A Robust Algorithm for Automatic Development of Neural-Network Models for Microwave Applications

Vijay K. Devabhaktuni, *Student Member, IEEE*, Mustapha C. E. Yagoub, *Member, IEEE*, and Qi-Jun Zhang, *Senior Member, IEEE*

***Abstract*—For the first time, we propose a robust algorithm for automating the neural-network-based RF/microwave model development process. Starting with zero amount of training data and then proceeding with neural-network training in a stage-wise manner, the algorithm can automatically produce a neural model that meets the user-desired accuracy. In each stage, the algorithm utilizes neural-network error criteria to determine additional training/validation samples required and their location in model input space. The algorithm dynamically generates these new data samples during training, by automatic driving of simulation tools (e.g., OSA90, Ansoft-HFSS, Agilent-ADS). Initially, fewer hidden neurons are used, and the algorithm adjusts the neural-network size whenever it detects under-learning. Our technique integrates all the subtasks involved in neural modeling, thereby facilitating a more efficient and automated model development framework. It significantly reduces the intensive human effort demanded by the conventional step-by-step neural modeling approach. The algorithm inherently distinguishes nonlinear and smooth regions of model behavior and uses relatively fewer samples in smooth subregions. It automatically deals with large data errors that can occur during dynamic sampling by using Huber quasi-Newton technique. The algorithm is demonstrated through practical microwave device and circuit examples.**

***Index Terms*—Design automation, modeling, neural-network applications, optimization.**

I. Introduction

RECENTLY, a computer-aided-design (CAD) approach based on neural networks has been introduced for microwave modeling and design [1]–[3]. Neural networks are trained from measured/simulated microwave data and the resulting neural models are used during microwave design [2]–[4]. Neural modeling techniques have been applied to a wide variety of microwave problems, e.g., coplanar waveguide (CPW) components [4], transistors [5], transmission lines [6], bends [7], filters [8], and amplifiers [9]. Significant speed-up of CAD by using neural models in place of CPU-intensive electromagnetic (EM)/physics models resulted in a drive to develop advanced neural modeling techniques [10]–[12].

Manuscript received March 31, 2001; revised August 6, 2001. This work was supported in part by the Natural Science and Engineering Research Council of Canada and in part by Micronet, a Canadian Network Center of Excellence in Microelectronic Devices, Circuits and Systems.

V. K. Devabhaktuni and Q.-J. Zhang are with the Department of Electronics, Carleton University, Ottawa, ON, Canada K1S 5B6.

M. C. E. Yagoub was with the Department of Electronics, Carleton University, Ottawa, ON, Canada K1S 5B6. He is now with the School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada K1N 6N5.

Publisher Item Identifier S 0018-9480(01)10467-9.

Neural model development involves several subtasks like data generation, neural-network selection, training, and validation [13]. Conventionally, these subtasks are manually carried out in a sequential manner independent of one another. Such an approach referred to as the step-by-step neural modeling approach requires intensive human effort. Modeling is just one aspect of microwave CAD and the designers wish to develop neural models even though they do not have enough exposure to neural-network issues. There is a definite need for automation of neural model development process. However, a successful automation technique needs to address multiple complicated and interdependent challenges.

Microwave modeling problems are often highly nonlinear and multidimensional. The number of samples needed for developing a neural model with desired accuracy and their distribution in the input space are not obvious. While too many samples are expensive (e.g., three-dimensional (3-D) EM simulations), too few samples lead to over-learning of the neural network. The neural-network size required to develop an accurate model is not known *a priori*. Too many hidden neurons need more CPU and too few neurons result in under-learning of the neural network. Microwave data obtained from measurement/simulation could have a few accidental large errors that can result in nonreliable neural models. In an automated procedure, there is no place for human intervention based on problem-specific knowledge. It is, therefore, mandatory for the automation technique to take an algorithmic approach to simultaneously handle the above-mentioned issues.

Several notable techniques were proposed in the microwave area. Knowledge-based neural networks (KBNNs) [5], hierarchical neural networks [6], difference method [14], and the prior knowledge input method [15] utilize existing microwave knowledge to improve neural model accuracy with fewer data. A variety of sample distributions, e.g., star distribution [8] and central-composite distribution [14], were used. The mulilayer perceptrons neural network (MLPNN) process [13] discussed the possibility of neural-network size adjustment. Research in the neural-network area also led to several techniques, e.g., CasCor [16] and CasPer [17] begin with a minimal neural network and add neurons during training, and techniques based on robust statistical method [18] and higher order cumulants [19] deal with large errors in training data. While some of these techniques are derived for signal processing applications (not microwave), others address specific challenges or subtasks of neural model development.

In this paper, we propose a novel automatic neural model development algorithm that integrates all subtasks in neural modeling into one unified task. Starting with minimal amounts of training data and a small network, neural model development proceeds in a stage-wise manner. In subsequent stages, the algorithm can perform dynamic sampling and neural-network structure adaptation. Accidental large errors in training data that occur during dynamic sampling would be successfully dealt with.

II. Neural Modeling Problem and Automation Issues

Let $\boldsymbol{x}$ represent an N_x vector containing physical parameters of a microwave device/circuit, e.g., gate length of an FET. Let $\boldsymbol{y}$ represent an N_y vector containing the responses of the device/circuit under consideration, e.g., drain current of an FET. The EM/physics relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$ can be represented as

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

This relation can be highly nonlinear and multidimensional. The theoretical model for $\boldsymbol{y}(\boldsymbol{x})$ may be computationally too intensive for online microwave design. A fast neural model is developed by training a neural network with a set of measured/simulated samples $\{(\boldsymbol{x}_q, \boldsymbol{y}_q), \boldsymbol{x}_q \in L\}$, where $\boldsymbol{y}_q$ represents measured/simulated output for sample input $\boldsymbol{x}_q$, and L represents the set of training (learning) data. Let the input–output mapping by the neural network be defined as

$$\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{w}) \tag{2}$$

where $\tilde{\boldsymbol{y}}$ is an N_y vector of neural model outputs and $\boldsymbol{w}$ represents trainable parameters (weights) of the neural-network structure S. The objective of training is to find $\boldsymbol{w}$ that minimizes the error between neural model outputs $\tilde{\boldsymbol{y}}$ and training data $\boldsymbol{y}$.

In order to highlight the critical issues that need to be addressed by an automation technique, we briefly compare and contrast step-by-step manual approach versus automation. In the step-by-step approach, there is a scope for applying problem-specific human experience in sampling the input space for training data generation, and in testing the quality of the trained neural model. Conversely, an automated approach must have a built-in mechanism for sampling issues such as number of samples and their distribution. The technique must incorporate a systematic neural-network training and simultaneous model testing, by growing training and validation data sets in an intertwined fashion. It must be able to distinguish nonlinear and smooth regions of the model and generate samples accordingly.

The next issue is neural-network size, i.e., number of hidden layer neurons. In the manual approach, network size determination is carried out using a tedious trial-and-error process. On the other hand, an automation technique needs a systematic methodology for adapting neural-network size, e.g., adding neurons based on neural-network errors. Another key issue is the presence of accidental large errors in training data caused by equipment limitations (e.g., measurements at extreme frequencies) and nonconvergence of simulators (e.g., simulations with inputs in extreme locations). In the step-by-step approach, it may be possible (with great difficulty) to manually check and remove erroneous data samples from training set. A successful automated approach, therefore, needs to implement a periodic check for large errors and requires a training technique that can yield reliable neural models even in the presence of such errors. In summary, a lot of manual effort and periodic human decisions are involved in the step-by-step approach that needs to be avoided in an automated approach. These challenges are the motivation for our research toward automation.

III. Automatic Neural Model Development Algorithm

A. Introduction

Contrary to the step-by-step approach, the proposed automatic neural model development algorithm focuses on the mutual dependence of various subtasks like data generation, neural-network size, neural-network training, and validation. Our algorithm links these subtasks through neural-network learning phenomena (e.g., over-learning, under-learning) and integrates them into one unified process. The integrated process is computerized and is carried out automatically in a stage-wise fashion. Within a stage, the algorithm facilitates periodic communication between various subtasks, thus enabling adjustment or enhancement in the execution of a subtask based on the feedback from other subtasks. As a result, each stage could involve dynamic incremental data generation, neural-network size adjustment, neural-network training with training data, and neural model testing with validation data. The algorithm has built-in simulation drivers (e.g., OSA90 driver, HFSS driver, ADS driver) for facilitating automatic data generation during neural-network training. Accidental large errors that could occur in training data during dynamic data generation are detected and the effect of these errors is neglected by automatic switching to Huber quasi-Newton (HQN) training algorithm. The technique establishes a quantitative link between neural model accuracy, the number and distribution of training/validation data, and the neural-network size. Automation of the model development process shifts the workload from human to computer thus making model development more efficient and less prone to errors. In the following subsections, we present a detailed description of the proposed algorithm.

B. Notation

We use two disjoint sets of data namely, the training data and the validation data. Training data are used to update neural-network weights during training and validation data are used to monitor the quality of neural model during training. We define L^k and V^k as training (learning) and validation data sets during the kth stage, i.e., $L^k = \{\boldsymbol{x}_i | (\boldsymbol{x}_i, \boldsymbol{y}_i)$ is a training sample$\}$ and $V^k = \{\boldsymbol{x}_j | (\boldsymbol{x}_j, \boldsymbol{y}_j)$ is a validation sample$\}$. Let S^k represent neural-network structure in kth stage with outputs $\tilde{\boldsymbol{y}}^{\boldsymbol{k}} = \tilde{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{w}^k)$, where $\boldsymbol{w}^k$ is the corresponding weight vector. The range of input parameter space in which the neural model would be used during design is referred to as input space of interest and is denoted by R_0. Normalized training error of neural network S^k at the end of kth stage is defined as

$$E_l^k = \frac{1}{N_l^k} \sum_{\boldsymbol{x}_q \in L^k} \left(e_q(\boldsymbol{w}^k)\right)^p \tag{3}$$

where p represents the pth norm, N_l^k is the number of samples in L^k, and $e_q(\boldsymbol{w}^k)$ is the error due to qth sample in L^k given by

$$e_q(\boldsymbol{w}^k) = e(\boldsymbol{x}_q, \boldsymbol{w}^k) = \frac{1}{N_y} \left[\frac{1}{p} \sum_{j=1}^{N_y} \left| \frac{\tilde{y}_j(\boldsymbol{x}_q, \boldsymbol{w}^k) - y_{qj}}{y_{\max, j} - y_{\min, j}} \right|^p \right]^{1/p} \tag{4}$$

where $\tilde{y}_j(\boldsymbol{x}_q, \boldsymbol{w}^k)$ is the jth neural-network output at the end of the kth stage training for input sample $\boldsymbol{x}_q$ and y_{qj} is the jth element of $\boldsymbol{y}_q$. In (4), $y_{\min, j}$ and $y_{\max, j}$ are the minimum and maximum possible values of y_j data in the input space of interest. The objective of neural-network training in kth stage is to find $\boldsymbol{w}^k$ such that E_l^k is minimized. Normalized validation error of neural model S^k at the end of kth stage training is defined as

$$E_v^k = \frac{1}{N_v^k} \sum_{\boldsymbol{x}_q \in V^k} \left(e_q(\boldsymbol{w}^k)\right)^p \tag{5}$$

where N_v^k is the number of samples in V^k and $e_q(\boldsymbol{w}^k)$ is the error due to the qth sample in V^k, computed using (4). Let E_d represent user-desired neural model accuracy (validation error). The objective of the algorithm is to automatically carry out stage-wise model development process until $E_v^k \leq E_d$.

C. Key Aspects of the Algorithm

To begin with, the algorithm considers the original bounded N_x-dimensional input space of interest R_0 as one region. For the first stage, training data (L^1) and validation data (V^1) are systematically generated in R_0 in a predefined way (e.g., star distribution). In our work, training and validation data for a given region are generated as shown in Fig. 1(a). A first stage neural-network S^1 with relatively fewer hidden layer neurons is trained with data samples in L^1, i.e., $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $\boldsymbol{x}_i \in L^1$. The resulting neural model is validated (tested) with data samples in V^1, i.e., $(\boldsymbol{x}_j, \boldsymbol{y}_j), \boldsymbol{x}_j \in V^1$. The algorithm stops if $E_v^1 \leq E_d$. Otherwise, based on the neural-network error criteria, the algorithm automatically takes suitable actions and proceeds to the next stage of model development.

1) Automatic Sampling and Generation of Training/Validation Data: Over-learning of the neural network may be detected at the end of the kth stage, using error information E_l^k and E_v^k. Over-learning is a phenomenon in which the neural network memorizes the training data accurately but cannot generalize well, i.e., E_l^k is small enough (compared to E_d) but $E_v^k \gg E_l^k$. When the algorithm detects over-learning, it dynamically adds more data samples to the training and validation sets. Motivated by the concepts of sampling techniques based on multinomials [20], rational interpolation [21], and error-based data exploration [22], we developed a unique neural-network-oriented technique featuring the growth of training and validation sets in an intertwined way. Utilizing this technique, the algorithm handles the issues of number of additional samples and their distribution in model input space. The validation sample $\boldsymbol{x}^* \in V^k$ with maximum (worst-case) error is identified by

$$\boldsymbol{x}^* = \arg \max_{\boldsymbol{x} \in V^k} e(\boldsymbol{x}, \boldsymbol{w}^k). \tag{6}$$

The worst region R^* to which $\boldsymbol{x}^*$ belongs is further divided (split) into 2^{N_x} new subregions. Considering these new regions, L^k and V^k are updated, i.e.,

$$L^{k+1} = L^k \bigcup L^{\text{new}} \tag{7}$$

$$V^{k+1} = V^k \bigcup V^{\text{new}} \tag{8}$$

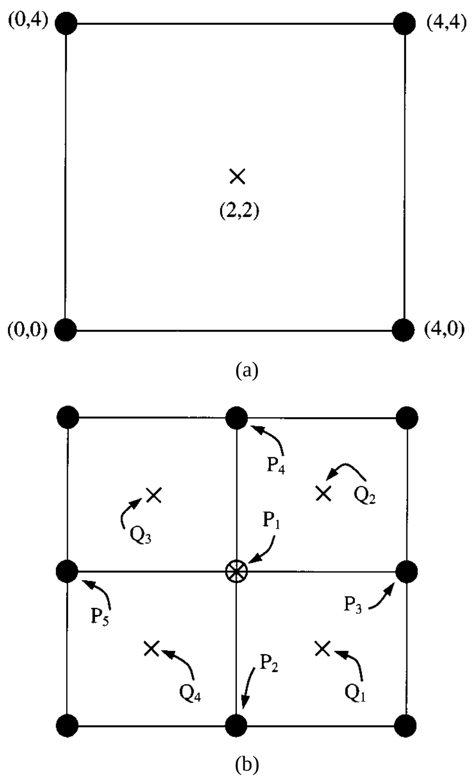


Fig. 1. (a) Training and validation data in a typical subregion of two-dimensional input space. (b) When the subregion is identified as the worst region, it is further divided into smaller regions. Newly generated training and validation data are clearly indicated. Training data (●). Validation data (×). Validation data in a previous stage and training data in the current stage (⊗).

where L^{new} and V^{new} represent the new data samples to be generated. Each new training input point $\boldsymbol{x}_{\text{new}} \in L^{\text{new}}$ is obtained as

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}^* + \boldsymbol{P}_i \frac{\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min}}{2^{r+1}}, \qquad i = 1, 2, \ldots, n_{\text{new}} \tag{9}$$

where $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ are the extreme boundaries of R^*, r is the number of splits after which R^* was formed, n_{new} is the number of newly added training points, and $\boldsymbol{P}_i$ is a $N_x \times N_x$ diagonal matrix. Each diagonal element of $\boldsymbol{P}_i$ could take one of the values 0, $+1$, or -1, depending upon the predefined sample distribution in each subregion. Each new validation input point $\boldsymbol{x}_{\text{new}} \in V^{\text{new}}$ is obtained as

$$\boldsymbol{x}_{\text{new}} = \boldsymbol{x}^* + \boldsymbol{Q}_j \frac{\boldsymbol{x}_{\max} - \boldsymbol{x}_{\min}}{2^{r+2}}, \qquad j = 1, 2, \ldots, m_{\text{new}} \tag{10}$$

where m_{new} is the number of newly added validation points and $\boldsymbol{Q}_j$ is a diagonal matrix similar to $\boldsymbol{P}_i$. Incremental training and validation data are generated corresponding to the newly added input points by dynamically driving the data generator, e.g., EM simulator.

For example, suppose the original two-dimensional (2-D) input space (region) shown in Fig. 1(a) needs to be split because the first stage of training did not yield a satisfactory neural model. We then have $\boldsymbol{x}^* = [2\ 2]^{\text{T}}$, $\boldsymbol{x}_{\min} = [0\ 0]^{\text{T}}$,

$\boldsymbol{x}_{\max} = [4 \ \ 4]^{\mathrm{T}}$, and $r = 0$. For the second stage, five new training samples are generated with

$$\boldsymbol{P}_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$\boldsymbol{P}_2 = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$$
$$\boldsymbol{P}_3 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$
$$\boldsymbol{P}_4 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\boldsymbol{P}_5 = \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Four new validation samples are generated with

$$\boldsymbol{Q}_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
$$\boldsymbol{Q}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\boldsymbol{Q}_3 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\boldsymbol{Q}_4 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

as shown in Fig. 1(b).

2) Automatic Neural-Network Structure Adaptation: Under-learning of the neural network may be detected after the kth stage, using training error E_l^k and its gradient $\Delta E_l^k = E_l^{k-1} - E_l^k$. Under-learning is a phenomenon in which the neural network has difficulties in learning the training data itself, i.e., E_l^k is large and ΔE_l^k is small. When the algorithm detects under-learning, it dynamically adds more hidden layer neurons. A larger neural network would then provide increased freedom to better learn the nonlinearities in training data. If the neural-network structure S^k has N_h^k hidden neurons, S^{k+1} with $N_h^{k+1} = N_h^k + \delta$ hidden neurons is used by the algorithm in the $k + 1$th stage. Suggested range for the newly added hidden neurons δ is 10%–20% of N_h^k.

3) Automatic Handling of Large Errors in Dynamically Generated Samples: In general, most of the microwave samples have small measurement/simulation errors and a few samples could even have large errors. A few accidental large errors could occur in training data during dynamic data generation of the automatic model development approach. It is essential to automatically detect these large errors and neglect them during neural-network training, because there is no place for human intervention in an automated approach. The large errors are detected as a special case of under-learning, i.e., when E_l^k continues to be large and ΔE_l^k remains small for several consecutive stages even after repeatedly adding hidden neurons. Once large errors are detected, automatic training switches from conventional neural-network training algorithms (e.g., quasi-Newton) to HQN technique.

The objective functions of conventional training algorithms are formulated in l_2-sense (i.e., $p = 2$). Although l_2-norm-based training handles small errors in data, it is misled by large errors. On the other hand, l_1-norm-based training is robust against large errors, but is not very effective in dealing with small errors [23]. Huber function [24], which is a smooth combination of l_1 and l_2-norms, is used here for neural-network training. We compute the per-sample error function $e_q(\boldsymbol{w}^k)$ in (4) with $p = 2$. Normalized training error of the neural-network structure S^k is re-defined using the Huber function as

$$E_l^k = \frac{1}{N_l^k} \sum_{\boldsymbol{x}_q \in L^k} \rho\big[e_q(\boldsymbol{w}^k)\big] \tag{11}$$

where $\rho(\cdot)$ is the Huber function. The Huber-norm of $e_q(\boldsymbol{w}^k)$ is given by

$$\rho\big[e_q(\boldsymbol{w}^k)\big] = \begin{cases} \big[e_q(\boldsymbol{w}^k)\big]^2/2, & \text{if } \big|e_q(\boldsymbol{w}^k)\big| \le \Phi \\ \Phi\big|e_q(\boldsymbol{w}^k)\big| - \dfrac{\Phi^2}{2}, & \text{if } \big|e_q(\boldsymbol{w}^k)\big| > \Phi \end{cases} \tag{12}$$

where Φ is the Huber constant. By varying Φ, the proportion of neural-network error functions to be treated in l_1 or l_2 sense can be controlled. Consequently, the Huber-norm-based training objective can be robust against both small and large errors in data. When accidental large errors are detected in the kth-stage training data, our algorithm switches neural-network training process to HQN. The HQN algorithm ensures that the network S^{k+1} learns only the original problem behaviors, neglecting large errors.

4) Overall Automation: At the end of each stage, the algorithm checks for various possible neural-network training situations and takes relevant actions, e.g., update data, adjust neural-network size, etc. In the subsequent stage, neural network S^{k+1} is trained with samples in L^{k+1} and the neural model is tested with samples in V^{k+1}. A framework of the proposed algorithm is shown in Fig. 2. We also incorporated a few conservative options to make the algorithm more general: 1) periodically, after a

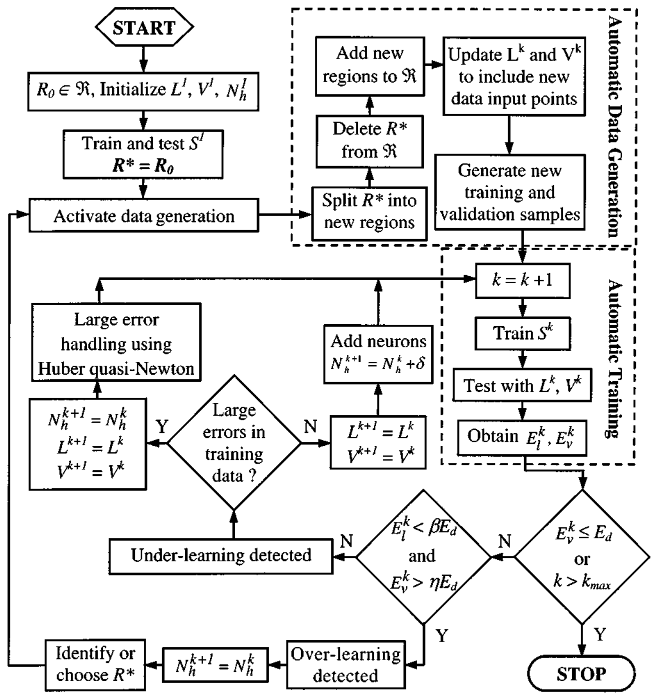

START
$R_0 \in \Re$, Initialize L^l, V^l, N_h^l
Train and test S^l $R^* = R_0$
Activate data generation
Add new regions to $\Re$
Update L^k and V^k to include new data input points
Delete R^* from $\Re$
Split R^* into new regions
Generate new training and validation samples
Automatic Data Generation
$k = k+1$
Train S^k
Test with L^k, V^k
Obtain E_l^k, E_v^k
Automatic Training
Large error handling using Huber quasi-Newton
Add neurons $N_h^{k+1} = N_h^k + \delta$
$N_h^{k+1} = N_h^k$ $L^{k+1} = L^k$ $V^{k+1} = V^k$
Y
Large errors in training data ?
N
$L^{k+1} = L^k$ $V^{k+1} = V^k$
Under-learning detected
N
$E_l^k < \beta E_d$ and $E_v^k > \eta E_d$
N
$E_v^k \le E_d$ or $k > k_{max}$
Y
Y
Identify or choose R^*
$N_h^{k+1} = N_h^k$
Over-learning detected
STOP

Fig. 2. Systematic framework of the proposed automatic neural model development algorithm in the form of a flowchart.

fixed number of stages, incremental training/validation samples are generated in a randomly chosen sub-region instead of a worst subregion and 2) the algorithm terminates only if $E_v^k \leq E_d$ consecutively for a given number of stages.

D. Implementation Details

Let E_l^k, E_v^k, and ΔE_l^k represent training error, validation error, and gradient of the training error of the neural network S^k at the end of the kth stage. Let $\Re$ represent a set of regions in the neural model input space. The main user inputs to our algorithm are the user-desired neural model accuracy E_d and the maximum allowable number of stages $k_{\max}$. There are other user-inputs k_c, β, η, k_r, α, ΔE, k_u, and k_e. For $E_d = 1\%$, suggested ranges of these constants are $1 \leq k_c \leq 5$, $0 \leq \beta \leq 1$, $\eta \geq 1$, $4 \leq k_r \leq 6$, $2 \leq \alpha \leq 4$, $0.01 \leq \Delta E \leq 0.1$ (in % E_l^k), $1 \leq k_u \leq 6$, and $4 < k_e \leq 6$. The pseudocode of the algorithm is presented below.

Initialization: $k = 1$, $N_h^1 = N_0$, $L^1 = \{\boldsymbol{x}_i | \boldsymbol{x}_i$ is a vertex of $R_0\}$, $V^1 = \{\boldsymbol{x}_j | \boldsymbol{x}_j$ is the center of $R_0\}$, and $flag = 0$. Automatically generate $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ for all $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. There is only one region, $R_0 \in \Re$. Train neural network S^1 using samples $(\boldsymbol{x}_i, \boldsymbol{y}_i), \boldsymbol{x}_i \in L^1$, and test the neural model with $(\boldsymbol{x}_j, \boldsymbol{y}_j), \boldsymbol{x}_j \in V^1$. *Activate Data Generation* (R_0).

Activate Data Generation (R): Split R into 2^{N_x} new regions. Delete R from $\Re$ and add the new regions to $\Re$. Update L^k and V^k, and generate training and validation data for the new subregions by automatically driving the simulator. Go to *Automatic Training.*

Automatic Training: $k = k+1$. Train S^k using data samples in L^k. Test the neural model with samples in L^k and V^k, to obtain E_l^k and E_v^k respectively.

```
if ($k > k_{\max}$) or ($E_v^k \leq E_d$ for $k_c$ consecutive stages) Stop Training.
else if ($E_l^k < \beta E_d$) and ($E_v^k > \eta E_d$), over-learning is detected. Set $flag = 0$.
    if (($k$ % $k_r$) $\neq$ 0) choose worst sample $\boldsymbol{x}^* \in V^k$ and identify the corresponding sub-region $R^*$. Set $N_h^{k+1} = N_h^k$ and Activate Data Generation ($R^*$).
    else randomly choose a region $R^* \in \Re$. Set $N_h^{k+1} = N_h^k$. Activate Data Generation ($R^*$).
else if ($E_l^k > \alpha E_d$) and ($\Delta E_l^k < \Delta E$ for $k_u$ consecutive stages), under-learning is detected.
    if (($flag$ % $k_e$) = 0), training data has accidental large errors. Set $N_h^{k+1} = N_h^k$, $L^{k+1} = L^k$, $V^{k+1} = V^k$. Switch Training Algorithm from quasi-Newton to HQN. Go to Automatic Training.
    else Add Hidden Neurons, i.e., $N_h^{k+1} = N_h^k + \delta$. Set $L^{k+1} = L^k$, $V^{k+1} = V^k$. $flag = flag + 1$. Go to Automatic Training.
else possible local minimum in neural-network training. Set $N_h^{k+1} = N_h^k$, $L^{k+1} = L^k$, $V^{k+1} = V^k$. Set $flag = 0$.
    Randomly perturb network weights $\boldsymbol{w}^k$. Go to Automatic Training.
```

IV. Demonstration Examples

A. Neural Modeling of MESFET

The proposed algorithm is incorporated into our NeuroModeler software [25]. The input space $\boldsymbol{x}$ of the neural network contains gate-length (l), channel thickness (a), gate–source voltage (v_g), and drain–source voltage (v_d). Drain current (i_d) is the neural-network output $\boldsymbol{y}$. For a user-desired neural model accuracy E_d, the number of training and validation samples needed and their distribution in (l, a, v_g, v_d) space and the neural-network size are not known. The proposed algorithm is used to develop a neural model to represent the physics-based Khatibzadeh and Trew MESFET model of OSA90.[1]

The algorithm starts with an initial neural network (three-layer MLP) with $N_h^1 = 9$ hidden neurons. In the first stage, the algorithm dynamically generates $N_l^1 = 16$ samples to train the neural network and $N_v^1 = 1$ validation sample to test it. During subsequent stages, the algorithm automatically decides the number and distribution of additional training and validation samples needed and dynamically drives the OSA90 simulator using our OSA90 driver. Extra hidden layer neurons are also automatically added as needed. For $E_d = 0.5\%$, we set the user-inputs of the algorithm as $k_c = 1$, $\beta = 1$, $\eta = 1$, $k_r = 5$, $\alpha = 2$, $\Delta E = 0.05\%$, $k_u = 1$, and $k_e = 6$. The algorithm produced a neural model with a validation error $E_v^8 = 0.48\%$ after eight stages of model development. A total of $N_l^8 = 318$ training samples and $N_v^8 = 90$ validation samples are used and the final neural model has $N_h^8 = 16$ hidden neurons.

The neural model is further tested with a large set of independent test data never seen during training. The average test error is observed to be 0.49%, confirming the reliability of the neural model. Using the manual step-by-step neural modeling approach, MLP neural networks with 9, 12, and 16 hidden neurons are trained using uniform-grid samples and the average test errors are reported in Table I. It can be seen that the proposed algorithm gives more accurate neural models with the same amount of training data, as compared to step-by-step approach. This is because the proposed method uses efficient distribution, i.e., more (less) data are generated in nonlinear (smooth) regions as shown in Fig. 3.

As a further step, we applied our algorithm to an advanced nonlayered neural-network structure called the KBNN [5]. For the MESFET, microwave knowledge in the form of empirical equations is available [26]. In the KBNN structure, these empirical functions are used as hidden neuron activation functions. Within four stages of model development, the KBNN achieved an accuracy of 0.53% using 208 training samples. At the end of five stages, the KBNN model accuracy is 0.29% and the number of training samples used is 248. This shows that our algorithm is applicable to arbitrary nonlayered neural-network structures. Using KBNN together with the proposed model development algorithm achieved further improved neural model accuracy with fewer data.

[1]OSA90, Optimization Syst. Assoc., Dundas, ON, Canada (now Agilent Technologies, Palo Alto, CA).

TABLE I
MODEL ACCURACY COMPARISON BETWEEN THE PROPOSED ALGORITHM BASED ON AUTOMATIC SAMPLING AND THE STEP-BY-STEP APPROACH BASED ON CONVENTIONAL GRID DISTRIBUTION FOR THE MESFET EXAMPLE. MODEL ACCURACY REPORTED FOR THE MANUAL APPROACH IS AN AVERAGE VALIDATION ERROR OF THREE NEURAL MODELS WITH 9, 12, AND 16 HIDDEN NEURONS

Proposed Automatic Neural Model Development Algorithm					Manual Step-by-Step Neural Modeling Approach	
Stage No. (k)	No. of Training Samples (N_l^k)	No. of Hidden Neurons (N_h^k)	Action taken at the end of Stage	Model Accuracy (E_v^k in %)	No. of Training Samples used	Model Accuracy (%)
1	16	9	Add Samples	23.38	16	25.74
2	81	9	Add Samples	6.64	81	8.05
3	146	9	Add Neurons	4.32	144	6.42
4	146	12	Add Samples	3.54		
5	211	12	Add Samples	1.82	225	3.20
6	256	12	Add Samples	0.90	256	1.28
7	318	12	Add Neurons	0.75	320	0.75
8	318	16	Stop	0.48		

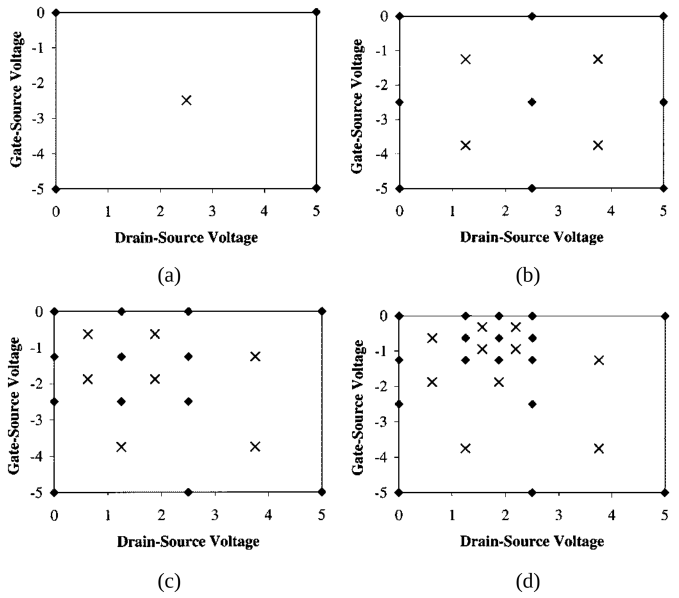


Fig. 3. Intertwined automatic distribution of training data (♦) and validation data (×) by the proposed algorithm for the MESFET example. (a) First stage, (b) second stage, (c) third stage, and (d) fourth stage of training. The algorithm identifies nonlinear subregions of the MESFET input space and automatically generates more samples in such regions.

B. Embedded Capacitor in Multilayer Printed Circuit Boards

Accurately modeling 3-D EM behaviors of embedded components [27] used in high-speed multilayer printed circuit boards (PCB) is important for efficient CAD. In this example, neural model of an embedded capacitor shown in Fig. 4 is developed. The $\boldsymbol{y}(\boldsymbol{x})$ relationship of the capacitor is available from CPU-intensive EM simulations of Ansoft-HFSS simulator.[2]

[2]Ansoft HFSS 7.0.11, Ansoft Corporation, Pittsburgh, PA.

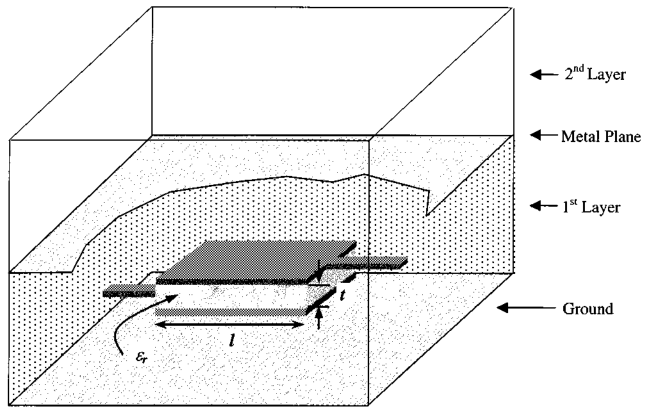


Fig. 4. Embedded capacitor used in multilayer PCBs. S-parameter neural model of the capacitor is developed from 3-D EM data of Ansoft-HFSS using the proposed algorithm.

TABLE II
COMPARISON OF THE TIME TAKEN BY THE PROPOSED ALGORITHM AND MANUAL STEP-BY-STEP NEURAL MODELING APPROACH FOR THE EMBEDDED CAPACITOR

	Proposed Automatic Model Development Algorithm		Manual Step-By-Step Neural Modeling Approach	
	Human Time	CPU Time	Human Time	CPU Time
	5 min	1158 min	498 min	1625 min
Total Time	1163 min		2123 min	

The step-by-step neural modeling approach based on manual data generation is prohibitive because it involves the following

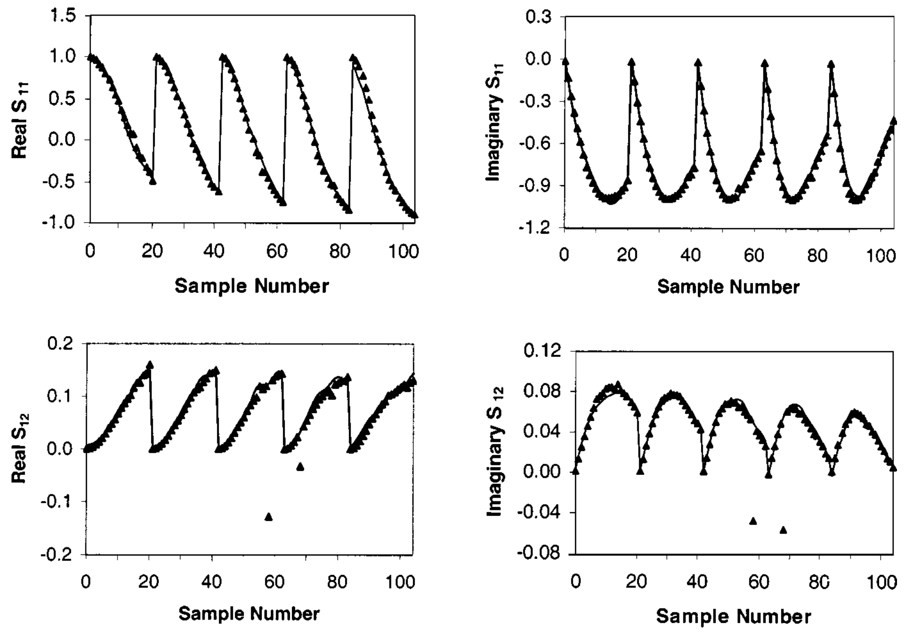


Fig. 5. Comparison between neural model (—) prediction and training data (▲) for the embedded capacitor. The effect of large errors introduced in S_{12} training data during automatic data generation are neglected by the neural model developed using the proposed algorithm.

tasks: 1) re-drawing the capacitor with new physical parameter values graphically (or editing simulator input file for every input vector); 2) submitting and waiting for the CPU-intensive data computation to finish; 3) viewing simulator output file for the outputs of interest; and 4) appending the new $\boldsymbol{x}$–$\boldsymbol{y}$ sample to the training or validation data file. These exhaustive tasks make manual data generation highly time-consuming, human intensive, and error-prone. The proposed algorithm based on automatic data generation is used.

The input space $\boldsymbol{x}$ of the capacitor neural network includes length (l), thickness (t), dielectric constant (ε_r), and frequency ($f = 0.1–20$ GHz). Real and imaginary parts of two-port S-parameters S_{11} and S_{12} are the neural model outputs $\boldsymbol{y}$. The user specification of the neural model accuracy is $E_d = 1.25\%$. In the first stage, a neural network S^1 with $N_h^1 = 12$ hidden neurons is used. A total of 16 training and 1 validation samples are used. Whenever the automatic algorithm decides to add more data, it dynamically drives the Ansoft-HFSS simulator using our HFSS driver. After ten stages, the final neural model S^{10} with $N_h^{10} = 20$ hidden neurons, 554 training and 136 validation samples, achieved an accuracy of $E_v^{10} = 0.92\%$. On the other hand, an MLP neural network with 20 hidden neurons trained using the manual step-by-step neural modeling approach with 554 uniform grid samples achieved a validation error of 6.8%. A comparison of training data shows that the automatic algorithm used 176 training samples in the sub-region $f \in (0.1 - 3\,\text{GHz})$ where S-parameters exhibit a relatively nonlinear behavior, while the manual uniform-grid sampling used only 128 samples. The manual approach required 768 training samples to achieve a model accuracy of 1%. Neural model developed by our algorithm is subjected to an independent test with a large set of data (8200 samples) never seen during training and the test error is observed to be 1.04%, further confirming the reliability of our algorithm.

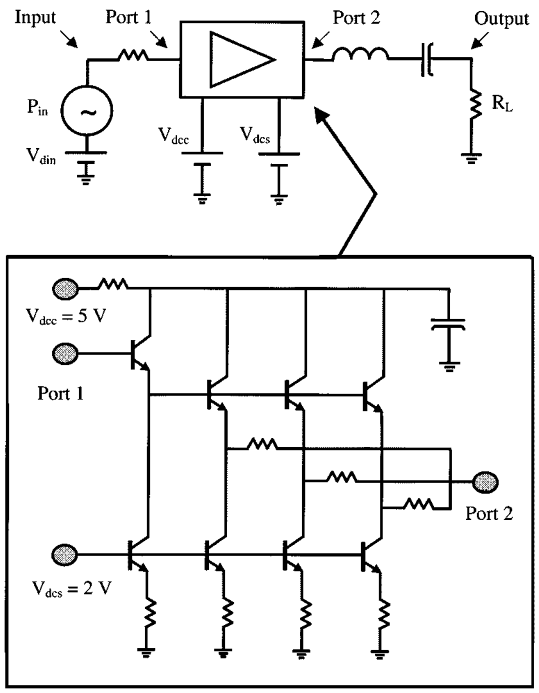


Fig. 6. Circuit diagram of a power amplifier used in wireless communication systems.

A time comparison between the proposed algorithm and the step-by-step manual neural modeling approach is shown in Table II. It can be seen that the human time required in the case of the proposed algorithm is very small as compared to the manual approach. The reason is that, the data generation in our algorithm is automatic as opposed to manual data generation in step-by-step approach. The CPU time required by our approach is also relatively smaller. This is because the manual approach requires relatively larger number of training samples to achieve similar model accuracy.

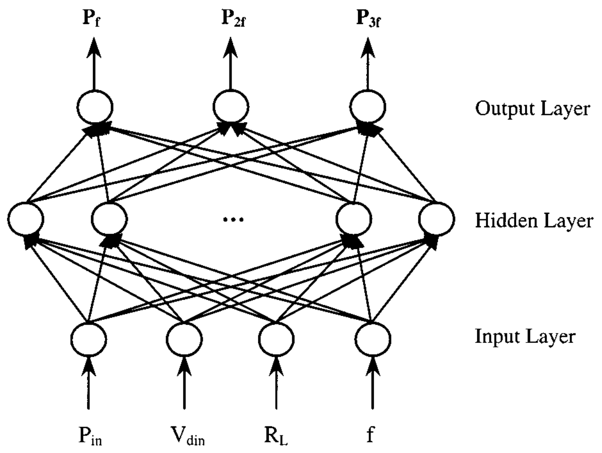


Fig. 7. Three-layer MLP neural-network structure used by the proposed algorithm to model the nonlinear behaviors of the power amplifier circuit.

During automatic data generation using HFSS, a few large accidental errors are detected in S_{12}. These errors can be attributed to simulations in extreme locations of the model input space. The proposed algorithm successfully detected these nonconvergence simulation errors and automatically switched the neural-network training process from quasi-Newton algorithm to HQN. The resulting neural model neglected these large errors and modeled only the nonerroneous data samples as can be seen in Fig. 5.

C. Nonlinear Modeling of a Power Amplifier

This example is for the purpose of illustrating the applicability of our algorithm to circuit modeling problems. The algorithm is used to develop a neural model to represent nonlinear behaviors of a power amplifier circuit used in wireless communication systems. The amplifier shown in Fig. 6 has eight n-p-n bipolar junction transistors (BJTs) represented by two internal nonlinear models Q34 and Q37 of Agilent-ADS.[3] Input parameters for the power amplifier neural network are power input ($P_{\rm in}$), input bias voltage ($V_{\rm din}$), resistive load (R_L), and input frequency (f). Outputs of the neural model include power outputs at fundamental frequency (P_f), second harmonic (P_{2f}), and third harmonic (P_{3f}). The $\boldsymbol{y}(\boldsymbol{x})$ relationship of the amplifier is originally produced by Agilent-ADS simulator. A three-layered MLP neural-network structure shown in Fig. 7 is used by the proposed algorithm to model microwave characteristics of the power amplifier circuit.

The input parameter space of interest R_0 is bounded by $P_{\rm in} = -20$ to $+20$ dBm, $V_{\rm din} = 2$ to 4 V, $R_L = 20$ to 140 Ω, and $f = 0.7$ to 4.2 GHz. To facilitate dynamic data generation during neural-network training, we implemented the ADS driver to automatically drive Agilent-ADS. After 28 training stages, a neural network with 60 hidden neurons achieved an accuracy of 1.25%. A total of 1758 training and 436 validation samples are used. A comparison of neural model prediction of the amplifier outputs with original data from ADS is shown in Fig. 8. A quantitative relationship between amount of training data and neural model accuracy, as provided by the proposed algorithm is shown in Fig. 9.

[3]ADS 1.5, Agilent Technologies, Palo Alto, CA.

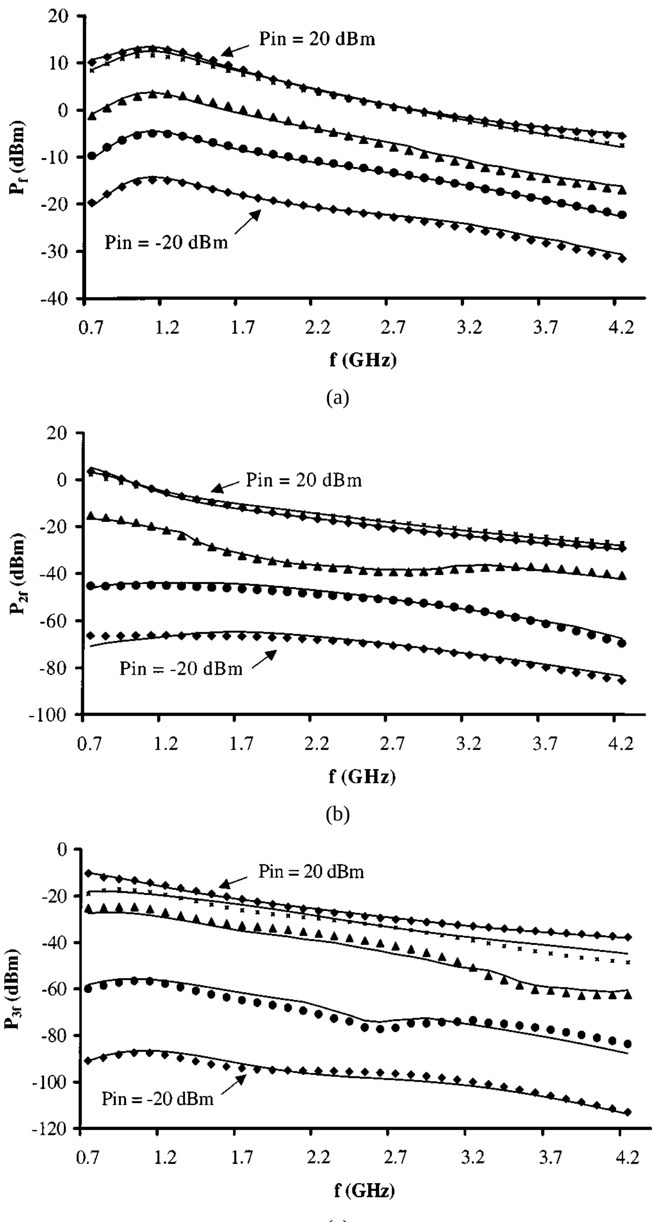


Fig. 8. Comparison of the neural model prediction (—) of the amplifier power outputs at: (a) fundamental, (b) second harmonic, and (c) third harmonic, with original Agilent-ADS data (symbols) for different values of $P_{\rm in}(V_{\rm din} = 3$ V and $R_L = 80\ \Omega)$.

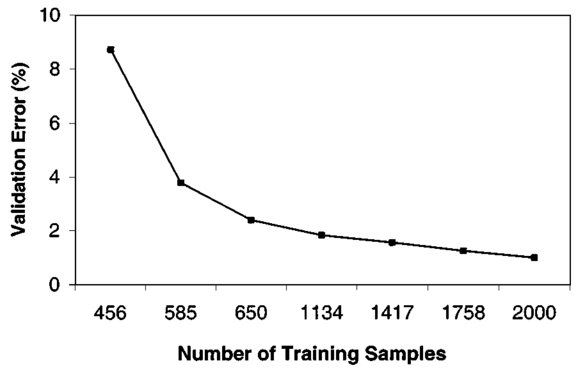


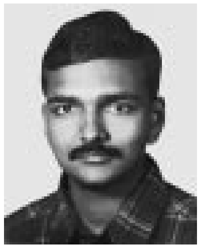
Fig. 9. Neural model accuracy versus number of training samples for the power amplifier circuit. This quantitative relationship is provided by the proposed algorithm.

V. Conclusion

We proposed a robust algorithm for automatic development of neural-network models for RF/microwave devices and circuits. It has been used to build neural models, starting with zero or minimal amounts of training and validation data. In each stage of model development, the algorithm can add samples or neurons as needed. New samples were dynamically generated during training by automatic driving of simulation tools. The proposed technique uses relatively fewer samples than the step-by-step approach to achieve similar model accuracy, and significantly reduces the human time. It is applicable to layered neural networks as well as arbitrary structures (e.g., KBNN). Accurate neural model development is made possible even in the presence of accidental large errors in training data. The algorithm can automatically produce a neural model with user-specified accuracy, without requiring the user's understanding of the neural-network issues. The technique provides a systematic framework for automated neural modeling approach, which can be incorporated into the overall microwave CAD environment.

References

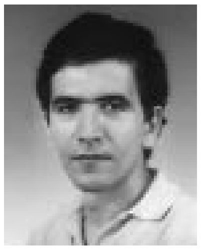
[1] V. K. Devabhaktuni, M. C. E. Yagoub, Y. Fang, J. Xu, and Q. J. Zhang, "Neural networks for microwave modeling: Model development issues and nonlinear modeling techniques," *Int. J. RF Microwave CAE*, vol. 11, pp. 4–21, 2001.
[2] Q. J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*. Norwood, MA: Artech House, 2000.
[3] F. Wang, V. K. Devabhaktuni, C. Xi, and Q. J. Zhang, "Neural network structures and training algorithms for RF and microwave applications," *Int. J. RF Microwave CAE*, vol. 9, pp. 216–240, 1999.
[4] P. Watson, G. Creech, and K. Gupta, "Knowledge based EM-ANN models for the design of wide bandwidth CPW patch/slot antennas," in *IEEE AP-S Int. Symp. Dig.*, Orlando, FL, July 1999, pp. 2588–2591.
[5] F. Wang and Q. J. Zhang, "Knowledge-based neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2333–2343, Dec. 1997.
[6] F. Wang, V. K. Devabhaktuni, and Q. J. Zhang, "A hierarchical neural network approach to the development of a library of neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 2391–2403, Dec. 1998.
[7] P. M. Watson, K. C. Gupta, and R. L. Mahajan, "Applications of knowledge-based artificial neural network modeling to microwave components," *Int. J. RF Microwave CAE*, vol. 9, pp. 254–260, 1999.
[8] J. W. Bandler, M. A. Ismail, J. E. Rayas-Sanchez, and Q. J. Zhang, "Neuromodeling of microwave circuits exploiting space-mapping technology," *IEEE Trans. Microwave Theory Tech.*, vol. 47, pp. 2417–2427, Dec. 1999.
[9] M. Vai and S. Prasad, "Neural networks in microwave circuit design—Beyond black box models," *Int. J. RF Microwave CAE*, vol. 9, pp. 187–197, 1999.
[10] V. K. Devabhaktuni, C. Xi, F. Wang, and Q. J. Zhang, "Robust training of microwave neural models," in *IEEE MTT-S Int. Microwave Symp. Dig.*, Anaheim, CA, June 1999, pp. 145–148.
[11] V. K. Devabhaktuni and Q. J. Zhang, "Neural network training-driven adaptive sampling algorithm for microwave modeling," in *Proc. Eur. Microwave Conf.*, Paris, France, Oct. 2000, pp. 222–225.
[12] V. K. Devabhaktuni, C. Xi, F. Wang, and Q. J. Zhang, "An iterative multi-stage algorithm for robust training of RF/microwave neural models," in *Proc. IEEE Asia–Pacific Circuits Syst. Conf.*, Tianjin, China, Dec. 2000, pp. 327–330.
[13] 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.*, vol. 45, pp. 794–802, May 1997.
[14] P. M. Watson and K. C. Gupta, "EM-ANN models for microstrip vias and interconnects in dataset circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 44, pp. 2495–2503, Dec. 1996.
[15] P. M. Watson, K. C. Gupta, and R. L. Mahajan, "Development of knowledge based artificial neural network models for microwave components," in *IEEE Int. Microwave Symp. Dig.*, Baltimore, MD, June 1998, pp. 9–12.
[16] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing*, D. S. Truretzky, Ed. San Mateo, CA: Morgan Kauffman, 1990, vol. II, pp. 524–532.
[17] N. K. Treadgold and T. D. Gedeon, "Extending CasPer: A regression survey," in *Proc. Int. Neural Inform. Processing Conf.*, Dunedin, New Zealand, Nov. 1997, pp. 310–313.
[18] D. S. Chen and R. C. Jain, "A robust back propagation learning algorithm for function approximation," *IEEE Trans. Neural Networks*, vol. 5, pp. 467–479, May 1994.
[19] C. T. Leung and T. W. S. Chow, "Noise robustness enhancement using fourth-order cumulants cost function," in *Proc. IEEE Int. Neural Networks Conf.*, Washington, DC, June 1996, pp. 1918–1923.
[20] J. D. Geest, T. Dhaene, N. Fache, and D. D. Zutter, "Adaptive sampling algorithm for accurate modeling of general interconnection structures," in *Proc. European Microwave Conf.*, Munich, Germany, Oct. 1999, pp. 178–181.
[21] R. Lehmensiek and P. Meyer, "An efficient adaptive frequency sampling algorithm for model-based parameter estimation as applied to aggressive space mapping," *Microwave Opt. Technol. Lett.*, vol. 24, pp. 71–78, 2000.
[22] U. Beyer and F. Smieja, "Data exploration with reflective adaptive models," *Comp. Stat. Data Anal.*, vol. 22, pp. 193–211, 1996.
[23] J. W. Bandler, W. Kellermann, and K. Madsen, "A nonlinear l_1 optimization algorithm for design, modeling and diagnosis of networks," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 174–181, Feb. 1987.
[24] P. Huber, *Robust Statistics*. New York, NY: Wiley, 1981.
[25] Q. J. Zhang, "NeuroModeler," Dept. Electron., Carleton Univ., Ottawa, ON, Canada, 2000.
[26] P. H. Ladbrooke, *MMIC Design: GaAs FET's and HEMT's*. Norwood, MA: Artech House, 1989.
[27] T. Lenihan, L. Schaper, Y. Shi, G. Morcan, and J. Parkerson, "Embedded thin film resistors, capacitors and inductors in flexible polyimide films," in *Proc. Electron. Comp. Tech. Conf.*, Orlando, FL, May 1996, pp. 119–124.



Vijay K. Devabhaktuni (S'97) received the B.Eng. degree in electrical and electronics and the M.Sc. degree in physics from the Birla Institute of Technology and Science, Pilani, Rajasthan, India, in 1996, and is currently working toward the Ph.D. degree in electronics at Carleton University, Ottawa, ON, Canada.

He is a Sessional Lecturer with the Department of Electronics, Carleton University. His research interests include artificial neural networks, microwave device and circuit modeling, and CAD of VLSI circuits.

Mr. Devabhaktuni was a two-time recipient of the 1999–2000 and 2000–2001 Ontario Graduate Scholarship (OGS) presented by the Ministry of Education and Training, Ontario, Canada. He was also the recipient of one of the 1999 Best Student Research Exhibit Prizes awarded by Nortel.



Mustapha C. E. Yagoub (M'96) received the Diplôme d'Ingénieur degree in electronics and the Magister degree in telecommunications from the Ecole Nationale Polytechnique, Algiers, Algeria, in 1979 and 1987, respectively, and the Ph.D. degree from the Institut National Polytechnique, Toulouse, France, in 1994.

He was with the Institute of Electronics, Université des Sciences et de la Technologie Houari Boumédienne, Algiers, Algeria, initialy as an Assistant from 1983 to 1991 and then as an Assistant Professor from 1994 to 1999. From 1999 to 2001, he was a Visiting Research Scholar in the Department of Electronics, Carleton University, Ottawa, ON, Canada. He is currently an Assistant Professor in the School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada. His research interests include neural networks, CAD of linear and nonlinear microwave devices and circuits, and applied electromagnetics. He has authored or co-authored over 60 publications in international journals and conferences. He also co-authored *Conception de Circuits Linéaires et Non Linéaires Micro-ondes* (Toulouse, France: Cépadues, 2000).

Qi-Jun Zhang (S'84–M'87–SM'95) received the B.Eng. degree from the East China Engineering Institute, Nanjing, China, in 1982, and the Ph.D. degree in electrical engineering from McMaster University, Hamilton, ON, Canada, in 1987.

From 1982 to 1983, he was with the Systems Engineering Institute, Tianjin University, Tianjin, China. From 1988 to 1990, he was with Optimization Systems Associates Inc. (OSA), Dundas, ON, Canada, where he developed advanced microwave optimization software. In 1990, he joined the Department of Electronics, Carleton University, Ottawa, ON, Canada, where he is currently a Professor. His research interests are neural networks and optimization methods for high-speed/high-frequency circuit design. He has authored and co-authored over 150 papers in international conferences and journals. He also co-authored *Neural Networks for RF and Microwave Design* (Norwood, MA: Artech House, 2000), co-edited *Modeling and Simulation of High-Speed VLSI Interconnects* (Boston, MA: Kluwer,1994), and contributed to *Analog Methods for Computer-Aided Analysis and Diagnosis* (New York: Marcel Dekker, 1988). He was a guest co-editor for a "Special Issue on High-Speed VLSI Interconnects" of the *International Journal of Analog Integrated Circuits and Signal Processing* (1994) and a guest editor for the first "Special Issue on Applications of ANN to RF and Microwave Design" of the *International Journal of RF and Microwave CAE* (1999). He is also a guest editor for the second "Special Issue on Applications of ANN to RF and Microwave Design" of the *International Journal of RF and Microwave CAE* (scheduled for publication in 2001–2002).

Dr. Zhang is a member of the Association of Professional Engineers of the Province of Ontario, Canada.