3

Neural Network Structures

This chapter describes various types of neural network structures that are useful for RF and microwave applications. The most commonly used neural network configurations, known as multilayer perceptrons (MLP), are described first, together with the concept of basic backpropagation training, and the universal approximation theorem. Other structures discussed in this chapter include radial basis function (RBF) network, wavelet neural network, and self-organizing maps (SOM). Brief reviews of arbitrary structures for ANNs and recurrent neural networks are also included.

3.1 Introduction

A neural network has at least two physical components, namely, the processing elements and the connections between them. The processing elements are called neurons, and the connections between the neurons are known as links. Every link has a weight parameter associated with it. Each neuron receives stimulus from the neighboring neurons connected to it, processes the information, and produces an output. Neurons that receive stimuli from outside the network (i.e., not from neurons of the network) are called input neurons. Neurons whose outputs are used externally are called output neurons. Neurons that receive stimuli from other neurons and whose output is a stimulus for other neurons in the neural network are known as hidden neurons. There are different ways in which information can be processed by a neuron, and different ways of connecting the neurons to one another. Different neural network structures can be constructed by using different processing elements and by the specific manner in which they are connected.

A variety of neural network structures have been developed for signal processing, pattern recognition, control, and so on. In this chapter, we describe several neural network structures that are commonly used for microwave modeling and design [1, 2]. The neural network structures covered in this chapter include multilayer perceptrons (MLP), radial basis function networks (RBF), wavelet neural networks, arbitrary structures, self-organizing maps (SOM), and recurrent networks.

3.1.1 Generic Notation

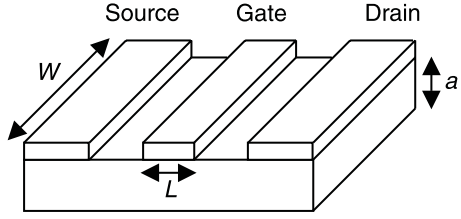
Let n and m represent the number of input and output neurons of the neural network. Let $\boldsymbol{x}$ be an n-vector containing the external inputs (stimuli) to the neural network, $\boldsymbol{y}$ be an m-vector containing the outputs from the output neurons, and $\boldsymbol{w}$ be a vector containing all the weight parameters representing the connections in the neural network. The function $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$ mathematically represents a neural network. The definition of $\boldsymbol{w}$ and the manner in which $\boldsymbol{y}$ is computed from $\boldsymbol{x}$ and $\boldsymbol{w}$, determine the structure of the neural network.

To illustrate the notation, we consider the neural network model of an FET shown in Figure 3.1. The inputs and outputs of the FET neural model are given by,

$$\boldsymbol{x} = [L \; W \; a \; N_d \; V_{GS} \; V_{DS} \; freq]^T \tag{3.1}$$

$$\boldsymbol{y} = [MS_{11} \; PS_{11} \; MS_{12} \; PS_{12} \; MS_{21} \; PS_{21} \; MS_{22} \; PS_{22}]^T \tag{3.2}$$

where *freq* is frequency, and MS_{ij} and PS_{ij} represent the magnitude and phase of the S-parameter S_{ij}. The input vector $\boldsymbol{x}$ contains physical/process/



Gate length:	L
Gate width:	W
Channel thickness:	a
Doping density:	N_d
Bias:	V_{GS}, V_{DS}

Figure 3.1 A physics-based FET.

bias parameters of the FET. The original physics-based FET problem can be expressed as

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \tag{3.3}$$

The neural network model for the problem is

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) \tag{3.4}$$

3.1.2 Highlights of the Neural Network Modeling Approach

In the FET example above, the neural network will represent the FET behavior only after learning the original $\boldsymbol{x} - \boldsymbol{y}$ relationship through a process called *training*. Samples of $(\boldsymbol{x}, \boldsymbol{y})$ data, called *training data*, should first be generated from original device physics simulators or from device measurements. Training is done to determine neural network internal weights $\boldsymbol{w}$ such that the neural model output best matches the training data. A trained neural network model can then be used during microwave design providing answers to the task it learned. In the FET example, the trained model can be used to provide S-parameters from device physical/geometrical and bias values during circuit design.

To further highlight features of the neural network modeling approach, we contrast it with two broad types of conventional microwave modeling approaches. The first type is the detailed modeling approach such as EM-based models for passive components and physics-based models for active components. The overall model, ideally, is defined by well-established theory and no experimental data is needed for model determination. However, such detailed models are usually computationally expensive. The second type of conventional modeling uses empirical or equivalent circuit-based models for passive and active components. The models are typically developed using a mixture of simplified component theory, heuristic interpretation and representations, and fitting of experimental data. The evaluation of these models is usually much faster than that of the detailed models. However, the empirical and equivalent circuit models are often developed under certain assumptions in theory, range of parameters, or type of components. The models have limited accuracy especially when used beyond original assumptions. The neural network approach is a new type of modeling approach where the model can be developed by learning from accurate data of the original component. After training, the neural network becomes a fast and accurate model of the original problem it learned. A summary of these aspects is given in Table 3.1.

An in-depth description of neural network training, its applications in modeling passive and active components and in circuit optimization will be

Table 3.1
A Comparison of Modeling Approaches for RF/Microwave Applications

Basis for Comparison	EM/Physics Models	Empirical and Equivalent Circuit Models	Pure Neural Network Models
Speed	Slow	Fast	Fast
Accuracy	High	Limited	Could be close to EM/physics models
Number of training data	0	A few	Sufficient training data is required, which could be large for high-dimensional problems
Circuit/EM theory of the problem	Maxwell, or semiconductor equations	Partially involved	Not involved

described in subsequent chapters. In the present chapter, we describe structures of neural networks, that is, the various ways of realizing $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$. The structural issues have an impact on model accuracy and cost of model development.

3.2 Multilayer Perceptrons (MLP)

Multilayer perceptrons (MLP) are the most popular type of neural networks in use today. They belong to a general class of structures called feedforward neural networks, a basic type of neural network capable of approximating generic classes of functions, including continuous and integrable functions [3]. MLP neural networks have been used in a variety of microwave modeling and optimization problems.

3.2.1 MLP Structure

In the MLP structure, the neurons are grouped into layers. The first and last layers are called input and output layers respectively, because they represent inputs and outputs of the overall network. The remaining layers are called hidden layers. Typically, an MLP neural network consists of an input layer, one or more hidden layers, and an output layer, as shown in Figure 3.2.

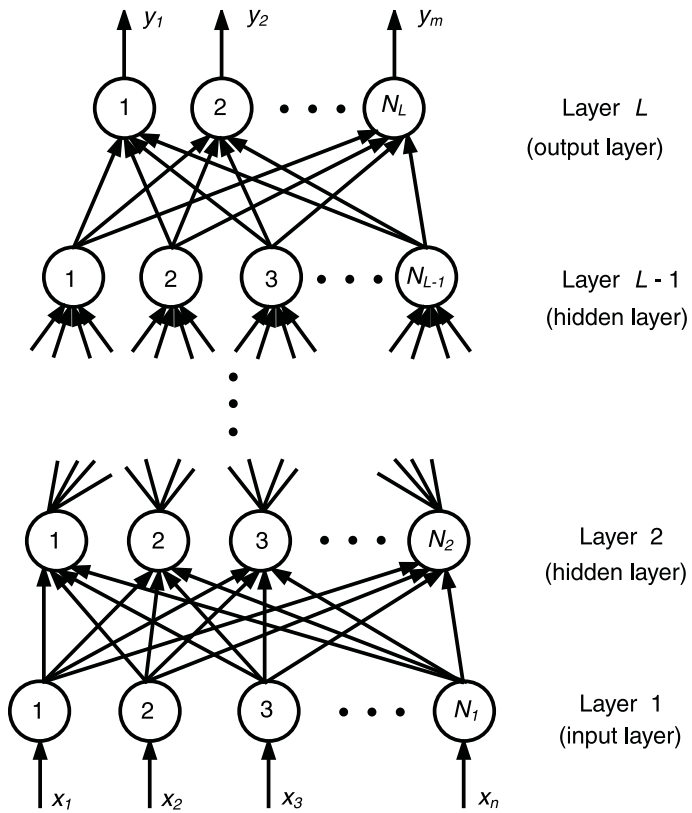


Figure 3.2 Multilayer perceptrons (MLP) structure.

Suppose the total number of layers is L. The 1st layer is the input layer, the Lth layer is the output layer, and layers 2 to $L-1$ are hidden layers. Let the number of neurons in lth layer be N_l, $l = 1, 2, \ldots, L$.

Let w_{ij}^{l} represent the weight of the link between jth neuron of $l-1$th layer and ith neuron of lth layer, $1 \le j \le N_{l-1}$, $1 \le i \le N_l$. Let x_i represent the ith external input to the MLP, and z_i^l be the output of ith neuron of lth layer. We introduce an extra weight parameter for each neuron, w_{i0}^{l}, representing the bias for ith neuron of lth layer. As such, $\boldsymbol{w}$ of MLP includes w_{ij}^{l}, $j = 0, 1, \ldots, N_{l-1}$, $i = 1, 2, \ldots, N_l$, $l = 2, 3, \ldots, L$, that is,

$$\boldsymbol{w} = [w_{10}^{2}\ w_{11}^{2}\ w_{12}^{2} \ldots, w_{N_L N_{L-1}}^{L}]^T \tag{3.5}$$

3.2.2 Information Processing by a Neuron

In a neural network, each neuron—with the exception of neurons at the input layer—receives and processes stimuli (inputs) from other neurons. The

processed information is available at the output end of the neuron. Figure 3.3 illustrates the way in which each neuron in an MLP processes the information. As an example, a neuron of the lth layer receives stimuli from the neurons of $l-1$th layer, that is, $z_1^{l-1}, z_2^{l-1}, \ldots, z_{N_{l-1}}^{l-1}$. Each input is first multiplied by the corresponding weight parameter, and the resulting products are added to produce a weighted sum γ. This weighted sum is passed through a neuron activation function $\sigma(\cdot)$ to produce the final output of the neuron. This output z_i^l can, in turn, become the stimulus for neurons in the next layer.

3.2.3 Activation Functions

The most commonly-used hidden neuron activation function is the sigmoid function given by

$$\sigma(\gamma) = \frac{1}{(1 + e^{-\gamma})} \tag{3.6}$$

As shown in Figure 3.4, the sigmoid function is a smooth switch function having the property of

$$\sigma(\gamma) \rightarrow \begin{cases} 1 & \text{as } \gamma \rightarrow +\infty \\ 0 & \text{as } \gamma \rightarrow -\infty \end{cases}$$

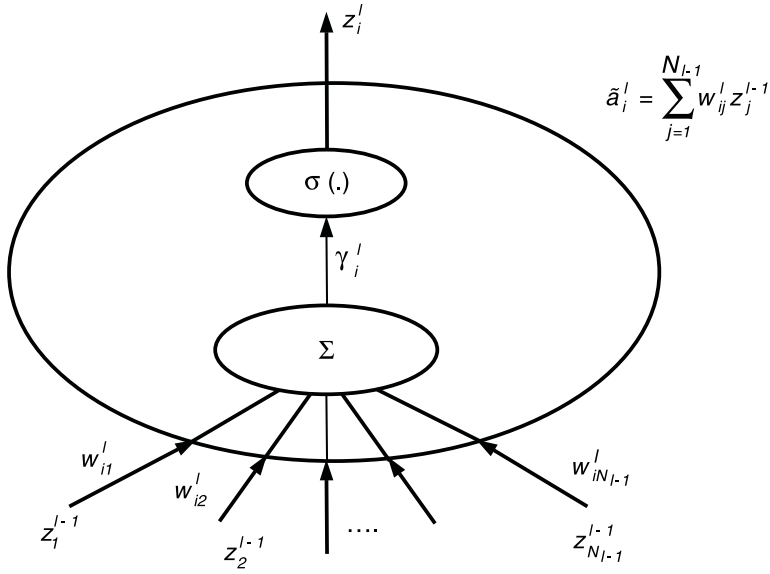


Figure 3.3 Information processing by ith neuron of lth layer.

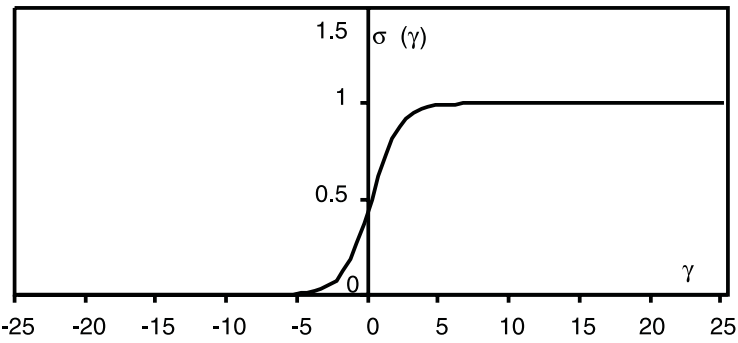


Figure 3.4 Sigmoid function.

Other possible hidden neuron activation functions are the arc-tangent function shown in Figure 3.5 and given by

$$\sigma(\gamma) = \left(\frac{2}{\pi}\right)\arctan(\gamma) \tag{3.7}$$

and the hyperbolic-tangent function shown in Figure 3.6 and given by

$$\sigma(\gamma) = \frac{(e^{\gamma} - e^{-\gamma})}{(e^{\gamma} + e^{-\gamma})} \tag{3.8}$$

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

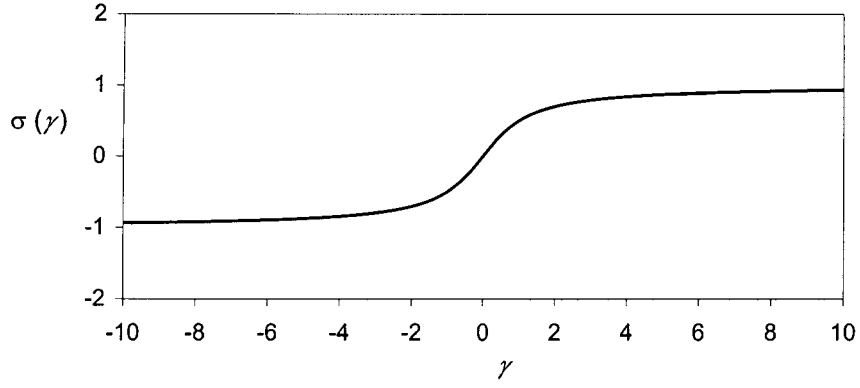


Figure 3.5 Arc-tangent function.

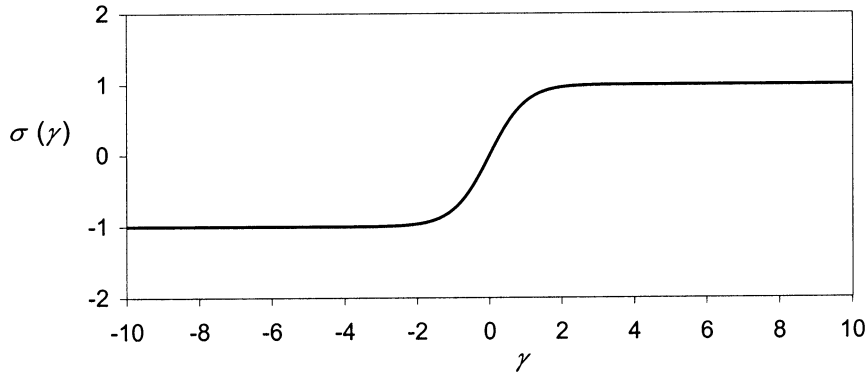


Figure 3.6 Hyperbolic-tangent function.

The input neurons simply relay the external stimuli to the hidden layer neurons; that is, the input neuron activation function is a relay function, $z_i^1 = x_i$, $i = 1, 2, \ldots, n$, and $n = N_1$. As such, some researchers only count the hidden and output layer neurons as part of the MLP. In this book, we follow a convention, wherein the input layer neurons are also considered as part of the overall structure. The activation functions for output neurons can either be logistic functions (e.g., sigmoid), or simple linear functions that compute the weighted sum of the stimuli. For RF and microwave modeling problems, where the purpose is to model continuous electrical parameters, linear activation functions are more suitable for output neurons. The linear activation function is defined as

$$\sigma(\gamma) = \gamma = \sum_{j=0}^{N_{L-1}} w_{ij}^{L} z_j^{L-1} \tag{3.9}$$

The use of linear activation functions in the output neurons could help to improve the numerical conditioning of the neural network training process described in Chapter 4.

3.2.4 Effect of Bias

The weighted sum expressed as

$$\gamma_i^l = w_{i1}^l z_1^{l-1} + w_{i2}^l z_2^{l-1} + \ldots + w_{iN_{l-1}}^l z_{N_{l-1}}^{l-1} \tag{3.10}$$

is zero, if all the previous hidden layer neuron responses (outputs) $z_1^{l-1}, z_2^{l-1}, \ldots, z_{N_{l-1}}^{l-1}$ are zero. In order to create a bias, we assume a fictitious neuron whose output is

$$z_0^{l-1} = 1 \tag{3.11}$$

and add a weight parameter w_{i0}^{l-1} called bias. The weighted sum can then be written as

$$\gamma_i^l = \sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1} \tag{3.12}$$

The effect of adding the bias is that the weighted sum is equal to the bias when all the previous hidden layer neuron responses are zero, that is,

$$\gamma_i^l = w_{i0}^l, \text{ if } z_1^{l-1} = z_2^{l-1} = \ldots = z_{N_{l-1}}^{l-1} = 0 \tag{3.13}$$

The parameter w_{i0}^l is the bias value for ith neuron in lth layer as shown in Figure 3.7.

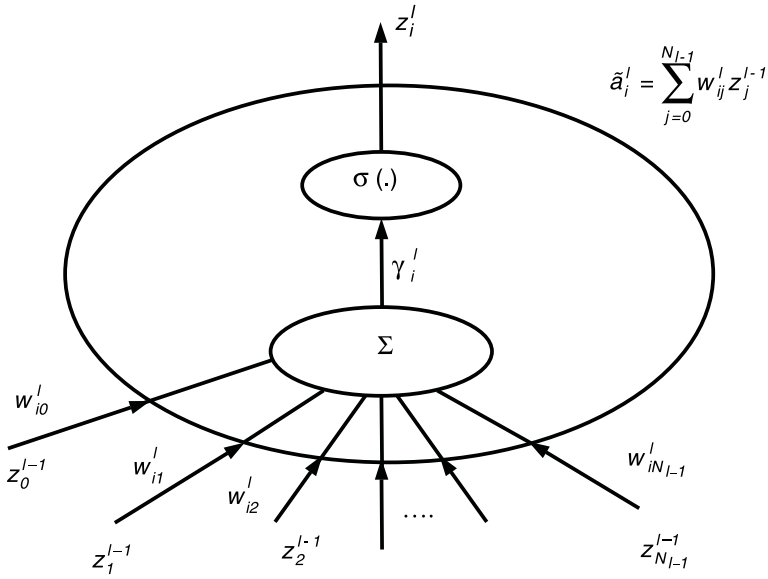


Figure 3.7 A typical ith hidden neuron of lth layer with an additional weight parameter called bias.

3.2.5 Neural Network Feedforward

Given the inputs $\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_n]^T$ and the weights $\boldsymbol{w}$, neural network feedforward is used to compute the outputs $\boldsymbol{y} = [y_1\ y_2\ \ldots\ y_m]^T$ from a MLP neural network. In the feedforward process, the external inputs are first fed to the input neurons (*1*st layer), the outputs from the input neurons are fed to the hidden neurons of the *2*nd layer, and so on, and finally the outputs of $L - 1$th layer are fed to the output neurons (Lth layer). The computation is given by,

$$z_i^1 = x_i,\ i = 1, 2, \ldots, N_1,\ n = N_1 \tag{3.14}$$

$$z_i^l = \sigma\left(\sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1}\right),\ i = 1, 2, \ldots, N_l,\ l = 2, 3, \ldots, L \tag{3.15}$$

The outputs of the neural network are extracted from the output neurons as

$$y_i = z_i^L,\ i = 1, 2, \ldots, N_L,\ m = N_L \tag{3.16}$$

During feedforward computation, the neural network weights $\boldsymbol{w}$ are fixed. As an example, consider a circuit with four transmission lines shown in Figure 3.8. Given the circuit design parameters, $\boldsymbol{x} = [l_1\ l_2\ l_3\ l_4\ R_1\ R_2\ R_3\ R_4\ C_1\ C_2\ C_3\ C_4\ V_{peak}\ \tau_{rise}]^T$, where V_{peak} and τ_{rise} are the peak amplitude and rise time of the source voltage, the signal delays at four output nodes *A, B, C, D*, represented by the output vector $\boldsymbol{y} = [\tau_1\ \tau_2\ \tau_3\ \tau_4]^T$ need to be computed. The original problem $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is a nonlinear relationship between the circuit parameters and the delays. The conventional way to compute the delays is to solve the Kirchoff's current/voltage equations of the circuit, and this process is CPU-intensive, especially if the delay has to be evaluated repetitively for different $\boldsymbol{x}$. In the neural network approach, a neural model can be developed such that each input neuron corresponds to a circuit parameter in $\boldsymbol{x}$, and each output neuron represents a signal delay in $\boldsymbol{y}$. The weights $\boldsymbol{w}$ in the model $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$ are determined through a neural network training process. The model is used to compute the signal delays $\boldsymbol{y}$ for given values of $\boldsymbol{x}$ using neural network feedforward operation. The feedforward computation involves simple sum, product, and sigmoid evaluations, and not the explicit Kirchoff's current/voltage equations. A question arises: can such a simple feedforward computation represent the complicated Kirchoff's current/voltage equations or maybe even the Maxwell's 3-D EM

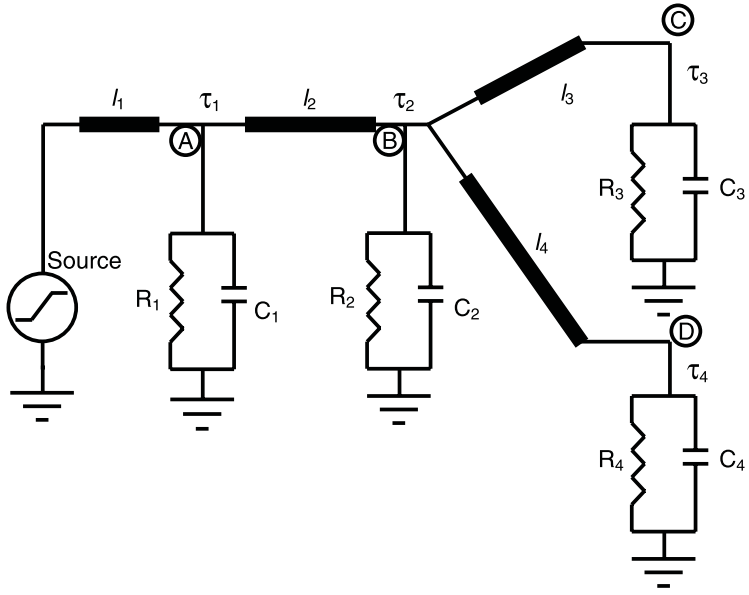


Figure 3.8 A circuit with four transmission lines.

equations? The universal approximation theorem presented in the following sub-section answers this exciting question.

3.2.6 Universal Approximation Theorem

The universal approximation theorem for MLP was proved by Cybenko [4] and Hornik et al. [5], both in 1989. Let $\boldsymbol{I}_n$ represent an n-dimensional unit cube containing all possible input samples $\boldsymbol{x}$, that is, $x_i \in [0, 1]$, $i = 1, 2, \ldots, n$, and $C(\boldsymbol{I}_n)$ be the space of continuous functions on $\boldsymbol{I}_n$. If $\sigma(\cdot)$ is a continuous sigmoid function, the universal approximation theorem states that the finite sums of the form

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{N_2} w_{ki}^3 \sigma\left(\sum_{j=0}^{n} w_{ij}^2 x_j \right) \quad k = 1, 2, \ldots, m \tag{3.17}$$

are dense in $C(\boldsymbol{I}_n)$. In other words, given any $f \in C(\boldsymbol{I}_n)$ and $\epsilon > 0$, there is a sum $y(\boldsymbol{x}, \boldsymbol{w})$ of the above form that satisfies $|y(\boldsymbol{x}, \boldsymbol{w}) - f(\boldsymbol{x})| < \epsilon$ for all $\boldsymbol{x} \in \boldsymbol{I}_n$. As such, there always exists a 3-layer perceptron that can approximate an arbitrary nonlinear, continuous, multi-dimensional function f with any desired accuracy.

However, the theorem does not state how many neurons are needed by the three-layer MLP to approximate the given function. As such, failure to develop an accurate neural model can be attributed to an inadequate number of hidden neurons, inadequate learning/training, or presence of a stochastic rather than a deterministic relation between inputs and outputs [5].

We illustrate how an MLP model matches an arbitrary one-dimensional function shown in Figure 3.9. A drop in the function y from 2.0 to 0.5 as x changes from 5 to 15, corresponds to a sigmoid $\sigma(-(x - 10))$ scaled by a factor of 1.5. On the other hand, a slower increase in the function from 0.5 to 4.5 as x changes from 20 to 60, corresponds to a sigmoid $\sigma(0.2(x - 40))$ scaled by a factor 4 (= 4.5 − 0.5). Finally, the overall function is shifted upwards by a bias of 0.5. The function can then be written as

$$\begin{aligned} y = y(\boldsymbol{x}, \boldsymbol{w}) &= 0.5 + 1.5\sigma(-(x - 10)) + 4\sigma(0.2(x - 40)) \\ &= 0.5 + 1.5\sigma(-x + 10) + 4\sigma(0.2x - 8) \end{aligned} \tag{3.18}$$

and the structure of the neural network model is shown in Figure 3.10. In practice, the optimal values of the weight parameters $\boldsymbol{w}$ are obtained by a training process, which adjusts $\boldsymbol{w}$ such that the error between the neural model outputs and the original problem outputs is minimized.

Discussion

This example is an illustration of how a neural network approximates a simple function in a manner similar to polynomial curve-fitting. The real power of neural networks, however, lies in its modeling capacity when the nonlinearity and dimensionality of the original problem increases. In such cases, curve-fitting techniques using higher-order, higher-dimensional polynomial functions are very cumbersome and ineffective. Neural network models can handle such

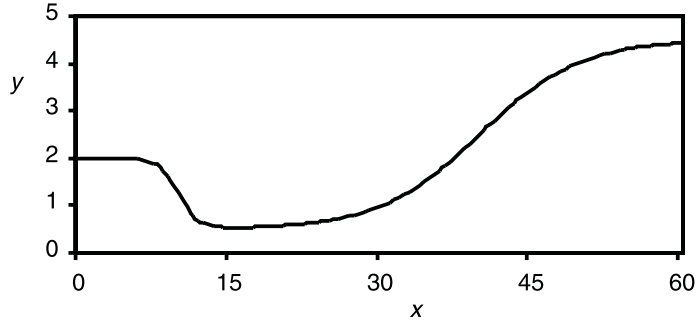


Figure 3.9 A one-dimensional function to be modeled by an MLP.

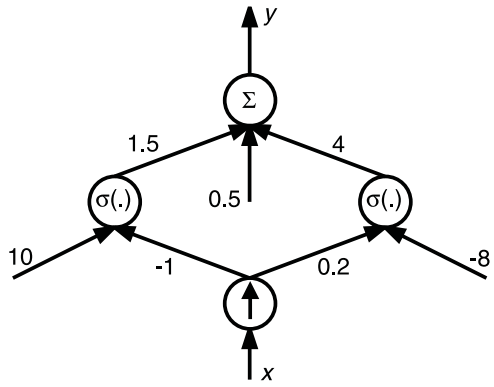


Figure 3.10 A neural network model for the function in Figure 3.9. In this figure, an arrow inside the input neuron means that the input neuron simply relays the value of the input (x) to the network. The hidden neurons use sigmoid activation functions, while the output neurons use linear functions.

problems more effectively, and can be accurate over a larger region of the input space. The most significant features of neural networks are:

- Neural networks are distributed models by nature. In other words, no single neuron can produce the overall ***x-y*** relationship. Each neuron is a simple processing element with switching activation function. Many neurons combined produce the overall ***x-y*** relationship. For a given value of external stimuli, some neurons are switched on, some are off, while others are in transition. It is the rich combination of the neuron switching states responding to different values of external stimuli that enables the network to represent a nonlinear input-output mapping.
- Neural networks have a powerful learning capability, that is, they can be trained to represent any given problem behavior. The weight parameters in the neural network represent the weighted connections between neurons. After training the neural network, the weighted connections capture/encode the problem information from the raw training data. Neural networks with different sets of weighted connections can represent a diverse range of input-output mapping problems.

3.2.7 Number of Neurons

The universal approximation theorem states that there exists a three-layer MLP that approximates virtually any nonlinear function. However, it did not specify

what size the network should be (i.e., number of hidden neurons) for a given problem complexity. The precise number of hidden neurons required for a modeling task remains an open question. Although there is no clear-cut answer, the number of hidden neurons depends on the degree of nonlinearity and the dimensionality of the original problem. Highly nonlinear problems need more neurons and smoother problems need fewer neurons. Too many hidden neurons may lead to overlearning of the neural network, which is discussed in Chapter 4. On the other hand, fewer hidden neurons will not give sufficient freedom to the neural network to accurately learn the problem behavior. There are three possible solutions to address the question regarding network size. First, experience can help determine the number of hidden neurons, or the optimal size of the network can be obtained through a trial and error process. Second, the appropriate number of neurons can be determined by an adaptive process—or optimization process—that adds/deletes neurons as needed during training [6]. Finally, the ongoing research in this direction includes techniques such as constructive algorithms [7], network pruning [8], and regularization [9], to match the neural network model complexity with problem complexity.

3.2.8 Number of Layers

Neural networks with at least one hidden layer are necessary and sufficient for arbitrary nonlinear function approximation. In practice, neural networks with one or two hidden layers, that is, three-layer or four-layer perceptrons (including input and output layers) are commonly used for RF/microwave applications. Intuitively, four-layer perceptrons would perform better in modeling nonlinear problems where certain localized behavioral components exist repeatedly in different regions of the problem space. A three-layer perceptron neural network—although capable of modeling such problems—may require too many hidden neurons. Literature that favors both three-layer perceptrons and four-layer perceptrons does exist [10, 11]. The performance of a neural network can be evaluated in terms of generalization capability and mapping capability [11]. In the function approximation or regression area where generalization capability is a major concern, three-layer perceptrons are usually preferred [10], because the resulting network usually has fewer hidden neurons. On the other hand, four-layer perceptrons are favored in pattern classification tasks where decision boundaries need to be defined [11], because of their better mapping capability. Structural optimization algorithms that determine the optimal number of layers according to the training data have also been investigated [12, 13].

3.3 Back Propagation (BP)

The main objective in neural model development is to find an optimal set of weight parameters $\boldsymbol{w}$, such that $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$ closely represents (approximates) the original problem behavior. This is achieved through a process called training (that is, optimization in $\boldsymbol{w}$-space). A set of training data is presented to the neural network. The training data are pairs of $(\boldsymbol{x}_k, \boldsymbol{d}_k)$, $k = 1, 2, \ldots, P$, where $\boldsymbol{d}_k$ is the desired outputs of the neural model for inputs $\boldsymbol{x}_k$, and P is the total number of training samples.

During training, the neural network performance is evaluated by computing the difference between actual neural network outputs and desired outputs for all the training samples. The difference, also known as the error, is quantified by

$$E = \frac{1}{2} \sum_{k \in T_r} \sum_{j=1}^{m} (y_j(\boldsymbol{x}_k, \boldsymbol{w}) - d_{jk})^2 \tag{3.19}$$

where d_{jk} is the jth element of $\boldsymbol{d}_k$, $y_j(\boldsymbol{x}_k, \boldsymbol{w})$ is the jth neural network output for input $\boldsymbol{x}_k$, and T_r is an index set of training data. The weight parameters $\boldsymbol{w}$ are adjusted during training, such that this error is minimized. In 1986, Rumelhart, Hinton, and Williams [14] proposed a systematic neural network training approach. One of the significant contributions of their work is the error back propagation (BP) algorithm.

3.3.1 Training Process

The first step in training is to initialize the weight parameters $\boldsymbol{w}$, and small random values are usually suggested. During training, $\boldsymbol{w}$ is updated along the negative direction of the gradient of E, as $\boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial E}{\partial \boldsymbol{w}}$, until E becomes small enough. Here, the parameter η is called the learning rate. If we use just one training sample at a time to update $\boldsymbol{w}$, then a per-sample error function E_k given by

$$E_k = \frac{1}{2} \sum_{j=1}^{m} (y_j(\boldsymbol{x}_k, \boldsymbol{w}) - d_{jk})^2 \tag{3.20}$$

is used and $\boldsymbol{w}$ is updated as $\boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial E_k}{\partial \boldsymbol{w}}$. The following sub-section describes how the error back propagation process can be used to compute the gradient information $\frac{\partial E_k}{\partial \boldsymbol{w}}$.

3.3.2 Error Back Propagation

Using the definition of E_k in (3.20), the derivative of E_k with respect to the weight parameters of the lth layer can be computed by simple differentiation as

$$\frac{\partial E_k}{\partial w_{ij}^l} = \frac{\partial E_k}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial w_{ij}^l} \tag{3.21}$$

and

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \frac{\partial \sigma}{\partial \gamma_i^l} \cdot z_j^{l-1} \tag{3.22}$$

The gradient $\frac{\partial E_k}{\partial z_i^l}$ can be initialized at the output layer as

$$\frac{\partial E_k}{\partial z_i^L} = (y_i(\boldsymbol{x}_k, \boldsymbol{w}) - d_{ik}) \tag{3.23}$$

using the error between neural network outputs and desired outputs (training data). Subsequent derivatives $\frac{\partial E_k}{\partial z_i^l}$ are computed by back-propagating this error from $l + 1$th layer to lth layer (see Figure 3.11) as

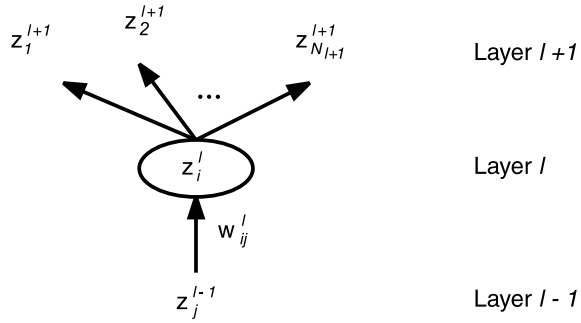


Figure 3.11 The relationship between ith neuron of lth layer, with neurons of layers $l - 1$ and $l + 1$.

$$\frac{\partial E_k}{\partial z_i^l} = \sum_{j=1}^{N_{l+1}} \frac{\partial E_k}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial z_i^l} \tag{3.24}$$

For example, if the MLP uses sigmoid (3.6) as hidden neuron activation function,

$$\frac{\partial \sigma}{\partial \gamma} = \sigma(\gamma)(1 - \sigma(\gamma)) \tag{3.25}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = z_i^l(1 - z_i^l)z_j^{l-1} \tag{3.26}$$

and

$$\frac{\partial z_i^l}{\partial z_j^{l-1}} = z_i^l(1 - z_i^l)w_{ij}^l \tag{3.27}$$

For the same MLP network, let δ_i^l be defined as $\delta_i^l = \dfrac{\partial E_k}{\partial \gamma_i^l}$ representing local gradient at ith neuron of lth layer. The back propagation process is then given by,

$$\delta_i^L = (y_i(\boldsymbol{x}_k, \boldsymbol{w}) - d_{ik}) \tag{3.28}$$

$$\delta_i^l = \left(\sum_{j=1}^{N_{l+1}} \delta_j^{l+1} w_{ji}^{l+1} \right) z_i^l(1 - z_i^l),\ l = L - 1, L - 2, \ldots 2 \tag{3.29}$$

and the derivatives with respect to the weights are

$$\frac{\partial E_k}{\partial w_{ij}^l} = \delta_i^l z_j^{l-1}\ l = L, L - 1, \ldots, 2 \tag{3.30}$$

3.4 Radial Basis Function Networks (RBF)

Feedforward neural networks with a single hidden layer that use radial basis activation functions for hidden neurons are called radial basis function (RBF) networks. RBF networks have been applied to various microwave modeling

purposes—for example, to model intermodulation distortion behavior of MESFETs and HEMTs [15].

3.4.1 RBF Network Structure

A typical radial-basis–function neural network is shown in Figure 3.12. The RBF neural network has an input layer, a radial basis hidden layer, and an output layer.

The parameters c_{ij}, λ_{ij}, are centers and standard deviations of radial basis activation functions. Commonly used radial basis activation functions are Gaussian and multiquadratic. The Gaussian function shown in Figure 3.13 is given by

$$\sigma(\gamma) = \exp(-\gamma^2) \tag{3.31}$$

The multiquadratic function shown in Figure 3.14 is given by

$$\sigma(\gamma) = \frac{1}{(c^2 + \gamma^2)^{\alpha}}, \ \alpha > 0 \tag{3.32}$$

where c is a constant.

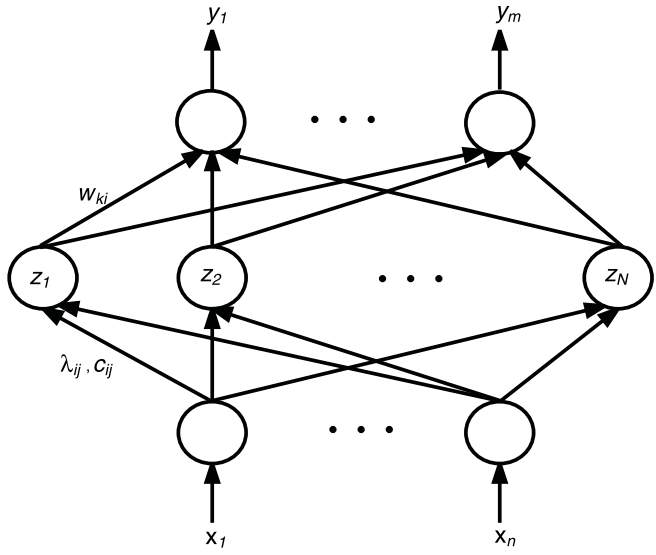


Figure 3.12 RBF neural network structure.

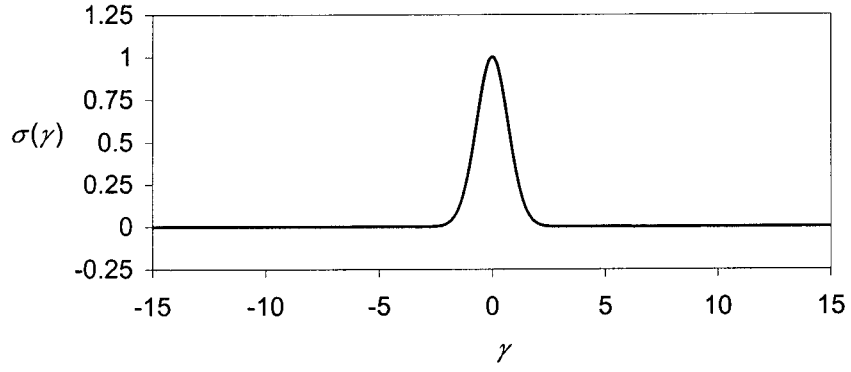


Figure 3.13 Gaussian function.

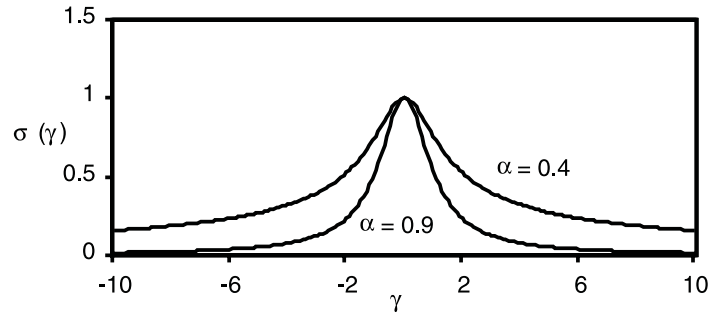


Figure 3.14 Multiquadratic function with $c = 1$.

3.4.2 Feedforward Computation

Given the inputs $\boldsymbol{x}$, the total input to the ith hidden neuron γ_i is given by

$$\gamma_i = \sqrt{\sum_{j=1}^{n} \left(\frac{x_j - c_{ij}}{\lambda_{ij}} \right)^2}, \quad i = 1, 2, \ldots, N \tag{3.33}$$

where N is the number of hidden neurons. The output value of the ith hidden neuron is $z_i = \sigma(\gamma_i)$, where $\sigma(\gamma)$ is a radial basis function. Finally, the outputs of the RBF network are computed from hidden neurons as

$$y_k = \sum_{i=0}^{N} w_{ki} z_i, \quad k = 1, 2, \ldots, m \tag{3.34}$$

where w_{ki} is the weight of the link between ith neuron of the hidden layer and kth neuron of the output layer. Training parameters $\boldsymbol{w}$ of the RBF

network include w_{k0}, w_{ki}, c_{ij}, λ_{ij}, $k = 1, 2, \ldots, m$, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, n$.

For illustration, we use an RBF network to approximate the one-dimensional function shown in Figure 3.15. The function has a narrow peak at $x = 2$, that can be approximated by a Gaussian function $\sigma(x - 2)$. The wider valley at $x = 9$ is represented by a Gaussian $\sigma[(x - 9)/3]$ scaled by a factor of -2. Finally, a bias of value 1 is added. As such, an RBF network with two hidden neurons given by

$$y = \sigma(x - 2) - 2\sigma\left(\frac{x - 9}{3}\right) + 1 \tag{3.35}$$

can approximate this function. The RBF network structure for this example is shown in Figure 3.16. In practice, the RBF weight parameters w_{ki}, c_{ij}, λ_{ij}, are determined through a training process.

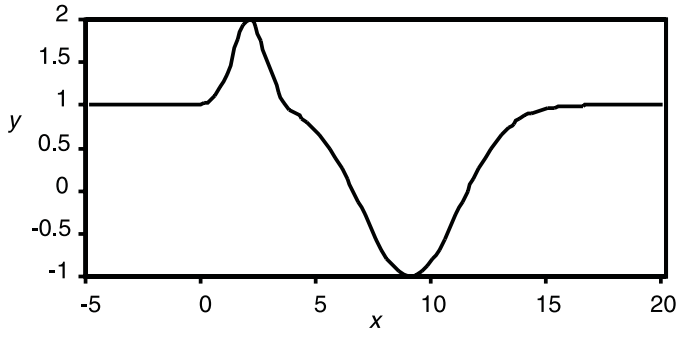


Figure 3.15 A one-dimensional function to be modeled by an RBF network.

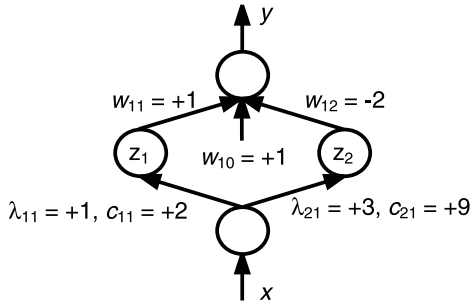


Figure 3.16 The RBF network structure for the function in Figure 3.15.

3.4.3 Universal Approximation Theorem

In 1991, Park and Sandberg proved the universal approximation theorem for RBF networks [16]. According to their work, an RBF neural network with a sufficient number of hidden neurons is capable of approximating any given nonlinear function to any degree of accuracy. In this section, we provide a simplified interpretation of the theorem. Let R^n be the space of n-dimensional real-valued vectors; $C(R^n)$ be the space of continuous functions on R^n; and μ be a finite measure on R^n. Given any function $f \in C(R^n)$ and a constant $\epsilon > 0$, there is a sum $y(\boldsymbol{x}, \boldsymbol{w})$ of the form (3.34), for which $|f(\boldsymbol{x}) - y(\boldsymbol{x}, \boldsymbol{w})| < \epsilon$ for almost all $\boldsymbol{x} \in R^n$. The size μ of the $\boldsymbol{x}$ region for which $|f(\boldsymbol{x}) - y(\boldsymbol{x}, \boldsymbol{w})| > \epsilon$ is very small, that is, $\mu < \epsilon$.

3.4.4 Two-Step Training of RBF Networks

Step 1: The centers c_{ij} of the hidden neuron activation functions can be initialized using a clustering algorithm. This step is called the unsupervised training and the purpose is to find the centers of clusters in the training sample distribution. A detailed explanation of the clustering algorithm is presented in Section 3.8. This provides better initial values for hidden neuron centers as compared to random initialization. While clustering for pattern-classification applications are well-defined [17], the question of effectively initializing RBF centers for microwave modeling still remains an open task.

Step 2: Update by gradient-based optimization techniques, such as $\boldsymbol{w} = \boldsymbol{w} - \eta \dfrac{\partial E}{\partial \boldsymbol{w}}$, where $\boldsymbol{w}$ includes all the parameters in RBF networks (i.e. λ_{ij}, c_{ij}, and w_{ki}) until neural network learns the training data well. This step is similar to that of MLP training.

3.5 Comparison of MLP and RBF Neural Networks

Both MLP and RBF belong to a general class of neural networks called feedforward networks, where information processing in the network structure follows one direction—from input neurons to output neurons. However, the hidden neuron activation functions in MLP and RBF behave differently. First, the activation function of each hidden neuron in an MLP processes 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 an RBF network processes 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. Conversely, RBF networks use exponentially decaying localized nonlinearities to construct local approximations to nonlinear input-output mapping. As a result, RBF neural networks learn at faster rates and exhibit reduced sensitivity to the order of presentation of training data [17]. In RBF, a hidden neuron influences the network outputs only for those inputs that are near to its center, thus requiring an exponential number of hidden neurons to cover the entire input space. From this perspective, it is suggested in [18] that RBF networks are suitable for problems with a smaller number of inputs.

For the purpose of comparison, both MLP and RBF networks were used to model a physics-based MESFET [2]. The physical, process, and bias parameters of the device (channel length L, channel width W, doping density N_d, channel thickness a, gate-source voltage V_{GS}, drain-source voltage V_{DS}) are neural network inputs. Drain-current (i_d) is the neural network output. Three sets of training data with 100, 300, and 500 samples were generated using OSA90 [19]. A separate set of data with 413 samples, called test data, was also generated by the same simulator in order to test the accuracy of the neural models after training is finished. The accuracy for the trained neural models is measured by the average percentage error of the neural model output versus test data, shown in Table 3.2. As can be seen from the table, RBF networks used more hidden neurons than MLP to achieve similar model accuracy. This can be attributed to the localized nature of radial basis activation functions. When training data is large and sufficient, RBF networks achieve better accuracy than MLP. As the amount of training data becomes less, the performance of the MLP degrades slowly as compared to the RBF networks. This shows that MLP networks have better generalization capability. However, the training process of RBF networks is usually easier to converge than the MLP.

Table 3.2
Model Accuracy Comparison Between MLP and RBF (from [2], Wang, F., et al., "Neural Network Structures and Training Algorithms for RF and Microwave Applications," *Int. J. RF and Microwave CAE,* pp. 216–240, © 1999, John Wiley and Sons. Reprinted with permission from John Wiley and Sons, Inc.).

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

3.6 Wavelet Neural Networks

The idea of combining wavelet theory with neural networks [20–22] resulted in a new type of neural network called wavelet networks. The wavelet networks use wavelet functions as hidden neuron activation functions. Using theoretical features of the wavelet transform, network construction methods can be developed. These methods help to determine the neural network parameters and the number of hidden neurons during training. The wavelet network has been applied to modeling passive and active components for microwave circuit design [23–25].

3.6.1 Wavelet Transform

In this section, we provide a brief summary of the wavelet transform [20]. Let R be a space of real variables and R^n be an n-dimensional space of real variables.

Radial Function

Let $\psi = \psi(\cdot)$ be a function of n variables. The function $\psi(\boldsymbol{x})$ is radial, if there exists a function $g = g(\cdot)$ of a single variable, such that for all $\boldsymbol{x} \in R^n$, $\psi(\boldsymbol{x}) = g(\|\boldsymbol{x}\|)$. If $\psi(\boldsymbol{x})$ is radial, its Fourier transform $\hat{\psi}(\boldsymbol{\omega})$ is also radial.

Wavelet Function

Let $\hat{\psi}(\boldsymbol{\omega}) = \eta(\|\boldsymbol{\omega}\|)$, where η is a function of a single variable. A radial function $\psi(\boldsymbol{x})$ is a wavelet function, if

$$C_\psi = (2\pi)^n \int_0^\infty \frac{|\eta(\xi)|^2}{\xi} d\xi < \infty \tag{3.36}$$

Wavelet Transform

Let $\psi(\boldsymbol{x})$ be a wavelet function. The function can be shifted and scaled as $\psi\left(\frac{\boldsymbol{x} - \boldsymbol{t}}{a}\right)$, where $\boldsymbol{t}$, called the translation (shift) parameter, is an n-dimensional real vector in the same space as $\boldsymbol{x}$, and a is a scalar called the dilation parameter. Wavelet transform of a function $f(\boldsymbol{x})$ is given by

$$W(a, \boldsymbol{t}) = \int_{R^n} f(\boldsymbol{x}) a^{-n/2} \psi\left(\frac{\boldsymbol{x} - \boldsymbol{t}}{a}\right) d\boldsymbol{x} \tag{3.37}$$

The wavelet transform transforms the function from original domain ($\boldsymbol{x}$ domain) into a wavelet domain (a, $\boldsymbol{t}$ domain). A function $f(\boldsymbol{x})$ having both smooth global variations and sharp local variations can be effectively represented in wavelet domain by a corresponding wavelet function $W(a, \boldsymbol{t})$. The original function $f(\boldsymbol{x})$ can be recovered from the wavelet function by using an inverse wavelet transform, defined as

$$f(\boldsymbol{x}) = \frac{1}{C_\Psi}\int_{0^+}^{\infty} a^{-(n+1)} \int_{R^n} W(a, \boldsymbol{t}) a^{-n/2} \psi\left(\frac{\boldsymbol{x}-\boldsymbol{t}}{a}\right) d\boldsymbol{t}da \tag{3.38}$$

The discretized version of the inverse wavelet transform is expressed as

$$f(\boldsymbol{x}) = \sum_i W_i a_i^{-n/2} \psi\left(\frac{\boldsymbol{x}-\boldsymbol{t}_i}{a_i}\right) \tag{3.39}$$

where $(a_i, \boldsymbol{t}_i)$ represent discrete points in the wavelet domain, and W_i is the coefficient representing the wavelet transform evaluated at $(a_i, \boldsymbol{t}_i)$. Based on (3.39), a three-layer neural network with wavelet hidden neurons can be constructed. A commonly used wavelet function is the inverse Mexican-hat function given by

$$\psi\left(\frac{\boldsymbol{x}-\boldsymbol{t}_i}{a}\right) = \sigma(\gamma_i) = (\gamma_i^2 - n)\exp\left(-\frac{\gamma_i^2}{2}\right) \tag{3.40}$$

where

$$\gamma_i = \left\|\frac{\boldsymbol{x}-\boldsymbol{t}_i}{a_i}\right\| = \sqrt{\sum_{j=1}^{n}\left(\frac{x_j - t_{ij}}{a_i}\right)^2} \tag{3.41}$$

A one-dimensional inverse Mexican-hat with $t_{11} = 0$ is shown in Figure 3.17 for two different values of a_1.

3.6.2 Wavelet Networks and Feedforward Computation

Wavelet networks are feedforward networks with one hidden layer, as shown in Figure 3.18. The hidden neuron activation functions are wavelet functions. The output of the ith hidden neuron is given by

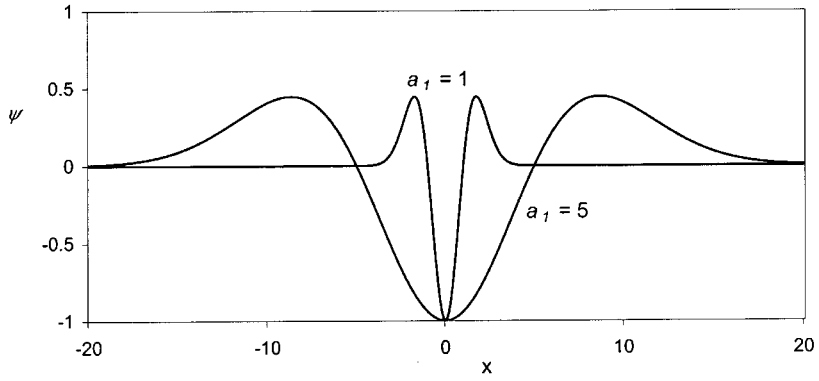


Figure 3.17 Inverse Mexican-hat function.

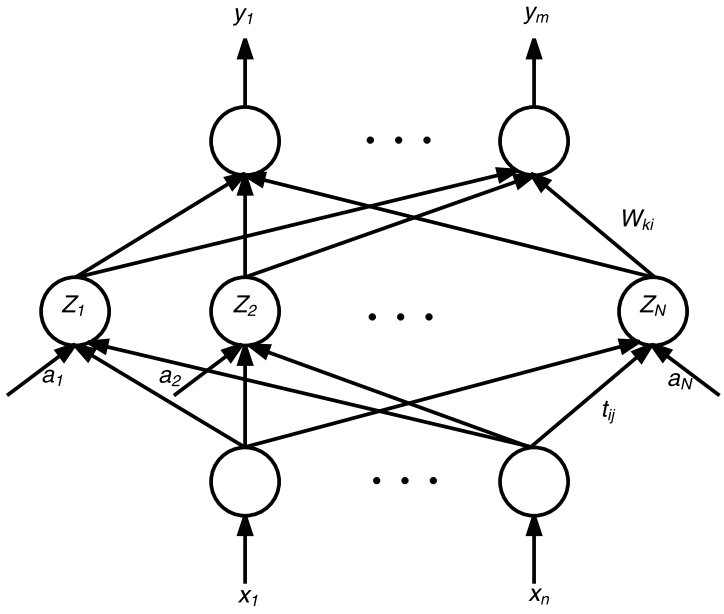


Figure 3.18 Wavelet neural network structure.

$$z_i = \sigma(\gamma_i) = \psi\left(\frac{\boldsymbol{x} - \boldsymbol{t}_i}{a_i}\right),\ i = 1, 2, \ldots, N \tag{3.42}$$

where N is the number of hidden neurons, $\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_n]^T$ is the input vector, $\boldsymbol{t}_i = [t_{i1}\ t_{i2}\ \ldots\ t_{in}]^T$ is the translation parameter, a_i is a dilation parameter, and $\psi(\cdot)$ is a wavelet function. The weight parameters of a wavelet

network $\boldsymbol{w}$ include a_i, t_{ij}, w_{ki}, w_{k0}, $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, n$, $k = 1, 2, \ldots, m$.

In (3.42), γ_i is computed following (3.41). The outputs of the wavelet network are computed as

$$y_k = \sum_{i=0}^{N} w_{ki} z_i, \; k = 1, 2, \ldots, m \tag{3.43}$$

where w_{ki} is the weight parameter that controls the contribution of the ith wavelet function to the kth output. As an illustration, we use a wavelet network to model the one-dimensional function shown in Figure 3.19.

The narrow peak at $x = 4$ that looks like a hat can be approximated by a wavelet function with $t_{11} = 4$, $a_1 = 0.5$, and a scaling factor of -5. The wider valley at $x = 12$ that looks like an inverted hat can be represented by a wavelet with $t_{12} = 12$ and $a_2 = 1$ scaled by 3. As such, two hidden wavelet neurons can model this function. The resulting wavelet network structure is shown in Figure 3.20. In practice, all the parameters shown in the wavelet network are determined by the training process.

3.6.3 Wavelet Neural Network With Direct Feedforward From Input to Output

A variation of the wavelet neural network was used in [23], where additional connections are made directly from input neurons to the output neurons. The outputs of the model are given by

$$y_k = \sum_{i=0}^{N} w_{ki} z_i + \sum_{j=1}^{n} \nu_{kj} x_j, \; k = 1, 2, \ldots, m \tag{3.44}$$

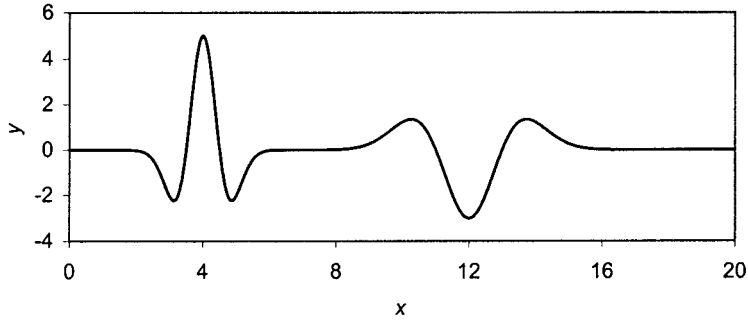


Figure 3.19 A one-dimensional function to be modeled by a wavelet network.

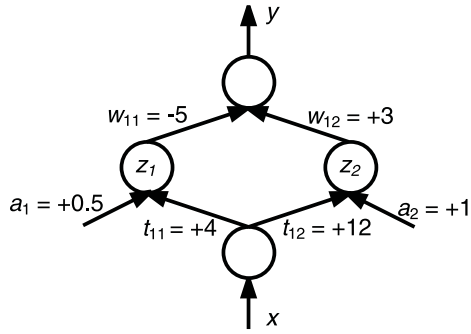


Figure 3.20 Wavelet neural network structure for the one-dimensional function of Figure 3.19.

where z_i is defined in (3.42) and ν_{kj} is the weight linking jth input neuron to the kth output neuron.

3.6.4 Wavelet Network Training

The training process of wavelet networks is similar to that of RBF networks.

Step 1: Initialize translation and dilation parameters of all the hidden neurons, $\boldsymbol{t}_i$, a_i, $i = 1, 2, \ldots, N$.

Step 2: Update the weights $\boldsymbol{w}$ of the wavelet network using a gradient-based training algorithm, such that the error between neural model and training data is minimized. This step is similar to MLP and RBF training.

3.6.5 Initialization of Wavelets

An initial set of wavelets can be generated in two ways. One way is to create a wavelet lattice by systematically using combinations of grid points along dilation and translation dimensions [20]. The grids of the translation parameters are sampled within the model input-space, or $\boldsymbol{x}$-space. The grids for dilation parameters are sampled from the range $(0, a)$, where a is a measure of the "diameter" of the $\boldsymbol{x}$-space that is bounded by the minimum and maximum values of each element in the $\boldsymbol{x}$-vector in training data. Another way to create initial wavelets is to use clustering algorithms to identify centers and widths of clusters in the training data. The cluster centers and width can be used for initializing translation and dilation parameters of the wavelet functions. A detailed description of the clustering algorithm is given in Section 3.8.

The initial number of wavelets can be unnecessarily large, especially if the wavelet set is created using the wavelet lattice approach. This is because of the existence of many wavelets that are redundant with respect to the given problem. This leads to a large number of hidden neurons adversely affecting the training and the generalization performance of the wavelet network. A wavelet reduction algorithm was proposed in [20], which identifies and removes less important wavelets. From the initial set of wavelets, the algorithm first selects the wavelet that best fits the training data, and then repeatedly selects one of the remaining wavelets that best fits the data when combined with all previously selected wavelets. For computational efficiency, the wavelets selected later are ortho-normalized to ones that were selected earlier.

At the end of this process, a set of wavelet functions that best span the training data outputs are obtained. A wavelet network can then be constructed using these wavelets, and the network weight parameters can be further refined by supervised training.

3.7 Arbitrary Structures

All the neural network structures discussed so far have been layered. In this section, we describe a framework that accommodates arbitrary neural network structures. Suppose the total number of neurons in the network is N. The output value of the jth neuron in the network is denoted by z_j, $j = 1, 2, \ldots, N$. The weights of the links between neurons (including bias parameters) can be arranged into an $N \times (N + 1)$ matrix given by

$$\boldsymbol{w} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1N} \\ w_{20} & w_{21} & \cdots & w_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ w_{N0} & w_{N1} & \cdots & w_{NN} \end{bmatrix} \tag{3.45}$$

where the *1*st column represents the bias of each neuron. A weight parameter w_{ij} is nonzero if the output of the jth neuron is a stimulus (input) to the ith neuron. The features of an arbitrary neural network are:

- There is no layer-by-layer connection requirement;
- Any neuron can be linked to any other neuron;
- External inputs can be supplied to any predefined set of neurons;
- External outputs can be taken from any predefined set of neurons.

The inputs to the neural network are $\boldsymbol{x} = [x_1\ x_2\ \ldots\ x_n]^T$, where the input neurons are identified by a set of indices—$i_1, i_2, \ldots, i_n$, $i_k \in \{1, 2, \ldots, N\}$, and $k = 1, 2, \ldots, n$. The outputs of the neural network are $\boldsymbol{y} = [y_1\ y_2\ \ldots\ y_m]^T$, where the output neurons are identified by a set of indices $j_1, j_2, \ldots, j_m$, $j_k \in \{1, 2, \ldots, N\}$, and $k = 1, 2, \ldots, m$. All the neurons could be considered to be in one layer.

This formulation accommodates general feedforward neural network structures. Starting from the input neurons, follow the links to other neurons, and continue following their links and so on. If, in the end, we reach the output neurons without passing through any neuron more than once (i.e., no loop is formed), then the network is a feedforward network since the information processing is one-way, from the input neurons to the output neurons.

Figure 3.21 illustrates the method of mapping a given neural network structure into a weight matrix. In the weight matrix, the ticks represent non-zero value of weights, indicating a link between corresponding neurons. A blank entry in the (i, j)th location of the matrix means there is no link between the output of the jth neuron and the input of the ith neuron. If an entire row (column) is zero, then the corresponding neuron is an input (output)

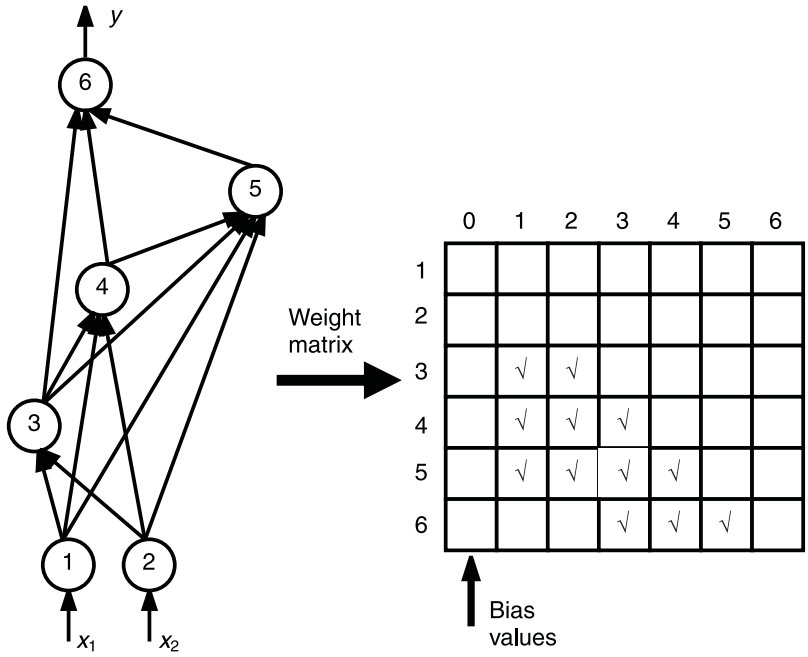


Figure 3.21 Mapping a given arbitrary neural network structure into a weight matrix.

neuron. A neuron is a hidden neuron if its corresponding row and column both have nonzero entries.

Feedforward for an Arbitrary Neural Network Structure

First, identify the input neurons by input neuron indices and feed the external input values to these neurons as

$$\begin{cases} z_{i_1} = x_1 \\ z_{i_2} = x_2 \\ \vdots \\ z_{i_n} = x_n \end{cases} \tag{3.46}$$

Then follow the nonzero weights in the weight matrix to process subsequent neurons, until the output neurons are reached. For example, if the kth neuron uses a sigmoid activation function, it can be processed as

$$\gamma_k = \sum_{j=1}^{N} w_{kj} z_j + w_{k0} \tag{3.47}$$

$$z_k = \sigma(\gamma_k) \tag{3.48}$$

where w_{k0} is the bias for kth neuron and $\sigma(\cdot)$ represents a sigmoid function. Once all the neurons in the network have been processed, the outputs of the network can be extracted by identifying the output neurons through output neuron indices as

$$\begin{cases} y_1 = z_{j1} \\ y_2 = z_{j2} \\ \vdots \\ y_m = z_{jm} \end{cases} \tag{3.49}$$

The arbitrary structures allow neural network structural optimization. The optimization process can add/delete neurons and connections between neurons during training [13, 26].

3.8 Clustering Algorithms and Self-Organizing Maps

In microwave modeling, we encounter situations where developing a single model for the entire input space becomes too difficult. A decomposition

approach is often used to divide the input space, such that within each subspace, the problem is easier to model. A simple example is the large-signal transistor modeling where the model input space is first divided into saturation, breakdown, and linear regions; different equations are then used for each subregion. Such decomposition conventionally involves human effort and experience. In this section, we present a neural network approach that facilitates automatic decomposition through processing and learning of training data. This approach is very useful when the problem is complicated and the precise shape of subspace boundaries not easy to determine. A neural-network–based decomposition of E-plane waveguide filter responses with respect to the filter geometrical parameters was carried out in [27]. The neural network structure used for this purpose is the self-organizing map (SOM) proposed by Kohonen [28]. The purpose of SOM is to classify the parameter space into a set of clusters, and simultaneously organize the clusters into a map based upon the relative distances between clusters.

3.8.1 Basic Concept of the Clustering Problem

The basic clustering problem can be stated in the following way. Given a set of samples $\boldsymbol{x}_k$, $k = 1, 2, \ldots, P$, find cluster centers $\boldsymbol{c}_p$, $p = 1, 2, \ldots, N$, that represent the concentration of the distribution of the samples. The symbol $\boldsymbol{x}$—defined as the input vector for the clustering algorithm—may contain contents different than those used for inputs to MLP or RBF. More specifically, $\boldsymbol{x}$ here represents the parameter space that we want classify into clusters. For example, if we want to classify various patterns of frequency response (e.g., S-parameters) of a device/circuit, then $\boldsymbol{x}$ contains S-parameters at a set of frequency points. Having said this, let us call the given samples of $\boldsymbol{x}$ the training data to the clustering algorithm. The clustering algorithm will classify (decompose) the overall training data into clusters—that is, training data similar to one another will be grouped into one cluster and data that are quite different from one another will be separated into different clusters. Let R_i be the index set for cluster center $\boldsymbol{c}_i$ that contains the samples close to $\boldsymbol{c}_i$. A basic clustering algorithm is as follows:

Step 1: Assume total number of clusters to be N.

$$\text{Set } \boldsymbol{c}_i = \text{initial guess}, \quad i = 1, 2, \ldots, N.$$

Step 2: For each cluster, for example the ith cluster, build an index set R_i such that

$$R_i = \{k \mid \|\boldsymbol{x}_k - \boldsymbol{c}_i\| < \|\boldsymbol{x}_k - \boldsymbol{c}_j\|, \quad j = 1, 2, \ldots, N, i \neq j, k = 1, 2, \ldots, P\}$$

$$\text{Update center } \boldsymbol{c}_i \text{ such that } \boldsymbol{c}_i = \frac{1}{size(R_i)} \sum_{k \in R_i} \boldsymbol{x}_k \tag{3.50}$$

Step 3: If the changes in all $\boldsymbol{c}_i$'s are small, then stop. Otherwise go to Step 2.

After finishing this process, the solution (i.e., resulting cluster centers) can be used in the following way. Given an input vector $\boldsymbol{x}$, find the cluster to which $\boldsymbol{x}$ belongs. This is done by simply comparing the distance of $\boldsymbol{x}$ from all the different cluster centers. The closest cluster center wins. This type of input-output relationship is shown in Figure 3.22.

An Example of Clustering Filter Responses

The idea of combining neural networks with SOM clustering for microwave applications was proposed in [27]. An E-plane waveguide filter with a metal insert was modeled using the combined approach. A fast model for the filter that represents the relationship between frequency domain response ($|S_{11}|$) and the input parameters (septa lengths and spacing, waveguide width, frequency) is needed for statistical design taking into account random variations and tolerances in the filter geometry [29].

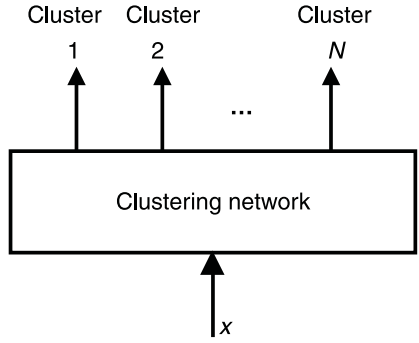


Figure 3.22 A clustering network where each output represents a cluster. The network divides the $\boldsymbol{x}$-space into N clusters. When $\boldsymbol{x}$ is closest to the ith cluster, the ith output will be 1 and all other outputs will be 0's.

The training data was collected for 65 filters (each with a different geometry), at 300 frequency points per filter. However, the variation of $|S_{11}|$ with respect to filter geometry and frequency is highly nonlinear and exhibits sharp variations. In [27], the filter responses are divided into four groups (clusters), in order to simplify the modeling problem and improve neural model accuracy. Figure 3.23 illustrates the four types of filter responses extracted from the 65 filter responses using a neural-network–based automatic clustering algorithm.

In order to perform such clustering, the frequency responses of the filters are used as inputs to the clustering algorithm. Let $\boldsymbol{Q}$ be a feature vector of a filter containing output $|S_{11}|$ at 300 frequency points. For 65 filters, there are 65 corresponding feature vectors: $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \boldsymbol{Q}_3, \ldots, \boldsymbol{Q}_{65}$. Using the feature vectors as training data (i.e., as $\boldsymbol{x}$), employ the clustering algorithm to find

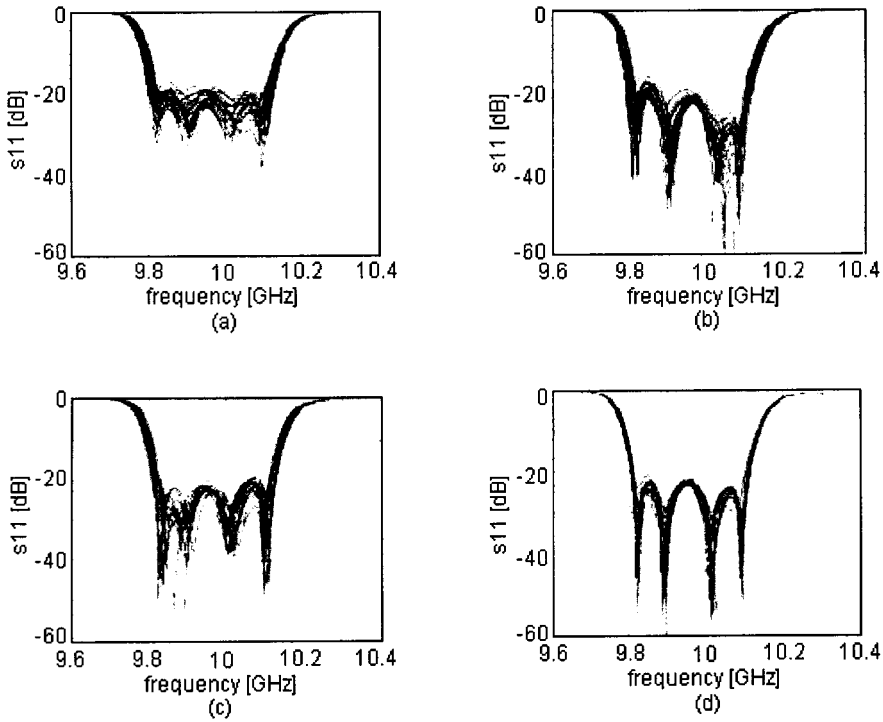


Figure 3.23 Four clusters of filter responses extracted using automatic clustering (from [27], Burrascano, P., S. Fiori, and M. Mongiardo, "A Review of Artificial Neural Network Applications in Microwave CAD," *Int. J. RF and Microwave CAE,* pp. 158–174, © 1999, John Wiley and Sons. Reprinted with permission from John Wiley and Sons, Inc.).

several clusters that correspond to several groups of filter responses. In this example, the filters were divided into 4 groups. The responses of the filters within each group are more similar to one another than those in other groups.

3.8.2 k-Means Algorithm

The k-means algorithm is a popular clustering method. In this algorithm, we start by viewing all the training samples as one cluster. The number of clusters are automatically increased in the following manner:

Step 1: Let $N = 1$ (one cluster)

$$\boldsymbol{c}_1 = \frac{1}{P}\sum_{k=1}^{P} \boldsymbol{x}_k$$

Step 2: Split cluster $\boldsymbol{c}_i$, $i = 1, 2, \ldots, N$, into two new clusters,

$$\boldsymbol{c}_{i1} = \lambda \boldsymbol{c}_i$$
$$\boldsymbol{c}_{i2} = (1 - \lambda)\boldsymbol{c}_i,\ 0 < \lambda < 1$$

Use them as initial values and run the basic clustering algorithm described earlier to obtain refined values of all the centers.

Step 3: If the solution is satisfactory, stop.

Otherwise go to Step 2.

This algorithm uses a basic binary splitting scheme. An improved k-means algorithm was proposed in [30], where only those clusters with large variance with respect to their member samples are split. In this way, the total number of clusters does not have to be 2^k. The variance of the ith cluster is defined as

$$\nu_i = \sum_{k \in R_i} (\boldsymbol{x}_k - \boldsymbol{c}_i)^T (\boldsymbol{x}_k - \boldsymbol{c}_i) \tag{3.51}$$

3.8.3 Self-Organizing Map (SOM)

A self-organizing map (SOM) is a special type of neural network for clustering purposes, proposed by Kohonen [28]. The total number of cluster centers to be determined from training data is equal to the total number of neurons in SOM. As such, each neuron represents a cluster center in the sample distribu-

tion. The neurons of the SOM are arranged into arrays, such as a one-dimensional array or two-dimensional array. A two-dimensional array—also known as a map—is the most common arrangement. The clusters or neurons in the map have a double index (e.g., $\boldsymbol{c}_{ij}$). There are two spaces in SOM: namely, the original problem space (i.e., $\boldsymbol{x}$-space), and the map-space (i.e., (i, j)-space). Figure 3.24 shows a 5×5 map-space.

SOM has a very important feature called topology ordering, which is defined in the following manner. Let $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be two points in $\boldsymbol{x}$-space that belong to clusters $\boldsymbol{c}_{ij}$ and $\boldsymbol{c}_{pq}$, respectively. If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are close to (or far away from) each other in $\boldsymbol{x}$-space, then the corresponding cluster centers $\boldsymbol{c}_{ij}$ and $\boldsymbol{c}_{pq}$ are also close to (or far away from) each other in (i, j)-space. In other words, if $\|\boldsymbol{c}_{ij} - \boldsymbol{c}_{pq}\|$ is small (or large), then $|i - p| + |j - q|$ is also small (or large).

3.8.4 SOM Training

The self-organizing capability of SOM comes from its special training routine. The SOM training algorithm is quite similar to the clustering algorithms except that an additional piece of information called the topological closeness is learned though a neighborhood update. First, for each training sample $\boldsymbol{x}_k$, $k = 1, 2, \ldots, P$, we find the nearest cluster $\boldsymbol{c}_{ij}$ such that

$$\|\boldsymbol{c}_{ij} - \boldsymbol{x}_k\| < \|\boldsymbol{c}_{pq} - \boldsymbol{x}_k\| \ \forall p, q, \ i \neq p, j \neq q \tag{3.52}$$

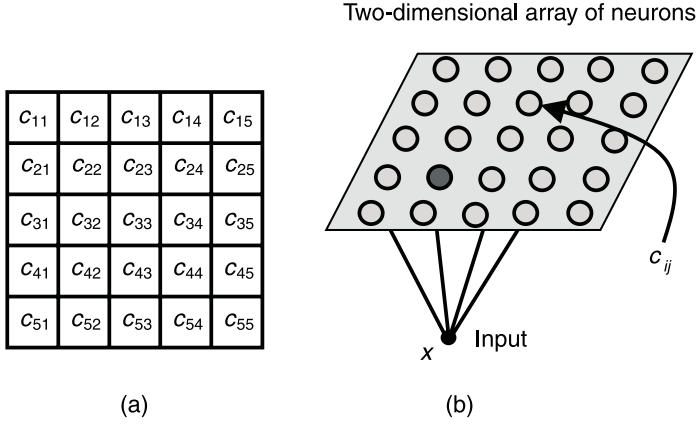


Figure 3.24 Illustration of a 5×5 map where each entry represents a cluster in the $\boldsymbol{x}$-space.

In the next step, the center $\boldsymbol{c}_{ij}$ and its neighboring centers $\boldsymbol{c}_{pq}$ are all updated as

$$\boldsymbol{c}_{pq} = \boldsymbol{c}_{pq} + \alpha(t)(\boldsymbol{x}_k - \boldsymbol{c}_{pq}) \tag{3.53}$$

The neighborhood of $\boldsymbol{c}_{ij}$ is defined as $|p - i| < N_c$ and $|q - j| < N_c$, where N_c is the size of the neighborhood, and $\alpha(t)$ is a positive constant similar to a learning rate that decays as training proceeds.

With such neighborhood updates, each neuron will be influenced by its neighborhood. The neighborhood size N_c is initialized to be a large value and gradually shrinks during the training process. The shrinking process needs to be slow and carefully controlled in order to find out the topological information of the sample distribution.

3.8.5 Using a Trained SOM

Suppose a SOM has been trained—that is, values of $\boldsymbol{c}_{ij}$ are determined. A trained SOM can be used in the following way. Given any $\boldsymbol{x}$, find a map entry (i, j) such that $\|\boldsymbol{x} - \boldsymbol{c}_{ij}\| < \|\boldsymbol{x} - \boldsymbol{c}_{pq}\|$ for all p and q.

Waveguide Filter Example

Continue with the waveguide filter example [27] described in Section 3.8.1. The overall filter model consists of a general MLP for all filters, a SOM that classifies the filter responses into four clusters, and four specific MLP networks representing each of the four groups of filters, as shown in Figure 3.25. The feature vectors of the 65 filters (that is $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, $\boldsymbol{Q_3}$, . . . , $\boldsymbol{Q_{65}}$) are used as training vectors for SOM, and four clusters of filter responses were identified. The general MLP and the four specific MLP networks have the same definition of inputs and outputs. The inputs to the MLPs (denoted as $\boldsymbol{x}$ in Figure 3.25) include frequency as well as filter geometrical parameters such as septa lengths and spacing, and waveguide width. The output of the MLPs (denoted as $\mathbf{y}$) is $|S_{11}|$. The general MLP was trained with all 19,500 training samples (S-parameter at 300 frequency points per-filter geometry for 65 filter geometries). Each of the four specific MLP networks was trained with a subset of the 19,500 samples, according to the SOM classification. These specific MLP networks can be trained to provide more accurate predictions of the S-parameter responses of the filters in the corresponding cluster. This is because within the cluster, the responses of different filters are similar to each other and the modeling problem is relatively easier than otherwise. After training is finished, the overall model can be used in this way. Given a new filter geometry (not

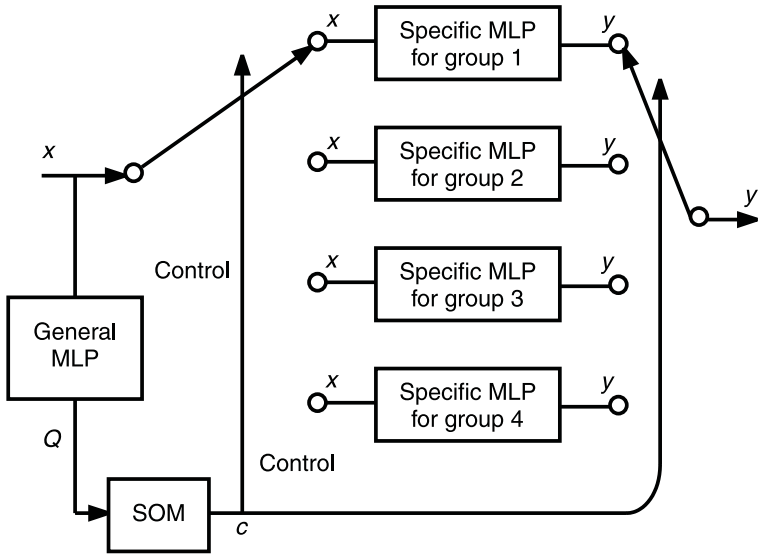


Figure 3.25 The overall neural network model using SOM to cluster the filter responses (after [27]).

seen during training), the general MLP first provides an estimation of the frequency response, which is fed to the SOM. The SOM identifies which cluster or group the filter belongs to. Finally, a specific MLP is selected to provide an accurate filter response. This overall process is illustrated in Figure 3.25.

3.9 Recurrent Neural Networks

In this section, we describe a new type of neural network structure that allows time-domain behaviors of a dynamic system to be modeled. The outputs of a dynamic system depend not only on the present inputs, but also on the history of the system states and inputs. A recurrent neural network structure is needed to model such behaviors [31–34].

A recurrent neural network structure with feedback of delayed neural network outputs is shown in Figure 3.26. To represent time-domain behavior, the time parameter t is introduced such that the inputs and the outputs of the neural network are functions of time as well. The history of the neural network outputs is represented by $\boldsymbol{y}(t-\tau)$, $\boldsymbol{y}(t-2\tau)$, $\boldsymbol{y}(t-3\tau)$, and so forth, and the history of the external inputs is represented by $\boldsymbol{x}(t-\tau)$, $\boldsymbol{x}(t-2\tau)$,

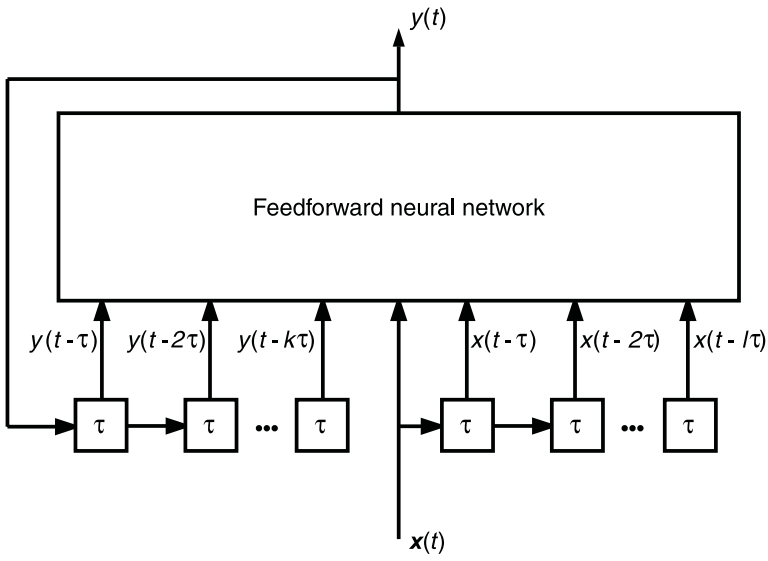


Figure 3.26 A recurrent neural network with feedback of delayed neural network output.

$\boldsymbol{x}(t - 3\tau)$, and so forth. The dynamic system can be represented by the time-domain equation,

$$\boldsymbol{y}(t) = \boldsymbol{f}(\boldsymbol{y}(t - \tau), \boldsymbol{y}(t - 2\tau), \ldots, \boldsymbol{y}(t - k\tau), \boldsymbol{x}(t), \boldsymbol{x}(t - \tau), \ldots, \boldsymbol{x}(t - l\tau)) \tag{3.54}$$

where k and l are the maximum number of delay steps for $\boldsymbol{y}$ and $\boldsymbol{x}$, respectively.

The combined history of the inputs and outputs of the system forms an intermediate vector of inputs to be presented to the neural network module that could be any of the standard feedforward neural network structures (e.g., MLP, RBF, wavelet networks). An example of a three-layer MLP module for a recurrent neural network with two delays for both inputs and outputs is shown in Figure 3.27. The feedforward network together with the delay and feedback mechanisms results in a recurrent neural network structure. The recurrent neural network structure suits such time-domain modeling tasks as dynamic system control [31–34], and FDTD solutions in EM modeling [35].

A special type of recurrent neural network structure is the Hopfield network [36, 37]. The overall structure of the network starts from a single layer (as described in arbitrary structures). Let the total number of neurons be N. Each neuron, say the ith neuron, can accept stimulus from an external input x_i, outputs from other neurons, y_j, $j = 1, 2, \ldots, N, j \neq i$, and output of the neuron itself, that is, y_i. The output of each neuron is an external output

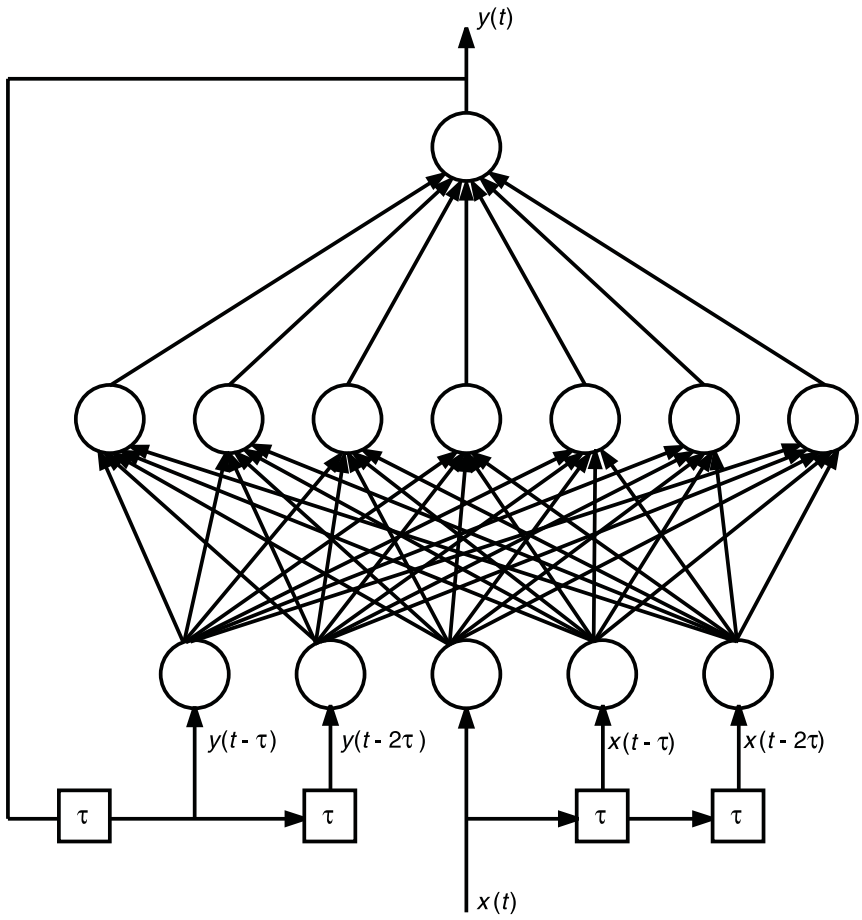


Figure 3.27 A recurrent neural network using MLP module and two delays for output feedback.

of the neural network. The input to the activation function in the ith neuron is given by

$$\gamma_i(t) = \sum_{j=1}^{N} w_{ij} y_j(t) + x_i(t) \tag{3.55}$$

and the output of the ith neuron at time t is given by

$$y_i(t) = \sigma(\gamma_i(t-1)) \tag{3.56}$$

for the discrete case, and

$$y_i(t) = \sigma(\lambda u_i) \tag{3.57}$$

for the continuous case. Here, λ is a constant and u_i is the state variable of the ith neuron governed by the simple internal dynamics of the neuron. The dynamics of the Hopfield network are governed by an energy function (i.e., Lyapunov function), such that as the neural network dynamically changes with respect to time t, the energy function will be minimized.

3.10 Summary

This chapter introduced a variety of neural network structures potentially important for RF and microwave applications. For the first time, neural network problems are described using a terminology and approach more oriented toward the RF and microwave engineer's perspective. The neural network structures covered in this chapter include MLP, RBF, wavelet networks, arbitrary structures, SOM, and recurrent networks. Each of these structures has found its way into RF and microwave applications, as reported recently by the microwave researchers. With the continuing activities in the microwave-oriented neural network area, we expect increased exploitation of various neural network structures for improving model accuracy and reducing model development cost.

In order to choose a neural network structure for a given application, one must first identify whether the problem involves modeling a continuous parametric relationship of device/circuits, or whether it involves classification/decomposition of the device/circuit behavior. In the latter case, SOM or clustering algorithms may be used. In the case of continuous behaviors, time-domain dynamic responses such as FDTD solutions require a recurrent neural model that is currently being researched. Nondynamic modeling problems, on the other hand, (or problems converted from dynamic to nondynamic using methods like harmonic balance) may be solved by feedforward neural networks such as MLP, RFB, wavelet, and arbitrary structures.

The most popular choice is the MLP, since the structure and its training are well-established, and the model has good generalization capability. RBF and wavelet networks can be used when the problem behavior contains highly nonlinear phenomena or sharp variations. In such cases, the localized nature of the RBF and wavelet neurons makes it easier to train and achieve respectable model accuracy. However, sufficient training data is needed. For CAD tool developers, an arbitrary neural network framework could be very attractive since it represents all types of feedforward neural network structures and facilitates structural optimization during training. One of the most important research directions in the area of microwave-oriented neural network structures is the

combining of RF/microwave empirical information and existing equivalent circuit models with neural networks. This has led to an advanced class of structures known as knowledge-based artificial neural networks, described in Chapter 9.

References

[1] Zhang, Q. J., F. Wang, and V. K. Devabhaktuni, "Neural Network Structures for RF and Microwave Applications," *IEEE AP-S Antennas and Propagation Int. Symp*., Orlando, FL, July 1999, pp. 2576–2579.

[2] Wang, F., et al., "Neural Network Structures and Training Algorithms for RF and Microwave Applications," *Int. Journal of RF and Microwave CAE*, Special Issue on Applications of ANN to RF and Microwave Design, Vol. 9, 1999, pp. 216–240.

[3] Scarselli, F., and A. C. Tsoi, "Universal Approximation using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results," *Neural Networks*, Vol. 11, 1998, pp. 15–37.

[4] Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function," *Math. Control Signals Systems*, Vol. 2, 1989, pp. 303–314.

[5] Hornik, K., M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, 1989, pp. 359–366.

[6] Devabhaktuni, V. K., et al., "Robust Training of Microwave Neural Models," *IEEE MTT-S Int. Microwave Symposium Digest*, Anaheim, CA, June 1999, pp. 145–148.

[7] Kwok, T. Y., and D. Y. Yeung, "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems," *IEEE Trans. Neural Networks*, Vol. 8, 1997, pp. 630–645.

[8] Reed, R., "Pruning Algorithms—A Survey," *IEEE Trans. Neural Networks*, Vol. 4, Sept. 1993, pp. 740–747.

[9] Krzyzak, A., and T. Linder, "Radial Basis Function Networks and Complexity Regularization in Function Learning," *IEEE Trans. Neural Networks*, Vol. 9, 1998, pp. 247–256.

[10] de Villiers, J., and E. Barnard, "Backpropagation Neural Nets with One and Two Hidden Layers," *IEEE Trans. Neural Networks*, Vol. 4, 1992, pp. 136–141.

[11] Tamura, S., and M. Tateishi, "Capabilities of a Four-Layered Feedforward Neural Network: Four Layer Versus Three," *IEEE Trans. Neural Networks*, Vol. 8, 1997, pp. 251–255.

[12] Doering, A., M. Galicki, and H. Witte, "Structure Optimization of Neural Networks with the A-Algorithm," *IEEE Trans. Neural Networks*, Vol. 8, 1997, pp. 1434–1445.

[13] Yao, X., and Y. Liu, "A New Evolution System for Evolving Artificial Neural Networks," *IEEE Trans. Neural Networks*, Vol. 8, 1997, pp. 694–713.

[14] Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, Vol. 1, D.E. Rumelhart and J. L. McClelland, Editors, Cambridge, MA: MIT Press, 1986, pp. 318–362.

[15] Garcia, J. A., et al., "Modeling MESFET's and HEMT's Intermodulation Distortion Behavior using a Generalized Radial Basis Function Network," *Int. Journal of RF and*

Microwave CAE, Special Issue on Applications of ANN to RF and Microwave Design, Vol. 9, 1999, pp. 261–276.

[16] Park, J., and I. W. Sandberg, "Universal Approximation Using Radial-Basis–Function Networks," *Neural Computation*, Vol. 3, 1991, pp. 246–257.

[17] Haykin, S., *Neural Networks: A Comprehensive Foundation*, NY: IEEE Press, 1994.

[18] Moody, J., and C. J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, Vol. 1, 1989, pp. 281–294.

[19] *OSA90 Version 3.0*, Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, now HP EESof (Agilent Technologies), 1400 Fountaingrove Parkway, Santa Rosa, CA 95403.

[20] Zhang, Q. H., "Using Wavelet Network in Nonparametric Estimation," *IEEE Trans. Neural Networks*, Vol. 8, 1997, pp. 227–236.

[21] Zhang, Q. H., and A. Benvensite, "Wavelet Networks," *IEEE Trans. Neural Networks*, Vol. 3, 1992, pp. 889–898.

[22] Pati, Y. C., and P. S. Krishnaprasad, "Analysis and Synthesis of Fastforward Neural Networks using Discrete Affine Wavelet Transformations," *IEEE Trans. Neural Networks,* Vol. 4, 1993, pp. 73–85.

[23] Harkouss, Y., et al., "Modeling Microwave Devices and Circuits for Telecommunications System Design," *Proc. IEEE Intl. Conf. Neural Networks*, Anchorage, Alaska, May 1998, pp. 128–133.

[24] Bila, S., et al., "Accurate Wavelet Neural Network Based Model for Electromagnetic Optimization of Microwaves Circuits," *Int. Journal of RF and Microwave CAE*, Special Issue on Applications of ANN to RF and Microwave Design, Vol. 9, 1999, pp. 297–306.

[25] Harkouss, Y., et al., "Use of Artificial Neural Networks in the Nonlinear Microwave Devices and Circuits Modeling: An Application to Telecommunications System Design," *Int. Journal of RF and Microwave CAE*, Special Issue on Applications of ANN to RF and Microwave Design, Guest Editors: Q. J. Zhang and G. L. Creech, Vol. 9, 1999, pp. 198–215.

[26] Fahlman, S. E., and C. Lebiere, "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

[27] Burrascano, P., S. Fiori, and M. Mongiardo, "A Review of Artificial Neural Networks Applications in Microwave CAD," *Int. Journal of RF and Microwave CAE*, Special Issue on Applications of ANN to RF and Microwave Design, Vol. 9, April 1999, pp. 158–174.

[28] Kohonen, T., "Self Organized Formulation of Topologically Correct Feature Maps," *Biological Cybermatics,* Vol. 43, 1982, pp. 59–69.

[29] Burrascano, P., et al., "A Neural Network Model for CAD and Optimization of Microwave Filters," *IEEE MTT-S Int. Microwave Symposium Digest,* Baltimore, MD, June 1998, pp. 13–16.

[30] Wang, F., and Q. J. Zhang, "An Improved K-Means Clustering Algorithm and Application to Combined Multi-Codebook/MLP Neural Network Speech Recognization," *Proc. Canadian Conf. Electrical and Computer Engineering*, Montreal, Canada, September 1995, pp. 999–1002.

[31] Aweya, J., Q. J. Zhang, and D. Montuno, "A Direct Adaptive Neural Controller for Flow Control in Computer Networks," *IEEE Int. Conf. Neural Networks*, Anchorage, Alaska, May 1998, pp. 140–145.

[32] Aweya, J., Q. J. Zhang, and D. Montuno, "Modelling and Control of Dynamic Queues in Computer Networks using Neural Networks," *IASTED Int. Conf. Intelligent Syst. Control*, Halifax, Canada, June 1998, pp. 144–151.

[33] Aweya, J., Q. J. Zhang, and D. Montuno, "Neural Sensitivity Methods for the Optimization of Queueing Systems," *1998 World MultiConference on Systemics, Cybernetics and Infomatics*, Orlando, Florida, July 1998 (invited), pp. 638–645.

[34] Tsoukalas, L. H., and R. E. Uhrig, *Fuzzy and Neural Approaches in Engineering*, NY: Wiley-Interscience, 1997.

[35] Wu, C., M. Nguyen, and J. Litva, "On Incorporating Finite Impulse Response Neural Network with Finite Difference Time Domain Method for Simulating Electromagnetic Problems," *IEEE AP-S Antennas and Progpagation International Symp*., 1996, pp. 1678–1681.

[36] Hopfield, J., and D. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, 1985, pp. 141-152.

[37] Freeman, J. A., and D. M. Skapura, *Neural Networks: Algorithms, Applications and Programming Techniques*, Reading, MA: Addision-Wesley, 1992.