Neural networks for characterizing magnetic flux leakage signals

by

Marian M. Chao

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Electrical Engineering

Iowa State University
Ames, Iowa

1995

TABLE OF CONTENTS

ACKNOWLEDGMENTS

My deepest gratitude goes to my adviser, Dr. Satish Udpa, for his unfaltering encouragement, guidance, patience, and understanding. Dr. Udpa: Thank you for being my teacher, my mentor, and most of all, my friend. To Dr. Lalita Udpa: Thank you for your guidance and support. To Dr. William Lord: Thank you for your encouragements, long philosophical chats, and deep emotional probing into pursuing my dreams and finding my passions in life. I wish to sincerely thank Dr. Eric Bartlett for graciously agreeing to be on my committee. I wish to thank Dr. Mani Mina for always having his door opened to me. Also, I wish to thank all my colleagues, especially my "pig" associates who have made this so memorable and actually fun! Grateful acknowledgment is also due to Gas Research Institute for funding me as a research assistant for my entire duration at Iowa State University.

Lastly, I wish to take this opportunity to thank the utmost important people in my life: To my parents, Chun-Yun and Pi-Lien: *I love you* and thank you for all that you have sacrificed for me. To my brother, Joe: Thank you for your encouragement, support, morale-boosts, optimistic attitude, idealistic dreams, and care-free spirit when I needed it most. I couldn't have survived without you here. And to my sister, Mei-Ling: Thank you for your constant support and encouragement and always being there for me.

CHAPTER 1. INTRODUCTION

1.1 Background

Nondestructive evaluation (NDE) has been an extremely important area of study that is involved with testing a product or material without destroying its integrity or serviceability. The field of NDE has recently gained recognition for its usefulness and importance, especially as a critical component in quality control of manufactured parts in many industries.

The concept of NDE is based on the analysis of information generated during the interaction between an energy source and the test specimen. The form of the energy source is chosen appropriately in accordance with the properties of the specimen and the objectives of the test. Examples of energy sources include acoustic waves, x-rays, and magnetic fields [1]. The nondestructive testing (NDT) system contains an energy source that interacts with the test specimen. The response of this interaction is measured and analyzed to determine the condition of the specimen. The fundamental concept of NDT is to inject energy in the test specimen and then measure the resulting energy source/test specimen interaction without causing damage to the test specimen. The inverse problem in NDE is concerned with deducing the state or integrity of the object under

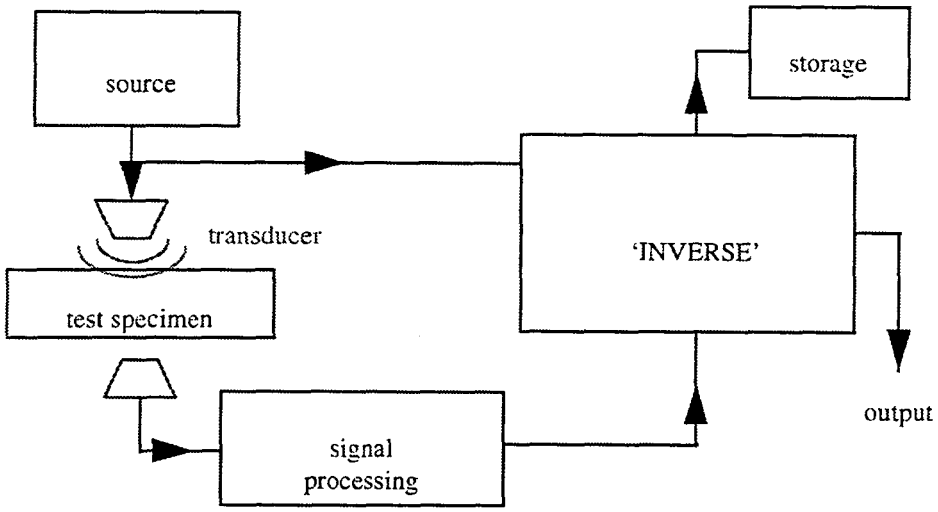

Figure 1.1. Generic NDE system [2].

inspection, i.e., estimate the size, shape, orientation of defects which may be present in the test specimen. A block diagram of a typical NDE system is shown in Figure 1.1 [2].

Practical NDE applications find usage in the inspection of a wide range of engineering components such as integrated chips, aging aircraft, railroad tracks and wheels, heat exchanger tubes, and natural gas transmission pipelines. The accurate detection and characterization of flaws is critical in containing manufacturing costs as well as in saving human lives and property. Indeed, the primary motivation behind the research work described in this thesis is related to the issue of ensuring safety through NDE by characterizing defects in natural gas transmission

pipelines. These flaws could potentially have fatal consequences if left undetected.

The NDT method utilized in this research work is the electromagnetic method; in particular, the magnetic flux leakage (MFL) technique. MFL methods are used for inspecting ferromagnetic material structures. It is known that the presence of a defect in a magnetized ferromagnetic specimen results in a redistribution of the flux lines causing some of these flux lines to "leak" into the surrounding medium. The leakage flux may be sensed and measured by a flux sensitive device such as a Hall probe. This is illustrated in Figure 1.2.

The approach proposed for solving the inverse problem is through the use of neural networks. Artificial neural networks have been studied extensively for many years by researchers who were motivated by a

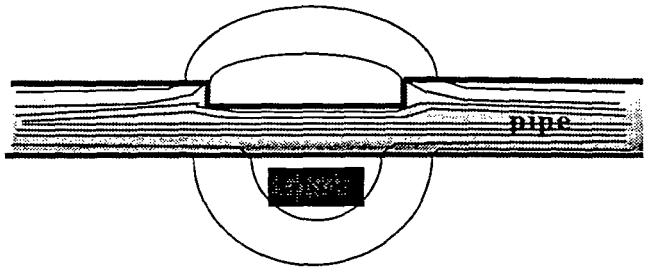

Figure 1.2. Leakage flux in the vicinity of a surface-breaking anomaly.

desire to mimic the information processing strategies of the human brain and develop systems that match the pattern recognition and cognitive skills of biological creatures. In fact, considerable progress in achieving this goal is steadily being made. Neural networks have become known for their impressive classification capabilities of sample patterns. Different neural network paradigms employ different learning rules; however, all of them in some way determine pattern statistics from a set of training samples and then classify new patterns on the basis of information acquired from the exemplars [3].

The back-propagation method proposed by Rumelhart and McClelland [4] is one of the most commonly used algorithms for training neural networks. The most well-known neural network that employs this learning algorithm is the multilayer perceptron (MLP). Multilayer perceptrons are feedforward networks with at least one layer of nodes between the input and output nodes, called the "hidden" nodes. Feedforward layered neural networks are used extensively in many areas of signal processing. The use of these networks for processing complex signals can be interpreted as performing a curve-fitting operation in a multidimensional space. Such networks can be employed for realizing complex nonlinear decision functions or to approximate certain complicated data-generating mechanisms.

A drawback of these traditional neural networks is related to the excessive computation effort involved in training them as well as a tendency to gravitate towards suboptimal solutions. The learning employs nonlinear optimization techniques, and the parameter estimate may become trapped at a local minimum of the chosen optimization criterion during the learning procedure [4].

Recently, many researchers have turned their attention to a number of alternate neural network models, among which is the radial basis function network [5]. The radial basis function (RBF) network is a fairly new concept that has recently gained wide interest and attention in the area of artificial neural networks.

Similar to the MLP, RBF networks are two-layer networks with good approximation capabilities. Originally, the RBF method was introduced strictly as a tool for interpolation in multidimensional space. In this scheme, the RBF method employs as many basis function centers as there are data points. This is extremely impractical in signal processing applications, since the number of data points is usually very large [4]. The RBF network developed for this research more closely adopts Broomhead and Lowe's [7] approximation to the original RBF model. The modified approach is more suitable for signal processing applications where we typically encounter overdetermined systems [7].

This thesis describes the application of RBF networks for characterizing defects in natural gas transmission pipelines. Artificial neural networks have been applied extensively for defect sizing in the past. The radial basis function network, in particular, has been shown to be particularly successful in defect sizing applications [8]. However, in all these applications, the RBF network has been used only to predict simple characteristics of the defect such as the size, location, or orientation. This thesis describes an extension of the concept where an RBF network is used to characterize the complete defect profile. Results obtained to date have proven the feasibility of using neural networks for solving inverse problems in nondestructive evaluation [2].

1.2 Scope of Thesis

This thesis focuses on the characterization of MFL signals using artificial neural networks. To provide an appreciation and understanding of the problem under investigation, Chapter 2 begins with the problem statement and a brief background and motivation for this research. This is followed by a description of the approach employed for solving the problem. The defect characterization network requires an extensive data set for training. Since experimental data is relatively scarce and expensive to obtain, numerical models simulating the test are employed

for generating training data. Chapter 2 provides a brief description of the finite element model that was used for generating the data.

In Chapter 3, a detailed discussion of artificial neural networks is presented. In particular, the multilayer perceptron and radial basis function networks are described and compared. The chapter describes the concept of a "three-dimensional" artificial neural network that is currently being evaluated for defect characterization.

Chapter 4 is devoted to a discussion of various methods used in selecting centers that are needed in RBF networks. The K-means clustering algorithm has traditionally been used to calculate the centers required by the RBF network. The K-means algorithm is well established and widely used because of its simplicity. The chapter examines alternative methods for selecting the centers and compares their performances. The chapter describes a new optimal procedure as well as a method of using potential functions approach for determining the basis function centers. The superiority of these approaches is shown through validation studies in Chapter 5. Finally, Chapter 6 presents conclusions together with a discussion of difficulties encountered and suggestions for future work.

CHAPTER 2. MAGNETIC FLUX LEAKAGE TECHNIQUES

2.1 Problem Statement

The motivation behind this research work comes from a desire to detect and characterize defects which occur in natural gas transmission pipelines. Natural gas is a vital resource in meeting many of the nation's high energy demands. The gas is transported from the well to the consumer using a network of pipelines. Most of the pipeline systems in this country were built within the last four decades, although some constructed before World War II are still in use today. There are over 90,000 miles of natural gas gathering and field pipelines, 280,000 miles of U. S. transmission pipelines, and more than 835,000 miles of gas distribution mains and service lines. This pipeline system has become a critical means of supplying energy that would otherwise be impractical and extremely costly to replace. Consequently, preventive maintenance methods are used to secure the integrity and serviceability required to meet future demands of transporting natural gas [10]. Figure 2.1 illustrates the pipeline system more clearly.

One of the most popular methods for inspecting pipelines is the magnetostatic technique. Magnetostatic methods of nondestructive evaluation are used extensively for the inspection of ferromagnetic specimens. In the past, magnetic "inks" or powders were widely accepted

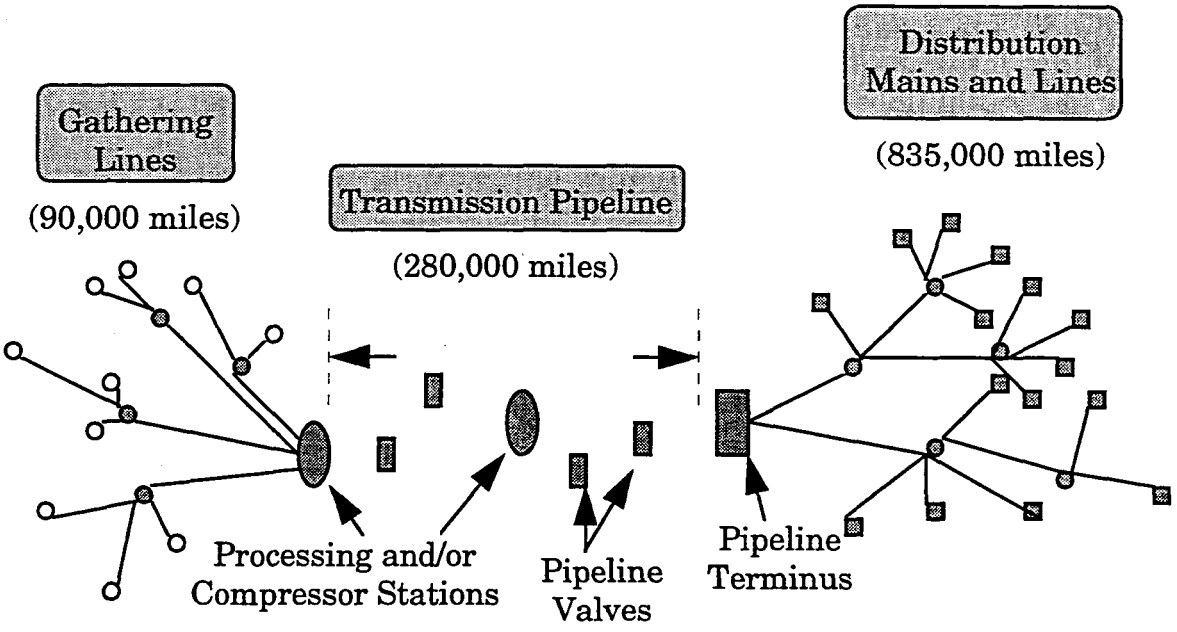

Figure 2.1. Natural gas pipeline system [10].

as reliable indicators of flaws and defects found in components and assemblies manufactured from ferrous metals. Today, improved methods of detecting magnetic leakage flux associated with defects at or near the surface of a magnetized ferromagnetic material are in use. The magnetostatic method can be classified on the basis of the state of the excitation source during the inspection. If the excitation source is energized during the inspection, the method is called an active leakage field test. If the test relies on the measurement of the residual field present in the specimen after the source is de-energized, the method is called the residual leakage field test. Active leakage field methods are

one of the most commonly used techniques for the in-line inspection of natural gas transmission pipelines.

The inspection is achieved by launching an inspection vehicle; otherwise known as a "pig," through the pipeline. The pig is propelled by the flow of the natural gas in the transmission pipes. Fully equipped with appropriate instrumentation and devices, the pig detects and records the NDE signals generated due to corrosion and cracks existing on the inner, as well as the outer diameter of the pipe. The pig employs permanent magnets and a magnetic circuit to saturate the pipe wall. In the presence of surface-breaking anomalies, magnetic flux "leaks" into the region surrounding the test object. This leakage flux may then be detected by a flux sensitive device such as a Hall probe. The characteristics of this magnetic flux leakage (MFL) profile is indicative of the nature of the defect. Figure 2.2 shows the axial and radial components of a typical MFL signal. The objective is to determine the profile of the defect based on information contained in the MFL signal. Figure 2.3 shows such a mapping.

The characterization of defects found during in-line inspection of the pipelines is, however, fraught with several problems. Chief among the problems is the sensitivity of the signal to a number of operational variables. These variables include probe velocity and pipeline stress

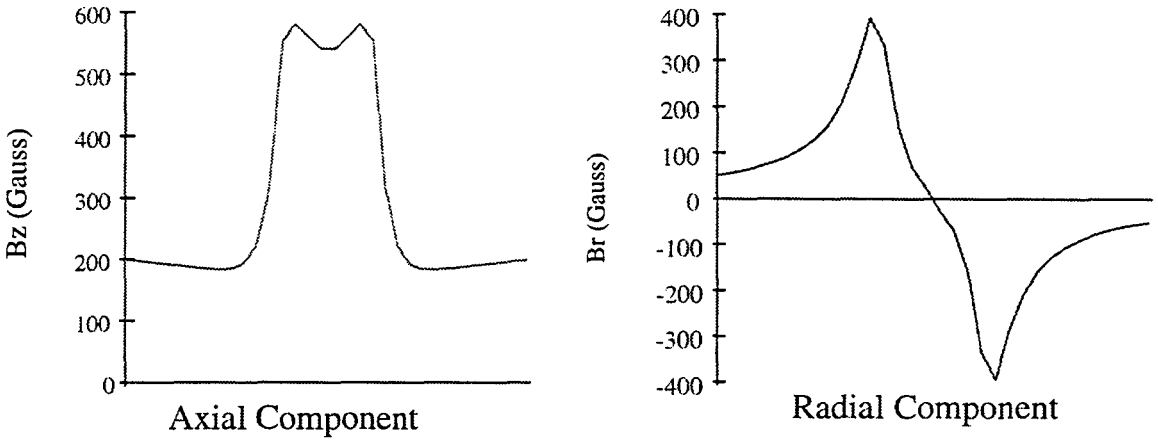

Figure 2.2. Axial and radial components of a typical MFL signal.

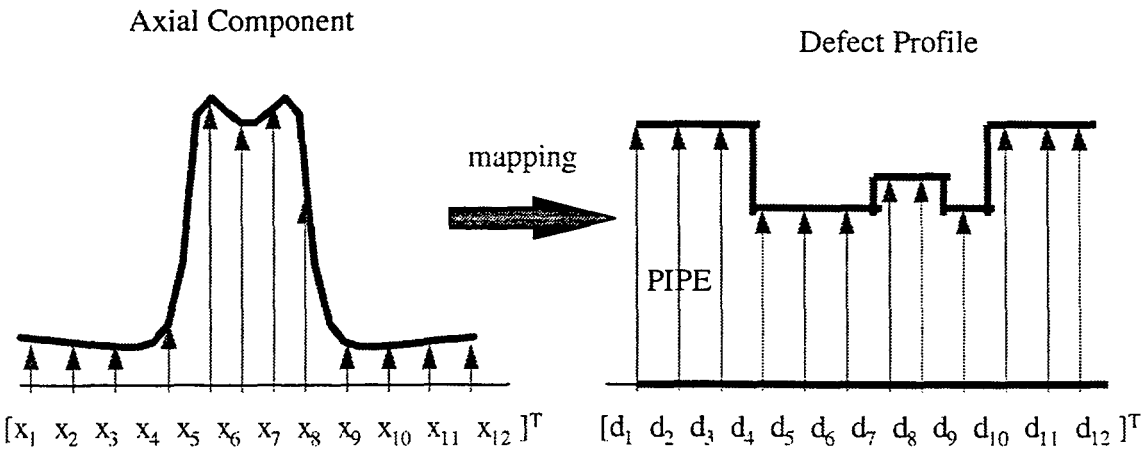

Figure 2.3. Inverse mapping from an MFL signal to defect profile.

levels. The latter affects the permeability of the pipeline which, in turn, affects the MFL signal. Another factor that complicates the characterization process is variation in the permeability of the pipe. Accurate characterization of the defect requires a proper understanding of these factors and methods to compensate for the effects [6]. It is very difficult and expensive to study the effects of these variables using experimental methods. An alternative approach is to use a numerical model as a test bed and simulate the test conditions. Such a test bed can also serve as a source for generating signals required for designing defect characterization systems. One of the more powerful tools for simulation is the finite element technique. The modeling technique is described in section 2.2.

Accurate defect characterization offers significant benefits: Pipeline companies benefit from an understanding of when and how operational variables affect inspection results. These results naturally lead to better planning of the operational controls needed for accurate inspections. Maintenance and repair operations benefit by reducing the number of bellholes required as a consequence of increased inspection accuracy. Inspection vendors benefit by understanding their systems and where improvements are beneficial and needed. Pipeline owners gain better knowledge of the accuracy, strengths, and limitations of present MFL

inspection tools. Researchers benefit from helping further current state-of-the-art technology. All of these support the enhancement of safety, reliability, integrity, and serviceability of natural gas transmission pipelines [10].

2.2 Finite Element Method

2.2.1 Introduction

The finite element method is a numerical technique for solving partial differential equations to obtain approximate solutions to a wide variety of engineering problems. The basic premise of the finite element method is that a solution region can be analytically modeled or approximated by representing it as an assemblage of discrete elements. Instead of solving the partial differential equation directly, the finite element method involves the minimization of an energy functional. Since these elements can be assembled in various ways, they have the ability to represent exceedingly complex shapes. In other words, the finite element method takes the approach of dividing the solution domain into a finite number of subdomains, or elements. These elements are connected only at nodal points in the domain and on the element boundaries: The solution domain is discretized and represented as a patchwork of elements. To summarize

in general terms, the finite element modeling technique involves the following steps [7]:

1. Identify an appropriate energy functional corresponding to the partial differential equation
2. Discretize the continuum
3. Select interpolation functions
4. Determine the element "stiffness" matrix
5. Assemble the local stiffness matrices to obtain the global matrix equation
6. Solve the system of equations to obtain the solution
7. Use the solution to compute other parameters of interest

In discretizing the continuum, once the element mesh for the solution domain is defined, the behavior of the unknown field variable over each element is approximated by continuous functions expressed in terms of the nodal values. The function defined over each finite element is called an interpolation or shape function. The set of interpolation functions for the whole solution domain yields a piecewise polynomial approximation to the field variable [7]. The numerical solution of the partial differential equation reduces to solving a system of algebraic equations in terms of parameters defining the approximate solution [13].

2.2.2 Governing Equations

Maxwell's equations are the fundamental equations that govern all electromagnetic phenomena. The equations may be expressed in both differential and integral form, but are presented here in differential form since they lead to differential equations that can be solved using the finite element method [14].

For general time-varying fields, Maxwell's equations can be written as follows [14]:

$$\nabla \times \mathbf{E} + \frac{\partial \mathbf{B}}{\partial t} = 0 \qquad (2.1)$$

$$\nabla \times \mathbf{H} - \frac{\partial \mathbf{D}}{\partial t} = \mathbf{J} \qquad (2.2)$$

$$\nabla \cdot \mathbf{D} = \rho \qquad (2.3)$$

$$\nabla \cdot \mathbf{B} = 0 \qquad (2.4)$$

where

E = electric field intensity (volts/meter)

D = electric flux density (coulombs/meter^2)

H = magnetic field intensity (amperes/meter)

B = magnetic flux density (webers/meter^2)

J = electric current density (amperes/meter^2)

ρ = electric charge density (coulombs/meter^3)

In the case when the field quantities do not vary with time, they are called static fields and may be expressed as follows:

$$\nabla \times \mathbf{E} = 0 \qquad (2.5)$$

$$\nabla \times \mathbf{H} = \mathbf{J} \qquad (2.6)$$

Equations (2.3) and (2.4) remain unchanged.

Clearly, under this situation, no interaction between electric and magnetic fields exist and can, thereby, be described as an *electrostatic* case (equations (2.3) and (2.5)) or a *magnetostatic* case (equations (2.4) and (2.6)).

Additionally, assuming isotropy, the following constitutive relations relate the macroscopic properties of the medium and the field variables:

$$\mathbf{D} = \varepsilon \mathbf{E} \qquad (2.7)$$

$$\mathbf{B} = \mu \mathbf{H} \qquad (2.8)$$

$$\mathbf{J} = \sigma \mathbf{E} \qquad (2.9)$$

where the parameters ε, μ, and σ denote, respectively, the permittivity (Farads/meter), permeability (Henrys/meter), and conductivity (Siemens/meter) of the medium. The parameters are tensors for anisotropic media and scalars for isotropic media. For inhomogeneous

media, they are position-dependent, while they are not for homogeneous media.

To solve Maxwell's equations, the first-order differential equations involving two field quantities may first be converted into second-order differential equations involving a field quantity:

Exploiting the fact that **B** is divergence free we can write:

$$\mathbf{B} = \nabla \times \mathbf{A} \qquad (2.10)$$

where **A** is called the *magnetic vector potential*.

Substituting equation (2.10) into equation (2.6) and utilizing equation (2.8) yields the second-order differential equation:

$$\nabla \times (\frac{1}{\mu} \nabla \times \mathbf{A}) = \mathbf{J} \qquad (2.11)$$

This does not uniquely define **A** since if **A** is a solution to equation (2.11), any function that can be written as $\mathbf{A}' = \mathbf{A} + \nabla \mathbf{f}$ is also a solution regardless of the form of f. Therefore, in order to uniquely define **A**, a condition on its divergence also needs to be defined. Such a condition is called a *gauge condition* and a natural choice is

$$\nabla \cdot \mathbf{A} = 0 \qquad (2.12)$$

The finite element model used in this study exploits the axisymmetric nature of the pig geometry. Finite element analysis methods solve the partial differential equation governing the physical process in an indirect manner.

An alternative to solving equation (2.11) directly is to embed the governing partial differential equation in an energy functional. The energy functional corresponding to equation (2.11) is given by

$$\iiint_{v} (\mathbf{H} \cdot d\mathbf{B} - \mathbf{J} \cdot d\mathbf{A}) dv \quad (2.13)$$

where, again, **H** represents the magnetic field intensity, **B** is the magnetic flux density, and v is the volume of interest.

Minimizing this energy functional is tantamount to solving the partial differential equation. The method involves discretization of the region with an appropriate mesh. In minimizing the functional at each of the nodes, a matrix equation is generated. Solving this matrix equation yields the vector magnetic potential, **A**, which can then be used to determine other quantities of interest such as the flux density in the material and the leakage field profile.

2.3 Data Collection

The MFL signals used for training the neural network are generated using a finite element model. The performance of a neural network is largely dependent on the amount and quality of data presented during the training process. In other words, in order for a neural network to properly learn the properties inherent in a given data set, an extensive and comprehensive amount of training data is required. Since the amount of

experimental data is limited, this study relied primarily on MFL signals generated using the finite element model. The model has been validated using experimental results. Examples of simulated signals (axial components) for various rectangular defect lengths are shown in Figure 2.4. It is known that the peak-to-peak distance is equivalent to the length of the defect. Moreover, the peak-to-peak magnitude is equivalent to the depth of the defect [15]. In order to minimize the computational effort, the finite element model exploits the axisymmetric nature of the geometry as illustrated in Figure 2.5. A detailed two-dimensional tool geometry is shown in Figure 2.6. It shows the various components and materials that comprises the tool. The axisymmetric defect encircles the outer diameter of the pipe as shown in the shaded region.

Once these signals are generated and a substantial data base constructed, the neural network may be trained. The approach taken in using neural networks to solve the inverse problem in NDE is one of multidimensional mapping. The concept of neural networks is explained in details in the next chapter.

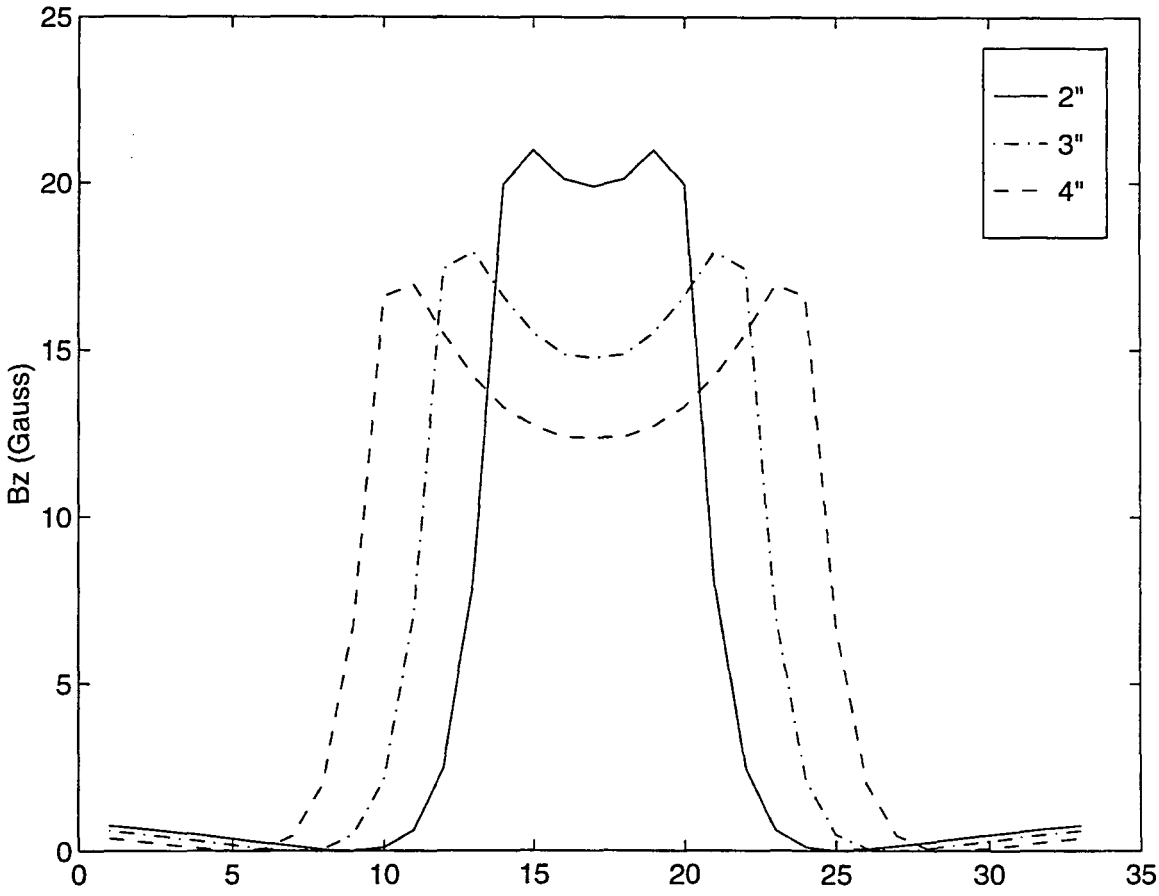

Figure 2.4. Examples of simulated signals for 2", 3", and 4" long defects.

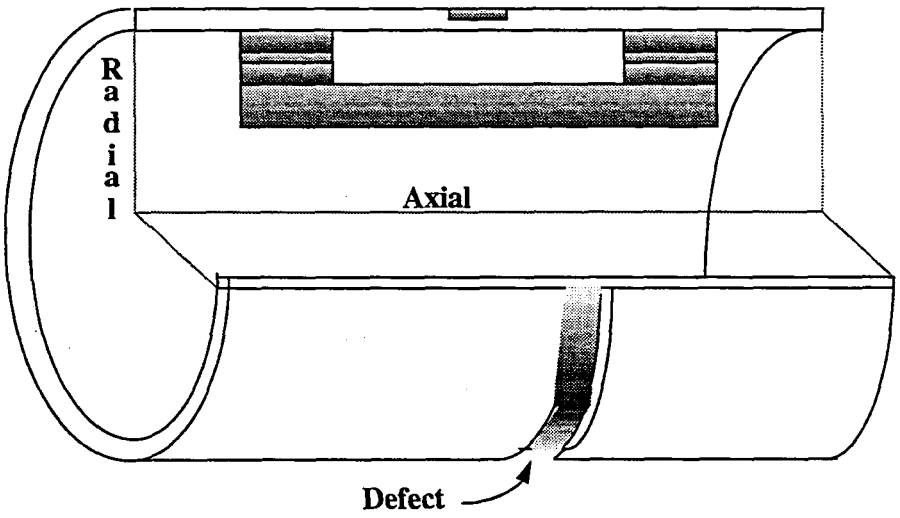

Figure 2.5. Axisymmetric approximation of a pig.

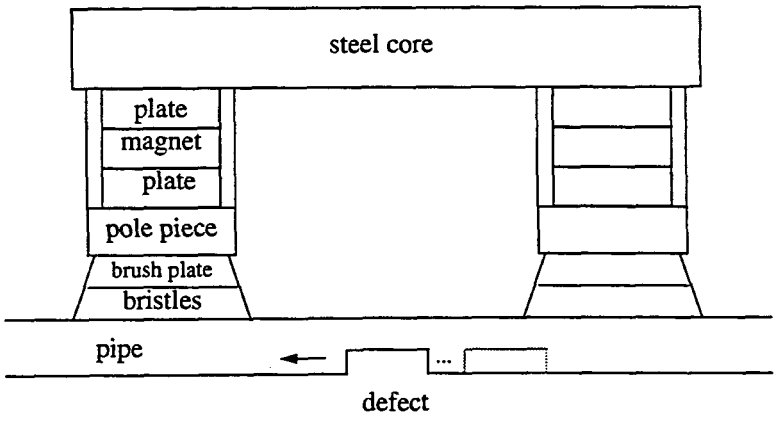

Figure 2.6. Two-dimensional tool geometry.

CHAPTER 3. NEURAL NETWORKS

3.1 Background

Artificial neural networks have been the focus of extensive studies by researchers in hopes of achieving human-like performance in solving problems that require cognitive skills. Examples of such problems include those encountered in speech and image recognition [9]. Artificial neural networks are composed of simple processing elements that are densely interconnected. These networks are trained to perform arbitrary mappings between sets of input-output pairs through the adjustment of interconnection weights. In this sense, the architecture of artificial neural networks emulates that of a biological nervous system. Neural networks are attractive in that they require no *a priori* information or built-in rules; rather, they acquire knowledge of the data through the presentation of examples. This characteristic allows neural networks to approximate mappings for functions that do not appear to have a clearly defined algorithm or theory.

The computational elements or nodes used in neural networks are nonlinear in nature. A simple node sums N weighted inputs and passes the result through a nonlinearity; otherwise known as an activation function, as shown in Figure 3.1. The node is characterized by an internal bias θ and by the type of nonlinearity. The neural network

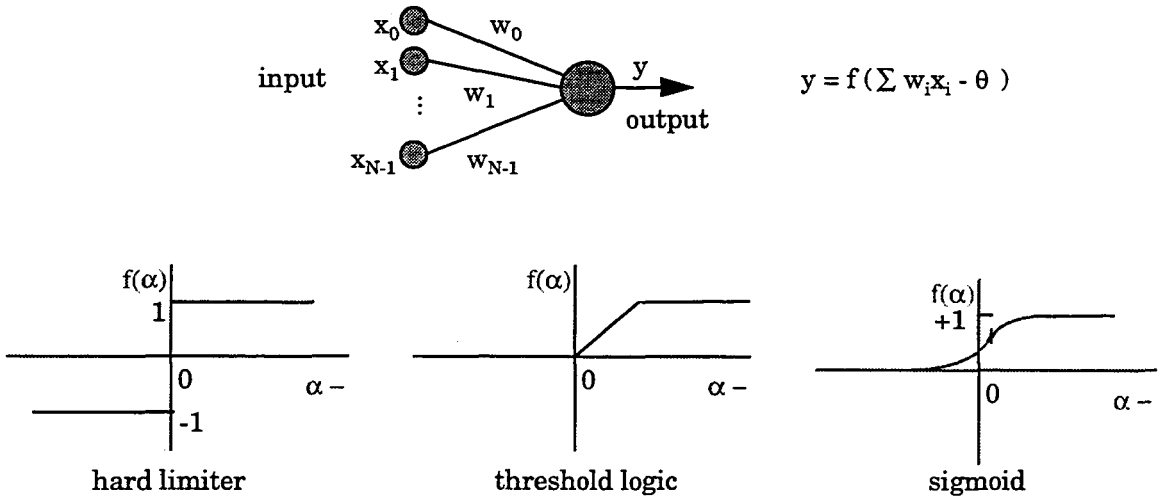

Figure 3.1. Activation functions [16].

characteristics are determined by the network topology, node characteristics, and training or learning algorithms. The algorithms dictate the initial weight values and subsequent adaptation during the training process so as to improve performance.

Neural networks are known for their robustness or fault tolerance, in that the failure of a few processing nodes or links will not have a significant effect on the overall performance. In addition, most neural networks adapt connection weights with new data so as to improve performance continually with time. The ability to adapt and continually learn is an important asset unique to neural networks and learning systems. Adaptation provides a degree of robustness by compensating for

minor variability in the characteristics of processing elements. The functionality of the network is defined by the nature of interconnection weights and the type of processors used. The determination of the interconnection weights is essentially equivalent to the determination of the input-output relation of the network and, hence, constitutes the training procedure of the neural network. In other words, the information inherent in sample patterns, required for discrimination, is automatically extracted and embedded into the network in the form of interconnection weights.

Several types of neural networks have been proposed and are primarily distinguished by their architecture and the learning rule employed to train them. Examples of these include the multilayer perceptron (MLP), Hopfield network, Kohonen network, and the more recently developed, Radial Basis Function (RBF) network.

Each network offers its own set of advantages and disadvantages and certain networks are preferred over others depending on the particular application of interest. For this research, we use an RBF network. A justification of this choice will be presented later. However, the MLP is also used to facilitate a comparison of the results and demonstrate the superiority of the RBF network relative to the MLP for the application on hand. To further understand the properties of neural networks, we will

look at two primary application categories of neural networks. Neural network applications can be classified into two main categories: *recognition* and *generalization*. The training for both types of neural network applications involves the presentation of a set of input-output pairs (exemplars) (I_1,O_1), (I_2,O_2), ..., (I_n,O_n). The main distinction between the two categories is that in *recognition* problems, the trained network is tested with an exemplar signal I_j $(1\leq j\leq n)$ corrupted by noise, as shown in Figure 3.2. The trained network is expected to reproduce the output O_j, corresponding to I_j, in the presence of noise. Examples of these types of applications include shape and handwriting recognition. In *generalization* problems, the trained neural network is tested with input I_{n+1}, which is distinct from the inputs I_1, I_2,..., I_n, used for training the network as shown in Figure 3.3. The network is expected to predict correctly the output O_{n+1} for the input I_{n+1} from the model it has learned through training.

There are many real-world applications that would benefit from the use of neural networks for solving generalization problems, because it is extremely difficult to successfully apply either conventional mathematical techniques (e.g. statistical regression) or standard artificial intelligence approaches (e.g. rule-based systems) for solving such problems. The generalization ability of a neural network is useful since it does not

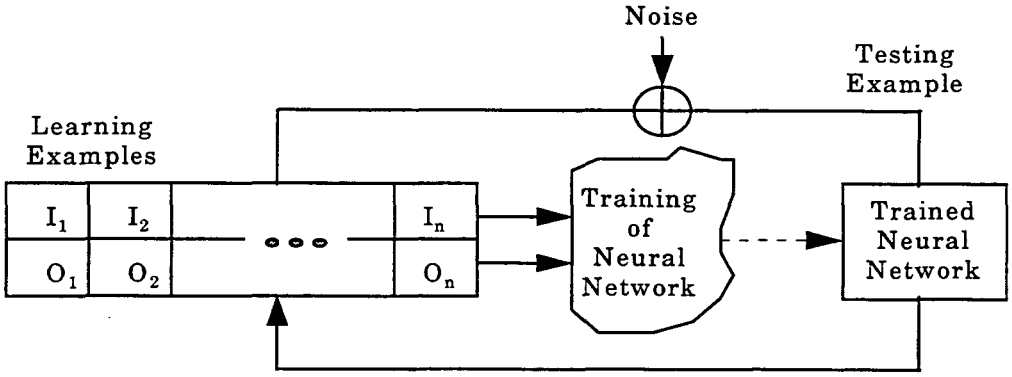

Figure 3.2. Recognition problem [17].

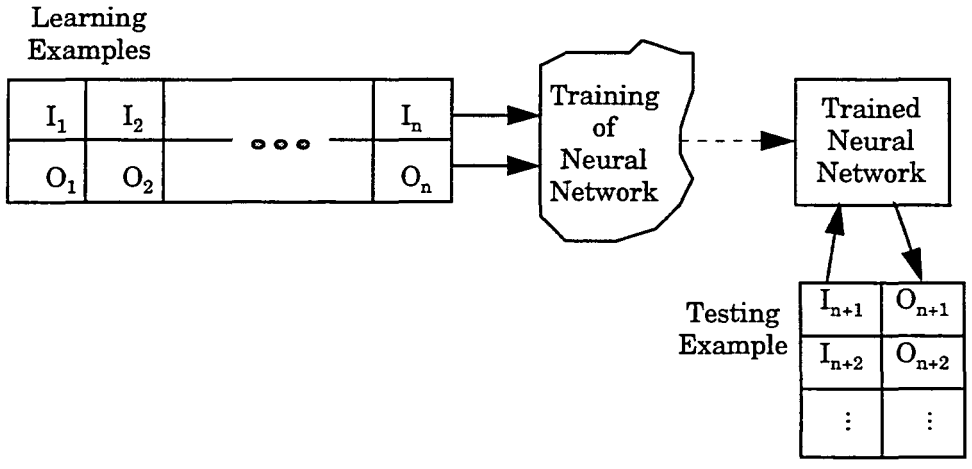

Figure 3.3. Generalization problem [17].

require an *a priori* specification of a functional domain model; rather, it attempts to learn the underlying structure relating input-output data from the training samples [17].

Learning algorithms for generalization and recognition problems are different. In the case of recognition problems, the neural network is expected to reproduce one of the previously seen outputs. The network may recall the outputs and inputs by fitting a curve through the (I_i,O_i) pairs used for training. To remember the outputs, a large network with numerous nodes and weights may be employed. However, the memorization of learning samples is not appropriate for generalization problems since this may result in overfitting. Overfitting which results in poor performance can be measured in terms of the ability of the network to correctly predict the output when novel inputs are presented. Networks designed for solving generalization problems can tolerate a small amount of error in the predicted output; therefore, the fitted curve is not required to pass through any (I_i,O_i) pair used in the training phase. Neural networks designed for solving generalization problems may instead fit a simple curve (e.g. a low degree polynomial, or basic analytical functions such as log(x), sine(x), tangent(x), etc.) through the input-output pairs. Neural networks employed for generalization applications are usually simpler, employing a small number of hidden nodes, layers, and

interconnection edges and weights, allowing the usage of more computationally sophisticated algorithms [17]. The RBF networks used in this work are designed for generalization applications.

3.2 Multilayer Perceptrons

Multilayer perceptrons (MLPs) are used very widely in diverse applications. These networks are usually trained in a supervised manner with a popular algorithm known as the error back-propagation algorithm [11]. A typical example of an MLP network is shown in Figure 3.4.

The resurgence in the popularity of layered, feedforward networks (perceptrons) has been credited to the development of the error backward propagation algorithm for the determination of the synaptic coupling strengths in multilayered networks with hidden layers.

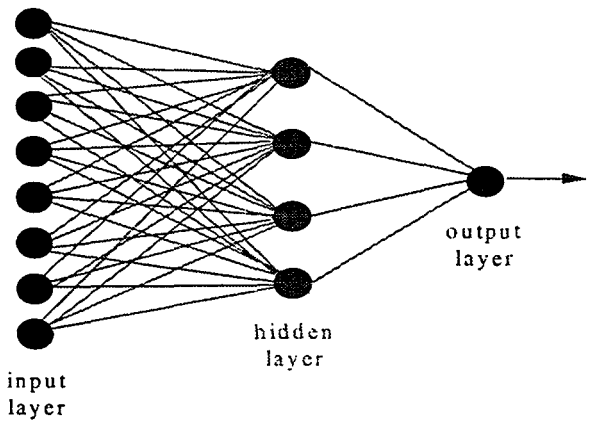

Figure 3.4. MLP network architecture.

The learning algorithm is extremely simple and yet powerful. The synaptic strengths w_{ij} are iteratively modified such that the output signal differs minimally from the desired one. This may be achieved by using the gradient method, which yields the required modifications δw_{ij}. The operation of this network corresponds to a highly nonlinear mapping between the input and the output; consequently, the method is applied recursively until a predefined convergence criterion is reached.

Error back-propagation is a particular example of a larger class of learning algorithms that are classified as *supervised learning* approaches since at each step the network parameters are adjusted appropriately by comparing the actual output with the desired output [18].

The success of back-propagation was first demonstrated by Hinton [19] in training neural networks for nonlinear XOR problems. Since then, its application has become widespread in numerous pattern recognition problems including its use for solving generalization problems [17]. Back-propagation is a learning algorithm for the derivation of weights in feed-forward neural networks. The algorithm minimizes the error of fit to learning samples by fine-tuning the weights during the learning process. In each iteration, there are two phases: forward propagation and reverse propagation. In the forward propagation, the output of the network is

computed using the input vector. The total error, E, is computed in this phase by comparing the desired with the actual outputs:

$$E = (1/2)\Sigma(y_{jp} - d_{jp})^2 \qquad (3.1)$$

In the reverse propagation, the error derivative with respect to all the network weights is computed. The error derivative associated with a weight is an estimate of the effect of that weight on the total error. In other words, the total error, with respect to a given set of learning (or training) samples and a given set of weights, is given by equation (3.1), where y_{jp} is the actual output of node j in training sample p, and d_{jp} is the desired output. The error derivative with respect to the weight w_{ij} is employed to calculate the change in weight w_{ij} as given by equation (3.2). The weight change is accordingly computed such that it moves the network in the direction of maximum error reduction, or gradient of error surface.

$$\Delta w_{ij} = -k(\partial E/\partial w_{ij}) = \varepsilon \delta_{pi} a_{pj} \qquad (3.2)$$

δp_i in equation (3.2) is the effect of a change in the input of the network to unit j on the output of unit i in the training sample p. The determination of the incremental change in weights is an iterative process starting at the output unit. This computation is done in the reverse propagation phase. In reference to equation (3.2), the term a_{pj} represents the output of unit j for training sample p; and, ε and k represent constants [17].

To verify how well the neural network has been properly trained, i.e. learned the underlying input-output model, the same set of weights (on the connections) derived during the learning phase, and the accuracy of the predicted output for a new set of input vectors is tested and checked. In general, the success of the predictions for the neural network depends upon the range covered by the input-output vectors of the training samples.

The performance of a neural network that is trained using the error back-propagation learning algorithm depends on two performance parameters: the learning rate and momentum. Learning rate is associated with the change in weights from the error derivatives, and is the constant of proportionality between the two. Ideally, the change in weights should be infinitesimal for a true gradient descent. The momentum term is used to reduce the amount of oscillation caused by large values of learning rates. It modifies the weight changes calculated using the present derivative by an amount proportional to the weight changes in the previous iteration. Also, it is representative of the relative importance of the weight change in the previous iteration [17].

The MLP network used in this study for defect characterization is coded using MATLAB with the built-in neural network functions. The network is trained with a backpropagation learning rule. The training is

stopped when either the maximum number of epochs has been reached or the network sum-squared error attains a value below the error goal.

The time required to train an MLP is typically in the order of a few hours on a DECAxp workstation. Obviously, the MLP approach is not suitable for this application. Also, the performance is not very good, as is evident from some of the typical results presented in Figure 3.5. In fact, with the same training and testing data set, the RBF network offers much better performance with significantly lower training time. This is obvious from the characterization results illustrated in Figure 3.6.

3.3 Radial Basis Functions Network

The design of the radial basis function (RBF) network can be viewed as an exercise in curve-fitting or solving an approximation problem in multi-dimensional space. The learning of this network is, in essence, equivalent to determining a surface in multidimensional space that provides a best fit to the training data, with the definition for "best fit" being measured in some statistical sense. The hidden units of the RBF network provide a set of "functions" that constitute an arbitrary "basis" for the input vectors when they are expanded into the hidden-unit space [11].

RBF networks have recently gained prominence and increased usage as a tool for multidimensional interpolation. The architecture of these

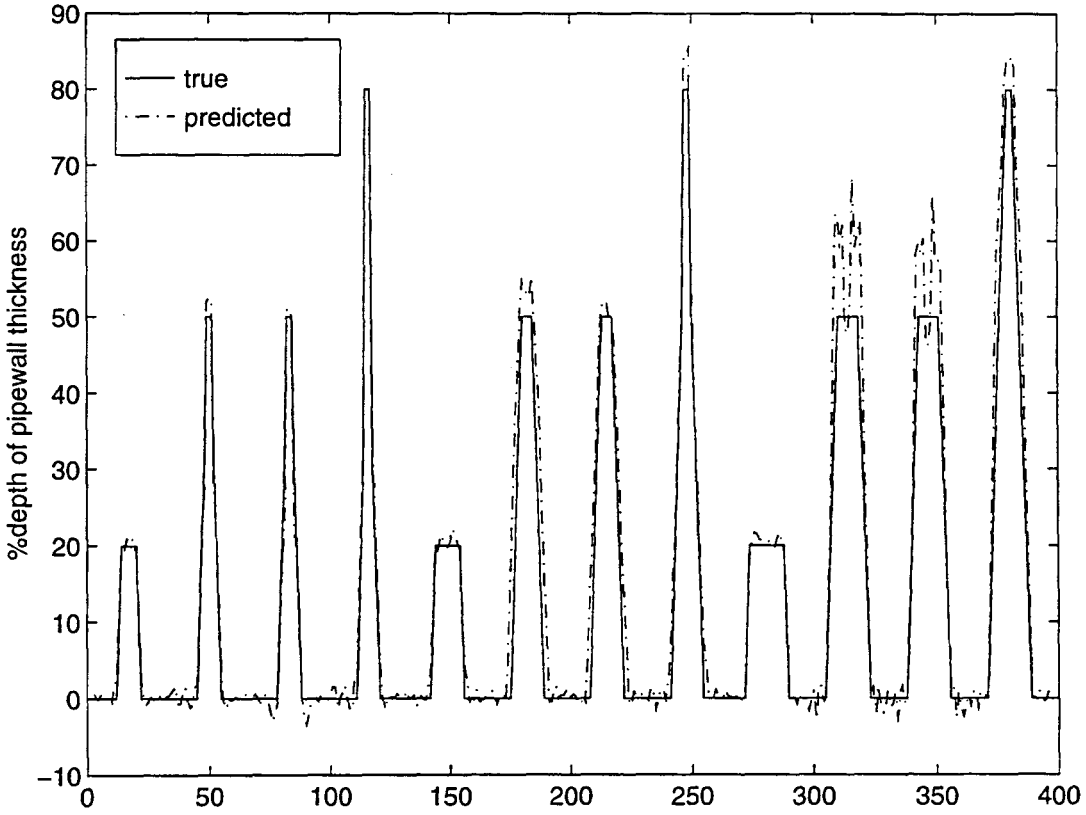

Figure 3.5. MLP characterization results.

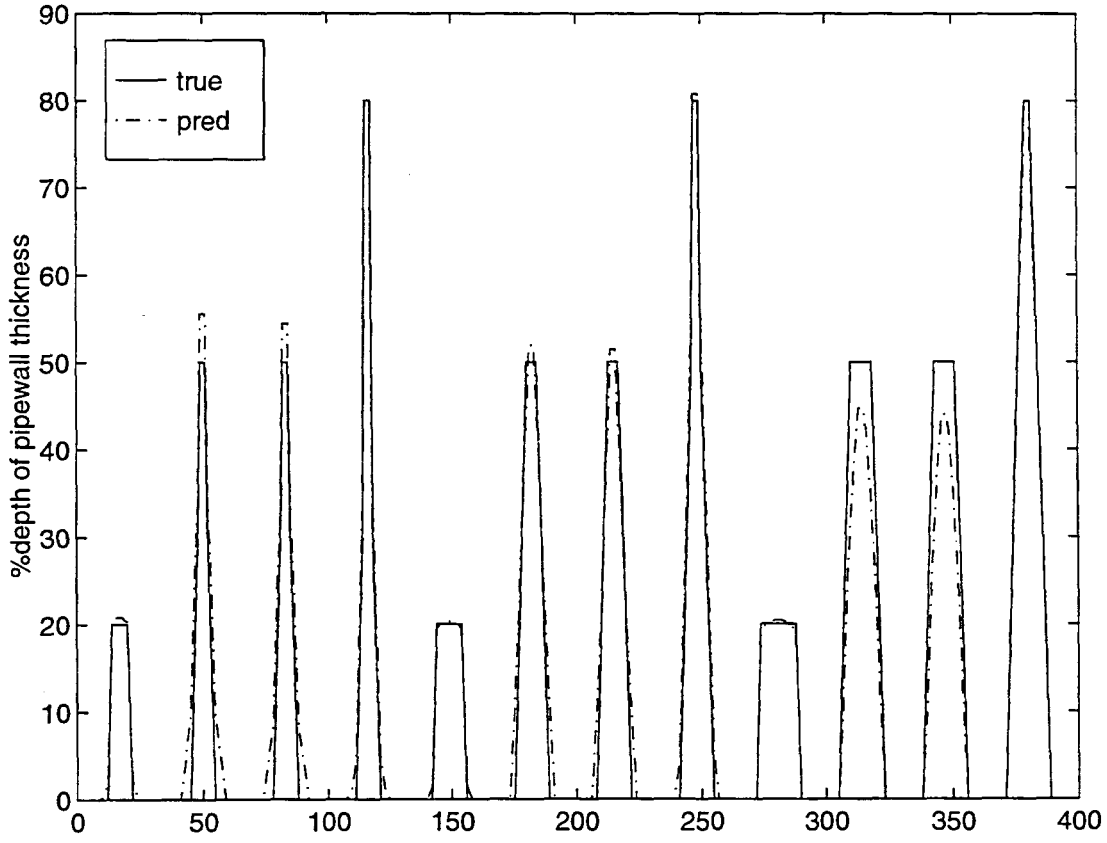

Figure 3.6. RBF characterization results.

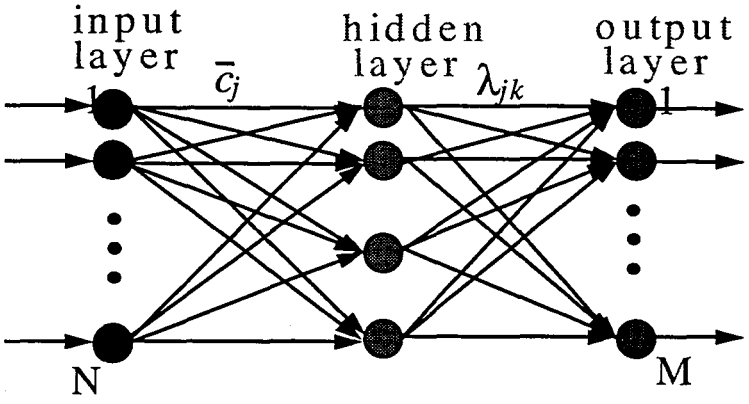

Figure 3.7. RBF network architecture.

networks closely resembles that of the multilayer perceptron. The architecture of a typical RBF network is shown in Figure 3.7.

RBF networks are two-layer networks that can be employed as a tool for multivariate dimensional mapping. They map an n-dimensional input function into an m-dimensional output function using a basis function expansion approach [7]:

Given a set of m distinct vectors or data points:

$$\{\mathbf{x}_j \mid j=1,2,...,m\}$$

and m real function values,

$$\mathbf{f}_j,\ j=1,2,...,m$$

the objective is to determine a function such that

$$\mathbf{s}(\mathbf{x}_j) = \mathbf{f}_j \qquad j=1,2,...,m \qquad (3.3)$$

The function, **s**, is constrained to pass through the known data points.

The approach involves constructing a linear function space that is dependent on the positions of the given data points according to an arbitrary distance measure. Therefore, a set of m arbitrary "basis" functions $\phi(||\mathbf{x} - \mathbf{c}_i||)$ are used. The vectors $\mathbf{c}_i$, i=1,2,...,m are centers of the radial basis functions and usually chosen from sample data points.

Using the concept of basis function expansion, we consider interpolating functions of the form:

$$\mathbf{s}(\mathbf{x}_j) = \sum \lambda_i \phi(||\mathbf{x} - \mathbf{c}_i||) \qquad (3.4)$$

where $\phi(||\bullet||)$ is an appropriately chosen basis function and $||\bullet||$ denotes an appropriate norm, usually Euclidean.

Inserting the interpolation conditions, i.e., equation (3.3) into equation (3.4), yields a set of linear equations for the coefficients, $\{\lambda_i\}$, which can be expressed in the following matrix form:

$$\begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & & \vdots \\ A_{m1} & \cdots & A_{mm} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_m \end{bmatrix} \qquad (3.5)$$

where $A_{ij} \equiv \phi(||\mathbf{x}_i - \mathbf{c}_j||)$, $i,j = 1,2,...,m$.

If the inverse of matrix A with elements A_{ij} exists, equation (3.5) allows the expansion coefficients λ_j to be computed using

$$\lambda \equiv A^{-1}\,\mathbf{f} \qquad (3.6)$$

It has been proven by Micchelli that for all positive integers m,n and for a significant class of functions ϕ, the matrix A is non-singular if the data points are all distinct [7].

Once the radial basis function, ϕ, is appropriately chosen and a distance measure defined, the above relations specify the interpolation problem exactly. The solution under these conditions is guaranteed. The above analysis is for the case when the number of centers is equal to the number of data samples. This is impractical in many applications where the number of data points is fairly large.

Broomhead and Lowe propose an alternative by weakening the interpolation conditions. They propose a scheme where the number of centers is less than the number of data samples. Under this situation, the problem becomes overspecified; hence, the matrix A is no longer square and consequently, an inverse cannot be computed. An alternative is to determine a λ vector which minimizes $||A\lambda - \mathbf{f}||^2$. The solution is given by $\lambda = A^{+}\mathbf{f}$, where A^{+} is Moore-Penrose pseudo-inverse [7].

A multitude of basis functions may be used in the expansion. Examples include [20]:

$$\phi(p) = \exp\{-p^2/2\sigma^2\} \quad \text{(Gaussian)}$$

$\phi(p) = \log(1+p)$ (logarithmic)

$\phi(p) = p$ (linear)

$\phi(p) = (p^2 + c^2)^{1/2}$ (multiquadric)

where c is a positive constant.

3.4 Comparison of MLP and RBF Networks

The MLP and RBF networks are similar in the sense that they are both nonlinear layered feedforward networks. In fact, an RBF network is capable of accurately emulating a specified MLP network, and vice versa. Nevertheless, many differences exist that distinguish the two networks. Some of these include [21]:

1. The hidden layer of an RBF network is nonlinear, but the output layer is linear. This is in contrast to an MLP where both the hidden and output layers are usually nonlinear. (It should be noted that when an MLP is used to solve nonlinear regression problems, a linear output layer is the preferred choice.)

2. An MLP constructs a global approximation to nonlinear input-output maps. Hence, reasonable generalization capabilities in regions of the input space, where little or no training data is available, may be acquired. On the other hand, an RBF uses localized nonlinearities, such as Gaussian functions; and hence, constructs local approximations to

nonlinear input-output maps. As a result, RBF networks are capable of learning fast and offer less sensitivity to the order of training data presented. Unfortunately, it performs poorly in function extrapolation applications since the basis functions that are chosen usually have very limited support.

3. RBF networks generally have only one hidden layer, whereas MLP networks have one or more hidden layers.

The work described in this thesis is primarily focused on two-dimensional signals. However, studies done to date indicate that it may be necessary to process the signals from all the sensors using a three-dimensional processing scheme. The motivation for pursuing this approach is explained in the following section.

3.5 Extension to Three-Dimensional Neural Networks

It is imperative that the signals be rendered invariant to the various operational parameters for accurate characterization [11]. For example, the distortion effects due to velocity are dependent on the location of the Hall sensor with respect to the flaw. These effects are most significant only in the case of sensors located in the close proximity to the defect walls and MFL measurements located away from defects are not significantly affected. It is critical that velocity effects be properly

accounted for, since the angular position of the sensor relative to the defect may vary due to the tool rotation.

Furthermore, "blooming" of the field far beyond the confines of the defect can lead to considerable error in estimating the width of the defect. Other issues that need to be addressed to ensure accurate characterization of defects include errors introduced by poor sampling of the leakage field along the circumferential direction. A natural solution is to process the signals of all the Hall sensors as a three dimensional array of data and perform both the compensation schemes and the defect characterization using three-dimensional neural networks, as shown in Figure 3.8.

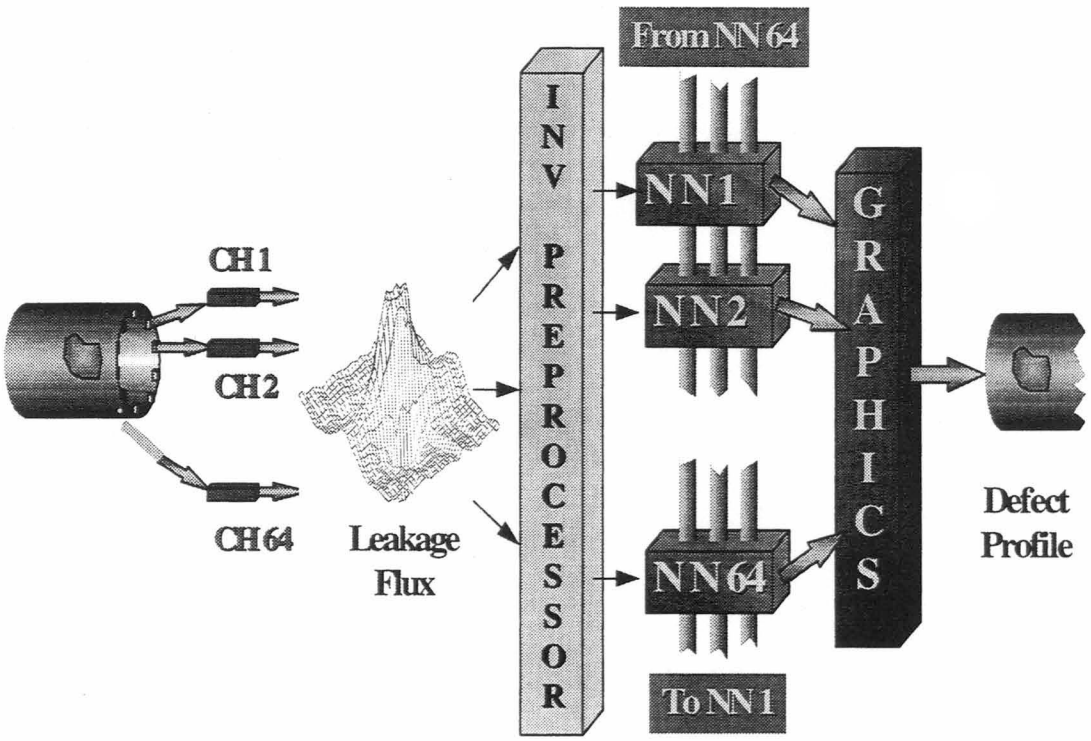

Figure 3.8. Three-dimensional neural network.

CHAPTER 4. CENTER SELECTION METHODS FOR RBF NETWORKS

4.1 Introduction

The accuracy of the defect characterization hinges on the proper choice of the centers of the radial basis functions associated with the network. The center may be selected using one of the following approaches:

1. Clustering algorithms
2. Self-organized selection of centers using potential functions approach
3. Optimal selection of center locations

This chapter describes each of the methods and offers a comparative assessment of the performance obtained. In all of these approaches, a Gaussian, with a fixed region of support, is used as a basis function. Later on, the region of support is also optimized. The Gaussian radial basis function centered at $\mathbf{c}_i$ is defined as [22]:

$$\phi(||\mathbf{x} - \mathbf{c}_i||^2) = \exp\{-||\mathbf{x} - \mathbf{c}_i||^2\} \quad i=1,2,...,N$$

where N is the specified number of centers.

Once the centers are determined, the only parameters that need to be computed are the linear weights, λ's, in the output layer of the RBF network. The determination of these expansion coefficients constitutes the training process. A straightforward procedure for computing the expansion coefficients is to determine the *pseudo-inverse* as described in

Broomehead and Lowe [7].

4.2 K-means Algorithm

The concept of pattern classification employing distance functions is a relatively straightforward method that is used in many applications. The motivation for using distance functions as a classification tool follows intuitively from the fact that the most obvious way of determining similarity among pattern vectors is to consider them as points in the Euclidean space and classify them based on their proximity in the spatial domain. This method of pattern classification may be expected to yield practical and reasonable results when the pattern classes tend to possess clustering properties.

The method of the K-means clustering algorithm is based on the minimization of a performance index that is defined as the sum of the squared distances from all points in a cluster domain to the cluster center. The basic procedure is outlined below [22]:

(1) Choose K initial cluster centers $\mathbf{z}_1(1)$, $\mathbf{z}_2(1)$,..., $\mathbf{z}_k(1)$

(2) Distribute the training samples $\{\mathbf{x}\}$ among the K cluster domains at the kth iterative step using the relation:

$\mathbf{x} \in S_j(k)$ if $||\mathbf{x} - \mathbf{z}_j(k)|| < ||\mathbf{x} - \mathbf{z}_i(k)||$

for all i=1,2,...,K, i≠j.

(3) Compute new cluster centers $\mathbf{z}_j(k+1)$, j=1,2,...,K as follows:

$$\mathbf{z}_j(k+1) = (1/M_j)\sum_{\mathbf{x}\in S_j(k)} \mathbf{x}, \qquad j=1,2,...,K$$

where M_j is the number of samples in $S_j(k)$.

(4) Algorithm has converged and procedure is terminated if

$$\mathbf{z}_j(k+1) = \mathbf{z}_i(k), \text{ for } j=1,2,...,K$$

To begin, the K initial centers are usually chosen to be the first K samples in the training sample set. The subscript specifies a particular center and the number in parentheses indicates the iteration number. In step (2), the term $S_j(k)$ denotes the set of samples whose cluster center is $\mathbf{z}_j(k)$. Next, the center calculation in step (3) is such that the sum of the squared distances from all points in $S_j(k)$ to the new cluster center is minimized, i.e., the new cluster center $\mathbf{z}_j(k+1)$ is computed such that the performance index defined as

$$J_j = \sum_{\mathbf{x}\in S_j(k)} ||\mathbf{x} - \mathbf{z}_j(k+1)||^2, \quad j=1,2,...,K$$

is minimized. The $\mathbf{z}_j(k+1)$ which minimizes this performance index is merely the sample mean of $S_j(k)$.

It is known that the performance of the K-means algorithm is largely influenced by the number of cluster centers chosen, the definition of the initial cluster centers, the order in which the training samples are presented, and the geometrical properties of the training data [22]. The K-means algorithm can be expected to yield reasonably decent results

when the data points are linearly separable. An example of this clustering method is shown in Figure 4.1 where the sample data patterns exhibit clustering properties (linear separability) [22].

Once these centers have been selected and fixed, the training of the RBF network involves the determination of the expansion coefficients, which defines the nature of the multivariate mapping.

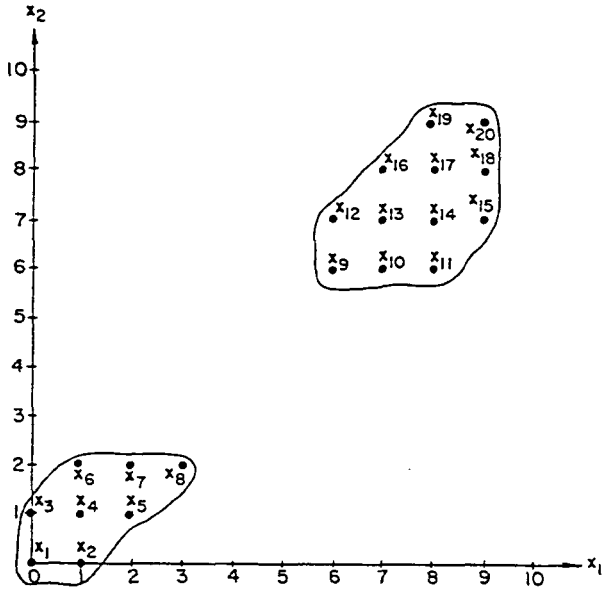

Figure 4.1. Sample data patterns that exhibit clustering property [22].

4.3 Potential Functions Approach

The second method of center selection may be viewed as a self-organized selection process that is based on the concept of the potential functions approach. Implied in the process is the determination of decision functions which generate the partition boundaries in the pattern space separating patterns of one class from another. Unlike the K-means clustering scheme, which focuses on local distance measurements, the potential functions approach integrates the overall error fit between the predicted and true sample values. To understand how the concept of potential functions may be applied in the decision function determination, consider two pattern classes to be distinguished, ω_1 and ω_2. The sample patterns may be either vectors or points in the n-dimensional pattern space. Suppose that these sample pattern points are viewed as some type of energy source, then the potential at any of these points acquires a peak value and then abruptly decreases at any point away from the sample pattern point, $\mathbf{x}_k$. Keeping this analogy in mind, the concept of equipotential contours may be visualized. The pattern class ω_1 may be represented by a "plateau" formed by all sample patterns in ω_1 with the sample points located at the peaks of a group of hills. Analogously, a "plateau" is formed by sample patterns of class ω_2. These two "plateaus" are separated by a "valley" where the potential is essentially zero. This is

illustrated in Figure 4.2, for the two pattern class case. This intuitive analogy naturally leads to the representation of decision functions for pattern classification using the concept of potential functions approach [22].

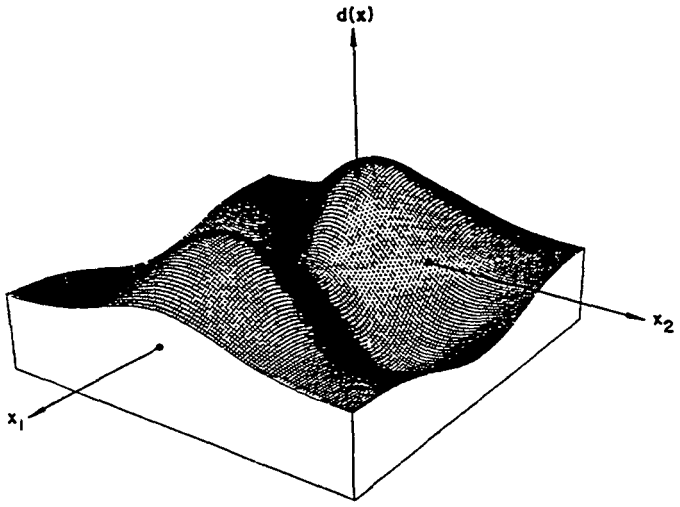

Figure 4.2. Plateaus and valleys of two pattern classes [22].

An alternative analogy to help explain the potential functions approach is to view the center selection process as setting up "tents" in multidimensional space. In this sense, "tents" are propped up in the spatial domain where the spatial derivatives in a region are high, as shown in Figure 4.3 for an arbitrary function in one dimension.

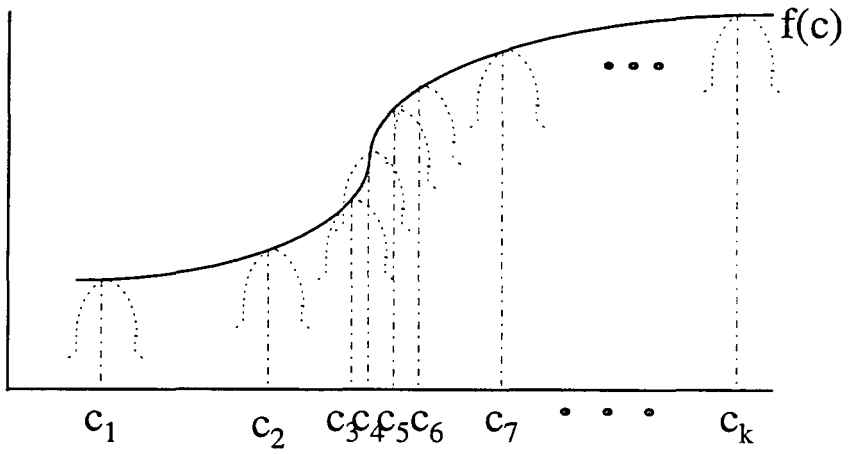

Figure 4.3. "Tents" representing center locations: k is number of centers.

To begin the procedure, the initial cumulative potential $K_0(\mathbf{x})$ is assigned to zero. Also, an error threshold, ε, needs to be defined. Once the first training sample pattern, $\mathbf{x}_1$, has been presented, the cumulative potential is updated as follows:

$$K_1(\mathbf{x}) = K_0(\mathbf{x}) + K(\mathbf{x},\mathbf{x}_1)$$

where, henceforth, the potential function used is of the form:

$$K(\mathbf{x},\mathbf{x}_k) = \exp\{-||\mathbf{x} - \mathbf{x}_k||^2\}$$

Since $K_0(\mathbf{x})=0$, the first computed value of the cumulative potential becomes

$$K_1(\mathbf{x}) = K(\mathbf{x},\mathbf{x}_1)$$

or, simply, the cumulative potential is equal to the potential function for sample pattern $\mathbf{x}_1$. At this point, the cumulative potential $K(\mathbf{x}_1)$ describes the initial partition boundary. Next, when the second training sample pattern, $\mathbf{x}_2$, is presented, the cumulative potential is determined as follows:

If $K_1(\mathbf{x}_2) > \varepsilon$, then the cumulative potential is adjusted using:

$$K_2(\mathbf{x}) = K_1(\mathbf{x}) + K(\mathbf{x}, \mathbf{x}_2),$$

Otherwise, the cumulative potential remains unchanged:

$$K_2(\mathbf{x}) = K_1(\mathbf{x})$$

This procedure is followed subsequently for all the training sample patterns.

Consequently, more "tents" are set up where the difference between the sample value and the function synthesized using the potential functions is high. The center selection process in this sense is done in an intuitively meaningful way by placing "tents" (centers) in only those regions of the spatial domain where it is needed the most. This method of center selection is more appropriate for the application under study since defect characterization is not a problem of classification; but rather, a problem of approximation. The procedure eliminates unnecessary or redundant centers and, thus, minimizes the overall computational effort associated with the characterization phase.

4.4 Optimization Technique

The last method evaluated for center selection is an optimal procedure. In this method, the centers and parameters of the radial basis functions, as well as the expansion coefficients, are determined optimally by minimizing the mean square error using an iterative procedure. Under this approach, the RBF network takes on its most generalized form. The technique employed is based on the minimization of an error function implemented using the *conjugate-gradient* procedure. The first step in the development of such a learning procedure is to define the cost function:

$$E = (1/2)\sum \mathbf{e}_j^2 \qquad (4.4.1)$$

where the summation is over all the training samples used in the training process, j=1,2,...,M (M is the number of training samples), and $\mathbf{e}_j$ is the error defined as

$$\mathbf{e}_j = \mathbf{d}_j - \sum \lambda_i \phi(||\mathbf{x} - \mathbf{c}_i||) \qquad (4.4.2)$$

where $\mathbf{d}_j$ is the desired vector, $\mathbf{x}$ is the training sample pattern, $\mathbf{c}_i$ represents the basis function centers, ϕ is the chosen basis function, and λ_i are the expansion coefficients. Since the error criterion, or cost function, is a nonlinear function of the variables, the problem of finding a globally optimum solution transforms into one of unconstrained nonlinear least squares minimization. These problems are usually solved using iterative

methods. A popular iterative scheme for minimizing the cost function is the conjugate-gradient method.

The goal in the optimization method is to find the parameters λ_i, $\mathbf{c}_i$, and σ_i (assuming that a Gaussian function is employed as the radial basis function), so as to minimize E.

$$\text{minimize } E(\mathbf{c}_i + h\, \mathbf{t}_i)$$

The update equations for λ_i and $\mathbf{c}_i$ are assigned different step parameters. The step size, h, is chosen such that it also minimizes the error function; hence, a similar approach to computing the centers is taken, i.e., use the first partial derivative with respect to h to obtain the gradient. In this sense, the step size calculated will yield the value that minimizes the error cost function. For computing the coefficients which minimize the cost function, the usual procedure is utilized to determine the expansion coefficients. This differs from the conventional MLP where the weights are slowly adjusted to obtain the optimum values. This novel approach of coupling the gradient-descent method of optimizing center locations in conjunction with the matrix inversion method of optimizing coefficients results in a more efficient and faster alternative to the MLP network learning mechanism.

In other words, unlike the conventional back-propagation algorithm, this gradient descent procedure does not involve error back-propagation.

And as mentioned before, the gradient descent procedure for optimizing the center selection is coupled with the pseudo-inverse method of determining the expansion coefficients. As a result, the training time is decreased. The initial values of the various parameters are estimated using a standard pattern-classification method such as an RBF network [11].

The method of conjugate gradients is used extensively. In employing the conjugate-gradient method for the problem under study, a modified form of the method is used. More specifically, in minimizing the error function defined in equation (4.4.1), the first-order partial derivatives with respect to each of the centers are computed. This yields the search direction vector, which is simply the negative of the slope. In general, the partial derivatives of a function with respect to each of the n variables are collectively called the gradient of the function. The gradient is a vector of n-components and it points in the direction of most rapid increase of function values in the n-dimensional space; consequently, the gradient direction is called the direction of steepest ascent. Hence, taking the negative of this gradient vector yields the direction of steepest descent. A major drawback of the steepest descent (or ascent) is that it is a local property and not a global one. Consequently, other means need to be employed to ensure that a global minimum is found. In the method of

conjugate gradients, a new search direction is established using a linear combination of all previous search directions, and the newly determined gradient. A detailed theorem and proof for the development of the conjugate-gradient method is given in [23]. The algorithm adopted for this work is described below:

Step 1. Start with an arbitrary initial guess of the solution $\mathbf{c}_1$

Step 2. Set the first search direction $\mathbf{t}_1 = -\nabla E(\mathbf{c}_1) = -\nabla E_1$

Step 3. Compute the vector $\mathbf{c}_2$ according to the relation:

$$\mathbf{c}_2 = \mathbf{c}_1 + h_1 \mathbf{t}_1$$

where h_1 is the optimal step length in the direction $\mathbf{t}_1$.

Set i=2 and go to the next step.

Step 4. Calculate $\nabla E_i = \nabla E(\mathbf{c}_i)$ and set

$$\mathbf{t}_i = -\nabla E_i + (|\nabla E_i|^2 / |\nabla E_{i-1}|^2)\, \mathbf{t}_{i-1}$$

Step 5. Find the optimum step length h_i in the direction $\mathbf{t}_i$, and update the solution using

$$\mathbf{c}_{i+1} = \mathbf{c}_i + h_i \mathbf{t}_i$$

Step 6. Test for optimality of the $\mathbf{c}_{i+1}$. If the error criterion is satisfied, stop the process; otherwise, set i=i+1, and go to step 4.

The search direction in the *i*th step, $\mathbf{t}_i$, is obtained using:

$$\mathbf{t}_i = -\nabla E_i + \beta_i \mathbf{t}_{i-1}$$

where the value of β_i is determined by making $\mathbf{t}_i$ conjugate to $\mathbf{t}_{i-1}$:

$$\beta_i = \frac{\nabla E_i^T \nabla E_i}{\nabla E_{i-1}^T \nabla E_{i-1}}$$

A general form of this minimization procedure was suggested by Fletcher and Reeves [23]. The flow chart that describes this process of optimizing the RBF centers is shown in Figure 4.4. Using a similar approach to calculate the optimum step size, the function $E(\mathbf{c}_i + h_i \mathbf{t}_i)$ is minimized using a procedure outlined in the flow chart shown in Figure 4.5.

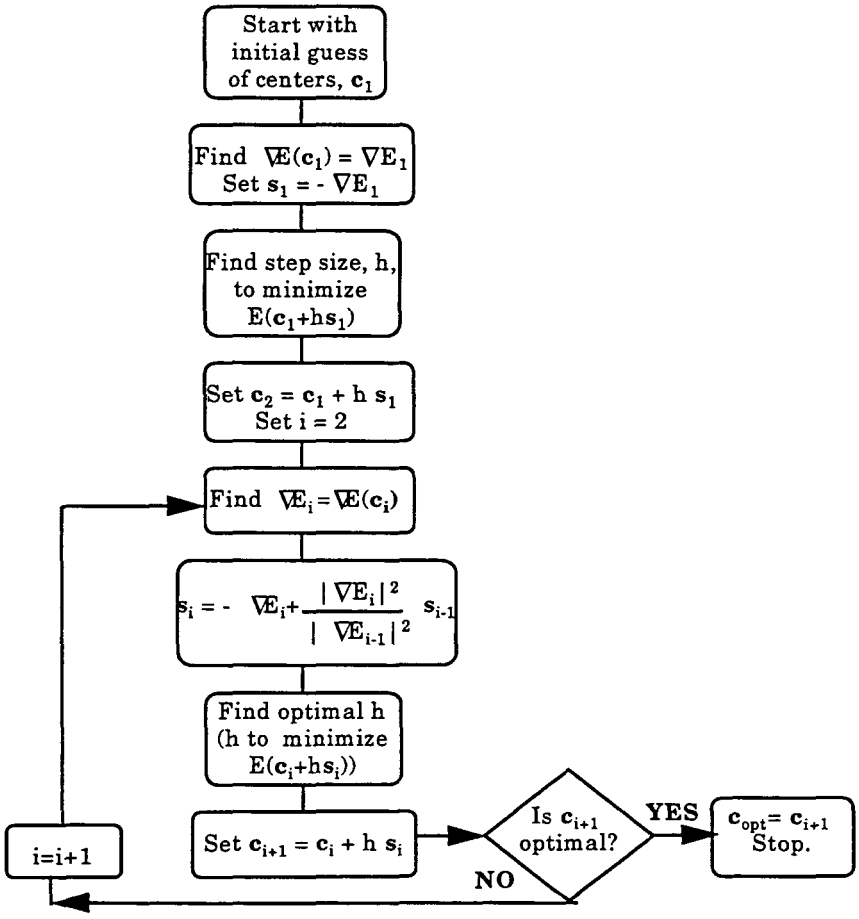

Figure 4.4. Flow chart for optimizing RBF centers.

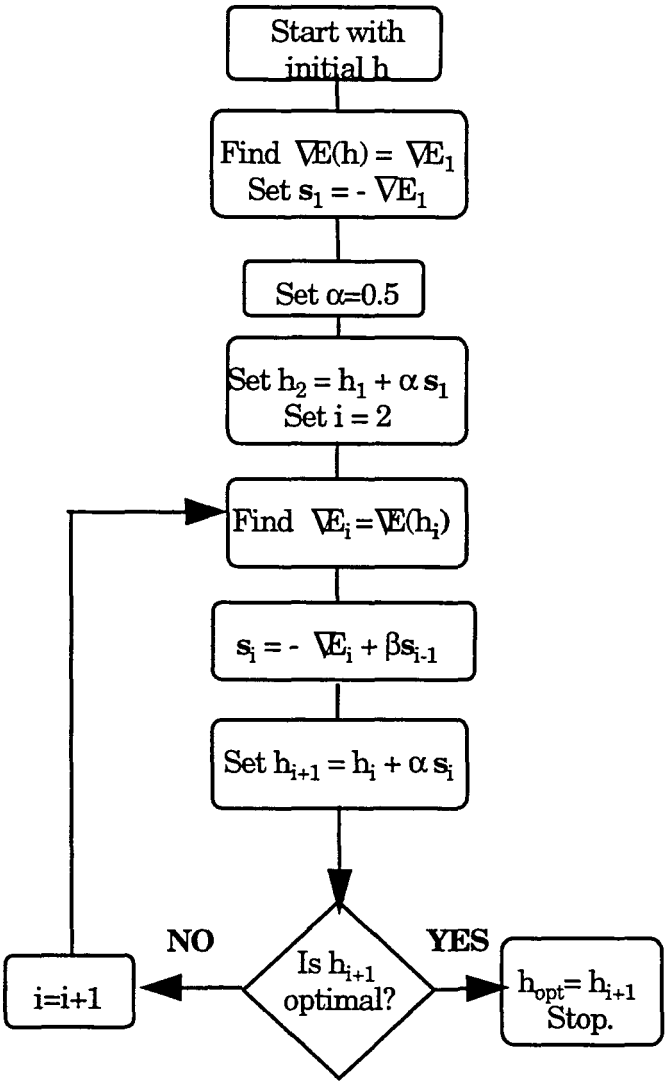

Figure 4.5. Flow chart for computing the optimum step size, h.

The error gradient with respect to the centers can be computed as follows:

$$\text{minimize } E = \frac{1}{2}\sum_{j=1}^{M}\left[\mathbf{d}_j - \sum_{i=1}^{N}\lambda_i e^{-\|\mathbf{x}_j - \mathbf{c}_i\|^2}\right]^2$$

$$\mathbf{t}_1 = \frac{\partial E}{\partial \mathbf{c}_1} = \sum_{j=1}^{M}\left[\mathbf{d}_j - \sum_{i=1}^{N}\lambda_i e^{-\|\mathbf{x}_j - \mathbf{c}_i\|^2}\right]\lambda_1(\mathbf{x}_j - \mathbf{c}_1)e^{-\|\mathbf{x}_j - \mathbf{c}_1\|^2} = 0$$

$$\mathbf{t}_2 = \frac{\partial E}{\partial \mathbf{c}_2} = \sum_{j=1}^{M}\left[\mathbf{d}_j - \sum_{i=1}^{N}\lambda_i e^{-\|\mathbf{x}_j - \mathbf{c}_i\|^2}\right]\lambda_2(\mathbf{x}_j - \mathbf{c}_2)e^{-\|\mathbf{x}_j - \mathbf{c}_2\|^2} = 0$$

$$\mathbf{t}_N = \frac{\partial E}{\partial \mathbf{c}_N} = \sum_{j=1}^{M}\left[\mathbf{d}_j - \sum_{i=1}^{N}\lambda_i e^{-\|\mathbf{x}_j - \mathbf{c}_i\|^2}\right]\lambda_N(\mathbf{x}_j - \mathbf{c}_N)e^{-\|\mathbf{x}_j - \mathbf{c}_N\|^2} = 0$$

$$\nabla \mathbf{t} = \begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_{21} \\ . \\ \mathbf{t}_N \end{pmatrix}$$

where M is the number of sample patterns and N is the number of center vectors. To compute the optimum step size, h_i, the error gradient with respect to the step size can be determined as follows:

$$\text{minimize } E(\mathbf{c}_i + h_i\,\mathbf{t}_i)$$

$$\text{minimize } E = \frac{1}{2}\sum_{j=1}^{M}[\,d_j - \sum_{i=1}^{N}\lambda_i e^{-\|x_j - c_i - h_i t_i\|^2}\,]^2$$

$$\frac{\partial E}{\partial h} = \frac{1}{2}\sum_{j=1}^{M}[d_j - \sum_{i=1}^{N}\lambda_i e^{-\|x_j - c_i - h_i t_i\|^2}][-\sum_{i=1}^{N}\lambda_i e^{-\|x_j - c_i - h_i t_i\|^2} t_i^T (x_j - c_i - h_i t_i)$$

In optimizing the RBF parameter, σ_i, the approach used in optimizing the centers and step size is employed. The error gradient with respect to the parameter, σ_i, corresponding to each basis function, can be calculated using:

$$\text{minimize } E = \frac{1}{2}\sum_{j=1}^{M}[\,d_j - \sum_{i=1}^{N}\lambda_i e^{\frac{-\|x_j - c_i\|^2}{2\sigma_i^2}}\,]^2$$

$$\frac{\partial E}{\partial \sigma_1} = \sum_{j=1}^{M}[\,d_j - \sum_{i=1}^{N}\lambda_i e^{\frac{-\|x_j - c_i\|^2}{2\sigma_i^2}}\,] - \lambda_1 e^{\frac{-\|x_j - c_1\|^2}{2\sigma_1^2}} \frac{\|x_j - c_1\|^2}{\sigma_1^3} = 0$$

$$\frac{\partial E}{\partial \sigma_2} = \sum_{j=1}^{M}[\,d_j - \sum_{i=1}^{N}\lambda_i e^{\frac{-\|x_j - c_i\|^2}{2\sigma_i^2}}\,] - \lambda_2 e^{\frac{-\|x_j - c_2\|^2}{2\sigma_2^2}} \frac{\|x_j - c_2\|^2}{\sigma_2^3} = 0.$$

.
.
.

$$\frac{\partial E}{\partial \sigma_N} = \sum_{j=1}^{M} [\, d_j - \sum_{i=1}^{N} \lambda_i e^{\frac{\|x_j - c_i\|^2}{2\sigma_i^2}} \,] - \lambda_N e^{\frac{\|x_j - c_N\|^2}{2\sigma_N^2}} \frac{\|x_j - c_N\|^2}{\sigma_N^3} = 0$$

The initial starting points of the centers are calculated using the K-means algorithm. After these centers are computed, the initial values of σ_i are set at one half of the distance between each center and the next center location. The flow chart for optimizing σ_i is shown in Figure 4.6. Note that the step size used in this process is the same as that used for determining the centers.

The expansion coefficients are optimized in the sense that they are obtained using the matrix inversion procedure outlined in section 3.3. The overall algorithm can be summarized as follows: (Note that the superscripts denote iteration number.)

Step 1. Start with initial guess $\mathbf{c}^1$ and σ^1.

Step 2. Calculate λ^1.

Step 3. Update $\mathbf{c}^k = \mathbf{c}^{k-1} + h\, \mathbf{t}^k$ and $\sigma^k = \sigma^{k-1} + h\, \mathbf{s}^k$.

Step 4. Calculate λ^k using $\mathbf{c}^k$ and σ^k using the matrix inversion procedure.

Step 5. If the error criterion is satisfied, terminate the process; otherwise, set k=k+1, and go to step 3.

The selection of the "optimal" number of centers and the specific locations, along with the basis function parameters, σ_i, are important since they have a significant impact on the quality of the interpolation algorithm. It is easy to declare each training sample as a center; however, this is impractical as the number of samples become large, as it often does in real-world applications. The goal of the optimization procedure is to minimize the computation effort by minimizing the number of RBF nodes: This is accomplished by manipulating the RBF parameters, such as the centers and σ_i associated with the functions. The widths σ_i of Gaussian functions control the overlap of the functions, thereby establishing the network generalization performance.

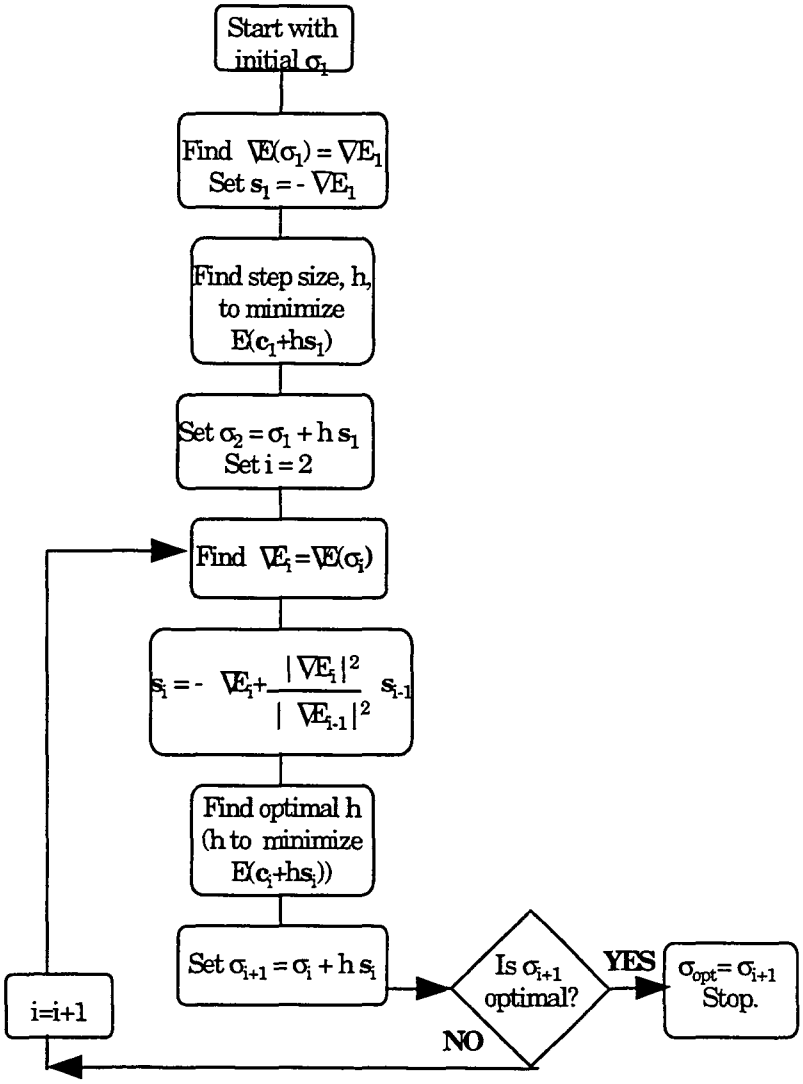

Figure 4.6. Flow chart for optimizing center widths.

CHAPTER 5. RESULTS AND DISCUSSIONS

As part of the initial effort in developing neural network-based schemes for defect characterization, MFL signals were generated using finite element models for a set of rectangular defects described in Table 5.1. As discussed in Chapter 1, the MFL signal needs to be rendered invariant to the effects of changes in magnetization characteristics before being applied to the defect characterization neural network. All the MFL signals was, therefore, preprocessed appropriately [11] before being presented to the defect characterization neural network. The network's performance, in terms of its ability to interpolate the depths and lengths of defects, was evaluated using 2.5" and 3.5" defects at all the depths. The testing data set is described in Table 5.2. The results obtained are shown in Figures 5.1 and 5.2. A remarkable aspect of the results lies in the fact that the neural network manages to predict the length of the flaw accurately without being explicitly told about the relation between the peak-to-peak separation distance and the length of the flaw.

Previously, the network was trained with defect sets that are spaced 1" apart in length and tested with defects that differed by 0.5" in length from the training sample set. In order to assess the performance of the network when a denser training set is used, the network was trained with defects that are spaced 0.5" apart in length as described in Table 5.3.

Table 5.1. Training data set.

	20%	30%	40%	50%	60%	70%	80%
2"	A1	A2	A3	A4	A5	A6	A7
3"	B1	B2	B3	B4	B5	B6	B7
4"	C1	C2	C3	C4	C5	C6	C7

Table 5.2. Testing data set.

	20%	30%	40%	50%	60%	70%	80%
2.5"	D1	D2	D3	D4	D5	D6	D7
3.5"	E1	E2	E3	E4	E5	E6	E7

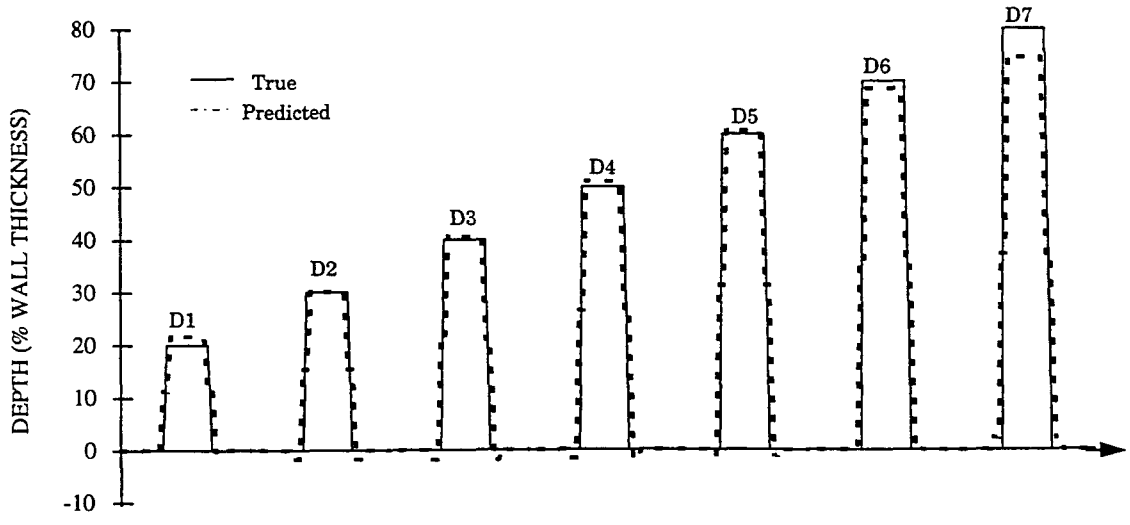

Figure 5.1. True and predicted defect profiles for 2.5" long defect.

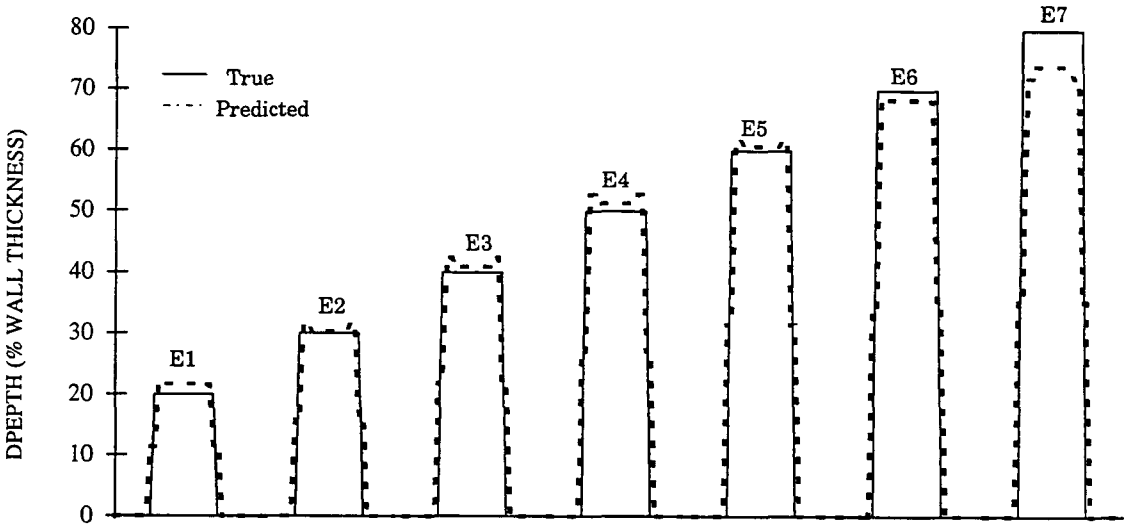

Figure 5.2. True and predicted defect profile for 3.5" long defect.

Table 5.3. Denser training data set.

	20%	30%	40%	50%	60%	70%	80%
2"	A1	A2	A3	A4	A5	A6	A7
2.5"	B1	B2	B3	B4	B5	B6	B7
3"	C1	C2	C3	C4	C5	C6	C7
3.5"	D1	D2	D3	D4	D5	D6	D7
4"	E1	E2	E3	E4	E5	E6	E7

Table 5.4. Denser testing data set.

	20%	30%	40%	50%	60%	70%	80%
2.25"	F1	F2	F3	F4	F5	F6	F7
3.25"	G1	G2	G3	G4	G5	G6	G7

The network's performance is then tested with an even denser defect set that is spaced 0.25" apart in length as summarized in Table 5.4. The results are shown in Figures 5.3 and 5.4. These results clearly indicate that the classification performance improves when the network is trained with a denser set of training samples. Training with signals from a denser defect set minimizes error introduced by quantization which is inherent in modeling the geometry using finite element analysis techniques.

The basis function employed in this initial study was the logarithmic interpolation function given by:

$$\phi(p) = \log(1+p)$$

However, since the primary goal was to improve the network performance, continual efforts were made to identify methods for attaining network improvements. As a result of this effort, alternative interpolation

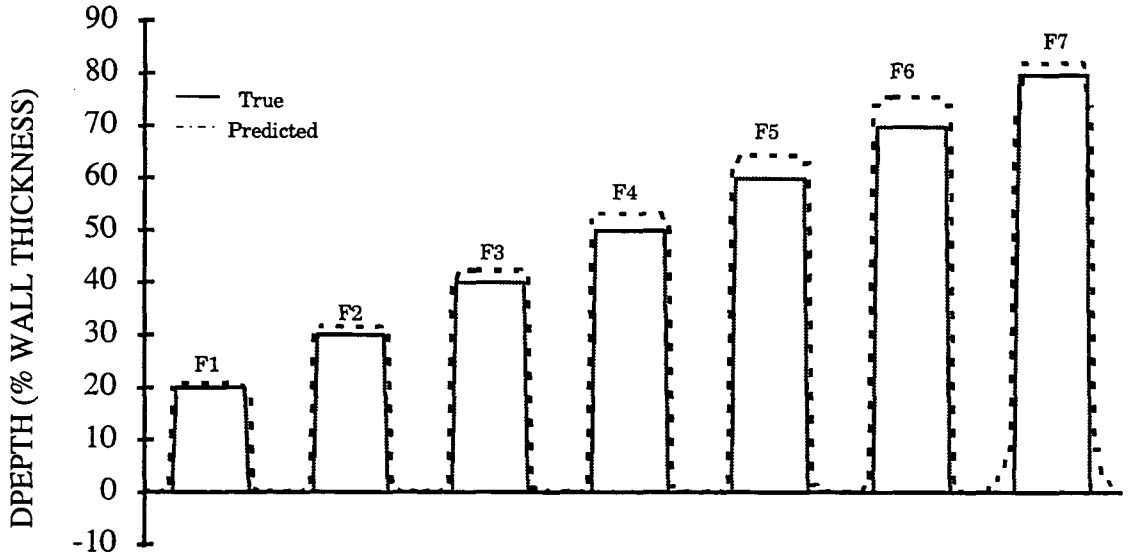

Figure 5.3. Defect profiles predicted by the characterization network for 2.25" long defects.

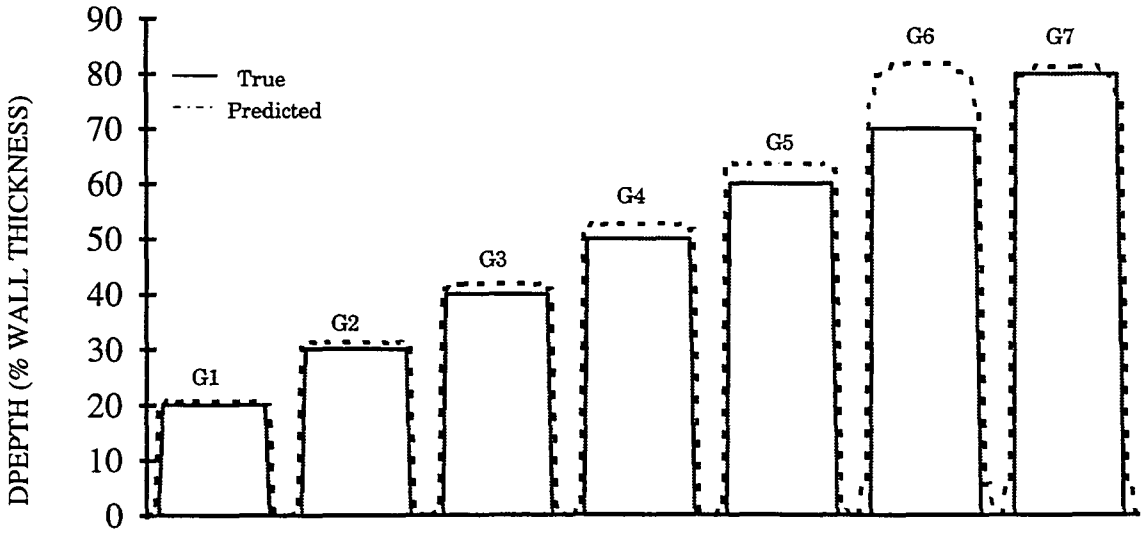

Figure 5.4. Defect profiles predicted by the characterization network for 3.25" long defects.

functions were investigated. These include the Gaussian, multiquadric, and linear interpolation functions. Studies to date indicate that linear interpolation functions offer the best performance. This is quite understandable since the signals used in the defect characterization schemes have all been preprocessed to be invariant to permeability effects. The preprocessing renders the signal substantially linear with respect to the defect dimension. The method used to accomplish this also employed an RBF network prior to presenting them to the defect characterization RBF network. Figure 5.5 shows some of the results obtained using an RBF network which utilized a linear interpolating basis function.

Since the code for computing the basis function centers was based on a Gaussian function, most of the validation studies were carried out using a Gaussian basis function.

Efforts were also devoted towards evaluating and comparing the performance of each of the three methods of determining the basis function parameters. Figures 5.6 through 5.8 show results obtained using the three techniques; namely, the K-means, potential functions approach, and a technique where the centers were selected by sampling the training sample data set. Although the potential function approach appears to offer superior representational accuracy, it typically requires a larger

training data set. Since these results indicate the superiority of the potential functions approach, the procedure was used to implement the defect characterization scheme. The results obtained for defect characterization using this approach are compared with results using the K-means approach in Figure 5.9.

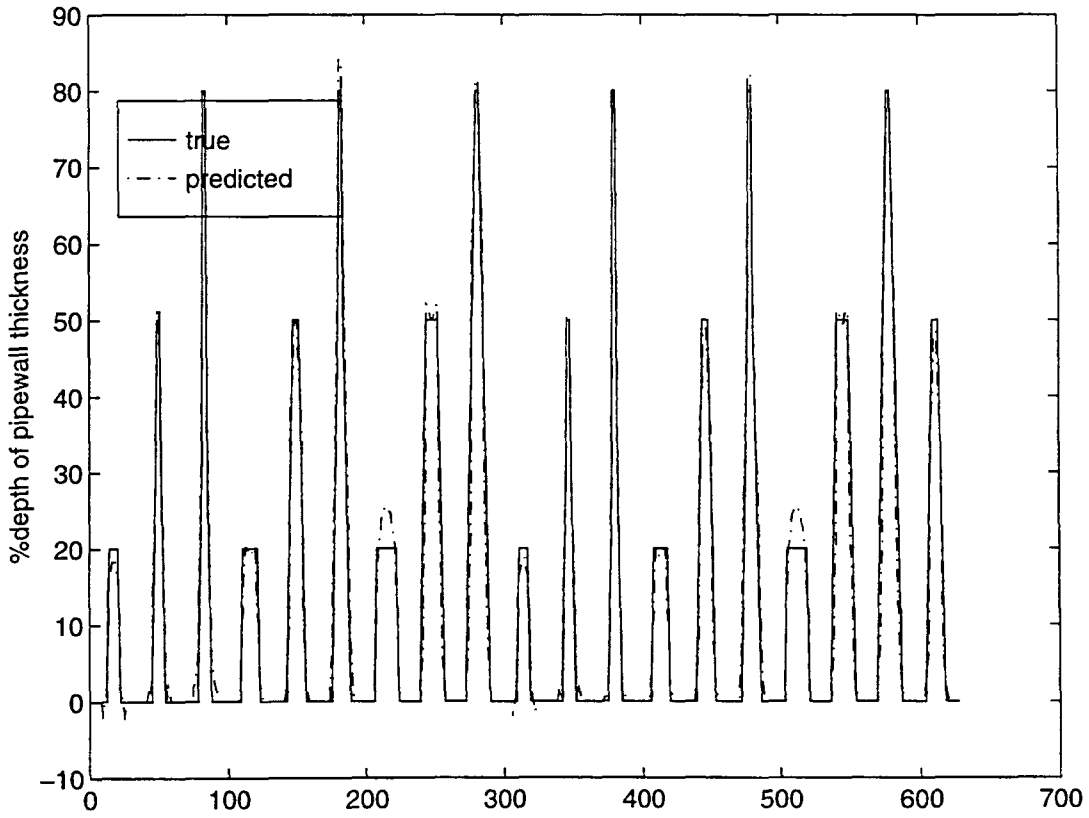

Figure 5.5. Defect characterization results obtained using an RBF network employing linear interpolation functions.

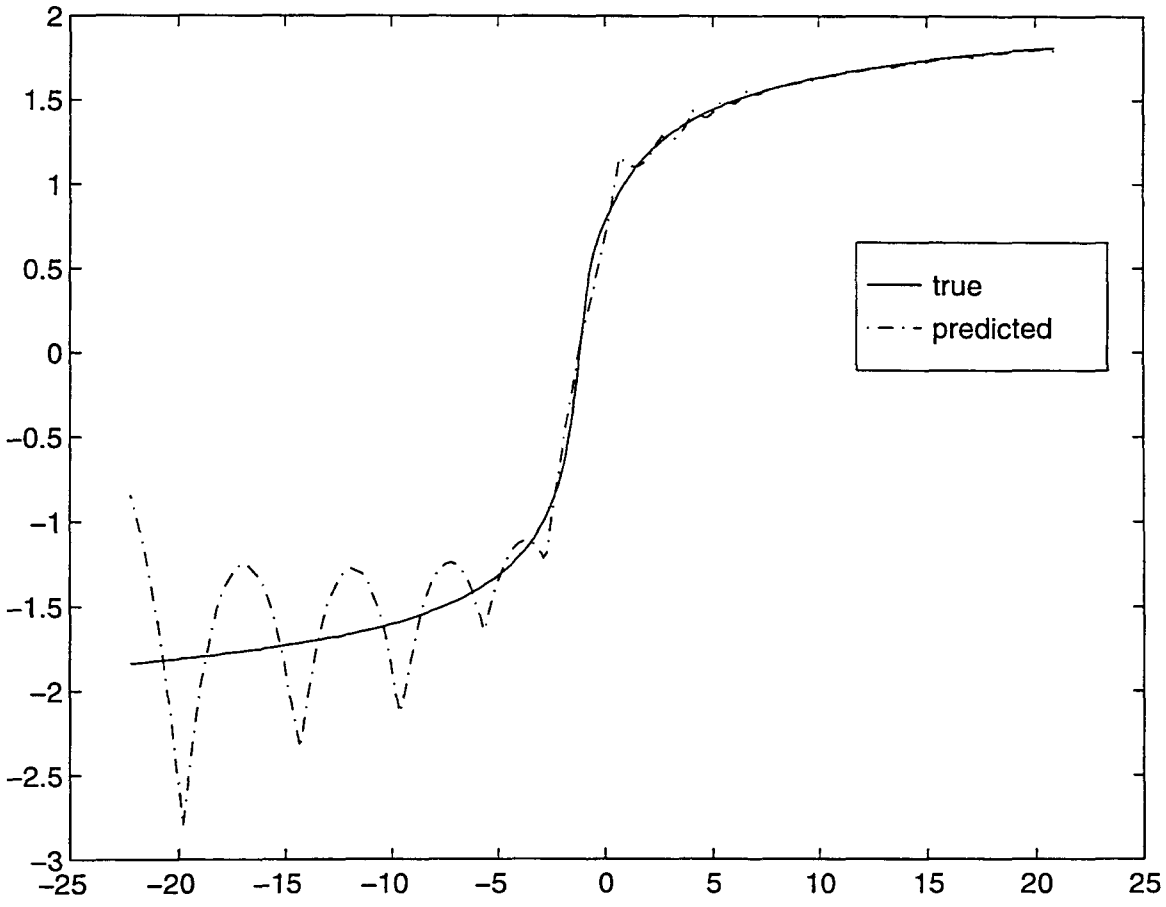

Figure 5.6. Interpolation results obtained using the K-means approach to determine centers for a simple arbitrary nonlinear function.

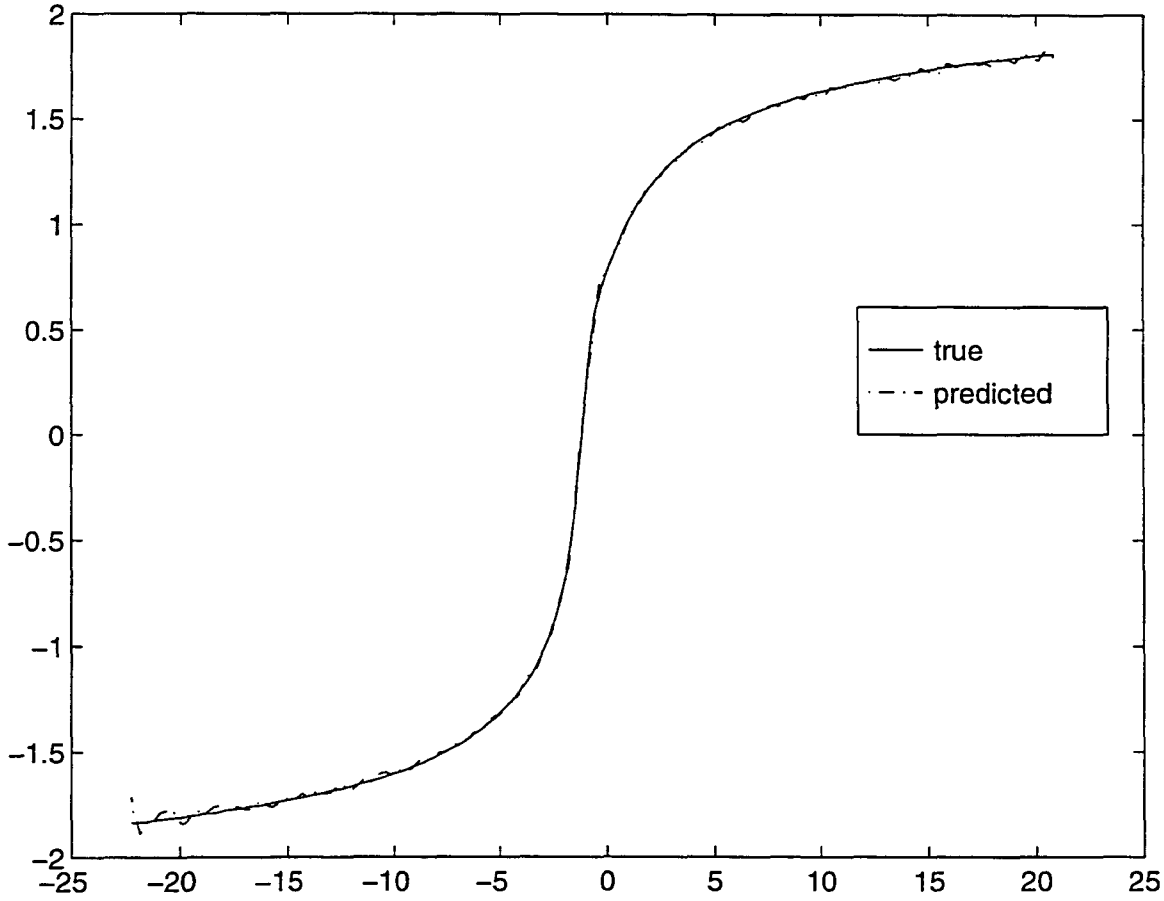

Figure 5.7 Interpolation results obtained using the potential functions approach to determine centers for a simple arbitrary nonlinear function.

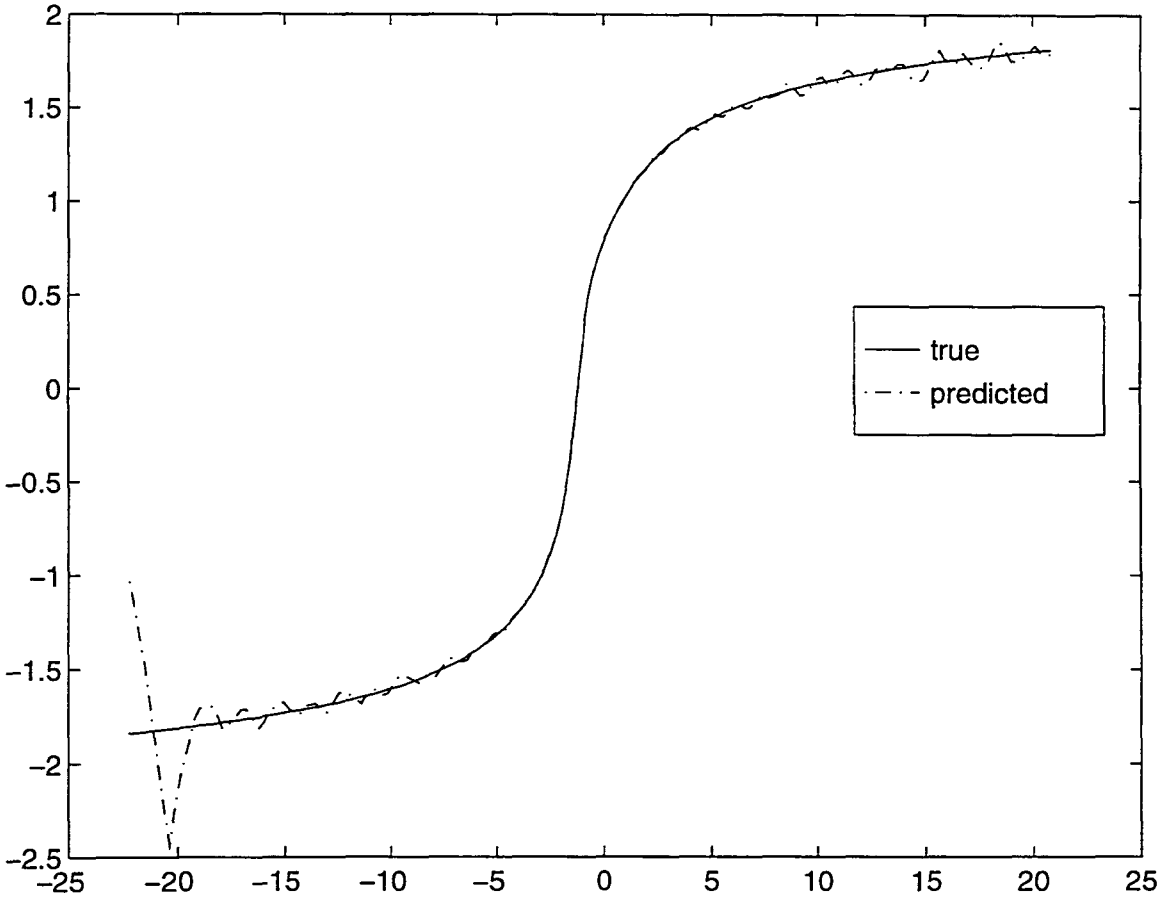

Figure 5.8. Interpolation results obtained using the decimation approach to determine centers for a simple arbitrary nonlinear function.

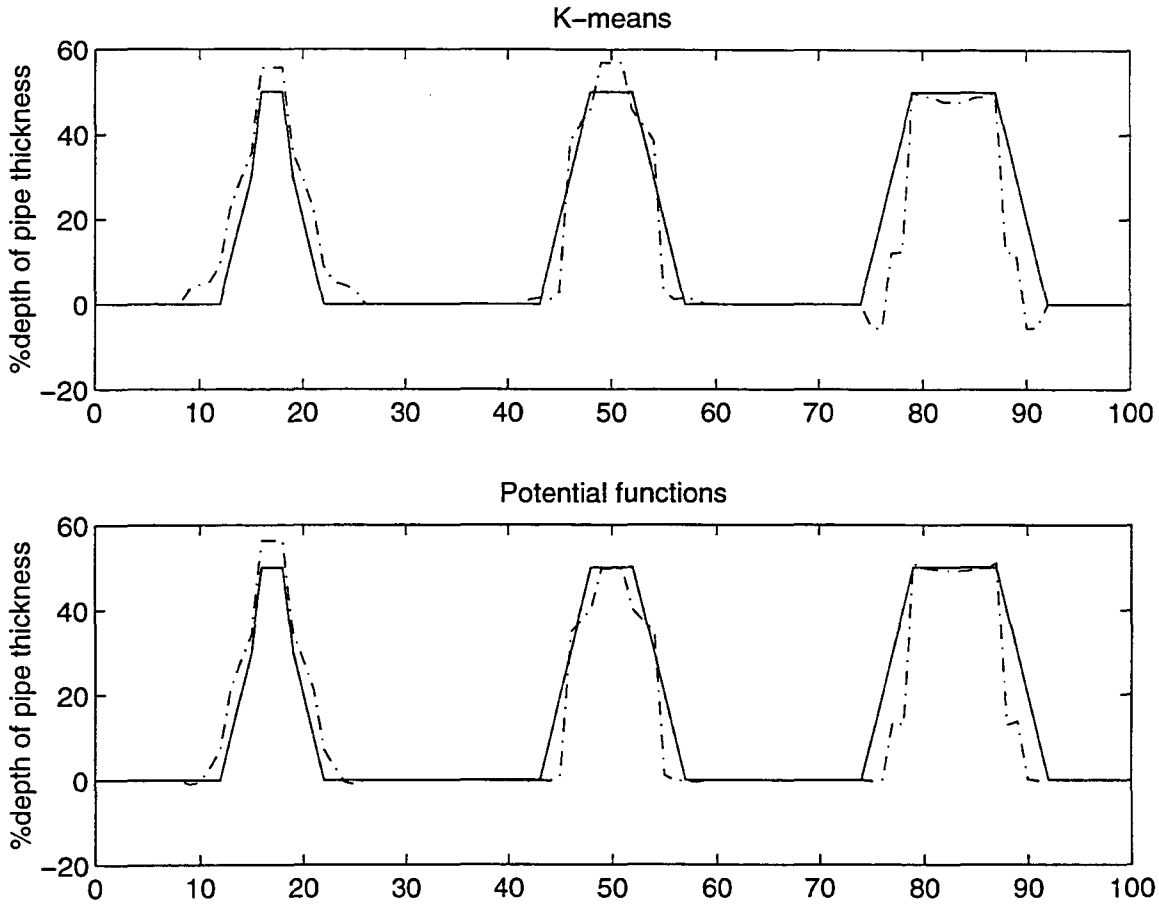

Figure 5.9. Results contrasting the network performance obtained using the K-means approach with the potential functions for selecting centers.

The improvement in results obtained using this network shows that a proper selection of centers can indeed greatly improve the performance of defect characterization scheme.

Finally, the optimal procedure for estimating the basis function parameters and expansion coefficients was implemented. The variables estimated using the approach include the following:

1. Locations of the radial basis function centers
2. Coefficients associated with the radial basis expansion
3. Parameters of the radial basis function

In developing the optimal network, the procedure was implemented in incremental steps starting with the development of software codes that optimized the first two of the three variables. The gradient descent method was used for minimizing the mean squared error. The results obtained using optimal values of these two variables indicated that the procedure yielded performance levels that are superior to the results obtained using the conventional K-means clustering procedure. In fact, the worst performance obtained using the procedure was approximately the same as that obtained using the K-means method.

Figure 5.10(a) shows the performance obtained using the same training and testing data set, and employing parameters that were obtained using the K-means algorithm. Figure 5.10(b) shows the results

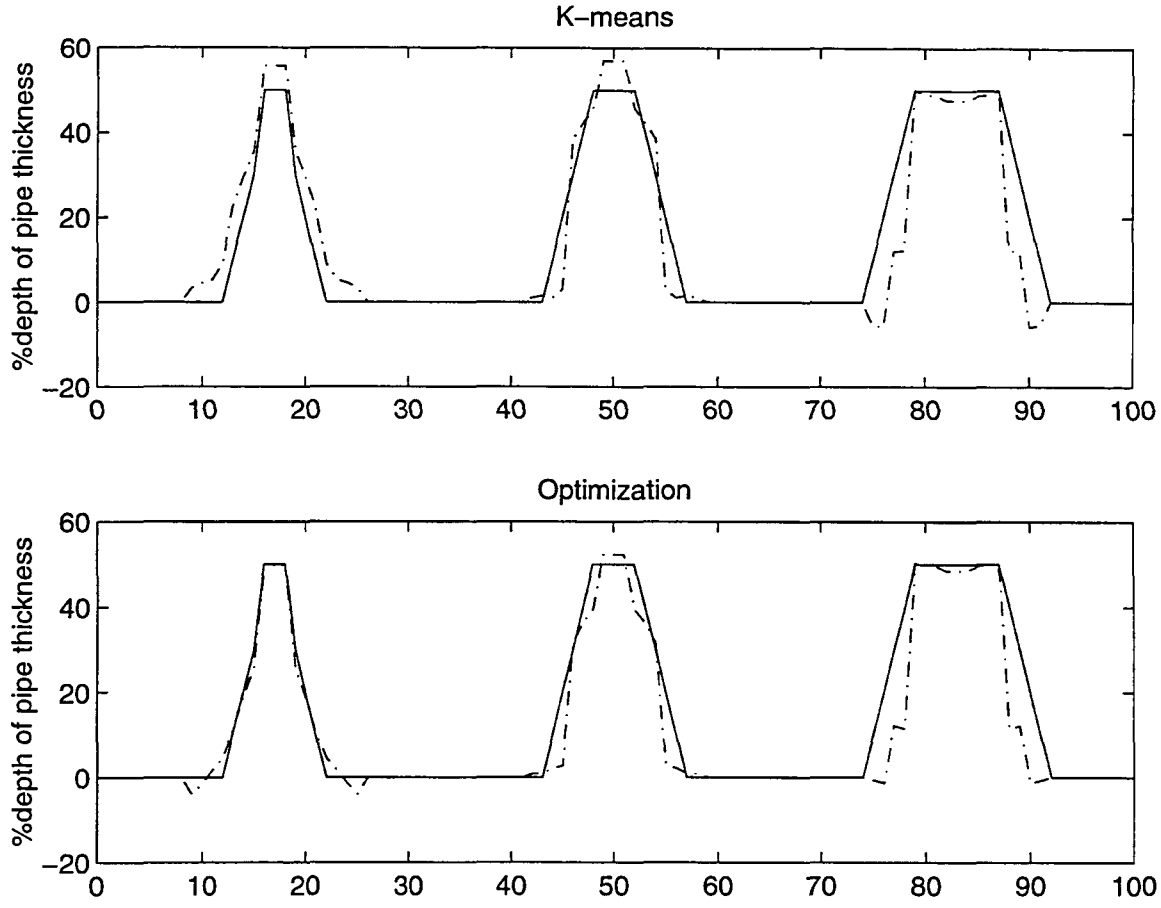

Figure 5.10. True and predicted profiles obtained using the defect characterization network employing (a) K-means algorithm, and (b) Optimal approach for identifying centers.

obtained when all three variables are optimized. The same training and testing data samples were used to illustrate the network performance.

The improvement in performance is evident. Although the optimization technique involves the solution of a set of nonlinear equations, only the training effort is increased. The optimization technique offers results that are superior to those obtained using the conventional K-means or potential functions approach. The increase in defect characterization performance is achieved with fewer number of centers. With decreased number of centers employed, the computational effort in the testing phase is also decreased. The improvement in the characterization performance comes at the expense of increased training times, which is typically not of any great consequence.

The optimization network performed better than those obtained using the potential functions or the K-means approach when the training and testing data set is the same. At times, the optimization network seems to perform slightly poorly when novel testing data samples are presented. The problem may be due to the fact that the result obtained may be suboptimal and that the iterative procedure may have been trapped in a local minimum. The procedure employed does not guarantee that a global minimum has been reached.

Finally, the defect characterization network was evaluated using actual experimental data. These results are preliminary due to the limited availability of experimental data. Thirteen defects were chosen out of the limited data set available. The three-dimensional defect parameters are defined in Figure 5.11. A fair amount of experimental data had to be discarded due to their poor quality. The network was trained with twelve defect signals and tested with the remaining signal. Figure 5.12 shows the defect profile predicted by the network of a defect that was not a member of the training set where the network only utilized six centers with twelve training samples. Figure 5.13 shows the same results, but where the network used twelve centers. This illustrates the point that, although, the network performance is better with increased centers, we can alleviate the tradeoff between characterization accuracy and computational complexity if the proper method of center selection is used. Therefore, an increase in defect characterization may be obtained simultaneously with a decrease in number of RBF nodes.

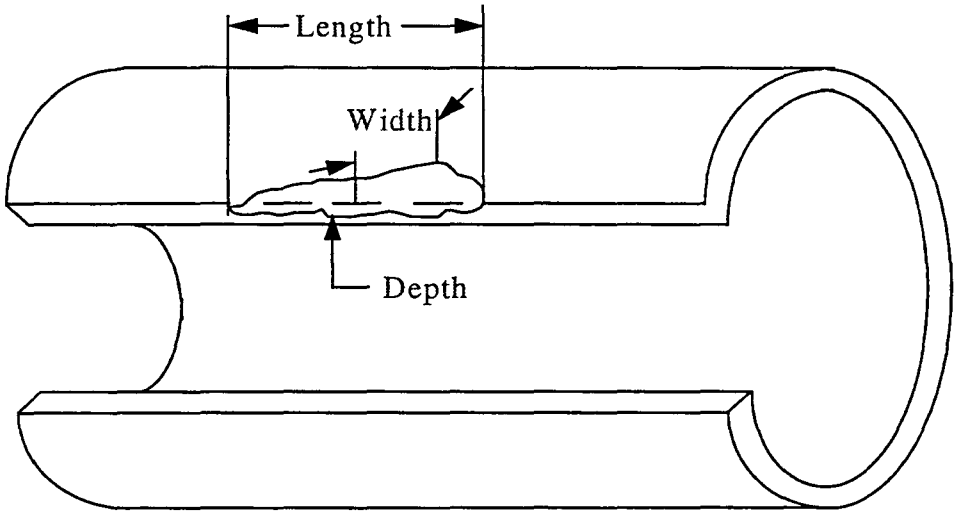

Figure 5.11. Parameters of three-dimensional defect.

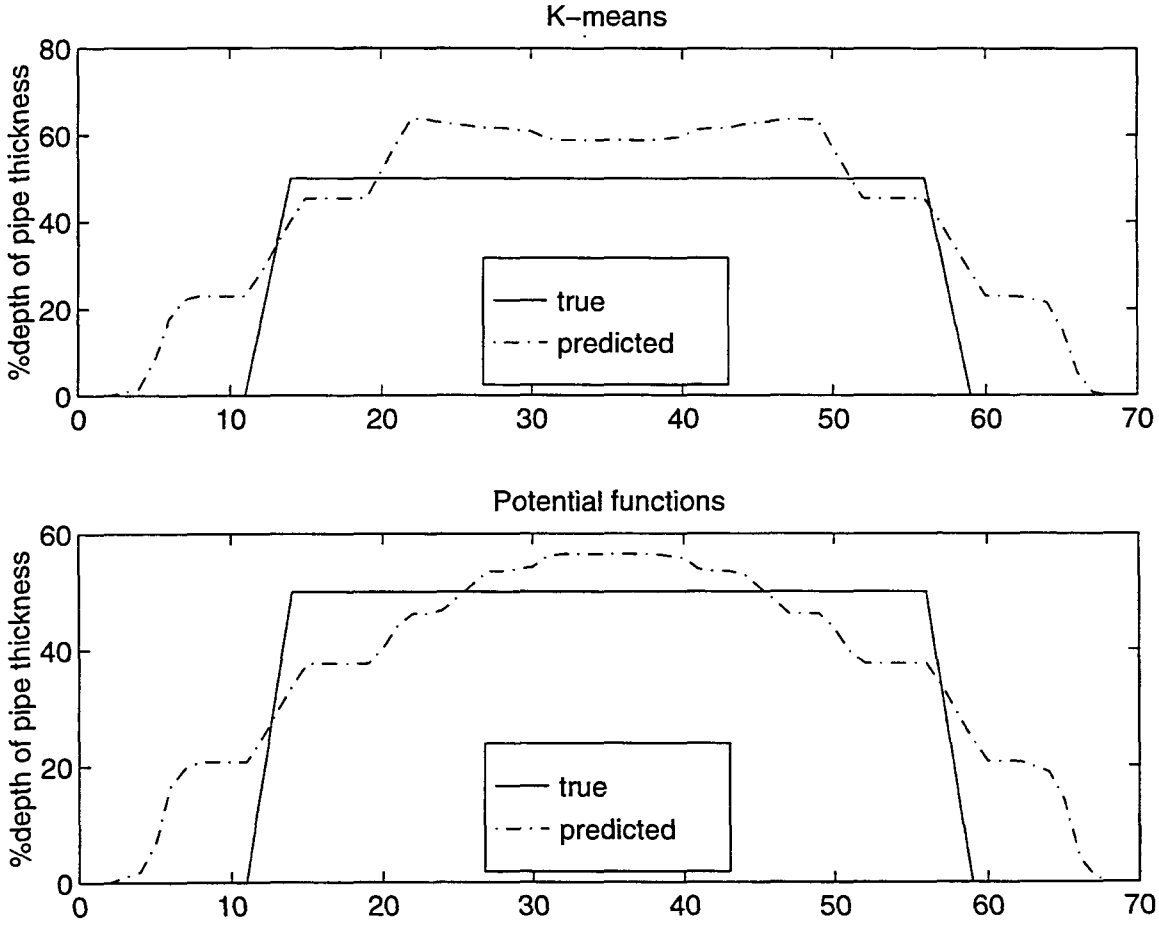

Figure 5.12. K-means approach compared with potential functions approach for center selection: Expanded view of the predicted profile of a defect that was not a member of the training set using 6 centers and 12 training samples.

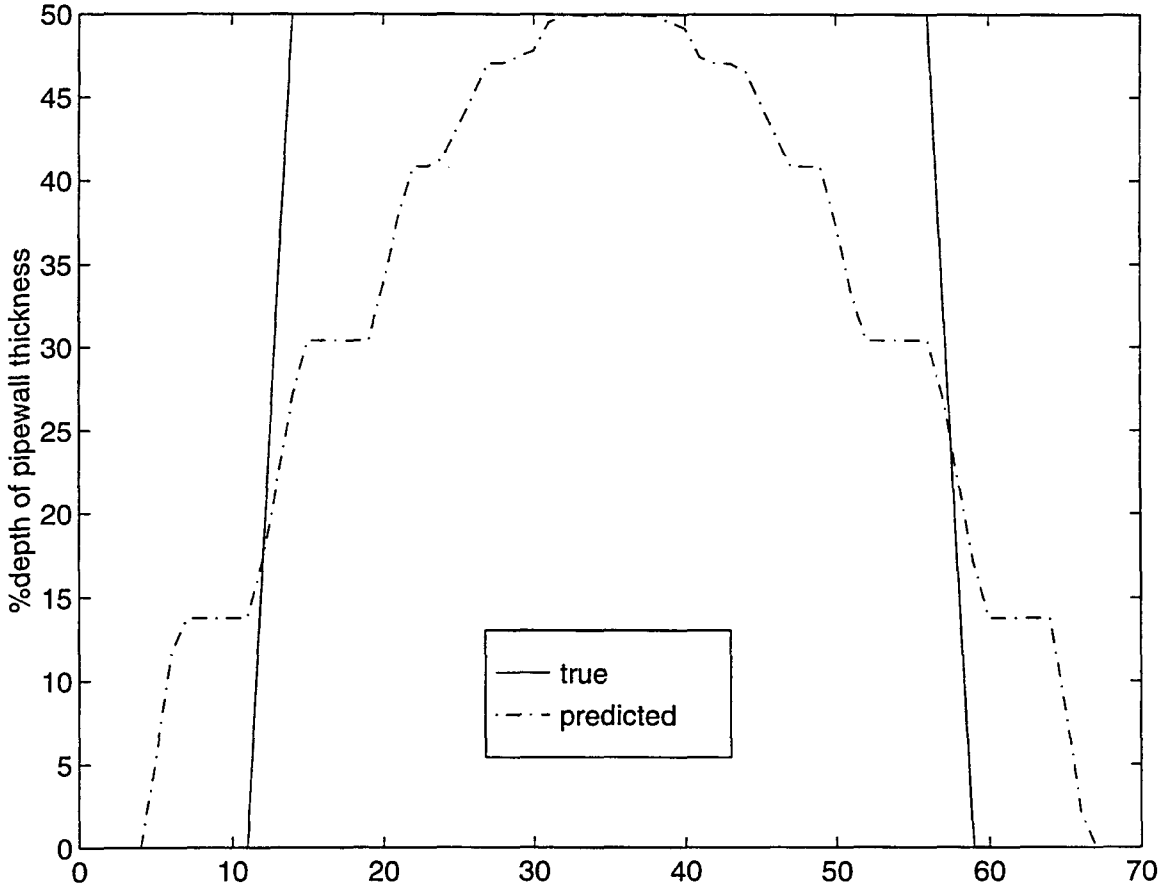

Figure 5.13. Expanded view of the predicted profile of a defect that was not a member of the training set using 12 centers and 12 training samples.

CHAPTER 6. SUMMARY AND FUTURE WORK

6.1 Summary

This thesis focuses on methods for characterizing MFL signals using artificial neural networks. To gain an appreciation and understanding of the problem, the thesis begins with a problem statement and a brief background and the motivation prompting this research. This is followed by a description of the method used for solving the problem. A brief introductory background of the finite element method and its use for studying the underlying physical process and generating training data is presented.

Next, a detailed discussion of artificial neural networks is presented. Two of the more popular neural networks; namely, the radial basis functions and the multilayer perceptrons are described.

A brief discussion of various methods used in selecting centers that are used in designing the RBF networks is presented. Without *a priori* knowledge, the centers are usually chosen by sampling the training data set. Another traditional approach involves the use of a K-means clustering algorithm to calculate the centers required by the RBF network. This thesis presents alternative methods for selecting the centers. In particular, two new methods: a potential functions approach and an optimal technique are proposed. The superiority of these novel

approaches are shown. In addition, comparisons of the performance obtained using various center selection techniques are made.

6.2 Future Work

Future work in the area should focus on evaluating the neural network performance using more experimental data. Most of the results presented in this thesis were obtained using MFL signals generated using the finite element model. The next step is to test the neural network with a more extensive set of experimental data. The network needs to be evaluated using signals from more complicated defect geometries, also. Studies done to date indicate the necessity of using a three-dimensional neural network.

Also, in all the results presented, the neural network performance was evaluated visually in terms of how closely the predicted defect profile fit that of the desired defect profile. This method is rather subjective and hence, a more substantial or quantitative means of determining the network performance is needed. Attention should also be devoted to methods for estimating the confidence interval for the predicted results. This will aid the pipeline vendors in evaluating the quality of the characterization results and assist them in arriving at the proper remediation efforts.

BIBLIOGRAPHY

[1] J. Hwang, *Defect characterization of magnetic leakage fields*, Ph. D. Dissertation, Colorado State University, Fort Collins, Colorado, 1975.

[2] W. Lord, Class notes, EE448x, Spring semester, Iowa State University, 1994.

[3] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109-118, 1990.

[4] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing,* vol. 1, Cambridge, MA: MIT Press., 1986.

[5] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. on Neural Networks*, vol. 2, no. 2, pp. 302-309, March 1991.

[6] L. Xu, A. Krzyzak, and A. Yuille, "On radial basis function nets and kernel regression: Statistical consistency, convergence rates, and receptive field size," *Neural Networks*, vol. 7, no. 4, pp. 609-628,1994.

[7] D. S. Broomhead and D. Lowe, "Multivariate functional interpolation and adaptive networks," *Complex Systems*, vol.2, pp. 321-355, 1988.

[8] S. Nair, S. Udpa, and L. Udpa, "Radial basis functions network for defect sizing," Proceedings of the Review of Progress in Quantitative Nondestructive Evaluation, San Diego, 1992.

[9] M. Chao, S. Udpa, L. Udpa, and W. Lord, "Characterization of magnetic flux leakage signals using radial basis function network," Proceedings of the 3rd annual Midwest Electro-Technology Conference, pp. 118-121, April 1994.

[10] G. J. Posakony, *Topical Report*, Assuring the Integrity of Natural Gas Transmission Pipelines, Gas Research Institute, Nov., 1992.

[11] S. Mandayam, L. Udpa, S. Udpa, and W. Lord, "New methods for processing magnetic flux leakage signals in NDE applications," Symposium on Adv. in Measur. Tech. and Instr. for Mag.Proper. Determination, pp. 93-102, May 1994.

[12] K. H. Huebner, *The Finite Element Method for Engineers*, John Wiley & Sons, Inc., New York, 1975.

[13] R. Wait and A. R. Mitchell, *Finite Element Analysis and Applications*, John Wiley & Sons, New York, 1985.

[14] J. Jin, *The Finite Element Method in Electromagnetics*, John Wiley & Sons, Inc., New York, 1993.

[15] W. Lord and J. H. Hwang, "Defect characterization from magnetic leakage fields," British Journal of NDT, pp. 14-18, Jan., 1977.

[16] R. P. Lippmann, "An Introduction to Computing with Neural Nets,"*IEEE Acoust., Speech and Signal Processing Mag.,* vol. 61, pp.4-22, 1987.

[17] S. Shekhar, M. B. Amin, and P. Khandelwal, "Generalization Performance of Feed-Forward Neural Networks," in Neural Networks: Advances and Applications 2, E. Gelenbe, Ed., Elsevier Science Publishers B. V., The Netherlands, pp. 13-38, 1992.

[18] B. Muller and J. Reinhart, *Neural Networks: An Introduction*, Springer-Verlag, Berlin Heidelberg, 1990.

[19] G. E. Hinton, "Learning distributed representation of concepts," in *Proc. 8th Annual Conf. Cognitive Science Society,* Amherst, MA, pp.1-12, 1986.

[20] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," in Algorithms for Approximation, J. C. Madison and M. G. Cox, Eds., Oxford, pp. 143-167, 1987.

[21] S. Haykin, *Neural Networks, A Comprehensive Foundation*, Macmillan College Publishing Company, Inc., New Jersey, 1994.

[22] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley Publishing Company, Inc., Massachusetts, 1974.

[23] S. S. Rao, *Optimization Theory and Applications*, Wiley Eastern Limited, India, 1978.

APPENDIX

Program Listing:

```
C  K-MEANS ALGORITHM FOR CENTER SELECTION
     parameter (numsam=12, vsize=33, classes=12, out=33)
     double precision    x(numsam,vsize), center(classes,vsize),
+                        dist, mindist, dum(classes,vsize),dum1
     integer             iclass(numsam), n(classes)

     open(7,file='train_sample',status='unknown')
     open(8,file='cluster_center',status='unknown')

c  read in input data
     do i=1,numsam
      do j=1,vsize
       read(7,*) x(i,j)
      enddo
      do j=1,out
       read(7,*) dum1
      enddo
     enddo

c  initialize cluster centers to first "classes" number of input sample
     do i=1,classes
      do j=1,vsize
       center(i,j) = x(i,j)
      enddo
     enddo

c  calculate distances and find the minimum distance
10   do i=1,numsam
       mindist = 1E18
       do j=1,classes
        dist=0.0
        do k=1,vsize
         dist = dist + (x(i,k) - center(j,k))**2
        enddo
        dist = sqrt(dist)
```

```
        if (dist.lt.mindist) then
          mindist=dist
          iclass(i) = j
        endif
      enddo
    enddo

c  count number of samples within a class
    do i=1,classes
      n(i) = 0
      do j=1,numsam
        if (iclass(j).eq.i) n(i) = n(i) + 1
      enddo
    enddo

c  save cluster centers first, then zero out clusters
    do i=1,classes
      do j=1,vsize
        dum(i,j) = center(i,j)
        center(i,j) = 0.0
      enddo
    enddo

c  create new cluster center
    do i=1,classes
      do j=1,numsam
        if (iclass(j).eq.i) then
          do k=1,vsize
            center(i,k) = center(i,k) + (x(j,k)/n(i))
          enddo
        endif
      enddo
    enddo

c  check for convergence
    m = 0
    do i=1,classes
      do j=1,vsize
        if (center(i,j).ne.dum(i,j)) m = 1
      enddo
    enddo
```

```
      if (m.ne.0) goto 10
c  write center to file "cluster_center"
      do i=1,classes
        do j=1,vsize
          write(8,*) center(i,j)
        enddo
        write(8,*) ' '
      enddo

      end


C ***************************************************************
C  RADIAL BASIS FUNCTION ALGORITHM FOR TRAINING

c   ni = number of input nodes
c   nh = number of hidden nodes
c   no = number of output nodes
c   nsam = number of samples

      parameter (ni=33, nh=1, no=33, nsam=2)
      double precision    A(nsam,nh), S(nsam,no), x(nsam,ni), lamb(nh,no),
+                         dum1(nh,nh), dum2(nh,nsam), Atran(nh,nsam),
+                         p, center(nh,ni), b(nh), dinv(nh,nh), z(nh), rcond
      integer             job, ipvt(nh)

c   Gaussian basis function
c         rbf(p) = exp(-p)
c   Logarithmic basis function
c         rbf(p) = log10((1+p)*4.)
c   Multi-quadric
c         rbf(p) = (p**2 + c**2)**0.5
c   Linear
c         rbf(p) = p

      open(7,file='train_sample',status='unknown')
      open(8,file='cluster_center',status='unknown')

      open(10,file='lambda',status='unknown')
```

```
c   read in data points and S outputs
    do i=1,nsam
     do k=1,ni
      read(7,*) x(i,k)
     enddo
     do k=1,no
      read(7,*) s(i,k)
     enddo
    enddo

c   read in cluster centers
    do i=1,nh
     do k=1,ni
      read(8,*) center(i,k)
     enddo
    enddo

c   calculates A-matrix:  A=phi(||x-c||)
    do i=1,nsam
     do j=1,nh
      do k=1,ni
       A(i,j) = (x(i,k) - center(j,k))**2  + A(i,j)
      enddo
     A(i,j) = rbf(sqrt(A(i,j)))
     enddo
    enddo

c   calculates the transpose of A-matrix
    do i=1,nsam
     do j=1,nh
      Atran(j,i) = A(i,j)
     enddo
    enddo

c   multiplies transpose of A-matrix by A-matrix
    do i=1,nh
     do j=1,nh
      do k=1,nsam
       dum1(i,j) = dum1(i,j) + Atran(i,k)*A(k,j)
      enddo
     enddo
```

```
      enddo

c     calculate the inverse matrix of dum1
      call dgeco(dum1,nh,nh,ipvt,rcond,z)
      rconda=rcond
      do 20 i=1,nh
      do 25 j=1,nh
       b(j)=0.0
25    continue

      b(i)=1.0
      call dgesl(dum1,nh,nh,ipvt,b,job)
      do 22 jj=1,nh
       dinv(jj,i)=b(jj)
22    continue
20    continue

c     multiplies the result by A-matrix transpose
      do i=1,nh
       do j=1,nsam
        do k=1,nh
         dum2(i,j) = dum2(i,j) + dinv(i,k)*Atran(k,j)
        enddo
       enddo
      enddo

c     calculates lambda (expansion coefficients) and write to file
      do i=1,nh
       do j=1,no
        do k=1,nsam
         lamb(i,j) = lamb(i,j) + dum2(i,k)*S(k,j)
        enddo
       write(10,*) lamb(i,j)
       enddo
      enddo

      end


C ***************************************************************
C  RADIAL BASIS FUNCTION ALGORITHM FOR TESTING
```

```
c   ni = number of input nodes
c   nh = number of hidden nodes
c   no = number of output nodes
c   nsam = number of samples

    parameter (ni=33, nh=1, no=33, nsam=1)
    double precision    A(nsam,nh), S(nsam,no), x(nsam,ni), p,
+                       lamb(nh,no), center(nh,ni), t, twe(nsam,no)

c   Gaussian basis function
c           rbf(p) = exp(-p)
c   Logarithmic basis function
c           rbf(p) = log10((1+p)*4.)
c   Multi-quadric
c           rbf(p) = (p**2 + c**2)**0.5
c   Linear
c           rbf(p) = p

c   open files
    open(7,file='test_sample',status='unknown')
    open(8,file='cluster_center',status='unknown')
    open(9,file='lambda',status='unknown')

    open(10,file='output',status='unknown')

c   read in test data
    do i=1,nsam
     do k=1,ni
      read(7,*) x(i,k)
     enddo
     do k=1,no
      read(7,*) twe(i,k)
     enddo
    enddo


c   read in cluster centers
    do i=1,nh
     do k=1,ni
      read(8,*) center(i,k)
     enddo
    enddo
```

```
c   read in expansion coefficients (lambda)
    do i=1,nh
     do j=1,no
      read(9,*) lamb(i,j)
     enddo
    enddo

c   calculate A-matrix:  A=phi(||x-c||)
    do i=1,nsam
     do j=1,nh
      do k=1,ni
       A(i,j) = A(i,j) + (x(i,k) - center(j,k))**2
      enddo
     A(i,j) = rbf(sqrt(A(i,j)))
     enddo
    enddo

c   calculate S-output
    do i=1,nsam
     do j=1,no
      do k=1,nh
       S(i,j) = S(i,j) + A(i,k)*lamb(k,j)
      enddo
     enddo
    enddo

c  write predicted output to file
    do i=1,nsam
     do j=1,no
      write(10,*) twe(i,j), S(i,j)
     enddo
     write(10,*) ' '
    enddo

    end

C ***************************************************************
C  POTENTIAL FUNCTIONS ALGORITHM FOR CENTER SELECTION
c   numsam = number of training samples
c   vsize = vector size of each input sample (input nodes)
c   classes = number of centers (hidden nodes)
```

```
c   out = vector size of each desired output (output nodes)

    parameter (numsam=12,vsize=33,classes=6,out=33)
    double precision x(numsam,vsize),center(classes,vsize),dist,mindist,
+                        dum(classes,vsize), dum1
    real  thresh, tmp
    integer  iclass(numsam), n(classes)

c   various potential functions
    pot(q) = exp(-q)
c   pot(q) = 1/(1+q)

c   open input file: contains training data samples
    open(7,file='training_data',status='unknown')
c   open output file: will contain centers after program execution
    open(8,file='testing_data',status='unknown')

c   read in error threshold
    print*,'enter threshold value'
    read(*,*) thresh

c   read in data
c   file format containing training samples is such that the first vector of
c   data is the input sample vector followed by the corresponding desired
c   output vector.  Then the next training sample input vector followed by
c   the corresponding output vector is listed.  So on and so forth...
    do i=1,numsam
     do j=1,vsize
       read(7,*) x(i,j)
     enddo
     do k=1,out
      read(7,*) tmp
     enddo
    enddo

c   initialize potential function to be that of the first training sample
    do j=1,vsize
     center(1,j) = x(1,j)
    enddo
c   calculate potential function of each sample
    m=1
    do i=1,numsam
```

```
c   skip the first sample since it's already been initialized to this
    if (i.eq.1) goto 12

c   initialize cumulative potential to zero
    pf=0.0
    do j=1,m
     q=0.0
     do k=1,vsize
      q=q+(x(i,k)-center(j,k))**2
     enddo
c   compute the potential function of each sample
     pf=pf+pot(q)
    enddo

c   determine if the sample vector is retained as a center vector
c   ie., if the quantity exp{-||x-xc||} is greater than the error threshold,
c   (meaning that x is close to the existing center xc) then discard the
c   training sample; otherwise, keep sample as center and progress to next
c   sample
    if (pf.gt.thresh) goto 12
    m=m+1
    do k=1,vsize
c   keep sample as center
     center(m,k)=x(i,k)
    enddo
12  continue
    enddo

    print*,'number of centers = ',m

c   write centers to file
    do i=1,m
     do j=1,vsize
      write(8,*) center(i,j)
     enddo
     write(8,*) ' '
    enddo

    end

C  OPTIMIZATION PROGRAM
c   optimized center locations and widths of the radial basis functions,
```

```
c   along with the step length used in updating those two parameters.
c   the expansion coefficients are optimized in the sense that they
c   correspond to the optimal center locations.
c   nsam = number of training samples
c   vsize = vector size of the sample patterns
c   ncen = number of centers
c   itmax = maximum number of iterations

      parameter (nsam=19,vsize=33,ncen=8,itmax=200)
      double precision  c(ncen,vsize),oldc(ncen,vsize),lambda(ncen,vsize),
     +          oldlamb(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
     +          pred(nsam,vsize),delf(ncen,vsize),oldelf(ncen,vsize),
     +          oldelsig(ncen),sdir(ncen,vsize),sigdir(ncen),delsig(ncen),toler,
     +          h,oldsigma(ncen),sigma(ncen),delta
      double precision  cencon,lamcon,matnorm,step,error,olderror

c     open input file
      open(7,file='training_data',status='unknown')
      open(8,file='optimum_center',status='unknown')
      open(9,file='optimum_lambda',status='unknown')
      open(10,file='initial_sigma',status='unknown')

c     error criterion
      toler = 1e-1
c     initial step size
      h=.005
c     scale factor for updating optimal step length
      step = .0001
c     read in initial sigmas
      do j=1,ncen
       read(10,*) sigma(j)
      enddo
c     read in data file and scale appropriately
      do i=1,nsam
       do k=1,vsize
        read(7,*) x(i,k)
        x(i,k)=x(i,k)/1e2
       enddo
       do k=1,vsize
        read(7,*) s(i,k)
        s(i,k)=s(i,k)/1e2
       enddo
```

```
      enddo
      rewind(7)

c     read in initial guesses of centers
      do j=1,ncen
       do k=1,vsize
        read(8,*) c(j,k)
       enddo
      enddo

c     compute corresponding lambda's
      call train(c,x,s,lambda,sigma)

      iter=0
50 continue

c     find error gradient for first iteration
      call gradfun(delf,c,lambda,s,x,sigma)
      call gradsig(c,lambda,s,x,sigma,delsig)

c     save gradient
      do j=1,ncen
       do k=1,vsize
        oldelf(j,k)=delf(j,k)
       enddo
      oldelsig(j)=delsig(j)
      enddo

c     initialize the first search direction
      do j=1,ncen
       do k-1,vsize
        sdir(j,k)=-delf(j,k)
       enddo
      sigdir(j)=-delsig(j)
      enddo

c     compute optimal step size which minimizes cost function
      call hupdate(h,c,lambda,s,x,sdir,step,sigma)

c     update centers
      call update(c,sdir,h,sigma,sigdir,delta)
```

```
c   compute corresponding lambda's
    call train(c,x,s,lambda,sigma)

c   begin iteration
10  continue
    iter=iter+1
    olderror = error

c   compute new gradient at updated values
    call gradfun(delf,c,lambda,s,x,sigma)
    call gradsig(c,lambda,s,x,sigma,delsig)
c   save centers, sigmas, and lambdas
    do j=1,ncen
    do k=1,vsize
     oldc(j,k)=c(j,k)
     oldlamb(j,k)=lambda(j,k)
    enddo
     oldsigma(j)=sigma(j)
    enddo

c   compute matrix norm of delf and oldelf with ratio stored in matnorm
    call norm2(oldelf,delf,matnorm)

c   compute new search direction
    do j=1,ncen
     do k=1,vsize
       sdir(j,k)=-delf(j,k)+sdir(j,k)*matnorm
     enddo
    sigdir(j)=-delsig(j)+(delsig(j)/oldelsig(j))*sigdir(j)
    oldelsig(j)=delsig(j)
    enddo

c   compute optimal step size
    call hupdate(h,c,lambda,s,x,sdir,step,sigma)

c   update centers and sigmas
    call update(c,sdir,h,sigma,sigdir,delta)

c   compute lambdas
    call train(c,x,s,lambda,sigma)

c   compute error function value
```

```
      call fun(error,c,lambda,s,x,sigma)

c     check for convergence
      if(error.gt.toler .and. iter.le.itmax) goto 10

c     write centers and lambdas to files
      do j=1,ncen
      do k=1,vsize
       write(8,*) oldc(j,k), c(j,k)
       write(9,*) oldlamb(j,k), lambda(j,k)
      enddo
      enddo
print*, 'total number of iterations =  ',iter

stop
end

c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c     gradsig: find width gradient
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      subroutine gradsig(c,lambda,s,x,sigma,delsig)
      parameter(nsam=19,vsize=33,ncen=8)
      double precision c(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
+           lambda(ncen,vsize),delsig(ncen),sigma(ncen),sigdir(ncen),
+           sigsign(ncen),pred(nsam,vsize),deriv(nsam,vsize),
+           xc(nsam,ncen),A(nsam,ncen),fac(nsam,ncen)

c     clear all arrays
      do i=1,nsam
       do j=1,ncen
         delsig(j)=0.0
         do k=1,vsize
          pred(i,k)=0.0
          deriv(i,k)=0.0
         enddo
         xc(i,j)=0.0
         A(i,j)=0.0
         fac(i,j)=0.0
       enddo
      enddo

      do i=1,nsam
```

```
      do j=1,ncen
       do k=1,vsize
        xc(i,j)=xc(i,j) + (x(i,k)-c(j,k))**2
       enddo
       A(i,j)=xc(i,j)/2*(sigma(j)**2)
       A(i,j)=exp(-A(i,j))
       fac(i,j)=xc(i,j)/(sigma(j)**3)
      enddo
     enddo

     do i=nsam
      do j=1,ncen
       do k=1,vsize
        pred(i,k)=pred(i,k)+A(i,j)*lambda(j,k)
        deriv(j,k)=deriv(j,k)+A(i,j)*lambda(j,k)*fac(i,j)
       enddo
      enddo
     enddo

     do j=1,ncen
      do i=1,nsam
       do k=1,vsize
        delsig(j)=delsig(j)+(s(i,k)-pred(i,k))*deriv(j,k)*(-1)
       enddo
      enddo
     enddo

     return
     end
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c    hupdate: calculate optimal step size, h
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
     subroutine hupdate(h,c,lambda,s,x,sdir,step,sigma)
     parameter (nsam=19,vsize=33,ncen=8)
     double precision c(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
+            lambda(ncen,vsize),step,sigma(ncen),h,deleh,search,
+            sdir(ncen,vsize),tol,oldh
     integer itt

     tol = 1e-2
     imatit = 50
     itt = 0
```

```
c   compute gradient wrt to h
    call gradh(h,c,lambda,s,x,sdir,deleh,sigma)

c   initialize search direction
    search=-deleh

    if(search.lt.0.0) then
          sign=-1
    else if(search.gt.0.0) then
          sign=1
    else
          sign=0.0
    endif
    h=abs(h+step*sign)

12  continue
    itt=itt+1
    oldh=h
    call gradh(h,c,lambda,s,x,sdir,deleh,sigma)

    search=-deleh+abs(h/oldh)*search
    if(search.lt.0.0) then
          sign=-1
    else if(search.gt.0.0) then
          sign=1
    else
          sign=0.0
    endif

    h=abs(h+step*sign)
    if(abs(h-oldh).gt.tol.and.itt.le.imatit) goto 12
    return
    end

c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c   gradh: calculate gradient wrt h
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    subroutine gradh(h,c,lambda,s,x,sdir,deleh,sigma)
    parameter (nsam=19,vsize=33,ncen=8)
    double precision c(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
+         lambda(ncen,vsize), sigma(ncen),h,deleh,pred(nsam,vsize),
```

```
+           sdir(ncen,vsize),tol,stmp(ncen,vsize),A(nsam,ncen),
+           fac(nsam,ncen),deriv(nsam,vsize),tmp

      deleh=0.0
      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         stpm(j,k)=0.0
         pred(i,k)=0.0
         deriv(i,k)=0.0
        enddo
       A(i,j)=0.0
       fac(i,j)=0.0
       enddo
      enddo

c     compute exp{-||x-c||^2}
      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         tmp=x(i,k)-c(j,k)-h*sdir(j,k)
        enddo
        A(i,j)=A(i,j)+tmp**2
        fac(i,j)=fac(i,j)+0.5*tmp*sdir(j,k)/(sigma(j)**2)
       enddo
       A(i,j)=exp(-A(i,j)*0.5/(sigma(j)**2))
      enddo

c     compute predicted output
      do j=1,ncen
       do k=1,vsize
        pred(i,k)=pred(i,k)+A(i,j)*lambda(j,k)
        deriv(j,k)=deriv(j,k)+A(i,j)*lambda(j,k)*fac(i,j)
       enddo
      enddo

c     compute gradient value
      do j=1,ncen
       do k=1,vsize
        deleh=deleh+(s(i,k)-pred(i,k))*(deriv(j,k))*(-1)
       enddo
      enddo
```

```
      return
      end
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c     update centers and sigmas
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      subroutine update(c,sdir,h,sigma,sigdir,delta)
      parameter(vsize=33,ncen=8)
      double precision c(ncen,vsize),sdir(ncen,vsize),tmps(ncen,vsize),
     +          h,delta,sigsign(ncen),sigma(ncen),sigdir(ncen)

      do j=1,ncen
       do k=1,vsize
        if(sdir(j,k).lt.0.0) then
             tmps(j,k)=-1.0
        else if (sdir(j,k).gt.0.0) then
             tmps(j,k)=1.0
        else
             tmps(j,k)=0.0
        endif
       enddo
       if (sigdir(j).lt.0.0) then
             sigsign(j)=-1.0
       else if (sigdir(j).gt.0.0) then
             sigsign(j)=1.0
       else
             sigsign(j)=0.0
       endif
       sigma(j) = sigma(j) + h*sigsign(j)
      enddo

      do j=1,ncen
       do k=1,vsize
        c(j,k)=c(j,k)+h*tmps(j,k)
       enddo
      enddo
      return
      end
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c     calculate function gradient
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      subroutine gradfun(delf,c,lambda,s,x,sigma)
      parameter(nsam=19,vsize=33,ncen=8)
```

```
      double precision c(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
     +          lambda(ncen,vsize),pred(nsam,vsize),delf(ncen,vsize),
     +          A(nsam,ncen),sigma(ncen),tmp1,tmp2

      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         delf(j,k)=0.0
         pred(i,k)=0.0
        enddo
        A(i,j)=0.0
       enddo
      enddo

c     compute exp{-||x-c||^2}
      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         A(i,j)=A(i,j)+(x(i,k)-c(j,k))**2
        enddo
        A(i,j)=exp(-A(i,j)/(2*(sigma(j)**2)))
       enddo
      enddo

c     compute predicted output
      do i=1,nsam
       do k=1,vsize
        do j=1,ncen
         pred(i,k)=pred(i,k)+A(i,j)*lambda(j,k)
        enddo
       enddo
      enddo

c     compute gradient
      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         tmp1=(s(i,k)-pred(i,k))*A(i,j)
         tmp2=2*(sigma(j)**2)
         delf(j,k)=delf(j,k)+tmp1*lambda(j,k)*(x(i,k)-c(j,k))*(-1/tmp2)
        enddo
       enddo
```

```
      enddo
      return
      end
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c     calculate matrix norm and ratio
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      subroutine norm2(oldelf,delf,matnorm)
      parameter (vsize=33,ncen=8)
      double precision oldelf(ncen,vsize),delf(ncen,vsize),oldnorm,newnorm,
     +           matnorm

      oldnorm=0.0
      newnorm=0.0

      do j=1,ncen
       do k=1,vsize
        oldnorm=oldnorm + oldelf(j,k)**2
        newnorm=newnorm + delf(j,k)**2
       enddo
      enddo

      matnorm = newnorm/oldnorm
      return
      end
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c     evaluate error function
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      subroutine fun(error,c,lambda,s,x,sigma)
      parameter(nsam=19,vsize=33,ncen=8)
      double precision c(ncen,vsize),s(nsam,vsize),x(nsam,vsize),
     +           lambda(ncen,vsize),pred(nsam,vsize),fnc(nsam,vsize),
     +           A(nsam,ncen),error,sigma(ncen)

      open(7,file='output',status='unknown')
      open(8,file='opt_sigma',status='unknown')
      error=0.0
      do i=1,nsam
       do j=1,ncen
        do k=1,vsize
         pred(i,k)=0.0
        enddo
       A(i,j)=0.0
```

```
      enddo
    enddo

c   compute exp(-||x-c||^2)
    do i=1,nsam
     do j=1,ncen
      do k=1,vsize
       A(i,j)=A(i,j)+(x(i,k)-c(j,k))**2
      enddo
     A(i,j)=exp(-A(i,j)/(2*(sigma(j)**2)))
     enddo
    enddo

c   compute predicted output
    do i=1,nsam
     do k=1,vsize
      do j=1,ncen
       pred(i,k)=pred(i,k)+A(i,j)*lambda(j,k)
      enddo
     write(7,*) s(i,k)*100.,pred(i,k)*100.
     enddo
    write(7,*) ' '
    enddo

    do j=1,ncen
     write(8,*) sigma(j)
    enddo

c   compute function
    do i=1,nsam
     do k=1,vsize
      error=error+(s(i,k)-pred(i,k))**2
     enddo
    enddo

    rewind(7)
    rewind(8)
    return
    end

c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c   RBF training network
```

```
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
c  ni = number of input nodes
c  nh = number of hidden nodes
c  no = number of output nodes
c  nsam = number of samples
   subroutine train(c,x,s,lambda,sigma)
   parameter (ni=33, nh=8, no=33, nsam=19)
   double precision    A(nsam,nh), S(nsam,no), x(nsam,ni), lamb(nh,no),
+                      dum1(nh,nh), dum2(nh,nsam), Atran(nh,nsam),
+                      p, center(nh,ni), b(nh), dinv(nh,nh), z(nh), rcond,
+                      sigma(nh),sig
   integer             job, ipvt(nh)

c  Gaussian basis function
c          rbf(p) = exp(-p/(2*(sig**2)))

c  read in data points and S outputs
   do i=1,nsam
     do j=1,nh
       A(i,j)=0.0
        b(j)=0.0
        dum2(j,i)=0.0
        Atran(j,i)=0.0
     enddo
   enddo

   do j=1,nh
     do k=1,no
      lambda(j,k)=0.0
     enddo
   enddo
   do i=1,nh
      do j=1,nh
        dum1(i,j)=0.0
      enddo
   enddo

   m=nsam
   ia=nsam
   n=nh
   iu=nh
   ifail=0
```

```
      job=0

c     calculates A-matrix:  A=phi(||x-c||)
      do i=1,nsam
       do j=1,nh
        sig=sigma(j)
        do k=1,ni
         A(i,j) = (x(i,k) - c(j,k))**2  + A(i,j)
        enddo
       A(i,j) = rbf(A(i,j))
       enddo
      enddo

c     calculates the transpose of A-matrix
      do i=1,nsam
       do j=1,nh
        Atran(j,i) = A(i,j)
       enddo
      enddo

c     multiplies transpose of A-matrix by A-matrix
      do i=1,nh
       do j=1,nh
        do k=1,nsam
         dum1(i,j) = dum1(i,j) + Atran(i,k)*A(k,j)
        enddo
       enddo
      enddo

c     calculate the inverse matrix of dum1
      call dgeco(dum1,nh,nh,ipvt,rcond,z)
      rconda=rcond
      do 20 i=1,nh
      do 25 j=1,nh
       b(j)=0.0
25    continue

      b(i)=1.0
      call dgesl(dum1,nh,nh,ipvt,b,job)
      do 22 jj=1,nh
       dinv(jj,i)=b(jj)
22    continue
```

```
20 continue

c   multiplies the result by A-matrix transpose
    do i=1,nh
     do j=1,nsam
      do k=1,nh
       dum2(i,j) = dum2(i,j) + dinv(i,k)*Atran(k,j)
      enddo
     enddo
    enddo

c   calculates lambda (expansion coefficients) and write to file
    do i=1,nh
     do j=1,no
      do k=1,nsam
       lambda(i,j) = lambda(i,j) + dum2(i,k)*S(k,j)
      enddo
     enddo
    enddo

    end
```