modeled by neural networks. In a brute-force way, these component neural models can be treated individually and trained from scratch. Development of large number of neural models is therefore very time consuming. This thesis also deals with such a new scenario in this research area, that is, the development of sets of neural models. In reality, electronics and microwave components are usually grouped into libraries in commercial CAD tools. The effectiveness of CAD tools in the analysis and design of microwave systems depends on the richness and accuracy of the libraries of models. An algorithm for tackling such massive neural network development is one of the challenges that need to be addressed. Issues associated with library problems which are expected to be different from those of pure component model problems need to be identified and utilized.

1.2 Contributions

The objectives of this thesis are to incorporate prior knowledge into neural network structure and the framework for developing a library of neural models for microwave design. The author contributed substantially to the following original developments presented in this thesis:

- (1) : A novel neural network structure, called the Knowledge Based Neural Network (KBNN) structure, which incorporates the microwave empirical knowledge directly into the neural network structure and reduces the need for large amounts of training data while maintaining model accuracy [1]. A detailed formulation

of the novel trainable structure, featuring multiple sets of knowledge neurons and the mechanism to switch among them, is developed. The advantages of KBNN are demonstrated through various passive and active microwave component neural models.

- (2) : A training method for Knowledge Based Neural Network structure featuring gradient based l_2 optimization. A generalized error backpropagation step is derived allowing derivative information of KBNN be calculated for every weight of the structure.
- (3) : A knowledge based hierarchical framework for developing a library of neural models, which significantly reduces the total number of training samples and overall development cost of library of neural models [2]. Component neural models are of hierarchical structures with two levels of modules, i.e., low- and high-level neural modules. A detailed formulation incorporating both functional and structural microwave knowledge into library development is developed. It is demonstrated through various libraries of passive and active microwave component models.
- (4) : A step-by-step algorithm for systematic development of a neural model library. The algorithm defines the sequence of module development and allows the reuse of base models throughout the library development.
- (5) : An object-oriented computer software package for neural network training and other utilities written in C++, which accommodates a very flexible and novel neural network structure description allowing arbitrary connections of neurons and user-defined activation functions for neurons.

1.3 Outline of the Thesis

The thesis is organized as follows.

In Chapter 2, a review of the literature is presented. Neural network applications in the microwave design area are first reviewed. A detailed review of various neural network structures and training algorithms useful to microwave design is then presented together with the examples to illustrate the comparisons of different structures and training algorithms. A review of high speed interconnect optimization featuring neural network models is also conducted.

Chapter 3 introduces a novel neural network structure, namely Knowledge Based Neural Networks. The structure features several groups of non-fully-connected neurons. A detailed description and discussion of each type of neuron is presented. A new training method utilizing a generalized error backpropagation for gradient calculation is developed for this structure. An example is included at the end of this chapter to illustrate the behavior and advantage of this proposed structure over traditional Multilayer Perceptrons (MLP).

In Chapter 4, three application examples of Knowledge Based Neural Networks are presented, namely circuit waveform modeling, transmission line modeling and physics-based MESFET modeling. It is demonstrated that the proposed technique enhances neural model accuracy, especially for unseen data, and reduces the need for large sets of training data.

Chapter 5 presents a knowledge based hierarchical neural network framework for the development of a library of neural models. The description starts with a statement of the problem of library development. In this hierarchical framework, base models are first defined. Hierarchical neural models are created for each component of library.

An algorithm is presented in this chapter which permits the hierarchical neural models to be developed systematically.

In Chapter 6, three examples of neural model library development are presented, namely, libraries of stripline models, microstrip models and physics-based MESFET models. The proposed knowledge based hierarchical method is compared with the conventional method of neural model development in these library examples. The proposed hierarchical approach substantially reduces the cost of library development due to less data collection and shorter training time, and at the same time improves model reliability.

Finally, Chapter 7 describes the conclusions and makes suggestions for the future.

Chapter 2

Literature Review

2.1 Neural Network Applications in Microwave Design

The drive for manufacturability-oriented design and reduced time-to-market in the microwave industry require design tools that are accurate and fast. Statistical 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] [3] [4] [5] [6], and microwave circuit design [3] [5] [7] [8]. 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] [4] [9], vias [4] [10], spiral inductors [6] [11], FET devices [1] [12] [13] [14],

HEMT devices [15] [16], power transistors and power amplifiers [17], coplanar waveguide (CPW) circuit components [5], packaging and interconnects [18] etc. Neural networks have been used in circuit simulation and optimization [3] [14] [19], signal integrity analysis and optimization of VLSI interconnects [9] [18], microstrip circuit design [20], microwave filter design [21], IC modeling [12], process design [22], synthesis [7] [23], Smith Chart representation [8] and microwave impedance matching [24]. The neural network technologies have been applied to microwave circuit optimization and statistical design with neural network models at both device and circuit levels [3] [13]. 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] [3], more accurate than polynomial and empirical models [25], allow more dimensions than table lookup models [26] and are easier to develop when a new device/technology is introduced [19].

Neural network structures and training are two of the most important issues in applying neural networks for solving engineering problems. Theoretically, neural network models are black box models, whose accuracy depends on the data presented to it during training. A good collection of the training data, i.e., data which is well-distributed, sufficient and accurately measured/simulated, is the basic requirement to obtain an accurate model. However, in the reality of microwave design, training data collection/generation may be very expensive/difficult. There is a trade off between the amount of training data needed for developing the neural model, and the accuracy demanded by the application. Other issues affecting the accuracy of neural models are due to the fact that many microwave problems are non-linear, non-smooth, or containing many variables. An appropriate structure would help to achieve higher model accuracy with fewer training data [27]. For example, a feedforward neural

network with smooth switching functions in the hidden layer is good at modeling smooth, slowly varying nonlinear functions, while a feedforward neural network with Gaussian functions in the hidden layer could be more effective in modeling nonlinear functions with large variations. The size of the structure, i.e., the number of neurons, is also an important criterion in the development of a neural network. Too small a network cannot learn the problem well, but too large a size will lead to over-learning.

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. The most popular training algorithm is Back Propagation (BP), which was proposed in mid 1980's. Later, a lot of variations to improve the convergence of BP were proposed. Optimization methods such as second order methods and decomposed optimization have also been used for neural network training in recent years. A noteworthy challenge encountered in the neural network training is the existence of numerous local minima. Global optimization techniques have been combined with conventional training algorithms to tackle this difficulty.

2.2 Neural Based Microwave Modeling: 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 lines. 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

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

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

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

where $\mathbf{w}$ are the parameters of the neural network model, which is also called the weight vector in neural network literature, and $\mathbf{x}$ and $\mathbf{y}$ are called the inputs and outputs of the neural model. The definition of $\mathbf{w}$, and how $\mathbf{y}$ 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 $\mathbf{y}$ and desired outputs $\mathbf{d}$ from simulation/measurement

$$E(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{N_p} \sum_{k=1}^{N_y} (y_{pk} - d_{pk})^2 = \frac{1}{2} \sum_{p=1}^{N_p} \sum_{k=1}^{N_y} (\bar{f}_k(\mathbf{x}_p, \mathbf{w}) - d_{pk})^2 \quad (2.4)$$

is minimized. In (2.4) d_{pk} is the k th element of vector $\mathbf{d}_p$, y_{pk} is the k th output of the neural network when the input presented to the network is $\mathbf{x}_p$, and $\bar{f}_k(\cdot)$ is the

k th element of function $\bar{f}(\cdot)$ of (2.3). 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 detailed review of neural network structures used for this purpose is presented.

2.3 Neural Network Structures

In this section, different ways of realizing $y = \bar{f}(\mathbf{x}, \mathbf{w})$ are described. The definition of $\mathbf{w}$ and how y is computed from $\mathbf{x}$ and $\mathbf{w}$ in the model determine different neural model architectures.

2.3.1 Standard Feedforward Neural Networks

Feedforward neural networks are a basic type of neural networks capable of approximating generic classes of functions, including continuous and integrable ones [28]. An important class of feedforward neural networks is Multilayer Perceptrons (MLP).

Figure 2.1: Multilayer Perceptron (MLP) structure. Typically, the network consists of an input layer, one or more hidden layers and an output layer.

and the hyperbolic tangent function

$$\sigma(t) = \frac{(e^t - e^{-t})}{(e^t + e^{-t})} \quad . \quad (2.11)$$

All these functions are bounded, continuous, monotonic and continuously differentiable. Training parameter vector $\mathbf{w}$ includes

$$\mathbf{w} = \begin{bmatrix} w_{ij}^l & j = 1, \dots, N_{l-1}, i = 1, \dots, N_l, l = 1, \dots, N_h; \\ \theta_i^l & i = 1, \dots, N_l, l = 1, \dots, N_h; \\ v_{ki} & i = 1, \dots, N_{N_h}, k = 1, \dots, N_y; \\ \beta_k & k = 1, \dots, N_y \end{bmatrix} . \quad (2.12)$$

It is well known that a two-layered MLP (no hidden layers) is not capable of approximating generic nonlinear continuous functions [29] [30]. The universal approximation theorem [31] [32] 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 [32].

However, in reality a neural network can only have a 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 in terms of generalization capability and mapping capability [33]. In the function approximation or regression area, generalization capability is a major concern. It is shown in [34] 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 [33] for the mapping capability of the network.

Feedforward neural networks which have only one hidden layer, and which 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 [35] [36]. Park and Sandberg showed that RBF networks also have universal approximation ability [37] [38]. Universal convergence of RBF nets in function estimation and classification has been proved by Krzyzak et al. [39].

The output neurons of RBF networks are also linear neurons. The overall input-output transfer function of RBF networks is defined as

$$y_k = \sum_{i=1}^{N_1} v_{ki} \sigma(\|\mathbf{x} - \theta_i\|), \quad k = 1, \dots, N_y \quad (2.13)$$

where θ_i is the center of radial basis function of the i th hidden neuron, v_{ki} is the weight of the link from i th hidden neuron to the k th output neuron. Some of the commonly used radial basis activation functions are [40]

$$\sigma(t) = \exp\left[-\left(\frac{t}{\vartheta}\right)^2\right] \quad (2.14)$$

$$\sigma(t) = (c^2 + t^2)^\zeta, \quad 0 < \zeta < 1 \quad (2.15)$$

where (2.14) is the Gaussian function, (2.15) is the multiquadratic function and, ϑ , c , and ζ are the function parameters. Training parameters $\mathbf{w}$ includes θ_i , v_{ki} , $i = 1, \dots, N_1$, $k = 1, \dots, N_y$ and ϑ or c and ζ .

Although MLP and RBF are both feedforward 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 [41]. 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 [42], 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 [43], network pruning [44]. Regularization [45] 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 [46], 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.

2.3.2 Neural Network Structures with Prior Knowledge

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. A possible way to address this problem is to use prior knowledge. There are two approaches to the use of prior knowledge during neural model development process. In the first approach, the prior knowledge is used to define a suitable preprocessing of the simulation/measurement data such that the input-output mapping is simplified. The first method in this approach, a hybrid EM-ANN model was proposed in [4]. Existing approximate models are used to construct the input-output relationship of the microwave component. The EM simulator data is then used to develop a neural network model to correct for the difference between the approximate model (source model) and the actual EM simulation results, and as such the method is also being called the difference method. In this way, the complexity of the input/output relationship that neural network has to learn is considerably reduced. This reduction in complexity helps to develop an accurate neural network model with less training data [4]. The second method in this approach, is the prior knowledge input (PKI) method [10]. In this method, the source model outputs are used as inputs for the neural network model in addition to the original problem inputs. As such, the input/output mapping that must be learnt by neural network is that between the output response of the source model and that of the target model. In the extreme case, where the target outputs are the same as the source model outputs, the learning problem is reduced to a one-to-one mapping.

It was shown [10] that PKI method performs better than the difference method for 2-port GaAs microstrip ground vias.

The second approach to use prior knowledge, is to incorporate the knowledge directly into neural network internal structure, e.g., [47]. This knowledge provides additional information of the original problem, which may not be adequately represented by the limited training data. The first method of this approach, uses symbolic knowledge in the form of rules to establish the structure and weights in a neural network [47] [48] [49]. The weights, e.g., the certainty factor associated with rules [50], or both the topology and weights of the network [51] can be revised during training. The second method of this approach uses prior knowledge to build a modular neural network structure, i.e., to decide the number of modules needed, and the way in which the modules interact with each other [52] [53]. Another neural network structure worth mentioning here is the local model network (LMN) [54] [55] [56] [57] which is an engineering- oriented network. It is based on the decomposition of a nonlinear dynamic system's operating range into a number of smaller operating regimes, and the use of simple local models to describe the system within each regime. The third method restricts the network architecture through the use of local connections and constraining the choice of weights by the use of weight sharing [41]. These existing approaches to incorporate knowledge are largely using symbolic information which exist in pattern recognition area. In the microwave modeling areas, however, the important problem knowledge is usually available in the form of functional empirical models [58] [59]. Although [52] tried to insert the mathematical model directly into modular neural network structure, the assumption of the method that the mathematical model is valid for the entire problem space, is too strong to be suitable for circuit modeling.

2.3.3 Combining Neural Networks

In the neural network research community, a recent development called combining neural networks are presently proposed, addressing issues of network accuracy and training efficiency. Two categories of approaches have been developed: ensemble-based approach and modular approach [60]. In the ensemble-based approach [60] [61], several networks are trained such that each network approximates the overall task in its own way. The outputs from these networks are then combined to produce a final output for the combined network. The aim is to achieve a more reliable and accurate ensemble outputs than would be obtained by selecting the best net. Optimal linear combinations (OLC's) of neural networks were proposed and investigated in [61] [62], which is constructed by forming weighted sums of the corresponding outputs of the individual networks. The second category, e.g., [41] [63], features a modular neural network structure that results from the decomposition of tasks. The decomposition may be either automatic (based on the blind application of a data partitioning algorithm, such as hierarchical mixtures-of-experts [64]) or explicit (based on prior knowledge of the task or the specialist capabilities of the modules, e.g., [52] [53]). The modular neural network consists of several neural networks, each optimized to perform a particular sub-task of an overall complex operation. An integrating unit then selects or combines the outputs of the networks to form the final output of the modular neural network. Thus, the modular approach not only simplifies the overall complexity of the problem [65], but also facilitates incorporation of problem knowledge into the network structure. This leads to improved overall network reliability and/or training efficiency [53] [66].

2.3.4 Other Neural Network Structures

As mentioned earlier, one of the major problems in the construction of neural network structure, is the determination of the number of hidden neurons. Many techniques have been proposed to address this problem. In this section, some of the structures will be reviewed, with emphasis on wavelet networks which are a very systematic approach already used in microwave area, cascade-correlation network and projection pursuit network, the later two networks being popular constructive networks.

The idea of combining wavelet theory with neural networks has been recently proposed [67] [68] [69]. 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

$$\bar{y}_i^1 = \psi(\kappa_i(\mathbf{x} - \mathbf{\Gamma}_i)), \quad i = 1, \dots, N_1 \quad (2.16)$$

where $\psi(\cdot)$ is the radial type mother wavelet function, κ_i are dilation parameters and $\mathbf{\Gamma}_i$ are translation vectors for the i th hidden wavelet neuron. Both κ_i and $\mathbf{\Gamma}_i$ are adapted together with v_{ki} and β_k of equation (2.7) during training.

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 κ_i , Γ_i and v_{ki} , 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 [68], 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 [67] 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 [69]. Besides retaining the advantage of faster training, wavelet networks have a guaranteed upper bound on the accuracy of approximation [70] with a multi-scale structure. Wavelet networks have been used in non-parametric regression estimation [67] and were trained based on noisy observation data to avoid the problem of undesirable local minima [70]. In [17], 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.

Besides wavelet network, there are a number of constructive neural network struc-

tures, the most representative among them being Cascade Correlation network (CasCor) [71]. A CasCor network begins with a minimal network (without hidden neurons), then automatically adds hidden neurons one by one during training. Each newly added hidden neuron receives a connection from each of the network's original inputs and also from every pre-existing hidden neuron, thus resulting in a multi-layer network. For regression tasks, a CasPer algorithm [72] which constructs a neural network structure in a similar way as CasCor was proposed. CasPer does not use the maximum correlation training criterion of CasCor, which tends to produce hidden neurons that saturate, thus making CasCor more suitable for classification tasks rather than regression tasks. However, both CasCor and CasPer may lead to very deep networks and high fan-in to the hidden neurons. A towered cascade network was proposed in [73] to alleviate this problem.

Another constructive neural network structure is projection pursuit learning network (PPLN), which adds neurons with trainable activation functions one-by-one, within a single hidden layer without cascaded connections [74]. The CasCor learns the higher-order features using cascaded connection while PPLN learns it using the trainable activation functions. Every time a new hidden neuron is added to PPLN, it first trains the new neuron by cyclically updating the output layer weights, the smooth activation function, and the input layer weights associated with this neuron. Then a backfitting procedure is employed to fine tune the parameters associated with the existing hidden neurons. PPLN is able to avoid the curse of dimensionality by interpreting high-dimensional data through well-chosen low-dimensional linear projections.

2.4 Neural Network 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, $\{(\mathbf{x}_p, \mathbf{d}_p), p = 1, \dots, N_p\}$, 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 MLP neural network $\mathbf{w}$ is given by (2.12).

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

$$\min_{\mathbf{w}} E(\mathbf{w}) \triangleq \sum_{p=1}^{N_p} \epsilon_p(\mathbf{w}) \quad (2.17)$$

where $\epsilon_p(\mathbf{w})$ is defined as the error in the neural network outputs due to p th sample,

$$\epsilon_p(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{N_y} (y_{pk} - d_{pk})^2 = \frac{1}{2} \sum_{k=1}^{N_y} (\bar{f}_k(\mathbf{x}_p, \mathbf{w}) - d_{pk})^2 \quad (2.18)$$

where d_{pk} is the k th element of vector $\mathbf{d}_p$, y_{pk} is the k th output of the neural network when the input presented to the network is $\mathbf{x}_p$, and $\bar{f}_k(\cdot)$ is the k th element of function $\bar{\mathbf{f}}(\cdot)$ of (2.3).

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. In iterative descent methods, we start with an initial guess of $\mathbf{w}$ and then iteratively update $\mathbf{w}$. The next point of $\mathbf{w}$, denoted as $\mathbf{w}_{next}$, is determined by a step down from the current point $\mathbf{w}_{now}$ along a direction vector $\mathbf{h}$,

$$\mathbf{w}_{next} = \mathbf{w}_{now} + \eta \mathbf{h} \quad (2.19)$$

where η is a positive step size regulating the extent to which we can proceed in that direction. Every training algorithm has its own scheme for updating the weights of the neural network.

2.4.2 Backpropagation Algorithm and Its Variants

One of the most popular algorithms for neural network training is Back Propagation (BP) algorithm [75], proposed by Rumelhart, Hinton and Williams in 1986. The BP algorithm is a stochastic algorithm based on the steepest descent principle [76], wherein the weights of the neural network are updated along the negative gradient direction in the weight space. The update formulae are given by

$$\Delta \mathbf{w}_{now} = \mathbf{w}_{next} - \mathbf{w}_{now} = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_{now}} \quad (2.20)$$

$$\Delta \mathbf{w}_{now} = \mathbf{w}_{next} - \mathbf{w}_{now} = -\eta \frac{\partial \epsilon_p(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_{now}} \quad (2.21)$$

where η called learning rate controls the step size of weight update. Update formula (2.21) is called update sample-by-sample, where the weights are updated after each sample is presented to the network. Update formula (2.20) 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 in (2.20) and (2.21) as proposed by [75], provided significant improvements to the basic backpropagation, reducing the weight oscillation:

$$\Delta \mathbf{w}_{now} = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_{now}} + \nu \Delta \mathbf{w}_{old} = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_{now}} + \nu (\mathbf{w}_{now} - \mathbf{w}_{old}) \quad (2.22)$$

$$\Delta \mathbf{w}_{now} = -\eta \frac{\partial \epsilon_p(\mathbf{w})}{\partial \mathbf{w}} \big|_{\mathbf{w}=\mathbf{w}_{now}} + \nu \Delta \mathbf{w}_{old} = -\eta \frac{\partial \epsilon_p(\mathbf{w})}{\partial \mathbf{w}} \big|_{\mathbf{w}=\mathbf{w}_{now}} + \nu (\mathbf{w}_{now} - \mathbf{w}_{old}) \quad (2.23)$$

where ν 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 [41]. Other approaches to reduce weight oscillation have also been proposed, such as invoking a correction term that uses the difference of gradients [77], and the constrained optimization approach where constraints on weights are imposed to achieve better alignment between weight updates in different epochs [78].

As neural network research moved from the state-of-the-art paradigm to real-world applications, the training time and computational requirements associated with training have become significant considerations [79] [80] [81]. Some of the real world applications involve large-scale networks, in which case the development of fast and efficient learning algorithms becomes extremely important [80]. A variety of techniques have been developed, and among them are two important classes of methods. One of them is based on advanced learning rate and momentum adaptation, and heuristic rules of BP, and the other is based on the use of advanced optimization techniques. The latter shall be discussed in Section 2.4.3.

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 [41], e.g., adaptation according to training errors [82]. One of the most interesting works in this area is the delta-bar-delta rule proposed by Jacobs [83]. 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. Sparsity of hidden neuron activation pattern has also been utilized in [81] [84] [85] to reduce the computation involved during training. 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 [86], an Enhanced Back Propagation algorithm [87] 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 [88]. The algorithm in [88] 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. An interesting adaptation scheme based on the concept of dynamic learning rate optimization is presented in [89], in which the first and second order derivatives of the objective function w.r.t. the learning rate are calculated from the information gathered during the forward and backward propagation. Another work [77], which is considered as an extension of Jacob's heuristics, corrects the values of weights near the bottom of the error surface ravine with a new acceleration algorithm. This correction term uses the difference between gradients, to reduce the weight oscillation during training. 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 line search methods [76], so that the training error is reduced or optimized

along the given updating direction. Examples in this category are line search based on quadratic model [90], and line search based on linear interpolation [91] [92]. One other way to improve training efficiency is the gradient reuse algorithm [93]. 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 Training Algorithms Using Gradient-based Optimization Techniques

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 [94] [95] with the development of second-order learning algorithms for neural networks. Papers [91] [96] reviewed the first- and second- order optimization methods for learning in feedforward

neural networks.

Let $\mathbf{h}$ be the direction vector, η be the learning rate, $\mathbf{w}_{now}$ be the current value of $\mathbf{w}$, then the optimization will update $\mathbf{w}$ such that

$$E(\mathbf{w}_{next}) = E(\mathbf{w}_{now} + \eta \mathbf{h}) < E(\mathbf{w}_{now}) \quad (2.24)$$

The principal difference between various descent algorithms lies in the procedure to determine successive update directions ($\mathbf{h}$) [97]. Once the update direction is determined, the optimal step size could be found by line search,

$$\eta^* = \min_{\eta > 0} E(\eta) \quad (2.25)$$

where

$$E(\eta) = E(\mathbf{w}_{now} + \eta \mathbf{h}) . \quad (2.26)$$

When downhill direction $\mathbf{h}$ is determined from the gradient $\mathbf{g}$ of the objective function E , such descent methods are called as gradient-based descent methods. The procedure for finding a gradient vector in a network structure is generally similar to backpropagation [75] in the sense that the gradient vector is calculated in the direction opposite to the flow of output from each neuron. Take MLP as an example, this is done by means of derivative chain rule starting from output layer,

$$\frac{\partial E}{\partial \mathbf{w}^l} = \frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \bar{\mathbf{y}}^l} \frac{\partial \bar{\mathbf{y}}^l}{\partial \mathbf{w}^l}, \quad l = N_h \quad (2.27)$$

and then through the various layers down towards the input layer,

$$\frac{\partial E}{\partial \mathbf{w}^l} = \frac{\partial E}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \bar{\mathbf{y}}^{N_h}} \cdots \frac{\partial \bar{\mathbf{y}}^{l+1}}{\partial \bar{\mathbf{y}}^l} \frac{\partial \bar{\mathbf{y}}^l}{\partial \mathbf{w}^l}, \quad l = N_h - 1, N_h - 2, \dots, 1 \quad (2.28)$$

where $\mathbf{y}$ represents the final outputs of neural network, and $\bar{\mathbf{y}}^l$ represents the outputs of the l th hidden layer of the neural network.

A. Conjugate Gradient Training Algorithms

The conjugate gradient methods are originally derived from quadratic minimization and the minimum of the objective function E can be efficiently found within N_w iterations. With initial gradient $\mathbf{g}_{initial} = \frac{\partial E}{\partial \mathbf{w}}|_{\mathbf{w}=\mathbf{w}_{initial}}$, and direction vector $\mathbf{h}_{initial} = -\mathbf{g}_{initial}$, the conjugate gradient method recursively constructs two vector sequences [92],

$$\mathbf{g}_{next} = \mathbf{g}_{now} + \lambda_{now} \mathbf{H} \mathbf{h}_{now} \quad (2.29)$$

$$\mathbf{h}_{next} = -\mathbf{g}_{next} + \gamma_{now} \mathbf{h}_{now} \quad (2.30)$$

$$\lambda_{now} = \frac{\mathbf{g}_{now}^T \mathbf{g}_{now}}{\mathbf{h}_{now}^T \mathbf{H} \mathbf{h}_{now}} \quad (2.31)$$

$$\gamma_{now} = \frac{\mathbf{g}_{next}^T \mathbf{g}_{next}}{\mathbf{g}_{now}^T \mathbf{g}_{now}} \quad (2.32)$$

or,

$$\gamma_{now} = \frac{(\mathbf{g}_{next} - \mathbf{g}_{now})^T \mathbf{g}_{next}}{\mathbf{g}_{now}^T \mathbf{g}_{now}} \quad (2.33)$$

where $\mathbf{h}$ is called the conjugate direction and $\mathbf{H}$ is the Hessian matrix of the objective function E . Here, (2.32) is called the Fletcher-Reeves formula and (2.33) 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 $\mathbf{g}_{next} = \frac{\partial E}{\partial \mathbf{w}}|_{\mathbf{w}=\mathbf{w}_{next}}$. This $\mathbf{g}_{next}$ can be used as the vector of (2.29), and as such (2.31) 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 the neural network feedforward operation for a complete cycle of samples. 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 (2.18), 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 [86], 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 [98] which avoids the line search per learning iteration by using Levenberg-Marquardt approach to scale the step size.

B. Quasi-Newton Training Algorithms

Similar to conjugate gradient method, the quasi-Newton method was derived from quadratic objective function. The inverse of Hessian matrix $\mathbf{A} = \mathbf{H}^{-1}$ is used to bias the gradient direction, following Newton's method. In quasi-Newton training method, the weights are updated using

$$\mathbf{w}_{next} = \mathbf{w}_{now} - \eta \mathbf{A}_{now} \mathbf{g}_{now} \quad (2.34)$$

The $\mathbf{A}$ matrix here is not computed. It is successively estimated employing rank 1 or rank 2 updates, following each line search in a sequence of search directions [99],

$$\mathbf{A}_{now} = \mathbf{A}_{old} + \Delta \mathbf{A}_{now} \quad (2.35)$$

There are two major rank 2 formulae to compute $\Delta \mathbf{A}_{now}$,

$$\Delta \mathbf{A}_{now} = \frac{\mathbf{h}\mathbf{h}^T}{\mathbf{h}^T \Delta \mathbf{g}} - \frac{\mathbf{A}_{old} \Delta \mathbf{g} \Delta \mathbf{g}^T \mathbf{A}_{old}}{\Delta \mathbf{g}^T \mathbf{A}_{old} \Delta \mathbf{g}} \quad (2.36)$$

or,

$$\Delta \mathbf{A}_{now} = \left(1 + \frac{\Delta \mathbf{g}^T \mathbf{A}_{old} \Delta \mathbf{g}}{\mathbf{h}^T \Delta \mathbf{g}}\right) \frac{\mathbf{h}\mathbf{h}^T}{\mathbf{h}^T \Delta \mathbf{g}} - \frac{\mathbf{h} \Delta \mathbf{g}^T \mathbf{A}_{old} + \mathbf{A}_{old} \Delta \mathbf{g} \mathbf{h}^T}{\mathbf{h}^T \Delta \mathbf{g}} \quad (2.37)$$

where

$$\mathbf{h} = \mathbf{w}_{now} - \mathbf{w}_{old}, \quad \Delta \mathbf{g} = \mathbf{g}_{now} - \mathbf{g}_{old} . \quad (2.38)$$

(2.36) is called as the DFP (Davidon-Fletcher-Powell) formula and (2.37) is called as the BFGS (Broyden-Fletcher-Goldfarb-Shanno) formula.

Standard quasi-Newton methods require N_w^2 storage space to maintain an approximation of the inverse Hessian matrix and a line search is indispensable to calculate a reasonably accurate step length, where N_w is the total number of weights in the neural network structure. Limited-memory (LM) or one-step BFGS is a simplification in which the inverse Hessian approximation is reset to the identity matrix after every iteration, thus avoiding the need to store matrices. In [100] a second-order learning algorithm is proposed based on LM BFGS update. A reasonably accurate step size is efficiently calculated in one- dimensional line search by a second-order approximation of the objective function. Parallel implementation of second-order gradient-based MLP training algorithms featuring full and limited memory BFGS algorithms were presented in [101]. Wavelet neural networks trained by BFGS algorithm were used for the modeling of large-signal hard-nonlinear behavior of power transistors in circuit design [17]. Quasi-Newton training algorithm that employs the exact line search possesses the quadratic termination property. Through the estimation of inverse Hessian matrix, quasi-Newton has faster convergence rate than conjugate gradient method.

2.4.4 Training Algorithms Utilizing Decomposed Optimization

As seen in the earlier discussions, implementation of powerful second-order optimization techniques for neural network training has resulted in significant advantages in training. The second-order methods are typically much faster than BP but could require the storage of inverse Hessian matrix, and its computation or an approximation thereof. For large neural networks, training could be a very large scale optimization. Decomposition is an important way to solve the large scale optimization problems. Several training algorithms that decompose the training task by training the neural network layer-by-layer have been proposed [106] [107] [108]. In [107], the weights ($\mathbf{v}$) of the output layer and the output vector ($\bar{\mathbf{y}}^{N_h}$) of the previous layer are treated as two sets of variables. An optimal solution pair ($\mathbf{v}, \bar{\mathbf{y}}^{N_h}$) is first determined to minimize the sum-squared-error between the desired neural network outputs and the actual outputs. The current solution $\bar{\mathbf{y}}^{N_h}$ is then set as the desired output of the previous hidden layer, and optimal weight vectors of the hidden layers are recursively obtained. In the case of continuous function approximation, [108] optimizes each layer with an objective to increase a measure of linearity between the internal representations and the desired output.

Linear programming can be used to solve large scale linearized optimization problems. Neural network training was linearized and formulated as a constrained linear programming in [109]. In this work, weights are updated with small local changes satisfying the requirement that none of the individual sample errors should increase, subject to the constraint of maximizing the overall reduction in the error. However, extension of such a method for efficient implementation in large networks needs spe-

cial considerations. In [106], a layer-by-layer optimization of neural network with linearization of the nonlinear hidden neurons was presented, which does not rely on the evaluation of local gradients. To limit the unavoidable linearization error, a special penalty term is added to the cost function and layers are optimized alternately in an iterative process.

A combination of linear/nonlinear programming techniques could reduce the degree of non-linearity of the error function with respect to the hidden layer weights and decrease the chances of being trapped in a local minimum. As such, a hybrid training algorithm would be favorable for a large-scale problem [57]. For a feedforward neural network with the linear or linearized output layer weights, training algorithms were developed by adapting the nonlinear hidden layer weights using BP [110] or BFGS [57], while employing a linear Least Mean Square error (LMS) algorithm to compute the optimum linear output layer weights.

RBF networks are usually trained by decomposed process. The nonlinear hidden layer weights of the RBF network can be interpreted as centers and widths, thus making it possible to organize these neurons in a manner which reflects the distribution of the training data. Utilizing this concept, the hidden layer weights can be fixed through unsupervised training, such as k-means clustering algorithm [111]. The output layer weights can then be obtained through linear LMS algorithm. On the other hand, the development of heuristics to initially assign the hidden layer weights of MLP, is very hard due to its black box characteristics. Consequently, it is not possible to train MLP neural networks with this decomposed strategy. However, the hybrid linear/nonlinear training algorithm presented in [57] integrates the best features of linear LMS algorithm of RBF, and non-linear optimization techniques of MLP into one routine. This routine could be suitable for any feedforward networks

with linear output layer weights. The advantages of this technique are the reduced number of independent parameters and guaranteed global minimum w.r.t. to output layer weights [57].

2.4.5 Global Training Algorithms

Another important class of methods use random optimization techniques which 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 [112]; genetic algorithms which evolve the structure and weights of the neural network through generations in a manner that is similar to biological evolution [113]; the Langevin updating (LV) rule in multilayer perceptron, in which noise is added to the weights during training [114]; and a stochastic minimization algorithm for training neural networks [115], which is basically a random optimization method with no gradient information needed. Since the convergence of pure random search techniques tends to be very slow, a more general method is the hybrid method which combines the conventional gradient-based training algorithms with random optimization, e.g., [90]. This work 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.

2.5 Examples: Feedforward Neural Networks and Their Training

Standard feedforward neural networks, MLP and RBF, have been used in many microwave applications. This section demonstrates the use of these neural model structures in several microwave examples and their training by various training algorithms.

MLP and RBF were used to model a physics-based MESFET. Device physical/process parameters (channel length l , channel width $\mathcal{W}_c$, doping density ρ_d , channel thickness a) and terminal voltages, i.e., gate-source voltage (V_G) and drain-source voltage (V_D), are neural network input parameters. Drain-current, i.e., i_d , is the neural network output. The training data and test data were simulated using OSA90 [116]. Three sets of training data with 100, 300, 500 samples, and one set of test data with 413 samples were used. The model accuracy is shown in Table 2.1.

Training Sample size	Model Type	MLP					RBF			
	No. of Hidden Neurons	7	10	14	18	25	20	30	40	50
100	Ave. Test Error(%)	1.65	2.24	2.60	2.12	2.91	6.32	5.78	6.15	8.07
300	Ave. Test Error(%)	0.69	0.69	0.75	0.69	0.86	1.37	0.88	0.77	0.88
500	Ave. Test Error(%)	0.57	0.54	0.53	0.53	0.60	0.47	0.43	0.46	0.46

Table 2.1: Model Accuracy Comparison Between MLP and RBF.

Generally, RBF will need more hidden neurons than MLP to achieve similar model accuracy due to their localization nature of activation function. RBF training requires sufficient amount of data. On the other hand, the training of RBF is easy to converge. The generalization capability of MLP is better than RBF as seen in the table as

available amount of training data becomes less.

The most popular training algorithms for standard feedforward neural networks are the adaptive backpropagation, conjugate gradient, quasi-Newton and Levenberg-Marquardt algorithms. Two examples, i.e., three conductor microstrip line and physics-based MESFET, were used to illustrate the performance of these algorithms.

For the microstrip line example, there are 5 input neurons corresponding to conductor width (W), spacing between conductors (s_1, s_2), substrate height (h_s) and relative dielectric constant (ϵ_r) as shown in Figure 2.2. There are 6 output neurons

Figure 2.2: Three conductor microstrip line.

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 [117]. A three layer MLP structure with 28 hidden neurons is chosen as the sample structure. The training results are shown in Table 2.2. CPU time is given for Pentium (200 MHz). The total CPU time used by Levenberg-Marquardt method is around 20 minutes. The adaptive back-

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 2.2: Comparison of Various Training Algorithms for Microstrip Line Example.

propagation used many epochs and settled down to good accuracy of 0.252%, with around 4 hours of training time. On the contrary, quasi-Newton method achieved similar accuracy only within 35 minutes. This confirms the faster convergence rate of second order method. Usually quasi-Newton has very fast convergence rate when approaching the minimum. But at the beginning of training, its performance may not be very strong. Another strategy is to use conjugate gradient method at the first stage of training, then followed by quasi-Newton method. If we take the MLP network already trained by conjugate gradient and then continue training by quasi-Newton method, the model test error was reduced to 0.167%. Total training time is around 2 hours.

For the MESFET example, the inputs to neural model include frequency (ω), channel thickness (a), gate-bias voltage (V_g), and drain-bias voltage (V_d). This particular MESFET has a fixed gate length of $0.55 \mu m$ and gate width of $1 mm$. The outputs include real and imaginary parts of S_{11} , S_{12} , S_{21} and S_{22} . Training and test samples were obtained using the simulator OSA90 [116] with the Khatibzadeh and Trew model [151]. This is a relatively complicated example compared to the microstrip line

necessitating the use of transmission line models. In recent years, there has been a thrust of research in this area and various modeling and simulation techniques are being proposed, e.g., [120]-[128].

An intuitive solution to delay reduction is to reduce the interconnect length. However this may lead to increased density of the system and consequently more coupling and crosstalk. For practical circuit boards with a large number of interconnects and various design criteria, the problem can become very complicated. There is an increasing need that the signal integrity criteria be directly used in interconnect design optimization [129] [130] [131] [132]. Design variables at different levels in package and interconnect network, constraints which affect the freedom of physical adjustments of interconnect parameters during optimization, and design criteria based on signal integrity must be formulated. Straight use of conventional circuit optimization approaches will not always be suitable due to several difficulties encountered in signal integrity oriented design. In reality the number of interconnects in a printed circuit board is usually large and design specifications at many interconnection ports need to be satisfied. Conventional optimization techniques may balk at such large scale problem [133]. Another difficulty is due to the highly repetitive simulation of distributed transmission line models or even EM models [120] [134] since interconnect geometry may be adjusted during optimization. Mixed time/frequency domain simulations required for frequency-dependent interconnect models [120] [122] are very expensive if used inside an optimization loop. Such an optimization will be much slower than that for conventional lumped circuit design, making it unsuitable for practical design. The hierarchical nature of signal integrity which exists at chip, multichip module (MCM), printed circuit board (PCB) levels, and which is dependent on interconnect parameters at all levels [120] also poses challenges to standard optimization approaches.

In addition, electrical design is now more and more constrained by other design factors such as thermal [135] and Electromagnetic (EM) effects [136], both of which are becoming important considerations in an overall design.

In recent years, research into the interconnect optimization problem has been very active, and several important progresses have been made. Different formulations of optimization have been investigated in both time- and frequency-domain approaches. Signal integrity optimization of interconnects at several levels of electronics systems, from chip level, MCM level to PCB level has been studied. Optimization techniques used for such purpose range from analytical solutions in single transmission line configuration to numerical techniques such as Quasi-Newton, to advanced techniques such as parallel and multilevel optimization. In this section, we review the macro-modeling techniques for interconnect optimization with the attention to how neural network technique is applied.

2.6.1 Macromodeling Techniques for Interconnect Optimization

A complete signal integrity analysis and optimization require repeated simulation of distributed networks which can be very CPU intensive. Yield analysis and optimization of high-speed interconnects is an even more computationally expensive task. Practical interconnect designs are often iterative in nature. For example designers may need to check signal integrity when ground noise budget (design specifications) is modified and the interconnects are reoptimized. Most existing optimization approaches using repeated circuit simulations are more oriented towards off-line computations and are not suitable for such an iterative design.

Here, modeling approaches for speeding up optimization are discussed. The pur-

pose of such approaches is to build a macromodel to replace partially or completely the original circuit simulation during optimization. Macromodels are designed to be much faster than the original simulator so that the resulting optimization process is accelerated. Another purpose is to facilitate hierarchical optimization by representing lower level components through macromodels [137].

Examples of models for such purpose include the conventional polynomial, e.g., [25], spline, e.g., [138], and table lookup models, e.g., [26]. A recent development in modeling large interconnect networks is the macromodeling based on Padé approximation [126]. Neural network is another recent development in this area [3] [9] and has been applied to interconnect optimization.

Among these models, polynomial, table lookup and neural network models can represent abstract nonlinear algebraic relationships of a function with several variables. The functions can be circuit responses of interest, or optimization objective function. The variables can be optimization variables. The model is usually built from samples of data obtained from simulating the original problem at several or many sampling points in the space of variables. The polynomial models, e.g., quadratic models, are popular conventional models for optimization but they usually have limited nonlinearity and require model building and updating during optimization. The table lookup models and neural network models can represent more nonlinearity, and can be used without updating during optimization. Model building can be completely done outside optimization. Such optimization is more suitable for interactive circuit design and optimization, since much online computation has been shifted to off-line model building, which is done only once and can be used and reused for all subsequent optimizations. The table lookup models are simpler to build but are practically limited to small dimensions. For high-dimensional problem with many variables in

Physical/EM level [141]

At this level, neural network models map the geometrical parameters of the interconnects to equivalent electrical parameters. For example, the interconnect neural network models would have width, length, height and distance parameters of interconnects as inputs $\mathbf{x}$ and the *RLCG* parameters as outputs $\mathbf{y}$. Such a relationship originally required a Quasi-TEM solution of EM problems [117].

Device level [3] [13]

At this level, a device can be modeled by neural networks. A physics based MESFET transistor was modeled in [13]. The inputs $\mathbf{x}$ could be gate-to-source voltage V_G , drain-to-source voltage V_D . Additional inputs can be geometrical parameters such as gate length, width, channel thickness, etc. The outputs $\mathbf{y}$ could be the drain current i_d as shown in Figure 2.3. Such relationship originally required a physics-based analysis.

Circuit level [3] [142]

At this level, the whole circuit may be replaced by a neural network model. With the inputs $\mathbf{x}$ as the values of different parameters in the circuit, neural network models can output the responses of the circuit such as delay, crosstalk and so on. Figure 2.4 shows an example circuit, consisting of a 3-transmission line network, and Figure 2.5 shows the neural network model for this circuit.

B. Usage of Neural Network Models

Once trained, the neural network model is ready to be integrated into simulations and optimizations. It will be used in the recalling mode predicting the outputs $\mathbf{y}$ upon the given input information $\mathbf{x}$ which may not have been seen during training.

Figure 2.3: Neural network model at the device level, an example of MESFET model after [3].

Figure 2.4: A 3-transmission line network with coupled transmission lines.

Figure 2.5: An example of neural network model at the circuit level for the circuit of Figure 2.4.

This property of neural networks is called generalization. Also this prediction only needs several product-and-sum operations, far less computation than solving the original problem, e.g., physics/EM/circuit equations. When neural network outputs $\mathbf{y}$ represent terminal voltages and currents of an element, the model needs to be formulated into a MNA (Modified Nodal Admittance) stamp and presented to the circuit simulator [3]. For circuit level neural network model, the neural network can directly represent circuit responses such as signal delay and crosstalk. In this case the error functions for optimization are directly formulated from neural network outputs $\mathbf{y}$. A powerful feature of the neural network approach is that neural network input parameters $\mathbf{x}$ can be optimization variables, therefore expensive repetitive simulation of the original problem can be replaced by repetitive neural network recalling.

2.6.3 Yield Optimization of a 3-Transmission Line Network Using Neural Models

A distributed network containing 3 two-conductor transmission lines is shown in Figure 2.4. The signal integrity analysis of this circuit involves eigenvalue solutions of the transmission line matrices and mixed time/frequency domain analysis such as FFT or numerical inversion of Laplace transform [143]. In [9], a neural-network macromodel was used to reduce CPU time in the yield optimization of this circuit. Six variables including the length and the separation between coupled conductors of each transmission line were chosen as neural network input parameters $\mathbf{x}$ and also as the optimization variables Φ . The neural network output $\mathbf{y}$ were signal integrity responses including the signal propagation delay of V_{out1} and V_{out2} , and the magnitude of crosstalks V_{cross1} and V_{cross2} . The numbers of neurons in the input, output and the hidden layers of the neural network were 6, 4 and 200, respectively. All the

six transmission line parameters were statistical variables following Gaussian distribution with 5% standard deviation. The specifications for the 4 responses were 1.8ns, 1.72ns, 0.02V and 0.02V, respectively. The specifications were severely violated before optimization and the yield was 0% based on exact circuit simulation of 500 random samples. Figure 2.6(a) shows the histogram of delay of Vout2. A yield optimization with the neural network approach was performed. The histogram of delay of Vout2 after optimization is plotted in Figure 2.6(b) based on 500 exact circuit simulations. The yield was increased to 88%.

2.7 Conclusions

A significant speed up of simulation and optimization of electronic and microwave circuits could be achieved by replacing the electronic and microwave component models represented in detailed physics/EM equations, with neural models trained by the corresponding physics/EM data. However, most of the existing neural network structures are of black box type without any problem dependent information embedded. To develop accurate microwave component models using these structures, a large amount of physics/EM training data is thus needed, which results in high cost of neural model development. Although a few approaches have been proposed to insert prior problem knowledge into neural network structures, they are largely using symbolic information and are often oriented to pattern recognition area. In microwave modeling areas, however, the most important problem knowledge is more functional than symbolic/structural, making the existing knowledge network methods not suitable for microwave applications.

(a)

(b)

Figure 2.6: Histogram of the signal delay (in seconds) for V_{out2} of the 3-transmission line network obtained from exact circuit simulation of 500 random samples. Samples with delay value more than 1.72ns violate specifications. (a) before optimization (b) after optimization using neural models.

Chapter 3

Proposed Knowledge Based Neural Networks

3.1 Introduction

In this chapter a new microwave-oriented knowledge based neural network (KBNN) [1] [144] [145] is proposed in which microwave knowledge in the form of empirical functions or analytical approximations is embedded into internal neural network structures. Switching boundary and region neurons are introduced in the model structure to reflect microwave cases where different equations or formulas with different parameters can be interchangeably used in different regions of the input parameter space. The proposed structure does not follow the rigorous layer-by-layer structure in MLP, and a new training algorithm is developed since conventional BackPropagation is not applicable. The proposed technique enhances neural model accuracy especially for unseen data and reduces the need of large set of training data.

3.2 Proposed Knowledge Based Neural Network Structure

The conventional neural networks, e.g., MLP, are blind black box models. The standard switching activation functions such as sigmoid or hyperbolic tangent functions used in MLP are far different from the various engineering models. The generalization ability of the MLP model comes only from the information encoded in the training data. In an effort to reduce the amount of needed training data, i.e., the development cost of neural models, the proposed Knowledge Based Neural Network inserts the available engineering knowledge about the circuit/components directly into neurons, creating knowledgeable neurons.

The proposed Knowledge Based Neural Network (KBNN) structure is a nonfully connected structure shown in Figure 3.1. There are 6 layers in the structure, namely input layer **X**, knowledge layer **Z**, boundary layer **B**, region layer **R**, normalized region layer **R'** and output layer **Y**. The input layer **X** accepts parameters **x** from outside the model. The knowledge layer **Z** is the place where microwave knowledge resides in the form of single or multidimensional functions $\Psi(\cdot)$. For knowledge neuron i in the **Z** layer:

$$z_i = \Psi_i(\mathbf{x}, \mathbf{w}_i), \quad i = 1, 2, \dots, N_z \quad (3.1)$$

where **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 as a function of its gate length, gate width, channel thickness, doping density, gate voltage and drain voltage [59]. The boundary layer **B** can incorporate knowledge in the form of problem dependent boundary functions $B(\cdot)$ or in

Figure 3.1: The proposed KBNN structure.

the absence of boundary knowledge just as linear boundaries. Neuron i in this layer is calculated by

$$b_i = B_i(\mathbf{x}, \mathbf{v}_i), \quad i = 1, 2, \dots, N_b \quad (3.2)$$

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:

$$r_i = \prod_{j=1}^{N_b} \sigma(\alpha_{ij} b_j + \theta_{ij}), \quad i = 1, 2, \dots, N_r \quad (3.3)$$

where α_{ij} and θ_{ij} are the scaling and bias parameters, respectively. The normalized region layer $\mathbf{R}'$ contains rational function based neurons [46] to normalize the outputs of the region layer:

$$r'_i = \frac{r_i}{\sum_{j=1}^{N_r} r_j}, \quad i = 1, 2, \dots, N_{r'}, \quad N_{r'} = N_r. \quad (3.4)$$

The output layer $\mathbf{Y}$ contains second order neurons [146] combining knowledge neurons and normalized region neurons:

$$y_j = \sum_{i=1}^{N_z} \beta_{ji} z_i \left(\sum_{k=1}^{N_{r'}} \xi_{jik} r'_k \right) + \beta_{j0}, \quad j = 1, 2, \dots, N_y \quad (3.5)$$

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

$$y_j = \sum_{i=1}^{N_z} \beta_{ji} z_i r'_i + \beta_{j0}, \quad j = 1, 2, \dots, N_y. \quad (3.6)$$

The prior knowledge encoded in $\Psi(\cdot)$ and/or $B(\cdot)$ need not to be very accurate and complete. Several forms of functions $\Psi(\cdot)$ and/or $B(\cdot)$ can coexist in the network. The

constant coefficients in the original empirical functions can be replaced by trainable parameters and more bias/scale parameters can be added to provide extra variability among different neurons. If some input parameters are not present in the original empirical models, they can be added to the knowledge function $\Psi(\cdot)$ in the weighted sum form.

3.3 Discussions

As reviewed in Chapter 2, there was an earlier attempt to put mathematical model, e.g., $\exp(-8.0(x^2 + y^2))$, into neural network structure as [52]. However, it needs the assumption that this mathematical model is valid for the entire problem space. In circuit design reality, the empirical knowledge and models cannot satisfy this assumption.

Our proposed structure was inspired by the fact that practical empirical functions are usually valid only in a certain region of the parameter space. Several empirical functions may be needed to represent different behavior in different regions of parameter space. Each knowledge neuron in Knowledge Based Neural Network represents an empirical model. A collection of several empirical models may cover a larger or even the entire problem space. This corresponds to a group of knowledge neurons in KBNN. The critical operation in this network structure is the selection of which empirical model is going to be picked up when a certain input vector is fed in. This selection is made by a set of normalized region neurons through the output layer. The functionality of this layer represents the integrating units in a modular neural network. For a particular input vector, ideally only one region neuron will have output 1 and other region neurons will have output 0, indicating that this input vector belongs

to that particular region. Only the empirical model which is valid in this region will be picked up by the output layer. The association of the empirical model and its valid region is represented by the weights in the output layer, which is learnt from training samples. This mechanism represents the Local Model Network (LMN) which selects different local models for different regimes. The original LMN was motivated from chemical engineering and the power industry, where definite regimes of LMN are identified during procedures such as start-up, shut-down and product shift of continuous processes [57]. However, in the microwave design area, manual definition of regions and association of regions corresponding to each particular component model is very tedious. In the proposed KBNN, regions are constructed through trainable boundaries. The location of the boundaries and the selection of particular boundaries for a region is determined during training. The default boundaries are straight lines in the parameter space. It is also possible to insert prior knowledge into region-building process by providing boundary function $B(\cdot)$. The switching mechanism for region neuron activation functions expands the feature of Sigmoidal Radial Basis Function [147] into high dimensional space and in a more generalized form. This KBNN 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.

Another perspective of the KBNN can be seen by comparing the neuron functions in KBNN, MLP and RBF. As reviewed in Chapter 2, RBF may need more hidden neurons than MLP in modeling nonlinear functions, because the sigmoid function of MLP can represent a relatively larger portion of the original problem than the Gaussian function of RBF. In KBNN, the empirical model can represent much larger portion of the original problem and can represent it more closely than the sigmoid

function of MLP. Therefore, the number of empirical functions and training samples needed can be much fewer. This structure can be applied to a wide range of problems in the sense that different problems can be solved by the same structure with different empirical models plugged in.

3.4 Training Approach of Knowledge Based Neural Networks

Let $\mathbf{y}$ represent the neural model output. Let $\mathbf{d}$ represent the corresponding outputs from the original problem, e.g., original EM simulation or measurement. Neural network learns from a set of training data $(\mathbf{x}_p, \mathbf{d}_p)$, $p = 1, \dots, N_p$, where N_p is the total number of data samples. The trainable parameters of the proposed KBNN, denoted as Φ , includes $\mathbf{w}_i$, $i = 1, \dots, N_z$, $\mathbf{v}_i$, $i = 1, \dots, N_b$, α_{ij} and θ_{ij} , $i = 1, \dots, N_r$, $j = 1, \dots, N_b$, β_{ji} , β_{j0} and ξ_{jik} , $k = 1, \dots, N_{r'}$, $i = 1, \dots, N_z$, $j = 1, \dots, N_y$. The purpose of training is to determine the weight parameters Φ inside the KBNN model such that the error between the desired outputs $\mathbf{d}$ and the actual outputs $\mathbf{y}$ from KBNN is minimized,

$$\min_{\Phi} \quad \frac{1}{2} \sum_{p=1}^{N_p} \sum_{j=1}^{N_y} (y_{pj} - d_{pj})^2 \quad . \quad (3.7)$$

A gradient based l_2 optimization technique is employed in the training of KBNN, which requires the derivative of error from individual outputs of individual training samples with respect to each weight in KBNN, i.e., a Jacobian matrix $\mathbf{J}$. Since our network does not follow a regularly layered Multilayer Perceptron structure and microwave empirical functions instead of standard activation functions are used in neurons, conventional backpropagation algorithm is not applicable. A new scheme extending the error backpropagating idea is derived to obtain this Jacobian matrix.

The error backpropagation path starts from the output layer $\mathbf{Y}$. Let the error of neural network prediction for a particular output of a sample, e.g., the q th output of the p th sample, be

$$\epsilon_{pq} = \frac{1}{2}(y_{pq} - d_{pq})^2 \quad (3.8)$$

where y_{pq} is the actual q th neural network output and d_{pq} is the q th output of the p th training sample. Jacobian matrix $\mathbf{J}$ requires the derivative of error ϵ_{pq} with respect to each weight in KBNN for each p and q combinations. For simplicity, define ϵ as a generic representation of ϵ_{pq} by dropping p and q , i.e., $\epsilon = \epsilon_{pq} = \frac{1}{2}(y_{pq} - d_{pq})^2$. Let the derivative of ϵ with respect to the output of individual neurons be denoted as g . For output layer($\mathbf{Y}$ layer), g_{y_j} is defined as $g_{y_j} = \partial\epsilon/\partial y_j$, which can be easily calculated from error function (3.8), as

$$\frac{\partial\epsilon}{\partial y_j} = \begin{cases} y_j - d_j = y_q - d_q, & j = q \\ 0, & j = 1, 2, \dots, N_y \text{ but } j \neq q \end{cases} \quad (3.9)$$

Then the derivatives of error ϵ with respect to weights β 's and ξ 's inside the output neuron are obtained as

$$\frac{\partial\epsilon}{\partial\beta_{ji}} = \begin{cases} (y_q - d_q)z_i(\sum_{k=1}^{N_{r'}} \xi_{jik}r'_k), & i = 1, 2, \dots, N_z, \quad j = q \\ 0, & i = 1, 2, \dots, N_z, \quad j = 1, 2, \dots, N_y \text{ but } j \neq q \end{cases}, \quad (3.10)$$

$$\frac{\partial\epsilon}{\partial\beta_{j0}} = \begin{cases} y_q - d_q, & j = q \\ 0, & j = 1, 2, \dots, N_y \text{ but } j \neq q \end{cases}, \quad (3.11)$$

$$\frac{\partial\epsilon}{\partial\xi_{jik}} = \begin{cases} (y_q - d_q)\beta_{ji}z_i r'_k, & k = 1, 2, \dots, N_{r'}, \quad i = 1, 2, \dots, N_z, \quad j = q \\ 0, & k = 1, 2, \dots, N_{r'}, \quad i = 1, 2, \dots, N_z \\ & j = 1, 2, \dots, N_y \text{ but } j \neq q \end{cases}. \quad (3.12)$$

The proposed KBNN training scheme begins to differ from conventional back-propagation below the output layer $\mathbf{Y}$, where the error propagation is split into two paths, one through the knowledge layer $\mathbf{Z}$ down to the input layer $\mathbf{X}$, and the other through the normalized region layer $\mathbf{R}'$, the region layer $\mathbf{R}$ and the boundary layer $\mathbf{B}$ down to the input layer $\mathbf{X}$. In the first path, g_z can be obtained as

$$g_{z_i} = \sum_{j=1}^{N_y} \frac{\partial \epsilon}{\partial y_j} \frac{\partial y_j}{\partial z_i} = (y_q - d_q) \beta_{qi} \left(\sum_{k=1}^{N_{r'}} \xi_{qik} r'_k \right), \quad i = 1, 2, \dots, N_z \quad (3.13)$$

Continuing the derivative chain rule, the derivatives of error ϵ with respect to the weights inside knowledge neurons (i.e., $\mathbf{w}_i$) are

$$\frac{\partial \epsilon}{\partial \mathbf{w}_i} = \frac{\partial \epsilon}{\partial z_i} \frac{\partial z_i}{\partial \mathbf{w}_i} = g_{z_i} \frac{\partial \Psi_i}{\partial \mathbf{w}_i}, \quad i = 1, 2, \dots, N_z \quad (3.14)$$

where $\frac{\partial \Psi_i}{\partial \mathbf{w}_i}$ is obtained from problem-dependent microwave empirical functions. In the second path, $g_{r'}$ is first obtained,

$$g_{r'_k} = \sum_{j=1}^{N_y} \frac{\partial \epsilon}{\partial y_j} \frac{\partial y_j}{\partial r'_k} = (y_q - d_q) \sum_{i=1}^{N_z} \beta_{qi} z_i \xi_{qik}, \quad k = 1, 2, \dots, N_{r'} \quad (3.15)$$

The g 's for the next two layers, i.e., $\mathbf{R}$ and $\mathbf{B}$ layers, are

$$g_{r_i} = \sum_{j=1}^{N_{r'}} \frac{\partial \epsilon}{\partial r'_j} \frac{\partial r'_j}{\partial r_i} = g_{r'_i} \frac{1}{\sum_{k=1}^{N_r} r_k} - \frac{1}{(\sum_{k=1}^{N_r} r_k)^2} \left(\sum_{j=1}^{N_{r'}} g_{r'_j} r_j \right), \quad i = 1, 2, \dots, N_r \quad (3.16)$$

$$g_{b_i} = \sum_{j=1}^{N_r} \frac{\partial \epsilon}{\partial r_j} \frac{\partial r_j}{\partial b_i} = \sum_{j=1}^{N_r} g_{r_j} r_j (1 - \sigma(\alpha_{ji} b_i + \theta_{ji})) \alpha_{ji}, \quad i = 1, 2, \dots, N_b \quad (3.17)$$

The derivatives of error ϵ with respect to weights α 's and θ 's inside region neurons are

$$\frac{\partial \epsilon}{\partial \alpha_{ij}} = \frac{\partial \epsilon}{\partial r_i} \frac{\partial r_i}{\partial \alpha_{ij}} = g_{r_i} r_i (1 - \sigma(\alpha_{ij} b_j + \theta_{ij})) b_j, \quad i = 1, 2, \dots, N_r, \quad j = 1, 2, \dots, N_b \quad (3.18)$$

$$\frac{\partial \epsilon}{\partial \theta_{ij}} = \frac{\partial \epsilon}{\partial r_i} \frac{\partial r_i}{\partial \theta_{ij}} = g_{r_i} r_i (1 - \sigma(\alpha_{ij} b_j + \theta_{ij})), \quad i = 1, 2, \dots, N_r, \quad j = 1, 2, \dots, N_b \quad (3.19)$$

KBNN and MLP, respectively. The original function is

$$y(x) = \sin(25x)e^{-x}, \quad x \in [0, 1] \quad (3.23)$$

Different number of training samples were randomly selected in $[0, 1]$. This is a highly nonlinear function. Many samples are needed in order to teach a standard MLP to a satisfactory performance. If some knowledge can be obtained by observation, e.g., a sinusoidal waveform with decay, a KBNN with this embedded knowledge would require much fewer training samples. A KBNN was built with the knowledge neuron function defined as

$$z_i = \sin(w_{i1}x + w_{i2}) \times (x + w_{i3})^{w_{i4}}, \quad w_{i4} < 0, \quad i = 1, 2, \dots, N_z \quad (3.24)$$

where w_{i1}, w_{i2}, w_{i3} and w_{i4} are the parameters of the function and are trainable. Figure 3.2 shows the comparison between MLP with 5 hidden neurons and KBNN with 1 boundary neuron, 2 knowledge neurons and 2 region neurons, denoted as blz2, in modeling this function when only given 11 training samples. MLP gave totally wrong prediction. KBNN gave much better generalization performance under such condition of inadequate training samples.

(a)

(b)

Figure 3.2: The decay sinusoidal example: (a) y versus x curve from MLP, (b) y versus x curve from KBNN.

Chapter 4

Application Examples of Knowledge Based Neural Networks

4.1 Circuit Waveform Modeling

This example is mainly for illustration purpose showing the concept of incorporation of electrical knowledge into KBNN. A simple circuit with three transmission lines is shown in Figure 4.1 representing signal integrity analysis of high-speed VLSI interconnects and terminations [19]. The excitation of the circuit is a step voltage source with $0.1ns$ rise time at node 1. Neural networks (both KBNN and MLP) were used to model the waveform output at nodes 3 and 4. The inputs of neural models are resistor value r and time t , and the outputs are output voltage v_3 and v_4 . As a circuit knowledge, the waveform can be estimated by a 2-pole approximation with exponentially decayed sinusoids [149]. This 2-pole approximation knowledge, plus an additional resistor variable r as part of the model input, was incorporated into KBNN by providing the knowledge function $\Psi(.)$ at layer $\mathbf{Z}$ as

$$z_i = \Psi_i(r, t, \mathbf{w}_i) = e^{-(w_{i1}r + w_{i2})t} \sin((w_{i3}r + w_{i4})t + w_{i5}r + w_{i6}), \quad i = 1, 2, \dots, N_z \quad (4.1)$$

Figure 4.1: The circuit for waveform modeling example with 3 transmission lines representing high-speed VLSI interconnects.

With this 2-pole approximation, the boundary in the parameter space is independent of time t . This is the knowledge embedded in layer **B** realized as a linear function of resistance only. The output layer is constructed following (3.6). The size of KBNN is represented by the number of neurons in **B** and **Z** layers, e.g., b1z2 representing a KBNN with 1 boundary neurons and 2 knowledge neurons. For the traditional MLP, the size of model is represented by the number of hidden neurons in the network, e.g., 12 representing a 3 layer MLP with 12 hidden neurons.

The training and testing data were obtained by simulation of the original circuit using HSPICE in the time interval from $0ns$ to $7ns$. The training data was created with 5 resistor values of 25, 50, 75, 100 and 125 Ω and with only 10 time points per resistor value and is used to train the neural models (both KBNN and MLP). A KBNN of size b1z2 and four MLPs of sizes 12, 20, 28 and 36 were trained. Testing data includes 69 time points in the waveform on a different set of resistor values ($r = 35, 55, 70, 90$ and 115Ω). Table 4.1 shows the testing results of both

neural net type	model size	no. of weights	average test error	largest test error	correlation coefficient
Standard (MLP)	12	62	2.40%	18.00%	0.9449
Standard (MLP)	20	102	2.06%	13.52%	0.9641
Standard (MLP)	28	142	1.65%	9.13%	0.9784
Standard (MLP)	36	182	2.04%	12.98%	0.9664
Knowledge based (KBNN)	b1z4	36	1.00%	8.71%	0.9894

Table 4.1: Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for Circuit Waveform Modeling Example. The Results Shown Are the Average of Three Different Trainings for Each Model.

models. For each model, the average accuracy from three trainings with different random starting points were used. The accuracy of the model is represented by the error

and correlation coefficient between neural model output and testing data. A value of correlation coefficient closer to 1 indicates good accuracy of neural model. As seen in the table, the training data with 10 time points per resistor value were insufficient for MLP to model this set of waveforms. Figure 4.2 show the waveforms from original HSPICE simulation, KBNN and MLP for resistor value of $115\ \text{ohm}$. The waveforms indicate that the MLP model does not match well with original waveforms. With the same set of insufficient training data, KBNN shows very good accuracy. This illustrates that the prior knowledge provides additional information which is not adequately represented in the original training data. The incorporation of such knowledge into neural models is very helpful to produce a reliable model especially when fewer training data is available.

Another interesting point is that the empirical knowledge alone, i.e., 2-pole approximation with decay sinusoids, is not an adequate model by itself, since it cannot represent the waveform change with respect to resistor values. This example illustrates that through KBNN a simple empirical function can be used in a large parameter space provided that variations of the function with different sets of parameters are used in different regions of the space. The smooth switching between the regions is realized in the KBNN network by region neurons.

It will be very much interesting to look inside KBNN to see how these ideas are realized. For simplification, we will first take a KBNN structure, which models only one output waveform. Figure 4.3 reveals the internal activities of this KBNN structure showing responses of all individual neurons for each pair of resistor value and time points. In all three dimensional graphs, x-axis represents time, y-axis represents resistance and z-axis represents neuron output. According to the simulation results, higher resistance value results in the waveform with smaller decay rate while lower

Figure 4.3: The internal activities of KBNN: responses of individual neurons for each pair of resistor value and time points. This is the illustrative KBNN structure which only models one output waveform.

Figure 4.4: The internal activities of KBNN: responses of individual neurons for each pair of resistor value and time points. This is the actual KBNN structure used in this waveform modeling example, which models multiple output waveforms.

resistance value results in the waveform with larger decay rate. In Figure 4.3, there are two knowledge neurons in the knowledge layer $\mathbf{Z}$. One neuron represents a two-pole approximation with poles relatively farther from origin, i.e., larger decay rate. Another neuron represents a two-pole approximation with poles relatively closer to origin, i.e., smaller decay rate. Neither of these two neurons can approximate the entire waveform across the full value range of the resistor. However, either of them can approximate the waveforms for part of resistor value range. The boundary neuron divides the problem space into two parts, i.e., lower resistance and higher resistance parts. The neurons in the region layer $\mathbf{R}$ show the switching behavior. The neuron output is 1 for one part of the space and is 0 for another part of the space. The switching is a smooth one following the smooth slope of sigmoid function. After the normalization operation on the region neurons, the normalized region neurons divide the entire space into two regions. The output of one normalized region neuron is high in lower resistance region and is low in higher resistance region; the output of the other normalized region neuron is low in lower resistance region and is high in higher resistance region. For illustration purpose, a normalized knowledge layer $\mathbf{Z}'$ is inserted between output layer $\mathbf{Y}$, knowledge layer $\mathbf{Z}$ and normalized region layer $\mathbf{R}'$. From the two neurons in layer $\mathbf{Z}'$, we can see how the knowledge neurons are modulated by normalized region neurons. For example, the second knowledge neuron, which originally represents two-pole approximation in the entire space with poles relatively closer to origin, is multiplied by the second normalized region neuron, i.e., a modulation effect. This results in a normalized knowledge neuron representing such a two-pole approximation only in higher resistance region with poles relatively closer to origin. In the lower resistance region, this normalized knowledge neuron gives almost zero output. On the other hand, the other normalized knowledge neuron represents a

two-pole approximation only in lower resistance region with poles relatively far from origin. In the higher resistance region, it gives almost zero output. The final output of the network is the sum of these two normalized knowledge neurons, which realizes the smooth switching of knowledge functions in the problem space. All the parameters in waveform functions, boundary and region neurons were determined during training through the information encoded in the limited training data.

Figure 4.4 shows the internal activities of the KBNN model (b1z4) used in this waveform modeling example in a similar format. This is the case of modeling multiple output waveforms. The two normalized region neurons are shared by the two output neurons for node 3 and node 4, respectively. Node 3 and node 4 use different set of knowledge neurons to be switched between regions. A similar internal activities can be observed in this graph.

4.2 Transmission Line Modeling

This example demonstrates the proposed KBNN in modeling cross sectional RLCG parameters of transmission lines for analysis of high speed VLSI interconnects [19] and its comparison with traditional Multilayer Perceptrons (MLP). Electromagnetic (EM) simulation of transmission lines is slow especially if it needs to be repeatedly evaluated. Neural networks learned from EM data have been found several hundred times faster than original EM simulation [18]. In this example, MLP and KBNN were used to model the cross sectional per unit length mutual inductance, L_{12} , between two conductors of a coupled microstrip transmission line. The inputs of the neural models are width of conductor (x_1), thickness of conductor (x_2), separation between two conductors (x_3), height of substrate (x_4), relative dielectric constant (x_5)

Parameters	Notation	Range
Conductor Width	x_1	0.10 - 0.25 <i>mm</i>
Conductor Thickness	x_2	17 - 71 μm
Conductor Separations	x_3	0.10 - 0.76 <i>mm</i>
Substrate Height	x_4	0.10 - 0.31 <i>mm</i>
Relative Dielectric Constant	x_5	3.7 - 4.8
Frequency	x_6	0.5 - 2 <i>GHz</i>

Table 4.2: Ranges of Training Data for Neural Model Input Parameters for the Transmission Line Modeling Example.

were never used in training. A further set of testing data with 4096 samples were deliberately selected around/beyond the boundary of the model effective region in input parameter space in order to compare extrapolation accuracy of KBNN and MLP as shown in Table 4.4. A significantly superior performance of KBNN over MLP is demonstrated in the case with smaller training data set, e.g., 100 samples. Furthermore, the overall tendency suggests that the accuracy of KBNN trained by a small set of training data is comparable to that of MLP trained by a larger set of training data. Figures 4.5 and 4.6 reveal more information by showing the error from individual trainings of KBNN and MLP in terms of average and the worst case testing error, respectively. A much more stable performance of KBNN when making an extrapolation prediction is observed over MLP. The error for KBNN increases much slowly compared to that of MLP when test data moves to extrapolation region. Figures 4.7 and 4.8 show the scattering plots of mutual inductance between neural models (MLP with 7 hidden neurons and KBNN(b2z3)) and original simulation for 500 testing samples within training data boundary. The ideal plot is that all points should be at the diagonal line. The points for KBNN are closer to the diagonal line and have smaller worst case error envelope as compared to the plots of MLP trained

training sample size	neural net type	model size	no. of weights	average test error	worst case test error	correlation coefficient
100	Standard (MLP)	7	57	0.95%	9.30%	0.9977
	Standard (MLP)	15	121	1.18%	10.07%	0.9970
	Standard (MLP)	20	161	1.33%	10.04%	0.9960
	Knowledge based (KBNN)	b2z3	51	0.51%	4.18%	0.9995
	Knowledge based (KBNN)	b4z6	128	0.64%	4.16%	0.9991
300	Standard (MLP)	7	57	0.58%	3.12%	0.9993
	Standard (MLP)	15	121	0.56%	3.29%	0.9994
	Standard (MLP)	20	161	0.58%	3.39%	0.9993
	Knowledge based (KBNN)	b2z3	51	0.44%	2.59%	0.9996
	Knowledge based (KBNN)	b4z6	128	0.41%	2.64%	0.9996
500	Standard (MLP)	7	57	0.51%	3.38%	0.9996
	Standard (MLP)	15	121	0.54%	3.30%	0.9994
	Standard (MLP)	20	161	0.56%	3.28%	0.9994
	Knowledge based (KBNN)	b2z3	51	0.41%	2.02%	0.9997
	Knowledge based (KBNN)	b4z6	128	0.38%	2.19%	0.9997

Table 4.3: Model Accuracy Comparison Between MLP and KBNN for Transmission Line Modeling Example with Testing Data in the Same Region as Training Data. The Results Shown Are the Average of Three Different Trainings for Each Model.

training sample size	neural net type	model size	no. of weights	average test error	worst case test error	correlation coefficient
100	Standard (MLP)	7	57	2.38%	12.09%	0.9966
	Standard (MLP)	15	121	2.66%	13.57%	0.9966
	Standard (MLP)	20	161	3.09%	16.00%	0.9941
	Knowledge based (KBNN)	b2z3	51	1.04%	5.43%	0.9993
	Knowledge based (KBNN)	b4z6	128	1.05%	5.01%	0.9993
300	Standard (MLP)	7	57	1.01%	3.61%	0.9994
	Standard (MLP)	15	121	0.91%	3.66%	0.9995
	Standard (MLP)	20	161	0.96%	5.11%	0.9993
	Knowledge based (KBNN)	b2z3	51	0.69%	2.95%	0.9996
	Knowledge based (KBNN)	b4z6	128	0.98%	3.47%	0.9993
500	Standard (MLP)	7	57	0.85%	3.05%	0.9996
	Standard (MLP)	15	121	0.89%	3.65%	0.9996
	Standard (MLP)	20	161	0.91%	4.87%	0.9995
	Knowledge based (KBNN)	b2z3	51	0.77%	2.70%	0.9996
	Knowledge based (KBNN)	b4z6	128	0.87%	2.66%	0.9995

Table 4.4: Model Accuracy Comparison Between MLP and KBNN for Transmission Line Modeling Example with Testing Data around/beyond Training Data Boundary. The Results Shown Are the Average of Three Different Trainings for Each Model.

Figure 4.5: Model accuracy comparison of KBNN and MLP in terms of average testing error for the transmission line example. (a) Testing data sampled within the same range as training data (b) Testing data sampled around/beyond the boundary of training data. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).

Figure 4.6: Model accuracy comparison of KBNN and MLP in terms of worst case testing error for the transmission line example. (a) Testing data sampled within the same range as training data (b) Testing data sampled around/beyond the boundary of training data. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).

Figure 4.7: Scattering plot of mutual inductance L_{12} (a) from MLP and (b) from KBNN for the transmission line modeling example for 500 testing samples. Both models were trained with insufficient training data of only 100 samples.

Figure 4.8: Scattering plot of mutual inductance L_{12} (a) from MLP and (b) from KBNN for the transmission line modeling example for 500 testing samples. Both models were trained with training data of 300 samples.

Figure 4.9: Histograms of testing error of (a) MLP and (b) KBNN for the transmission line modeling example for 4069 testing samples around/beyond training data boundary. Both models were trained by only 100 training samples. Since concentration of errors is closer to 0% for KBNN than that of MLP, KBNN shows better accuracy than MLP.

with the same amount of training data. The plot of KBNN trained by 100 samples has similar worst case error envelope as the plot of MLP trained by 300 samples. The points from KBNN are distributed even closer to the diagonal line than those from MLP. Figure 4.9 shows the histograms of error of MLP and KBNN for testing samples around/beyond training data boundary when trained by insufficient data of only 100 samples. The error for KBNN mostly concentrate in the vicinity of zero. MLP has some error distributed at much higher error level. This also indicates the reliability of KBNN model. Figure 4.10 shows the histograms of error of MLP and KBNN for the same set of testing samples when trained by 300 samples. Similar comparison is also observed.

Figure 4.10: Histograms of testing error of (a) MLP and (b) KBNN for the transmission line modeling example for 4069 testing samples around/beyond training data boundary. Both models were trained by 300 training samples.

4.3 MESFET Modeling

This example demonstrates the use of the proposed KBNN to model physics-based MESFET [3] and its comparison with traditional Multilayer Perceptrons. Device physical/process parameters(channel length l , channel width $\mathcal{W}_c$, doping density ρ_d , channel thickness a) and terminal voltages, i.e., gate-source voltage (V_G) and drain-source voltage (V_D), are neural network input parameters and drain current, i.e., i_d , is the neural network output. The original problem is physics-based [151] and requires a slow numerical simulation procedure. The neural network models (KBNN or MLP) are much faster than the original physics-based FET model, e.g., about 4 seconds by KBNN/MLP and 27 minutes by original FET model to do 1000 repetitive simulations in a MonteCarlo analysis with random values of device physical/geometrical parameters.

There exist empirical formulas for MESFET modeling, e.g., [59]. The knowledge-based neural network (KBNN) is developed incorporating empirical formulas in knowledge layer $\mathbf{Z}$. Training samples were first obtained by simulating original Khatibzadeh and Trew models [151] using OSA90 [116] at randomly selected points. The data range is shown in Table 4.5. Three sets of training data with 100, 300 and 500 samples, respectively, were used. The neural net training was done on SPARC station 5. The CPU time for MLP training by the conventional sample-by-sample error backpropagation approach ranged from 22 minutes to 60 minutes. The CPU time for MLP or KBNN training by the proposed gradient based l_2 optimization approach in batch mode ranged from 20 seconds to 9 minutes.

The ability to extrapolate beyond the boundary of training data is a challenge but an important aspect of a model. Three sets of testing data were created to test

Parameters	Notation	Range
Gate Length	l	0.336 - 0.504 μm
Gate Width	$\mathcal{W}_c$	0.8 - 1.2 mm
Channel Thickness	a	0.28 - 0.42 μm
Doping Density	ρ_d	1.68×10^{23} - 2.52×10^{23} $1/m^3$
Gate Voltage	V_G	-5 - 0 V
Drain Voltage	V_D	0 - 4 V

Table 4.5: Training Data Ranges of Neural Model Input Parameters in MESFET Modeling Example. This is also the Parameter Range of Test Data Set A.

Parameters	Notation	Range
Gate Length	l	0.315 - 0.336 μm and 0.504 - 0.525 μm
Gate Width	$\mathcal{W}_c$	0.75 - 0.8 mm and 1.2 - 1.25 mm
Channel Thickness	a	0.263 - 0.28 μm and 0.42 - 0.438 μm
Doping Density	ρ_d	1.58×10^{23} - 1.68×10^{23} $1/m^3$ and 2.52×10^{23} - 2.63×10^{23} $1/m^3$
Gate Voltage	V_G	-5 - 0 V
Drain Voltage	V_D	0 - 4 V

Table 4.6: Ranges of Neural Model Input Parameters for Test Data Set B in MESFET Modeling Example.

Parameters	Notation	Range
Gate Length	l	0.315 - 0.525 μm
Gate Width	$\mathcal{W}_c$	0.75 - 1.25 mm
Channel Thickness	a	0.263 - 0.438 μm
Doping Density	ρ_d	1.58×10^{23} - 2.63×10^{23} $1/m^3$
Gate Voltage	V_G	-5 - 0 V
Drain Voltage	V_D	0 - 4 V

Table 4.7: Ranges of Neural Model Input Parameters for Test Data Set C in MESFET Modeling Example.

neural network models of various sizes. The first set of testing data, called test data set A, is in the same range as training data in the input parameter space. The second set, called test data set B, contains the data distributed in the extrapolation range which is out of the training data parameter range by 25% as shown in Table 4.6. The other set, called test data set C, contains the data distributed both in the training data parameter range and the extrapolation range as shown in Table 4.7. Test data set A is for the purpose of testing generalization capability. Test data set B is for the purpose of testing extrapolation capability. Test data set C tests overall model performance for a model which is used in the parameter range larger than the range it was trained for.

The results are tabulated in Tables 4.8, 4.9 and 4.10, respectively. In all cases, KBNN outperforms MLP in all the accuracy measures. The superiority is even more significant when fewer training data is available. The overall tendency is that KBNN trained with 100 samples can achieve similar accuracy as that of MLP trained with 300 samples. And KBNN trained by 300 samples is as accurate as MLP trained by 500 samples. Figures 4.11 and 4.12 reveal more information by showing the errors from individual trainings. All the trained KBNNs perform better than any trained MLPs when training data set is small. Moving to the extrapolation region, the accuracy of KBNNs deteriorates much more slowly than that of MLPs. This is because the built-in knowledge in the KBNN gives it more information not seen in the training data. Figure 4.13 shows an example of IV curves from the best performing MLP(with 7 hidden neurons) and KBNN(b5z6) models, both trained by insufficient training data of 100 samples. KBNN is visibly better than MLP. Figure 4.14 shows an example of IV curves from the same models when trained by 300 samples. Both models give good accuracy. Figure 4.15 shows the histograms of error of MLP and KBNN with

training sample size	neural net type	model size	no. of weights	average test error	worst case test error	correlation coefficient
100	Standard (MLP)	7	57	2.14%	45.20%	0.9829
	Standard (MLP)	10	81	2.91%	30.80%	0.9776
	Standard (MLP)	14	113	3.38%	52.53%	0.9610
	Standard (MLP)	18	145	2.75%	35.05%	0.9678
	Standard (MLP)	25	201	3.78%	39.71%	0.9625
	Knowledge based (KBNN)	b5z6	114	1.12%	8.99%	0.9974
	Knowledge based (KBNN)	b6z8	163	1.03%	8.49%	0.9980
300	Standard (MLP)	7	57	0.90%	8.74%	0.9985
	Standard (MLP)	10	81	0.90%	12.89%	0.9978
	Standard (MLP)	14	113	0.96%	14.16%	0.9973
	Standard (MLP)	18	145	0.89%	11.09%	0.9978
	Standard (MLP)	25	201	1.11%	13.66%	0.9962
	Knowledge based (KBNN)	b5z6	114	0.74%	5.68%	0.9990
	Knowledge based (KBNN)	b6z8	163	0.72%	5.72%	0.9991
500	Standard (MLP)	7	57	0.74%	6.15%	0.9990
	Standard (MLP)	10	81	0.70%	5.39%	0.9990
	Standard (MLP)	14	113	0.68%	6.59%	0.9990
	Standard (MLP)	18	145	0.69%	7.51%	0.9989
	Standard (MLP)	25	201	0.78%	10.08%	0.9983
	Knowledge based (KBNN)	b5z6	114	0.61%	5.37%	0.9993
	Knowledge based (KBNN)	b6z8	163	0.61%	5.47%	0.9993

Table 4.8: Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set A. The Results Shown Are the Average of Three Different Trainings for Each Model.

training sample size	neural net type	model size	no. of weights	average test error	worst case test error	correlation coefficient
100	Standard (MLP)	7	57	2.88%	51.52%	0.9753
	Standard (MLP)	10	81	3.86%	53.38%	0.9659
	Standard (MLP)	14	113	4.22%	96.31%	0.9446
	Standard (MLP)	18	145	3.82%	71.04%	0.9615
	Standard (MLP)	25	201	5.04%	88.70%	0.9352
	Knowledge based (KBNN)	b5z6	114	1.56%	21.69%	0.9949
	Knowledge based (KBNN)	b6z8	163	1.43%	11.31%	0.9967
300	Standard (MLP)	7	57	1.24%	12.67%	0.9972
	Standard (MLP)	10	81	1.43%	28.31%	0.9946
	Standard (MLP)	14	113	1.47%	25.91%	0.9945
	Standard (MLP)	18	145	1.38%	39.71%	0.9936
	Standard (MLP)	25	201	1.52%	31.83%	0.9929
	Knowledge based (KBNN)	b5z6	114	1.04%	6.96%	0.9984
	Knowledge based (KBNN)	b6z8	163	1.02%	7.92%	0.9984
500	Standard (MLP)	7	57	1.10%	12.67%	0.9979
	Standard (MLP)	10	81	1.00%	7.64%	0.9985
	Standard (MLP)	14	113	0.96%	9.35%	0.9983
	Standard (MLP)	18	145	0.99%	10.86%	0.9982
	Standard (MLP)	25	201	1.29%	15.53%	0.9975
	Knowledge based (KBNN)	b5z6	114	0.83%	6.64%	0.9988
	Knowledge based (KBNN)	b6z8	163	0.83%	9.40%	0.9988

Table 4.9: Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set B. The Results Shown Are the Average of Three Different Trainings for Each Model.

training sample size	neural net type	model size	no. of weights	average test error	worst case test error	correlation coefficient
100	Standard (MLP)	7	57	2.47%	51.53%	0.9808
	Standard (MLP)	10	81	3.31%	66.73%	0.9726
	Standard (MLP)	14	113	3.70%	96.31%	0.9548
	Standard (MLP)	18	145	3.29%	71.04%	0.9688
	Standard (MLP)	25	201	4.31%	86.60%	0.9488
	Knowledge based (KBNN)	b5z6	114	1.37%	22.21%	0.9958
	Knowledge based (KBNN)	b6z8	163	1.24%	16.67%	0.9972
300	Standard (MLP)	7	57	1.09%	12.84%	0.9978
	Standard (MLP)	10	81	1.20%	28.61%	0.9961
	Standard (MLP)	14	113	1.21%	26.40%	0.9960
	Standard (MLP)	18	145	1.13%	39.71%	0.9956
	Standard (MLP)	25	201	1.27%	31.83%	0.9947
	Knowledge based (KBNN)	b5z6	114	0.90%	11.50%	0.9986
	Knowledge based (KBNN)	b6z8	163	0.89%	11.19%	0.9986
500	Standard (MLP)	7	57	0.94%	13.14%	0.9983
	Standard (MLP)	10	81	0.86%	10.82%	0.9987
	Standard (MLP)	14	113	0.83%	10.91%	0.9986
	Standard (MLP)	18	145	0.84%	11.74%	0.9986
	Standard (MLP)	25	201	0.95%	12.53%	0.9979
	Knowledge based (KBNN)	b5z6	114	0.74%	10.69%	0.9990
	Knowledge based (KBNN)	b6z8	163	0.72%	11.76%	0.9990

Table 4.10: Model Accuracy Comparison Between Standard MLP and Knowledge Based Neural Network (KBNN) for MESFET Modeling Example with Test Data Set C. The Results Shown Are the Average of Three Different Trainings for Each Model.

Figure 4.11: Model accuracy comparison of KBNN and MLP in terms of average testing error for the MESFET example. (a) Test data set A (b) Test data set B. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).

Figure 4.12: Model accuracy comparison of KBNN and MLP in terms of worst case testing error for the MESFET example. (a) Test data set A (b) Test data set B. The curves are from models of various sizes and trainings with different initial weights. The advantage of KBNN over MLP is even more significant when less training data is available. KBNN is also much more reliable than MLP in the extrapolation region, i.e., in case (b).

test data set B in the extrapolation range when trained by insufficient data of only 100 samples. Again, the error for KBNN mostly concentrate in the vicinity of zero. MLP has some error distributed at much higher error level. Figure 4.16 shows the histograms of error of MLP and KBNN for the same test data set when trained by 300 samples.

Figure 4.13: An example of IV curves from (a) MLP and (b) KBNN for MESFET modeling example. Both models were trained with insufficient training data of only 100 samples. The 100 samples were generated by changing 6 FET parameters including gate width, length, channel thickness, doping density, V_G and V_D . KBNN is visibly better than standard MLP.

Figure 4.15: Histograms of testing error of (a) MLP and (b) KBNN for MESFET modeling example with test data set B, i.e., extrapolation error. Both models were trained by only 100 training samples. Since concentration of errors is closer to 0% for KBNN than that of MLP, KBNN shows better accuracy than MLP.

Figure 4.16: Histograms of testing error of (a) MLP and (b) KBNN for MESFET modeling example with test data set B, i.e., extrapolation error. Both models were trained by 300 training samples.

Chapter 5

Knowledge Based Hierarchical Framework for the Development of Library of Neural Models

5.1 Introduction

A new task, i.e., the development of library of microwave neural models, is addressed in this chapter. This is of practical significance, since the realistic power of many CAD tools depends upon the richness, speed and accuracy of their library models. For neural models, while the cost for individual model development has been made manageable, massively developing neural models for libraries requires massive data generation, and repeated model training. This is a highly intensive process, and practically very expensive using today's neural model technique.

Motivated by the concept of combining neural networks, a new hierarchical neural network approach is proposed for the development of library of microwave neural models [2] [152] [153]. In the approach, a distinctive set of base neural models is established. The basic microwave functional characteristics common to various models in a library are first extracted and incorporated into base neural models. Then

a hierarchical neural network is constructed for each model of the library with low level modules realized by base neural models. A high level neural module is trained to map the low level module solution to the final output of the microwave component model for each model in the library. Compared to standard neural model techniques, the proposed hierarchical neural network approach substantially reduces the cost of library development due to less data collection and shorter training time, and at the same time improves model reliability.

5.2 Problem Statement: Library Development

The objective is to develop libraries of passive and active microwave component models. Suppose a library consists of N_C microwave component models. For each model, say, the n th model in the library, the input and output parameters are represented by vectors $\mathbf{X}^n$ and $\mathbf{Y}^n$, respectively. The library development is to create models to represent the multidimensional nonlinear relationship of

$$\mathbf{Y}^n = \mathbf{Y}^n(\mathbf{X}^n) \quad (5.1)$$

for each value of $n, n = 1, 2, \dots, N_C$. We call the spaces spanned by $\mathbf{X}^n$ and $\mathbf{Y}^n$ as the $\mathbf{X}^n$ space and the $\mathbf{Y}^n$ space, respectively.

For example, to model a multi-conductor transmission line for use in designing high-speed VLSI interconnects, $\mathbf{Y}^n$ could represent self and mutual inductances of the coupled conductors. $\mathbf{X}^n$ could represent the physical/geometrical parameters of the transmission line such as conductor width, separation between coupled conductors, substrate height and dielectric constants. Many such neural models would be needed in order to cover a variety of transmission lines in a VLSI interconnect design, such as single-conductor line, dual strip lines, 3 conductor coupled lines etc., leading

to the need of a library of transmission line models, such as the stripline library of Figure 5.1. Using the standard neural model approach, e.g., multilayer perceptron structure (MLP), costly data collection and extensive model training have to be performed for each model in the library. The total cost for library development will be very high.

5.3 Base Models

In the proposed approach, we first develop a set of base models to capture the basic electrical or microwave characteristics common to the entire set of models in the library. For example, in a library of various multiconductor transmission line models, the self inductance of a conductor is one of the common characteristics needed for all the models in the library. Let $\mathbf{X}_B^j$ and $\mathbf{Y}_B^j$ be vectors representing the inputs and outputs of the j th base model, $j = 1, \dots, N_B$, where N_B is the total number of base models in the library. Let the two spaces spanned by $\mathbf{X}_B^j$ and $\mathbf{Y}_B^j$ be called the $\mathbf{X}_B^j$ space and the $\mathbf{Y}_B^j$ space, respectively. The j th base model, realized by a neural network, relates $\mathbf{X}_B^j$ and $\mathbf{Y}_B^j$ by

$$\mathbf{Y}_B^j = \mathbf{B}_j(\mathbf{X}_B^j, \mathbf{W}_j) \quad (5.2)$$

where $\mathbf{B}_j$ represents the j th base model and $\mathbf{W}_j$ is a vector including all the neural network weights of the j th base model.

The definition of the $\mathbf{X}_B^j$ (or $\mathbf{Y}_B^j$) spaces are done by choosing a form of space conversion between $\mathbf{X}_B^j$ and $\mathbf{X}^n$ (or, $\mathbf{Y}_B^j$ and $\mathbf{Y}^n$), and/or examining the common characteristics in the library. Examples of typical forms of space conversion are: same-space mapping, subspace mapping, or linear transformation. The space conversions used here are of the same concept as space mapping, by Bandler, Biernacki and Chen

Figure 5.1: A library of stripline models. The n th model in the library represents an n -conductor coupled stripline component.

5.4 Hierarchical Neural Network Structure

For each model in the library, a hierarchical neural network structure is defined as shown in Figure 5.2. The purpose of this structure is to construct an overall model from several modules so that library base relationship can be maximally reused for every model throughout the library. This structure consists of a high level neural module denoted as $\mathbf{H}^n$ and several low level neural modules denoted as $\mathbf{L}_i^n$, $i = 1, \dots, N_L^n$. The low level modules are realized by directly using the base models. Let index function $j = \phi^n(i)$ be defined such that base model $\mathbf{B}_j$ is selected as the i th low level neural module and

$$j = \phi^n(i) = \begin{cases} 1, & \text{for } 0 < i \leq N_{B_1}^n \\ 2, & \text{for } N_{B_1}^n < i \leq N_{B_1}^n + N_{B_2}^n \\ \dots, & \dots \\ N_B, & \text{for } \sum_{k=1}^{N_B-1} N_{B_k}^n < i \leq \sum_{k=1}^{N_B} N_{B_k}^n \end{cases} \quad (5.4)$$

where $N_{B_j}^n$ is the number of times $\mathbf{B}_j$ is reused in the low level of the n th library model. Let $\mathbf{u}$ and $\mathbf{z}$ be vectors representing the inputs and outputs of low level modules. Since the i th low level module of the n th library model is realized by the $\phi^n(i)$ th base model,

$$\mathbf{L}_i^n = \mathbf{B}_j, \quad j = \phi^n(i) \quad (5.5)$$

$$\mathbf{u}_i^n = \mathbf{X}_B^j, \quad j = \phi^n(i) \quad (5.6)$$

$$\mathbf{z}_i^n = \mathbf{Y}_B^j, \quad j = \phi^n(i) \quad (5.7)$$

where $\mathbf{u}_i^n$ and $\mathbf{z}_i^n$ represent the input and output vectors of low level module i in library model n . For each $\mathbf{L}_i^n$, i.e., the i th low level module, we define a structural knowledge hub $\mathbf{U}_i^n(\cdot)$, such that it extracts inputs only relevant to the $\phi^n(i)$ th base model out of $\mathbf{X}^n$ based on the particular configuration of the n th library component, i.e.,

$$\mathbf{u}_i^n = \mathbf{X}_B^j = \mathbf{U}_i^n(\mathbf{X}^n), \quad \text{where } j = \phi^n(i) \quad (5.8)$$

Figure 5.2: The proposed hierarchical neural network structure. X and Y represent the inputs and outputs of the overall network. L_i is the i th low level module with an associated i th knowledge hub $U_i(\cdot)$. u and z represent the inputs and outputs of low level modules. This structure can be used for each model in a library. For example, for the n th model, $Y = Y^n$, $H = H^n$, $z_i = z_i^n$, $L_i = L_i^n$, $u_i = u_i^n$, $U_i = U_i^n$ and $X = X^n$.

The low level neural modules produce $\mathbf{z}^n$ by recalling the trained base models in the library,

$$\mathbf{z}_i^n = \mathbf{L}_i^n(\mathbf{u}_i^n) = \mathbf{B}_{\phi(i)}(\mathbf{u}_i^n, \mathbf{W}_{\phi(i)}), \quad i = 1, 2, \dots, N_L^n \quad (5.9)$$

where $\mathbf{W}_{\phi(i)}$ are the weights of the $\phi(i)$ *th* base model, and $\phi(i) = \phi^n(i)$. Let vectors $\mathbf{U}^n$ and $\mathbf{Z}^n$ be defined by concatenating the $\mathbf{u}_i^n$ and the $\mathbf{z}_i^n$ for all i , $i = 1, 2, \dots, N_L^n$, respectively, i.e.,

$$\mathbf{U}^n = \begin{bmatrix} \mathbf{u}_1^n \\ \mathbf{u}_2^n \\ \vdots \\ \mathbf{u}_{N_L^n}^n \end{bmatrix}, \quad \mathbf{Z}^n = \begin{bmatrix} \mathbf{z}_1^n \\ \mathbf{z}_2^n \\ \vdots \\ \mathbf{z}_{N_L^n}^n \end{bmatrix}. \quad (5.10)$$

All the low level modules combined provides a map from the $\mathbf{U}^n$ space to the $\mathbf{Z}^n$ space. A high level module $\mathbf{H}^n$ is defined mapping the $\mathbf{Z}^n$ space to the $\mathbf{Y}^n$ space for each *n**th* model in the library. The high level module is realized by a neural network,

$$\mathbf{Y}^n = \mathbf{H}^n(\mathbf{Z}^n, \mathbf{V}^n) \quad (5.11)$$

where $\mathbf{V}^n$ includes all neural network weights for module $\mathbf{H}^n$. The relationship in Equation (5.11) is much easier to model than the original $\mathbf{Y}^n = \mathbf{Y}^n(\mathbf{X}^n)$ relationship since much information is already contained in the base models in the low level. For example, even a linear 2 layer perceptron for $\mathbf{H}^n$ might be sufficient to produce the final $\mathbf{Y}^n$. Consequently the amount of data needed to train $\mathbf{H}^n$ is much less than that for training standard MLP to learn original $\mathbf{Y}^n = \mathbf{Y}^n(\mathbf{X}^n)$.

Figure 5.3 shows a sequence of space mapping in the proposed hierarchical neural network structure. Suppose $(\mathbf{X}^{n,k}, \mathbf{D}^{n,k})$ are pairs of training samples for the *n**th* library model, where $k = 1, \dots, M^n$, and M^n is the total number of training samples. The $\mathbf{X}^{n,k}$ data are mapped to the $\mathbf{U}^n$ space through knowledge hubs, and then feed-forwarded through the low level modules (i.e., various reuse of base neural models) into the $\mathbf{Z}^n$ space. Consequently a new set of training samples, denoted by pairs

of $(\mathbf{Z}^{n,k}, \mathbf{D}^{n,k})$ is obtained, where $\mathbf{Z}^{n,k}$ is the vector constructed by concatenating $\mathbf{z}_i^{n,k} = \mathbf{L}_i^n(\mathbf{U}_i^n(\mathbf{X}^{n,k}))$, for all i , $i = 1, \dots, N_L^n$. The high level neural module $\mathbf{H}^n$ should be trained such that

$$\min_{\mathbf{V}^n} \sum_{k=1}^{M^n} \|\mathbf{H}^n(\mathbf{Z}^{n,k}, \mathbf{V}^n) - \mathbf{D}^{n,k}\|^2, \quad \text{for each } n, \quad n = 1, \dots, N_C \quad (5.12)$$

With a linear 2 layer perceptron neural network as the high level module, this optimization is simply a quadratic programming problem. In this case, any training method will easily and quickly lead to a globally optimal training solution. This is in contrast to standard MLP approach with the original nonlinear $\mathbf{Y}^n(\mathbf{X}^n)$ relationship, where training usually takes long time and might end at a local optimal solution of the neural network, further prolonging the training process.

Under the proposed method, the training of the high level module is the only training needed for each model in the library. No training is needed for the low level modules because all such modules are reuse of the same set of base models which were trained only once in the beginning of library development.

5.5 Algorithm for Overall Library Development

The overall library development is summarized in the following steps:

Step 1 : Define the input and output spaces of base models, i.e., $\mathbf{X}_B^j$ and $\mathbf{Y}_B^j$, for $j = 1, 2, \dots, N_B$, and extract basic characteristics from library, using microwave empirical knowledge if available.

Step 2 : Collect training data corresponding to each base model inputs and outputs, i.e, generate sample data $(\mathbf{X}_B^{j,k}, \mathbf{D}_B^{j,k})$ for the j th base model, where $k = 1, 2, \dots, M_B^j$, and $j = 1, 2, \dots, N_B$.

Remark : Training data for base models should be adequate in order to obtain reliable base models.

Step 3 : Construct and train base neural models incorporating the knowledge from Step 1. Specifically, solve the optimization problem of (5.3) to find $\mathbf{W}_j$ such that $\mathbf{B}_j(\mathbf{u}, \mathbf{W}_j)$ matches base model training data, for $j = 1, \dots, N_B$. Let $n = 1$.

Remark : Steps 1, 2 and 3 are done in the beginning of library development and are considered as the overhead effort for the library. The next several steps, i.e., Step 4 to Step 8, are the incremental effort for each component model in the library.

Step 4 : According to the base model input space definition in Step 1, setup the structural knowledge hubs $\mathbf{u}_i^n = \mathbf{U}_i^n(\mathbf{X}^n)$, which maps the model input space $\mathbf{X}^n$ into base model input space $\mathbf{X}_B^j$, where $j = \phi^n(i)$ as defined in Equation (5.4), and $i = 1, \dots, N_L^n$. This automatically sets up the low level modules for the n th model.

Step 5 : Collect training data corresponding to the model in the library, i.e., generate sample data $(\mathbf{X}^{n,k}, \mathbf{D}^{n,k})$ for the n th model, where $k = 1, 2, \dots, M^n$.

Remark : Only a small amount of training data is needed here under the proposed technique.

Step 6 : Map the $\mathbf{X}^{n,k}$ data into the $\mathbf{Z}^n$ space through knowledge hubs and low level modules following equations (5.8) and (5.9).

Step 7 : Train the high level neural module $\mathbf{H}^n$, i.e., solve the optimization problem of (5.12) to find $\mathbf{V}^n$ such that the outputs of high level module match training

data.

Remark : This training step is very easy and fast since the module $\mathbf{H}^n$ is very simple and in most cases, a simple linear two layer perceptron network. Therefore only a small and incremental effort is needed to obtain each model in the library. It also means that only a small amount of training data is needed.

Step 8 : If $n = N_C$, then stop, otherwise proceed to train the next library model by setting $n = n + 1$ and go to Step 4.

The algorithm described above permits the hierarchical neural models be developed systematically, and enables the library development process be maximally automated. A flow chart representation of this method is shown in Figure 5.4.

5.6 Discussions

Our formulation allows the standard MLP approach to library development as an extreme special case in our theory. To illustrate this case, consider each library model as a base model, and $N_B = N_C$. The base model input and output spaces are defined the same as the library model input and output spaces, i.e.,

$$\mathbf{X}_B^j = \mathbf{X}^n, \quad j = n, \quad \text{and} \quad n = 1, \dots, N_C \quad (5.13)$$

$$\mathbf{Y}_B^j = \mathbf{Y}^n, \quad j = n, \quad \text{and} \quad n = 1, \dots, N_C \quad (5.14)$$

There is only one low level module in each library model. The knowledge hub is simply a relay block passing $\mathbf{X}^n$ directly to the $\mathbf{U}^n$ space,

$$\mathbf{u}_1^n = \mathbf{U}_1^n(\mathbf{X}^n) = \mathbf{X}^n \quad (5.15)$$

Figure 5.4: The flow chart of the algorithm for overall library development.

The high level module $\mathbf{H}^n$ will also perform a relay from $\mathbf{Z}^n$ space to the $\mathbf{Y}^n$ space,

$$\mathbf{Y}^n = \mathbf{H}^n(\mathbf{Z}^n) = \mathbf{Z}^n \quad (5.16)$$

Therefore in the worst extreme case where basic characteristics common to various models in the same library are difficult to identify, our technique falls back to the standard MLP approach. However in many practical cases, models are grouped into a library due to certain common features. The proposed approach becomes very advantageous when a few base models can capture the common characteristics in a library of many models as demonstrated through the examples in the next chapter.

Chapter 6

Examples of Neural Model Library Development

6.1 Library of Stripline Models

Multiconductor transmission line models are essential for delay and crosstalk analysis in high-speed VLSI interconnect design [19]. EM simulation of transmission line responses is slow especially if it needs to be repetitively evaluated. Neural models, trained off-line from EM data, can be used online during VLSI interconnect design providing instant solutions of the original EM problem. For practical VLSI interconnect design, libraries of 1-conductor, 2-conductor, ..., N-conductor transmission line models are needed. A brute force approach is to train each library model separately, requiring massive data generation and training. Here we apply the proposed hierarchical approach to the development of a library of neural models for N-conductor striplines shown in Figure 6.1 for different values of N. In this example, the modeling of self and mutual inductances is presented for illustration. There are five models in the library, $n = 1, 2, 3, 4, 5$ as shown in Figure 5.1. And for each n th model, the number of conductors $N = n$. Table 6.1 defines the notations for input and

output parameters of stripline neural models and the effective range of their input parameters. The detailed list of input and output parameters of each model in the stripline library is shown in Table 6.2. Training and test data were obtained using LINPAR [117] simulator which is based on the method of moments.

Parameters	Notation	Range
the i th conductor width	W_i	$0.05mm \sim 0.25mm$
the separation between the i th and $i + 1$ th conductors	s_i	$0.1mm \sim 0.82mm$
conductor height above ground	h_c	$0.08mm \sim 0.25mm$
substrate height	h_s	$0.16mm \sim 0.5mm$
relative dielectric constant	ϵ_r	$2 \sim 10.2$
self inductance of the i th conductor	L_{ii}	N/A
mutual inductance between the i th and j th conductors	L_{ij}	N/A

Table 6.1: The Notations for Input and Output Parameters of Stripline Neural Models and the Effective Range of Their Input Parameters.

Base Model Selections: Two base models, B_1 for self inductance and B_2 for mutual inductance are defined. The inputs to the base models include physical /geometrical parameters such as conductor width (W), conductor height (h_c), substrate height (h_s), separation between conductors (s), and relative dielectric constant (ϵ_r). The outputs of B_1 and B_2 are self and mutual inductances, respectively. Since for any model in the library, shown in Figure 5.1, the relation between the self inductance of a single conductor (and the mutual inductance between two conductors) and the corresponding physical/geometrical parameters is always a useful partial solution to the modeling problem, these 2 base models do represent basic characteristics useful to all the 5 stripline models in the entire library. The stripline empirical formulas in [58] were adopted as functional knowledge incorporated into the KBNNs (Knowl-

Library Model index n	Model name	Neural Model inputs $\mathbf{X}^n$	Neural Model outputs $\mathbf{Y}^n$	Number of times each base model $(\mathbf{B}_1, \mathbf{B}_2)$ is used $N_{B_j}^n \times \mathbf{B}_j$
$n = 1$	1 conductor stripline model	$\mathcal{W} \ h_c \ h_s \ \epsilon_r$	L_{11}	$1 \times \mathbf{B}_1$
$n = 2$	2 conductor stripline model	$\mathcal{W}_1 \ \mathcal{W}_2 \ s \ h_c \ h_s \ \epsilon_r$	$L_{11} \ L_{12} \ L_{22}$	$2 \times \mathbf{B}_1, 1 \times \mathbf{B}_2$
$n = 3$	3 conductor stripline model	$\mathcal{W}_1 \ \mathcal{W}_2 \ \mathcal{W}_3 \ s_1 \ s_2 \ h_c \ h_s \ \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{22}, L_{23}, L_{33}$	$3 \times \mathbf{B}_1, 3 \times \mathbf{B}_2$
$n = 4$	4 conductor stripline model	$\mathcal{W}_1 \ \mathcal{W}_2 \ \mathcal{W}_3 \ \mathcal{W}_4 \ s_1 \ s_2 \ s_3 \ h_c \ h_s \ \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{22}, L_{23}, L_{24}, L_{33}, L_{34}, L_{44}$	$4 \times \mathbf{B}_1, 6 \times \mathbf{B}_2$
$n = 5$	5 conductor stripline model	$\mathcal{W}_1 \ \mathcal{W}_2 \ \mathcal{W}_3 \ \mathcal{W}_4 \ \mathcal{W}_5 \ s_1 \ s_2 \ s_3 \ s_4 \ h_c \ h_s \ \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{15}, L_{22}, L_{23}, L_{24}, L_{25}, L_{33}, L_{34}, L_{35}, L_{44}, L_{45}, L_{55}$	$5 \times \mathbf{B}_1, 10 \times \mathbf{B}_2$

Table 6.2: Stripline Library Models.

edge Based Neural Networks), which are the realizations of the base models $\mathbf{B}_1$ and $\mathbf{B}_2$. The KBNN structural parameters are represented by number of boundary and knowledge neurons, e.g., $b2z3$ representing 2 boundary and 3 knowledge neurons. The base models $\mathbf{B}_1$ and $\mathbf{B}_2$ are trained to an average testing accuracy of 0.39% and 0.16% respectively, as shown in Table 6.3. Linear transformation is used as the form of space mapping between $\mathbf{X}_B^j$ (of Table 6.3) and $\mathbf{X}^n$ (of Table 6.2). Subspace mapping is used between $\mathbf{Y}_B^j$ (of Table 6.3) and $\mathbf{Y}^n$ (of Table 6.2). The number of times base models $\mathbf{B}_1$ and $\mathbf{B}_2$ are reused in each library model, i.e., $N_{B_j}^n$, is shown in Table 6.2.

Example of Library Model: $n=1$: For $n = 1$, the library model is for single conductor transmission line, and is directly the base model $\mathbf{B}_1$. Therefore, $N_{B_1}^n = 1$,

Figure 6.2: The hierarchical neural model for the 3rd model in the stripline library, i.e., $n = 3$.

mutual inductances from the 6 low level modules) and 6 outputs (i.e., the final and refined predictions of self and mutual inductances of the overall 3-conductor stripline model) and with linear functions in all output neurons. This is a linear combination of low level neural modules taking advantage of modular neural network concept without any gating function. Each low level neural module provides a portion of the inductance prediction contributing to the overall inductance prediction at the high level neural module. Only a small amount of training data (15 samples) is needed to train this high level module of 3-conductor stripline model since the preliminary relationships of the model have already been captured in the base models. However, with the conventional MLP neural model (8, 12 and 16 hidden neurons), 500 samples are needed to achieve a model of similar accuracy, shown in Table 6.5.

No. of training samples	MLP (8-8-6)	MLP (8-12-6)	MLP (8-16-6)	Proposed model
15	15.40%	14.25%	14.17%	0.52%
25	10.61%	9.66%	9.96%	0.48%
50	4.01%	1.79%	5.30%	0.41%
100	1.36%	0.96%	1.80%	0.39%
300	0.87%	0.83%	0.86%	0.38%
500	0.84%	0.73%	0.79%	0.39%

Table 6.5: Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for 3-Conductor Stripline Model.

The tendency of library model accuracy versus the amount of training data is plotted in Figure 6.3. As available training data becomes less and less, the error of standard MLP grows quickly but the proposed hierarchical library model remains reasonable and reliable.

All Library Models: All library models, $n = 2, 3, 4, \dots$ in the library, can be developed systematically in a similar way as model #3. It should be noted that

Figure 6.3: Model accuracy comparison (average error on test data) between standard MLP and the proposed model for 3-Conductor stripline model.

efforts in developing those additional library models are small and incremental, since only few training data is needed, and only the high-level neural module $\mathbf{H}^n$ needs to be trained for each n .

Overall Library Accuracy and Development Cost: A Comparison Using standard MLP for each library model, the total amount of training data needed for the library is 2764 samples, and using the proposed approach the amount is only 649 (including 564 samples for base models, and 85 samples for subsequent library models) as shown in Table 6.6. The total training time for all library models using standard MLP approach is 2 hours and 10 minutes on Sun Ultra 1 Workstation, for such an illustrative library example. Using the proposed approach, the total training time is only 12 minutes.

6.2 Library of Microstrip Models

In this example, a library of neural models for N -conductor lossless microstrip lines is developed, $N = 1, 2, 3, \dots, 5$, i.e., a library of 5 models as shown in Figure 6.4. Figure 6.5 shows the details of a typical microstrip line from the library with the physical/geometrical parameters of the model defined. In this library, we model the self and mutual inductance, and capacitance as neural model outputs. All conductors have equal width, which is a reasonable assumption in many situations of signal integrity analysis and design of VLSI interconnects. The notations of parameters and parameter ranges of library neural models are defined similarly as those in Table 6.1.

Table 6.7 shows the detailed list of input and output parameters of each model in the microstrip library. Training and test data again were obtained using LINPAR [117] simulator which is based on the method of moments.

Figure 6.4: The microstrip library. The n th model in the library represents an n -conductor coupled microstrip model.

Figure 6.5: Details of a typical N-conductor microstrip component showing the physical and geometrical parameters.

Library model index	Stripline model name	Number of training samples needed (and corresponding model accuracy)	
		standard MLP	Proposed model
Overhead for base models		0	$264^1 + 300^2$
$n = 1$	1 conductor stripline model	264, (0.42%)	0, (0.39%)
$n = 2$	2 conductor stripline model	400, (0.75%)	10, (0.56%)
$n = 2$	3 conductor stripline model	500, (0.73%)	15, (0.52%)
$n = 4$	4 conductor stripline model	700, (0.78%)	25, (0.74%)
$n = 5$	5 conductor stripline model	900, (0.99%)	35, (0.63%)
	stripline library	Total = 2764	Total = 649

Table 6.6: Comparison of Number of Training Samples Needed and Library Model Accuracy for Stripline Library When Developed by Standard MLP and the Proposed Hierarchical Neural Network Structure, respectively. ¹ for Base Model $\mathbf{B}_1$ Training and ² for Base Model $\mathbf{B}_2$ Training.

Base Model Selections: In this library, the most important basic characteristics of all models can be the relationship between electrical parameters of self inductance and capacitance of a conductor (and mutual inductance and capacitance between 2 conductors) and the microstrip physical/geometrical parameters. Four base models, $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$, and $\mathbf{B}_4$ are created to represent these characteristics, respectively. The inputs and outputs of the base models are defined in Table 6.8.

There exist empirical formulas for these characteristics in [58], approximating the relation for the self and mutual inductance and capacitance of single or dual microstrip lines. The base neural models are constructed incorporating such functional

Library Model index n	Model name	Neural Model inputs $\mathbf{X}^n$	Neural Model outputs $\mathbf{Y}^n$	Number of times each base model ($\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4$) is used $N_{B_j}^n \times \mathbf{B}_j$
$n = 1$	1 conductor microstrip model	$\mathcal{W} h_s \epsilon_r$	L_{11}, C_{11}	$1 \times \mathbf{B}_1, 1 \times \mathbf{B}_3$
$n = 2$	2 conductor microstrip model	$\mathcal{W} s h_s \epsilon_r$	$L_{11} L_{12} L_{22},$ $C_{11} C_{12} C_{22}$	$1 \times \mathbf{B}_1, 1 \times \mathbf{B}_2$ $1 \times \mathbf{B}_3, 1 \times \mathbf{B}_4$
$n = 3$	3 conductor microstrip model	$\mathcal{W} s_1 s_2$ $h_s \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{22},$ $L_{23}, L_{33}, C_{11}, C_{12},$ $C_{13}, C_{22}, C_{23}, C_{33}$	$1 \times \mathbf{B}_1, 3 \times \mathbf{B}_2$ $1 \times \mathbf{B}_3, 3 \times \mathbf{B}_4$
$n = 4$	4 conductor microstrip model	$\mathcal{W} s_1 s_2$ $s_3 h_s \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{22},$ $L_{23}, L_{24}, L_{33}, L_{34}, L_{44},$ $C_{11}, C_{12}, C_{13}, C_{14}, C_{22},$ $C_{23}, C_{24}, C_{33}, C_{34}, C_{44}$	$1 \times \mathbf{B}_1, 6 \times \mathbf{B}_2$ $1 \times \mathbf{B}_3, 6 \times \mathbf{B}_4$
$n = 5$	5 conductor microstrip model	$\mathcal{W} s_1 s_2 s_3$ $s_4 h_s \epsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{15},$ $L_{22}, L_{23}, L_{24}, L_{25}, L_{33},$ $L_{34}, L_{35}, L_{44}, L_{45}, L_{55},$ $C_{11}, C_{12}, C_{13}, C_{14}, C_{15},$ $C_{22}, C_{23}, C_{24}, C_{25}, C_{33},$ $C_{34}, C_{35}, C_{44}, C_{45}, C_{55}$	$1 \times \mathbf{B}_1, 10 \times \mathbf{B}_2$ $1 \times \mathbf{B}_3, 10 \times \mathbf{B}_4$

Table 6.7: Microstrip Library Models.

knowledge through the Knowledge Based Neural Network (KBNN) structure combining the empirical information with the learning power of neural networks. The base models $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ and $\mathbf{B}_4$ are trained to an average testing accuracy of 0.16%, 0.13%, 0.18% and 0.31%, respectively, as shown in Table 6.8. Linear transformation is used as the form of space mapping between $\mathbf{X}_B^j$ (of Table 6.8) and $\mathbf{X}^n$ (of Table 6.7). Subspace mapping is used between $\mathbf{Y}_B^j$ (of Table 6.8) and $\mathbf{Y}^n$ (of Table 6.7). The number of times base models are reused in each library model is shown in Table 6.7.

Example of Library Model: $n = 1$: For $n = 1$, the library model is constructed

Base model index j	Base model symbol $\mathbf{B}_j$	Base model inputs $\mathbf{X}_B^j$	Base model outputs $\mathbf{Y}_B^j$	Base model structure (KBNN)	Model accuracy (average error)
1	$\mathbf{B}_1$	$\mathcal{W} h_s \epsilon_r$	self inductance L_s	$b2z3$	0.16%
2	$\mathbf{B}_2$	$\mathcal{W} s h_s \epsilon_r$	mutual inductance L_m	$b2z3$	0.13%
3	$\mathbf{B}_3$	$\mathcal{W} h_s \epsilon_r$	self capacitance C_s	$b1z2$	0.18%
4	$\mathbf{B}_4$	$\mathcal{W} s h_s \epsilon_r$	mutual capacitance C_m	$b4z6$	0.31%

Table 6.8: Base Models for Microstrip Library.

simply by putting base models $\mathbf{B}_1$ and $\mathbf{B}_3$ together without any further training.

Therefore, $\mathbf{L}_1^n = \mathbf{B}_1$, $\mathbf{L}_2^n = \mathbf{B}_3$. $\mathbf{H}^n$ relays from the $\mathbf{Z}^n$ space to the $\mathbf{Y}^n$ space.

Example of Library Model: $n = 3$: For library model $n = 3$, we reuse the base models as the lower level neural modules as shown in Figure 6.6. There are 8 low level modules and 4 are for inductance prediction and 4 for capacitance prediction. The knowledge hubs for the model are defined in Table 6.9. The high-level neural module

Low level modules $\mathbf{L}_i^n$	Inputs to module/Knowledge hub $\mathbf{u}_i^n = \mathbf{U}_i^n(\mathbf{X}^n)$	Index function $\phi^n(i) = j$	Base model used $\mathbf{B}_{\phi(i)}$
$\mathbf{L}_1^3$	$\mathbf{u}_1^3 = \mathbf{U}_1^3(\mathbf{X}^3) = [\mathcal{W} h_s \epsilon_r]$	$\phi^3(1) = 1$	$\mathbf{B}_1$
$\mathbf{L}_2^3$	$\mathbf{u}_2^3 = \mathbf{U}_2^3(\mathbf{X}^3) = [\mathcal{W} s_1 h_s \epsilon_r]$	$\phi^3(2) = 2$	$\mathbf{B}_2$
$\mathbf{L}_3^3$	$\mathbf{u}_3^3 = \mathbf{U}_3^3(\mathbf{X}^3) = [\mathcal{W} s_2 h_s \epsilon_r]$	$\phi^3(3) = 2$	$\mathbf{B}_2$
$\mathbf{L}_4^3$	$\mathbf{u}_4^3 = \mathbf{U}_4^3(\mathbf{X}^3) = [\mathcal{W} s_1 + s_2 h_s \epsilon_r]$	$\phi^3(4) = 2$	$\mathbf{B}_2$
$\mathbf{L}_5^3$	$\mathbf{u}_5^3 = \mathbf{U}_5^3(\mathbf{X}^3) = [\mathcal{W} h_s \epsilon_r]$	$\phi^3(5) = 3$	$\mathbf{B}_3$
$\mathbf{L}_6^3$	$\mathbf{u}_6^3 = \mathbf{U}_6^3(\mathbf{X}^3) = [\mathcal{W} s_1 h_s \epsilon_r]$	$\phi^3(6) = 4$	$\mathbf{B}_4$
$\mathbf{L}_7^3$	$\mathbf{u}_7^3 = \mathbf{U}_7^3(\mathbf{X}^3) = [\mathcal{W} s_2 h_s \epsilon_r]$	$\phi^3(7) = 4$	$\mathbf{B}_4$
$\mathbf{L}_8^3$	$\mathbf{u}_8^3 = \mathbf{U}_8^3(\mathbf{X}^3) = [\mathcal{W} s_1 + s_2 h_s \epsilon_r]$	$\phi^3(8) = 4$	$\mathbf{B}_4$

Table 6.9: Low Level Modules and Structural Knowledge Hubs for 3-Conductor Microstrip, i.e., Library Model $n = 3$.

$\mathbf{H}^3$ is realized by a non-fully connected 2-layer perceptron with 8 inputs (i.e., preliminary inductance and capacitance prediction from low level) and 12 outputs (i.e.,

Figure 6.6: The hierarchical neural model for the 3rd model in the microstrip library, i.e., $n = 3$.

final inductance and capacitance of the overall library model). This example takes advantage of the modular neural network feature such that the overall library model is a linear combination with gating functions connecting 4 inductance (4 capacitance) predictions from low level modules to 6 inductance (6 capacitance) outputs at the high level. Only a small amount of training data (15 samples) is needed to train this 3-conductor microstrip model since the preliminary relationships of the model have already been captured in the base models. However, with the conventional MLP neural model (25, 30, 35 hidden neurons), 300 samples are needed to achieve a model of similar accuracy, shown in Table 6.10.

No. of training samples	MLP (5-25-12)	MLP (5-30-12)	MLP (5-35-12)	Proposed model
15	6.16%	6.63%	7.10%	0.42%
25	4.01%	4.57%	5.28%	0.40%
50	1.34%	2.33%	2.87%	0.38%
100	1.45%	1.67%	1.93%	0.38%
300	0.53%	0.43%	0.42%	0.35%

Table 6.10: Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for 3-Conductor Microstrip Model.

Figure 6.7 shows the tendency of model accuracy as the amount of available training data is reduced. The error for the proposed hierarchical model goes up only slowly whereas the error for the standard MLP models grows very quickly as the amount of available training data is reduced.

Overall Library Accuracy and Development Cost: A Comparison All library models, $n = 2, 3, 4, \dots$ in the library, can be developed systematically in a similar way as model #3. The total amount of training data needed by standard MLP for the library is 1700 samples collected through electromagnetic simulations. The total

Figure 6.7: Model accuracy comparison (average error on test data) between standard MLP and the proposed model for 3-Conductor microstrip model.

amount of training data required by the proposed approach is only 550 (including 400 samples for base models, and 150 samples for all subsequent library models) as shown in Table 6.11. Using standard MLP for each library model, the total training time for all library models is 16.7 hours on Sun Ultra 1 workstation, and using the proposed approach the total training time is only 19.2 minutes. Figure 6.8 shows the tendency of amount of required training data versus the size of library. Figure 6.9 shows the tendency of total training time required versus the size of the library.

Library model index	MESFET model name	Number of training samples needed (and corresponding model accuracy)	
		standard MLP	Proposed model
Overhead for base models		0	100 ¹ , (0.17%, 22sec) 300 ² , (0.22%, 14min)
$n = 1$	1 conductor microstrip model	100 , (0.45%, 2.5min)	0 , (0.17%, 0sec)
$n = 2$	2 conductor microstrip model	300 , (0.42%, 14.8min)	10 , (0.37%, 2sec)
$n = 3$	3 conductor microstrip model	300 , (0.42%, 73.2min)	15 , (0.42%, 4.8sec)
$n = 4$	4 conductor microstrip model	500 , (0.45%, 495min)	50 , (0.56%, 35sec)
$n = 5$	5 conductor microstrip model	500 , (0.37%, 416min)	75 , (0.51%, 4.1min)
	microstrip library	Total = 1700 , (16.7hrs)	Total = 550 , (19.2min)

Table 6.11: Comparison of Number of Training Samples Needed and Training Time Used for Microstrip Library When Developed by Standard MLP and the Proposed Neural Network Structure, respectively. ¹ for Base Models **B**₁ and **B**₃ Training and ² for Base Models **B**₂ and **B**₄ Training.

Figure 6.8: The total amount of training data required for developing neural model library of microstrip lines versus the total number of models in the library. The overhead data of 400 required for the proposed approach due to base model training is represented by the nonzero value when $N_C = 0$. But the incremental amount of data needed for training each new model in the library is very small under the proposed approach. As the total number of models in the library increases, the total amount of training data required by the proposed approach becomes substantially less than that required by the standard MLP approach.

Figure 6.9: The total training time for developing neural model library of microstrip lines versus the total number of models in the library. The overhead training time of 14 minutes for the proposed approach due to base model training is represented by the nonzero value at $N_C = 0$. But the incremental training time for adding each new model to the library is very small under the proposed approach. As the total number of models in the library increases, the total training time required by the proposed approach becomes substantially less than that of the standard MLP approach.

6.3 Library of MESFET Models

The drive for first-pass-success in designing active microwave circuits leads to the need of physics-based transistor device models which give more accurate predictions of device behavior than empirical or equivalent models. However such physics based models are too slow when used repetitively in circuit design. Neural models, trained from physics-based FET data, can be used to instantly predict physics-level device behavior for repetitive use during simulation and optimization [3]. Here we demonstrate the proposed hierarchical approach for a set of FET device models. For this specific example, we assume that the library consists of bias dependent S-parameter models for MESFETs with different gate length values. A typical MESFET model in the library represents the intrinsic FET structure following Khatibzadeh and Trew [151], as shown in Figure 6.10. The library contains 10 models, $n = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

Figure 6.10: Physics-based intrinsic MESFET device model following [151].

and each model corresponds to a FET with a fixed gate length of $0.35\mu m$, $0.4\mu m$, $0.45\mu m$, $0.5\mu m$, $0.55\mu m$, $0.6\mu m$, $0.65\mu m$, $0.7\mu m$, $0.75\mu m$, $0.8\mu m$, respectively. The library neural models are trained to predict the scattering parameters from physical and electrical parameters of the device. $\mathbf{Y}^n$ includes real and imaginary parts of S_{11} , S_{12} , S_{21} and S_{22} . $\mathbf{X}^n$ includes frequency(ω), channel thickness (a), gate bias voltage (V_g), and drain bias voltage (V_d). Training and test data were obtained by using OSA90 [116] with Khatibzadeh and Trew models [151]. In this library, all transistors have assumed gate width of $1mm$. The model parameters and their ranges are shown in Table 6.12.

Parameters	Notation	Range
gate length	l	$0.35 \sim 0.80\mu m$
gate width	$\mathcal{W}_c$	$1mm$
channel thickness	a	$0.28 \sim 0.42\mu m$
frequency	ω	$3.14 \sim 157 \times 10^9 rad/sec$
gate bias voltage	V_g	$-5 \sim 0V$
drain bias voltage	V_d	$0.5 \sim 4.5V$

Table 6.12: Effective Ranges of Neural Model Input Parameters for MESFET Library.

Base Model Selections: In this library, the relationships between the real and imaginary parts of the scattering parameters, namely S_{11} , S_{12} , S_{21} and S_{22} , and model input parameters ω , V_d , V_g and a , are taken as the common characteristics required for all transistor models. To represent these common characteristics, eight base models, $\mathbf{B}_1$, $\mathbf{B}_2$, ..., and $\mathbf{B}_8$ are defined corresponding to four scattering parameters of 2 MESFETs, one with small gate length ($l = 0.40\mu m$) and another with large gate length ($l = 0.80\mu m$) as shown in Table 6.13. Same-space mapping is used between $\mathbf{X}_B^j$ (of Table 6.13) and $\mathbf{X}^n$, the inputs to both base models being the same as those for the other transistor models in the library. Subspace mapping is used between $\mathbf{Y}_B^j$

(of Table 6.13) and $\mathbf{Y}^n$. The outputs of the base models are the real and imaginary of individual S-parameters of the transistor. In this example, we demonstrate that conventional MLP structure can also be used to construct the base models, with testing accuracy shown in Table 6.13.

Base model index j	Base model symbol $\mathbf{B}_j$	Base model inputs $\mathbf{X}_B^j$	Base model outputs $\mathbf{Y}_B^j$	Base model structure (MLP)	Model accuracy (average error)
1	$\mathbf{B}_1$	$a \ \omega \ V_g \ V_d$	S_{11} of $l = 0.4\mu m$	4 – 60 – 2	0.17%
2	$\mathbf{B}_2$	$a \ \omega \ V_g \ V_d$	S_{12} of $l = 0.4\mu m$	4 – 60 – 2	0.20%
3	$\mathbf{B}_3$	$a \ \omega \ V_g \ V_d$	S_{21} of $l = 0.4\mu m$	4 – 60 – 2	0.21%
4	$\mathbf{B}_4$	$a \ \omega \ V_g \ V_d$	S_{22} of $l = 0.4\mu m$	4 – 80 – 2	0.18%
5	$\mathbf{B}_5$	$a \ \omega \ V_g \ V_d$	S_{11} of $l = 0.8\mu m$	4 – 80 – 2	0.28%
6	$\mathbf{B}_6$	$a \ \omega \ V_g \ V_d$	S_{12} of $l = 0.8\mu m$	4 – 80 – 2	0.38%
7	$\mathbf{B}_7$	$a \ \omega \ V_g \ V_d$	S_{21} of $l = 0.8\mu m$	4 – 80 – 2	0.36%
8	$\mathbf{B}_8$	$a \ \omega \ V_g \ V_d$	S_{22} of $l = 0.8\mu m$	4 – 80 – 2	0.29%

Table 6.13: Base Models for MESFET Library.

Example of Library Model: $n = 2$ For $n = 2$, the library model is constructed by four base models, $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$ and $\mathbf{B}_4$, directly, without any further training, i.e., $\mathbf{L}_1^2 = \mathbf{B}_1$, $\mathbf{L}_2^2 = \mathbf{B}_2$, $\mathbf{L}_3^2 = \mathbf{B}_3$, $\mathbf{L}_4^2 = \mathbf{B}_4$, $\mathbf{H}^2(\mathbf{Z}^2) = \mathbf{Z}^2$.

Example of Library Model: $n = 5$ For library model $n = 5$, the input and output definition of the model is the same as that of the base models. The difference is that the gate length is equal to $0.55\mu m$. The overall model structure is shown in Figure 6.11. There are eight low level modules, i.e., $N_L^5 = 8$, and $\mathbf{L}_1^5 = \mathbf{B}_1$, $\mathbf{L}_2^5 = \mathbf{B}_2$, ..., $\mathbf{L}_8^5 = \mathbf{B}_8$. Base models are used in the low level neural modules to predict the S-parameter pattern for different model inputs. Since model input space is exactly the same as that of base models, knowledge hubs simply perform relay operations, i.e., $\mathbf{u}_i^5 = \mathbf{U}_i^5(\mathbf{X}_5) = \mathbf{X}^5$, $i = 1, 2, \dots, 8$. The high level neural module $\mathbf{H}^5$ is a 2-layer

Figure 6.11: The hierarchical neural model for FET library model #5, i.e., $n = 5$.

perceptron with 16 inputs and 8 outputs (real and imaginary parts of S_{11} , S_{22} , S_{12} and S_{21}). Out of the 16 inputs, 8 inputs correspond to the predictions from base models B_1 , B_2 , B_3 , and B_4 , while the other 8 inputs correspond to the predictions from base models B_5 , B_6 , B_7 , and B_8 . This example takes advantage of modular network concept without any gating function. Table 6.14 and Figure 6.12 show the comparison of model accuracy when this transistor is modeled by standard MLP (with 60, 80, and 100 hidden neurons) and the proposed model.

No. of training samples	MLP (4-60-8)	MLP (4-80-8)	MLP (4-100-8)	Proposed model
25	13.97%	15.15%	14.78%	2.14%
50	4.51%	4.30%	4.97%	0.99%
100	2.29%	2.25%	2.57%	0.87%
150	1.71%	1.57%	1.62%	0.82%
200	1.46%	1.35%	1.37%	0.79%
300	0.96%	0.83%	0.94%	0.74%

Table 6.14: Model Accuracy Comparison (average error on test data) Between Standard MLP and the Proposed Model for Library Model, $n = 5$, of MESFET Library.

Overall Library Accuracy and Development Cost Comparison Model $n = 10$ can be developed similarly as model $n = 2$. All other library models, $n = 1, 3, 4, 6, 7, 8$, and 9 , can be developed easily in a similar fashion as library model #5. Using the proposed library approach for each library model, the training time and training data required are much less as compared to the standard MLP approach as shown in Table 6.15.

Figure 6.12: Model accuracy comparison (average error on test data) between standard MLP and the proposed model for the MESFET library model, $n = 5$, whose gate length equals $0.55 \mu m$.

Library model index	MESFET model name	Number of training samples needed (and corresponding model accuracy)	
		standard MLP	Proposed model
Overhead for base models		0	300¹ , (0.23%, 4.22hrs) 300² , (0.33%, 4.54hrs)
$n = 1$	MESFET $l = 0.35 \mu m$	300 , (0.81%.2.52hrs)	50 , (0.70%, 1.4min)
$n = 2$	MESFET $l = 0.40 \mu m$	300 , (0.88%.2.64hrs)	0 , (0.23%, 0min)
$n = 3$	MESFET $l = 0.45 \mu m$	300 , (0.86%.2.48hrs)	50 , (0.66%, 1.4min)
$n = 4$	MESFET $l = 0.50 \mu m$	300 , (0.88%.2.32hrs)	50 , (0.71%, 1.4min)
$n = 5$	MESFET $l = 0.55 \mu m$	300 , (0.86%.2.66hrs)	50 , (0.99%, 1.4min)
$n = 6$	MESFET $l = 0.60 \mu m$	300 , (0.89%.2.18hrs)	50 , (1.22%, 1.4min)
$n = 7$	MESFET $l = 0.65 \mu m$	300 , (0.82%.2.45hrs)	50 , (1.08%, 1.4min)
$n = 8$	MESFET $l = 0.70 \mu m$	300 , (0.87%.2.58hrs)	50 , (1.05%, 1.4min)
$n = 9$	MESFET $l = 0.75 \mu m$	300 , (0.79%.2.50hrs)	50 , (0.77%, 1.4min)
$n = 10$	MESFET $l = 0.80 \mu m$	300 , (0.88%.2.78hrs)	0 , (0.33%, 0min)
	MESFETlibrary	Total = 3000 , (25.11hrs)	Total = 1000 , (8.95hrs)

Table 6.15: Comparison of Number of Training Samples Needed and Training Time Used for MESFET Library When Developed by Standard MLP and the Proposed Neural Network Structure, respectively. ¹ for Base Models **B**₁, **B**₂, **B**₃ and **B**₄ Training and ² for Base Models **B**₅, **B**₆, **B**₇ and **B**₈ Training.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis addressed efficient neural modeling methods for active and passive microwave component models and their library development. A new concept advocating the use of electrical engineering knowledge in the development of neural network models for microwave applications is introduced for the first time in the literature. A knowledge based neural network (KBNN) has been proposed combining microwave empirical experience with the power of learning of neural networks. A new error backpropagation training scheme for the KBNN structure utilizing gradient based l_2 optimization has been developed. For the examples presented in this thesis, the model testing errors from KBNN are less than those from MLP. The advantage of KBNN is even more significant when training data is insufficient. Reductions in the cost of model development through reduced need of generating large amount of training data and more efficient training algorithm have been demonstrated.

A new research task, i.e., development of library of microwave neural models, is also introduced for the first time in the literature. A new hierarchical neural model approach has been developed exploiting the inherent base relations between library

models and incorporating both functional and structural knowledge. A systematic training procedure was developed in this approach. The proposed approach can be applied to any microwave neural model library development in which basic electrical/microwave characteristics common to the library exists. The efficiency of the proposed approach increases when the library size increases, i.e., when a small set of base models can be extracted to represent basic information of a large number of library models. A significant cost reduction of neural model library development has been achieved, due to faster training and reduced need for data generation.

To accelerate the practical use of the techniques proposed in this thesis, an efficient implementation is necessary. An object oriented computer software package, **POWERNET**, has been developed for neural network training and other utilities. A novel neural network structure description has been developed and adopted in this software package, allowing arbitrary connections of neurons and user-defined activation functions for neurons. This novel neural network structure description and the components of the software package are described in Appendix A.

An overview of neural network applications in microwave area, including application fields, neural network structures and training algorithms, has been presented for an in-depth understanding of the state-of-the-art of the field. Signal integrity oriented optimization of high speed VLSI interconnects utilizing neural networks has also been summarized. The neural models can learn and predict component behaviors originally seen in detailed physics/EM models, and predict such behavior much faster than original models. This thesis work addressed important issues that are specifically crucial in developing microwave neural network models, i.e., cost of model development due to the nature of microwave data generation and the reliability of neural networks in modeling continuous multidimensional microwave problems. This

work is significant for the growing use of neural networks as economical and accurate models in microwave design. It will also have an impact on statistical analysis and design of microwave circuits.

7.2 Suggestions for Future Directions

Neural networks have a very promising future in the microwave design area. Benefits of applying neural network technology can be potentially achieved at all levels of microwave design from device, components, to circuits and systems, and from modeling, simulation, to optimization and synthesis. From the research point of view, future work in structures and training algorithms will shift from demonstration of basic significance of the neural network technology to addressing challenges from real microwave applications.

Modeling of complicated 3D EM problems is one of such work. In this case, the cost of generating training data by EM simulations is very high. How to develop reliable neural model with very small amount of data remains an important research. In many practical cases, training data from simulation/measurement will contain accidental but large errors due to convergence difficulties in simulators or equipment limits that may happen when data generation goes to extreme points in the parameter space. Existing training algorithms can be susceptible to such large errors, and consequently the neural model obtained is not reliable. Robust algorithms automatically dealing with such cases need to be developed, avoiding manual debugging for clues of model inaccuracy.

Filter design with full EM simulation is one of the important, yet difficult tasks for many engineers. Research on neural networks to help for such design is already under-

way. The highly nonlinear, and non-smooth relationship in such microwave models needs to be addressed by an effective neural network structure. Using standard structures, more neurons are typically needed for such cases leading to requirement of more training data, and higher accuracy is difficult to obtain. Another scenario leading to the same challenge occurs when models contain many variables, for example, many geometrical and physical parameters in a microwave model. Addressing these challenges will be an important direction in future research.

The potential of combining microwave and circuit information with neural networks, continues to motivate research leading to advanced knowledge based neural models. The proposed KBNN divides the original parameter space into several regions, with the original problem modeled by different empirical models in each region. This is a modular oriented decomposition. Different ways of incorporating functional knowledge should also be investigated, e.g., allowing empirical formulas to be continuously modified through a sub-neural-network. More advanced forms of knowledge beyond empirical formulas can be considered, such as a circuit simulation. Sometimes, the responses of a device or a circuit may be composed of both smooth portions and sharply varying portions, e.g., the frequency response of the S parameters of microwave filters is a smooth function with superimposed dips on it. A neural network structure based on this decomposition will improve learning of this kind of functional relationship which otherwise would be very difficult to learn using the normal neural network structure.

The hierarchical neural model concept can be extended even beyond the library boundary. In reality, microwave circuits and systems are by nature hierarchical. Neural networks can represent the hierarchical relationship from physical parameters to electrical parameters, from components to circuits. This decomposition is in line

with the structure of the original problem, which helps incorporate structural knowledge and improve model development efficiency. However, the challenge would be to develop a suitable formulation such that neural models at different levels of the hierarchy can be mathematically linked. Another challenge is to develop an effective and efficient training strategy for this structure regarding the sequence and iterations of changing the training parameters in different levels of the hierarchy.

Another significant milestone in this area would be to incorporate the microwave oriented features and techniques of the neural technology into readily usable software tools. These tools would enable more microwave engineers to quickly benefit from this technology, and their feedback could further stimulate advanced research in the area. Neural networks, with their unparalleled speed advantage, and their ability to learn and generalize wide variety of problems, promise to be one of the most powerful vehicles helping microwave design today and tomorrow.

Appendix A

A General Neural Network Structure Description and POWERNET Software Package

For the conventional neural network structures, a structure can be identified by a name, such as MLP, RBF or Wavelet Networks, and a set of numbers, such as number of layers and number of neurons for each layer. The activation functions for neurons are the standard activation functions which are independent of application examples. One self contained program would be enough to deal with various applications. In this thesis, a Knowledge Based Neural Network (KBNN) is proposed, which can incorporate the problem dependent knowledge into neural network structure. This essentially means that a program, which can be adapted easily when used for different applications, is needed. Along with this thesis work, a software package, **POWERNET**, has been developed. The examples in this thesis were trained using this software package. The **POWERNET** package was developed with object oriented concept and coded in *C++* on Unix. It is aimed for arbitrary neural networks with arbitrary inter-neuron connections and any user provided functions for neuron

activation functions. KBNN is just a special case. A novel and general neural network structure description was created to facilitate such capability of **POWERNET**. This description is adopted in *NeuroModeler* [155].

In this appendix, the general neural network structure description was first described and illustrated. The components of **POWERNET** software package are presented in the following section, together with its usage.

A.1 General Neural Network Structure Description

The general neural network structure description is saved in an ASCII text file, named as *.struc, which serves as the input file to **POWERNET**. The format of this *.struc file is shown in Table A.1, where the lines started with % are for comment only.

This description starts with a list of input and output neuron labels together with the total number of neurons in the structure. These are followed by the description of individual neurons. For each neuron, the interconnections between this neuron and other neurons are represented by a list of neurons, called FromNeurons whose outputs are the inputs of this neuron, and another list of neurons, called ToNeurons who take this neuron's output as one of their inputs. Each neuron is also described by their activation function name, the number of parameters in the function and a list of these parameters. At the end of structure description, a list of all neuron labels is given to describe the processing sequence of neurons during feedforward stage. This is necessary to release the requirement that the neuron description be presented in the same sequence as it should be processed.

```

NumInputNeurons InputNeuron1_Label InputNeuron2_Label ....

NumOutputNeurons OutputNeuron1_Label OutputNeuron2_Label ....

NumTotalNeurons

% Neuron#1

Neuron1_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ....

NumOutConnections ToNeuron1_Label ToNeuron2_Label ....

ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ....

% Neuron#2

Neuron2_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ....

NumOutConnections ToNeuron1_Label ToNeuron2_Label ....

ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ....

.....

% the last neuron: neuron #N

NeuronN_Label NumInConnections FromNeuron1_Label FromNeuron2_Label ....

NumOutConnections ToNeuron1_Label ToNeuron2_Label ....

ActivationFunctionName NumberFunctionParameters Parameter1 Parameter2 ....

% the processing sequence of neurons

Neuron_Label Neuron_Label Neuron_Label ....

```

Table A.1: Format of Neural Network Structure Description Files.

Example 1:

A 3 layer perceptron with 1 input neuron, 1 output neuron and 5 hidden neurons, can be described as shown in Table A.2. Since the input neuron gets its input from outside, there is no corresponding FromNeurons. A number with negative sign is put in the list of FromNeurons, which indicates that the input is from outside data array. The absolute value of this number will be the index of this neuron's input in the outside data array. Similarly for the output neuron, the negative number in the ToNeuron list indicates that its output is going to the outside data array. The absolute value of this number will be the index of this neuron's output in the outside data array.

Example 2:

The neural network structure in this example is one of the KBNN model trained for mutual inductance modeling of microstrip lines as in Section 4.2. This KBNN structure has 6 input neurons, 3 knowledge neurons, 2 boundary neurons and 1 output neuron. Two application dependent functions, i.e., user provided functions, *User_Empirical_l12* and *User_Bound_l12*, are used in this structure. The structure is shown in Tables A.3 and A.4.

A.2 POWERNET Software Package

A.2.1 The Components of POWERNET Software Package

The POWERNET software package has three major components, i.e., *powernet*, *optimizer* and *utility*. The functionalities of each component are described as fol-

12	2			10	11							
	3			15	16	17						
	Multi_Sigmoid	4	-0.26	0.14	0.17	0.89						
13	2			10	11							
	3			15	16	17						
	Multi_Sigmoid	4	-2.30	-0.62	-0.03	-1.92						
14	2			10	11							
	3			15	16	17						
	Multi_Sigmoid	4	1.44	-1.26	0.94	-0.47						
15	3			12	13	14						
	1			18								
	Normalize	1	12									
16	3			12	13	14						
	1			19								
	Normalize	1	13									
17	3			12	13	14						
	1			20								
	Normalize	1	14									
18	2			7	15							
	1			21								
	Multiply_2	0										
19	2			8	16							
	1			21								
	Multiply_2	0										
20	2			9	17							
	1			21								
	Multiply_2	0										
21	3			18	19	20						
	1			-1								
	Linear	4	0.11	0.52	0.53	0.03						
1	2	3	4	5	6	7	8	9	10	11	12	
13	14	15	16	17	18	19	20	21				

Table A.4: An Example KBNN Structure Description File for Mutual Inductance Modeling of Microstrip Lines (continued).

lowing:

powernet : building an arbitrary neural network structure; feedforwarding the inputs through the network structure; calculating the gradient of error functions; building activation function library.

optimizer : training neural network using Adaptive Error Backpropagation; training neural network using gradient based l_2 optimization technique.

utility : perturbing neural network weights; searching the feedforwarding sequence of neurons for arbitrary neural network; randomizing training data sequence; selecting input and output fields from a given data file; scaling the inputs and outputs of a given set of data; special network structure description conversion.

There are two points which need to be emphasized. Firstly, building a library of activation functions to easily accommodate application dependent functions, i.e., user provided functions, is very important to this software package. A dynamic function linking developed in this software makes the adaptation to new applications much easier. Only the new functions need to be added and compiled. Secondly, though the general neural network structure description is very flexible and useful, it would be tedious for users to write such a description file as **POWERNET** input file. This description file is intended for machine use. For human communication, sometimes, a neural network structure can be described clearly by a small amount of information. For example, for a 3 layer perceptron, users may only need to indicate number of input neurons, output neurons and hidden neurons, and give a list of weights following specific sequence. Similarly, for KBNN, users may only need to indicate the number of input neurons, output neurons, knowledge neurons and boundary neurons.

The special network structure description conversion utility is developed to automatically convert the structure description in the human preferred form into the machine oriented form as described in Appendix A.1.

The **POWERNET** software package was developed following object oriented concept and coded in *C++*. The designed and implemented classes for each component are listed in Table A.5.

Powernet Components	Classes
powernet	PowerNeuron; PowerNet; FuncLib; LinkList (template class); String;
optimizer	Optimizer(base class); EBP; L2;
utility	Utility (virtual base class); UtilityNetwork (base class); Pertuber; SequenceSearcher; UtilityPattern (base class); InOutSelector; Scaler; Randomizer; MLPConverter; RBFCConverter; KBNNConverter;

Table A.5: The Classes within Each Components of **POWERNET** Software Package.

A.2.2 The Usage of **POWERNET** Software Package

The **POWERNET** software package can be used as is for any normal neural network structures which use the built-in activation functions, or relinked with the user implemented activation functions for special neural network structures, such as KBNN.

For Normal Neural Network Structures:

Any neural network structures, which use the built-in activation functions in

POWERNET software package, are called normal neural network structures. The built-in functions include Relay, Linear, Sigmoid, Arctangent, Hyperbolic_Tangent, Gaussian, Multi_Sigmoid, Normalize, Multiply_2, Bias and PiSigma.

Three files are needed to develop a neural network model using **POWERNET**, that is **DEFAULT.struc**, **DEFAULT.scale** and **DEFAULT.train**. The file format of **DEFAULT.struc** has been described in Section A.1. At the beginning of the training, this file contains the randomly initialized weights of the neural network. After the training, this file is updated by the trained weights.

DEFAULT.scale file contains the information about the data range of inputs and outputs, with the format shown in Table A.6. The training data is scaled to $[0\ 1]$ or

+/-			
Input#1_Min	Input#2_Min		
Input#1_Max	Input#2_Max		
Output#1_Min	Output#2_Min		
Output#1_Max	Output#2_Max		

Table A.6: Format of **DEFAULT.scale** File.

$[-1\ 1]$ range using the information in **DEFAULT.scale** file, for efficient training. The + or - in the first line controls what range the inputs are going to be scaled to. + means range $[0\ 1]$ and - means $[-1\ 1]$. In the normal case, ranges $[-1\ 1]$ and $[0\ 1]$ are both good choices. However, for knowledge based neural networks, the knowledge functions based on empirical experience may have specific physical meaning, which requires that the inputs be positive values, e.g., the geometric parameters. Range $[0$

1] should be used in this case. Outputs are always scaled to the range [0 1].

DEFAULT.train file contains information about training process, with the format shown in Table A.7. l_2 training used Jacobian matrix whose row number is the

NumDataPerGroup
NumTotalData
TrainingStopCriterionForError

Table A.7: Format of DEFAULT.train File.

product of the number of data and the number of outputs. When large amount of training data is used, the efficiency of the l_2 training is decreased due to the large size of Jacobian matrix. A local batch technique is used in **POWERNET** which divides the total training data into several groups, with one l_2 optimization per group. The training is performed cyclically through the groups. For example, if a set of training data with 500 samples is divided into 2 groups for l_2 training and the stop criterion is 1%, DEFAULT.train file is shown in Table A.8. For Error Backpropagation training,

250
500
0.01

Table A.8: An Example DEFAULT.train File.

there is no need to divide the training data into groups. NumDataPerGroup can be

set as NumTotalData.

Training of a neural network by **POWERNET** can be started by issuing command

```
% learn_EBP TrainingDataFile
```

or

```
% learn_l2 TrainingDataFile
```

where TrainingDataFile has the most common data file format with input columns followed by output columns as shown in Table A.9.

Data1_Input1	Data1_Input2		Data1_Output1	Data1_Output2	
Data2_Input1	Data2_Input2		Data2_Output1	Data2_Output2	
.....					

Table A.9: Format of Data Files.

After training, the neural network can be tested by issuing command

```
% recall TestDataFile
```

where TestDataFile has the same file format as TrainingDataFile. After testing, two files will be created, i.e., *predict.dat* and *out.dat*. The file *predict.dat* contains pairs of inputs and neural network predicted outputs. The file *out.dat* contains neural network predicted outputs, desired outputs and the sum_squared_error for individual data.

The usage of other utilities of **POWERNET** can be obtained by issuing the command of the executable file of each utility without any argument. The rest executable files include **randomize**, **perturb**, **inout_select**, **sequence_search**, **mlp_conv**,

rbf_conv and **kbnn_conv**.

For Special Neural Network Structures:

Any neural network structures, which use at least one activation function which is not in the list of the built-in activation functions in the **POWERNET** software package, are called special neural network structures. All the steps, file preparation and commands, described above are still applicable except that an extra step is needed before all those steps, that is to program and compile those application dependent activation functions and link with the objective files provided by the **POWERNET** package.

The *C++* source code file of these special functions, e.g., named *user_func.cpp*, should follow the format shown in Table A.10. The source file should be compiled into an object file, e.g., *user_func.o*. Then a file *user_func.def* should be edited to register these functions in the function library of **POWERNET**. The format of file *user_func.def* is shown in Table A.11. The function library should be recompiled and linked with *user_func.o* and the other object files provided by **POWERNET** software package to make a list of new executable files, such as **learn_l2**, **learn_ebp**, **recall**, and so on. These newly made executable files can be employed to develop special neural network models using all the steps described earlier for normal neural network model development.

```

FuncList[11].Name = "USER_Empirical1";

FuncList[11].NumParaSc = 7;

FuncList[11].NumParaBi = 0;

FuncPointer[11] = USER_Empirical1;


FuncList[12].Name = "USER_Empirical2";

FuncList[12].NumParaSc = 6;

FuncList[12].NumParaBi = 0;

FuncPointer[12] = USER_Empirical2;


NumFunc = 13;

```

Table A.11: Format of File: user.func.def. This File Lists User Defined Functions for POWERNET.

Bibliography

- [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] F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "A hierarchical neural network approach to the development of library of neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 46, December 1998 (accepted).
- [3] 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.
- [4] 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.
- [5] P. M. Watson and K. C. Gupta, "Design and optimization of CPW circuits using EM-ANN models for CPW components," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2515–2523, 1997.
- [6] G. L. Creech, B. Paul, C. Lesniak, T. Jenkins, R. Lee, and M. Calcaterra, "Artificial neural networks for accurate microwave CAD applications," In *IEEE Intl. Microwave Symp. Digest*, pp. 733–736, San Francisco, CA, 1996.

- [14] G. Kothapalli, "Artificial neural networks as aids in circuit design," *Microelectronics Journal*, vol. 26, pp. 569–678, 1995.
- [15] K. Shirakawa, M. Shimizu, N. Okubo, and Y. Daido, "A large signal characterization of an HEMT using a multilayered neural network," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 1630–1633, 1997.
- [16] K. Shirakawa, M. Shimizu, N. Okubo, and Y. Daido, "Structural determination of multilayered large signal neural-network HEMT model," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 1367–1375, 1998.
- [17] Y. Harkouss, J. Rousset, H. Chehade, D. Barataud, and J. P. Teyssier, "Modeling microwave devices and circuits for telecommunications system design," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 128–133, Anchorage, Alaska, May 1998.
- [18] 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.
- [19] Q. J. Zhang, F. Wang, and M. S. Nakhla, "Optimization of high-speed VLSI interconnects: A review," *Intl. J. of Microwave and Millimeter-Wave CAD*, vol. 7, pp. 83–107, 1997 (Invited).
- [20] T. Horng, C. Wang, and N. G. Alexopoulos, "Microstrip circuit design using neural networks," In *IEEE Intl. Microwave Symp. Digest*, pp. 413–416, Atlanta, Georgia, 1993.

- [21] P. Burrascano, M. Dionigi, C. Fancelli, and M. Mongiardo, "A neural network model for CAD and optimization of microwave filters," In *IEEE Intl. Microwave Symp. Digest*, pp. 13–16, Baltimore, MD, June 1998.
- [22] 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, 1995.
- [23] M. Vai, S. Wu, B. Li, and S. Prasad, "Reverse modeling of microwave circuits with bidirectional neural network models," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 1492–1494, 1998.
- [24] M. Vai and S. Prasad, "Automatic impedance matching with a neural network," *IEEE Microwave Guided Wave Letter*, vol. 3, pp. 353–354, 1993.
- [25] 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.
- [26] P. Meijer, "Fast and smooth highly nonlinear multidimensional table models for device modeling," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 335–346, 1990.
- [27] F. Wang, V. K. Devabhaktuni, C. G. Xi, and Q. J. Zhang, "Neural network structures and training algorithms for RF and microwave applications," *International Journal of RF and Microwave Computer-Aided Engineering: Special Issue on Applications of Artificial Neural Networks to RF and Microwave Design*, vol. 9, 1999 (accepted).

- [28] F. Scarselli and A. C. Tsoi, "Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results," *Neural Networks*, vol. 11, pp. 15–37, 1998.
- [29] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: Perceptron, madaline and backpropagation," *IEEE Proc.*, vol. 78, pp. 1415–1442, 1990.
- [30] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.
- [31] C. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Systems*, vol. 2, pp. 303–314, 1989.
- [32] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
- [33] 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.
- [34] 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.
- [35] F. Girosi, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, chapter Regularization Theory, Radial Basis Functions and Networks, pp. 166–187. Springer-Verlag, Berlin, NY, 1992.
- [36] M. J. D. Powell, *Algorithms for Approximation*, chapter Radial Basis Functions for Multivariate Interpolation: A Review, pp. 143–167. Oxford University Press, Oxford, UK, 1987.

- [46] H. Leung and S. Haykin, "Rational function neural network," *Neural Computation*, vol. 5, pp. 928–938, 1993.
- [47] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, pp. 119–165, 1994.
- [48] R. Maclin and J. W. Shavlik, "Using knowledge-based neural networks to improve algorithms: Refining the chou-fasman algorithm for protein folding," *Machine Learning*, vol. 11, pp. 195–215, 1993.
- [49] L. M. Fu, "Integration of neural heuristics into knowledge-based inference," *Connection Science*, vol. 1, pp. 325–340, 1989.
- [50] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, "Back-propagation learning in expert networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 62–72, January 1992.
- [51] J. J. Mahoney and R. J. Mooney, "Combining connectionist and symbolic learning to refine certainty factor rule bases," *Connection Science*, vol. 5, pp. 339–364, 1993.
- [52] W. J. Jansen, M. Diepenhorst, J. A. G. Nijhuis, and L. Spaanenburg, "Assembling engineering knowledge in a modular multi-layer perceptron neural networks," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 232–237, Houston, TX, June 1997.
- [53] G. G. Lendaris, A. Rest, and T. R. Misley, "Improving ANN generalization using a priori knowledge to pre-structure ANNs," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 248–253, Houston, TX, June 1997.

- [54] T. A. Johansen and B. A. Foss, "Constructing NARMAX models using ARMAX models," *Intl. Journal of Control*, vol. 58, pp. 1125–1153, 1993.
- [55] T. A. Johansen and B. A. Foss, "Identification of nonlinear system structure and parameters using regime decomposition," *Automatica*, vol. 31, pp. 321–326, 1995.
- [56] O. Nelles, S. Sinsel, and R. Isermann, "Local basis function networks for identification of a turbocharger," In *Proc. UKACC Intl. Conf. Control*, pp. 7–12, Exeter, UK, July 1996.
- [57] S. Mcloone, M. D. Brown, G. Irwin, and G. Lightbody, "A hybrid linear/nonlinear training algorithm for feedforward neural networks," *IEEE Trans. Neural Networks*, vol. 9, pp. 669–684, 1998.
- [58] C. S. Walker, *Capacitance, Inductance and Crosstalk Analysis*, Artech House, Boston, MA, 1990.
- [59] P. H. Ladbrooke, *MMIC Design: GaAs FETs and HEMTs*, Artech House, Boston, MA, 1989.
- [60] A. J. Sharkey, "On combining artificial neural nets," *Connection Science*, vol. 8, pp. 299–314, 1996.
- [61] S. Hashem and B. Schmeiser, "Improving model accuracy using optimal linear combinations of trained neural network," *IEEE Trans. Neural Networks*, vol. 6, pp. 792–794, 1995.

- [62] S. Hashem, "Algorithms for optimal linear combinations of neural networks," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 242–247, Houston, TX, June 1997.
- [63] A. J. Sharkey, "Modularity, combining and artificial neural nets," *Connection Science*, vol. 9, pp. 3–10, 1997.
- [64] R. A. Jacobs, F. Peng, and M. A. Tanner, "A bayesian approach to model selection in hierarchical mixtures-of-experts architectures," *Neural Networks*, vol. 10, pp. 231–241, 1997.
- [65] G. Auda, M. Kamel, and H. Raafat, "A new neural network structure with cooperative modules," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 1301–1306, Orlando, FL, June 1994.
- [66] L. C. Wang, N. M. Nasrabadi, and S. Der, "Asymptotical analysis of a modular neural network," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 1019–1022, Houston, TX, June 1997.
- [67] Q. H. Zhang, "Using wavelet network in nonparametric estimation," *IEEE Trans. Neural Networks*, vol. 8, pp. 227–236, 1997.
- [68] Q. H. Zhang and A. Benvensite, "Wavelet networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 889–898, 1992.
- [69] B. R. Bakshi and G. Stephanopoulos, "Wave-net: a multiresolution, hierarchical neural network with localized learning," *America Institute of Chemical Eng. Journal*, vol. 39, pp. 57–81, 1993.

- [70] B. Delyon, A. Juditsky, and A. Benveniste, "Accuracy analysis for wavelet approximations," *IEEE Trans. Neural Networks*, vol. 6, pp. 332–348, 1995.
- [71] S. E. Fahlman and C. Lebiere, *Advances in Neural Information Processing*, chapter The Cascade-correlation Learning Architecture, pp. 524–532. Morgan Kaufman, San Mateo, CA, 1990.
- [72] N. K. Treadgold and T. D. Gedeon, "Extending CasPer: a regression survey," In *Intl. Conf. Neural Information Processing*, pp. 310–313, 1997.
- [73] N. K. Treadgold and T. D. Gedeon, "Exploring architecture variation in constructive cascade networks," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 343–348, Anchorage, Alaska, May 1998.
- [74] J. N. Hwang, S. S. You, S. R. Lay, and I. C. Jou, "The cascade-correlation learning: a projection pursuit learning perspective," *IEEE Trans. Neural Networks*, vol. 7, pp. 278–289, 1996.
- [75] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume I, pp. 318–362, MIT Press, Cambridge, Massachusetts, 1986.
- [76] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1989.
- [77] 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.

- [78] S. J. Perantonis and D. A. Karras, "An efficient constrained learning algorithm with momentum acceleration," *Neural Networks*, vol. 8, pp. 237–249, 1995.
- [79] N. B. Karayiannis and A. N. Venetsanopoulos, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications*, Kluwer Academic Publishers, Boston, Massachusetts, 1993.
- [80] N. Morgan, "Big dumb neural nets: A working brute force approach to speech recognition," In *Proc. IEEE Intl. Conf. Neural Networks*, volume VII, pp. 4462–4465, Orlando, Florida, July 1994.
- [81] F. Wang and Q. J. Zhang, "A sparse matrix approach to neural network training," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 2743–2747, Perth, Australia, November 1995.
- [82] *Neural Network Toolbox: For Use with Matlab*, The MathWorks Inc., Natick, Massachusetts, 1993.
- [83] R. A. Jacobs, "Increased rate of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.
- [84] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "Device and circuit-level modeling using neural networks with faster training based on network sparsity," *IEEE Trans. Microwave Theory and Tech.*, vol. 45, pp. 1696–1704, 1997.
- [85] F. Wang and Q. J. Zhang, "An adaptive and fully sparse training approach for multilayer perceptrons," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 102–107, Washington, D.C., June 1996.

- [86] R. Battiti, "Accelerated backpropagation learning: Two optimization methods," *Complex Systems*, vol. 3, pp. 331–342, 1989.
- [87] M. Arisawa and J. Watada, "Enhanced back-propagation learning and its application to business evaluation," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 155–160, Orlando, Florida, July 1994.
- [88] 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, 1994.
- [89] X. H. Yu, G. A. Chen, and S. X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 669–677, 1995.
- [90] 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.
- [91] P. P. Van-Der-Smagt, "Minimisation methods for training feedforward neural networks," *Neural Networks*, vol. 7, pp. 1–11, 1994.
- [92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, MA, 1986.
- [93] D. R. Hush and J. M. Salas, "Improving the learning rate of back propagation with the gradient reuse algorithm," In *Proc. IEEE Intl. Conf. Neural Networks*, volume I, pp. 441–447, San Diego, California, 1988.

- [94] D. B. Parker, "Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation and second order hebbian learning," In *Proc. IEEE First Intl. Conf. Neural Networks*, volume II, pp. 593–600, San Diego, California, 1987.
- [95] R. L. Watrous, "Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization," In *Proc. IEEE First Intl. Conf. Neural Networks*, volume II, pp. 619–627, San Diego, California, 1987.
- [96] R. Battiti, "First and second-order methods for learning: Between steepest descent and newton's method," *Neural Computation*, vol. 4, pp. 141–166, 1992.
- [97] J. S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing, A Computational Approach to Learning and Machine Intelligence*, chapter Derivative-based Optimization, pp. 129–172. Prentice Hall, Upper Saddle River, NJ, 1997.
- [98] M. F. Moller, "A scaled conjugate gradient algorithms for fast supervised learning," *Neural Networks*, vol. 6, pp. 525–533, 1993.
- [99] T. R. Cuthbert(Jr.), *Optimization using Personal Computers*, chapter Quasi-Newton Methods and Constraints, pp. 233–314. John Wiley and Sons, New York, NY, 1987.
- [100] K. R. Nakano, "Partial BFGS update and efficient step-length calculation for three layer neural network," *Neural Computation*, vol. 9, pp. 123–141, 1997.
- [101] S. Mcloone and G. W. Irwin, "Fast parallel off-line training of multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 8, pp. 646–653, 1997.

- [102] A. J. Shepherd, *Second-order Methods for Neural Networks*, chapter Second-order Optimization Methods, pp. 43–72. Springer-Verlag, Berlin, NY, 1997.
- [103] 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.
- [104] M. G. Bello, “Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks,” *IEEE Trans. Neural Networks*, vol. 3, pp. 864–875, November 1992.
- [105] 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, 1998.
- [106] S. Ergezinger and E. Thomsen, “An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer,” *IEEE Trans. Neural Networks*, vol. 6, pp. 31–42, January 1995.
- [107] G. J. Wang and C. C. Chen, “A fast multilayer neural network training algorithm based on the layer-by-layer optimizing procedures,” *IEEE Trans. Neural Networks*, vol. 7, pp. 768–775, 1996.
- [108] R. Lengelle and T. Denceux, “Training MLPs layer by layer using an objective function for internal representations,” *Neural Networks*, vol. 9, pp. 83–97, 1996.
- [109] J. S. Shawe-Taylor and D. A. Cohen, “Linear programming algorithm for neural networks,” *Neural Networks*, vol. 3, pp. 575–582, 1990.

- [110] B. Berma, "Fast training of multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 8, pp. 1314–1320, 1997.
- [111] F. Wang and Q. J. Zhang, "An improved k-means clustering algorithm and application to combined multi-codebook/MLP neural network speech recognition," In *Proc. Canadian Conf. Electrical and Computer Engineering*, pp. 999–1002, Montreal, Canada, September 1995.
- [112] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.
- [113] J. C. F. Pujol and R. Poli, "Evolving neural networks using a dual representation with a combined crossover operator," In *Proc. IEEE Intl. Conf. Evolutionary Computing*, pp. 416–421, Anchorage, Alaska, May 1998.
- [114] T. Rönkvallsson, "On Langevin updating in multilayer perceptron," *Neural Computation*, vol. 6, pp. 916–926, 1994.
- [115] R. Brunelli, "Training neural nets through stochastic minimization," *Neural Networks*, vol. 7, pp. 1405–1412, 1994.
- [116] *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.
- [117] A. Djordjevic, R. F. Harrington, T. Sarkar, and M. Bazdar, *Matrix Parameters for Multiconductor Transmission Lines: Software and User's Manual*, Artech House, Boston, MA, 1989.

- [118] F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "A new modeling approach for high-speed and high-frequency circuits and systems," In *Proc. Intl. Conf. Microwave and Millimeter Wave Tech.*, pp. 951–952, Beijing, China, August 1998.
- [119] Q. J. Zhang, G. Wilson, R. Venkatachalam, A. Sarangan, J. Williamson, and F. Wang, "Ultra fast neural models for analysis of electro/optical interconnects," In *Proc. IEEE Electronic Components and Tech. Conf.*, San Jose, CA, May 1997.
- [120] M. S. Nakhla and Q. J. Zhang, editors, *Modeling and Simulation of High-Speed VLSI Interconnects*, Kluwer, Boston, MA, 1994.
- [121] H. Hasegawa and S. Seki, "Analysis of interconnection delay on very high-speed LSI/VLSI chips using a microstrip line model," *IEEE Trans. Electron Devices*, vol. ED-31, pp. 1954–1960, 1984.
- [122] A. R. Djordjevic, T. K. Sarkar, and R. F. Harrington, "Analysis of lossy transmission lines with arbitrary nonlinear terminal networks," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-34, pp. 660–666, 1986.
- [123] F. Y. Chang, "The generalized method of characteristics for waveform relaxation analysis of lossy coupled transmission lines," *IEEE Trans. Microwave Theory Tech.*, vol. 37, pp. 2028–2038, 1989.
- [124] R. Wang and O. Wing, "A circuit model of a system of VLSI interconnects for time response computation," *IEEE Trans. Microwave Theory Tech.*, vol. 39, pp. 688–693, 1991.

- [125] W. W. M. Dai, editor, *IEEE Trans. Circuits and Systems Part I: Special Issue on Modeling, Simulation and Electrical Design of High-Speed High-Density Interconnects*, volume 39, November 1992.
- [126] E. Chiprout and M. S. Nakhla, *Asymptotic Waveform Evaluation and Moment Matching for Interconnect Analysis*, Kluwer, Boston, MA, 1994.
- [127] Q. J. Zhang, F. Wang, M. S. Nakhla, J. Bandler, and R. Biernacki, "Signal integrity optimization of high-speed VLSI packages and interconnects," In *Proc. IEEE Electronic Components and Tech. Conf.*, pp. 1073–1076, Seattle, Washington, May 1998.
- [128] Q. J. Zhang, F. Wang, and M. S. Nakhla, "Optimization of high-speed VLSI interconnects," In *Proc. Symp. Antenna Tech. and Applied Electromagnetics*, pp. 265–268, Montreal, QC, August 1996.
- [129] Q. J. Zhang, S. Lum, and M. S. Nakhla, "Minimization of delay and crosstalk in high-speed VLSI interconnects," *IEEE Trans. Microwave Theory Tech.*, vol. 40, pp. 1555–1563, 1992.
- [130] R. Liu, Q. J. Zhang, and M. S. Nakhla, "A frequency-domain approach to performance optimization of high-speed VLSI interconnects," *IEEE Trans. Microwave Theory Tech.*, vol. 40, pp. 2403–2411, 1992.
- [131] T. Rahal-Arabi and R. Suarez-Gartner, "A frequency-domain technique for the optimization of the electrical performance of high-speed multiconductor transmission line networks in VLSI regimes," *IEEE Trans. Circuits Syst.*, vol. 40, pp. 262–269, 1993.

- [132] S. Pullela, N. Menezes, and L. Pillage, "Skew and delay optimization for reliable buffered clock trees," In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 556–562, Santa Clara, CA, November 1993.
- [133] Y. Wei, Q. J. Zhang, and M. S. Nakhla, "Multilevel optimization of high-speed VLSI interconnect networks by decomposition," *IEEE Trans. Microwave Theory Tech.*, vol. 42, pp. 1638–1650, 1994.
- [134] R. Achar, M. S. Nakhla, and Q. J. Zhang, "Moment matching techniques for transient analysis of high-speed VLSI interconnects with full wave model," In *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 1C.3, San Jose, CA, November 1995.
- [135] K. Mihan, B. Stacey, M. S. Nakhla, and Q. J. Zhang, "Concurrent thermal and electrical optimization of high-speed packages and systems," In *INTERpack: ASME International Intersociety Electronic Packaging Conference*, pp. 221–227, Lahaina, Hawaii, March 1995.
- [136] R. Achar, M. S. Nakhla, and Q. J. Zhang, "Addressing high-frequency effects in VLSI interconnects with full-wave model and CFH," In *Proc. European Microwave Conf.*, pp. 820–823, Bologna, Italy, September 1995.
- [137] A. E. Ruehli, P. K. Wolff, and G. Goertzel, "Analytical power/timing optimization technique for digital system," In *Proc. Design Automation Conference*, New Orleans, LA, June 1977.
- [138] J. A. Barby, J. Vlach, and K. Singhal, "Polynomial splines for MOSFET model approximation," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 557–567, 1988.

- [139] Q. Zhu, W. W. M. Dai, and J. G. Xi, "Optimal sizing of high-speed clock networks based on distributed RC and lossy transmission line models," In *Proc. Intl. Conf. Computer-Aided Design*, pp. 628–633, Santa Clara, CA, November 1993.
- [140] H. F. Liao, W. W. M. Dai, R. Wang, and F. Y. Chang, "S-parameter based macro model of distributed-lumped networks using exponentially decayed polynomial function," In *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 726–731, 1993.
- [141] A. Veluswami, "Neural network-based CAD for high-speed interconnects and monolithic inductors," Master's thesis, Carleton University, Ottawa, Canada, November 1995.
- [142] A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, "Application of neural networks in circuit analysis," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 423–426, Perth, Australia, November 1995.
- [143] R. Griffith, E. Chiprout, Q. J. Zhang, and M. S. Nakhla, "A CAD framework for simulation and optimization of high-speed VLSI interconnects," *IEEE Trans. Circuits Syst. I*, vol. 39, pp. 893–906, 1992.
- [144] F. Wang and Q. J. Zhang, "Knowledge based neural models for microwave design," In *IEEE Intl. Microwave Symp. Digest*, pp. 627–630, Denver, CO, June 1997.
- [145] 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.

- [146] A. Abdelbar and G. Tagliarini, "Honest: A new high order feedforward neural networks," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 1257–1262, Washington, D.C., June 1996.
- [147] J. R. Tsai, P. C. Chung, and C. I. Chang, "A sigmoidal radial basis function neural network for function approximation," In *Proc. IEEE Intl. Conf. Neural Networks*, pp. 496–501, Washington, D.C., June 1996.
- [148] J. W. Bandler and Q. J. Zhang, "Optimization techniques for modeling, diagnosis and tuning," In T. Ozawa, editor, *Analog Methods for Computer-Aided Circuit Analysis and Diagnosis*, Marcel Dekker, New York, NY, 1988.
- [149] D. Zhou, S. Su, F. Tsui, D. S. Gao, and J. Cong, "A simplified synthesis of transmission lines with a tree structure," *Analog Integrated Circuits and Signal Processing: Special Issue on Modeling and Simulation of High Speed VLSI Interconnects*, vol. 5, pp. 19–30, 1994.
- [150] *SALI*, Northern Telecom, P.O. Box 3511, Ottawa, Canada, 1994.
- [151] 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, 1988.
- [152] F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "A hierarchical neural network approach to the development of library of neural models for microwave design," In *IEEE Intl. Microwave Symp. Digest*, pp. 1767–1770, Baltimore, MD, June 1998.

- [153] F. Wang and Q. J. Zhang, "Incorporating combined functional/structural knowledge in library of neural networks," In *Proc. Intl. Conf. Neural Networks*, pp. 363–366, Anchorage, Alaska, May 1998.
- [154] J. W. Bandler, R. M. Biernacki, S. H. Chen, P. A. Grobelny, and R. H. Hemmers, "Space mapping technique for electromagnetic optimization," *IEEE Trans. Microwave Theory Tech.*, vol. 42, pp. 2536–2544, 1994.
- [155] *NeuroModeler Version 1.0*, Professor Q. J. Zhang and his Research Team, Department of Electronics, Carleton University, 1125 Colonel By Drive, Ottawa, . Ontario, K1S 5B6, Canada.

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved