Neural Network Structures and Training Algorithms for RF and Microwave Applications

Fang Wang, Vijaya K. Devabhaktuni, Changgeng Xi, Qi-Jun Zhang

Department of Electronics, Carleton University, Ottawa, Canada, K1S 5B6; e-mail: fwang@doe.carleton.ca; vijay@doe.carleton.ca; cgxi@doe.carleton.ca; qjz@doe.carleton.ca

Received 11 August 1998; revised 1 December 1998

ABSTRACT: Neural networks recently gained attention as fast and flexible vehicles to microwave modeling, simulation, and optimization. After learning and abstracting from microwave data, through a process called training, neural network models are used during microwave design to provide instant answers to the task learned. Appropriate neural network structure and suitable training algorithm are two of the major issues in developing neural network models for microwave applications. Together, they decide amount of training data required, accuracy that could possibly be achieved, and more importantly developmental cost of neural models. A review of the current status of this emerging technology is presented, with emphasis on neural network structures and training algorithms suitable for microwave applications. Present challenges and future directions of the area are discussed. © 1999 John Wiley & Sons, Inc. Int J RF and Microwave CAE 9: 216–240, 1999.

Keywords: neural networks; structures; training; modeling; RF; microwave

I. INTRODUCTION

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–electromagnetic (EM) models of active and passive components can be an important step toward a design for first-pass success, but it is computationally intensive. In recent years a novel computer-aided design (CAD) approach based on neural network technology has been introduced in the microwave community, for the modeling of passive and active microwave components [1–5], and microwave circuit design [2, 4, 6, 7]. A neural network model for a device–circuit can be developed by learning and abstracting from measured–simulated microwave data, through a process called training. Once trained, the neural network model can be used during microwave design to provide instant answers to the task it learned [1]. Recent work by microwave researchers demonstrated the ability of neural networks to accurately model a variety of microwave components, such as microstrip interconnects [1, 3, 8], vias [3, 9], spiral inductors [5, 10], FET devices [1, 11–13], power transistors and power amplifiers [14], coplanar waveguide (CPW) circuit components [4], packaging and interconnects [15], etc. Neural networks have been used in circuit simulation and optimization [2, 13, 16], signal integrity analysis and optimization of very large scale integrated (VLSI) circuit interconnects [8, 15], microstrip circuit design [17], microwave filter design [18], integrated circuit (IC) modeling [11], process design [19], synthesis [6], Smith chart representation [7], and microwave impedance matching [20]. The neural network technologies have been applied to microwave circuit optimization and statistical design with neural network models at both device and circuit levels [2, 12]. These pioneering works helped to establish the

Correspondence to: Q.-J. Zhang

framework of neural modeling technology in microwave applications. Neural models are much faster than original detailed EM–physics models [1, 2], more accurate than polynomial and empirical models [21], allow more dimensions than table lookup models [22], and are easier to develop when a new device–technology is introduced [16].

Theoretically, neural network models are a kind of 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 nonlinear, nonsmooth, or containing many variables. An appropriate structure would help to achieve higher model accuracy with fewer training data. 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 criteria in the development of a neural network. Too small a network cannot learn the problem well, but too large a size will lead to overlearning. An important type of neural network structure is the knowledge-based neural networks where microwave empirical information is embedded into neural network structures [1], enhancing reliability of the neural model and reducing the amount of training data needed.

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 backpropagation (BP), which was proposed in the mid 1980s. 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.

In Section II, the problem statement of neural based microwave modeling is described. In Section III, a detailed review of various neural network structures useful to microwave design area such as standard feedforward networks (multilayer perceptrons, radial basis functions), networks with prior knowledge (electromagnetic artificial neural networks or EM-ANN, knowledge based neural networks), combined networks, wavelet and constructive networks, is presented. Section IV discusses various training algorithms of interest such as backpropagation, second-order algorithms (conjugate gradient, quasi-Newton, Levenberg–Marquardt), decomposed optimization algorithms, and global training algorithms. Section V presents a comparison of neural network structures and training algorithms, through microwave examples. Finally, Section VI contains conclusions and future challenges in this area.

II. NEURAL BASED MICROWAVE MODELING: PROBLEM STATEMENT

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

$$\mathbf{y} = \mathbf{f}(\mathbf{x}). \tag{1}$$

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

$$\{(\mathbf{x}_p, \mathbf{d}_p),\ p = 1, 2, \ldots, N_p\}, \tag{2}$$

where $\mathbf{x}_p$ and $\mathbf{d}_p$ are N_x- and N_y-dimensional vectors representing the pth 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 (1) be represented by,

$$\mathbf{y} = \mathbf{y}(\mathbf{x}, \mathbf{w}), \tag{3}$$

where $\mathbf{w}$ is the parameter 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 (3) does not represent the original problem of (1), unless the neural model is trained by data in (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,

$$E(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{N_p} \sum_{k=1}^{N_y} \left(y_{pk}(\mathbf{x}_p, \mathbf{w}) - d_{pk} \right)^2 \tag{4}$$

is minimized. In (4) d_{pk} is the kth element of vector $\mathbf{d}_p$, $y_{pk}(\mathbf{x}_p, \mathbf{w})$ is the kth output of the neural network when the input presented to the network is $\mathbf{x}_p$. Once trained, the neural network model can be used for predicting the output values given only the values of the input variables. In the model testing stage, an independent set of input–output samples, called the testing data is used to test the accuracy of the neural model. Normally, the testing data should lie within the same input range as the training data. The ability of neural models to predict $\mathbf{y}$ 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.

III. NEURAL NETWORK STRUCTURES

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

A. 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 [23]. An important class of feedforward neural networks is multilayer perceptrons (MLP). Typically, the MLP neural network consists of an input layer, one or more hidden layers, and an output layer, as shown in Figure 1. Suppose the total number of hidden layers is L. The input layer is considered as layer 0. Let the number of neurons in hidden layer l be N_l, $l = 1, 2,, \ldots, L$. Let w_{ij}^l represent the weight of the link between the jth neuron of the $l-1$th hidden layer and ith neuron of lth hidden layer, and θ_i^l be the bias parameter of ith neuron of lth hidden layer. Let x_i represent the ith input parameter to the MLP. Let $\bar{y}_i^l$ be the output of ith neuron of lth hidden layer, which can be computed according to the standard MLP formulas

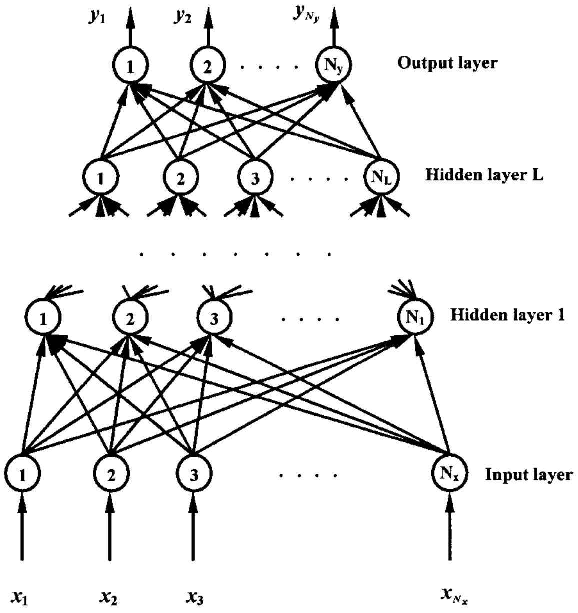


Figure 1. Multilayer perceptron (MLP) structure. Typically, the network consists of an input layer, one or more hidden layers, and an output layer.

as,

$$\bar{y}_i^{\ell} = \sigma\left(\sum_{j=1}^{N_{l-1}} w_{ij}^{\ell}\cdot\bar{y}_j^{l-1} + \theta_i^l\right), \quad i = 1,\dots,N_l, \quad l = 1,\dots,L \tag{5}$$

$$\bar{y}_i^0 = x_i, \quad i = 1,\dots,N_x, \quad N_x = N_0, \tag{6}$$

where $\sigma(.)$ is usually a monotone squashing function. Let ν_{ki} represent the weight of the link between the ith neuron of the Lth hidden layer and the kth neuron of the output layer, and β_k be the bias parameter of the kth output neuron. The outputs of MLP can be computed as,

$$y_k = \sum_{i=1}^{N_L} \nu_{ki}\cdot\bar{y}_i^L + \beta_k, \qquad k = 1,\dots,N_y. \tag{7}$$

For function approximation, output neurons can be processed by linear functions as shown in (7). The most commonly used function for hidden neurons $\sigma(.)$, also called the activation function, is the logistic sigmoid function given by,

$$\sigma(t) = \frac{1}{(1+e^{-t})}, \tag{8}$$

which has the property of,

$$\sigma(t) \to \begin{cases} 1, & \text{as } t \to +\infty, \\ 0, & \text{as } t \to -\infty. \end{cases} \tag{9}$$

Other possible candidates for $\sigma(.)$ are the arctangent function given by,

$$\sigma(t) = \left(\frac{2}{\pi}\right)\arctan(t), \tag{10}$$

and the hyperbolic tangent function given by,

$$\sigma(t) = \frac{(e^t - e^{-t})}{(e^t + e^{-t})}. \tag{11}$$

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

$$\mathbf{w} = \Big[w_{ij}^l, j = 1,\dots,N_{l-1}, i = 1,\dots,N_l, \; l = 1,\dots,L;\ \theta_i^l, i = 1,\dots,N_l, \; l = 1,\dots,L; \; \nu_{ki}, i = 1,\dots,N_L, k = 1,\dots,N_y; \; \beta_k, k = 1,\dots,N_y\Big]. \tag{12}$$

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

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

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 [30, 31]. Park and Sandberg showed that RBF networks also have universal approximation ability [32, 33]. Universal convergence of RBF nets in function estimation and classification has been proven by Krzyzak, Linder, and Lugosi [34].

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

$$y_j = \sum_{i=1}^{N_1} \nu_{ji}\sigma(\|\mathbf{x} - \boldsymbol{\theta}_i\|), \qquad j = 1,\dots,N_y, \tag{13}$$

where $\boldsymbol{\theta}_i$ is the center of radial basis function of the ith hidden neuron, ν_{ji} is the weight of the link from the ith hidden neuron to the jth output neuron. Some of the commonly used radial basis activation functions are [35],

$$\sigma(t) = \exp\left[-\left(\frac{t}{\lambda}\right)^2\right], \tag{14}$$

$$\sigma(t) = (c^2 + t^2)^\beta, \qquad 0 < \beta < 1, \tag{15}$$

where (14) is the Gaussian function, (15) is the multiquadratic function and, λ, c, and β are the function parameters. Training parameters $\mathbf{w}$ includes $\boldsymbol{\theta}_i$, ν_{ji}, $i = 1, \dots, N_1$, $j = 1, \dots, N_y$ and λ or c and β.

Although MLP and RBF are both feedforward neural networks, the different nature of the hidden neuron activation functions makes them behave very differently. First, the activation function of each hidden neuron in an 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. Second, MLP networks construct global approximations to nonlinear input–output mapping. Consequently, they are capable of generalizing in those regions of the input space where little or no training data is available. On the contrary, RBF networks use exponentially decaying localized nonlinearities (e.g., Gaussian functions) to construct local approximations to nonlinear input–output mapping. As a result RBF neural networks are capable of faster learning and exhibit reduced sensitivity to the order of presentation of training data [36]. Consequently, a hidden neuron influences the outputs of the network only for inputs near to its center, and an exponential number of hidden neurons are required to cover the entire domain. In [37], it is suggested that RBF networks are suited for problems with a smaller number of inputs.

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

B. 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, a large amount of training data is usually needed to ensure model accuracy. In microwave applications, obtaining a relatively larger set of training data by either EM–physics simulation, or by measurement, is expensive and/or difficult. The underlying reason is that simulation–measurement may have to be performed for many combinations of geometrical–material–process parameters. On the contrary, if we try to reduce the training data, the resulting neural models may not be reliable. Neural network structures with prior knowledge address this problem. 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 [3]. 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 [3]. The second method in this approach, is the prior knowledge input (PKI) method [9]. 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 learned by a 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 [9] that PKI method performs better than the difference method for 2-port GaAs microstrip ground via.

The second approach to incorporate prior knowledge, is to incorporate the knowledge directly into neural network internal structure, e.g., [42]. 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 [42–44]. The weights, e.g., the certainty factor associated with rules [45] or both the topology and weights of the network can be revised during training [46]. 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 [47, 48]. Another neural network structure worth mentioning here is the local model network (LMN) [49–52] 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 [36]. These existing approaches to incorporate knowledge are largely using symbolic information which exist in the pattern recognition area. In the microwave modeling areas however, the important problem knowledge is usually available in the form of functional empirical models [53, 54].

In [1], a new microwave-oriented knowledge-based neural network (KBNN) was introduced. In this method the microwave knowledge in the form of empirical functions or analytical approximations is embedded into the neural network structure. This structure was inspired from the fact that practical empirical functions are usually valid only in a certain region of the parameter space. To build a neural model for the entire space, several empirical formulas and the mechanism to switch between them are needed. The switching mechanism expands the feature of the sigmoidal radial basis function [55] into high-dimensional space and with more generalized activation functions. This model retains the essence of neural networks in that the exact location of each switching boundary, and the scale and position of each knowledge function are initialized randomly and then determined eventually during training. The knowledge-based neural network (KBNN) structure is a nonfully connected structure as shown in Figure 2. There are six 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 mi-

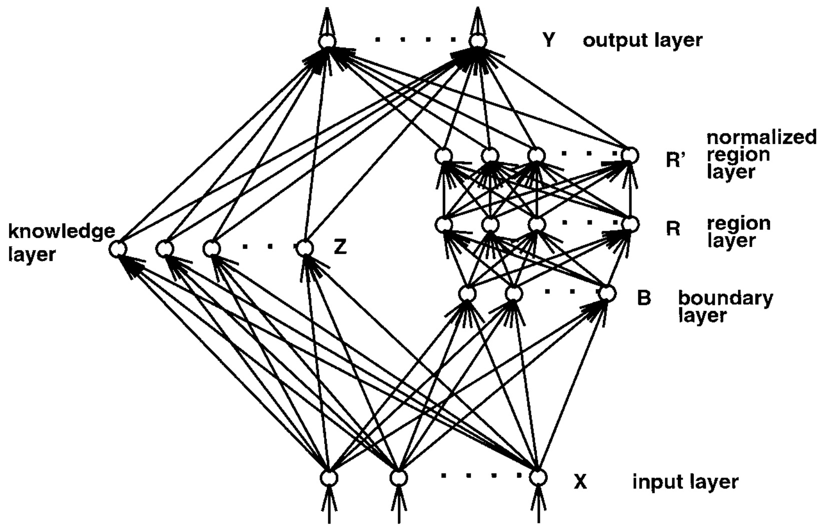


Figure 2. Knowledge-based neural network (KBNN) structure.

crowave knowledge resides in the form of single or multidimensional functions $\boldsymbol{\psi}(.)$.

The output of the ith knowledge neuron is given by,

$$\mathbf{z}_i = \boldsymbol{\psi}_i(\mathbf{x}, \mathbf{w}_i), \qquad i = 1, 2, \ldots, N_z, \quad (16)$$

where $\mathbf{x}$ is a vector including neural network inputs x_i, $i = 1, 2, \ldots, N_x$ and $\mathbf{w}_i$ is a vector of parameters in the knowledge formula. The knowledge function $\boldsymbol{\psi}_i(\mathbf{x}, \mathbf{w}_i)$ is usually in the form of empirical or semi-analytical functions. For example, the drain current of a FET is a function of its gate length, gate width, channel thickness, doping density, gate voltage, and drain voltage [54]. The boundary layer $\mathbf{B}$ can incorporate knowledge in the form of problem dependent boundary functions $\mathbf{B}(.)$; or in the absence of boundary knowledge just as linear boundaries. Output of the ith neuron in this layer is calculated by,

$$b_i = B_i(\mathbf{x}, \boldsymbol{\nu}_i), \qquad i = 1, 2, \ldots, N_b, \quad (17)$$

where $\boldsymbol{\nu}_i$ is a vector of parameters in B_i defining an open or closed boundary in the input space $\mathbf{x}$. Let $\sigma(.)$ be a sigmoid function. The region layer $\mathbf{R}$ contains neurons to construct regions from boundary neurons,

$$r_i = \prod_{j=1}^{N_b} \sigma(\alpha_{ij} b_j + \theta_{ij}), \qquad i = 1, 2, \ldots, N_r, \quad (18)$$

where α_{ij} and θ_{ij} are the scaling and bias parameters, respectively. The normalized region layer $\mathbf{R}'$ contains rational function based neurons [41] to normalize the outputs of region layer,

$$r_i' = \frac{r_i}{\sum_{j=1}^{N_r} r_j}, \qquad i = 1, 2, \ldots, N_{r'}, \; N_{r'} = N_r. \quad (19)$$

The output layer $\mathbf{Y}$ contains second-order neurons [56] combining knowledge neurons and normalized region neurons,

$$y_j = \sum_{i=1}^{N_z} \beta_{ji} z_i \left(\sum_{k=1}^{N_{r'}} \rho_{jik} r_k' \right) + \beta_{j0}, \qquad j = 1, 2, \ldots, N_y, \quad (20)$$

where β_{ji} reflects the contribution of the ith knowledge neuron to output neuron y_j and β_{j0} is the bias parameter. ρ_{jik} is one indicating that region r_k' is the effective region of the ith knowledge neuron contributing to the jth output. A total of $N_{r'}$ regions are shared by all the output neurons. Training parameters $\mathbf{w}$ for the entire KBNN model includes,

$$\begin{aligned} \mathbf{w} = [&\mathbf{w}_i, i = 1, \ldots, N_z; \boldsymbol{\nu}_i, i = 1, \ldots, N_b; \\ &\alpha_{ij}, \theta_{ij}, i = 1, \ldots, N_r, j = 1, \ldots, N_b; \\ &\beta_{ji}, j = 1, \ldots, N_y, i = 0, \ldots, N_z; \\ &\rho_{jik}, j = 1, \ldots, N_y, i = 1, \ldots, N_z, \\ &k = 1, \ldots, N_{r'}.]. \end{aligned} \quad (21)$$

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

C. Combining Neural Networks

In the neural network research community, a recent development called combining neural networks is presently proposed, addressing issues of network accuracy and training efficiency. Two categories of approaches have been developed: ensemble-based approach and modular approach [57]. In the ensemble-based approach [57, 58], 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 output than would be obtained by selecting the best net. Optimal linear combinations (OLCs) of neural networks were proposed and investigated in [58–60], which is constructed by forming weighted sums of the corresponding outputs of the individual networks. The second category, e.g., [36, 61], 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 [62]) or explicit (based on prior knowledge of the task or the specialist capabilities of the modules, e.g., [47, 48]). The modular neural network consists of several neural networks, each optimized to perform a particular subtask 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 [63], but also facilitates incorporation of problem knowledge into the network structure. This leads to improved overall network reliability and/or training efficiency [48, 64].

A new hierarchical neural network approach for the development of the library of microwave neural models, motivated by the concept of combining neural networks, was proposed in [65]. This approach is targeted for massively developing neural network models for building libraries of microwave models. Library development is of practical significance, since the realistic power of many CAD tools depends upon the richness, speed, and the accuracy of their library models. In this 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. A hierarchical neural network structure, as shown in Figure 3, is constructed for each model in the library with lower level modules realized by base neural models. The purpose of this structure is to construct an overall model from several modules so that the library base relationship can be maximally reused for every model throughout the library. For each lower level module, an index function selects the appropriate base model, and a structural knowledge hub selects inputs relevant to the base model out of the whole input vector based on the configuration of the particular library component. The lower level neural modules recall the trained base models in the library. The higher level module realized by another neural network, models a much easier relationship than the original relationship since most of the information is already contained in the base models in the lower level. For example, even a linear two-layer perceptron might be sufficient. Consequently, the amount of data needed to train this higher level module is much less than that required for training standard MLP to learn the original problem.

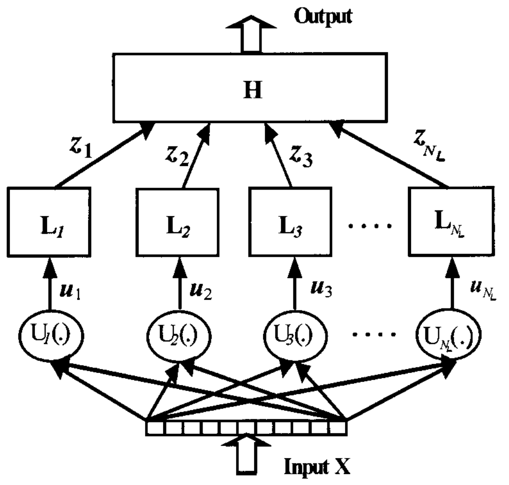


Figure 3. The hierarchical neural network structure. **X** and **Y** represent the inputs and outputs of the overall network. $\mathbf{L}_i$ is the ith low level module with an associated ith knowledge hub $\mathbf{U}_i(.)$. **u** and **z** represent the inputs and outputs of low-level modules.

The overall library development is summarized in the following steps:

Step 1. Define the input and output spaces of the base models, and extract basic characteristics from the library, using microwave empirical knowledge if available.

Step 2. Collect training data corresponding to each base model inputs and outputs.

Step 3. Construct and train base neural models incorporating the knowledge from Step 1.

Step 4. Take one unmodeled component from the library. According to the base model input space definition in Step 1, set up the structural knowledge hubs, which maps the model input space into base model input space. This automatically sets up the lower level modules.

Step 5. Collect training data corresponding to the model in the library.

Step 6. Preprocess the training data by propagating the inputs through knowledge hubs and lower level modules.

Step 7. Train the higher level neural module from preprocessed training data.

Step 8. If all done, then stop, otherwise proceed to train the next library model and go to Step 4.

The algorithm described above permits the hierarchical neural models to be developed systematically, and enables the library development process to be maximally automated. Examples of transmission line neural model libraries, useful for the design of high-speed VLSI interconnects, were developed in [65]. Compared to standard neural model techniques, the hierarchical neural network approach substantially reduces the cost of library development through reduced need for data generation and shortened time of training, while yielding reliable neural models.

D. 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 are reviewed, with emphasis on wavelet networks which is 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 [66–68]. 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,

$$\bar{y}_i^1 = \psi(d_i(\mathbf{x} - \mathbf{T}_i)), \qquad i = 1, \ldots, N_1, \quad (22)$$

where $\psi(.)$ is the radial type mother wavelet function, d_i are dilation parameters, and $\mathbf{T}_i$ are translation vectors for the ith hidden wavelet neuron. Both d_i and $\mathbf{T}_i$ are adapted together with ν_{ki} and β_k of Eq. (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 d_i, $\mathbf{T}_i$, and ν_{ki}, and the decomposition formula. As such, the initial values of network parameters can be estimated from the training data using decomposition formulas. However, if the initialization uses regularly truncated wavelet frames [67], 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 [66] 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 RBF networks, since both of them have the localized basis functions. The difference is that the wavelet function is localized both in the input and frequency domains [68]. Besides retaining the advantage of faster training, wavelet networks have a guaranteed upper bound on the accuracy of approximation [69] with a multiscale structure. Wavelet networks have been used in nonparametric regression estimation [66] and were trained based on noisy observation data to avoid the problem of undesirable local minima [69]. In [14], wavelet networks and stepwise selection by orthogonalization regression technique were used to build a neural network model for a 2.9 GHz microwave power amplifier. In this technique, a library of wavelets was built according to the training data and the wavelet that best fits the training data was selected. Later in an iterative manner, wavelets in the remainder of the library that best fits the data in combination with the previously selected wavelets were selected. For computational efficiency, later selected wavelets were orthonormalized to earlier selected ones.

Besides wavelet network, there are a number of constructive neural network structures, the most representative among them being the cascade correlation network (CasCor) [70]. 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 multilayer network. For regression tasks, a CasPer algorithm [71] 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 [72] to alleviate this problem.

Another constructive neural network structure is the projection pursuit learning network (PPLN), which adds neurons with trainable activation functions one-by-one, within a single hidden layer without cascaded connections [73]. 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.

IV. TRAINING ALGORITHMS

A. Training Objective

A neural network model can be developed through a process called training. Suppose the training data consists of N_p sample pairs, $\{(\mathbf{x}_p, \mathbf{d}_p), p = 1, 2, \ldots, N_p\}$, 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 (12), and for KBNN $\mathbf{w}$ is given by (21).

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

$$\min_{w} E(\mathbf{w}), \tag{23}$$

where

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{p=1}^{N_p}\sum_{k=1}^{N_y}\left(y_{pk}(\mathbf{x}_p,\mathbf{w}) - d_{pk}\right)^2 \\ &= \frac{1}{2}\sum_{p=1}^{N_p} e_p(\mathbf{w}), \end{aligned} \tag{24}$$

and d_{pk} is the kth element of vector $\mathbf{d}_p$, $y_{pk}(\mathbf{x}_p, \mathbf{w})$ is the kth output of the neural network when the input presented to the network is $\mathbf{x}_p$. The term $e_p(\mathbf{w})$ is the error in the output due to the pth sample.

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 $\mathbf{w}_{\text{next}}$, is determined by a step down from the current point $\mathbf{w}_{\text{now}}$ along a direction vector $\mathbf{d}$,

$$\mathbf{w}_{\text{next}} = \mathbf{w}_{\text{now}} + \eta\mathbf{d}, \tag{25}$$

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.

B. Backpropagation Algorithm and Its Variants

One of the most popular algorithms for neural network training is the backpropagation (BP) algorithm [74], proposed by Rumelhart, Hinton, and Williams in 1986. The BP algorithm is a stochastic algorithm based on the steepest descent principle [75], wherein the weights of the neural network are updated along the negative gradient direction in the weight space. The updated formulas are given by,

$$\Delta\mathbf{w}_{\text{now}} = \mathbf{w}_{\text{next}} - \mathbf{w}_{\text{now}} = -\eta \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_{\text{now}}}, \tag{26a}$$

$$\Delta\mathbf{w}_{\text{now}} = \mathbf{w}_{\text{next}} - \mathbf{w}_{\text{now}} = -\eta \left.\frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_{\text{now}}}, \tag{26b}$$

where η called learning rate controls the step size of weight update. Update formula (26b) is called update sample-by-sample, where the weights are updated after each sample is presented to the network. Update formula (26a) 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 formulas in (26a) and (26b) as proposed by [74], provided significant improvements to the basic backpropagation, reducing the weight oscillation. Thus,

$$\Delta \mathbf{w}_{\text{now}} = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}_{\text{now}}} + \alpha \Delta \mathbf{w}_{\text{old}}$$
$$= -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}_{\text{now}}} + \alpha (\mathbf{w}_{\text{now}} - \mathbf{w}_{\text{old}}), \tag{27a}$$

$$\Delta \mathbf{w}_{\text{now}} = -\eta \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}_{\text{now}}} + \alpha \Delta \mathbf{w}_{\text{old}}$$
$$= -\eta \frac{\partial e_p(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}=\mathbf{w}_{\text{now}}} + \alpha (\mathbf{w}_{\text{now}} - \mathbf{w}_{\text{old}}), \tag{27b}$$

where α is the momentum factor which controls the influence of the last weight update direction on the current weight update, and $\mathbf{w}_{\text{old}}$ represents the last point of $\mathbf{w}$. This technique is also known as the generalized delta-rule [36]. Other approaches to reduce weight oscillation have also been proposed, such as invoking a correction term that uses the difference of gradients [76], and the constrained optimization approach where constraints on weights are imposed to achieve better alignment between weight updates in different epochs [77].

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 [78–80]. Some of the real-world applications involve large-scale networks, in which case the development of fast and efficient learning algorithms becomes extremely important [79]. 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 IV.C.

An important way to improve efficiency of training by backpropagation is to use adaptation schemes that allow the learning rate and the momentum factor to be adaptive during learning [36], e.g., adaptation according to training errors [81]. One of the most interesting works in this area is the delta-bar-delta rule proposed by Jacobs [82]. 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 [80, 83, 84] 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 [85], an enhanced backpropagation algorithm [86] 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 [87]. The algorithm in [87] 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 [88], 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 [76], 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 [75], 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 [89], and line search based on linear interpolation [90, 91]. One other way to improve training efficiency is the gradient reuse algorithm [92]. 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.

C. Training Algorithms Using Gradient-Based Optimization Techniques

The backpropagation based on the 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 the 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 [93, 94] with the development of second-order learning algorithms for neural networks. Papers [90, 95] reviewed the first- and second-order optimization methods for learning in feedforward neural networks.

Let **d** be the direction vector, η be the learning rate, $\mathbf{w}_{\text{now}}$ be the current value of **w**, then the optimization updates **w** such that,

$$E(\mathbf{w}_{\text{next}}) = E(\mathbf{w}_{\text{now}} + \eta\mathbf{d}) < E(\mathbf{w}_{\text{now}}). \quad (28)$$

The principal difference between various descent algorithms lies in the procedure to determine successive update directions (**d**) [96]. Once the update direction is determined, the optimal step size could be found by line search,

$$\eta^* = \min_{\eta > 0} \phi(\eta), \quad (29)$$

where

$$\phi(\eta) = E(\mathbf{w}_{\text{now}} + \eta\mathbf{d}). \quad (30)$$

When downhill direction **d** is determined from the gradient **g** of the objective function E, such descent methods are called gradient-based descent methods. The procedure for finding a gradient vector in a network structure is generally similar to backpropagation [74] in the sense that the gradient vector is calculated in the direction opposite to the flow of output from each neuron. For MLP as an example, this is done by means of a derivative chain rule starting from output layer,

$$\frac{\partial E}{\partial \mathbf{w}^l} = \frac{\partial E}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \bar{\mathbf{y}}^l} \cdot \frac{\partial \bar{\mathbf{y}}^l}{\partial \mathbf{w}^l}, \qquad l = L, \quad (31a)$$

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

$$\frac{\partial E}{\partial \mathbf{w}^l} = \frac{\partial E}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \bar{\mathbf{y}}^L} \cdots \frac{\partial \bar{\mathbf{y}}^{l+1}}{\partial \bar{\mathbf{y}}^l} \cdot \frac{\partial \bar{\mathbf{y}}^l}{\partial \mathbf{w}^l},$$
$$l = L-1, L-2, \ldots, 1, \quad (31b)$$

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

1. 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 iterations. With initial gradient $\mathbf{g}_{\text{initial}} = \partial E / \partial \mathbf{w}\,|_{\mathbf{w}=\mathbf{w}_{\text{initial}}}$, and direction vector $\mathbf{d}_{\text{initial}} = -\mathbf{g}_{\text{initial}}$, the conjugate gradient method recursively constructs two vector sequences [91],

$$\mathbf{g}_{\text{next}} = \mathbf{g}_{\text{now}} + \lambda_{\text{now}}\mathbf{H}\mathbf{d}_{\text{now}}, \quad (32)$$

$$\mathbf{d}_{\text{next}} = -\mathbf{g}_{\text{next}} + \gamma_{\text{now}}\mathbf{d}_{\text{now}}, \quad (33)$$

$$\lambda_{\text{now}} = \frac{\mathbf{g}_{\text{now}}^T\mathbf{g}_{\text{now}}}{\mathbf{d}_{\text{now}}^T\mathbf{H}\mathbf{d}_{\text{now}}}, \quad (34)$$

$$\gamma_{\text{now}} = \frac{\mathbf{g}_{\text{next}}^T\mathbf{g}_{\text{next}}}{\mathbf{g}_{\text{now}}^T\mathbf{g}_{\text{now}}}, \quad (35)$$

or,

$$\gamma_{\text{now}} = \frac{(\mathbf{g}_{\text{next}} - \mathbf{g}_{\text{now}})^T\mathbf{g}_{\text{next}}}{\mathbf{g}_{\text{now}}^T\mathbf{g}_{\text{now}}}, \quad (36)$$

where **d** is called the conjugate direction and **H** is the Hessian matrix of the objective function E. Here, (35) is called the Fletcher–Reeves formula and (36) is called the Polak–Ribiere formula. To avoid the need of the Hessian matrix to compute the conjugate direction, we proceed from $\mathbf{w}_{\text{now}}$ along the direction $\mathbf{d}_{\text{now}}$ to the local minimum of E at $\mathbf{w}_{\text{next}}$ through line minimization, and then set $\mathbf{g}_{\text{next}} = \partial E/\partial \mathbf{w}\,|_{\mathbf{w}=\mathbf{w}_{\text{next}}}$. This $\mathbf{g}_{\text{next}}$ can be used as the vector of (32), and as such (34) is no longer needed. We make use of this line minimization concept to find the 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. First, 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. Second, since for neural network training, the error function is not quadratic w.r.t. the variable as defined in (24), 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 [85], 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 [97] which avoids the line search per learning iteration by using the Levenberg–Marquardt approach to scale the step size.

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

$$\mathbf{w}_{\text{next}} = \mathbf{w}_{\text{now}} - \eta \mathbf{B}_{\text{now}} \mathbf{g}_{\text{now}}. \tag{37}$$

The $\mathbf{B}$ 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 [98],

$$\mathbf{B}_{\text{now}} = \mathbf{B}_{\text{old}} + \Delta\mathbf{B}_{\text{now}}. \tag{38}$$

There are two major rank 2 formulas to compute $\Delta\mathbf{B}_{\text{now}}$,

$$\Delta\mathbf{B}_{\text{now}} = \frac{\mathbf{d}\mathbf{d}^T}{\mathbf{d}^T\,\Delta\mathbf{g}} - \frac{\mathbf{B}_{\text{old}}\Delta\mathbf{g}\,\Delta\mathbf{g}^T\,\mathbf{B}_{\text{old}}}{\Delta\mathbf{g}^T\,\mathbf{B}_{\text{old}}\,\Delta\mathbf{g}}, \tag{39}$$

or,

$$\Delta\mathbf{B}_{\text{now}} = \left(1 + \frac{\Delta\mathbf{g}^T\mathbf{B}_{\text{old}}\,\Delta\mathbf{g}}{\mathbf{d}^T\,\Delta\mathbf{g}}\right)\frac{\mathbf{d}\mathbf{d}^T}{\mathbf{d}^T\,\Delta\mathbf{g}} - \frac{\mathbf{d}\Delta\mathbf{g}^T\,\mathbf{B}_{\text{old}} + \mathbf{B}_{\text{old}}\,\Delta\mathbf{g}\mathbf{d}^T}{\mathbf{d}^T\,\Delta\mathbf{g}}, \tag{40}$$

where

$$\mathbf{d} = \mathbf{w}_{\text{now}} - \mathbf{w}_{\text{old}}, \qquad \Delta\mathbf{g} = \mathbf{g}_{\text{now}} - \mathbf{g}_{\text{old}}, \tag{41}$$

(39) is called the DFP (Davidon–Fletcher–Powell) formula and (40) is called 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 [99] a second-order learning algorithm is proposed based on a LM BFGS update. A reasonably accurate step size is efficiently calculated in a 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 [100]. Wavelet neural networks trained by the BFGS algorithm are used for the modeling of large-signal hard-nonlinear behavior of power transistors in circuit design [14]. The quasi-Newton training algorithm that employs the exact line search possesses the quadratic termination property. Through the estimation of the inverse Hessian matrix, quasi-Newton has a faster convergence rate than the conjugate gradient method.

3. Levenberg–Marquardt and Gauss-Newton Training Algorithms. Neural network training is usually formulated as a nonlinear least-squares problem. Methods dedicated to least-squares such as

Gauss–Newton can be employed to estimate the neural model parameters. The Gauss–Newton method is a linearization method. Let **r** be a vector containing the individual error terms in (24). Let **J** be the $(N_p \times N_y) \times N_w$ Jacobian matrix including the derivative of **r** w.r.t. **w**. **J** has N_w columns and $N_p \times N_y$ rows, where N_p is the number of samples and N_y is the number of outputs. The Gauss–Newton update formula can be expressed as,

$$\mathbf{w}_{\text{next}} = \mathbf{w}_{\text{now}} - (\mathbf{J}_{\text{now}}^T \mathbf{J}_{\text{now}})^{-1} \mathbf{J}_{\text{now}}^T \mathbf{r}_{\text{now}}. \quad (42)$$

In the preceding formula (42) $\mathbf{J}_{\text{now}}^T \mathbf{J}_{\text{now}}$ is positive definite unless $\mathbf{J}_{\text{now}}$ is rank deficient. The Levenberg–Marquardt [101] method can be applied when $\mathbf{J}_{\text{now}}$ is rank deficient and the corresponding weight update is given by,

$$\mathbf{w}_{\text{next}} = \mathbf{w}_{\text{now}} - (\mathbf{J}_{\text{now}}^T \mathbf{J}_{\text{now}} + \mu \mathbf{I})^{-1} \mathbf{J}_{\text{now}}^T \mathbf{r}_{\text{now}}, \quad (43)$$

where μ is a nonnegative number. A modified Levenberg–Marquardt training algorithm using a diagonal matrix instead of the identity matrix **I** in (43) was proposed [102] for the efficient training of multilayer feedforward neural networks. In [103], to reduce the size of the Jacobian matrix, the training samples are divided into several groups called local batches. The training is performed successively through these local batches. The computational requirement and memory complexity of the Levenberg–Marquardt methods were reduced by utilizing the deficient Jacobian matrix [104]. A combined Levenberg–Marquardt and quasi-Newton training technique was used in [1] to train the KBNN structure. When training parameters are far from the local minimum of the error surface Levenberg–Marquardt algorithm was used, and when they are close to local minimum quasi-Newton was used for faster convergence.

It has been proved that the conjugate gradient method is equivalent to error backpropagation with momentum term [90]. The theoretical convergence rate and practical performance of second-order gradient-based methods are generally superior to the first-order methods.

D. 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 the 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 [105–107]. In [106], the weights (**V**) of the output layer and the output vector ($\mathbf{y}^L$) of the previous layer are treated as two sets of variables. An optimal solution pair ($\mathbf{V}, \mathbf{y}^L$) is first determined to minimize the sum-squared-error between the desired neural network outputs and the actual outputs. The current solution $\mathbf{y}^L$ 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 the continuous function approximation, [107] 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 [108]. 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 special considerations. In [105], a layer-by-layer optimization of a 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 nonlinearity of the error function with respect to the hidden layer weights and decrease the chances of being trapped in a local minima. As such, a hybrid training algorithm would be favorable for a large-scale problem [52]. 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 [109] or BFGS [52], while employing a linear least mean square error (LMS) algorithm to compute the optimum linear output layer weights.

RBF networks are usually trained by the 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 [110]. The output layer weights can then be obtained through the 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 it's 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 [52] integrates the best features of the linear LMS algorithm of RBF, and the nonlinear optimization techniques of MLP into one routine. This routine could be suitable for any feedforward network 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 [52].

D. 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 [111]; genetic algorithms which evolve the structure and weights of the neural network through generations in a manner that is similar to biological evolution [112]; the Langevin updating (LV) rule in multilayer perceptron, in which noise is added to the weights during training [113]; and a stochastic minimization algorithm for training neural networks [114], 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., [89]. 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 the 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 the conjugate gradient algorithm.

V. EXAMPLES

A. 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 W, doping density N_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 [115]. Three sets of training data with 100, 300, and 500 samples, and one set of test data with 413 samples were used. The model accuracy is shown in Table I.

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 a 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 the available amount of training data becomes less.

The most popular training algorithms for standard feedforward neural networks are adaptive backpropagation, conjugate gradient, quasi-Newton, and Levenberg−Marquardt. Two examples, i.e., 3-conductor microstrip line and physics-based MESFET, were used to illustrate the performance of these algorithms.

For the microstrip line example, there are five input neurons corresponding to conductor width (w), spacing between conductors (s_1, s_2), sub-

TABLE I. Model Accuracy Comparison Between Multilayer Perceptrons and Radial Basis Functions

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

strate height (h), and relative permittivity (ε_r) as shown in Figure 4. There are six output neurons corresponding to the self inductance of each conductor l_{11}, l_{22}, l_{33} and the mutual inductance between any two conductors l_{12}, l_{23}, l_{13}. There are totally 600 training samples and 640 test samples generated by LINPAR [116]. A three-layer MLP structure with 28 hidden neurons is chosen as the sample structure. The training results are shown in Table II. CPU time is given for Pentium (200 MHz).

The total CPU time used by the Levenberg–Marquardt method is around 20 min. The adaptive backpropagation used many epochs and settled down to good accuracy of 0.252%, with around 4 h of training time. On the contrary, the quasi-Newton method achieved similar accuracy only within 35 min. This confirms the faster convergence rate of the second-order method. Usually quasi-Newton has a very fast convergence

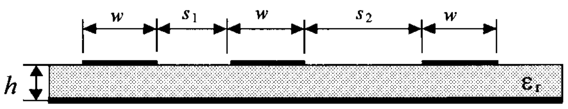


Figure 4. A 3-conductor microstrip line.

rate when approaching the minimum. However, at the beginning of training, its performance may not be very strong. Another strategy is to use the conjugate gradient method at the first stage of training, then followed by the quasi-Newton method. If we take the MLP network already trained by the conjugate gradient and then continue training by the quasi-Newton method, the model test error was reduced to 0.167%. Total training time is around 2 h.

For the MESFET example, the inputs to the neural model include frequency (f), channel thickness (a), gate-bias voltage (V_g), and drain-bias voltage (V_d). This particular MESFET has a fixed gate length of 0.55 μ and a 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 are obtained using the simulator OSA90 [115] with the Katibzadeh and Trew model. This is a relatively complicated example compared to the microstrip line example. The sample neural network structure has 60 hidden neurons. The training results by the four training algorithms are shown in Table III. This example demonstrates that as the size of the neural network becomes large, Levenberg–Marquardt becomes

TABLE II. Comparison of Various Training Algorithms for the Microstrip Line Example

Training Algorithm	No. of Epochs	Training Error (%)	Avg. Test Error (%)	CPU (s)
Adaptive backpropagation	10,755	0.224	0.252	13,724
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 III. Comparison of Various Taining Algorithms for the MESFET Example

Training Algorithm	No. of Epochs	Training Error (%)	Avg. Test Error (%)	CPU (s)
Adaptive backpropagation	15,319	0.98	1.04	11,245
Conjugate gradient	1605	0.99	1.04	4391
Quasi-Newton	570	0.88	0.89	1574
Levenberg–Marquardt	12	0.97	1.03	4322

slow as compared to the quasi-Newton method, due to repeated inversion of the large matrix in (43).

B. Knowledge-Based Neural Network and Training

This example demonstrates the knowledge based neural network (KBNN) described in Section III.B in modeling cross-sectional resistance-inductance-capacitance-conductance (RLCG) parameters of transmission lines for analysis of high-speed VLSI interconnects [16] and its comparison with traditional multilayer perceptrons (MLP). The output of the model is the cross-sectional mutual inductance (per unit length), 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, and frequency x_6. There exist mutual inductance empirical formula, e.g., [53],

$$l_{12} = \frac{\mu_r \mu_0}{4\pi} \ln\left[1 + \frac{(2x_4)^2}{(x_1 + x_3)^2}\right]. \quad (44)$$

This equation becomes the knowledge to be incorporated into the knowledge neurons of Section III.B as,

$$\begin{aligned} z_i &= \psi_i(\mathbf{x}, \mathbf{w}_i) \\ &= \ln\left[1 + e^{w_{i1}} \frac{(x_4 - w_{i2})^2}{(x_1 + x_3 - w_{i3})^2}\right] + w_{i4}x_2 \\ &\quad + w_{i5}x_5 + w_{i6}x_6 + w_{i7}, \\ &\qquad i = 1, 2, \ldots, N_z. \quad (45) \end{aligned}$$

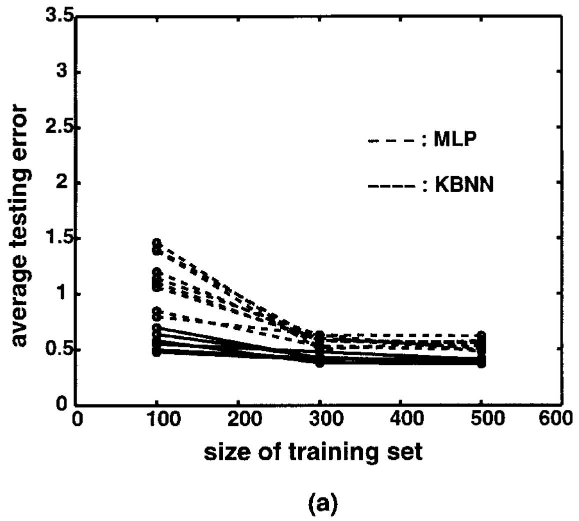
Linear boundary neurons were used in the layer **B**. Notice that this empirical formula is incorporated multiple times (N_z times), each with different values of **w**, ($\mathbf{w}_i$, $i = 1, 2, \ldots, N_z$). KBNN provides a complete–integrated $\mathbf{x} - \mathbf{y}$ relationship including those not available in the original empirical formula (e.g., y with respect to x_2, x_5, x_6).

Five sets of data were generated by EM simulation [117]. The first three sets have 100, 300, and 500 samples and were used for training purposes. The fourth set of another 500 samples was generated in the same range as the first three sets to test the trained neural models. These testing data were never used in training. The last set of data with 4096 samples were deliberately selected around–beyond the boundary of the model effective region in the input parameter space in order to compare extrapolation accuracy of KBNN and MLP.

Two KBNNs of different sizes were trained and compared with three MLPs (with the number of hidden neurons being 7, 15, and 20). All these neural networks were trained by the Levenberg–Marquardt algorithm. Figure 5 shows the error from individual trainings of KBNN and MLP in terms of the average testing error. 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. A much more stable performance of KBNN over MLP is observed when making an extrapolation prediction.

C. Hierarchical Neural Network and Its Training

This example demonstrates the usage of hierarchical neural networks in the development of a library of neural models for N-conductor striplines shown in Figure 6 for different values of N [65]. In the example there are five models in the library, $n = 1, 2, 3, 4, 5$. In addition, for each nth model, $N = n$. Table IV defines the inputs and outputs of each model.



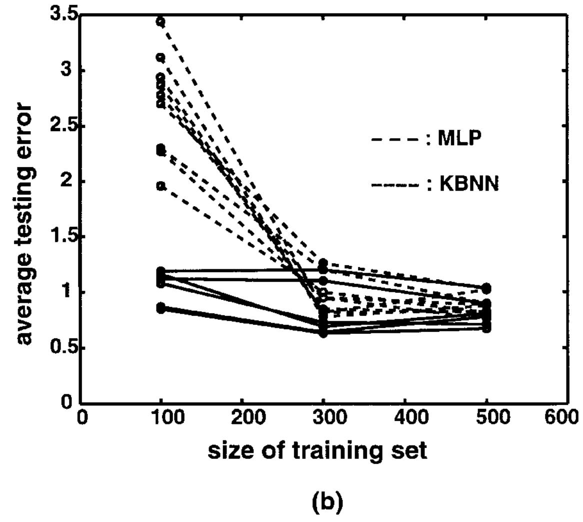


Figure 5. Comparison of error from individual trainings of KBNN and MLP in terms of the average testing error.

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 (g), substrate height (h), separation between conductors (s), and relative dielectric constant(ε_r). The outputs of B_1 and B_2 are self and mutual inductances, respectively. The training strategy for base models is to use the conjugate gradient at the first stage and followed by the quasi-Newton method. The base models B_1 and B_2 are trained to an average testing accuracy of 0.39 and 0.16%, respectively.

Example of Library Model: $n = 3$. For library model $n = 3$, we reuse the base models as the low-level neural modules shown in Figure 3. The high-level neural module is realized by a two-layer perceptron with six inputs and six outputs. Only a small amount of training data (15 samples) is needed to train this 3-conductor stripline model since the raw–fundamental relationships of the model have already been captured in the base models. Since this is a linear network, the training problem is actually a quadratic minimization. The conjugate gradient method was used here to find the unique global minimum. However, with the conventional MLP neural model, even 500 sam-

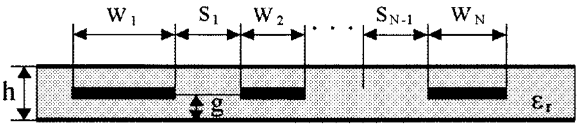


Figure 6. N-conductor stripline component. For the nth component in the stripline library, $N = n$.

TABLE IV. Stripline Library Components

Library Component	Component Name	Neural Model Inputs	Neural Model Outputs	Reuse of Base Models (B_1 and B_2)
$n = 1$	1-conductor model	$w g h\ \varepsilon_r$	L_{11}	B_1
$n = 2$	2-conductor model	$w_1\ w_2\ s g h\ \varepsilon_r$	L_{11}, L_{12}, L_{22}	$2 \times B_1, 1 \times B_2$
$n = 3$	3-conductor model	$w_1\ w_2\ w_3\ s_1\ s_2\ g h\ \varepsilon_r$	$L_{11}, L_{12}, L_{13}, L_{22}, L_{23}, L_{33}$	$3 \times B_1, 3 \times B_2$
$n = 4$	4-conductor model	$w_1\ w_2\ w_3\ w_4\ s_1\ s_2\ s_3\ g h\ \varepsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{22}, L_{23}, L_{24}, L_{33}, L_{34}, L_{44}$	$4 \times B_1, 6 \times B_2$
$n = 5$	5-conductor model	$w_1\ w_2\ w_3\ w_4\ w_5\ s_1\ s_2\ s_3\ s_4\ g h\ \varepsilon_r$	$L_{11}, L_{12}, L_{13}, L_{14}, L_{15}, L_{22}, L_{23}, L_{24}, L_{25}, L_{33}, L_{34}, L_{35}, L_{44}, L_{45}, L_{55}$	$5 \times B_1, 10 \times B_2$

ples are not enough to achieve a model of similar accuracy, as shown in Figure 7.

All Library Models. All library models in the library can be developed systematically in a similar way as model 3. It should be noted that efforts in developing those additional models are small and incremental, since only few training data are needed, and only the high-level neural modules need to be trained.

Overall Library Accuracy and Development Cost. A Comparison. Using standard MLP for each model, the total training time for all library models is 6 h 30 min on SparcStation 5. Using the hierarchical approach, the total training time is 36 min. The total amount of training data needed by standard MLP is 2664 samples and by the Hierarchical neural network approach is only 649 (including 564 samples for the base model, and 85 samples for subsequent library models) as shown in Table V. The hierarchical neural network structure yields reliable neural models even with a very small amount of training data.

VI. CONCLUSIONS

Neural network technology is an emerging technology in the microwave area for microwave modeling, simulation, optimization, and design. The efficient development of an accurate neural model requires a proper neural network structure and suitable training algorithms, two important as-

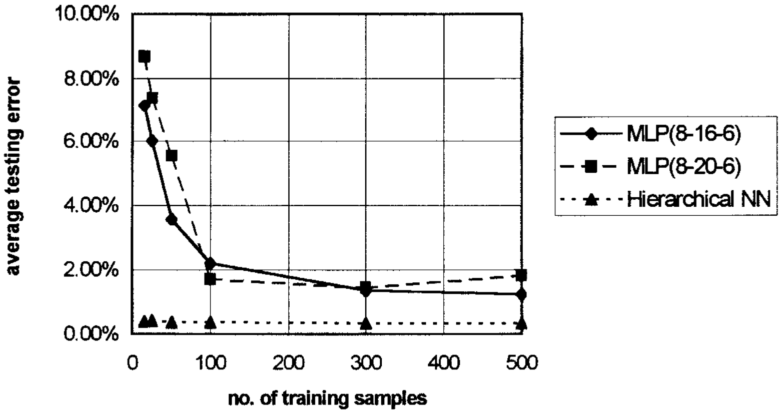


Figure 7. Model accuracy comparison (average error on test data) between standard MLP and the hierarchical neural network models for the 3-conductor stripline component.

TABLE V. Comparison of the Number of Training Samples Needed and Model Accuracy for the Stripline Library when Developed by Standard Multilayer Perceptrons and the Hierarchical Neural Network Structure, respectively

Stripline Component Name	No. of Training Samples Needed		Model Accuracy (Average Error on Test Data) %	
	Standard MLP	Hierarchical NN	Standard MLP	Hierarchical NN
Overhead	0	$264^1 + 300^2$		
1-conductor model	264	0	0.42	0.39
2-conductor model	300	10	1.01	0.56
3-conductor model	500	15	1.25	0.40
4-conductor model	700	25	1.30	0.79
5-conductor model	900	35	0.99	0.63
Library	Total = 2664	Total = 649	Average = 0.99	Average = 0.55

^{1}Base model B_1 training.
^{2}Base model B_2 training.

pects in successful applications of neural networks in solving microwave design problems. This paper presented a review of the current status of this area. The subject of neural network architectures and training algorithms in general is large and we have omitted many advances suitable for other applications such as in signal processing, pattern recognition, and so on. Instead, our paper focuses on the structures and algorithms which we feel are relevant or useful to RF and microwave applications. Standard feedforward neural networks, neural network structures with prior knowledge, combining neural networks and constructive network structures, are described. Also discussed are various training algorithms including the backpropagation algorithm and its variants, training algorithms based on classical optimization techniques such as conjugate gradient and quasi-Newton algorithms, training algorithms based on decomposed optimization, and global minimization techniques.

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 a reliable neural model with a very small amount of data remains an important research. In many practical cases, training data from simulation−measurement contains 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 designs is already underway. The highly nonlinear, and nonsmooth 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 the requirement of more training data, and higher accuracy is difficult to obtain. Another scenario leading to the same challenge is 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. Another significant milestone in this area would be to incorporate the microwave-oriented features and techniques of this emerging 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 powerful vehicles helping microwave design today and tomorrow.

ACKNOWLEDGMENTS

The authors thank Marco Antonio Pego Guerra of Carleton University for his cooperation in running *NeuroModeler* [118] for Example 1. This work was supported through the Natural Sciences and Engineering Research Council of Canada, and Micronet: a Canadian Network Centres of Excellence on Microelectronic Devices, Circuits and Systems. The authors are with the Department of Electronics, Carleton University, 1125 Colonel By Drive, Ottawa, Canada K1S 5B6.

REFERENCES

1. F. Wang and Q. J. Zhang, Knowledge based neural models for microwave design, IEEE Trans Microwave Theory Tech 45 (1997), 2333−2343.
2. A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, Neural network modeling approach to circuit optimization and statistical design, IEEE Trans Microwave Theory Tech 43 (1995), 1349−1358.
3. P. M. Watson and K. C. Gupta, EM-ANN models for microstrip vias and interconnects in dataset circuits, IEEE Trans Microwave Theory Tech 44 (1996), 2495−2503.

4. P. M. Watson and K. C. Gupta, Design and optimization of CPW circuits using EM-ANN models for CPW components, IEEE Trans Microwave Theory Tech 45 (1997), 2515–2523.
5. G. L. Creech, B. J. Paul, C. D. Lesniak, T. J. Jenkins, and M. C. Calcatera, Artificial neural networks for accurate microwave CAD applications, IEEE Int Microwave Symp Digest, San Francisco, CA, 1996, pp. 733–736.
6. M. Vai, S. Wu, B. Li, and S. Prasad, Creating neural network based microwave circuit models for analysis and synthesis, Proc Asia Pacific Microwave Conf, Hong Kong, Dec. 1997, pp. 853–856.
7. M. Vai and S. Prasad, Microwave circuit analysis and design by a massively distributed computing network, IEEE Trans Microwave Theory Tech 43 (1995), 1087–1094.
8. Q. J. Zhang and M. S. Nakhla, Signal integrity analysis and optimization of VLSI interconnects using neural network models, IEEE Int Symp Circuits Systems, London, U.K., 1994, pp. 459–462.
9. P. M. Watson, K. C. Gupta, and R. L. Mahajan, Development of knowledge based artificial neural network models for microwave components, IEEE Int Microwave Symp Digest, Baltimore, MD, 1998, pp. 9–12.
10. G. L. Creech, B. J. Paul, C. D. Lesniak, T. J. Jenkins, and M. C. Calcatera, Artificial neural networks for fast and accurate EM-CAD of microwave circuits, IEEE Trans Microwave Theory Tech 45 (1997), 794–802.
11. V. B. Litovski, J. I. Radjenovic, Z. M. Mrcarica, and S. L. Milenkovic, MOS transistor modeling using neural network, Electron Lett 28 (1992), 1766–1768.
12. A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, Analysis and optimization of microwave circuits and devices using neural network models, IEEE Int Microwave Symp Digest, San Diego, CA, May 1994, pp. 393–396.
13. G. Kothapalli, Artificial neural networks as aids in circuit design, Microelectron J 26 (1995), 569–578.
14. Y. Harkouss, J. Rousset, H. Chehade, D. Barataud, and J. P. Teyssier, Modeling microwave devices and circuits for telecommunications system design, Proc IEEE Int Conf Neural Networks, Anchorage, AK, May 1998, pp. 128–133.
15. A. Veluswami, M. S. Nakhla, and Q. J. Zhang, The application of neural networks to EM based simulation and optimization of interconnects in high speed VLSI circuits, IEEE Trans Microwave Theory Tech 45 (1997), 712–723.
16. Q. J. Zhang, F. Wang, and M. S. Nakhla, Optimization of high-speed VLSI interconnects: A review, Int J Microwave Millimeter-Wave CAD 7 (1997), 83–107.
17. T. Horng, C. Wang, and N. G. Alexopoulos, Microstrip circuit design using neural networks, IEEE Int Microwave Symp Digest, Altanta, GA, 1993, pp. 413–416.
18. P. Burrascano, M. Dionigi, C. Fancelli, and M. Mongiardo, A neural network model for CAD and optimization of microwave filters, IEEE Int Microwave Symp Digest, Baltimore, MD, 1998, pp. 13–16.
19. M. D. Baker, C. D. Himmel, and G. S. May, In-situ prediction of reactive ion etch endpoint using neural networks, IEEE Trans Comp Packag, Manufact Technol A 18 (1995), 478–483.
20. M. Vai and S. Prasad, Automatic impedance matching with a neural network, IEEE Microwave Guided Wave Lett 3 (1993), 353–354.
21. R. Biernacki, J. W. Bandler, J. Song, and Q. J. Zhang, Efficient quadratic approximation for statistical design, IEEE Trans Circuits Syst CAS-36 (1989), 1449–1454.
22. P. Meijer, Fast and smooth highly nonlinear multidimensional table models for device modelling, IEEE Trans Circuits Syst 37 (1990), 335–346.
23. F. Scarselli and A. C. Tsoi, Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results, Neural Networks 11 (1998), 15–37.
24. B. Widrow and M. A. Lehr, 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, IEEE Proc 78 (1990), 1415–1442.
25. M. Minskey and S. Papert, Perceptrons, MIT Press, Cambridge, MA, 1969.
26. G. Cybenko, Approximation by superpositions of a sigmoidal function, Math Control Signals Syst 2 (1989), 303–314.
27. K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989), 359–366.
28. S. Tamura and M. Tateishi, Capabilities of a four-layered feedforward neural network: four layer versus three, IEEE Trans Neural Networks, 8 (1997), 251–255.
29. J. de Villiers and E. Barnard, Backpropagation neural nets with one and two hidden layers, IEEE Trans Neural Networks 4 (1993), 136–141.
30. F. Girosi, Regularization theory, radial basis functions and networks, From statistics to neural networks: Theory and pattern recognition applications, V. Cherkassy, J.H. Firedman, and H. Wechsler (Editors), Springer-Verlag, Berlin/New York, 1992, pp. 166–187.
31. M. J. D. Powell, Radial basis functions for multivariate interpolation: a review, Algorithms for approximation, J.C. Mason and M.G. Cox (Editors), Oxford Univ. Press, Oxford, U.K., 1987, pp. 143–167.
32. J. Park and I. W. Sandberg, Universal approximation using radial-basis-function networks, Neural Comput 3 (1991), 246–257.

33. J. Park and I. W. Sandberg, Approximation and radial-basis-function networks, Neural Comput 5 (1991), 305–316.
34. A. Krzyzak, T. Linder, and G. Lugosi, Nonparametric estimation and classification using radial basis function nets and empirical risk minimization, IEEE Trans Neural Networks 7 (1996), 475–487.
35. W. Kaminski and P. Strumillo, Kernel orthonormalization in radial basis function neural networks, IEEE Trans Neural Networks 8 (1997), 1177–1183.
36. S. Haykin, Neural networks: A comprehensive foundation, IEEE Press, Piscataway, NJ, 1994.
37. J. Moody and C. J. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput 1 (1989), 281–294.
38. T. Y. Kwok and D. Y. Yeung, Constructive algorithms for structure learning in feedforward neural networks for regression problems, IEEE Trans Neural Networks 8 (1997), 630–645.
39. R. Reed, Pruning algorithms—a survey, IEEE Trans Neural Networks 4 (1993), 740–747.
40. A. Krzyzak and T. Linder, Radial basis function networks and complexity regularization in function learning, IEEE Trans Neural Networks 9 (1998), 247–256.
41. H. Leung and S. Haykin, Rational function neural network, Neural Comput 5 (1993), 928–938.
42. G. G. Towell and J. W. Shavlik, Knowledge-based artificial neural networks, Artif Intell 70 (1994), 119–165.
43. R. Maclin and J. W. Shavlik, Using knowledge-based neural networks to improve algorithms: refining the Chou–Fasman algorithm for protein folding, Mach Learn 11 (1993), 195–215.
44. L. M. Fu, Integration of neural heuristics into knowledge-based inference, Connection Sci 1 (1989), 325–340.
45. R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, Back-propagation learning in expert networks, IEEE Trans Neural Networks 3 (1992), 67–72.
46. J. J. Mahoney and R. J. Mooney, Combining connectionist and symbolic learning to refine certainty factor rule bases, Connection Sci 5 (1993), 339–364.
47. W. J. Jansen, M. Diepenhorst, J. A. G. Nijhuis, and L. Spaanenburg, Assembling engineering knowledge in a modular multi-layer perceptron neural network, Proc IEEE Int Conf Neural Networks, Houston, TX, June 1997, pp. 232–237.
48. G. G. Lendaris, A. Rest and T. R. Misley, Improving ANN generalization using a priori knowledge to pre-structure ANNs, Proc IEEE Int Conf Neural Networks, Houston, TX, June 1997, pp. 248–253.
49. T. A. Johansen and B. A. Foss, Constructing NARMAX models using ARMAX models, Int J Control 58 (1993), 1125–1153.
50. T. A. Johansen and B. A. Foss, Identification of non-linear system structure and parameters using regime decomposition, Automatica 31 (1995), 321–326.
51. O. Nelles, S. Sinsel, and R. Isermann, Local basis function networks for identification of a turbocharger, Proc UKACC Int Conf Control, Exeter, U.K., July 1996, pp. 7–12.
52. S. Mcloone, M. D. Brown, G. Irwin, and G. Lightbody, A hybrid linear/nonlinear training algorithm for feedforward neural networks, IEEE Trans Neural Networks 9 (1998), 669–684.
53. C. S. Walker, Capacitance, inductance and crosstalk analysis, Artech House, Boston, MA, 1990.
54. P. H. Ladbrooke, MMIC design: GaAs FET's and HEMTs, Artech House, Boston, MA, 1989.
55. J. R. Tsai, P. C. Chung, and C. I. Chang, A sigmoidal radial basis function neural network for function approximation, Proc IEEE Int Conf Neural Networks, Washington, DC, June 1996, pp. 496–501.
56. A. Abdelbar and G. Tagliarini, Honest: A new high order feedforward neural networks, Proc IEEE Int Conf Neural Networks, Washington, DC, June 1996, pp. 1257–1262.
57. A. J. Sharkey, On combining artificial neural nets, Connection Sci 8 (1996), 299–314.
58. S. Hashem and B. Schmeiser, Improving model accuracy using optimal linear combinations of trained neural networks, IEEE Trans Neural Networks 6 (1995), 792–794.
59. S. Hashem, Algorithms for optimal linear combinations of neural networks, Proc IEEE Int Conf Neural Networks, Houston, TX, June 1997, pp. 242–247.
60. S. Hashem, Optimal linear combinations of neural networks, Neural Networks 10 (1997), 599–614.
61. A. J. Sharkey, Modularity, combining and artificial neural nets, Connection Sci 9 (1997), 3–10.
62. R. A. Jacobs, F. Peng, and M. A. Tanner, A Bayesian approach to model selection in hierarchical mixtures-of-experts architectures, Neural Networks 10 (1997), 231–241.
63. G. Auda, M. Kamel, and H. Raafat, A new neural network structure with cooperative modules, IEEE Int Conf Neural Networks, Orlando, FL, June 1994, pp. 1301–1306.
64. L. C. Wang, N. M. Nasrabadi, and S. Der, Asymptotical analysis of a modular neural network, Proc IEEE Int Conf Neural Networks, Houston, TX, June 1997, pp. 1019–1022.
65. 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 Int Microwave Symp Digest, Baltimore, MD, 1998, pp. 1767–1770.

66. Q. H. Zhang, Using wavelet network in nonparametric estimation, IEEE Trans Neural Networks 8 (1997), 227–236.
67. Q. H. Zhang and A. Benvensite, Wavelet networks, IEEE Trans Neural Networks 3 (1992), 889–898.
68. B. R. Bakshi and G. Stephanopoulos, Wave-net: a multiresolution, hierarchical neural network with localized learning, Amer Inst Chemical Eng J 39 (1993), 57–81.
69. B. Delyon, A. Juditsky, and A. Benveniste, Accuracy analysis for wavelet approximations, IEEE Trans Neural Networks 6 (1995), 332–348.
70. S. E. Fahlman and C. Lebiere, The cascade-correlation learning architecture, Advances in neural information processing D.S. Truretzky (Editors), Vol. 2, Morgan Kauffman, San Mateo, CA, 1990, pp. 524–532.
71. N. K. Treadgold and T. D. Gedeon, Extending CasPer: a regression survey, Int Conf Neural Information Processing, 1997, pp. 310–313.
72. N. K. Treadgold and T. D. Gedeon, Exploring architecture variation in constructive cascade networks, Proc IEEE Int Conf Neural Networks, Anchorage, AK, May 1998, pp. 343–348.
73. 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 7 (1996), 278–289.
74. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation, Parallel distributed processing, Vol. I, D.E. Rumelhart and J.L. McClelland (Editors), Vol. I, MIT Press, Cambridge, MA, 1986, pp. 318–362.
75. D. G. Luenberger, Linear and nonlinear programming, Addison-Wesley, Reading, MA, 1989.
76. K. Ochiai, N. Toda, and S. Usui, Kick-out learning algorithm to reduce the oscillation of weights, Neural Networks 7 (1994), 797–807.
77. S. J. Perantonis and D. A. Karras, An efficient constrained learning algorithm with momentum acceleration, Neural Networks 8 (1995), 237–249.
78. N. B. Karayiannis and A. N. Venetsanopoulos, Artificial neural networks: learning algorithms, performance evaluation, and applications, Kluwer Academic, Boston, MA, 1993.
79. N. Morgan, Big dumb neural nets: a working brute force approach to speech recognition Proc IEEE Int Conf Neural Networks, vol. VII, Orlando, FL, July 1994, pp. 4462–4465.
80. F. Wang and Q. J. Zhang, A sparse matrix approach to neural network training, Proc IEEE Int Conf Neural Networks, Perth, Australia, Nov. 1995, pp. 2743–2747.
81. Neural network toolbox: For use with Matlab, MathWorks, Natick, MA, 1993.
82. R. A. Jacobs, Increased rate of convergence through learning rate adaptation, Neural Networks 1 (1988), 295–307.
83. A. H. Zaabab, Q. J. Zhang, and M. S. Nakhla, Device and circuit-level modeling using neural networks with faster training based on network sparsity, IEEE Trans Microwave Theory Tech 45 (1997), 1696–1704.
84. F. Wang and Q. J. Zhang, An adaptive and fully sparse training approach for multilayer perceptrons, Proc. IEEE Int Conf Neural Networks, Washington, DC, June 1996, pp. 102–107.
85. R. Battiti, Accelerated backpropagation learning: two optimization methods, Complex Syst 3 (1989), 331–342.
86. M. Arisawa and J. Watada, Enhanced back-propagation learning and its application to business evaluation, Proc IEEE Intl Conf Neural Networks, Vol. I, Orlando, Florida, July 1994, pp. 155–160.
87. 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 5 (1994), 493–497.
88. X. H. Yu, G. A. Chen, and S. X. Cheng, Dynamic learning rate optimization of the backpropagation algorithm, IEEE Trans Neural Networks 6 (1995), 669–677.
89. 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 7 (1994), 1253–1265.
90. P. P. Van-Der-Smagt, Minimisation methods for training feedforward neural networks," Neural Networks 7 (1994), 1–11.
91. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, Numerical recipes: The art of scientific computing, Cambridge Univ. Press, Cambridge, MA, 1986.
92. D. R. Hush and J. M. Salas, Improving the learning rate of back propagation with the gradient reuse algorithm, Proc IEEE Int Conf Neural Networks, Vol. I San Diego, CA, 1988, pp. 441–447.
93. D. B. Parker, Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation and second order hebbian learning, Proc IEEE First Int Conf Neural Networks, Vol. II, San Diego, CA, 1987, pp. 593–600.
94. R. L. Watrous, Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization, Proc IEEE First Int Conf Neural Networks, Vol. II, San Diego, CA, 1987, pp. 619–627.
95. R. Battiti, First and second-order methods for learning: between steepest descent and Newton's method, Neural Comput 4 (1992), 141–166.

96. J.-S.R. Jang, C.-T. Sun, and E. Mizutani, Derivative-based optimization, Neuro-fuzzy and soft computing, a computational approach to learning and Machine intelligence, Prentice-Hall, Upper Saddle River, NJ, 1997, pp. 129-172.
97. M. F. Moller, A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks 6 (1993), 525–533.
98. T. R. Cuthbert Jr., Quasi-Newton methods and constraints, Optimization using personal computers John Wiley & Sons, New York, 1987, pp. 233–314.
99. K. R. Nakano, Partial BFGS update and efficient step-length calculation for three-layer neural network, Neural Comput 9 (1997), 123–141.
100. S. McLoone and G. W. Irwin, Fast parallel off-line training of multilayer perceptrons, IEEE Trans Neural Networks 8 (1997), 646-653.
101. A. J. Shepherd, Second-order optimization methods, Second-order methods for neural networks, Springer-Verlag, Berlin/New York, 1997, pp. 43–72.
102. S. Kollias and D. Anastassiou, An adaptive least squares algorithm for the efficient training of artificial neural networks, IEEE Trans Circuits Syst 36 (1989), 1092–1101.
103. M. G. Bello, Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks, IEEE Trans Neural Networks 3 (1992), 864–875.
104. G. Zhou and J. Si, Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency, IEEE Trans Neural Networks 9 (1998), 448–453.
105. S. Ergezinger and E. Thomsen, An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer, IEEE Trans Neural Networks 6 (1995), 31–42.
106. G. J. Wang and C. C. Chen, A fast mulitlayer neural network training algorithm based on the layer-by-layer optimizing procedures, IEEE Trans Neural Networks 7 (1996), 768–775.
107. R. Lengelle and T. Denceux, Training MLPs layer by layer using an objective function for internal representations, Neural Networks 9 (1996), 83–97.
108. J. S. Shawe-Taylor and D. A. Cohen, Linear programming algorithm for neural networks, Neural Networks 3 (1990), 575–582.
109. B. Berma, Fast training of multilayer perceptrons, IEEE Trans Neural Networks 8 (1997), 1314–1320.
110. F. Wang and Q. J. Zhang, An improved *k*-means clustering algorithm and application to combined multi-codebook/MLP neural network speech recognition, Proc Canadian Conf Elec Comp Eng, Vol. II, Montreal, Quebec, 1995, pp. 999–1002.
111. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, Science 220 (1983), 671–680.
112. J. C. F. Pujol and R. Poli, Evolving neural networks using a dual representation with a combined crossover operator, Proc IEEE Int Conf Evol Comp, Anchorage, AK, May 1998, pp. 416–421.
113. T. Rögnvaldsson, On Langevin updating in multilayer perceptron, Neural Comput 6 (1994), 916–926.
114. R. Brunelli, Training neural nets through stochastic minimization, Neural Networks 7 (1994), 1405–1412.
115. OSA90 Version 3.0, Optimization Systems Associates Inc., P.O. Box 8083, Dundas, ON, Canada L9H 5E7, now HP EEsof, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403.
116. 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.
117. R. F. Harrington, Field computation by moment methods, Macmillan Co., New York, 1968.
118. NeuroModeler Version 1.0, Professor Q.J. Zhang and his neural network research team, Department of Electronics, Carleton University, 1125 Colonel By Drive, Ottawa, ON, Canada K1S 5B6.

BIOGRAPHIES

Fang Wang received the B.Eng. degree from Xi'an Jiaotong University, Xi'an, China in 1993, and the M.Eng. degree from Carleton University, Ottawa, Canada in 1995, both in electrical engineering and is currently completing the Ph.D. degree in the Department of Electronics at Carleton University, Ottawa, Canada. She is a recipient of a 1997–1999 Postgraduate Scholarship from the Natural Science and Engineering Research Council of Canada (NSERC). She is also a two-time recipient of the Ontario Graduate Scholarship in 1996 and 1997, respectively. She received a travel fellowship twice from the *International Conference on Neural Networks* in 1997 and 1998, respectively. Her research interests include neural networks, modeling, and their applications in CAD for electronics circuits. She is a member of the IEEE.

Vijaya Kumar Devabhaktuni received the B.Eng. degree in electrical and electronics and the M.Sc. degree in physics, both from Birla Institute of Technology and Science, Pilani, India in 1996. He is currently a Ph.D. candidate in the Department of Electronics, Carleton Univesity, Ottawa, Canada. His research interests include neural networks, microwave device and circuit modeling, computer-aided design of VLSI, and telecommunication circuits. He is a student member of the IEEE.

Changgeng Xi received the B.Eng. degree in Information Engineering and the M.Sc. degree in Pattern Recognition and Intelligent Control, from Xi'an Jiaotong University, Xi'an, China in 1994 and 1997, respectively. He is currently a Master's student in the Department of Electronics at Carleton University, Ottawa, Canada. His research intrests include training algorithms of neural networks, microwave modeling, and CAD for VLSI design.

Qi-Jun Zhang. For bio and photo see p. 157.