NOTE TO USERS

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ze Cheng

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Neural-based CAD Tool for RF/microwave Modeling

TITRE DE LA THÈSE / TITLE OF THESIS

M. Yagoub

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

R. Achar

D. McNamara

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

A NEURAL-BASED CAD TOOL FOR RF/MICROWAVE MODELING

Ze CHENG, B. Eng.,

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

Master of Applied Science
Electrical Engineering

August 2005

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa, Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 0-494-11239-5
Our file Notre référence
ISBN: 0-494-11239-5

Abstract

The dramatic development of the commercial markets for wireless communication products leads to an increasing need for accurate and fast models of RF and microwave components and circuits. The traditional modeling approaches have the disadvantage of being either expensive or time-consuming. Although the basic artificial neural network as a fast and accurate modeling approach has been applied in diverse situations, the use of knowledge-aided neural networks is quite new.

In this thesis, we focus on the development of a neural-based computer aided design (CAD) tool for the general Multi-Layer Perceptrons (MLP) neural network, the Knowledge-Based Neural Network (KBNN), and the Prior Knowledge Input (PKI) neural network. KBNN and PKI were used, for the first time in this thesis, in the modeling of a mixer and multistage amplifiers. Since in the RF and microwave field, the training data are usually obtained from measurements or simulations, which are either expensive in data generation or CPU time consuming, such applications of knowledge-aided neural networks (KBNN and PKI) have been proved to be capable of reducing the need for a large number of training data, and improving the accuracy and efficiency as well.

Acknowledgements

Throughout the whole course of my research, I have gathered immense technical research skills and abilities with the backup, support, and patience of several individuals- to whom the least I wish to express is a sincere word of gratitude and recognition.

I would like to thank my supervisor, Dr. Mustapha C.E. Yagoub, for his guidance, encouragement, patience and support. He was always there with his heart and mind to provide whatever is needed to achieve my task.

I would also like to thank the examination committee for sparing the time to review and criticize my manuscript.

Many sincere thanks to the SITE system staff. Special thanks to my friends at the RF and Microwave (RF&MW) group for their selfless help and valuable advices.

Finally I would like to thank my families. Their love is the light of my life for ever.

Thanks to everyone

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADS	Advanced Design System
BJT	Bipolar Junction Transistor
BP	Back Propagation
CAD	Computer Aided Design
EM	Electromagnetic
FET	Field Effect Transistor
GaAs	Gallium Arsenide
GHz	Giga Hertz
IC	Integrated Circuit
IFF	Intermediate Frequency Filter
KBNN	Knowledge-based Neural Network
KHz	Kilo Hertz
LNA	Low Noise Amplifier
MLP	Multilayer Perceptrons
PKI	Prior Knowledge Input
RF	Radio Frequency
Si	Silicon
VCO	Voltage Controlled Oscillator

Chapter 1

Introduction

1.1 Motivations

The effective use of Computer Aided Design (CAD) tools both in electrical and physical design stages is very important in RF and microwave circuit and system design for shrinking design margins and complexity. Furthermore, in the circuit design, the designer should take into consideration such repetitive processes as statistical analysis, yield optimization, which involve manufacturing tolerance, model uncertainties, variation of the process variables and so on [1] - [6]. Consequently, the drive for manufacturability-oriented design and reduced time-to-market in the industry needs accurate and fast models which could be used in the computer simulation rather than hardware prototyping [7]. Thus, fast and accurate modeling is a major issue, yet it is still a bottleneck for CAD in certain class of RF and microwave circuit.

In general, there are two kinds of conventional approaches in the microwave modeling. The first type consists of EM-based models for passive components and physical-based models for active components. The overall models of this type are defined by well-

established theory, rather than empirical data. Although accurate, this kind of models is really computationally intensive. The other kind of conventional modeling is empirical or equivalent circuit-based models for both passive and active components, which is developed by using a mixture of simplified component theory, heuristic interpretation and representations and fitting of experimental data. The speed for this kind of modeling is fast, but the accuracy is less than EM-based or physical-based models. Furthermore, the data extraction of the equivalent circuits is quite a long and complex process. Therefore, trying to find an approach that can efficiently develop fast and accurate models for some RF and microwave component and circuit is the basic motivation of this thesis.

Artificial neural networks are information processing systems motivated by the ability of human brains which can learn from observation and can generalize from abstraction. It represents a technology rooted in many disciplines such as mathematics, statistics, computer science and engineering. Accordingly, neural networks can find applications in various fields. By virtue of their ability to learn from input data, which represents the environment of interest, it will play an important role in the twenty first century, particularly when we are confronted with difficult problems that are characterized by nonlinearity, non-stationarity, and unknown statistics [8].

The modeling with neural networks is based on the experimental data. Through the process of learning, the neural network could learn the relationship between the inputs and outputs. Usually depending on the number of the training data and the scale of the neural network, the time of learning ranges from seconds to hours. Once the model is

developed, we can get the accurate result of any input within the range of training data in seconds which is much less than the classic simulations. Meanwhile the accuracy is better than the empirical models. Several publications have shown the efficiency and accuracy of the neural networks [1] [3] [5]. However, in the RF and microwave field, many components and circuits will perform highly nonlinearly in higher frequency and power level. The basic neural network structure may not achieve the desired accuracy, in such cases some knowledge-aided neural networks such as knowledge-based neural networks (KBNN) and prior knowledge input (PKI) neural networks [4] can fill the gap. Using the knowledge-aided neural networks (where the basic neural networks do not work well) to improve the accuracy and efficiency of the modeling process is another motivation of this thesis.

1.2 Thesis Objective

The main objective of this thesis is to develop a neural network CAD tool for the basic multilayer perceptrons, knowledge-based neural networks and prior knowledge input neural networks, so that it can be used to efficiently model RF and microwave components and circuits. Based on the neural networks we have constructed, different RF and microwave components and circuits were modeled to show the advantages of different kinds of neural networks.

1.3 Thesis Outline

This thesis is organized to develop a neural network tool and then to apply it in RF and microwave component- and circuit- modeling. It is composed of 6 chapters.

In chapter 2, a detailed review of different neural network structures is presented. Chapter 3 presents a systematic description of the procedures of developing neural network models. The key issues of developing neural network models are discussed in details. The main goal of these two chapters is to provide the theoretical foundation for the development of the neural network CAD tool. In chapter 4, the development of the MLP KBNN and PKI is depicted.

Some RF and microwave component- and circuit-modeling examples using the neural network tool we have mentioned in chapter 4 are presented in chapter 5. Through these examples the advantages of the knowledge-aided neural networks are shown very clearly.

Finally, a conclusion is drawn in chapter 6, followed with suggestions for future work.

1.4 Thesis Contribution

Two primary contributions of the RF and microwave modeling are presented in this thesis:

1. The development of the neural-based CAD tool for RF and microwave component/circuit modeling. With it being more specified in RF and microwave field

and with the possession of the friendly user interface, the tool can be more easily used by RF and microwave circuit designers.

2. The applications of KBNN and PKI to the component/circuit modeling. Based on the empirical information, KBNN and PKI were used to model the performance of the mixer and multistage amplifiers for the first time. It has been proved that both KBNN and PKI can improve the accuracy and efficiency of the neural network models.

The above work resulted in the following publications:

1. S. Gaoua, L. Ji, Z. Cheng, F.A. Mohammadi, M.C.E. Yagoub, "From component to circuit: advanced CAD tools for efficient RF/microwave integrated communication system design," *WSEAS Trans. on Communications*, Vol. 4, No 10, pp. 1028-1039, Oct. 2005.

2. Z. Cheng, L. Ji, S. Gaoua, F.A. Mohammadi and M.C.E. Yagoub, "Robust framework for efficient RF/microwave system modeling using neural- and fuzzy-based CAD tools," *4th Int. Conf. on Electronics, Signal Processing and Control* (ESPOCO 2005), Rio de Janeiro, Brazil, April 25-27, 2005.

Chapter 2

Neural Network Structures

Although neural network technique has been used for a long time, the introduction of neural network to RF and microwave field is only done in recent years. It offers a new way to solve many modeling and design problems in this field. Nowadays, its efficiency and accuracy are drawing more and more attention from the designers around the world. Neural network structure is one of the most important factors in developing neural network models. A variety of neural network structures have been developed in the neural network community that are useful for RF and microwave applications. In this chapter, we will briefly review some existing neural network structures which include multilayer perceptrons (MLP), knowledge-based neural networks (KBNN), and prior knowledge input neural networks (PKI).

2.1 Introduction to Neural Network

A neural network is composed of two basic components, namely, neurons and synapses or connecting links. Neurons are the places where the information is processed. Each neuron receives stimuli from the neighbor neurons connected to it, processes the information and then produces an output. Those neurons receiving stimuli directly from

outside of the neural network are called input neurons; those receiving stimuli from other neurons inside the neural network are called hidden neurons; and those neurons whose outputs are used outside the neuron networks are output neurons. For each of synapses there is always a weight parameter associated with it [8].

There are different ways in which a neuron processes the information, and in which the neurons connect. Different neural network structures can be constructed by defining how the neurons process the information and how the neurons are connected with each other. The different way a neuron processes the information is represented by the different activation functions, which give the output signal of a neuron according to the weighted inputs.

2.2 Multilayer Perceptrons (MLP)

Multilayer perceptrons (MLP) is the most popular type of feed forward neural networks used today. Typically, this kind of networks consists of a set of neuron layers, namely, one input layer, one or more hidden layers, and one output layer, as shown in Figure 2-1 [4]. The neurons in the hidden layers and the output layer act as computational neurons. The input signals propagate throughout the network in forward direction layer by layer from the input layer to the output layer. The neuron network could approximate generic classes of functions including continuous and integral ones [9].

2.2.1 MLP Structure

Suppose the total number of layers is L. The 1^{st} layer is the input layer and the L^{th} layer is the output layer. The 2^{nd} layer to the $(L-1)^{th}$ layer are hidden layers. The number of neurons in the l^{th} layer is N_l, $l = 1, 2, \cdots, L$. Suppose the number of neurons in the input layer and output layer is n and m respectively, where $n = N_1$ and $m = N_L$.

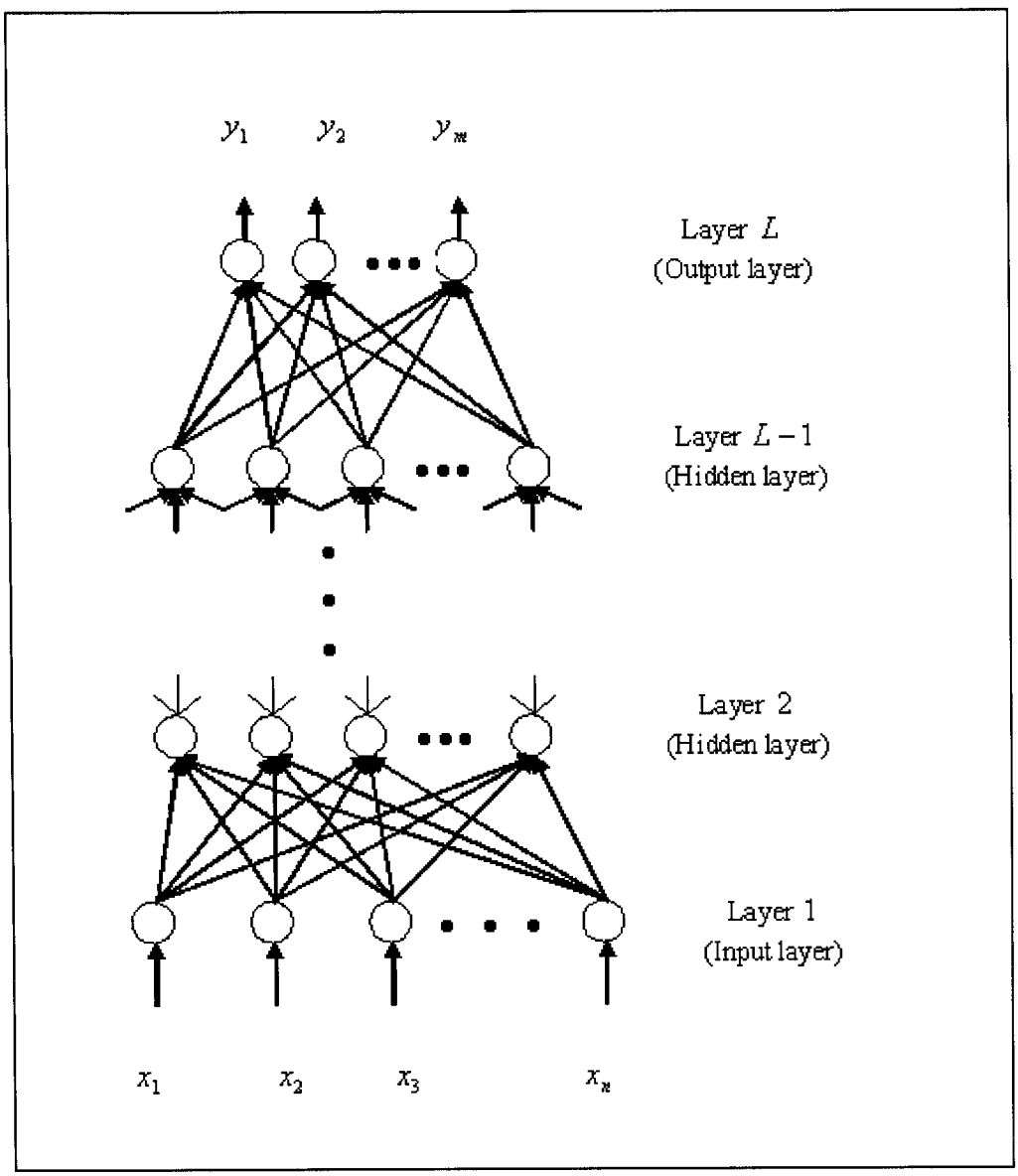


Figure 2-1. Structure of MLP neural network

Let w_{ji}^{l} represent the weight of the synapse between i^{th} neuron of $(l-1)^{th}$ layer and j^{th} neuron of l^{th} layer, where $1 \le i \le N_{l-1}, 1 \le j \le N_l$. Let x_i represent the i^{th} input parameter and y_j be the j^{th} output of the MLP. An extra weight parameter w_{j0}^{l} is introduced for each neuron to represent its bias. As such, the weight vector $\boldsymbol{w}$ includes w_{ji}^{l}, $i = 0, 1, \cdots, N_{l-1}$, $j = 1, 2, \cdots, N_l$, $l = 2, 3, \cdots, L$, that is,

$$\boldsymbol{w} = [w_{10}^{2}, w_{11}^{2}, \cdots, w_{N_L N_{L-1}}^{L}]^T \tag{2.1}$$

Let z_j^l, $j = 1, 2, \cdots, N_l$, $l = 1, 2, \cdots, L$ be the output of j^{th} neuron of l^{th} layer. Therefore, according to w_{j0}^{l}, $j = 1, 2, \cdots, N_l$, $l = 2, 3, \cdots, L$, we have $z_0^{l-1} = 1$, for $l = 2, 3, \cdots, L$.

2.2.2 Activation Function

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

$$\sigma(\gamma) = \frac{1}{(1 + e^{\gamma})} \tag{2.2}$$

As shown in figure 2-2, the sigmoid function is a smooth switch function having the property of,

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

where γ is defined as:

$$\gamma_j^l = \sum_{i=0}^{N_{l-1}} w_{ji}^l z_i^{l-1} \quad j = 1, 2, \cdots, N_l; \; l = 2, 3, \cdots, (L-1) \tag{2.3}$$

There are also some other kinds of activation functions for hidden neurons, such as the arc-tangent function and the hyperbolic-tangent function. The activation function for output neurons could be either logic or simple linear function. Generally we choose the simple linear activation function, which is the weighted sum of the outputs of the previous layer, for output neurons. One of the advantages of linear function in this case is that it can improve the numerical conditioning of the neural network training process. The linear activation function for the output layer neurons is defined as:

$$\sigma(\gamma_j) = \gamma_j = \sum_{i=0}^{N_{L-1}} w_{ji}^L z_i^{L-1} \quad j = 1, 2, \cdots, N_L \tag{2.4}$$

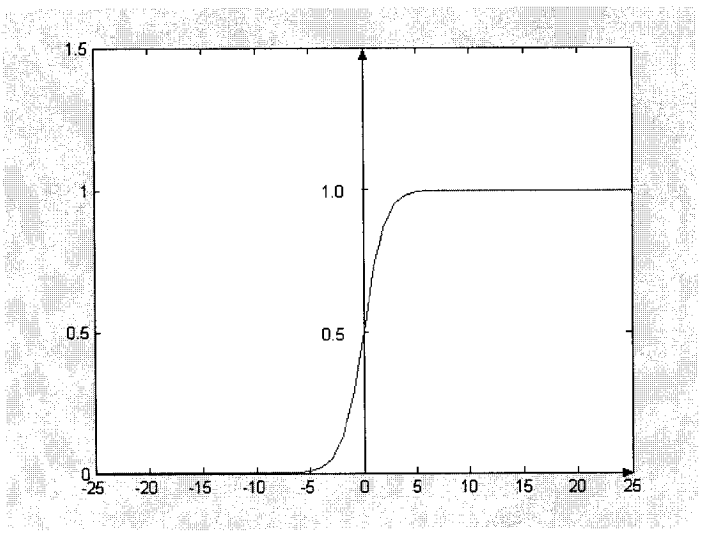


Figure 2-2. Sigmoid function

2.2.3 Neural Network Feed Forward

Given the input vector $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^T$ and the weight parameter vector $\boldsymbol{w}$, neural network feed forward process is to calculate the output vector $\boldsymbol{y} = [y_1, y_2, \cdots, y_m]^T$ from a MLP neural network. During the feed forward process, the external inputs are first fed to the input neurons (1^{st} layer), then the outputs from the input layer are fed to the hidden neurons of the 2^{nd} layer, and so on, and finally the outputs of the $(L-1)^{th}$ layer are fed to the output neurons (L^{th} layer). The computation is given by:

$$z_j^1 = x_j \quad j = 1, 2, \cdots, n; \quad n = N_1 \tag{2.5}$$

$$z_j^l = \sigma(\sum_{i=0}^{N_{l-1}} w_{ji}^l z_i^{l-1}) \quad j = 1, 2, \cdots, N_l; \quad l = 2, 3, \cdots, L-1 \tag{2.6}$$

$$z_j^L = \sum_{i=0}^{N_{L-1}} w_{ji}^L z_i^{L-1} \quad j = 1, 2, \cdots, m; \quad m = N_L \tag{2.7}$$

2.2.4 Universal Approximation Theory

In 1989, both Cybenko [10] and Hornik [11] proved the universal approximation theorem for MLP. Let $\boldsymbol{I}_n$ be an n-dimensional unit cube containing all possible samples $\boldsymbol{x}$, that is, $x_i \in [0,1], i = 1, 2, \cdots, n,$ and $C(I_n)$ be the space of continuous function on $\boldsymbol{I}_n$. If $\sigma(\cdot)$ is a continuous sigmoid function, the universal approximation theorem states that the finite sums in the form of:

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{N_2} w_{kj}^3 \sigma(\sum_{i=0}^{n} w_{ji}^2 x_i) \quad k = 1, 2, \cdots, m \tag{2.8}$$

are dense in $C(\boldsymbol{I}_n)$ space. In other words, given any $f \in C(\boldsymbol{I}_n)$ and $\varepsilon > 0$, there is a sum $y(\boldsymbol{x}, \boldsymbol{w})$ of the above form that satisfies $|y(\boldsymbol{x}, \boldsymbol{w}) - f(\boldsymbol{x})| < \varepsilon$ *for all* $\boldsymbol{x} \in \boldsymbol{I}_n$. That means for MLP there always exits a 3-layer perceptrons (MLP3) that can theoretically approximate an arbitrary nonlinear, continuous, multi-dimensional function f with any desired accuracy.

2.2.5 Number of Hidden Layers and Number of Hidden Neurons

Although the universal approximation theorem tells us that a 3-layer MLP is enough to model any problem, it does not tell us how many hidden neurons and input vector samples are needed to achieve a given accuracy. As such, the reason for failing to develop an accurate 3-layer MLP neural model can be insufficient number of hidden neurons, fewer training data or inadequate training and so on. In practice, the number of hidden neurons depends on the nonlinearity of the original problem and the dimension of the input space. Therefore, too many hidden neurons for a 3-layer MLP may be not a good choice. We could use a structure of more hidden layers, but fewer neurons for each layer as an alternative. For RF and microwave applications, 3-layer and 4-layer MLP are the most used structures.

Generalization ability and mapping ability are two specifications to evaluate the performance of a neural model. Generalization or test ability is the ability of a neural model to estimate output $\boldsymbol{y}$ accurately when presented with input $\boldsymbol{x}$ never seen during training [4]. While mapping ability is the ability to estimate $\boldsymbol{y}$ accurately for given training sample input $\boldsymbol{x}$. According to [12], when the generalization capability is a major

concern, 3-layer MLP is preferred, otherwise when the mapping ability is more important 4-layer MLP could be better.

2.3 Knowledge-Based Neural Network (KBNN)

MLP usually needs a large number of data to learn the problem behavior and then to achieve the desired accuracy. However, the most widely used approaches to getting data in RF and microwave field are measurements and EM/physical theoretical equations. Unfortunately, both of them are either expensive in data generation or CPU-time-consuming. On the other hand, some of the physical equations are only valid for a certain range of the input space. All these factors above give rise to a novel neural network structure, that is, the knowledge-based neural network (KBNN). In this kind of network, some empirical information from EM/physical equations is added to help the training process in order that the number of the data needed to achieve certain accuracy can be reduced. This kind of neural network not only inherits accuracy from EM/physical models, but also keeps the speed of a neural network model.

The basic idea of KBNN is shown in Figure 2-3 [4]. The empirical information is embedded into the network structure. The detailed structure is shown in Figure 2-4 [7].

There are 6 layers in this structure, which are input layer x, knowledge layer z, boundary layer b, region layer r, normalized region layer r' and the output layer y. The input layer like that in MLP accepts the external signals. The knowledge layer is where the empirical information exists in the form of single or multidimensional functions $\Psi(\cdot)$.

Output of the knowledge neuron j in this layer is expressed by:

$$z_j = \Psi_j(\boldsymbol{x}, \boldsymbol{w}_j), \quad j = 1, 2, \cdots, N_z \tag{2.9}$$

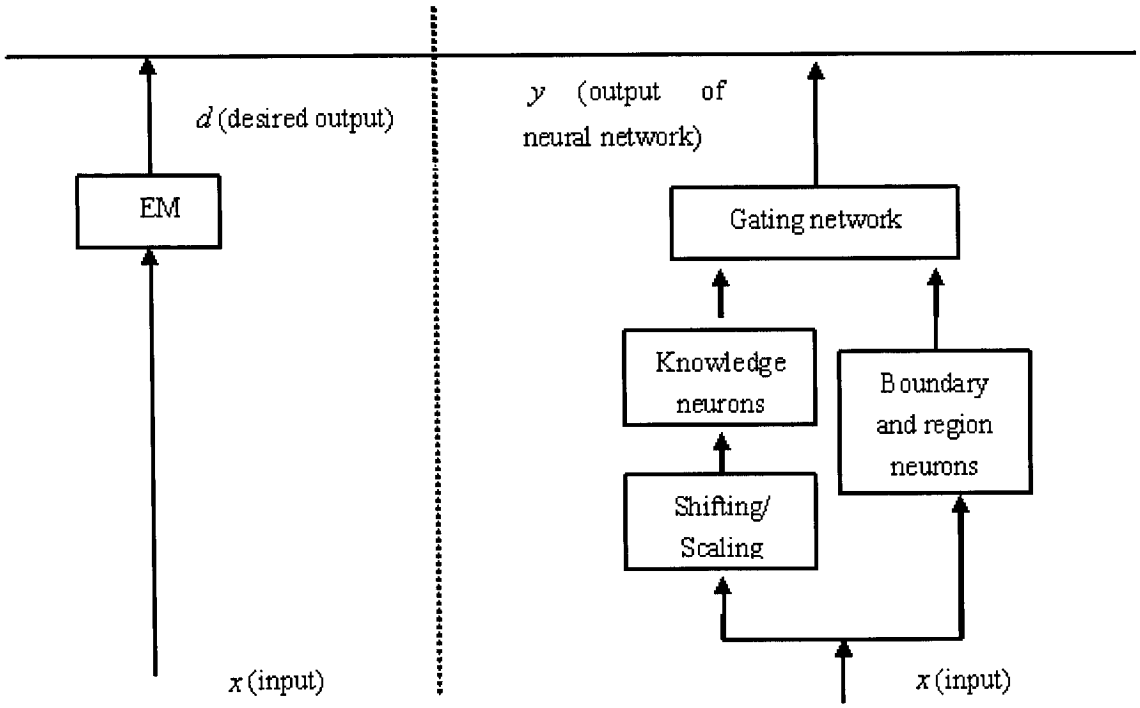


Figure 2-3. Basic idea behind knowledge-based neural network model development

where $\boldsymbol{x}$ is the input vector including x_i $(i = 1, 2, \cdots, n)$, N_z is the number of the knowledge neurons, and $\boldsymbol{w}_j$ is a vector of parameters in the knowledge formula. The function $\Psi_j(\boldsymbol{x}, \boldsymbol{w}_j)$ is always in the form of empirical or semi-analytical function. The boundary layer b can either incorporate the empirical information in the form of problem-dependent function, or just in problem-independent form of linear combination of the inputs. The output of the neuron j of this layer is:

$$b_j = b_j(\boldsymbol{x}, \boldsymbol{v}_j), \quad j = 1, 2, \cdots, N_b \tag{2.10}$$

where v_j is the parameter vector and N_b is the number of boundary neurons. For the region layer r, the output of the neuron j can be represented by:

$$r_j = \prod_{i=1}^{N_b} \sigma(\alpha_{ji} b_i + \theta_{ji}), \quad j = 1, 2, \cdots, N_r \tag{2.11}$$

where α_{ji} and θ_{ji} are the scaling and bias parameters respectively and N_r is the number of region neurons. The normalized region layer r' represents the normalized value of the output of the region layer,

$$r_j' = \frac{r_j}{(\sum_{i=1}^{N_r} r_i)}, \quad j = 1, 2, \cdots, N_r' \quad where\ N_r = N_r' \tag{2.12}$$

The overall output of the network y is composed of the output of the knowledge neurons and that of the normalized region layer neurons.

$$y_j = \sum_{i=1}^{N_z} \beta_{ji} z_i \left(\sum_{k=1}^{N_r'} \rho_{jik} r_k' \right) + \beta_{j0}, \quad j = 1, 2, \cdots, m \tag{2.13}$$

where β_{ji} represents the contribution of the knowledge neuron i to the output neuron j, and β_{j0} is the corresponding bias parameter. The whole normalized region neurons are

shared by all of the outputs. Usually the number of the neurons in region layer and normalized region layer is the same as the number of the knowledge layer neurons.

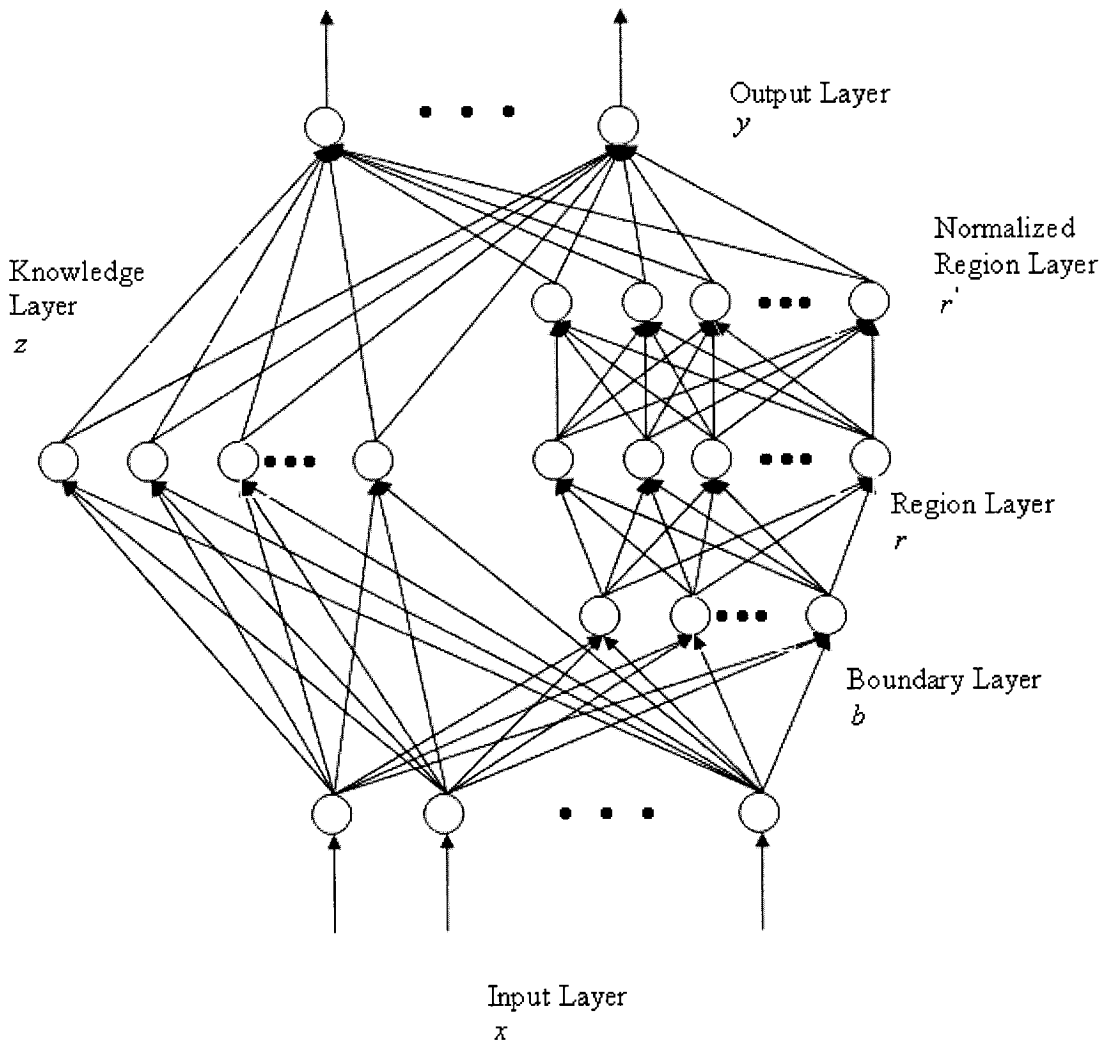


Figure 2-4. Structure of KBNN

Compared to MLP, KBNN includes empirical information in the neural network, which can help to speed up the training process and improve the accuracy for the same number of training data.

2.4 Prior Knowledge Input Neural Networks (PKI)

Neural networks of another kind which include the empirical information are prior knowledge input neural networks (PKI) [13]. In such networks, the outputs of some empirical models are used as part of the inputs in addition to the original inputs of the problem concerned. The mapping is between the original inputs plus the empirical model outputs and the original outputs. The general model structure is shown in figure 2-5 [4].

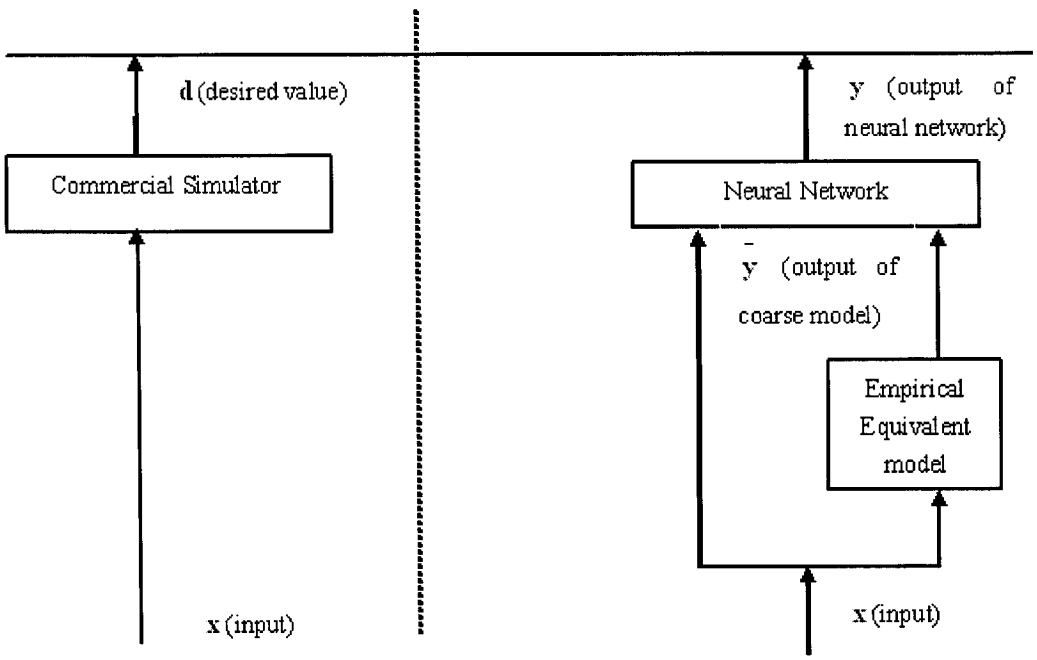


Figure 2-5. Structure of PKI

Since part of the mapping is between the empirical model outputs and the actual outputs, the former being close to the latter, this one to one mapping can remarkably speed up the training process. At the same time, to achieve the same accuracy, it needs fewer training

data. General neural network structures such as MLP can be used to learn the relationships here.

2.5 Comparison of Different Neural Network Structures

MLP is most commonly used for its simplicity and generality. However, without the empirical information of the specific problem being considered, it always needs a large number of training samples to get certain accuracy. Therefore, when training samples are very expensive, some other neural networks such as KBNN and PKI are preferable. With empirical information, both of them can enhance neural model accuracy and generalization ability, and can reduce the need for a large number of training samples. Especially, when the given input is beyond training range, the empirical information could help KBNN and PKI neural models to produce much more accurate output in comparison with MLP.

Since the output information of a coarse model is taken as input parameters, PKI can improve the speed of learning although the same structure as that of MLP is used.The one-to-one learning makes the training algorithm much more easily converged and less sensitive to the initial values of the weight parameters. However, when the problem behavior is highly nonlinear, PKI will face some trouble due to the inaccuracy of the coarse models. In this case, with empirical formula representing the relationship between input and output parameters, KBNN could offer better result compared to PKI.

There are also many other kinds of neural network structures, such as radial basis function neural networks, which can offer better accuracy than MLP when training data are large in number and sufficient; recurrent networks [4] and dynamic neuron networks for the modeling of the time-domain behaviors of a dynamic system.

2.6 Conclusion

In this chapter, 3 neural networks potentially important for RF and microwave applications are presented in detail. MLP is the simplest and most commonly used neural network structure. It has a great variety of applications. However, it could not always yield satisfactory results. When some empirical information is available, KBNN and PKI could be used to improve the accuracy, the converging rate and the generalization capability of the neural network.

Chapter 3

Neural Network Model Development

In chapter 2, several neural network structures were described for the development of a neural network model. However, a neural network cannot represent any device/circuit behavior unless it is trained with corresponding measured/simulated data. Typically, a neural network model development procedure includes problem identification, data generation, data splitting, data scaling, initialization of the weight parameters, training, testing, and result analysis. Presented in this chapter is a systematic description of the neural network model development covering all the above steps.

3.1 Problem Identification

At the first step, the input and output vectors $(\boldsymbol{x}, \boldsymbol{y})$ should be identified according to the particular problem to solve. For instance, the inputs could be the frequency or physical dimensions of a component, and outputs could be the S parameters of a two port network. The purpose of training of the neural network is to learn the relationship between the input vector $\boldsymbol{x}$ and the output vector $\boldsymbol{y}$.

3.2 Data Generation

Different from other modeling approaches such as equivalent circuit modeling, modeling with neural networks needs a set of data to train and test the neural network. Usually, the data can be generated from simulation or measurement. In practice, in order to get an accurate model, the data should be as accurate as possible. However, since the aim of this thesis is to develop a neural network modeling tool, we will highlight the tool efficiency through examples from the RF and microwave area. The data will be used to show the learning ability of the neural network package.

Typically, data are generated by pairs, (x_k, y_k) $k = 1, 2, \cdots, N$, where N is the total number of data samples. We should determine the range and distribution of the data samples. The ranges of the input parameters should cover all of the model application range. Because of basic mathematical properties of fitting functions, the error could be a little bit larger at the boundaries of the input parameter space. So we suggest wherever it is possible, the data sample range should be a litter bit beyond the application range to ensure a better performance at the boundaries of the input parameter space.

Once the range of the input vector is determined, one needs to choose a sampling strategy. The most frequently used sampling strategies are described as follows:

- Uniform grid distribution

  In this distribution, the input parameters are sampled at equal intervals. This is the simplest sampling strategy, but it could always lead to a large number of samples;

- Non-uniform grid distribution

Opposite to the uniform grid distribution, the input parameters are sampled at unequal intervals. This kind of distribution is especially suitable when the problem behavior is highly nonlinear in some sub-regions of the input space while quite linear in other sub-regions. We can use dense distribution in the highly nonlinear sub-regions and sparse distribution in the smooth sub-regions. Thus we can reduce the number of sampling data but at the same time we still have enough data to guarantee the accuracy of the neural model;

- Random distribution

In this distribution, the data are sampled randomly in the input parameter space. Since fewer samples are generated, when the input space dimension is higher, the random distribution can be applied to improve the efficiency.

3.3 Data Splitting

Normally , three sets of data are required in developing a neural model. They are training data T_r, validation data V and test data T_e. Training data are used to train the neural network - that is to update the weight parameters during the process of training. Validation data are used to supervise the training quality so that once the quality reaches the desired level, the training process could be terminated. Test data are used to examine the quality of the neural network after the development of the neural model. For simplicity, we use training data to monitor the quality of the neural network as well as to

guide the training process, and we use test data to examine the final quality of the neural network.

Ideally, each set of data should be adequate enough to represent the original problem in the input parameter range but should avoid overlapping. In practice, we could split the whole sampling data into T_r and T_e. The ratio of T_r to T_e is problem-specific. Usually we could split the data as 80%-20% between T_r and T_e.

3.4 Data Scaling

The orders of magnitude of the input and output parameters in microwave applications can be very different. For example, the frequency could be in the order of Giga Hz (10^9), and the dimension of a component could be in the order of millimeter (10^{-3}). On the other hand, from the characteristics of an activation function such as sigmoid function, we can see that if the input value of the activation function is much larger than 1, it will make the function saturated. In other words, if the input of the activation function is too large, its output will always be 1. Therefore, the scaling of the training data is necessary for the efficiency and accuracy of the neural network.

Usually, we scale the input parameters before the weight update and after all the processes descale the output parameters to give the actual values of the outputs. Various scaling schemes, such as linear scaling, log-arithmetic scaling, and two-side log arithmetic scaling, are all applicable for this situation. Among those, we choose the most commonly used and the simplest linear scaling.

Let x, x_{min} and x_{max} represent a generic element in the vector $\boldsymbol{x}, \boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$ of the original data respectively. Let $\bar{x}, \bar{x}_{min}$ and $\bar{x}_{max}$ represent a generic element in the vector $\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}_{min}$ and $\bar{\boldsymbol{x}}_{max}$ of the scaled data respectively, where $\left[\bar{x}_{min}, \bar{x}_{max}\right]$, which is [-1, 1] for the sigmoid activation function we choose, represents the input parameter range after scaling. The linear scaling is given by:

$$\bar{x} = \bar{x}_{min} + \frac{x - x_{min}}{x_{max} - x_{min}} (\bar{x}_{max} - \bar{x}_{min}) \tag{3.1}$$

The corresponding descaling function is given by:

$$x = x_{min} + \frac{\bar{x} - \bar{x}_{min}}{\bar{x}_{max} - \bar{x}_{min}} (x_{max} - x_{min}) \tag{3.2}$$

Linear scaling can improve the condition of the weight parameters, and balance the difference between the input parameters. Linear scaling of the output parameters can also balance the difference between output parameters whose magnitudes may vary significantly from one to another.

3.5 Initialization of Neural Network Weight Parameters

The weight parameters of the neural network need to be initialized to provide a start point before the beginning of the training (optimization) process. Random initialization scheme

is the most widely used method for the initialization of MLP weight parameters. In this scheme, the weight parameters are initialized with small random values (e.g., in the range of [-0.5, 0.5]). The random initialization can improve the convergence of training process. One can use different distributions (uniform or normal), different ranges, different variances for this kind of random number generation.

For the weight parameter initialization of the KBNN neural networks, one can use the values of the coefficients of the empirical formula or use the same scheme as that for the MLP neural network weight parameter initialization.

3.6 Training

Besides neural network structures, training algorithm is another important aspect in developing a neural network model. An appropriate structure doesn't guarantee the efficiency of the neural network unless a proper training algorithm is chosen. A good training algorithm can speed up the training process and achieve higher accuracy as well. After the Back Propagation (BP) was proposed in the middle of 1980's, a variety of optimization algorithms have been used on the basis of it to improve the efficiency and accuracy of the training process.

Generally speaking, all the training techniques can be classified into 2 classes: gradient-based class such as conjugate gradient algorithm and non-gradient-based class such as simplex method. Another way to classify the techniques is based on their ability to escape from the traps of local optimum, leading to local optimization methods and global ones.

All the algorithms discussed here are gradient-based local optimization algorithms, which include steepest descent method, conjugate gradient method, quasi-Newton method and Levenberg-Marquardt and Gauss-Newton method. The gradient derivation of different neural networks is described in Appendix A.

3.6.1 Training Objective

Training is an optimization process of the neural network to find the optimal values of the weight parameters so that the difference between the outputs of the neural network model and the actual outputs is minimized.

A set of training data are fed to the neural network in pairs of $(\boldsymbol{x}_k, \boldsymbol{d}_k)$, $k = 1, 2, \cdots, P$, where $\boldsymbol{d}_k$ is the desired output of the neural model for the input $\boldsymbol{x}_k$ and P is the total number of training samples. The performance of the neural network model is evaluated by training error, E_{T_r}, which is the difference between the actual neural network outputs and desired outputs of the training data, and by test error, E_{T_e}, which is similarly defined. The objective of training process is to minimize E_{T_r}, which is quantified by

$$E_{T_r} = \frac{1}{2}\sum_{k\in T_r}\sum_{j=1}^{m}(y_j(\boldsymbol{x}_k, \boldsymbol{w}) - d_{jk})^2 \tag{3.3}$$

where d_{jk} is the j^{th} element of $\boldsymbol{d}_k$, $y_j(\boldsymbol{x}_k, \boldsymbol{w})$ is the j^{th} neural network output for input

x_k and T_r is an index set of training data. The values of the weight parameters w are updated during the training process to minimize training error.

3.6.2 Back Propagation

Rumelhart, Hinton and Williams proposed an algorithm for neural network training called Back Propagation (BP) [14] in 1986. In this algorithm, the input signals are first fed to the neural network to carry out a forward calculation. After that the outputs of the neural network are compared with the desired values to get the error signals. Then the error signals propagate back from the output layer to the input layer (layer by layer) through the network to update the weight parameters. That is why it is called back propagation.

Figure 3-1 depicts a portion of the multilayer perceptrons and illustrates the concept of back propagation: input signal forward propagation and error signal backward propagation [8].

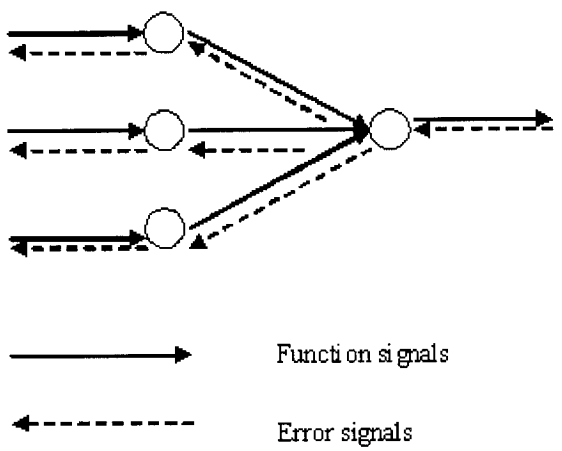


Figure 3-1. Illustration of back propagation

3.6.3 Gradient-based Training Methods

Since supervised learning process of neural network could be considered as an optimization problem to which various optimization methods using gradient information could be applied.

For a general multi-dimensional optimization problem, if we start at a point P in a N-dimension space, and proceed from there in some vector direction $\boldsymbol{n}$, then any function of N variables $f(P)$ can be minimized along the direction of vector $\boldsymbol{n}$. Including a scalar η, the function can be minimized along the direction of vector $\eta \cdot \boldsymbol{n}$. In a neural network problem, the start point is the initial values of the weight parameters $\boldsymbol{w}_{initial}$, we want to update the values of the weights $\boldsymbol{w}$ epoch by epoch (according to the iteration in optimization field) along some direction to minimize the error function $E_{T_r}(\boldsymbol{w})$. Let $\boldsymbol{h}$ be the direction vector, η be the learning rate, $\boldsymbol{w}_{now}$ be the current value of $\boldsymbol{w}$, then the optimization will update $\boldsymbol{w}$ so that

$$E(\boldsymbol{w}_{next}) = E(\boldsymbol{w}_{now} + \eta \boldsymbol{h}) \ . \tag{3.4}$$

We can use the gradient information to get the direction vector. The main difference of the gradient-based algorithms lies in the procedure of determining successive update directions $\boldsymbol{h}$ [15].

Learning rate η can control the step size of weight update. It is a quite sensitive parameter in the training algorithm. Proper learning rate can make the algorithm much faster. Generally speaking, a small learning rate could make the training process more stable, but on the other hand it will need more interactions to get the optimal result. A big learning rate will speed up the training process, but it could lead to the oscillation of the weight parameters, making the training process unstable. Usually, η is a positive number in the range of [0, 1]. We could set it as a constant, for instance 0.1, or we could use some adaptation schemes. There are several kinds of schemes for such adaptation as follows:

- η can be adapted following stochastic theory, for example [16], as

$$\eta = \frac{c}{epoch} \tag{3.5}$$

- η adapts itself based on training errors [17], for an instance

  $\eta = \eta \times 125\%$ if E_{T_r} decreasing steadily during the recent epochs;

  $\eta = \eta \times 80\%$ otherwise.

Furthermore, once the update direction is determined, the optimal step size could be found by employing line search (e.g. golden search, bisection search) along the specified direction,

$$\eta^* = \min_{\eta>0} E(\eta) \tag{3.6}$$

In the formula:

$$E(\eta) = E(\boldsymbol{w}_{now} + \eta \boldsymbol{h}) \tag{3.7}$$

Although line search could give more accurate value of the learning rate for each step, if the training data are quite huge, it will be computationally intensive. In other words, we will spend a greater amount of time in calculating the step size. Therefore, we choose equation 3.5 as our adapting scheme.

3.6.3.1 Steepest Descent Method

The original BP is derived from the steepest descent method. So the weight parameters of the neural network are updated along the direction of negative gradient direction in the weight space. The update formula is shown as follows:

$$\Delta \boldsymbol{w}_{now} = \boldsymbol{w}_{next} - \boldsymbol{w}_{now} = -\eta \frac{\partial e_i(\boldsymbol{w})}{\partial \boldsymbol{w}}\Big|_{\boldsymbol{w}=\boldsymbol{w}_{now}} \tag{3.8}$$

One advantage of this method is that it is relatively easy to implement and to understand. Its update direction is always along the steepest descent direction (negative gradient direction). However, the error surface of the training objective function contains some very gentle slop planes due to the commonly used logistic activation functions. The values of the error gradients are too small for the weights to move rapidly on those planes. Although by choosing the adaptation scheme or using linear search to find a proper value of the learning rate the efficiency of the original BP algorithm could be improved, its converging speed is still slow for MLP. A quite large number of

interactions are needed to reach the optimization goal. This situation is more serious when the steepest descent method encounters a "narrow valley" in the error surface where the direction of the gradient is almost perpendicular to the direction of the valley. As such, many higher order optimization methods using gradient information are applied to improve the rate of convergence. Compared with the steepest descent method, these higher order methods have a sound theoretical basis and guaranteed convergence with a limited number of iterations. The early work in this area was demonstrated in [18], [19], with the development of second-order training algorithm for neural networks. We will review some of the higher order algorithms including conjugate gradient method, quasi-Newton method and Levenberg-Marquardt and Gauss-Newton method in the following section.

3.6.3.2 Conjugate Gradient Method

In the steepest descent method, the direction of updating is always along the local downhill gradient. This method involves many small steps in going down a long, narrow valley, even if the valley is in a perfect quadratic form [20]. The new gradient at the minimum point of any line minimization is always perpendicular to the direction it just traversed. At this point what we want to do is not to proceed down the new gradient but rather a direction which is conjugate to the old gradient direction and all other previous directions. Such an algorithm is called conjugate gradient method. One of the most important conjugate gradient methods is Fletcher-Reeves method which will be described in detail in the following paragraphs.

The conjugate gradient method is originally derived from quadratic minimization, where the minimum of the objective function E_{T_r} could be found within N_w iterations (number of weight parameters). With initial gradient $\boldsymbol{g}_{initial} = \frac{\partial E}{\partial \boldsymbol{w}}\Big|_{w=w_{initial}}$, and direction vector $\boldsymbol{h}_{initial} = -\boldsymbol{g}_{initial}$, the conjugate gradient method recursively constructs two vector sequences [21],

$$\boldsymbol{w}_{next} = \boldsymbol{w}_{now} - \eta \boldsymbol{h}_{next} \boldsymbol{g}_{now} \tag{3.9}$$

$$\boldsymbol{g}_{next} = \boldsymbol{g}_{now} + \lambda_{now} \boldsymbol{H}\boldsymbol{h}_{\text{now}} \tag{3.10}$$

$$\boldsymbol{h}_{next} = -\boldsymbol{g}_{now} + \gamma_{now} \boldsymbol{h}_{\text{now}} \tag{3.11}$$

$$\lambda_{now} = \frac{\boldsymbol{g}_{now}^T \boldsymbol{g}_{now}}{\boldsymbol{h}_{now}^T \boldsymbol{H}\boldsymbol{h}_{now}} \tag{3.12}$$

$$\gamma_{now} = \frac{\boldsymbol{g}_{next}^T \boldsymbol{g}_{next}}{\boldsymbol{g}_{now}^T \boldsymbol{g}_{now}} \tag{3.13}$$

where $\boldsymbol{h}$ is called the conjugate direction and $\boldsymbol{H}$ is the Hessian matrix of the objective function E_{T_r}. To avoid the calculation of the intensive Hessian matrix in determining the conjugate direction, we can proceed $\boldsymbol{w}_{now}$ along the direction $\boldsymbol{h}_{now}$ to the local minimum of E_{T_r} at $\boldsymbol{w}_{next}$ through line minimization, and then set $\boldsymbol{g}_{next} = \frac{\partial E_{T_r}}{\partial \boldsymbol{w}}\Big|_{w=w_{next}}$. In this method, the descent direction is along with a series of conjugate directions which can be calculated without much matrix computations. Meanwhile the memory it needs is only for a few N_w

(number of weight parameters) long matrices. That is also why the conjugate gradient method is very efficient.

3.6.3.3 Quasi-Newton Method

Similar to the conjugate gradient algorithm, quasi-Newton algorithm is derived from quadratic object function as well. A matrix $\boldsymbol{B}$ is used to approximate the inverse of the Hessian matrix $\boldsymbol{H}^{-1}$ to bias the gradient direction. In this method, the neural network weight parameters are updated as:

$$\boldsymbol{w}_{next} = \boldsymbol{w}_{now} - \eta \boldsymbol{B}_{now} \boldsymbol{g}_{now} \tag{3.14}$$

$$\boldsymbol{B}_{now} = \boldsymbol{B}_{old} + \Delta \boldsymbol{B}_{now} \tag{3.15}$$

We can estimate matrix $\boldsymbol{B}$ successively from the history of gradient directions, using rank 1 or rank 2 updates, and following each line search in a sequence of search directions [22]. The formula of computing $\Delta \boldsymbol{B}_{now}$ for rank 2 is:

$$\Delta \boldsymbol{B}_{now} = \frac{\Delta \boldsymbol{w} \Delta \boldsymbol{w}^T}{\Delta \boldsymbol{w}^T \Delta \boldsymbol{g}} - \frac{\boldsymbol{B}_{old} \Delta \boldsymbol{g} \Delta \boldsymbol{g}^T \boldsymbol{B}_{old}^T}{\Delta \boldsymbol{g} \boldsymbol{B}_{old} \Delta \boldsymbol{g}^T} \tag{3.16}$$

In the formula:

$$\Delta \boldsymbol{w} = \boldsymbol{w}_{now} - \boldsymbol{w}_{old} \tag{3.17}$$

$$\Delta \boldsymbol{g} = \boldsymbol{g}_{now} - \boldsymbol{g}_{old} \tag{3.18}$$

The initial of the $\boldsymbol{B}$ matrix can be set as the identity matrix.

Since N_w^2 units of space is required to store the approximation of the inverse of the Hessian matrix, the standard quasi-Newton method is not efficient for large scale neural networks. However, due to the estimation of the inverse of the Hessian matrix, this method can converge even faster than conjugate gradient method. For quadratic minimization, it can converge just in one iteration.

3.6.3.4 Levenberg-Marquardt and Gauss-Newton Method

In neural network training, the object function is always formulated as a nonlinear least-square form so that some methods to least-squares, such as Gauss-Newton, can be employed to update the weight parameters. Let $\boldsymbol{e}$ be a vector containing the individual error terms,

$$\boldsymbol{e} = [e_{11}\ e_{12}\ \cdots\ e_{mP}]^T \tag{3.19}$$

In the formula:

$$e_{jk} = y_j(\boldsymbol{x}_k, \boldsymbol{w}) - d_{jk},\ j \in \{1,2,\cdots,m\}, k \in T_r \tag{3.20}$$

Let $\boldsymbol{J}$ be Jacobin matrix containing the derivatives of error $\boldsymbol{e}$ with respect to $\boldsymbol{w}$. The Gauss-Newton update formula can be represented as [23],

$$\boldsymbol{w}_{next} = \boldsymbol{w}_{now} - (\boldsymbol{J}_{now}^T \boldsymbol{J}_{now})^{-1} \boldsymbol{J}_{now}^T \boldsymbol{e}_{now} \tag{3.21}$$

In this formula, $\boldsymbol{J}_{now}^{T}\boldsymbol{J}_{now}$ is positive defined unless $\boldsymbol{J}_{now}$ is rank deficient. When $\boldsymbol{J}_{now}$ is rank deficient, Levenberg-Marquart method can be applied [24]. The weight updating is given by,

$$\boldsymbol{w}_{next} = \boldsymbol{w}_{now} - (\boldsymbol{J}_{now}^{T}\boldsymbol{J}_{now} + \mu\boldsymbol{I})^{-1}\boldsymbol{J}_{now}^{T}\boldsymbol{e}_{now} \tag{3.22}$$

In this formula, μ is a non-negative number, $\boldsymbol{I}$ is the identity matrix. In this method, we need to calculate matrix inverse which is computationally expensive and requires a large memory (N_w^2). Furthermore, we have to consider a lot of problems accompanying matrix inverse calculation such as renumbering the matrix for sparse and accuracy which makes the matrix inverse calculation more computationally intensive. Therefore, this method is not suitable for large scale neural network training.

3.6.3.5 Comparison between the Different Training Methods

Among all the training methods we have reviewed, the steepest descent method is no doubt the simplest one, but due to its slow converging rate it is seldom used in real applications. In conjugate gradient method, the update direction is along a set of directions conjugate to each other, which speeds up the converging rate considerably. At the same time, because only simple matrix/vector calculation, such as summation, subtraction and production, is involved, it needs small memory space, which makes it very efficient. Compared to conjugate gradient method, quasi-Newton method and Levenberg-Marquardt and Gauss-Newton method converge even in fewer iterations. However, estimation of the Hessian matrix of quasi-Newton method and matrix inverse

calculation of Levenberg-Marquardt and Gauss-Newton method make them much more computationally intensive. Although fewer iterations are needed, the calculation time for each iteration increases remarkably. Consequently, quasi-Newton and Levenberg-Marquardt method are not suitable for large scale neural network training. Figure 3-2 [4] shows more clearly the comparison of the speed and memory needed for these different training methods.

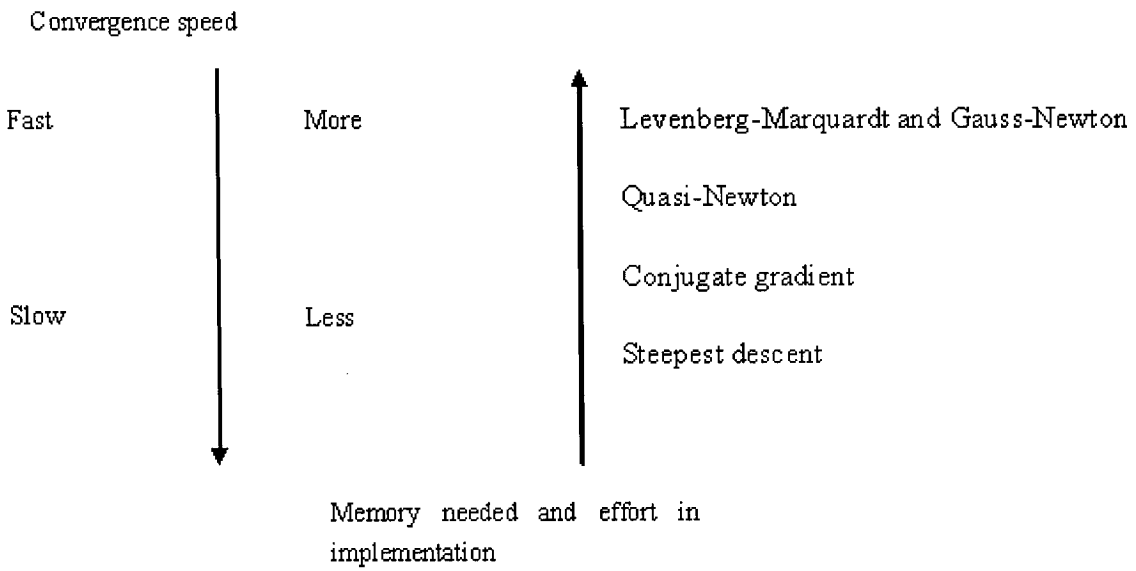


Figure 3-2. Training method comparison

3.6.4 Type of Training Process

The training process can be categorized in several ways. In one of them, it can be categorized into sample-by-sample training and batch-mode training.

In sample-by-sample training, which is also called online training, the neural network weight parameters ($\boldsymbol{w}$) are updated each time one training sample is fed to the neural network. That is to say, the neural network learns the sample data one by one. The weight

parameters are updated for iterations until the neural network learns the single sample well enough, which means the training error for this sample is very small and can meet the accuracy requirement. Then the neural network goes on with the next sample data. After the neural network learns all the training samples, a training error for all the training samples is calculated to see whether there is an accuracy improvement of the whole model . As we can see from the procedure of sample-by-sample training, it needs many iterations to learn even one sample. The learning process of the other sample may kill the improvement of the learning of this sample. This kind of training process is only applicable when the training data are extremely large.

For most of the RF and microwave applications, the number of training data is not too large, and we usually could get all of them at once. Sample-by-sample training is not very efficient. Then the batch-mode training process is preferred. Batch-mode training, also known as offline training, updates the weight parameters only after all the training samples are fed to the neural network. It uses the gradient information of all the samples to update the weight parameters. So, for each iteration the improvement is based on all the training samples rather than a single one. In comparison with the sample-by-sample training, it could save a lot of time in our case.

3.7 Result Analysis

We could get training error and test error after using the neural network tools. Usually, these errors are given in the form of percentage which is different from what we defined in equation 3.3. These percentage errors are called normalized errors which can represent

the accuracy of a neural network to estimate output $\boldsymbol{y}$ for given input $\boldsymbol{x}$. The normalized training error is defined as follow:

$$\hat{E}_{T_r}(\boldsymbol{w}) = \frac{1}{size(T_r)\cdot m}\sum_{k\in T_r}\sum_{j=1}^{m}\left|\frac{y_j(\boldsymbol{x}_k,\boldsymbol{w}) - d_{jk}}{d_{\max,j} - d_{\min,j}}\right| \tag{3.23}$$

In the formula $d_{\max,j}$ and $d_{\min,j}$ are the maximum and minimum values of the j^{th} element from all vectors of $\boldsymbol{d}_k, k \in T_r$. The normalized test error $\hat{E}_{T_e}$ can be similarly defined. Training error E_{T_r} of equation 3.3 is used to update the weight parameters, while normalized training and test error - $\hat{E}_{T_r}, \hat{E}_{T_e}$ are used to evaluate the accuracy of the neuron model. Hence, whenever we mention training and test error at result analysis, we usually refer to the normalized errors.

If both the normalized training error and test error we get are very small and close to each other, this kind of learning is defined as good learning, which means the neural model matches both the training data and test data well.

On the other hand, we could get overlearning or underlearning. Overlearning is a situation in which the neural network can learn the training data well, but cannot generalize well ($\hat{E}_{T_e} >> \hat{E}_{T_r}$). That is to say, the training error could be very small, but the test error is quite large compared with the training error. There are several possible reasons for overlearning:

- Too many hidden neurons leading to too much freedom in the $\boldsymbol{x} - \boldsymbol{y}$ relationship represented by the neural network;

- Insufficient number of training data which can not represent the characteristics of the original problem.

Correspondingly, we can delete the number of hidden neurons or add more training data to make some improvement.

Opposite to overlearning, when the training error itself is quite large ($\hat{E}_{T_r} >> 0$), we have underlearning. Possible reasons for underlearning could be:

- Insufficient number of hidden neurons;
- Training procedure is stuck in a local minimum;
- Insufficient training.

We can add more hidden neurons, start at a different initial point or continue training process to solve this problem.

3.8 Conclusion

In this chapter, we have reviewed the neural network modeling procedures. Some of the key issues relating to the process were discussed in detail. With a good theoretical basis, neural networks have a wide application in the field of RF and microwave modeling.

Chapter 4

Neural Network Tool Development

Based on the theoretical foundations of neural networks we have described in the previous chapters, MLP, KBNN and PKI neural networks were developed in Java programming language, running on a MS-Windows operating system. The executing time varies depending on the size of training samples and the nonlinearity of the problem concerned. Then, we validated our neural network tool with a well established commercial neural network tool, namely Matlab Neural Network Toolbox [17]. Our neural network is proving to have similar accuracy as Matlab Neural Network Toolbox.

4.1 Tool Development

The flow chart of the development of the neural networks is shown in figure 4-1. To use this CAD tool, the user only needs to define a network structure and feed training data and test data to the network as the inputs. After the training and the testing, the tool gives the training and test errors to evaluate respectively the training process and the test results, as well as the scaling information and the trained weight parameters that represent the problem model. To develop such a tool, we need to preprocess the input data

(scaling), give proper initial values to weight parameters, and then use appreciated training algorithm to update the weight parameters. Once the training and the testing are terminated, we have to descale the outputs and display the final results.

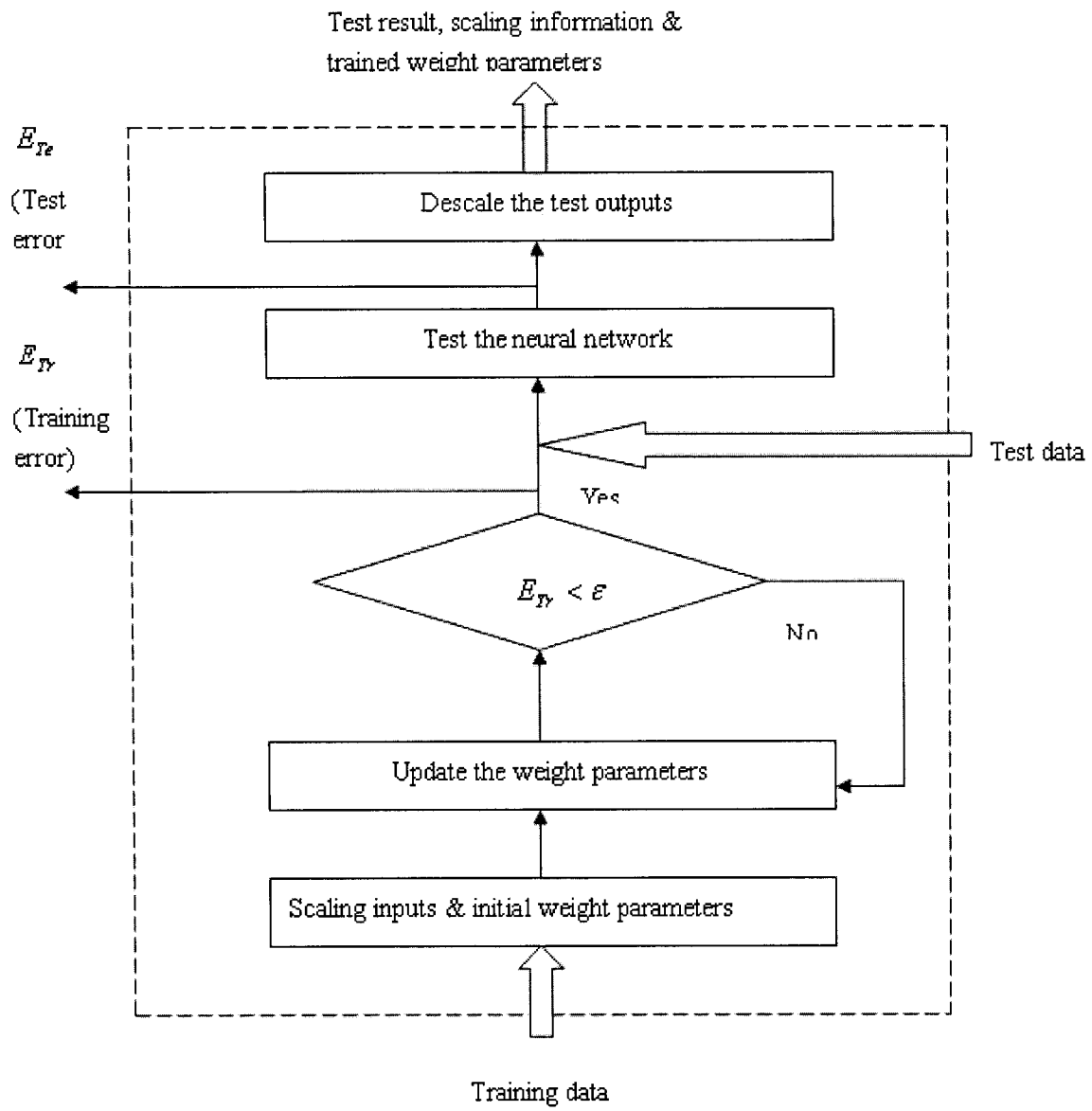


Figure 4-1. Neural network development flow chart

The detailed algorithm of neural networks is implemented. The user interfaces for MLP, KBNN and PKI are shown in figures 4-2 to 4-4.

According to the universal approximation theory, 3–layer MLP could approximate any nonlinear, continuous, multi-dimensional function f with any desired accuracy. In practice, it is applicable in most of the cases. Thus, for simplicity, our default MLP neural network is a 3-layer MLP. The user has to define the number of inputs, the number of hidden neurons and the number of outputs, and has to specify the learning rate, the desired accuracy and also has to enter the maximum number of iterations. As to the training methods, we offer four options which are steepest descent method, conjugate gradient method, quasi-Newton method and Levenberg-Marquart and Gauss-Newton method.

Since KBNN is a problem-dependent neural network, we have to write the empirical formulas for each specific problem we want to model. However, the user interface is similar to that of MLP. Two training methods are provided in this case, which are steepest descent method and conjugate gradient method.

As to PKI, because we use MLP to learn the input and output relationship, the only preparation work prior to MLP is to combine the outputs of the coarse model with the original inputs (as shown in figure 4.4 with one more button for combining compared with MLP).

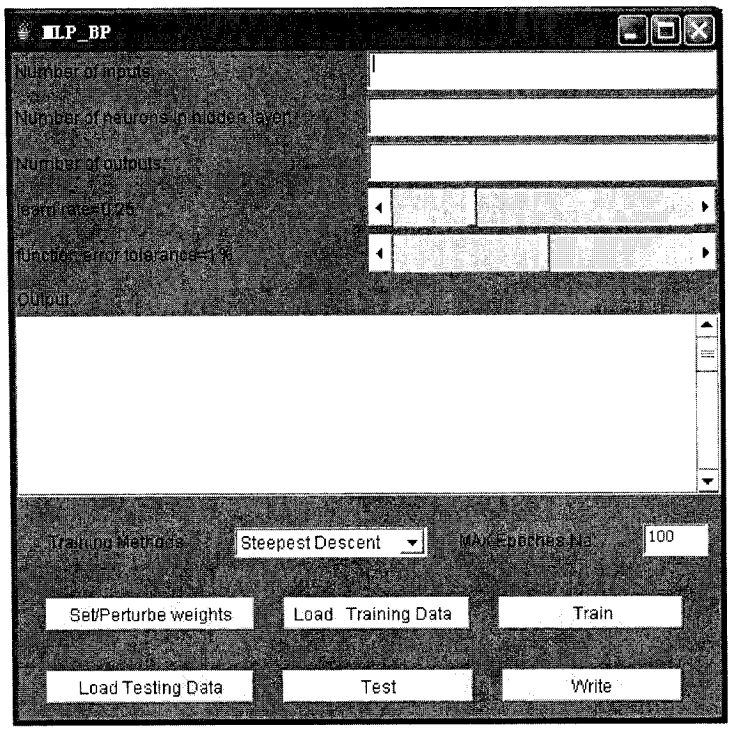


Figure 4-2. MLP user interface

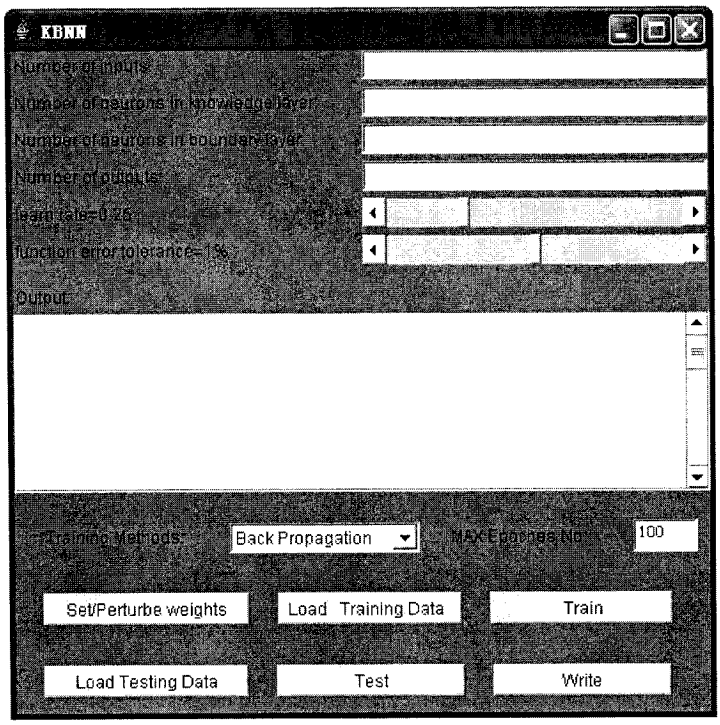


Figure 4-3. KBNN user interface

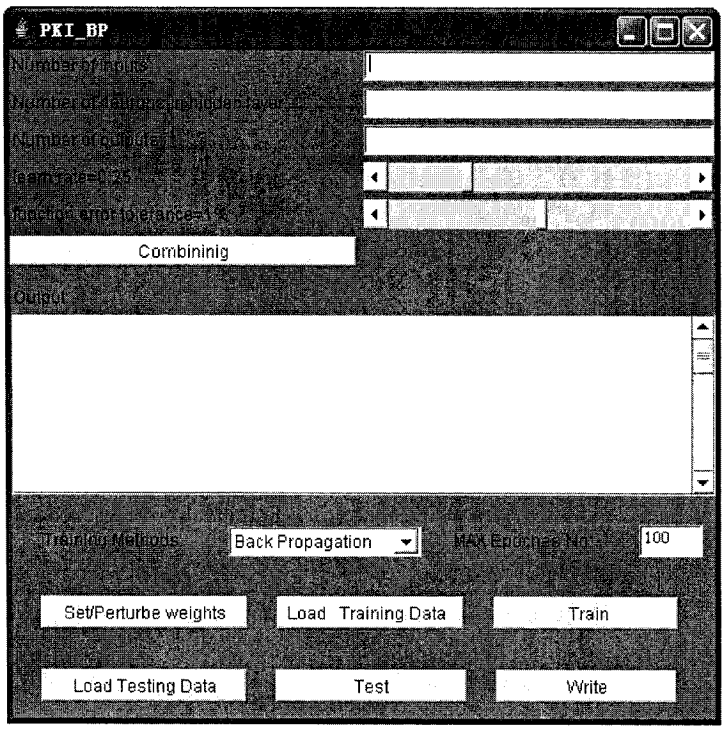


Figure 4-4. PKI user interface

4.2 Validation

After developing the codes for the neural network structures, we validated their performance.

4.2.1 MLP Validation

We have used three examples both to validate the performance of MLP neural network code and to highlight its limitations.

Example #1:

We use quadratic function with 2 inputs and 1 output to generate the data. The total sample number is 201, from which we randomly choose 20% (40) as test data and the rest (161) as training data.

The results are shown in table 4-1. We can see Matlab and our neural model have the same level of errors.

Table 4-1. MLP validation example #1

	Training error	Test error	Number of hidden neurons
MATLAB	0.467%	0.475%	3
Our Neural Model	0.506%	0.520%	3

The comparison of the original data and the neuron model output from our tool and Matlab is shown in figure 4-5, using test samples. As expected from the training error, we can see from the figure that our neuron model matches the original data and the Matlab model very well. We also validated our tool by another commercial software, NeuroSolutions [25], through this example in Appendix B.

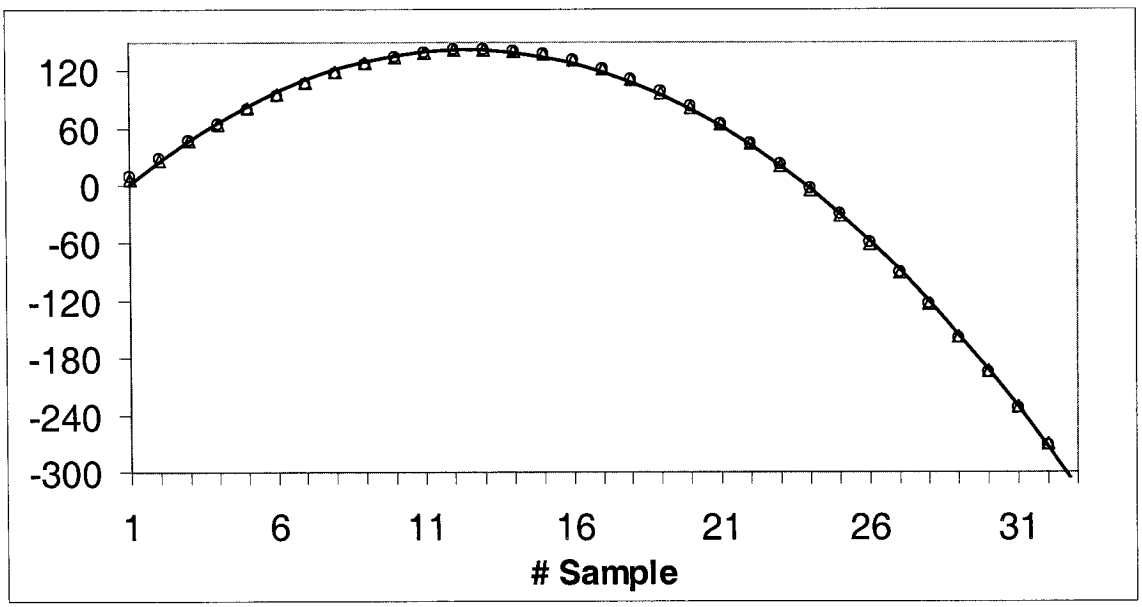


Figure 4-5. Neural model output (∘) compared with the original data (-) and Matlab neural model (Δ) in MLP neural network validation example #1

Example #2:

Similar work has been done for the example #2 using a set of mathematical functions combining low and large output variations. The results in table 4-2 show that Matlab and our neural model have similar errors.

Table 4-2. MLP validation example #2

	Training error	Test error	Number of hidden neurons
MATLAB	1.152%	1.356 %	8
Our Neural Model	1.398%	1.554%	8

The comparison of the original data and the neuron model outputs from our tool and Matlab is shown in figure 4-6. From this comparison, several points can be raised. First, we can see that when the curve is quite smooth, our neural model could match the original data very well, while the error will increase a little bit in the other part. This is not due to a malfunction of our tool since the results from Matlab exhibit a similar error. The next example will enforce this conclusion. Second, a neural model cannot, as expected, learn the problem very accurately at the boundaries of the input range. One solution would be to extend the input data range while keeping the same step size for data generation. Since the problem presents a strong nonlinearity in the specific sub-region, the other solution would be adding more data to help the tool to learn the problem behavior better in such sub-region, i.e, reducing the step size in this sub-region while keeping the original data range. Third, one can think of an underlearning case. Therefore, more hidden neurons would be required. However, due to the small test error, such direction should not improve the results. We will try to highlight all these fundamental points regarding the neural model training in the next example.

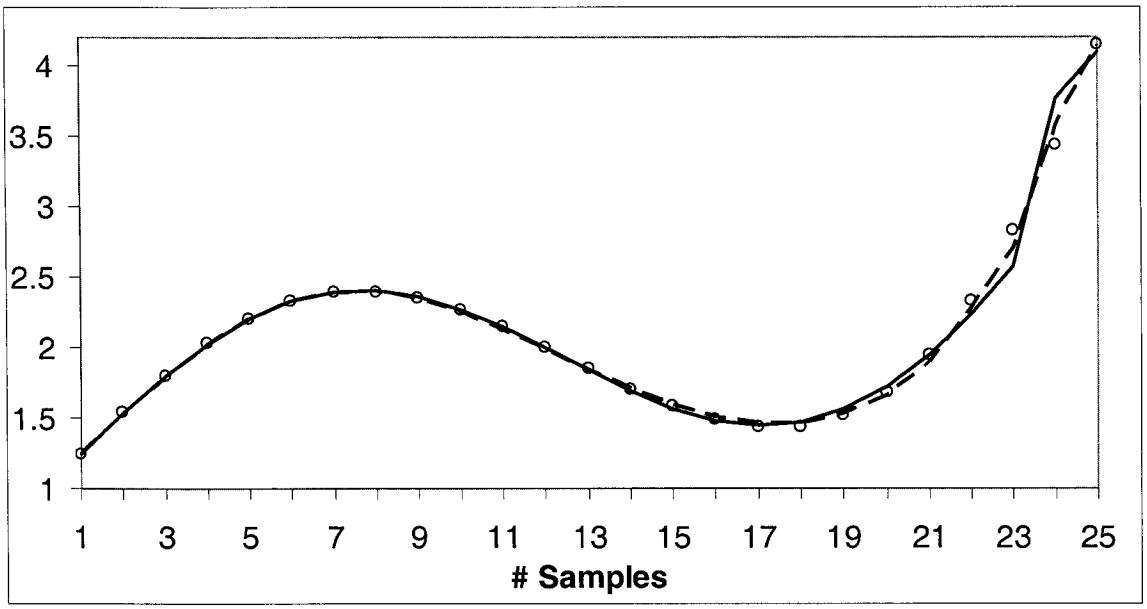


Figure 4-6. Neural model output (◦) compared with the original data (-) and Matlab neural model (--) in MLP neural network validation example #2

Example #3:

We reused example #2 in order to improve the results. First, we added more data keeping the same input data range, i.e, reducing the data step size and adding more data in the upper range (where the error is large) to learn the problem behavior better. Second, the input range was extended to cover the nonlinear part of the problem while keeping the initial step size as in example #2. Finally, we added the number of hidden neurons. The results are shown in table 4-3 along with those shown in figures 4-7 to 4-9. More hidden neurons will lead to a more complex weight parameter surface. As a result, the optimization process is much more easily stuck at some local optimum point. That may be the reason why the error is a little bit larger with more hidden neurons (as shown in table 4-3). Both adding more data and extending the data range would improve the

accuracy. Such an example helped us to demonstrate the accuracy of our tool vs. Matlab and to highlight the properties of MLP structures.

Table 4-3. MLP validation example #3

		Training error	Test error	Number of hidden neurons
Adding more data in the upper sub-region (keeping the original data range)	MATLAB	1.038%	1.218%	8
	Our Neural Model	1.060%	1.196%	8
Extending the data range (keeping the original step size)	MATLAB	0.907%	0.989%	8
	Our Neural Model	0.927%	1.045%	8
Adding more hidden neurons (keeping the original data)	MATLAB	1.755%	1.810%	10
	Our Neural Model	1.732%	1.839%	10

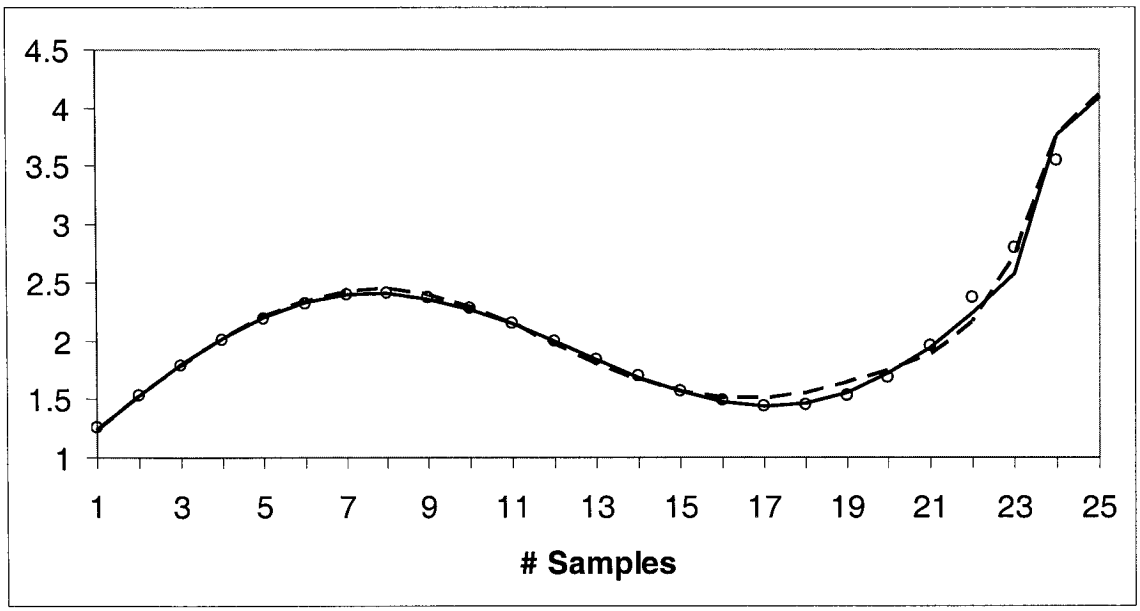


Figure 4-7. Reduction of the step size: Neural model output (◦) compared with the original data (-) and Matlab neural model (--) in MLP neural network validation example #3

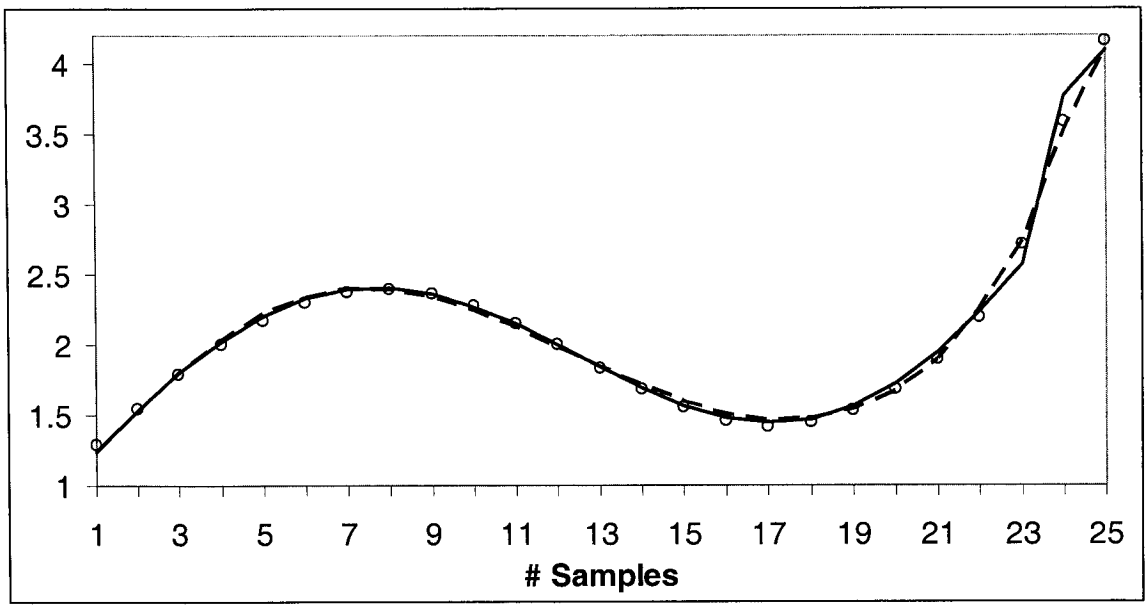


Figure 4-8. Extension of the data range: Neural model output (◦) compared with the original data (-) and Matlab neural model (--) in MLP neural network validation example #3

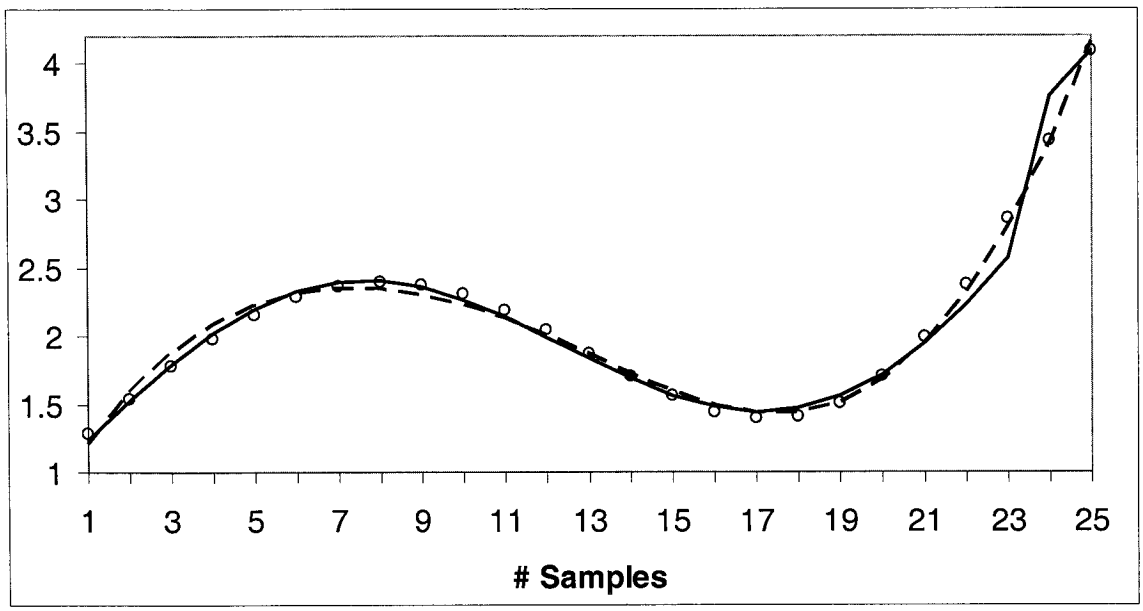


Figure 4-9. Addition of hidden neurons: Neural model output (◦) compared with the original data (-) and Matlab neural model (--) in MLP neural network validation example #3

4.2.2 KBNN Validation

We want to approximate the function of $f(x) = 2x^2 + e^x + 5$ in the input x space of [-5, 5] to see whether the KBNN neural network works well. We use $f(x) \approx e^x$ as the empirical function which is included in the neural network. The results are shown in table 4-4.

Table 4-4. KBNN validation

	Training error	Test error	Number of knowledge neurons	Number of boundary neurons
Our Neural Model	0.230%	0.275%	2	2

The comparison of the original data and our neuron model output is shown in figure 4-10. We can see from the figure that our neuron model output curve almost overlaps the original data curve.

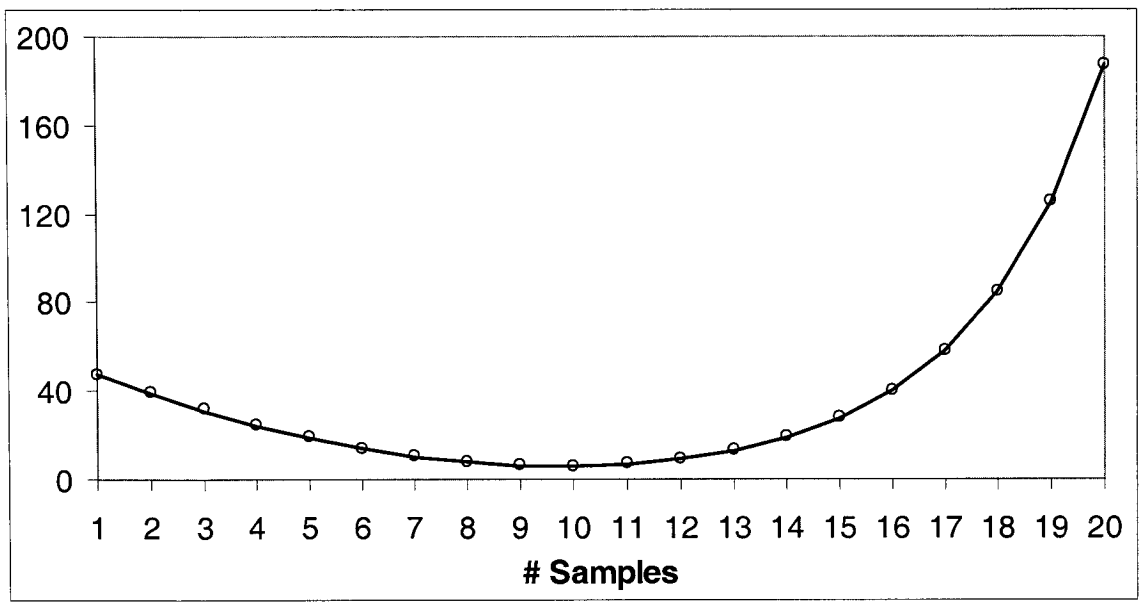


Fig. 4-10. Neural model output (◦) compared with the original data (-) in KBNN neural network validation

4.2.3 Comparison of Matlab Neural Network Toolbox and Our Neural Network Tool

Matlab Neural Network Toolbox is a general neural network tool which can be applied in a variety of fields such as pattern reorganization, speech processing, control, medical applications, and so on. For users in a specific field to use such a general tool, they must define a lot of variables for the structure and algorithm, which means a lot of preparation work before using the tool. Typically, in the RF and microwave field, the most commonly used neural network structure is MLP, and for some complex applications, KBNN and PKI may be used to help the training process. Furthermore, the algorithms usually used are quite limited. In such a situation, a more specified neural network tool is preferred. Our MLP neural network having the same level of accuracy with MATLAB leaves fewer variables to define, which proves very convenient for users who have less inside knowledge of neural networks.

Meanwhile, in the RF and microwave field, the performance of some components/circuits is quite complex and only some coarse empirical/equivalent models are available. Taking advantages of this empirical information to improve the accuracy and efficiency is a promising trend. For some problem-dependent neural networks such as KBNN and PKI, a small change in our program is enough to achieve the objective, which is contrary to Matlab.

As for the data format, Matlab only accepts the data in MAT file format which is quite strict. For our neural network tool, the most basic *.txt files, *.dat files, or *.xls files are

acceptable. These data files with basic format are easy to get whether the source data are from measurements or from simulations.

On the other hand, Matlab Neural Network Toolbox still has its advantages. It provides a complete set of functions and a graphical user interface for the design, implementation, visualization, and simulation of neural networks. It can support the most commonly used supervised and unsupervised network architecture and it has a comprehensive set of training and learning functions [17]. Furthermore, the routines implementing different algorithms inside the toolbox are much more complete and mature, which makes them more efficient when handling a large number of data.

4.3 Conclusion

In this chapter, we depicted the detailed process of developing our own neural network tool for MLP, KBNN and PKI. The final neural network tool is validated through examples. It has been proved that our neural network tool can model some nonlinear functions with certain accuracy. Compared with the Matlab Neural Network Toolbox, it is more specified in RF and microwave field, and more flexible when the user wants to include some empirical information into the neural network.

Chapter 5

Design Examples Using Neural Networks

In this chapter, we demonstrate the features of our neural network tool such as speed, accuracy, and efficiency. At the component level, we used MLP, KBNN, and PKI to model both embedded passive components such as resistor, capacitor, and square-spiral inductor and active components such as FET. At the circuit level, a mixer and multistage amplifiers were modeled. The advantages of each neural structure were demonstrated through theses examples.

In this chapter, the model size, for MLP, is the number of hidden neurons; and for KBNN, means that the number after letter "b" is the number of neurons in the boundary layer, and that the number after letter "z" is the number of neurons in the knowledge layer.

5.1 Resistor Modeling Using MLP and KBNN

Embedded passives represent an emerging technology area that has the potential for increased reliability, improved electrical performance, shrunk size, and reduced cost [26]. The conventional approach for circuit and system design is the equivalent circuit capturing the response of embedded passives. However, the existing equivalent circuit method may not be accurate enough to reflect high frequency EM effects. Even if we can find an accurate equivalent circuit to represent high frequency EM effects, the component values in the equivalent circuit do not directly represent the embedded passives' structural geometrical/physical parameters. Therefore, accurate models of embedded passives which can relate physical parameters to the components' value for high frequency are needed.

In this example we employed both MLP and PKI to model resistors (figure 5-1) whose physical parameters are within the ranges reported in table 5-1. The data were generated [12] from the EM simulator Sonnet-Lite [27].

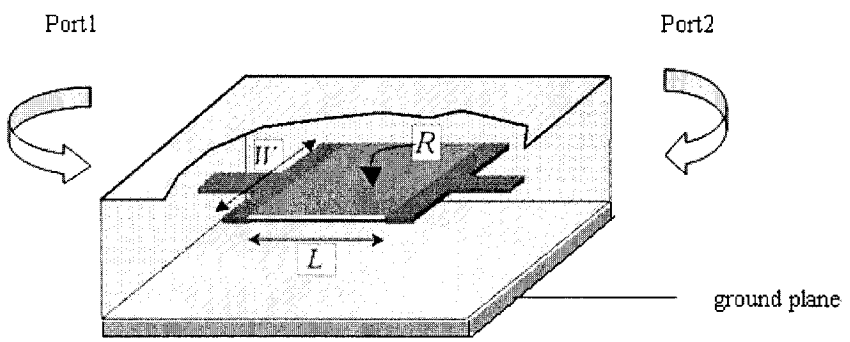


Figure 5-1. Resistor: Physical structure

Table 5-1. Resistor: Ranges of input parameters

Input parameter	Symbol	Range	Step size
Frequency (GHz)	f	1-10	0.1
Width (mils)	W	6-20	2
Length (mils)	L	6-20	2
Permittivity	ε	2-7	1
Resistivity ($\Omega/\mu m^2$)	R	10-200	50

In this example, the neural model output parameters are the real part and the imaginary part of the S-parameters, S_{11} and S_{12}. For the KBNN neural networks, we use the equation of the resistance for low frequencies as the empirical formula.

The final results are shown in table 5-2. We can conclude that with the same number of training samples, KBNN can get much better accuracy than MLP. Furthermore, when the test data are beyond the training range, KBNN could offer better accuracy. Due to the large number of training data and the large scale of neural networks, the calculation time of MLP increased tremendously to around 17 minutes per iteration which makes it almost inapplicable, while for KBNN the calculation time is only around 20 seconds per iteration. Obviously, KBNN is much more efficient in this example.

Table 5-2. Resistor: Accuracy comparison between MLP and KBNN

structure	# Model size	E_{T_r}	E_{T_e} within training range	E_{T_e} beyond training range	Number of iterations	Calculation time per iteration
MLP	20	5.438%	11.088%	10.282%	354	17min
KBNN	b8z9	2.861%	4.448%	4.231 %	376	20 sec

The model accuracy comparison of MLP and KBNN in terms of the output parameters is shown in figures 5-2 to 5-5, which exhibit good agreement with the original data from EM simulator.

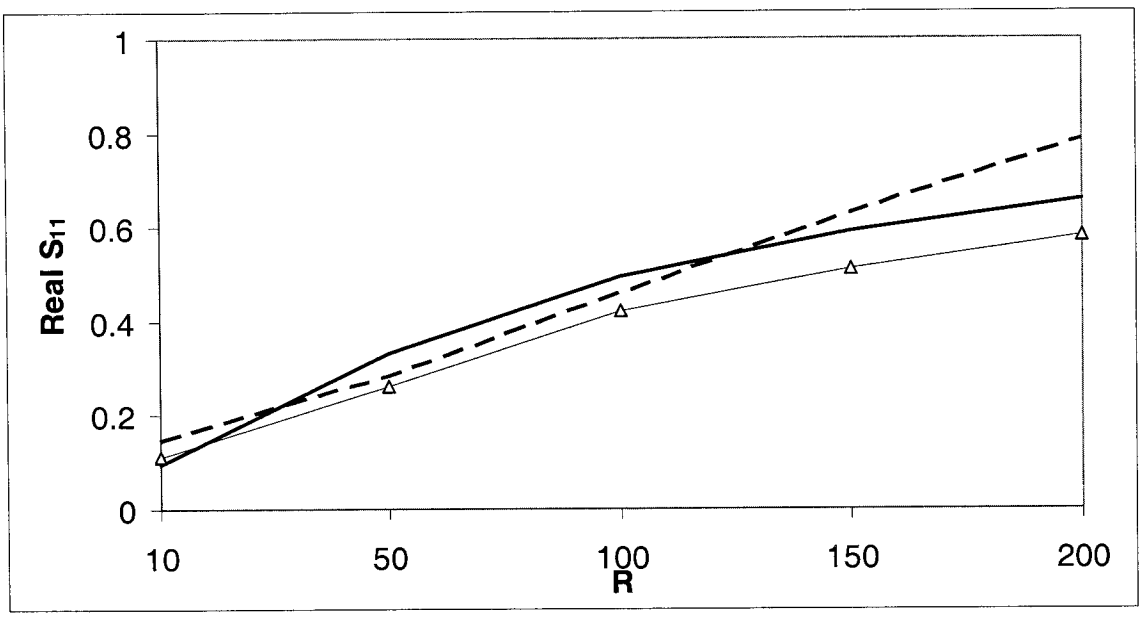


Figure 5-2. Resistor: Real part of S_{11}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

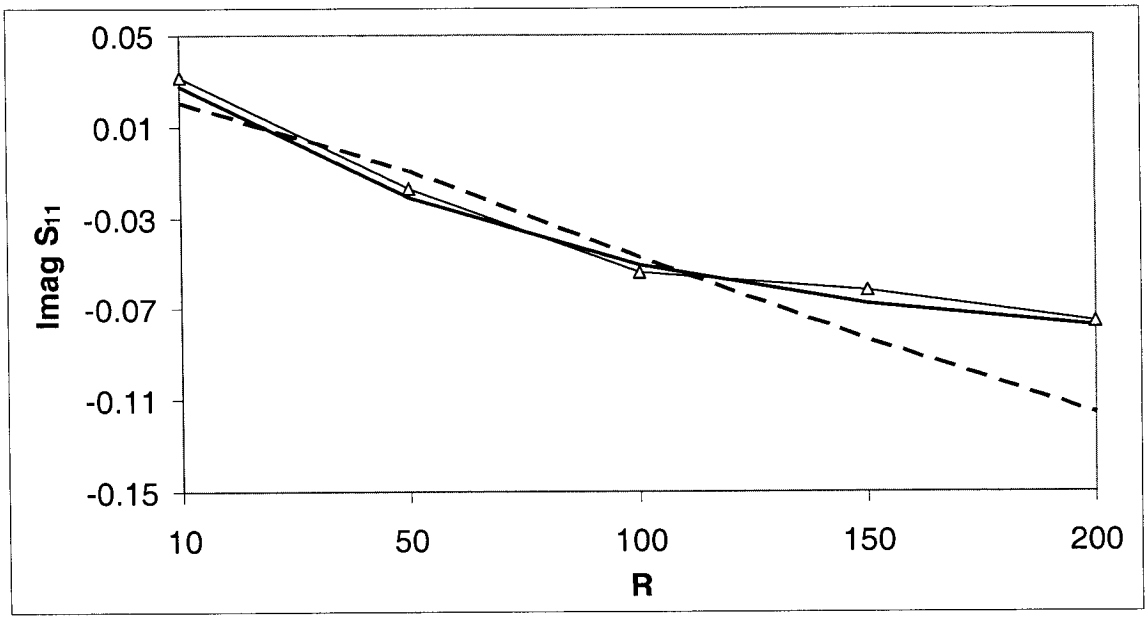


Figure 5-3. Resistor: Imaginary part of S_{11}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

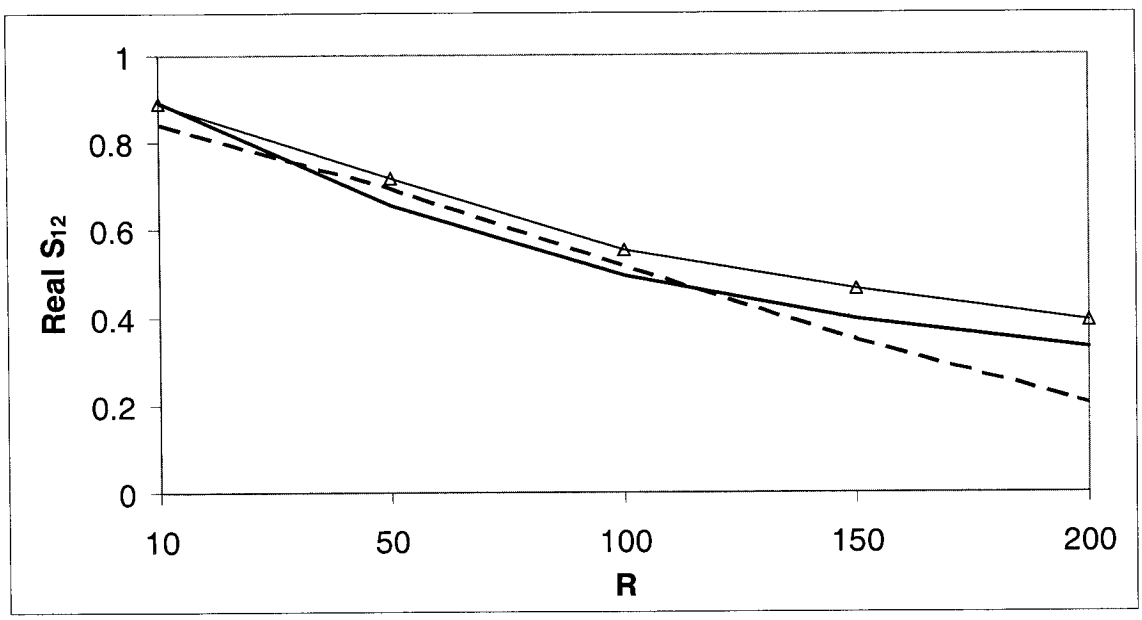


Figure 5-4. Resistor: Real part of S_{12}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

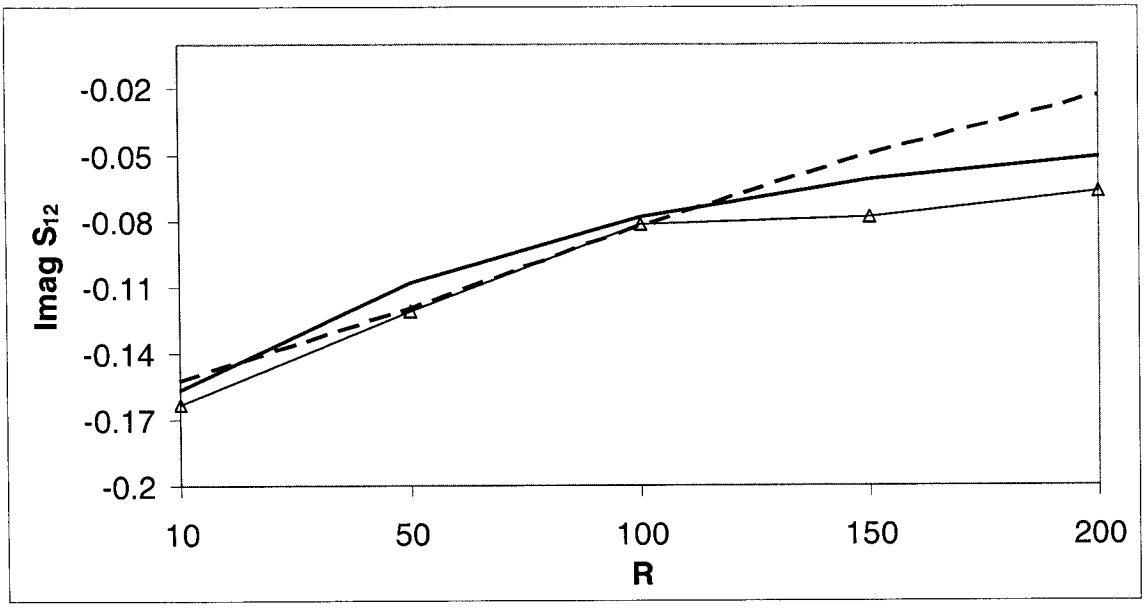


Figure 5-5. Resistor: Imaginary part of S_{12}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

5.2 Capacitor Modeling Using MLP and KBNN

Similar work has been done for the capacitor. By varying the frequency (f), the side length (L), the thickness (T) between plates, the relative permittivity (ε_r) of the environment and the capacitor dielectric constant (ε_{rcap}), EM-data for square capacitors (figure 5-6) have been generated.

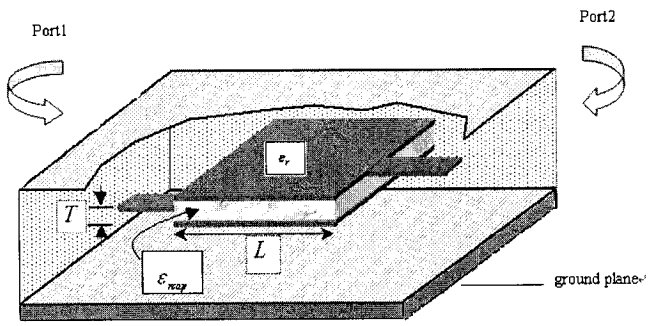


Figure 5-6. Capacitor: Physical structure

Table 5-3. Capacitor: Ranges of input parameters

Input parameter	Symbol	Range	Step size
Frequency (GHz)	f	1-10	0.1
Length (mils)	L	6-20	2
Thickness (mils)	T	0.2-0.6	0.2
Permittivity of the environment	ε_r	2-7	1
Capacitor dielectric constant	ε_{rcap}	100-3000	500

The final results are shown in table 5-4. We can reach the conclusion that KBNN can achieve higher accuracy with the same number of training samples. For example, with 7978 training samples, the accuracy of KBNN is 1.915% while the accuracy of MLP is only 2.792%. Knowing this error is the mean error, an error for 1% in training may help to improve the model accuracy significantly in some highly nonlinear region.Due to the help of the empirical formula, KBNN could have a much smaller weight parameter space which consequently leads to fewer iterations and less calculation time.

Table 5-4. Capacitor: Accuracy comparison between MLP and KBNN

Structure	Model size	Training data number	Test data number	E_{T_r}	E_{T_e}	Number of iterations	Calculation time per iteration
MLP	20	7978	1994	2.792%	2.812%	577	15 min
KBNN	b8z9			1.915%	1.956%	173	10 sec
MLP	20	7479	2493	3.642%	3.615%	501	15 min
KBNN	b8z9			3.191%	3.178%	194	10 sec

The model comparison of MLP and KBNN in terms of the output parameters is shown in figures 5-7 to 5-10, which exhibit good agreement with the original data from EM simulator.

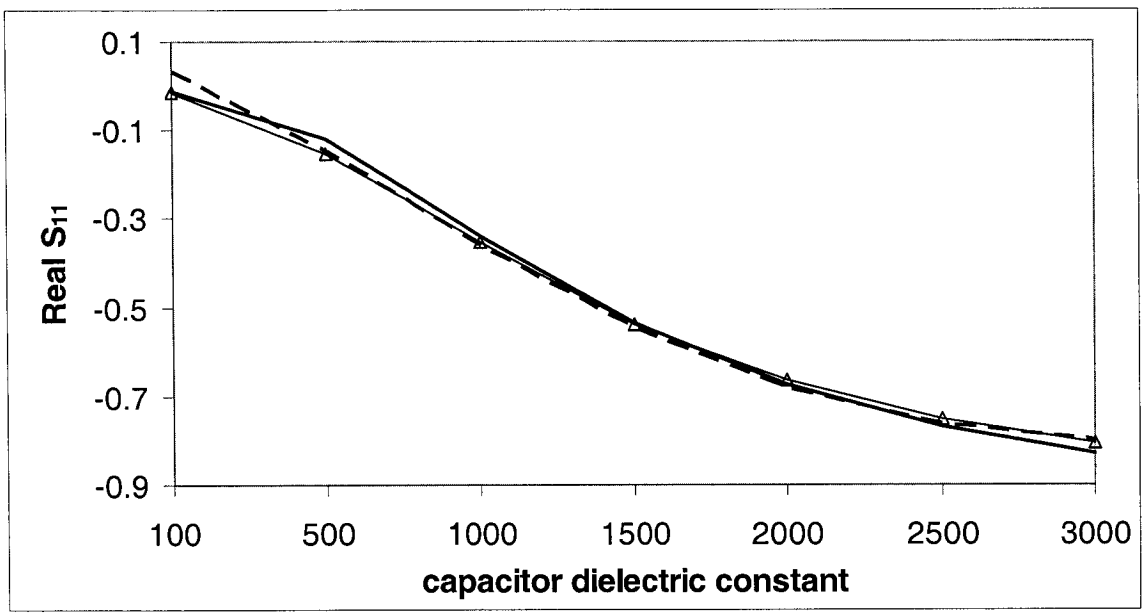


Figure 5-7. Capacitor: Real part of S_{11}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

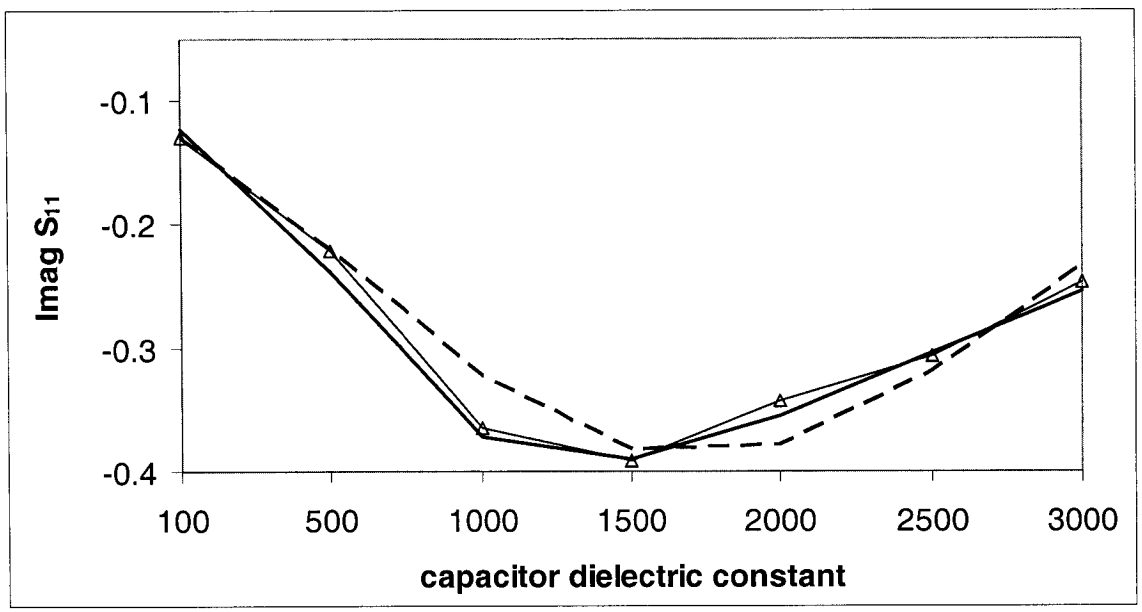


Figure 5-8. Capacitor: Imaginary part of S_{11}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

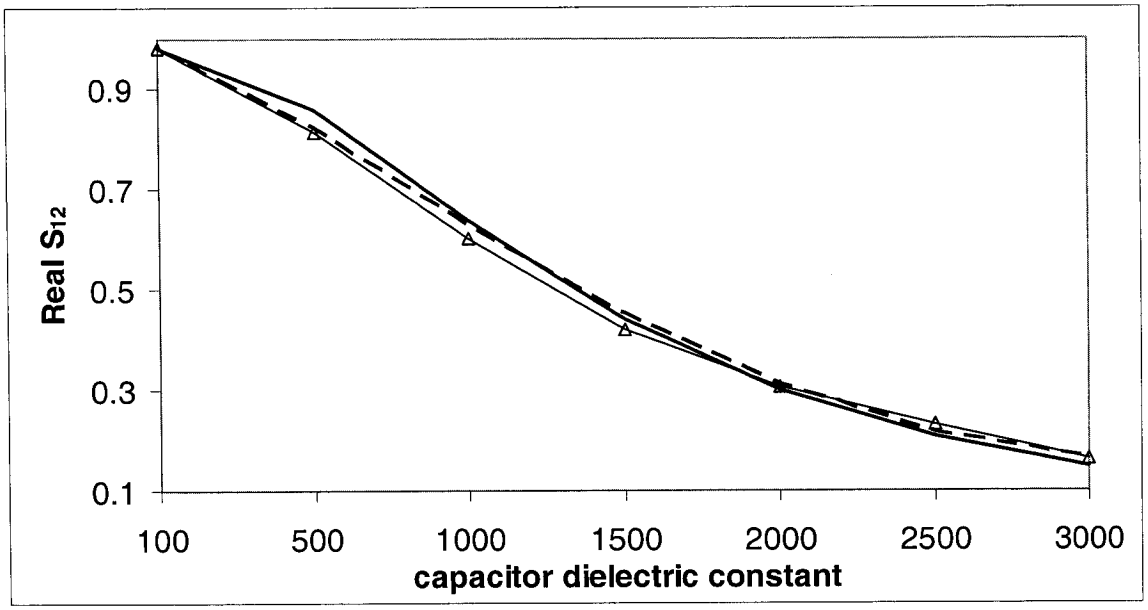


Figure 5-9. Capacitor: Real part of S_{12}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

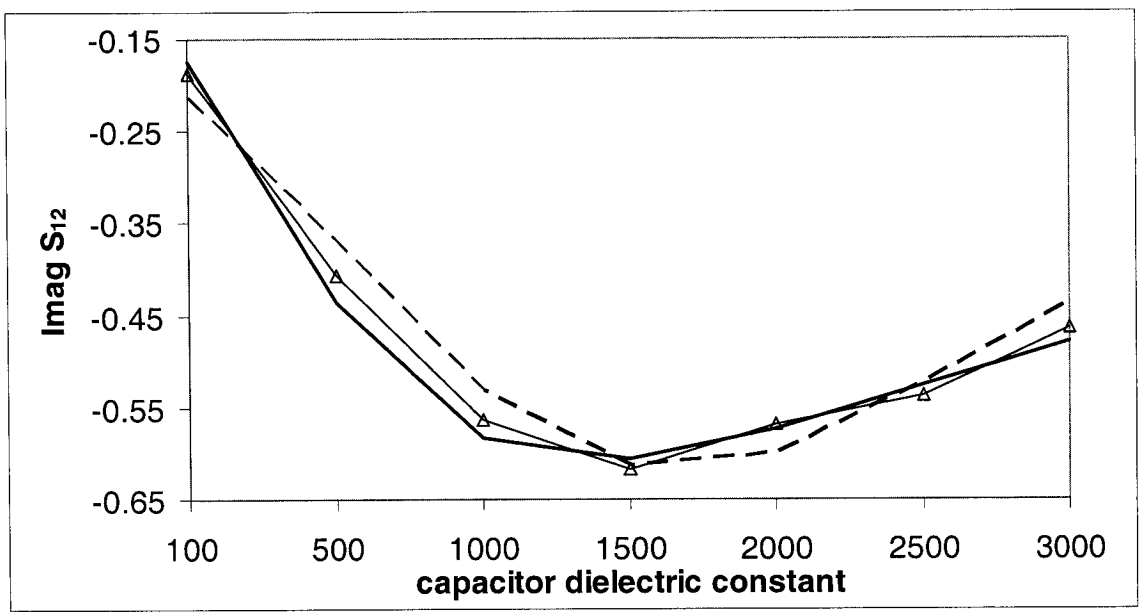


Figure 5-10. Capacitor: Imaginary part of S_{12}, comparing the results of MLP (--), KBNN (-Δ-) and the original data from EM simulator (-)

5.3 Square-spiral Inductor Modeling Using MLP and PKI

The demands placed on wireless communication circuits include low supply voltage, low cost, low power dissipation, low noise, high operation frequency and low distortion. These design requirements cannot be met satisfactorily in many cases without the use of RF inductors. Consequently, planar spiral inductors have become essential elements of communication circuit blocks such as voltage controlled oscillators (VCO), low-noise amplifiers (LNA), mixers, and intermediate frequency filters (IFF) [28]. As such, considerable effort has been put into the modeling of planar inductors. For the ease of the layout, square spirals become the most popular ones among all the planar inductors.

In this example, we used neural network to model a 10 nH Si IC square spiral inductor based on the design of T. H. Bui [29]. The geometrical values of the square-spiral inductor are shown in table 5-5, and its layout is shown in figure 5-11. Both MLP and PKI neural networks are applied. The results are compared at the end.

Table 5-5. Square-spiral inductor**:** Geometric values

Thickness t (μm)	Space between segment s (μm)	Width w (μm)	Outer length l_1 (μm)	Number of turns	Total inductance length L (nH)
1	4	6	231	7	10.2

The input parameter for MLP neural network is frequency. We sweep the frequency at the range of 0.1 to 10 GHz, at the step size of 0.1 GHz. For PKI, we use the S-parameters of the equivalent circuit of this inductor together with frequency as the input parameters.

The equivalent circuit of the inductor is shown in figure 5-12 [29]. We will use the magnitude and phase of S_{11} and S_{12} as our output parameters.

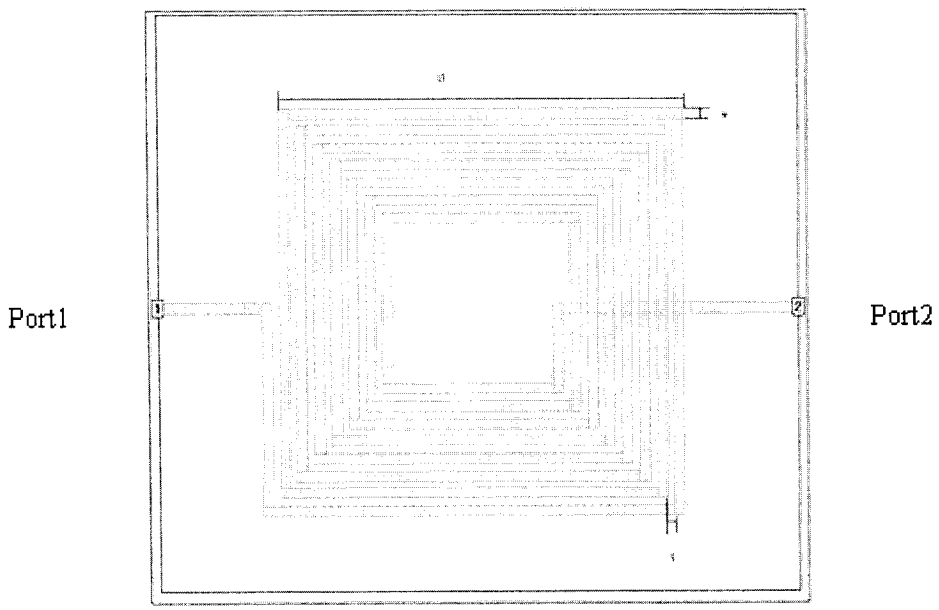


Figure 5-11. Square-spiral inductor: Layout

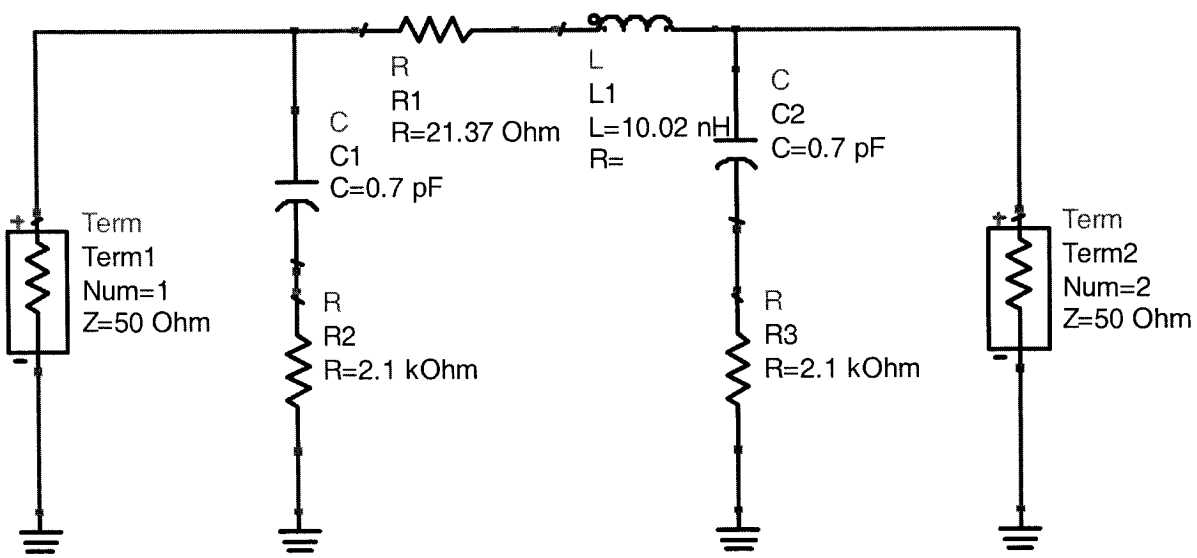


Figure 5-12. Square-spiral inductor: Equivalent circuit

Table 5-6. Square-spiral inductor: Accuracy comparison between MLP and PKI

Structure	Model size	Training sample number	Test sample number	E_{T_r}	E_{T_e}	Number of iterations
MLP	7	80	20	1.009%	0.896%	352
PKI	7			0.569%	0.563%	173
MLP	8	67	33	1.268%	1.166%	283
PKI	7			0.792%	0.770%	194

From table 5-6, we can draw the conclusion that for an acceptable accuracy, MLP may need more hidden neurons, e.g. MLP with 67 training data needs 8 hidden neurons to get a training error of 1.268%. Secondly, PKI can achieve higher accuracy even with less training samples, e.g. the training error of PKI is 0.792% with 67 training samples, while that of MLP is 1.009% with 80 training samples. Thirdly, due to the one to one mapping of the PKI, it always needs fewer iterations than MLP to achieve a similar accuracy.

The model accuracy comparison of MLP and PKI is shown in figures 5-13 to 5-16. The results show that, as expected, PKI can match the original data better than MLP.

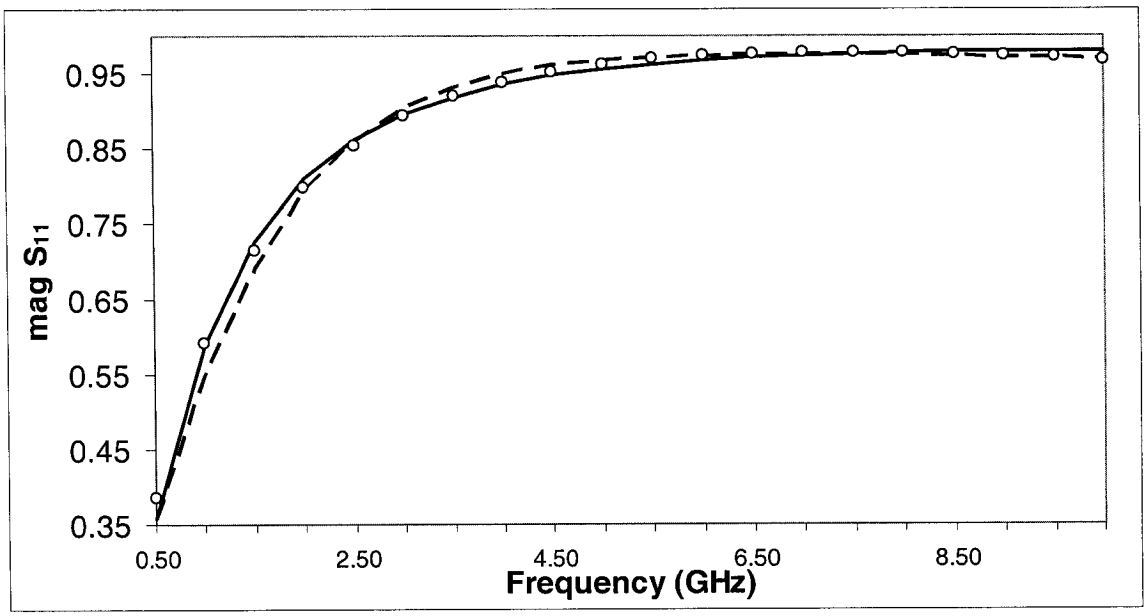


Figure 5-13. Square-spiral inductor: Magnitude of S_{11}, comparing the results of MLP(--), PKI (◦) and the original data from EM simulator (-)

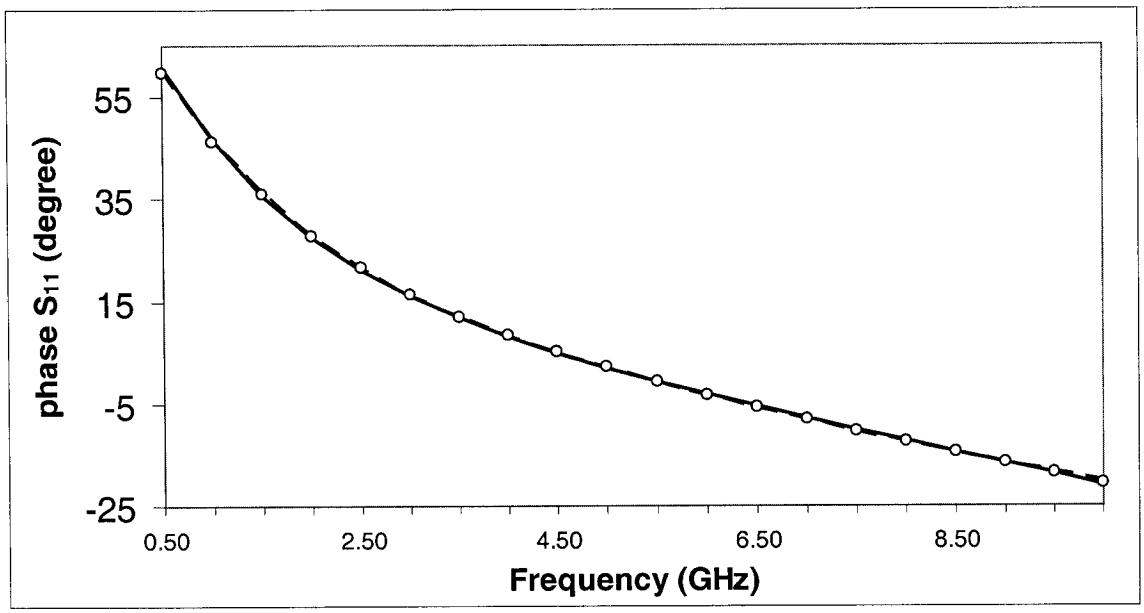


Figure 5-14. Square-spiral inductor: Phase of S_{11}, comparing the results output of MLP (--), PKI (◦) and the original data from EM simulator (-)

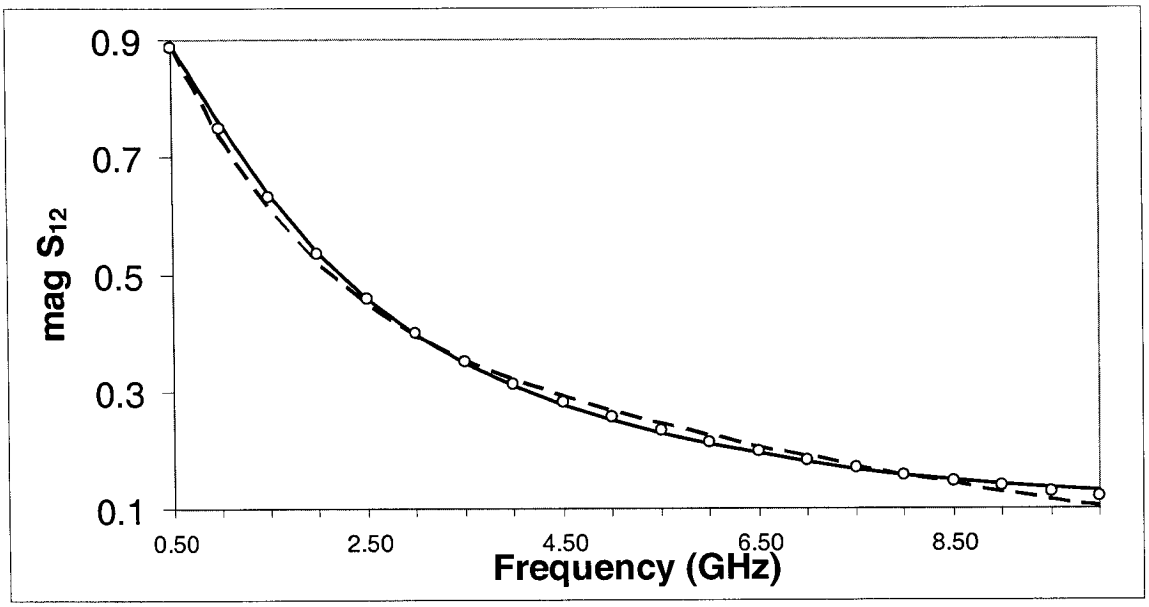


Figure 5-15. Square-spiral inductor: Magnitude of S_{12}, comparing the results of MLP(--), PKI (◦) and the original data from EM simulator (-)

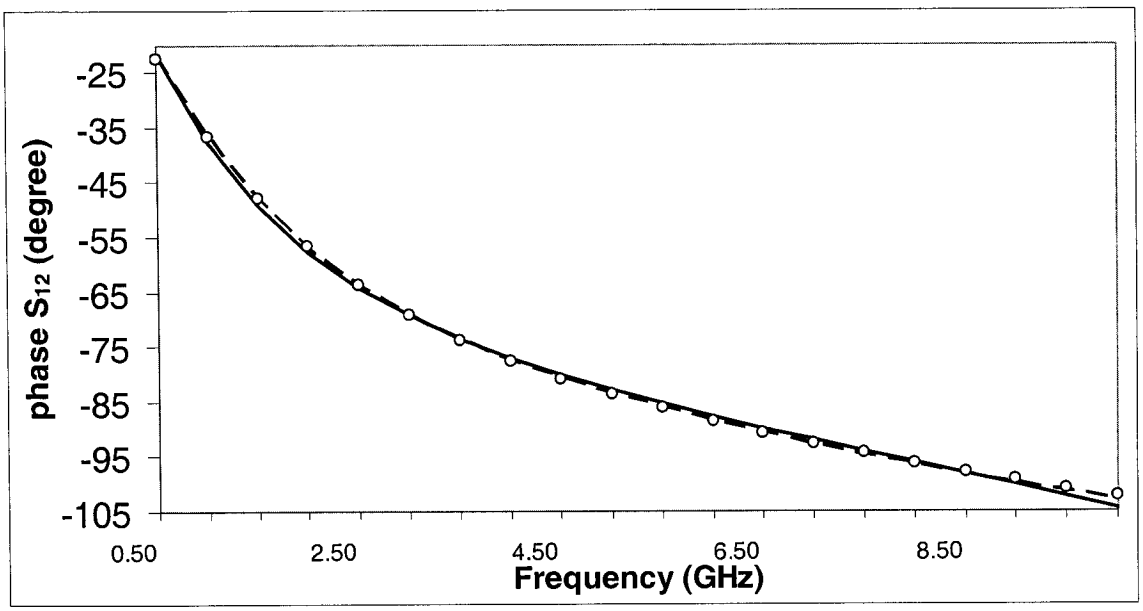


Figure 5-16. Square-spiral inductor: Phase of S_{12}, comparing the results output of MLP (--), PKI (◦) and the original data from EM simulator (-)

5.4 FET Modeling Using MLP and PKI

This information age leads to the development of new and more complex models for active devices such as Field Effect Transistors (FETs). The complexity of the model and the element values of the model mainly depend on the operating frequency and DC bias levels [30]. Based on the most widely used FET topology (here referred to as the standard topology) [31] and [32], as shown in figure 5-17, more complex and accurate topologies have been proposed in recent years for different FETs [33], [34] and [35].

In this example, we want to model an AlGaAs/InGaAs-GaAs PHEMT with bias point of $V_{gs} = 0.3\,V$. $V_{ds} = 2\,V$ [36]. According to the work of L. Ji [37], a better topology is chosen and the parameters are extracted.

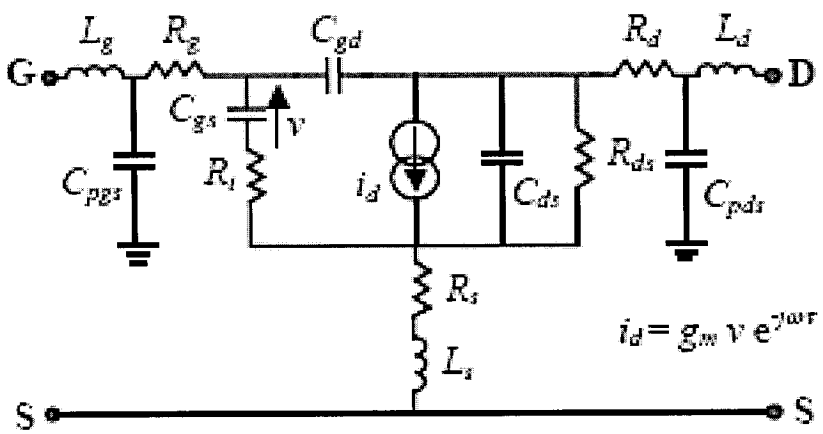


Figure 5-17. FET: Standard topology of the equivalent circuit

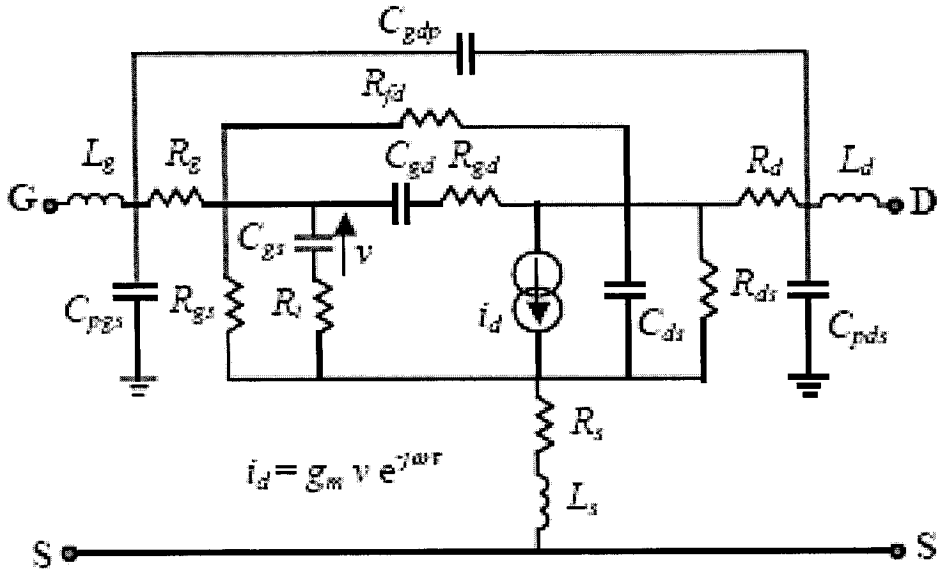


Figure 5-18. FET: Chosen topology of the equivalent circuit

Both of MLP and PKI will be applied in this example. We also use frequency swept in the range of 3 to 18 GHz with a step size of 0.1 GHz as the input parameter for MLP. The outputs of MLP are the S-parameters of the device. As to PKI, we use the standard topology as the coarse model. Then it has 9 input parameters which are the frequency, magnitude and phase of S_{11}, S_{12}, S_{21} and S_{22} from the coarse model.

The final results are shown in table 5-7. We can come to the conclusion that with the same number of training data, PKI can get better accuracy in comparison with MLP. Furthermore, when the test data is beyond the training range, PKI would offer much better accuracy compared to MLP. In this situation, the empirical part of PKI, which is the coarse model in this example, plays a very important role. Due to its help, PKI could give more accurate output value when the input parameter is beyond the training range. We can see this more clearly from table 5-8. The model accuracy comparison of MLP

and PKI is shown in figures 5-19 to 5-26. Both of them match the original data from ADS well.

Table 5-7. FET: Accuracy comparison between MLP and PKI

structure	Model size	E_{T_r}	E_{T_e} within training range	E_{T_e} beyond training range	Number of iterations
MLP	5	0.864%	0.845%	2.9313%	287
PKI	5	0.599%	0.595%	0.598%	145

Table 5-8. FET: Test result comparison between MLP and PKI, when input parameter is beyond training range

structure	Input (GHz)	s_{11} (*mag / angle*)	s_{12} (*mag / angle*)	s_{21} (*mag / angle*)	s_{22} (*mag / angle*)
ADS	1	0.9980/ -5.9260	0.0139/ 85.9210	4.6990 / 175.1570	0.6980/ -4.7180
MLP	1	1.0091/ -13.4695	0.0284/ 80.3813	4.7583/ 169.9567	0.7056/ -10.7118
PKI	1	0.9995/ -8.4572	0.0139/ 84.4630	4.7175/ 174.2520	0.7014/ -6.5501

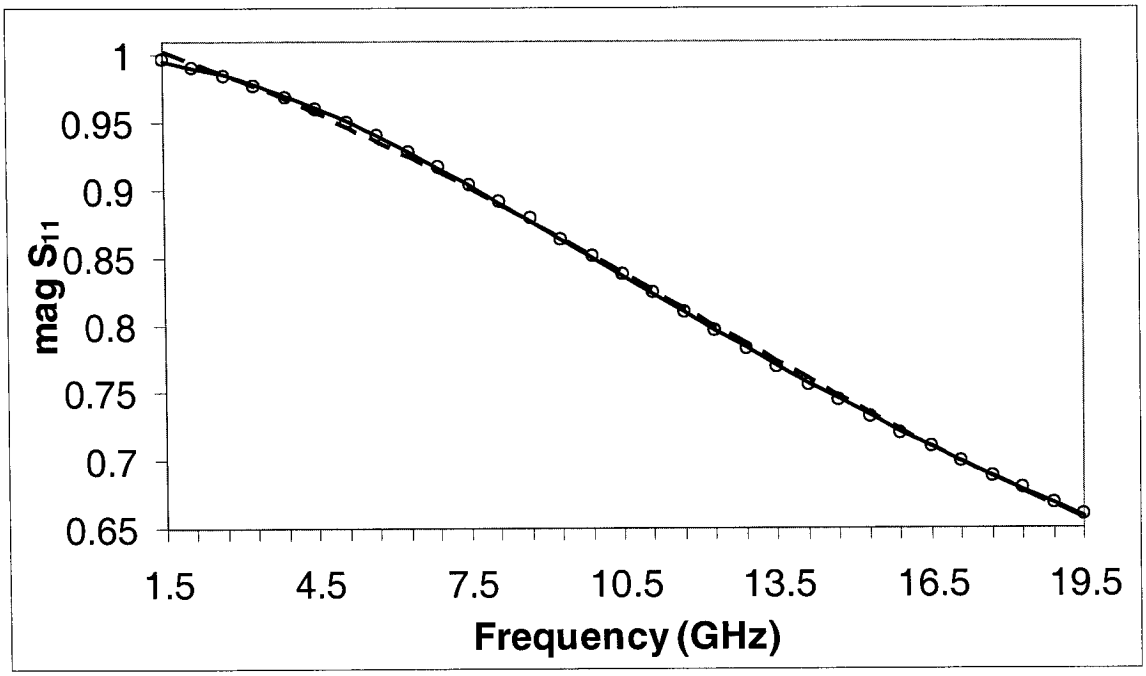


Figure 5-19. FET: Magnitude of S_{11}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

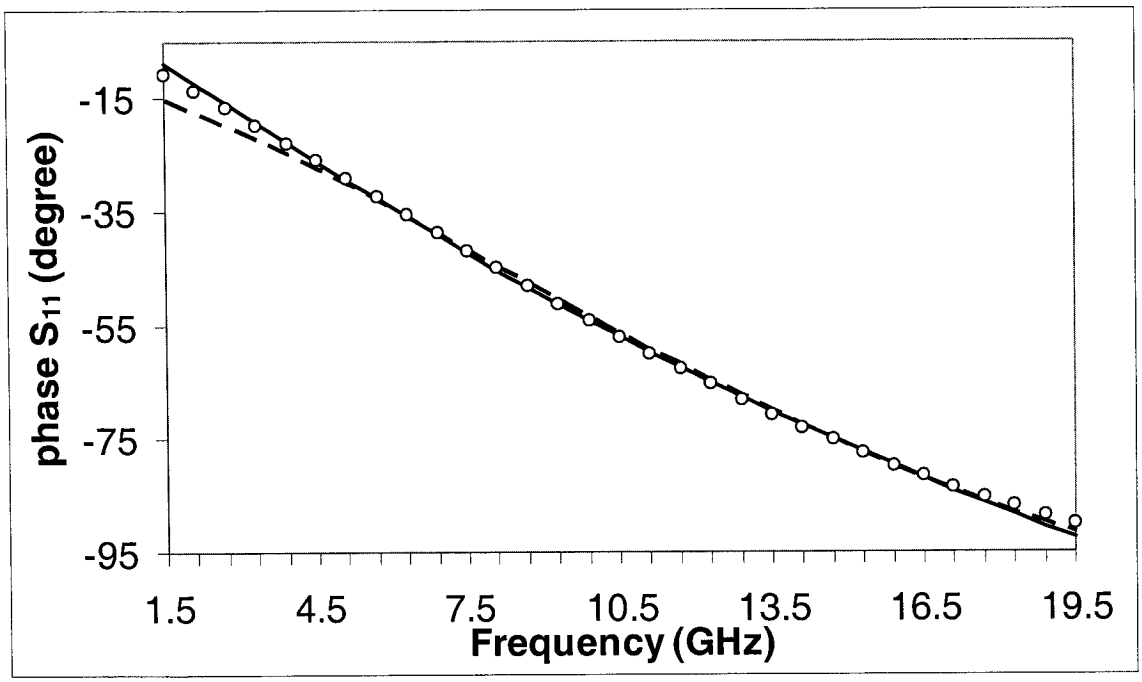


Figure 5-20. FET: Phase of S_{11}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

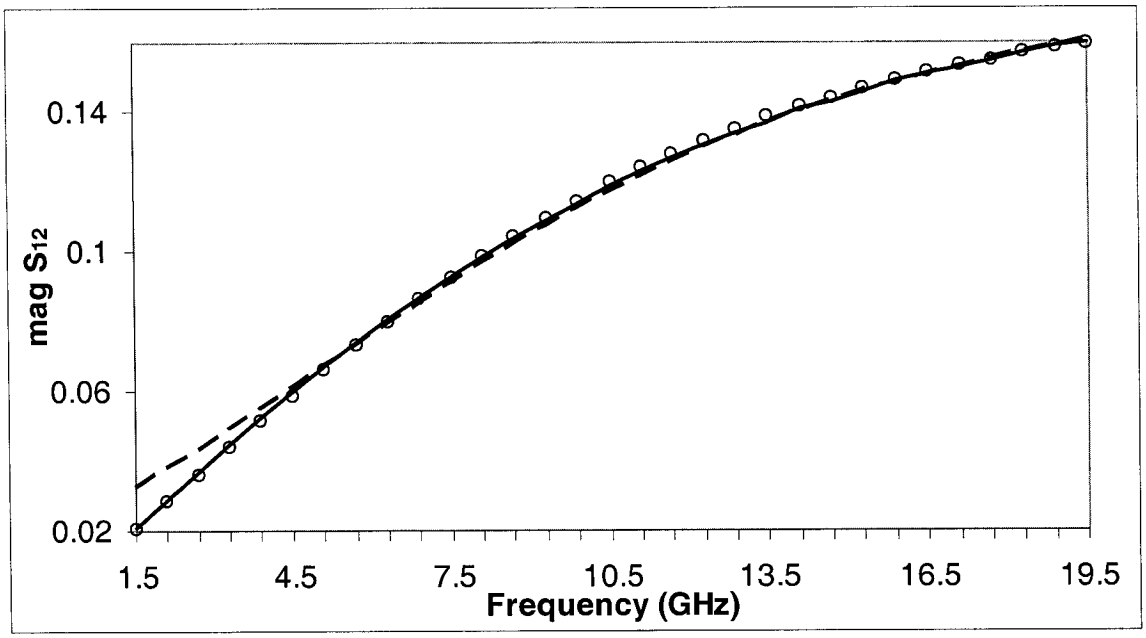


Figure 5-21. FET: Magnitude of S_{12}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

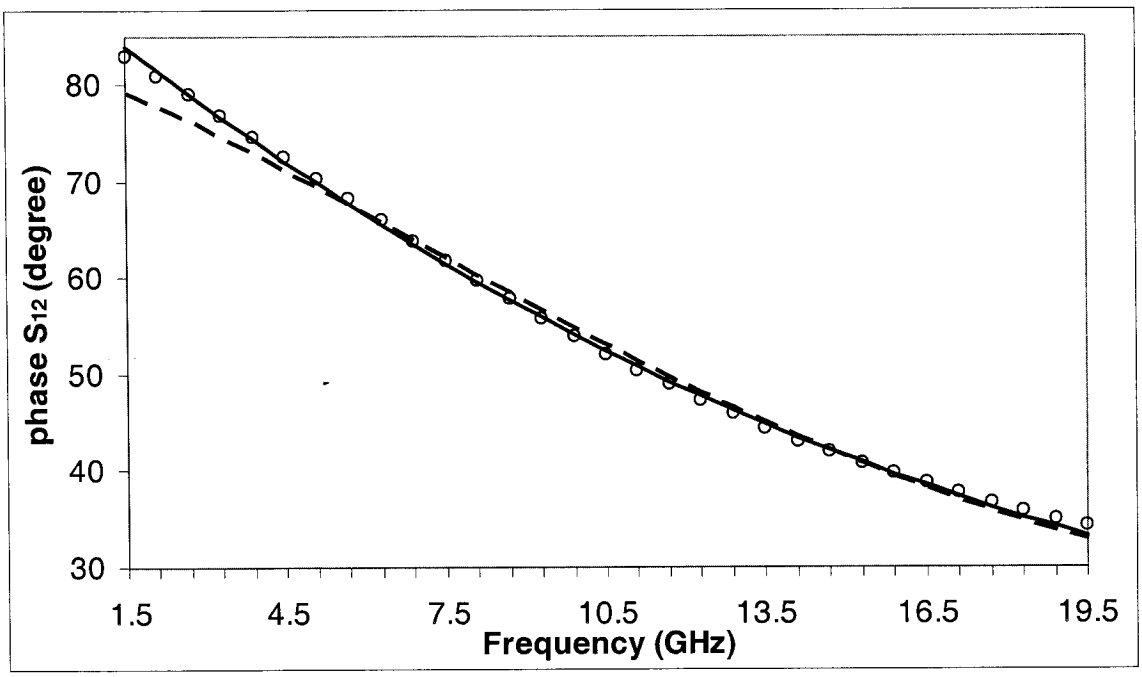


Figure 5-22. FET: Phase of S_{12}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

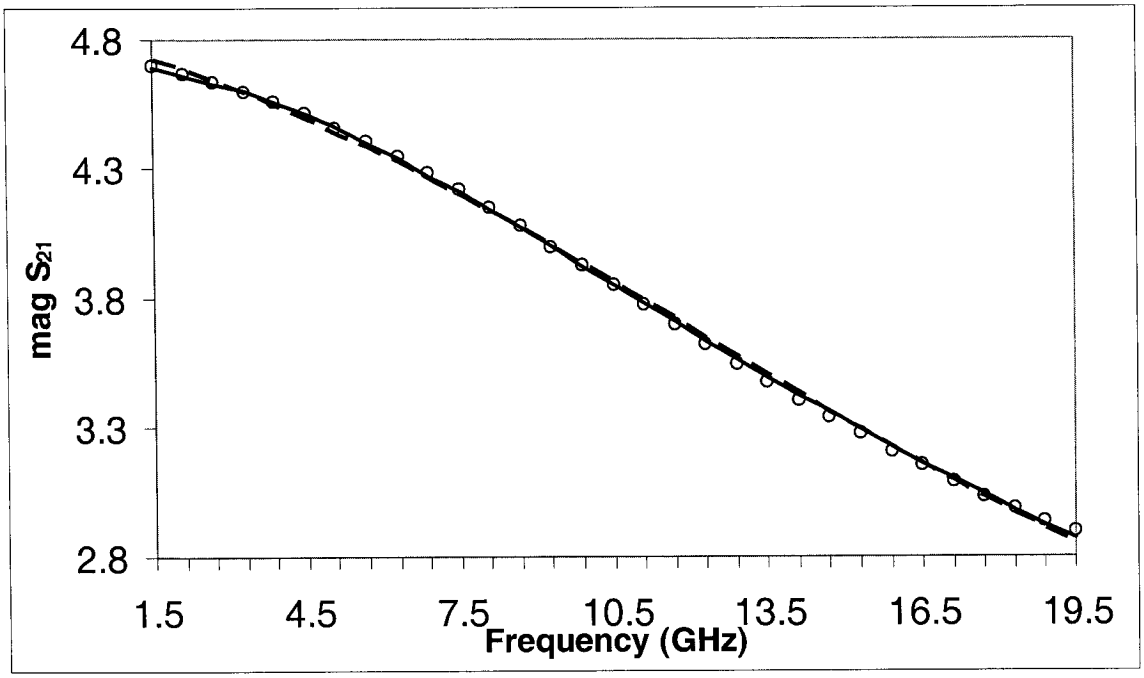


Figure 5-23. FET: Magnitude of S_{21}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

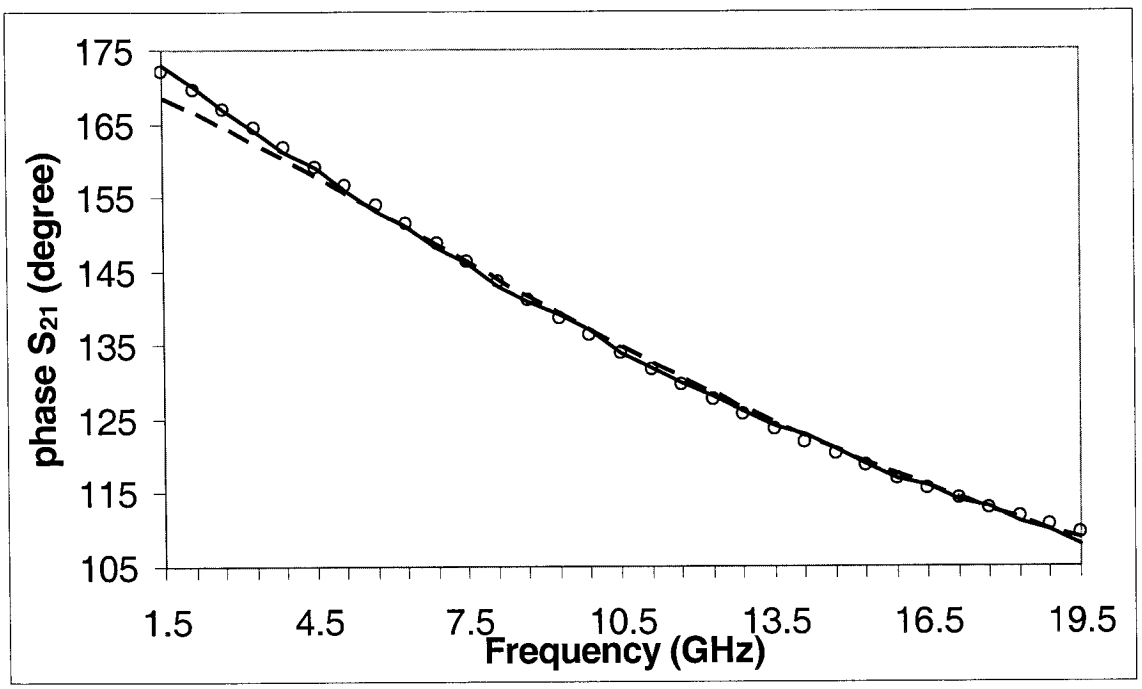


Figure 5-24. FET: Phase of S_{21}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

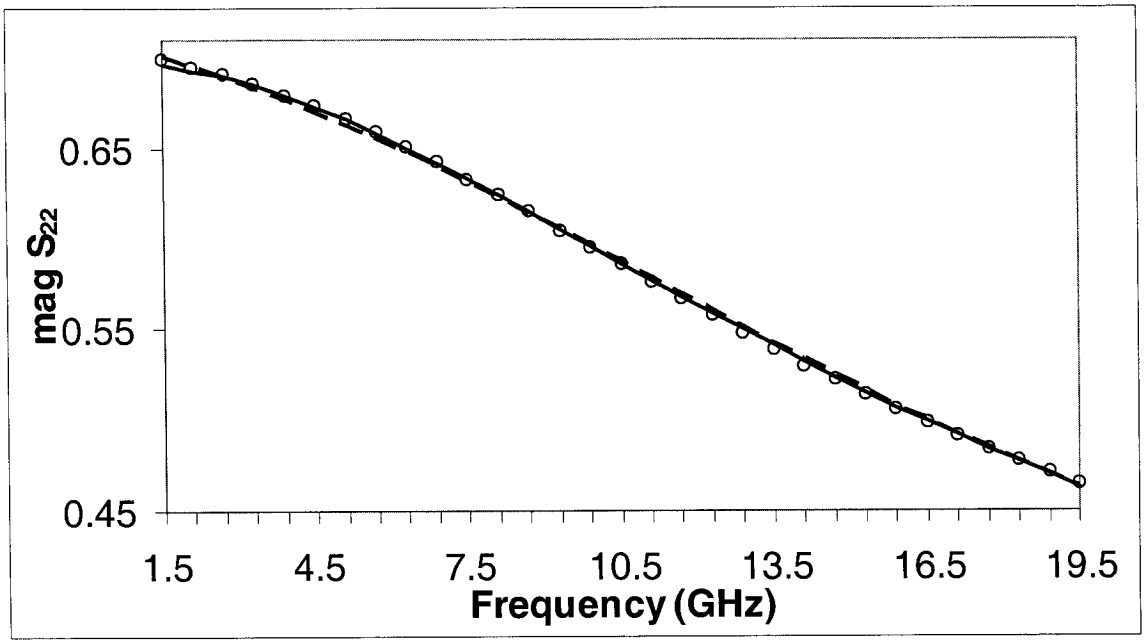


Figure 5-25. FET: Magnitude of S_{22}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

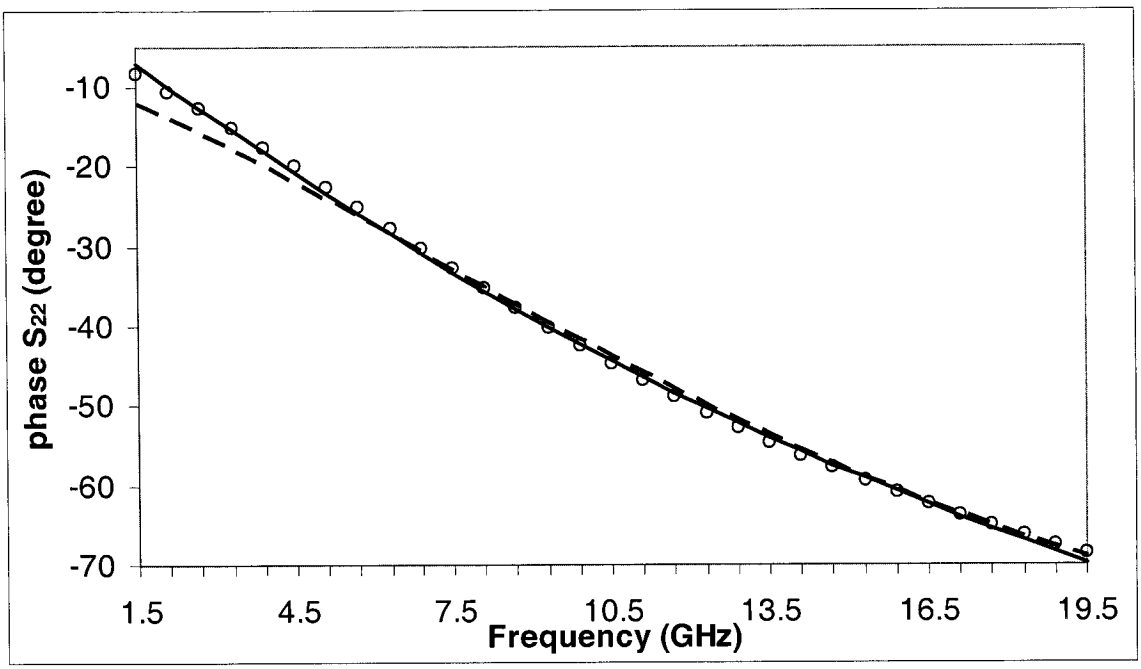


Figure 5-26. FET: Phase of S_{22}, comparing the results of MLP (--), PKI (◦) and the original data from ADS (-)

5.5 Mixer Modeling Using MLP and PKI

This new example shows the capability of neural networks in modeling circuit behavior. A sensitive way to evaluate the large signal behavior of a mixer is to apply two or more signals to the input. These dual or multiple signals (tones) will mix together and form intermodulation products [38]. However, when two or more signals with close frequencies are fed to the input port, two difference terms of third order intermodulation products are located near the input signal and so cannot be easily filtered using the pass band filter of the mixer. Figure 5-27 [39] shows a typical spectrum of the second- and third-order two-tone intermodulation products. For an arbitrary input signal consisting of many frequencies of various amplitudes and phases, the resulting in-band intermodulation products will cause distortions of the output signal.

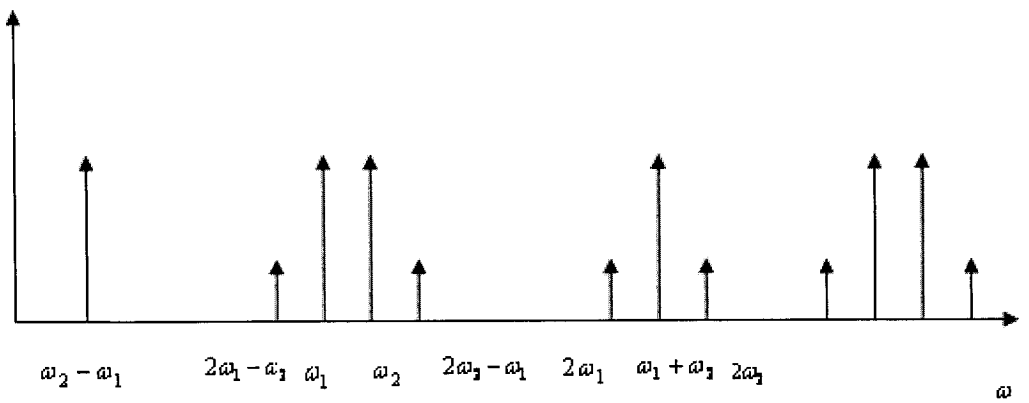


Figure 5-27. Input spectrum of second- and third-order two-tone intermodulation products, assuming $\omega_1 < \omega_2$

In this example, we will model the conversion gain of a down-converter mixer which has two RF signals centered at a certain center frequency. Between the two RF signals there is a frequency space. We swept the center frequency in the range of 1.8 – 2.2 GHz with a

step size of 0.01 GHz and the f_{space} (frequency interval between input signal frequencies) in the range of 0.5 – 100 kHz with a step size of 10 kHz. The down converter mixer circuit is shown in figure 5-28 [38].

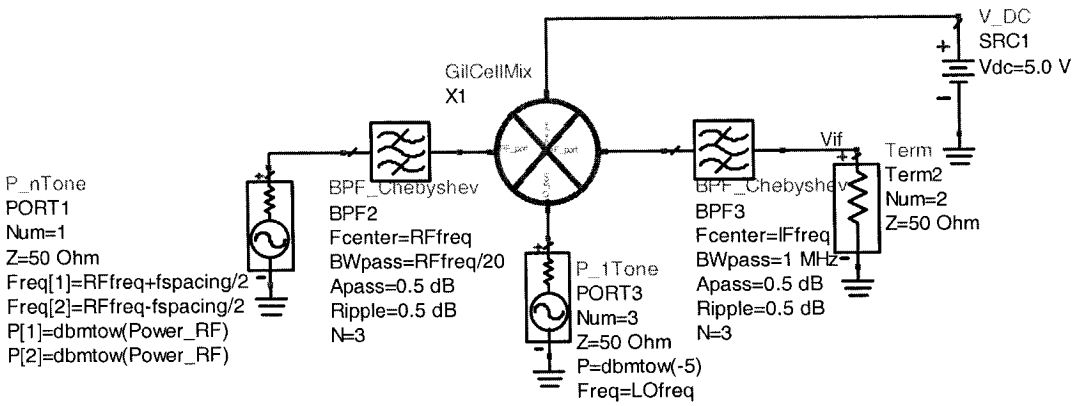


Figure 5-28. Mixer: Circuit from ADS Mixer Example

For the PKI neural network, we use the mixer with only one input signal, that is, with frequency spacing equaling to zero, as the coarse model to help the training process. Table 5-9 and 5-10 show that, compared with MLP, PKI can converge with fewer iterations and higher accuracy and when the input signal is beyond the training range, it can also give more accurate output.

Table 5-9. Mixer: Accuracy comparison between MLP and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	4	0.794%	0.792 %	289
PKI	3	0.597%	0.598%	136

Table 5-10. Mixer: Test result comparison between MLP and PKI, when input parameter is beyond training range

Structure	RF Frequency (GHz)	Conversion Gain (dB)
ADS	2.3	9.327
MLP	2.3	9.576
PKI	2.3	9.279

The model accuracy comparison of MLP and PKI is shown in figures 5-29 to 5-30. We also modeled the time domain response of the mixer, the result being shown in figure 5-31. In the highly nonlinear situation (as shown in figure 5-31), the performance of PKI is much better than that of MLP with the help of the coarse model.

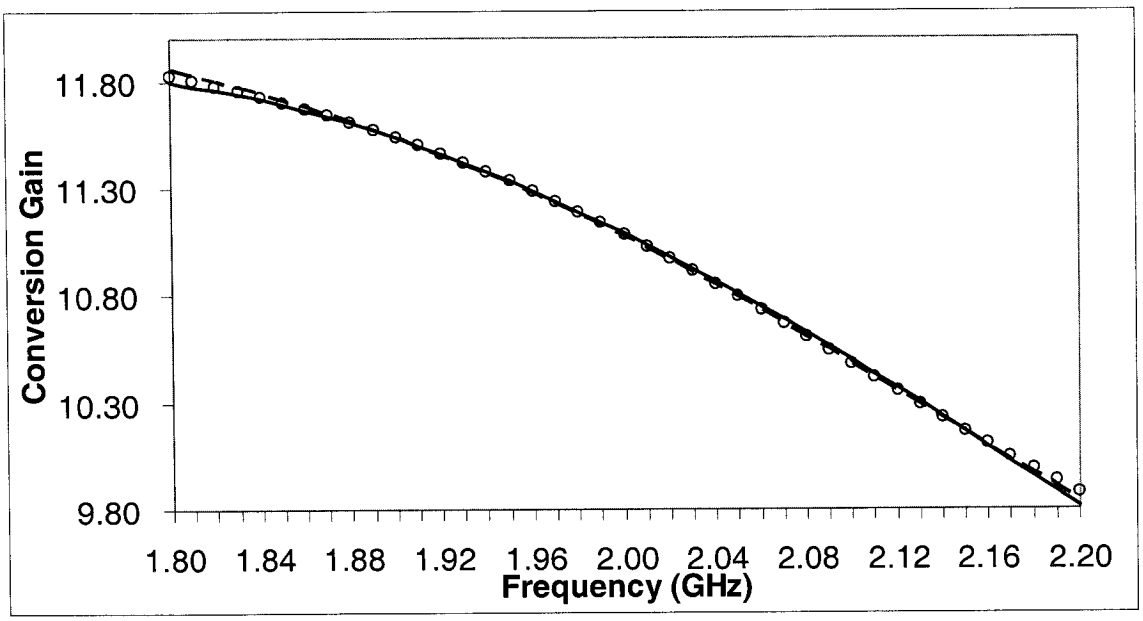


Figure 5-29. Mixer: Conversion gain VS frequency, comparing the results of MLP (--), PKI (-∘-) and the original data from ADS (-)

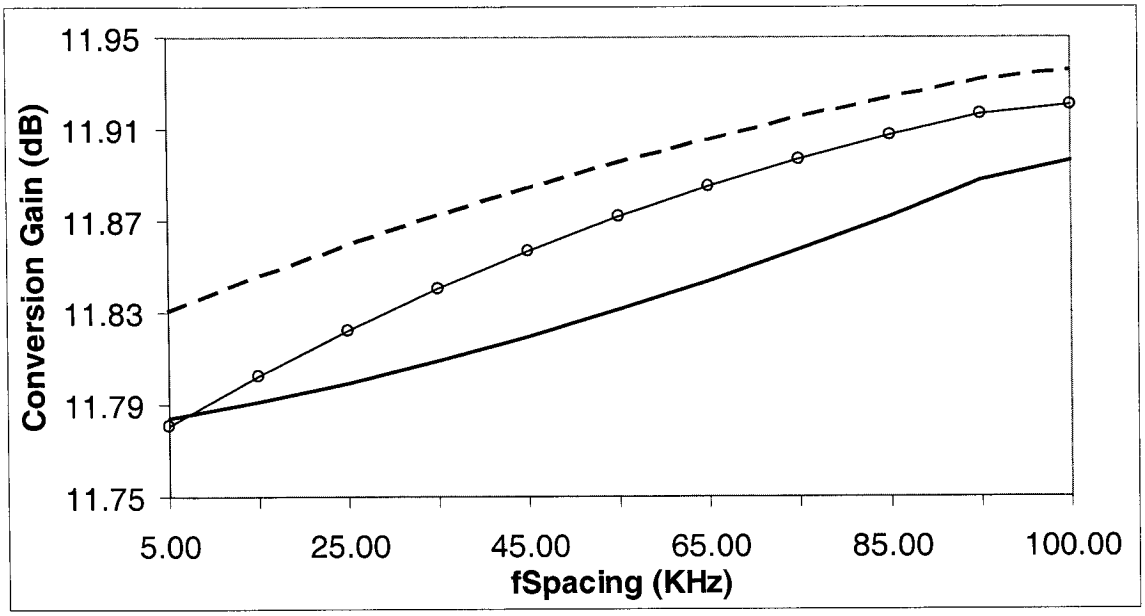


Figure 5-30. Mixer: Conversion gain VS fspacing , comparing the results of MLP (--), PKI (-∘-) and the original data from ADS (-)

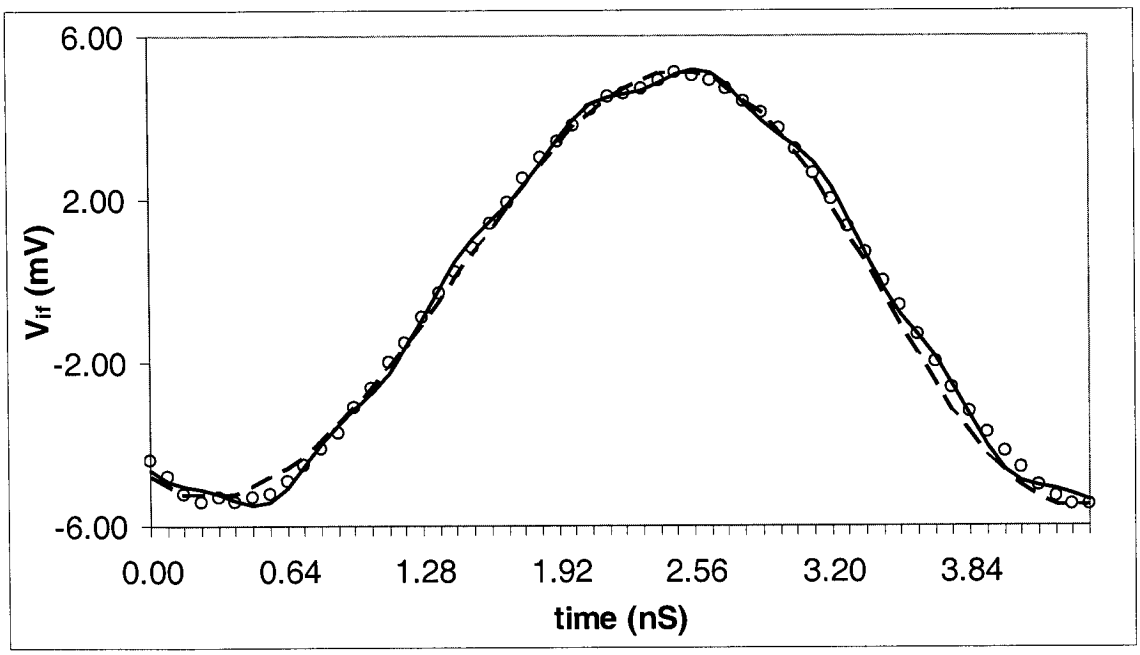


Figure 5-31. Mixer: Time domain response when RF frequency at 2.0 GHz and LO frequency at 1.75 GHz, comparing the results of MLP (--), PKI (∘) and the original data from ADS (-)

5.6 Amplifier Modeling Using MLP, KBNN and PKI

In practice, transistor amplifiers usually consist of stages connected in cascade, forming a multistage amplifier. Compared to single stage amplifiers, multistage amplifiers can provide increased input resistance, reduced output resistance, increased gain and increased power handling capability. A good multistage amplifier modeling can ease the burden of system complexity, and provide simulation speed and capacity improvement. Therefore, modeling has become the most significant issue for developing microwave multistage amplifiers.

Because only the input and final output relationships are considered, neural network modeling can simplify the modeling complexity of the system and cut short the simulation time. We will develop a neural network model for linear single stage amplifier with MLP and neural network models for linear and nonlinear amplifiers with 2 to 4 stages using MLP, KBNN and PKI respectively. We use the most basic transistor equivalent circuit to drive the empirical formula for the single stage amplifier. Then we could use this formula as the empirical formula in the KBNN modeling of multistage amplifiers. Similarly for PKI, we use the information of the single stage amplifier and the linear relationship between the single stage amplifier and the multistage amplifier to predict the nonlinear performance of the nonlinear multistage amplifiers, which could help the training process greatly.

For the neural network, input power and frequency are taken as the input parameters. Frequency is swept from 0.5 GHz to 1.2 GHz with a step size of 0.05 GHz. The sweeping

range of the input power depends on whether it is a linear amplifier or nonlinear one. The output parameters are the gains and powers at both the fundamental frequency and the second harmonics. The input and output parameters are same for all the amplifiers.

5.6.1 Single Stage Linear Amplifier with MLP

The single stage amplifier circuit is shown in figure 5-32 [40]. The transistor used in this amplifier is HP_AT41411_1_19921201 from the Package_BJT Library of ADS, at the bias point of V_{CE} being $8\,V$ and I_C being $10\,mA$.

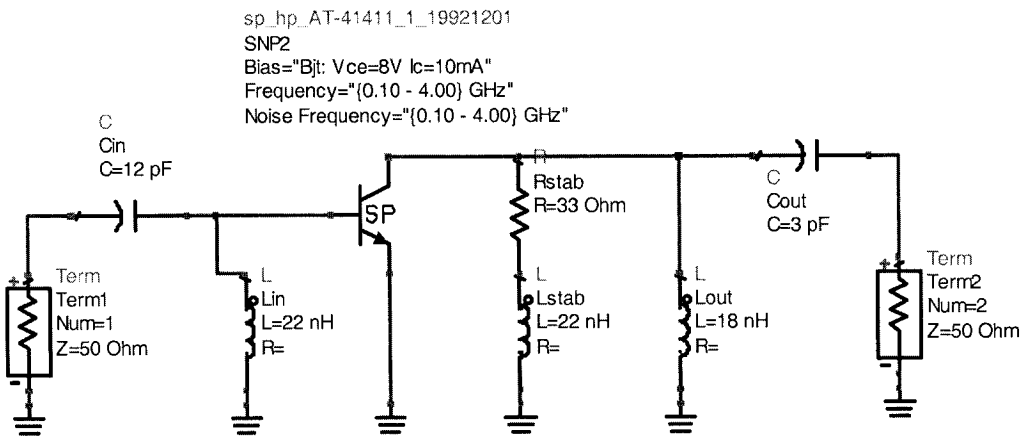


Figure 5-32. Single stage amplifier circuit from ADS Amplifier Example

The training result is shown in table 5-11.

Table 5-11. Single stage linear amplifier: Training results with MLP neural network

Number of inputs	Number of outputs	Model size	E_{T_r}	E_{T_e}	Number of iterations
2	4	8	0.497%	0.632%	328

The comparison between the original ADS simulation and MLP model is shown in figures 5-33 and 5-34. We can see the neural model matches the original data well.

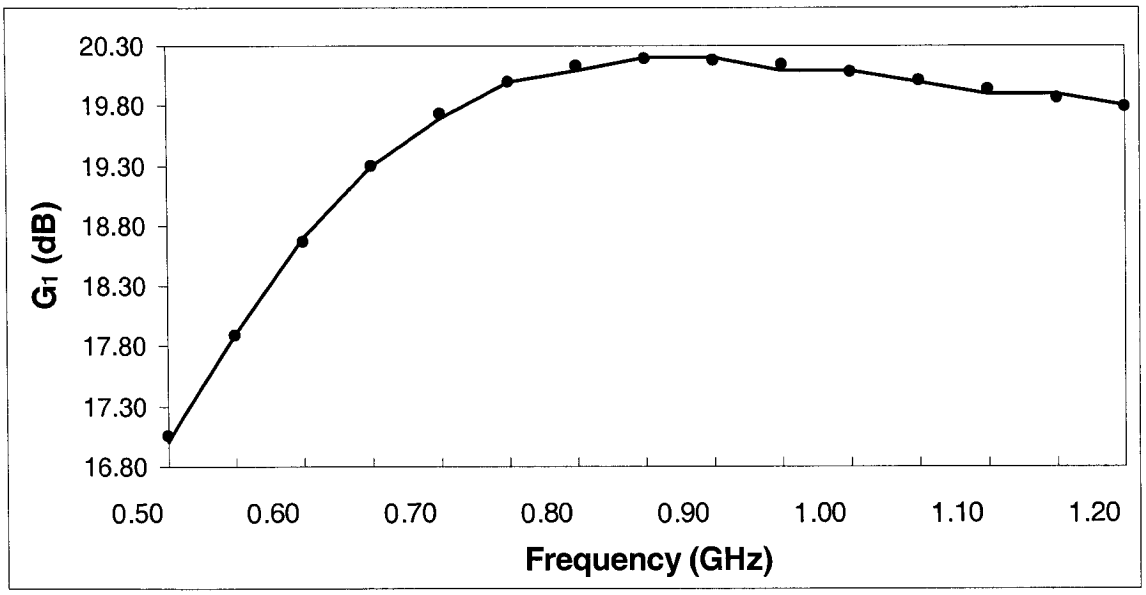


Figure 5-33. Single stage linear amplifier: Gain at fundamental frequency with -90 dBm input power, comparing the results of MLP (•) and original data from ADS (-)

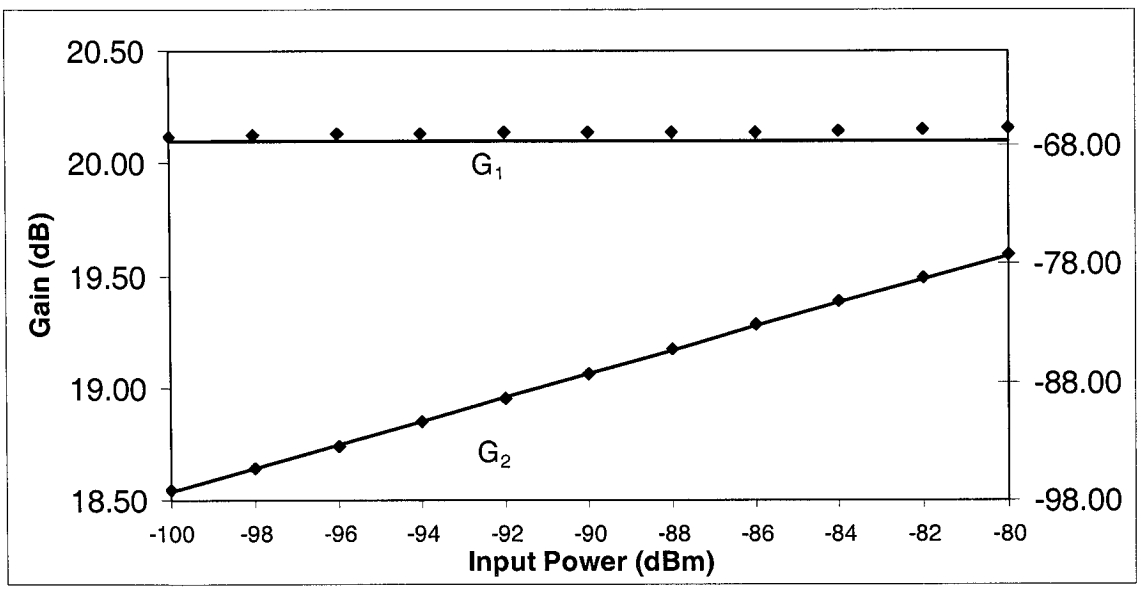


Figure 5-34. Single stage linear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (•) and original data from ADS (-)

5.6.2 Multistage Linear Amplifier with MLP, KBNN and PKI

The modeling results of the 2- to 4-stage linear amplifiers are shown in tables 5-12 to 5-14 and figures 5-35 to 5-40. When modeling the not highly nonlinear gain and output power, the result of PKI is similar to KBNN and better than MLP. However when modeling the highly nonlinear time domain response, the result of KBNN is better than PKI not to say MLP.

Table 5-12. 2-stage linear amplifier: Accuracy comparison between MLP, KBNN and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	8	1.081%	1.735%	685
KBNN	b3z4	0.749%	0.880%	297
PKI	7	0.639%	0.775%	307

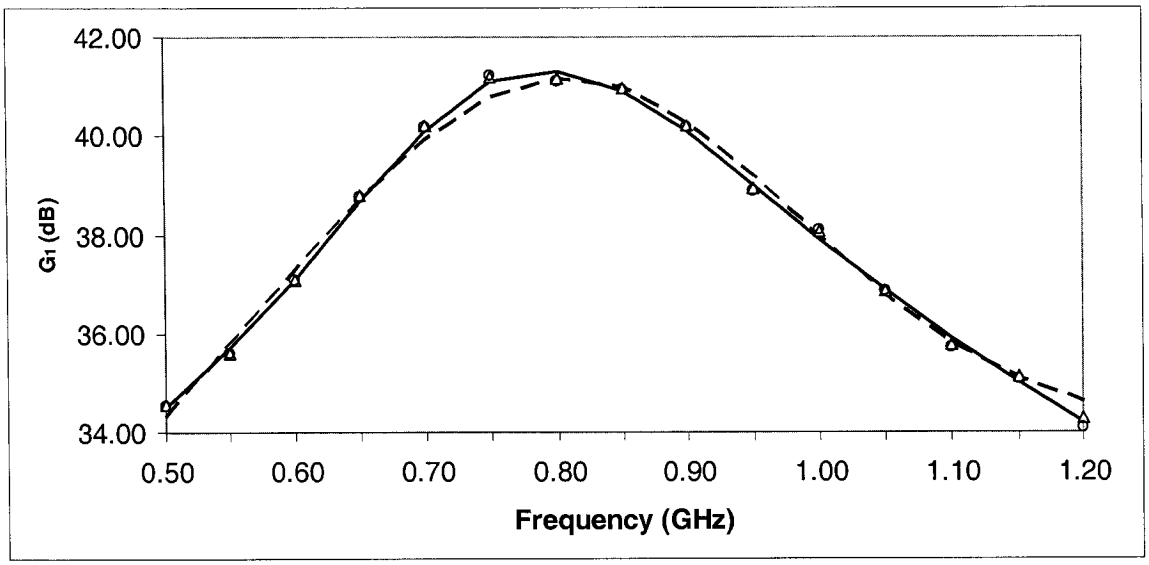


Figure 5-35. 2-stage linear amplifier: Gain at fundamental frequency with -90 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

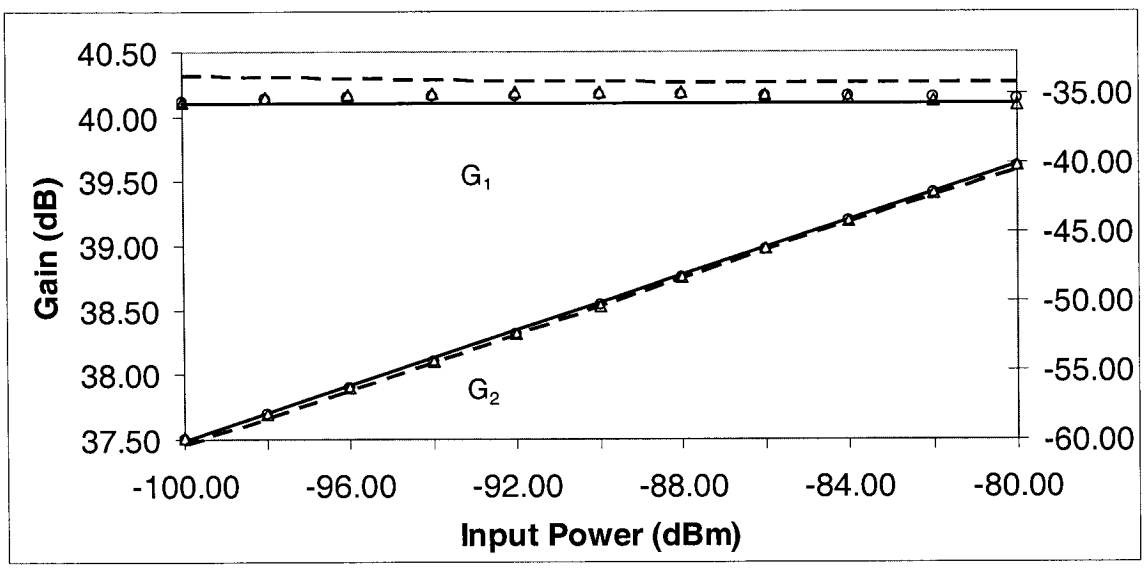


Figure 5-36. 2-stage linear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

Table 5-13. 3-stage linear amplifier: Accuracy comparison between MLP, KBNN and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	8	0.940%	1.633%	807
KBNN	b3z4	0.069%	1.116%	326
PKI	7	0.770%	1.137%	506

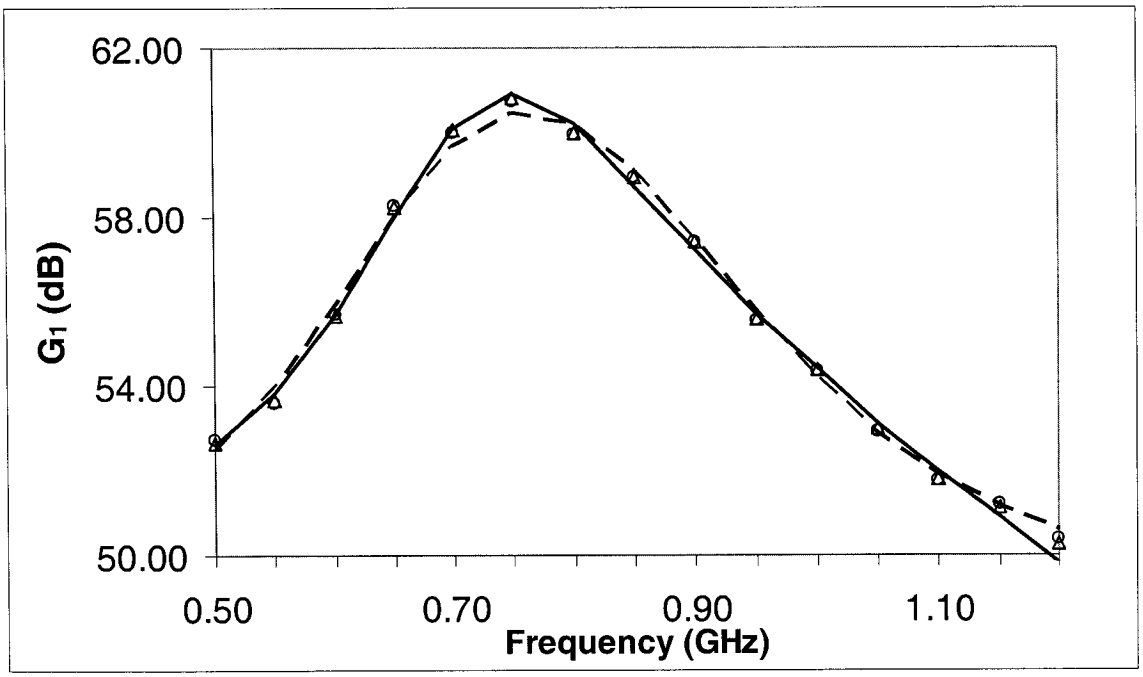


Figure 5-37. 3-stage linear amplifier: Gain at fundamental frequency with -90 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

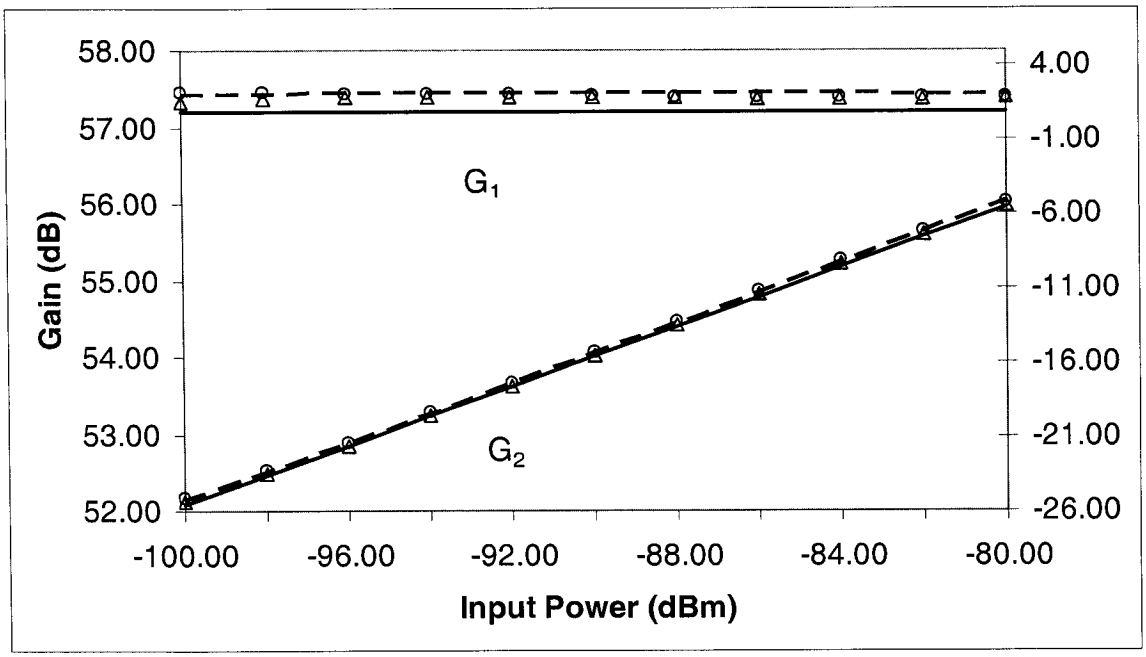


Figure 5-38. 3-stage linear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

Table 5-14. 4-stage linear amplifier: Accuracy Comparison between MLP, KBNN and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	9	0.920%	1.552%	566
KBNN	b3z4	0.702%	1.132%	359
PKI	8	0.793%	1.461%	444

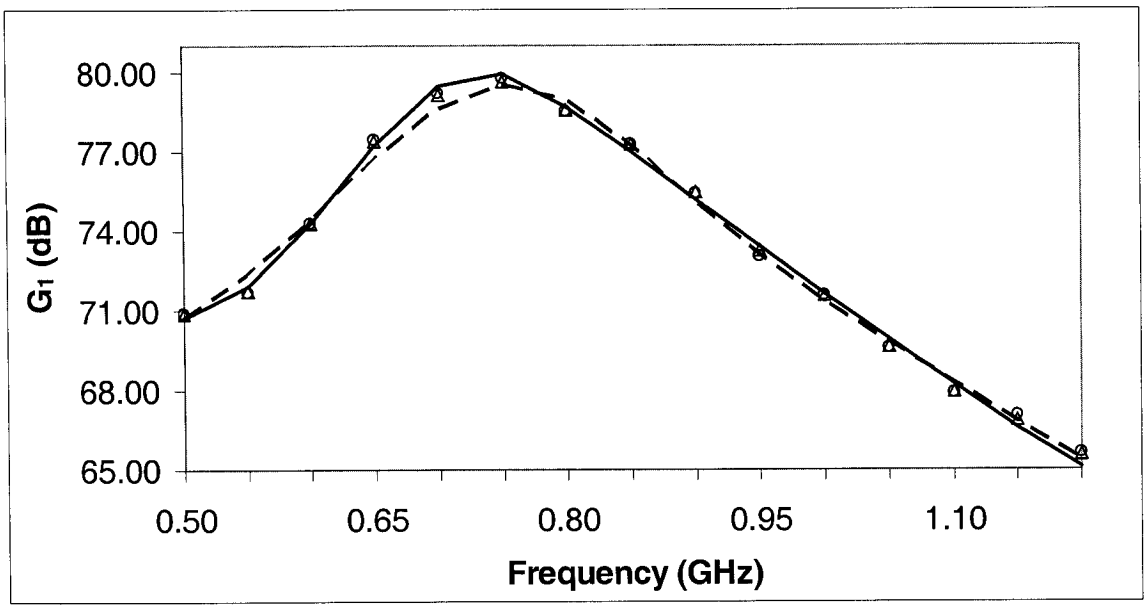


Figure 5-39. 4-stage linear amplifier: Gain at fundamental frequency with -90 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

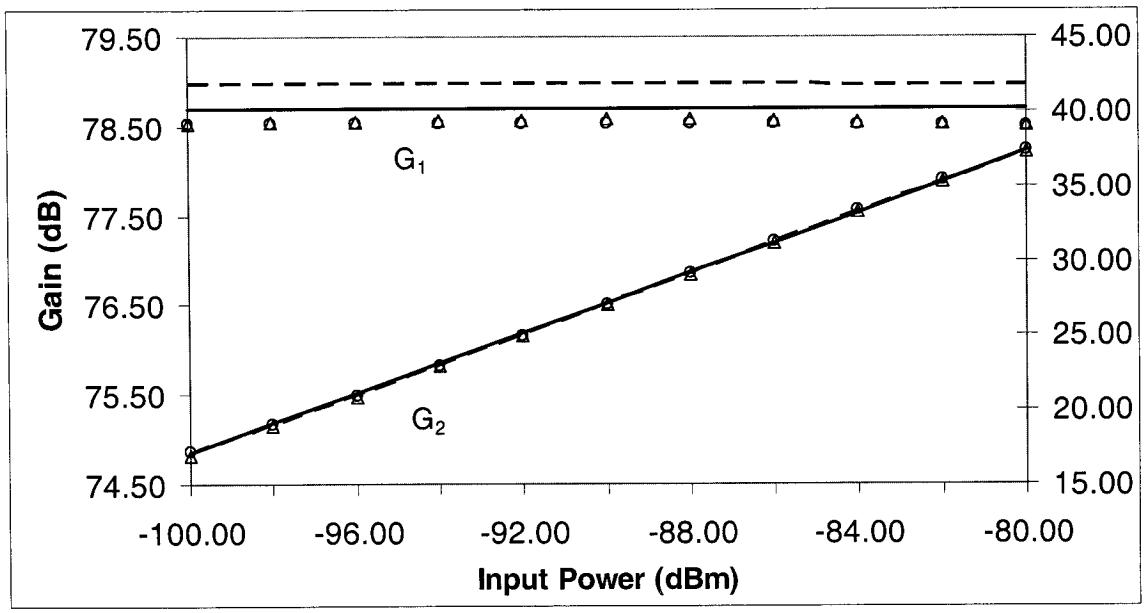


Figure 5-40. 4-stage linear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

We can see from the above results that when empirical information from the one stage amplifier is used, KBNN and PKI have a similar accuracy which is better than that of MLP. Although more calculation time and sometimes maybe more hidden neurons are needed, MLP could obtain an acceptable result (around 1%) for such linear circuits.

5.6.3 Multistage Nonlinear Amplifier with MLP, KBNN and PKI

If the amplifier response is nonlinear, the results of MLP cannot be as accurate as those of KBNN and PKI, as shown in tables 5-15 to 5-17 and in figures 5-41 to 5-49 for the 2- to 4-stage nonlinear amplifiers.

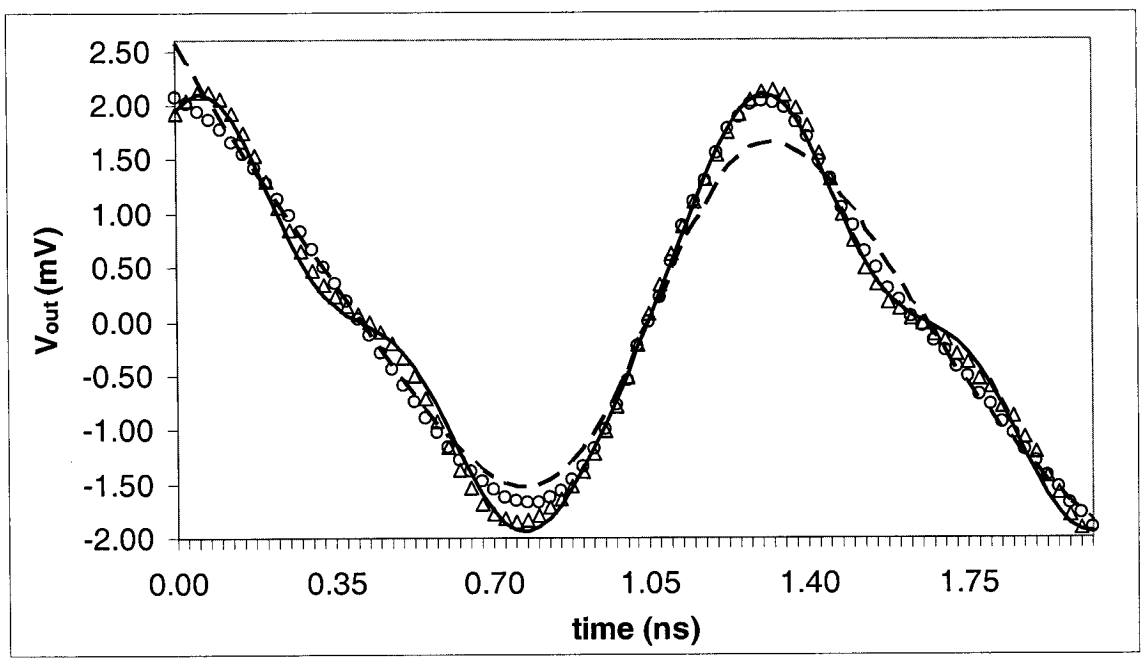


Figure 5-41. 2-stage nonlinear amplifier: Time domain response at 0.8 GHz and -20 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

Table 5-15. 2-stage nonlinear amplifier: Accuracy comparison between MLP, KBNN and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	8	1.015%	1.367%	1210
KBNN	b3z4	0.711%	0.956%	697
PKI	8	0.954%	1.186%	798

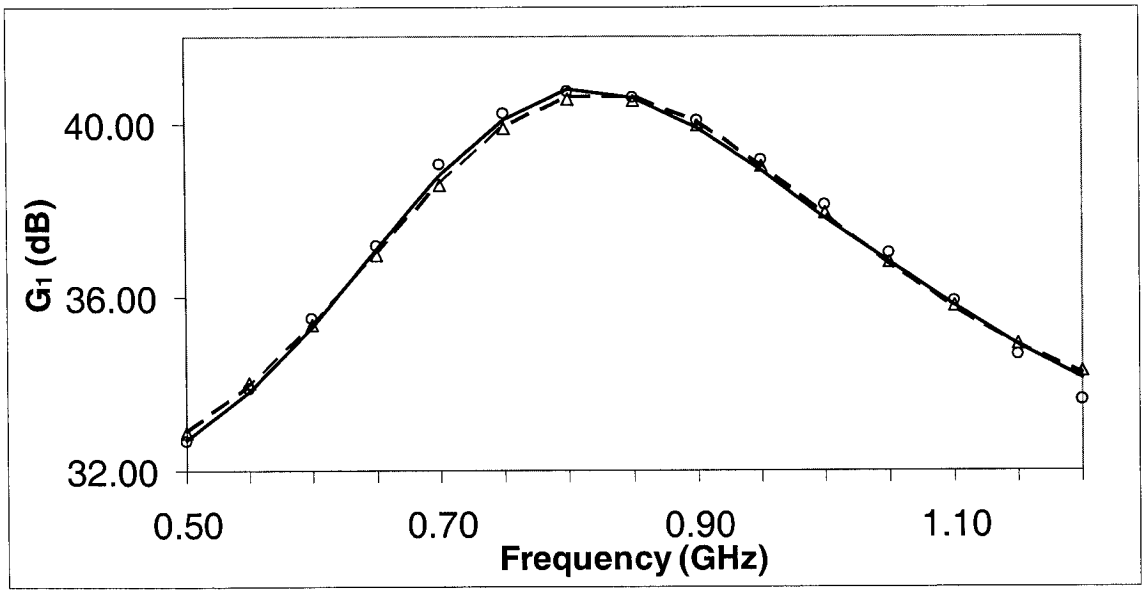


Figure 5-42. 2-stage nonlinear amplifier: Gain at fundamental frequency with -30 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

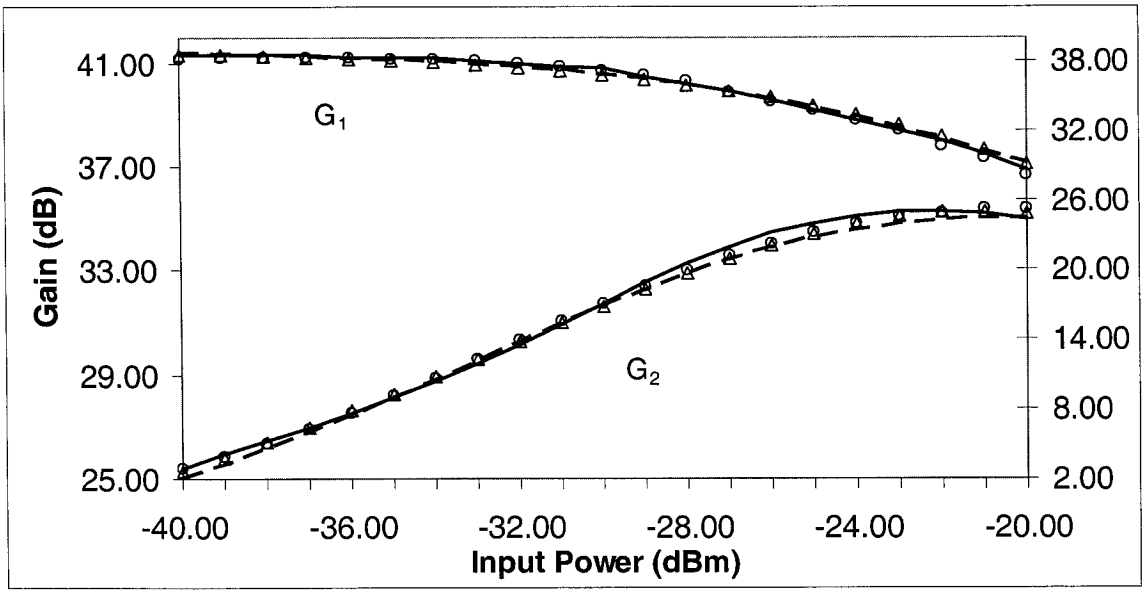


Figure 5-43. 2-stage nonlinear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

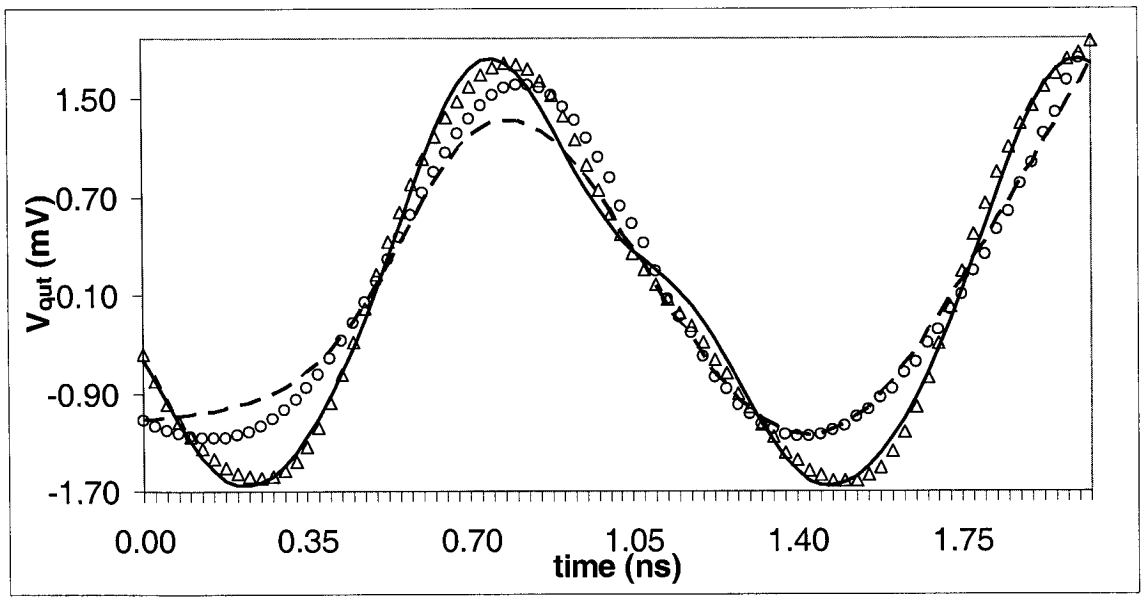


Figure 5-44. 3-stage nonlinear amplifier: Time domain response at 0.8 GHz and -40 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

Table 5-16. 3-stage nonlinear amplifier: Accuracy comparison between MLP, KBNN and PKI

Structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	8	1.113%	1.436%	1263
KBNN	b3z4	0.804%	0.885%	869
PKI	8	0.992%	0.937%	842

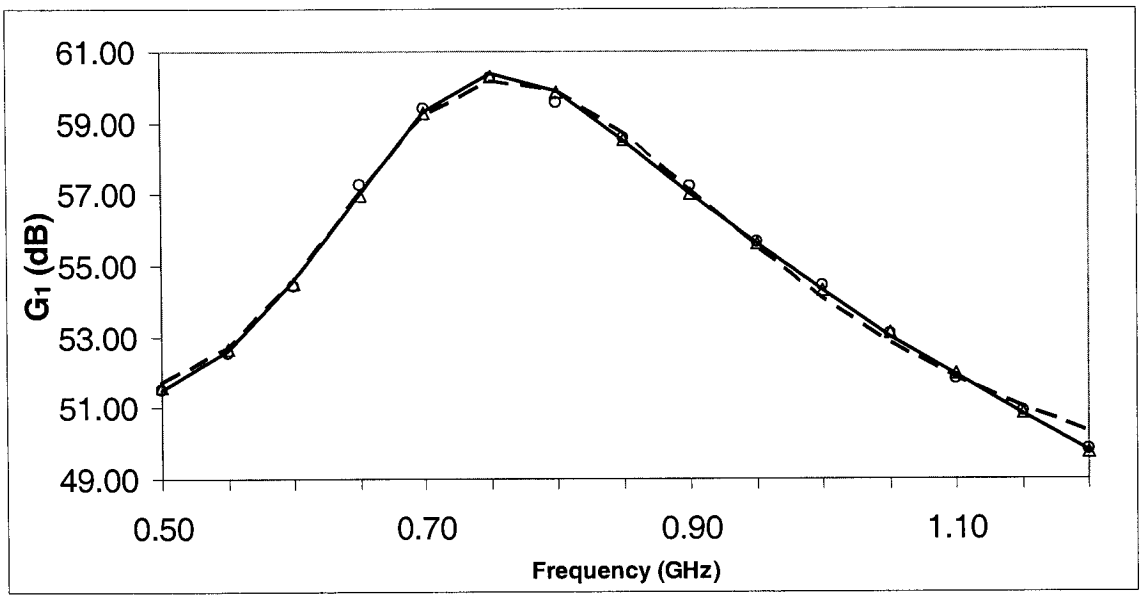


Figure 5-45. 3-stage nonlinear amplifier: Gain at fundamental frequency with -50 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

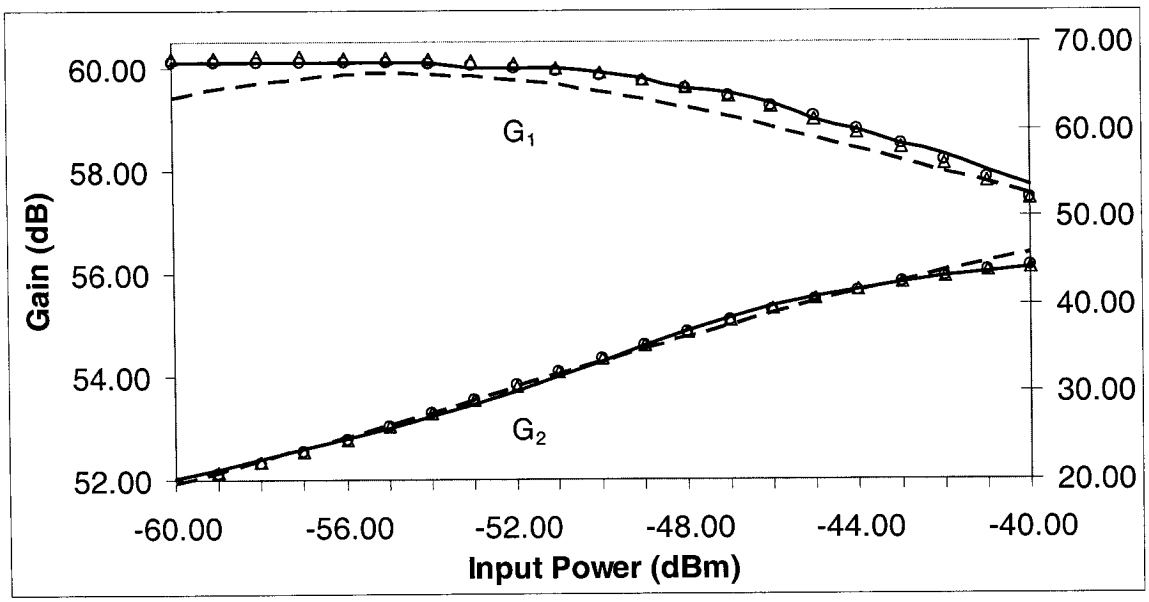


Figure 5-46. 3-stage nonlinear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

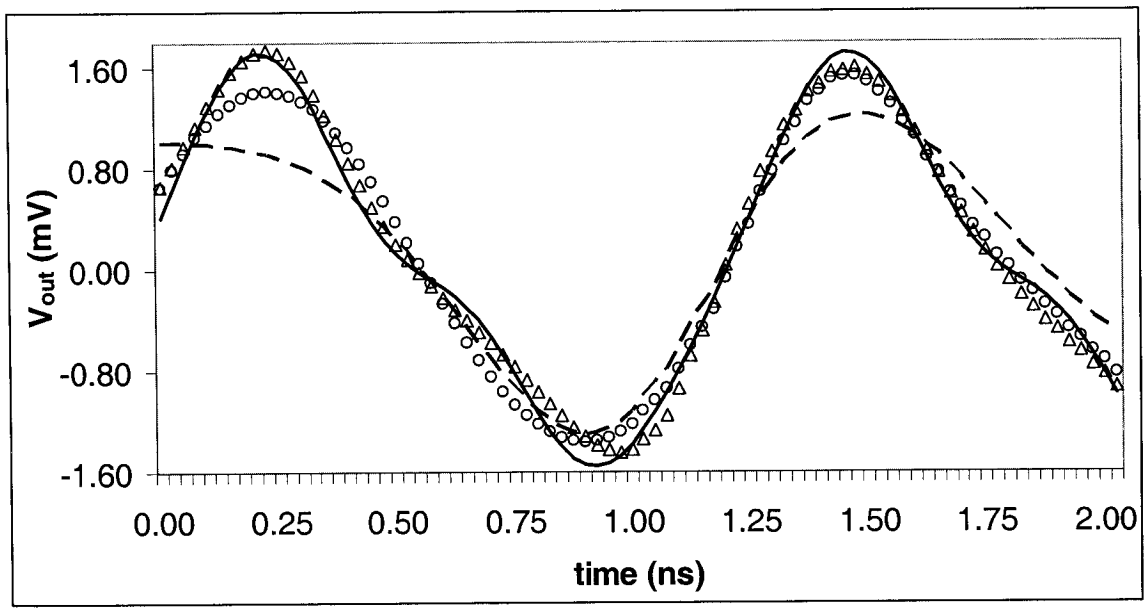


Figure 5-47. 4-stage nonlinear amplifier: Time domain response at 0.8 GHz and -60 dBm input power, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

Table 5-17. 4-stage nonlinear amplifier: Accuracy comparison between MLP, KBNN and PKI

structure	Model size	E_{T_r}	E_{T_e}	Number of iterations
MLP	8	0.966%	1.500%	1268
KBNN	b3z4	0.746%	0.880%	854
PKI	8	0.813%	0.966%	872

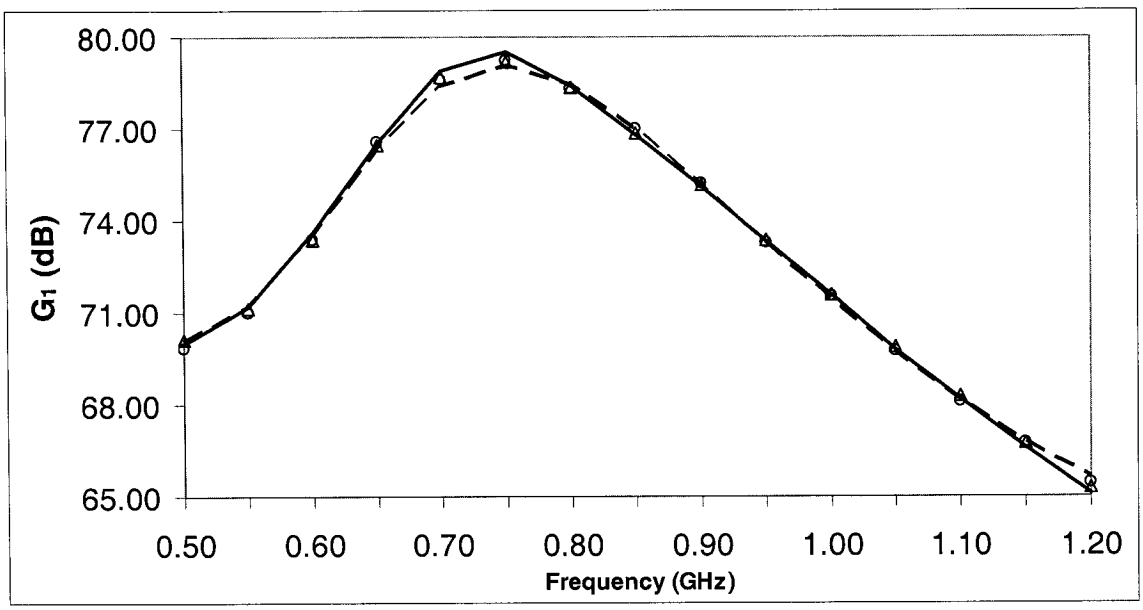


Figure 5-48. 4-stage nonlinear amplifier: Gain at fundamental frequency with -70 dBm input power , comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

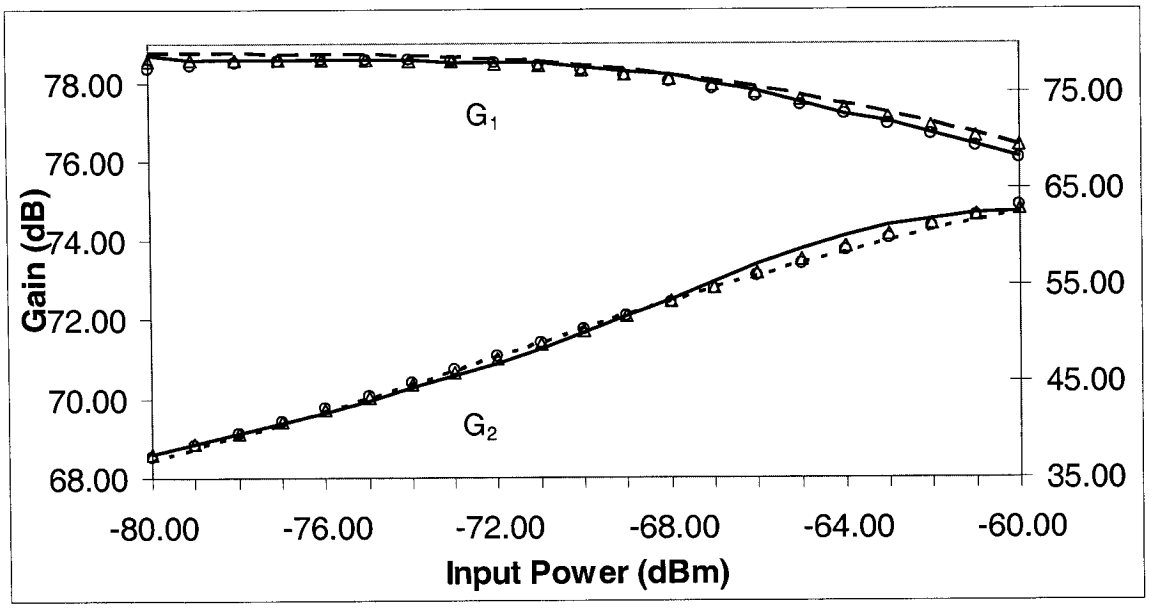


Figure 5-49. 4-stage nonlinear amplifier: Gain at fundamental frequency (0.8 GHz) and the second harmonics, comparing the results of MLP (--), KBNN (Δ), PKI (◦) and original data from ADS (-)

From the above results, we can see that when power and gain of nonlinear amplifier are modeled, although MLP is the worst, the results of it do not differ greatly from the original data since the outputs are weakly nonlinear. However, when the time domain response of the nonlinear amplifier is modeled, the MLP model is poorer. Both KBNN and PKI are much better than MLP in this case. Since what is included in the PKI neural network is only the prior input information, its capability to capture the nonlinear output is not so good as that of KBNN which includes the empirical input and output formula.

5.7 Conclusion

In this chapter, three different neural networks, MLP, KBNN and PKI, were applied to six different examples. The examples cover passive components (e.g. resistor), active

components (e.g. FET), and microwave circuits (e.g. amplifier, mixer). The modeling approach involving neural network is proved to be accurate and efficient. Although the basic MLP can not provide good accuracy and fast speed in modeling some highly nonlinear performance, neural networks such as KBNN and PKI which involve empirical information could greatly improve the accuracy and speed.

Chapter 6

Conclusions and Future Research

6.1 Conclusions

In this thesis, a neural network tool for MLP, KBNN and PKI is developed. The advantages of the knowledge-aided neural networks such as KBNN and PKI were demonstrated through practical examples.

The neural models can learn component behavior originally seen in detailed physics and EM models and can predict such behavior much faster than original models. Compared with the general problem-independent neural networks such as MLP, the knowledge-aided neural networks such as KBNN and PKI, combining microwave empirical experience with the power of learning ability of the neural networks, could offer smaller training error and smaller test error. This advantage is even more significant when the training data are insufficient. The cost of the model development will drop remarkably because of the reduced need for a large number of training data. Furthermore, when the input data are a little bit beyond the training range of the neural model, the knowledge-

aided neural networks could give more accurate outputs based on the knowledge information they involved.

Because the relationship of the original input parameters and output parameters is involved, KBNN has better performance than PKI which has only the prior knowledge of input parameters when the problem is quite complex.

In chapter 5, the empirical knowledge was for the first time used in predicting the performance of the intermodulation of the mixer and the multistage linear/nonlinear amplifiers. It has proved that the knowledge-aided neural networks could efficiently reduce the need for a large number of data and could improve the model accuracy.

6.2 Future Research

The modeling approach by means of neural networks is characterized as fast, accurate and capable of simplifying the complex system modeling by mapping only the input and output parameters. For complex and nonlinear component and circuit behavior, the knowledge-aided neural networks, even when based on simple coarse model, could help to improve the accuracy and efficiency considerately. The work may be a good start for applications of the knowledge-aided neural networks in RF and microwave field to solve more difficult modeling problems.

Only the resistors and capacitors in quite a narrow physical dimension range are modeled in this thesis. So part of the future work could be to develop a library of combined neural

network models for embedded passives in a wider range of physical dimensions and even at higher frequencies.

For the PKI neural networks, we used some coarse models and then combined the coarse model output parameters with the original input parameters manually to train the MLP. So another direction of future work could be to develop a PKI neural network which could be linked to some commercial simulators so that the combination of the coarse model outputs with the original inputs could be done automatically.

As we can see, the capability of the neural networks in the modeling of time domain response is not so good even for the pure sinusoidal signal. Therefore, the development of some advanced algorithms which can learn such performance well is indeed an interesting direction.

Appendices

Appendix A

Gradient Derivation for MLP and KBNN

In order to minimize training error, all the training methods we concern use the gradient information of the training error with respect to the weight parameters. We will derive the derivative of training error to weight parameters of MLP and KBNN neural network respectively.

If we use only one training sample at a time to update $\boldsymbol{w}$, the per-sample error function can be similarly defined as,

$$E_k = \frac{1}{2}\sum_{j=1}^{m} e_j^2 \qquad k \in T_r \tag{1}$$

where

$$e_j = y_j(\boldsymbol{x}_k, \boldsymbol{w}) - d_{jk} \quad j = 1, 2, \cdots, N_L \quad m = N_L \tag{2}$$

1 Gradient Derivation of MLP Neural Network

For the MLP neural network, as we define in chapter 2, the output of the neuron j at hidden layer l is z_j^l and the corresponding weighted sum is $\gamma_j^l = \sum_{i=0}^{N_{l-1}} w_{ji}^l z_i^{l-1}$.

Thus $z_j^l = \sigma(\gamma_j^l) = \dfrac{1}{1+e^{-\gamma_j^l}}$.

According to the chain rule of calculus, we can express the gradient of error function with respect to weight parameter as:

$$\frac{\partial E_k}{\partial w_{ji}^l} = \sum_{p=1}^{m} \frac{\partial E_k}{\partial e_p} \frac{\partial e_p}{\partial y_p} \frac{\partial y_p}{\partial z_j^l} \frac{\partial z_j^l}{\partial \gamma_j^l} \frac{\partial \gamma_j^l}{\partial w_{ji}^l} \quad j = 1, 2, \cdots, N_l; \quad i = 0, 1, \cdots, N_{l-1} \quad l = 2, 3, \cdots, N_L \tag{3}$$

For simplicity, the subscription k is dropped in the following description. We can easily find out:

$$\frac{\partial E}{\partial e_p} = y_p(\boldsymbol{x}, \boldsymbol{w}) - d_p = e_p \quad p = 1, 2, \cdots, N_L \tag{4}$$

$$\frac{\partial e_p}{\partial y_p} = 1 \quad p = 1, 2, \cdots, N_L \tag{5}$$

Let $y_p = \varphi(\gamma_j^l)$, then

$$\frac{\partial y_p}{\partial \gamma_j^l} = \frac{\partial y_p}{\partial z_j^l} \frac{\partial z_j^l}{\partial \gamma_j^l} = \varphi'(\gamma_j^l) \quad p = 1, 2, \cdots, N_L; \quad j = 1, 2, \cdots, N_l \quad l = 2, 3, \cdots, L \tag{6}$$

and

$$\frac{\partial \gamma_j^l}{\partial w_{ji}^l} = z_i^{l-1} \quad j = 1, 2, \cdots, N_l; \quad i = 0, 1, \cdots, N_{l-1}; \quad l = 2, 3, \cdots, L \tag{7}$$

The use of equation (4) to (7) in (3) yields:

$$\frac{\partial E}{\partial w_{ji}^{l}} = \sum_{p=1}^{m} e_p \varphi'(\gamma_j^l) z_i^{l-1} \quad j = 1, 2, \cdots, N_l;\ i = 0, 1, \cdots, N_{l-1};\ l = 2, 3, \cdots, L \tag{8}$$

$$\frac{\partial E}{\partial w_{ji}^{l}} = \delta_j^l z_i^{l-1} \quad j = 1, 2, \cdots, N_l;\ i = 0, 1, \cdots, N_{l-1};\ l = 2, 3, \cdots, L \tag{9}$$

where the local gradient δ_j^l is defined by,

$$\begin{aligned} \delta_j^l &= \frac{\partial E}{\partial \gamma_j^l} \\ &= \sum_{p=1}^{m} \frac{\partial E}{\partial e_p} \frac{\partial e_p}{\partial y_p} \frac{\partial y_p}{\partial z_j^l} \frac{\partial z_j^l}{\partial \gamma_j^l} \\ &= \sum_{p=1}^{m} e_p \varphi'(\gamma_j^l) \qquad j = 1, 2, \cdots, N_l;\ i = 0, 1, \cdots, N_{l-1};\ l = 2, 3, \cdots, L \end{aligned} \tag{10}$$

From equation (9) we can know, the key step in the calculation of the gradient is to find out the local gradient δ_j^l. We may identify two distinct cases depending on where the neuron j located in the neural network.

Case 1 Neuron j Is an Output Node

Since we use linear function for the output layer $y_j = z_j^L = \gamma_j^L$, so

$$\frac{\partial y_j}{\partial \gamma_j^L} = \frac{\partial y_j}{\partial z_j^L} \frac{\partial z_j^L}{\partial \gamma_j^L} = 1 \quad j = 1, 2, \cdots, N_L \tag{11}$$

According to (10) and (11) the local gradient for the neuron j at the output layer is,

$$\delta_j^L = e_j \quad j = 1, 2, \cdots, N_L \tag{12}$$

Consequently, the gradient of training error to the weight parameters relate to the output neuron is given by,

$$\frac{\partial E}{\partial w_{ji}^L} = \delta_j^L z_i^{L-1} = e_j z_i^{L-1} \quad j = 1, 2, \cdots, N_L;\ i = 0, 1, \cdots, N_{L-1} \tag{13}$$

Case 2 Neuron j Is a Hidden Node

For a hidden neuron j, as we chose the sigmoid activation function for the hidden layer neurons,

$$\frac{\partial \sigma(\gamma)}{\partial(\gamma)} = \sigma(\gamma)(1 - \sigma(\gamma)) \tag{14}$$

so

$$\frac{\partial z_j^l}{\partial \gamma_j^l} = z_j^l (1 - z_j^l) \quad j = 1, 2, \cdots, N_l;\ l = 2, 3, \cdots, L-1 \tag{15}$$

The local gradient is given by,

$$\delta_j^l = \left(\sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{kj}^{l+1} \right) z_j^l (1 - z_j^l) \quad j = 1, 2, \cdots, N_l \quad l = 2, 3, \cdots, L-1 \tag{16}$$

As a result the gradient of training error to the weight parameters relate to the hidden neuron is,

$$\frac{\partial E}{\partial w_{ji}^{l}} = \delta_j^l z_i^{l-1} = \left(\sum_{k=1}^{N_{l+1}} \delta_k^{l+1} w_{kj}^{l+1} \right) z_j^l z_i^{l-1} (1 - z_j^l) \qquad (17)$$

$$j = 1, 2, \cdots, N_l; i = 0, 1, \cdots, N_{l-1} \quad l = 2, 3, \cdots, L-1$$

2 Gradient Derivation of KBNN Neural Network

Let the derivative of E with respect to the output of individual neurons be denoted as $\boldsymbol{g}$.

For an example, for output layer (y layer), g_{y_j} is defined as $g_{y_j} = \frac{\partial E}{\partial y_j}$.

Then the derivative of error E with respect to β's and ρ's inside the output neuron are given by,

$$\frac{\partial E}{\partial \beta_{ji}} = e_j z_i \left(\sum_{k=1}^{N_r'} \rho_{jik} r_k' \right) \quad j = 1, 2, \cdots, m; \quad i = 1, 2, \cdots, N_z \qquad (18)$$

$$\frac{\partial E}{\partial \beta_{j0}} = e_j \quad j = 1,2,\cdots,m \qquad (19)$$

$$\frac{\partial E}{\partial \rho_{jik}} = e_j \beta_{ji} z_i r_k' \quad k = 1, 2, \cdots, N_r'; j = 1, 2, \cdots, m; \quad i = 1, 2, \cdots, N_z$$

(20)

The KBNN training scheme starts to differ from conventional error back propagation below the output layer y, and then the error propagation is split into two parts, one through the knowledge layer z down to the input layer x, and the other part is through the normalized region layer r', the region layer r and the boundary layer b down to the input layer x.

In knowledge part, $\boldsymbol{g}_z$ which is similar defined as $\boldsymbol{g}_y$ can be obtained as:

$$g_{z_i} = \sum_{j=1}^{N_y} e_j \beta_{ji} \left(\sum_{k=1}^{N_r'} \rho_{jik} r_k' \right) \qquad i = 1, 2, \cdots, N_z \tag{21}$$

Continuing with the derivative chain rule, the derivative of error E with respect to the weight parameters inside the neurons are w_{ji},

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}} = g_{z_j} \frac{\partial \varphi_j}{\partial w_{ji}} \qquad j = 1, 2, \cdots, N_z; \quad i = 1, 2, \cdots, n \tag{22}$$

Where $\dfrac{\partial \varphi_j}{\partial w_{ji}}$ is obtained from the problem-dependent microwave empirical functions. In the other part, $g_{r'}$ is first obtained,

$$g_{r_k'} = \sum_{j=1}^{N_u} e_j \sum_{i=1}^{N_z} \beta_{ji} z_i \rho_{jik} \qquad k = 1, 2, \cdots, N_r' \tag{23}$$

The derivative for the next two layer, r and b layer, are:

$$\begin{aligned} g_{r_i} &= \sum_{j=1}^{N_r} \frac{\partial E}{\partial r_j'} \frac{\partial r_j'}{\partial r_i} \\ &= \sum_{j=1}^{N_z} g_{r_j'} \left[\frac{\sum_{k=1}^{N_z} r_k - r_i}{\left(\sum_{k=1}^{N_z} r_k \right)^2} \right] \qquad i = 1, 2, \cdots, N_r \end{aligned} \tag{24}$$

$$\begin{aligned} g_{b_i} &= \sum_{j=1}^{N_r} \frac{\partial E}{\partial r_j} \frac{\partial r_j}{\partial b_i} \\ &= \sum_{j=1}^{N_r} g_{r_j} r_j (1 - \sigma(\alpha_{ji} b_i) + \theta_{ji}) \alpha_{ji} \qquad i = 1, 2, \cdots, N_b \end{aligned} \tag{25}$$

The derivative of E with respect to α's and θ's inside region layer neurons are:

$$\begin{aligned} \frac{\partial E}{\partial \alpha_{ji}} &= \frac{\partial E}{\partial r_j} \frac{\partial r_j}{\partial \alpha_{ji}} \\ &= g_{r_j} r_j (1 - \sigma(\alpha_{ji} b_i + \theta_{ji})) b_i \qquad j = 1, 2, \cdots, N_r; \quad i = 1, 2, \cdots, N_b \end{aligned} \tag{26}$$

$$\begin{aligned} \frac{\partial E}{\partial \theta_{ji}} &= \frac{\partial E}{\partial r_j} \frac{\partial r_j}{\partial \theta_{ji}} \\ &= g_{r_j} r_j (1 - \sigma(\alpha_{ji} b_i + \theta_{ji})) \qquad j = 1, 2, \cdots, N_r; \quad i = 1, 2, \cdots, N_b \end{aligned} \tag{27}$$

The derivatives of error E with respect to weight parameter v inside the boundary neurons is:

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial b_j}\frac{\partial b_j}{\partial v_{ji}} = g_{b_j} x_i \qquad j = 1, 2, \cdots, N_b; \quad i = 1, 2, \cdots, n \tag{28}$$

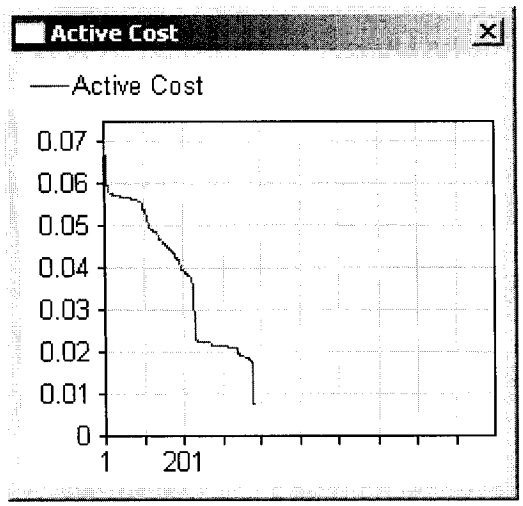
The derivatives of error with respect to all the weight parameters inside KBNN neural network are thus calculated.

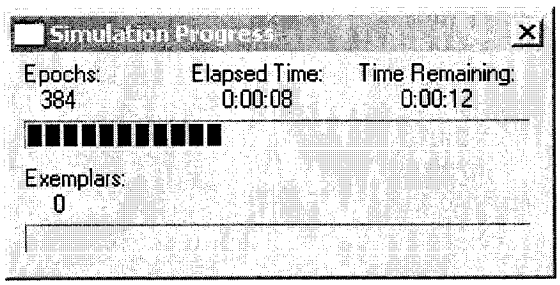
Appendix B

Neural Network Tool Validation with NeuroSolutions

NeuroSolutions [25] is a highly graphical neural network development tool that enables the user to easily create a neural network model. This software combines a modular design interface with advanced learning procedures, giving the power and flexibility needed to design the neural network that produces the best solution for specific problem.

With the data of validation example #1, a neural network was constructed and trained by using NeuroSolutions. The results are shown in the following figures. After 384 iterations, the active cost (training error) came down to around 0.8% which is similar to our neural network tool (0.520%) and Matlab (0.467%).


Active Cost
Active Cost
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
1
201


Simulation Progress
Epochs:
384
Elapsed Time:
0:00:08
Time Remaining:
0:00:12
Exemplars:
0

Bibliography

[1] X. Ding, B. Chattaraj, M.C.E. Yagoub, V.K. Devabhaktuni, Q.J. Zhang, "EM based statistical design of microwave circuits using neural models", *Int. Symp. on Microwave and Optical Technology* (ISMOT 2001), Montreal, Canada, June, 2001, pp.421-426.

[2] X. Ding, J.J. Xu, M.C.E. Yagoub, Q.J. Zhang, "A new modeling approach for embedded passives exploiting state space formulation", *European Microwave Conf.* (EuMC 2002), Milan, Italy, Sept. 23-27, 2002.

[3] Q.J. Zhang, M.C.E. Yagoub, X. Ding, D. Goulette, R. Sheffield, and H. Feyzbakhsh, "Fast and accurate modeling of embedded passives in multi-layer printed circuits using neural network approach", *Elect. Components & Tech. Conf.*, San Diego, CA, May 2002, pp. 700-703.

[4] Q.J. Zhang, and K.C. Gupta, *Neural Networks for RF and Microwave Design*, Artech House, Norwood, MA, 2000.

[5] A.H. Zaabab, Q.J. Zhang, and M.S. Nakhla, "A Neural network modeling approach to circuit optimization and statistical design", *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1349-1358, 1995.

[6] P. Burrascano, S. Fiori, and M. Mongiardo, "A review of artificial neural networks applications in microwave computer-aided design", *Int. J. RF and Microwave CAE*, vol. 9, pp. 158-174, 1999.

[7] F. Wang, and Q.J. Zhang, "Knowledge based neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2333-2343, 1997.

[8] S. Haykin, *Neural Networks: a Comprehensive Foundation*, Macmillan College Publishing, New York, 1994.

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

[10] G. Cybenko, "Approximation by Superpositions of a Sigmoid Function," *Math. Control Signal System*, vol. 2, pp.303-314, 1989.

[11] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, vol.2, pp. 356-366, 1989.

[12] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layer versus three", *IEEE Trans. Neural Networks*. Vol. 8, pp.251-255, 1997.

[13] 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. Digest*, Baltimore, MD, pp. 9-12, 1998.

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning internal representation by error propagation*, *Parallel Distributed Processing*, MIT Press, Cambridge, vol. I, pp.318-362, 1986.

[15] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T.Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, MA, 1986.

[16] H. Robbins, and S. Monro, "A Stochastic Approximation Method," *Annals of Mathematical Statistics*, vol.22, pp. 400-407, 1951.

[17] *Neural Network Toolbox: For Use with Mablab*, the Math Works Inc., Natick, Massachusetts, 1993.

[18] D. B. Parker, "Optimal Algorithms for Adaptive Neural Networks: Second Order Backpropagation, Second Order Direct Propagation and Second Order Hebbian Learning," In *Proc. IEEE First Intl. Conf. Neural Networks*, vol. II, San Diego, California, pp.593-600, 1987.

[19] R. L. Watrous, "Learning Algorithm for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," In *Proc. IEEE first Intl. conf. Neural Networks*, vol. II, San Diego, California, pp. 619-627, 1987

[20] W. H. Press, B. P. Flannery, S.A. Teukolsky, and W. T. Vetterling, Numerical Recipes in C, Cambridge University Press, Cambridge, 1988.

[21] S. S. Rao, *Engineering Optimization, Theory, and Practice,* New York: John Wiley and Sons, 1996.

[22] T. R. Cuthbert, "Quasi-Newton Methods and Constraints," In Optimization using Personal Computers, NY: John Wiley and Sons, pp. 233-314, 1987.

[23] A. J. Shepherd, *Second-Order Methods for Neural Networks,* London: Springer, 1997.

[24] A. J. Shepherd, "Second-Order Optimization Methods," In *Second-Order Methods for Neural Networks*, London: Springer-Verlag, pp.43-72, 1997.

[25] *NeuroSolutions 5.0,* NeuroDeminsion Inc., Gainesville, FL.

[26] X. Ding, *"Neural network based modeling technique for modeling embedded passives in multilayer printed circuits,"* Master thesis, Carleton University, 2002

[27] *Sonnet* 9.52, Sonnet Software Inc., Liverpool, NY.

[28] S. S. Mohan, M. D. M. Hershenson, S. P. Boyd, and T. H. Lee, "Simple Accurate Expressions for Planar Spiral Inductances," *In IEEE Solid-State Journals,* vol. 34, pp. 1419-1424, 1999.

[29] T. H. Bui, *"Design and Optimization of a 10 nH Square-Spiral inductor for Si RF Ics.,"* Master thesis, University of North Carolina at Charlotte, 1999.

[30] C. M. Snowden, *Semiconductor Device Modeling,* Peter Peregrinus (London, 1988).

[31] J. M. Golio, *Microwave MESFETs and HEMTs*, Artech House (Boston, 1991).

[32] J. W. Bandler, S. H. Chen, Y. Shen, and Q. J. Zhang, "Integrated Model Parameter Extraction using Large-Scale Optimization Concepts," *Microwave Theory and Techniques, IEEE Tansaction* on, vol. 36, pp. 1639-1638, 1998.

[33] M. Berroth, and R. Bosch, "High Frequency Equivalent Circuit of GaAs Depletion and Enhancement FETs for Large Signal Modeliing," *Workshop on Measurement Techniques for Microwave Device Characterization and Modeling*, pp. 122-127, 1990.

[34] R. Menozzi, A. Piazzi, and F. Contini, "Small –Signal Modeling for Microwave FET Linear Circuits Based on a Generic Algorithn," *IEEE Trans. Circuits and Systems*, vol.43, pp. 839-847.

[35] M. Fernandez-Barciela, P. J. Tasker, Y. Campos-Roca, M. Demmler, H. Massler, E. Sanchez, M. C. Curras-Francos, and M. Schlechtweg, "A Simplified Broad-band Large-Signal Nonquasi-static Table-based FET Model," *IEEE Trans. Microwave Theroy Tech.*, vol. 48, pp. 395-405, 2000.

[36] M. Berroth, and M. Bosch, "Broad-band Determination of the FET Small-signal Equivalent Circuit," Microwave Theory and Techniques, IEEE Trans on, vol.38, pp/ 891-895, 1990.

[37] L. Ji, "Fuzzy-neural Tool for Optimum Topology Extraction of RF/Microwave Transistors," Master thesis, University of Ottawa, 2005.

[38] *ADS 2003A Mixer Project,* Agilent Technologies, Palo Alto CA.

[39] D. M. Pozar, Microwave Engineering, New York: John Wiley & sons, 2005.

[40] *ADS 2003A Large Signal Amplifier Project*, Agilent Technologies, Palo Alto CA.