Neural networks using homotopy continuation methods

by

Joseph C. Chow

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

1991

TABLE OF CONTENTS

ABSTRACT

Neural networks offer a powerful tool for performing classification tasks due to their ability to generate complex decision surfaces. Unfortunately, in most realistic cases where the decision surface separating the data set is highly nonlinear, the classification obtained most likely represents a suboptimal solution. This is largely due to the fact that search techniques employed by training algorithms may converge to local minima.

This thesis proposes the application of the homotopy continuation method for training neural networks. Homotopy continuation methods overcome the disadvantages such as long training time and the convergence to local minima, encountered with the conventional training methods. The homotopy continuation method is a globally convergent numerical method where a homotopy function describes a continuous deformation of a simple performance surface into the surface associated with a complex, nonlinear optimization problem. The validity of the proposed method is demonstrated by means of an example where the conventional backpropagation method converges to a local minimum resulting in a large classification error (>50%). The homotopy continuation method, however, results in accurate classification performance without excessive computational effort. Preliminary results indicate the potential of this approach with respect to both training time and classification accuracy.

CHAPTER I.

INTRODUCTION

1.1. Introduction

Information handling has become an issue of great concern to modern society. For instance, the medical industry requires information to be processed efficiently in order to ensure prompt and accurate medical diagnosis. Similarly, the banking industry demands a fast information handling system to accommodate the great volume of transactions. A major step associated with the design of modern information systems, however, is automatic pattern classification.

Pattern classification is considered one of the fundamental attributes of human beings [1]. Human beings are faced with classification decisions every waking moment. Such a decision making process demands the identification and classification of spatial and temporal patterns. Some examples of temporal patterns include speech, electrocardiogram, and target signatures. On the other hand, examples of spatial patterns include characters, images, and weather maps. In general, pattern classification is a problem of discriminating the input data patterns among population members. Most of the conventional pattern classifiers utilize this concept in achieving the desired classification. The primary function of a pattern classifier is to render decisions concerning the class membership of the input patterns. In order to accomplish this, a decision

or discriminant function is necessary. These decision functions represent decision surfaces partitioning the feature space, needed for correct classification.

One of the simplest approaches in designing a pattern classifier is through the use of distance functions. This approach utilizes the concept of clustering in forming decision surfaces. Clusters with dissimilar attributes constitute different classes. The similarity in input attributes can usually be measured in terms of the Euclidean distances among the input patterns; the shorter the Euclidean distance, the closer the attributes. Pattern classifiers such as maximum distance classifier, K-means classifier, and isodata algorithm [1] are all based on the distance function theory.

A second class of pattern classifiers is based on an adaptive approach which relies on the process of training. Trainable pattern classifiers are generally taught by means of iterative learning processes. A typical learning algorithm involves presentations to the classifier with training patterns and corresponding desired outputs during the training phase. The classifier is then taught iteratively to associate the sample input patterns with their corresponding desired outputs. An example of such a type of classifier is the artificial neural network.

Work on artificial neural networks began over 40 years ago. Pioneers of this work include McCulloch and Pitt [2], Hebb [3], and Rosenblatt [4]. More recently, Hopfield and Rumelhart [14] [46] revived interest in the subject by introducing new network topologies and training algorithms. The work was primarily motivated by the desire to mimic the human brain. The recent realization of human-like performance in speech and image recognition helped

rekindle interest in neural networks [5]. In particular, neural networks attempt to mimic the architecture of the human nervous system.

The biological nervous system consists of neuron cells which represent the basic element in the cognitive learning process. These neuron cells are interconnected with each other so information can be propagated. The propagation process can be accomplished by either firing or resting of each neuron cell. Artificial neural networks consist of densely interconnected neuron-like elements called nodes. These nodes are interconnected via weight factors which signify the importance of a connection from one node to another node. Artificial neural networks are seen to be particularly effective in applications which require high computation rate, such as image and speech recognition. Studies have shown that classification rates as high as 100% have been attained in a two-class image identification problem [6].

Neural network structures are primarily classified based on the network topology and the training algorithm employed. The class of neural networks most commonly used for generating complex decision surfaces, useful for classification, is the multilayer perceptron network.

During the training phase, the network's interconnection weights are updated iteratively to minimize the merit error function [7]. The error function is therefore, employed as the principal criterion in determining whether a network has been properly trained. As a result, it is highly desirable to ensure that the minimum obtained for the error function be a global minimum point rather than a local minimum point.

Traditional algorithms used for training employ local searching techniques [5]. As an example, the commonly used backward error propagation

algorithm utilizes the gradient descent method where the performance is dependent on the initial starting point. If the initial point is in the vicinity of a local minimum, the solution will most likely converge to the local minimum [8] as illustrated in Figure 1.1. Another disadvantage of the gradient descent method is that under certain conditions, the iterative process may oscillate between two points and fail to converge.

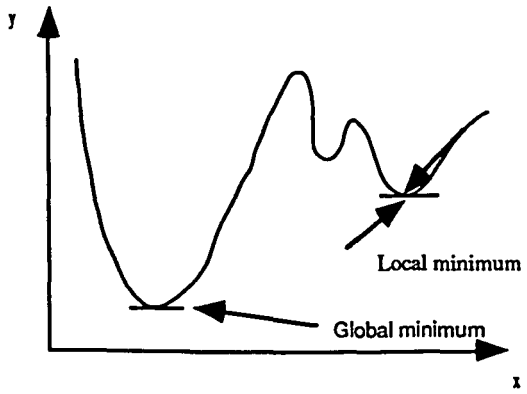


Figure 1.1 Local search technique

These drawbacks can be overcome by using homotopy continuation methods which are global search methods. These methods are globally convergent and in the case of polynomial systems offer an exhaustive set of solutions [9]. The term globally convergent implies that regardless of the location of the initial points, homotopy methods guarantee convergence to a solution. Such a solution may not, however, represent the desired solution. Solution exhaustive implies that all solutions to a given system can be found .

This thesis presents an innovative approach for training multilayer perceptron neural networks using the homotopy continuation method. Continuation methods are used extensively for solving optimization problems

in several fields. The proposed technique offers the globally optimal solution which usually translates into better classification performance.

1.2. Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 presents the background information needed to understand the operation of neural networks. Starting with a brief discussion of biological neurons, a comparison between a biological neural system and the artificial neural network is presented. Various types of artificial neural networks are described with special attention focused on the multilayer perceptron model. The operation of the multilayer perceptron and the backward error propagation technique, commonly used for training this network are described. Issues such as training time and convergence of local minima are also discussed.

Chapter 3 introduces the fundamental concepts of homotopy continuation methods. The discussion includes both the theoretical basis and the numerical procedure employed for arriving at the solution. The homotopy concept involves deformation of a simple system with known solution into a more complex system whose solution is desired. The method defines a family of paths which track the solutions of the known system to the solutions of an unknown system, thus obtaining the desired solutions. Conditions for path existence and path finiteness are also presented.

Chapter 4 describes the original contribution of this research involving the application of homotopy continuation methods for training neural networks.

Several approaches based on various types of homotopy methods along with the difficulties encountered with each approach are presented. Following a discussion of the polynomial approximation for the sigmoidal activation function of the network nodes, a training algorithm using the fixed-point homotopy method is presented.

Chapter 5 describes an alternate method capable of computing all real solutions to nonlinear minimization problems. The polynomial resultant matrix method is discussed with regard to both the advantages as well as the practical computing aspects which limit the application of the method for training neural networks.

Chapter 6 presents simulation results using the fixed-point continuation method. Results obtained for a few two class problems confirm the validity of the proposed homotopy methods for training neural networks.

Finally, Chapter 7 presents some concluding remarks and areas of future research.

CHAPTER II.

NEURAL NETWORKS

2.1. Background

Artificial neural networks have drawn considerable attention as a powerful approach for performing classification tasks. Initial work on neural networks was motivated by a desire to mimic the human nervous system in an attempt to emulate the human learning process [10] [11]. The underlying concept of neural networks can be more easily understood by first examining the human nervous system.

Biological nervous systems consist of a network of neuron cells communicating with each other and with various parts of the body [12]. A typical neuron cell is illustrated in Figure 2.1.

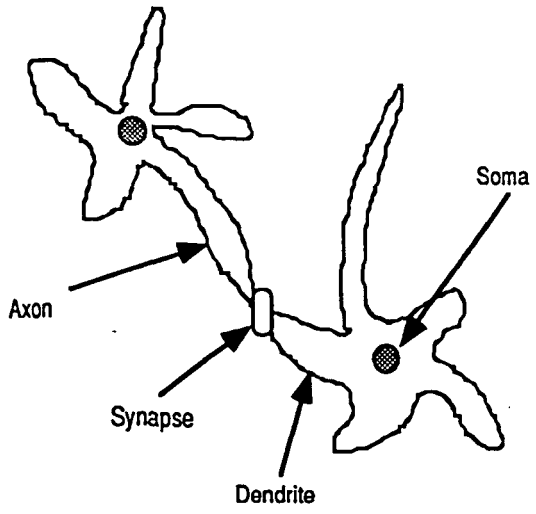
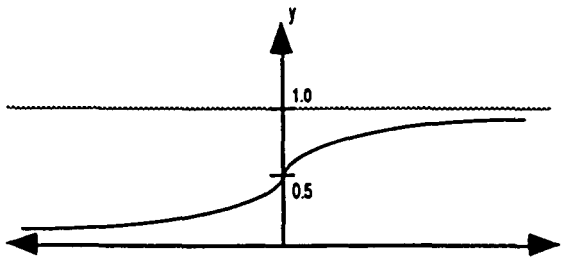


Figure 2.1 Biological neuron cell [12]

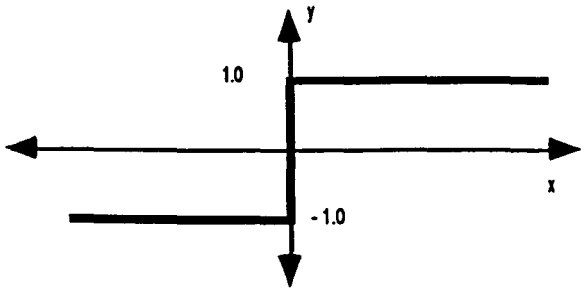
A neuron cell consists of a cell body called the soma, several cell extensions called dendrites, and a single nerve fiber called the axon. Dendrites of a cell are used to receive information from other cells. The axon, on the other hand, is used to propagate information to dendrites of neighboring cells. The junction where dendrites and axons meet is called the sypnapse. Inside and around the soma are various types of ions such as sodium (Na^+), calcium (Ca^{++}), potassium (K^+), and chloride (Cl^-). When the membrane of the soma is stimulated by a voltage change, it allows ions outside of the membrane to pass across the membrane and change the internal state of the soma. The voltage change in the cell body often results from information received by the dendrites. In other words, the neuron is excited by the received information through a voltage change in the cell body. This excitation process is called sypnaptic firing. A neural network is formed by the interconnections of all the neurons via the axons and dendrites. A single neuron can be modeled as a processing unit that sums up all of the inputs and passes the result through a threshold function. If the sum exceeds the threshold level, then an output will be produced. The output produced is then passed on to other neurons through an axon.

Artificial neural networks attempt to mimic the biological structure using the same framework. These networks consist of simple computational nodes and interconnecting weights. The nodes are often characterized by a nonlinear function called the nodal activation function. The nodal activation function is similar to that of the soma in that it takes the sum of all the inputs from other nodes and determines whether a firing process should occur. The sum of the inputs is passed through a nonlinear function which bounds the output

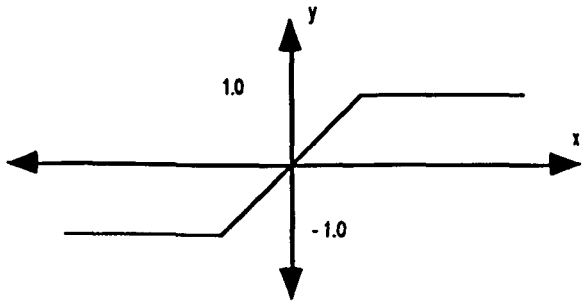
between one and zero, corresponding to the firing and non-firing state of the biological counterpart. Typical nonlinear activation functions are shown in Figure 2.2.



Sigmoid function



Hard limiter function



Soft limiter function

Figure 2.2 Nonlinear nodal functions

The interconnecting weights are used to determine the strength of a connection from one neuron node to another. This weighting scheme, in turn, plays an important role in the overall network configuration, and can be compared to the sypnaptic strengths of neurons.

Numerous neural networks have been introduced by researchers over the years, each suitable for a specific type of application problem [5]. Neural networks can be classified based on the architecture, the nodal activation function and the learning algorithm. The taxonomy of six major classes of neural networks are illustrated in Figure 2.3.

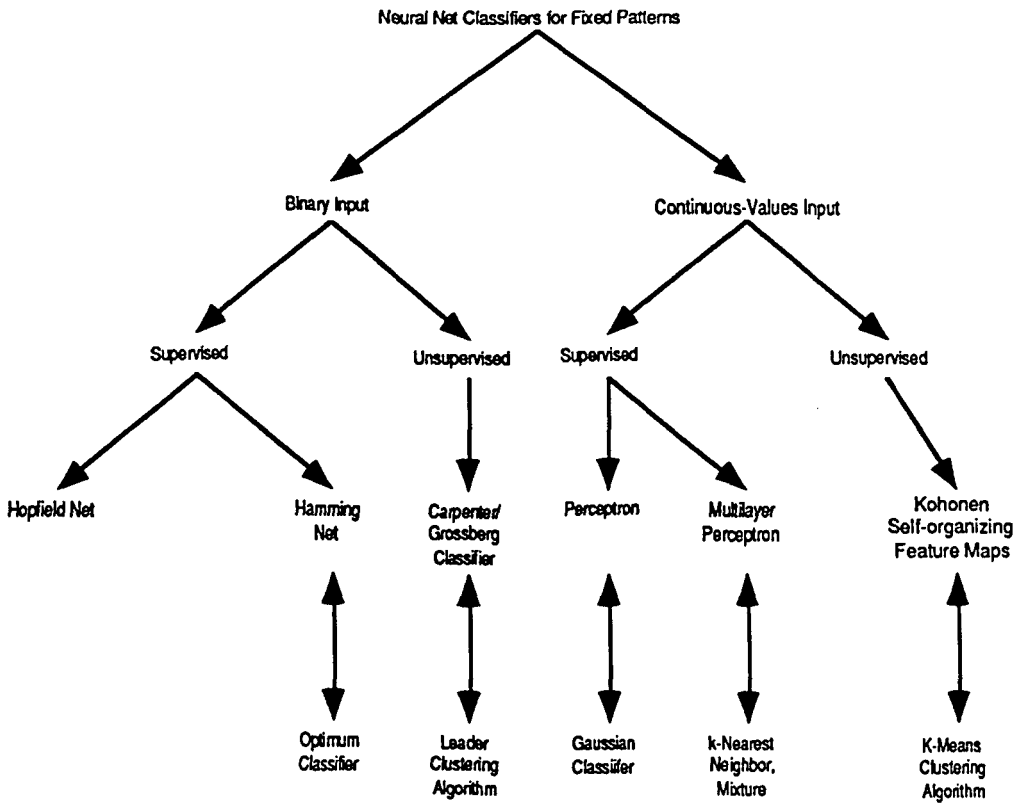


Figure 2.3 Taxonomy of neural networks [5]

Neural networks can be divided according to the type of inputs which can be binary or continuous valued. These networks can be classified further based on their training scheme which are in general, supervised, or unsupervised. In supervised training schemes, the networks are provided with the desired input/output pairs during the training procedure to achieve the desired classification [4] [14]. Unsupervised training usually results in generation of vector quantizers that can be used to form clusters of the input data [15] [16] [17].

The perceptron network is the primary focus of this thesis for two reasons. First, the perceptron network possesses the ability to handle continuous valued inputs as opposed to only binary inputs. Second, the perceptron network utilizes a supervised training scheme designed to train the network in a systematic manner. The perceptron network originated from the early work in bionics. In the 1950's, Rosenblatt [4] developed the perceptron model, which many researchers felt was the natural model for learning machines. This model employs the reward and punishment approach to train the network. The perceptron algorithm is adaptive and is relatively flexible and robust.

The simplest model of the perceptron networks is the single layer perceptron. This network consists of one input layer and one output layer. During the initial development of this network, much attention was drawn with regard to its ability to classify simple pattern sets. However, single layer perceptrons are only capable of distinguishing pattern sets which are linearly separable. In most realistic applications the data sets are not linearly separable, and single layer perceptron networks are, therefore, ineffective.

Multilayer perceptron networks, on the other hand, are capable of generating highly nonlinear decision surfaces. These networks employ one or more intermediate layers of hidden nodes. In addition, the use of nonlinear nodal activation functions allows for the generation of nonlinear decision surfaces. The multilayer perceptron network was not used commonly in the past due to the lack of effective training algorithms. However, new training algorithms developed in recent years have resulted in the widespread use of the multilayer perceptron network. The network is discussed in greater detail in the following section.

2.2. Multilayer Perceptron Network

The multilayer perceptron network generally consists of an input layer of nodes, one or more hidden layers of nodes, and one output layer of nodes [7] as illustrated in Figure 2.4.

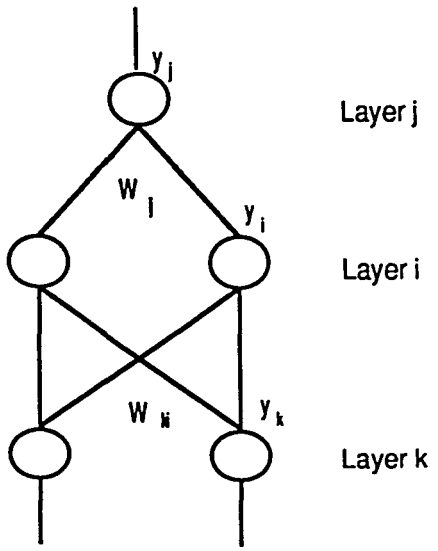


Figure 2.4 Simplified multilayer perceptron network

Connections between the nodes via weights are allowed from one layer to another layer. Although the layers need not be adjacent, connections within the same layer is forbidden. All units within a layer process data in parallel, but the outputs of different layers are calculated sequentially starting from the input layer and moving forward or upward to the output layer. Each node j in a layer k+1 performs the following computations:

Step 1:

$$x_j = \sum_{i=1}^{N_k} w_{ij}\, y_i \tag{2.1}$$

where y_i : is the output of nodes in layer i.

N_k : is the number of nodes in layer i.

w_{ij} : are the interconnection weights.

Step 2:

$$y_j = f(x_j) = \frac{1}{1 + e^{-(x_j+\theta_j)}} \tag{2.2}$$

where θ_j is a bias variable. This nonlinear function is primarily used to limit the output of a node between the values of 0 and 1 as shown in Figure 2.5. In addition, the sigmoid function of equation (2.2) has the advantage of being differentiable.

The artificial neural network basically operates in two modes: the classification mode and the training mode. In the classification mode, data flows only in the forward direction.

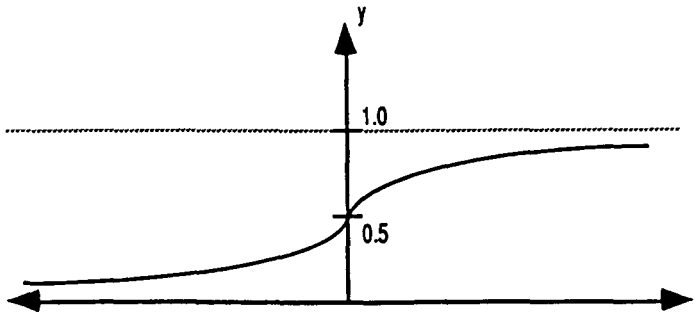


Figure 2.5 Sigmoid function

The input patterns are presented at the input nodes and the output of each node is propagated forward to the nodes of the next layer until an output is produced at the output node. The output, typically, signifies the classification of the input pattern. In the training mode, data flows in both, forward and reverse direction. The training data is first presented to the input nodes which produce an output at the output node, as is done in the classification mode. The network output, is then compared with the corresponding desired output and the error measured between the two is then used to adjust the weights in a way that minimizes the error.

2.3. The Learning Procedure

The overall objective of the learning process is to obtain a set of interconnection weights which will result in the actual output being as close as possible to the desired output. In order to accomplish this, a measure of error, E, is first defined by

$$E = \frac{1}{2} \sum_{c} \sum_{j} (y_j(c) - d_j(c))^2 \tag{2.3}$$

where

c is the input sample case index

j is the output node index

y is the actual output

d is the desired output

In order to minimize the error by adjusting the weights, it is necessary to compute the partial derivative of E with respect to each weight in the network. Since the training process entails propagating the error backwards from the output layer to the input layer, the backward pass starts by computing

$$\frac{\partial E}{\partial y_j} = y_j - d_j \tag{2.4}$$

for a particular value of c. Applying the chain rule,

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_j} \tag{2.5}$$

From equation (2.2), we have

$$\frac{\partial y_j}{\partial x_j} = y_j (1 - y_j) \tag{2.6}$$

Furthermore, minimization of the error with respect to the weights, w_{ij}, results in the equation

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ij}} = 0 \tag{2.7}$$

which reduces to

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} y_i = 0 \tag{2.8}$$

For the output of the i^{th} unit, the error contribution can be shown to be

$$\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial y_i} \tag{2.9}$$

which reduces to

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ij} \tag{2.10}$$

and

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial w_{ki}} \tag{2.11}$$

Figure 2.6 illustrates the above steps graphically.

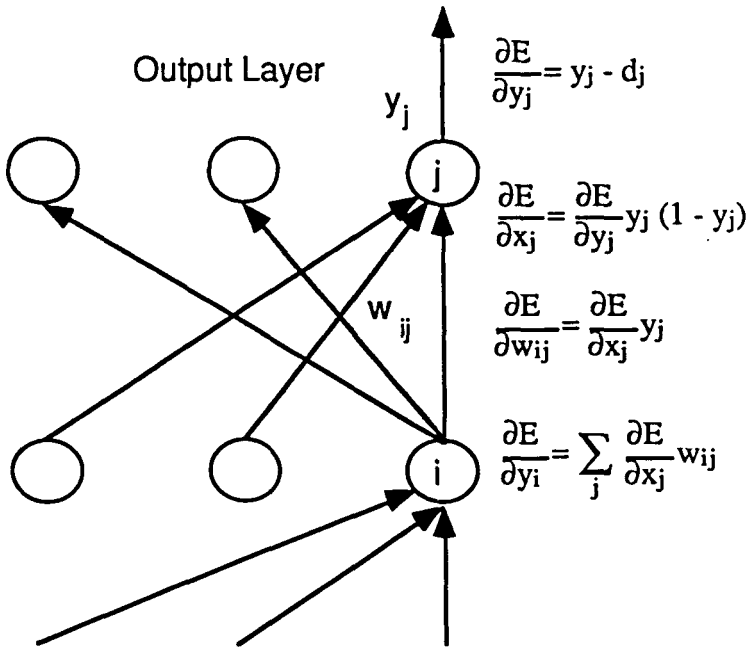


Figure 2.6 Graphical representation of learning equations [7]

This process is repeated until all layers have been reached so that all of the weights in the network are corrected. In addition to the interconnected weights, the bias variables used for enhancing the output convergence rate can also be adjusted to achieve the optimal solution. For the output layer nodes, we can estimate the bias value by solving the equation

$$\frac{\partial E}{\partial \theta_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \theta_j} = 0 \qquad (2.12)$$

Similarly for the hidden layer nodes, we have

$$\frac{\partial E}{\partial \theta_i} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial y_i}\frac{\partial y_i}{\partial \theta_i} = 0 \qquad (2.13)$$

By solving equations (2.7), (2.11), (2.12), and (2.13), a minimum error value can be obtained.

The computational effort required for the backward pass is comparable to the forward pass since it is linear in the number of connections and the mathematical operations are similar. In addition, equations (2.4) and (2.9) in the backward pass play the same role as y in the forward pass. The only difference is that the sum is passed through a nonlinear function in the forward pass whereas in the backward pass it is multiplied by equation (2.6).

In general, the connection weights are updated for each input-output pair. This method requires no additional memory for the storage of the derivative values. An alternate method, of course, is to sum the errors over all input-output cases before the weights are updated.

2.4. Gradient Descent Method

The iterative procedure for determining the error minimum, based on the gradient descent method, is shown in Figure 2.7. This weight correction scheme calculates the gradient of the error curve and corrects the interconnected weights by an amount, Δw, proportional to the gradient value. This correction technique will move the weight values in the direction of the error minimum.

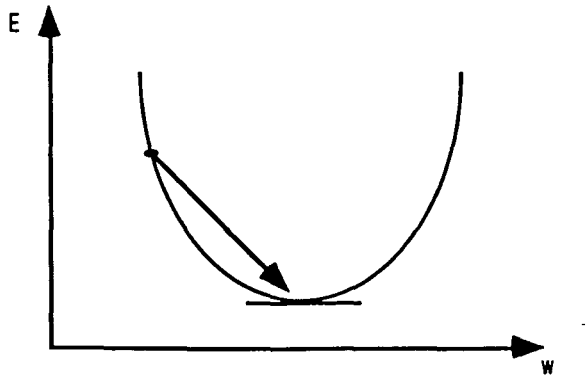


Figure 2.7 Steepest descent method

The incremental change in each weight is proportional to the accumulated derivative of error with respect to the weights. For example,

$$\Delta w = -\gamma \frac{\partial E}{\partial w} \tag{2.14}$$

where γ is a gain term. To increase the convergence rate of this method, a momentum term, α, can be added to give

$$\Delta w(t) = -\gamma \frac{\partial E}{\partial w(t)} + \alpha \, \Delta w(t-1) \tag{2.15}$$

where t is the iteration number. The momentum term, α, is generally varied from 0 to 1 in order to determine the relative contribution of the gradients with respect to the weight change.

A numerical algorithm employing the gradient-descent method for training neural networks can be found in [5]. It is worthwhile to note that in order to improve the convergence rate for the gradient method, several

additional parameters can be added [5]. These parameters include a gain term, γ, and a momentum term, α, as described earlier. Typically, these parameters have to be determined experimentally thereby, contributing to an increase in the complexity of the training process.

2.5. Characteristics of Weight Space

The dimensionality of the weight space is related to the number of weights in the network in that each weight in the network corresponds to a dimension in the weight space. A convenient way to understand the learning procedure is to observe the movement on the error surface in a multidimensional weight space. The error surface comprises of local minimums, saddle points, and a global minimum. The objective is to determine the global minimum point so as to achieve optimum classification. A major drawback of this technique is related to the fact that the error minimum achieved is highly dependent on the initial starting point, as shown in Figure 2.8. In this one dimensional example, initial weights w_a and w_b will both converge to local minima whereas w_c will converge to the global minimum. As a result, this method will perform poorly in cases where more than one minimum is present. In addition, it is also possible that the method will simply oscillate between two points and fail to converge completely. In Figure 2.8, the solution oscillates between the points w_d and w_e. This calls for the use of alternate search methods which are described in the next chapter.

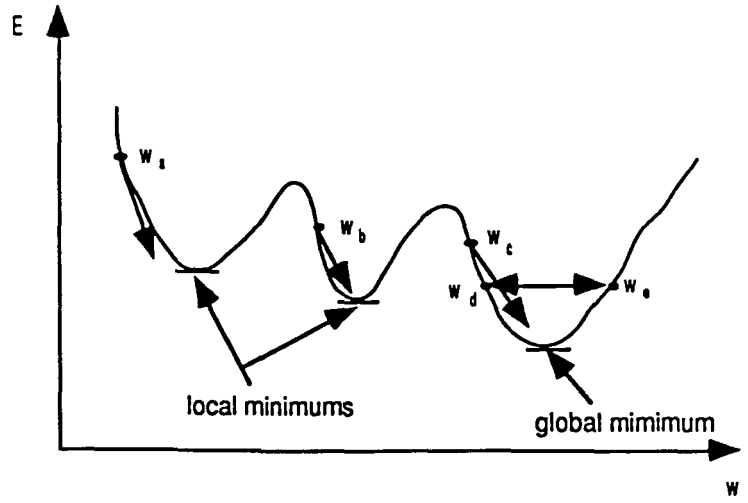


Figure 2.8 Convergence of gradient method (1-D case)

The effects of the gain and momentum terms have a direct impact on the physical movement on the error surface. Unfortunately, in most practical neural networks with large number of weight connections it is difficult to visualize the impact. One way to illustrate the error curve in a high dimensional weight space is to plot the movement of the error surface in significant directions of weight spaces, and compare the results with error surfaces plotted along random directions. This, in turn, aids the interpretation of the error surface movement in a high dimensional weight space.

CHAPTER III.

HOMOTOPY METHODS

3.1. Introduction

Homotopy continuation methods are numerical techniques used to determine the zeroes of a system of nonlinear equations [18] [19]. The underlying concept of the method involves identifying a simple system with known solution and slowly deforming it into the desired system with unknown solutions. During the deformation process, a family of paths is defined from the solutions of the known system to the solutions of the unknown system [20]. The continuation method is a numerical procedure used to track this solution path, where every point on the path represents the solution of the deformed system, for a particular value of the deformation parameter. When the known system is completely deformed into the unknown system, the desired solutions to the unknown system are obtained. Unlike the gradient based numerical methods, such as the Newton-Ralphson and the steepest descent methods [21], homotopy continuation methods are globally convergent and solution exhaustive. Globally convergent implies that the method will converge to a solution irrespective of the location of the initial starting point. Solution exhaustive signifies that all solutions to a desired system can be found. However, several conditions must be met for the solution exhaustive property to hold. First, the system of interest must

consist of polynomial equations of known degree. Second, the initial system must also consist of polynomial equations with the same degree as the unknown system. In addition, the initial system should have no common roots. This property was proved by Garcia and Zangwill in 1977 [22] and independently by Drexler in 1978 [23]. Information describing the relationship between Newton's method and homotopy method are discussed in [24] and [25]. Other homotopy methods and their applications can be found in [26], [27], [28], [29] and [30].

In the following sections, issues associated with the theory and numerical implementation of homotopy continuation method are discussed.

3.2. Homotopy Theory

The homotopy function h(x,t) is defined as [31]:

$$h(x,t) \equiv (1 - t)\, g(x) + t\, f(x) \tag{3.1}$$

where t is the homotopic tracking parameter, g(x) is the system with known solutions and f(x) is the system of interest whose solutions are desired. As the parameter t varies from 0 to 1, h(x,t) deforms the known system, g(x), to the the desired system, f(x). For example, at $t = 0$, $h(x,0) = g(x) = 0$ gives the initial starting solutions of the known system.When $t = 1$, $h(x,1) = f(x) = 0$ gives the desired solution of the system of interest. The intermediate values of t between 0 and 1 correspond to the different deformed functions, h(x,t).

The homotopy function can be generalized into a vector form,

$$h(\lambda,t) = t\, f(\lambda) + (1-t)\, g(\lambda) \tag{3.2}$$

where

$$h(\lambda,t) = \left[h_1(\lambda,t), \cdots\cdots, h_n(\lambda,t)\right]$$

$$f(\lambda) = \left[f_1(\lambda), \cdots\cdots, f_n(\lambda)\right]$$

$$g(\lambda) = \left[g_1(\lambda), \cdots\cdots, g_n(\lambda)\right]$$

As t varies from 0 to 1, a family of paths is tracked from the solutions of $g(\lambda) = 0$ to the solutions of $f(\lambda) = 0$.

3.2.1. path description

A typical continuation path for a one dimensional function is illustrated in Figure 3.1. The continuation path is tracked by solving $h(x,t) = 0$ at each incremental step of t. The continuation path, shown in Figure 3.1, is seen to be continuously differentiable in space. Mathematically, the path can be described by a set of differential equations. Since every point (x,t) on the path satifies the equation $h(x,t) = 0$, the derivative of the homotopy function with respect to the tracking parameter, t, is equal to zero.

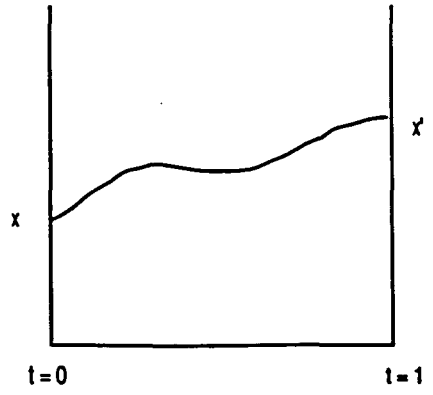


Figure 3.1 Monotonic homotopy path

x denotes a zero of g(x)
x' denotes a zero of f(x)

The homotopy differential equation is obtained by differentiating equation (3.1) with respect to the tracking parameter,

$$\frac{d}{dt}h(x(t)) = H_x(x(t))\frac{dx(t)}{dt} = 0 \tag{3.3}$$

where H_X is the Jacobian matrix of the homotopy function with repect to x. Partitioning equation (3.3) into a block matrix form and expressing x(t) = (λ(t), t) yields,

$$H_x(x(t))\,\frac{dx(t)}{dt} = \left[H_\lambda(x(t)) \mid \frac{\partial}{\partial t}h(x(t))\right]\begin{bmatrix}\frac{\partial\lambda(t)}{\partial t}\\ \vdots \\ 1\end{bmatrix} = 0 \tag{3.4}$$

where $H_\lambda(x(t))$ now represents the Jacobian matrix with respect to the λ parameters. Rearranging equation (3.4) results in,

$$\frac{d\lambda(t)}{dt} = - \left[H_\lambda(x(t)) \right]^{-1} \frac{\partial h(x(t))}{\partial t} \tag{3.5}$$

Equation (3.5) completely characterizes the continuation path of the homotopy function. The path is guaranteed to be continuously differentiable if the Jacobian matrix function $H\lambda(x(t))$ of the homotopy function is of full rank. While equation (3.5) offers a precise description of the solution path, it has limitations when path types as illustrated in Figure 3.2 are encountered. This figure demonstrates cases where the variation of x is not isomorphic with t.

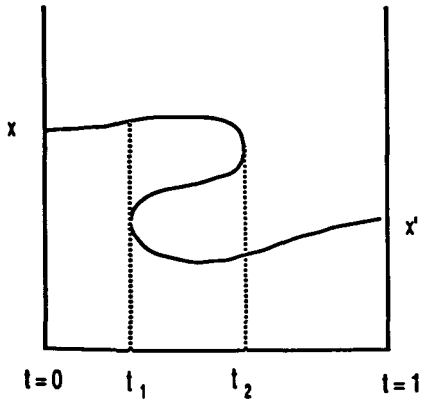


Figure 3.2 Nonmonotonic homotopy path

In this case, simply incrementing the tracking parameter t to solve the homotopy function will not yield an unique solution along the path. Such a case is shown in Figure 3.2, for $t_1 \leq t \leq t_2$, where there are multiple values of

x. The difficulty arises in "bending" the curve back to accommodate the turning of the solution path. To alleviate this problem, the tracking parameter is changed from t to the arc length, s, of the curve. Since the arc length is strictly a monotonically increasing function with respect to the homotopy parameters, the homotopy tracking parameter, t, can be decreased to allow the solution path to bend back.

The homotopy solution path using the arc length, s, as the tracking parameter can be described by differential equations similar to equation (3.4). Let x(s) be the point in the solution path after traveling a distance, s. Representing x(s) as

$$x(s) = (\lambda(s), t(s)) \tag{3.6}$$

We have as before

$$\frac{dh(s)}{ds} \equiv 0 \tag{3.7}$$

By applying the chain rule to equation (3.7) results in,

$$H_s \frac{dx}{ds} = 0 \tag{3.8}$$

where H_S is the {n x (n+1)}Jacobian matrix of h. The matrix elements of H_S are defined as

$$dh_{i,j} = \frac{dh_i(x)}{dx_j} \quad i = 1, \cdots\cdots, n \; ; j = 1, \cdots\cdots, n+1 \tag{3.9}$$

In terms of the λ parameters of equation (3.6), equation (3.8) can be rewritten as

$$H_{\lambda,t}(x(s)) \frac{dx}{ds} = 0 \tag{3.10}$$

Equation (3.10) can be solved in a manner similar to equation (3.3) to give

$$\left[\frac{dx(s)}{ds}\right]_{-i} = -\left[H_{\lambda,t}(x(s))\right]^{-1}_{-i} \frac{\partial h(x(s))}{\partial x_i} \tag{3.11}$$

where A_{-i} denotes the matrix A with its i^{th} column removed. The solution path can now be completely described by equation (3.11).

3.2.2. degree theory

Homotopy continuation method is particularly useful when the unknown system consists of polynomial functions. This is largely due to the fact that for polynomial systems, the solutions obtained using the homotopy methods are theoretically guaranteed to be exhaustive. In general, exhaustive solution implies that all solutions to the unknown system will be found. The degree of f is defined as,

$$\text{degree}(f) = \sum_{x \in f^{-1}} \text{sgn} \{ \det [F_x(x)] \} \tag{3.12}$$

where $f^{-1} = \{ x \mid f(x) = 0 \}$ represents the set containing all of the solutions to f and

$$\text{sgn}[x] = \begin{bmatrix} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{bmatrix} \tag{3.13}$$

The degree of a polynomial equation can be determined by simply taking the degree of the highest term. Similarly, the degree of a polynomial system can be found by taking the product of the highest order terms of each equation within the system [32]. For instance, let f(x) represent the polynomial system comprising of $f_i(x)$, i = 1, ..., n, and let the corresponding degree of $f_i(x) = d_i$, i = 1, ..., n. Therefore, the total degree, d, of the system,f, is:

$$d = \prod_{i=1}^{n} d_i \tag{3.14}$$

Generally, the number of solutions to a polynomial system can be shown to equal to the degree of the system. This can be proved using Bezout's theorem [20] which is stated below.

Bezout's Theorem

Let $d = d_1 \cdot d_2 \cdots d_n$ be the total degree of **f** where $f = 0$ is the polynomial system of interest. Then

1) The total number of geometrically isolated solutions and solutions at infinity, of $f = 0$, is no more than d.

2) If $f = 0$ has neither an infinite number of solutions nor an infinite number of solutions at infinity, then it has exactly d solutions and solutions at infinity, including multiplicities.

3.3. Homotopy Path Existence and Finiteness

In order to apply the homotopy method for tracking the solution paths, the conditions for path existence and path finiteness must first be met.

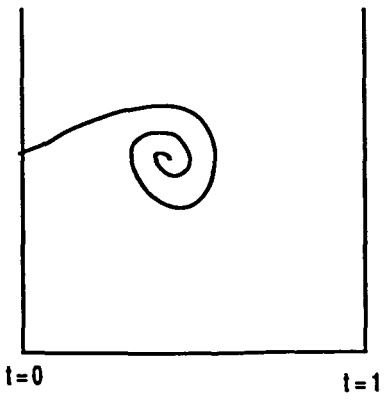


Figure 3.3 Spiral path

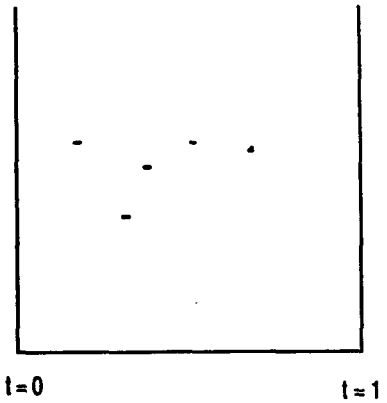


Figure 3.4 Point path

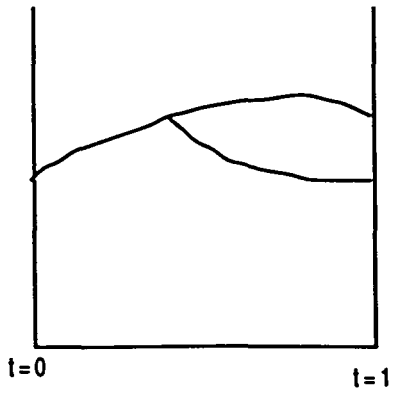


Figure 3.5 Bifurcating path

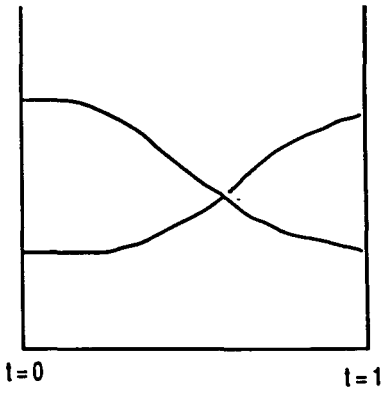


Figure 3.6 Crossing path

Some examples of non-paths are illustrated in Figures 3.3, 3.4, 3.5, and 3.6. The path existence condition provides the criteria for the existence of solution path and is described below.

Define

$$h^{-1} = \{ (x,t) \mid h(x,t) = 0 \} \tag{3.15}$$

as the set of all solutions $(x,t) \in R^{n+1}$ to the system $h(x,t) = 0$. Let

$$H_{x,t} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} & \frac{\partial h_1}{\partial t} \\ \vdots & & \vdots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_n} & \frac{\partial h_n}{\partial t} \end{bmatrix} \tag{3.16}$$

be the Jacobian matrix of the homotopy function. If $y = (x,t)$ is defined so that $y_i = x_i$ for $i = 1, \ldots\ldots, n$ and $y_{n+1} = t$ where $x \in R^n$ and $y \in R^{n+1}$, then the partial Jacobian matrix H_{-i}, which is an n x n matrix with the $i^{\underline{th}}$ column removed, is defined as,

$$H_{-i} = \begin{bmatrix} \frac{\partial h_1}{\partial y_1} & \cdots & \frac{\partial h_1}{\partial y_{i-1}} & \frac{\partial h_1}{\partial y_{i+1}} & \cdots & \frac{\partial h_1}{\partial y_{n+1}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial h_n}{\partial y_1} & \cdots & \frac{\partial h_n}{\partial y_{i-1}} & \frac{\partial h_n}{\partial y_{i+1}} & \cdots & \frac{\partial h_n}{\partial y_{n+1}} \end{bmatrix} \qquad (3.17)$$

It can be shown that if H_{-i} is invertible for some value of i, then a single continuously differentiable path exists for $(x,t) \in h^{-1}$ in the neighborhood of (x,t) where (x,t) is a point in h^{-1}. In other words, the solution path exists only when H_{-i} is of full rank for some value of i.

A homotopy solution path is said to be finite if all points on the path stay bounded for $0 \leq t \leq 1$. For instance, if the homotopy function is path finite for t in [0,1) then the path can diverge only when t approaches 1. The finiteness property ensures that a solution to $g(\lambda)$ will converge to a solution to $f(\lambda)$ provided that the number of solutions to $f(\lambda)$ is finite. In general, for polynomial functions, the finiteness property merely restricts the form of the equation of interest, but not the type of equation. The path finiteness condition is stated below.

Consider the problem of solving

$$f_i(\mathbf{x}) = 0 \qquad i = 1, \ldots\ldots, n \qquad (3.18)$$

where x is an n dimensional vector, $x = \{x_1, x_2, \cdots\cdots, x_n\}$, in the complex space and f_i: $C^n \rightarrow C$ is analytic and have all bounded solutions. Furthermore, the system to be solved can be modeled as

$$f_i(x) = (x_i)^{q_i} + p_i(x) \tag{3.19}$$

where q_i is a positive integer and p_i : $C^n \rightarrow C$ is analytic. In addition,

$$\| \frac{p_i(x)}{x_i^{q_i}} \| \rightarrow 0 \qquad \text{as } \|x_i\| \rightarrow \infty \tag{3.20}$$

The path finiteness property is guaranteed only if equation (3.20) is satisfied. Rewriting the unknown function in a solvable form,

$$f_i(x) = (x_i^{q_i} - 1) + (p_i(x) + 1) \tag{3.21}$$

and defining the known function to be

$$Q_i = x_i^{q_i} - 1 \tag{3.22}$$

the homotopy equation to be solved is

$$Q_i + t\,(p_i(x) + 1) = 0 \qquad 0 \le t \le 1 \tag{3.23}$$

To solve equation (3.23), the solution paths start out at each of the trivial solutions to Q_i and follow the path until $t = 1$. Thus, all solutions to the desired system are found.

3.4. Types of Homotopy Functions

The homotopy function can be defined in many ways. Each of these definitions results in a different form of homotopy methods.

3.4.1. all-solution homotopy method

The all-solution homotopy function is defined as

$$h(x,t) = (1-t)\, g(x) + t\, f(x) \tag{3.24}$$

The all-solution homotopy scheme involves tracking of all the solutions to a given nonlinear problem. The solutions obtained may either be real or complex. Similarly, the initial points may also be either real or complex. The all-solution homotopy method is an ideal technique for solving nonlinear minimization problems since the solutions found include the global minimum point.

The all-solution homotopy method can be easily applied to polynomial systems since the number of solutions to the system is known a priori. Choosing the degree of the initial known system, g(x), to be equal to the degree of unknown system, f(x), then guarantees exhaustive set of solutions.

In contrast, applying the all-solution method to other types of nonlinear problems is not trivial if the total number of solutions is not known a priori. Furthermore, as is the case for neural networks, many applications require the estimation of only real solutions to the problem. This restriction inhibits the use of the all-solution homotopy method since the knowledge of the number of real solutions to a system is seldom available. A more appropriate choice of homotopy method for these problems is the single-solution homotopy method.

3.4.2. single-solution homotopy methods

Single solution homotopy methods are numerical methods which track one solution to a given problem at a time. These methods are, theoretically, both robust and globally convergent in that they guarantee convergence irrespective of the initial starting point. Single-solution homotopy methods are of two types. The first type is called the fixed-point method where the homotopy function is defined as

$$h(x,t) = (1-t)\,(x-x_o) + t\,f(x) \tag{3.25}$$

The fixed-point method is very much similar in form to the all-solution homotopy method. The fixed-point method can be considered as a special case of the all-solution method where g(x) is chosen as the linear function $(x-x_o)$, regardless of the form of f(x). The second type of the single solution method is called the Newton homotopy method,

$$h(x,t) = f(x) - (1-t)\, f(x_o) \tag{3.26}$$

where x_0 is the initial starting point .

The Newton homotopy method entails evaluating the system of interest at the initial starting point and slowly tracking the solution until t = 1. Since g(x) is not needed in this method, the tracking process is strictly performed using the unknown system, f(x). It is important to note that in the Newton homotopy method, only one solution is tracked at a time. A drawback of single-solution homotopy methods is that they do not provide exhaustive solutions. Nevertheless, the simplicity of implementation and global convergence make them an attractive choice in many applications. In addition, the fixed-point homotopy method can be used for tracking real solutions and is, therefore, ideal for the purpose of this research.

3.5. Applications of Homotopy Methods in Various Fields

Homotopy continuation methods have received considerable attention as a solution exhaustive approach for solving nonlinear optimization problems. Nonlinear optimization problems are widely found in many engineering applications such as digital signal processing, process control, etc. Homotopy continuation methods have been applied to solve nonlinear optimization problems to obtain globally optimum parameters .

In [9], Stonick calculates the minimum mean squared error pole/zero parameter estimates using the homotopy continuation method. This problem

was further extended to the design of optimal infinite impulse response filters [33] and also to obtain the auto-regressive-moving-average (ARMA) parameter in system identification problems [34].

In [35], Watson, et al. utilized the homotopy method to determine the DC operating point for an integrated circuit. The method is capable of solving for the DC bias point even in the presence of nonlinear circuit components such as diodes and capacitors.

Vasudevan, et al. [36] considered the fuel-optimal orbital rendezvous problem. The problem consists of finding a minimum fuel rendezvous trajectory between two points. Although the results obtained took an order of magnitude longer than conventional nonlinear programming algorithm, nonetheless, valuable insights on the choice of initial known system were gained.

Continuation methods have also been applied for determining the frequency response curve of a nonlinear network [37]. Different characteristics of the curve, such as the state of equilibrium, were also obtained.

In addition, research has also been done in the areas of chemical modeling [29] and kinematics [30] using homotopy methods.

3.6. Homotopy Algorithms

As mentioned in a previous section, two different formulations of the homotopy functions are used, based on the monoticity of the solution path with respect to the tracking parameter. In the case where the tracking

parameter t is monotonically increasing with respect to the solutions, the numerical algorithm is described below.

1) Set $h(x,t) = (1-t)\, g(x) + t\, f(x)$
2) Solve for $h(x,t) = 0$ using Newton's method
3) If $t = 1$ go to step 6)
4) Increment t by a small incremental step
5) Goto step 2)
6) Solutions obtained for $f(x) = 0$

In the case of problems where the solution path is nonmonotonic, the tracking parameter used must allow the value of t to decrease. In this case, the arc length, s, of the solution path is typically used due to its monoticity with respect to the homotopy parameter. The path is tracked by first calculating the tangent of the curve and a predictor- corrector numerical scheme is then used to take an incremental step in the direction of the tangent [38]. A predictor-corrector scheme is illustrated in Figure 3.7. The incremental step size depends on the degree of steepness of the curve. A bigger incremental step is taken when the curve is flat (ie. tangent value is small) to predict the solution to the function . A corrector scheme is then used to adjust the predicted value back to the true solution path. In instances where the curve is fairly steep, a smaller predictor step is taken to prevent solutions from diverging. The same corrector scheme can also be applied to adjust the predicted value. A simple algorithm utilizing the predictor-corrector method is illustrated below.

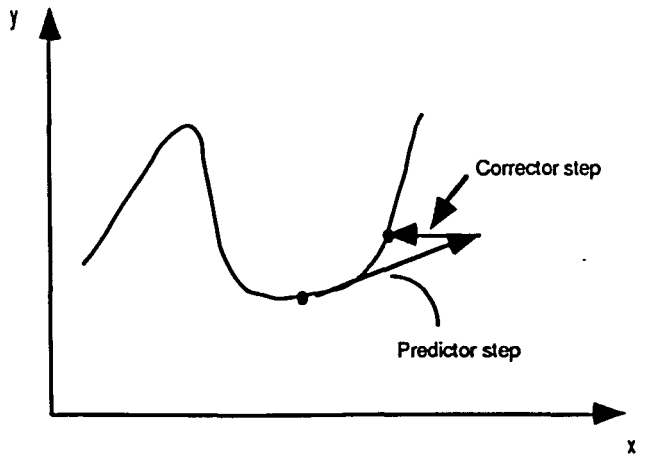


Figure 3.7 Predictor-corrector scheme

The homotopy function in terms of the arc length, s, is given by

$$h(s) = h(x(s)) = (1 - t(s))\, g(x(s)) + t(s)\, f(x(s)) \tag{3.27}$$

The step by step procedure is as follows:

1) Set $h(x,s) = (1-t(s))\, g(x) + t(s)\, f(x)$

2) Select m such that

$$\{ m \mid \left|\frac{dx_m}{ds}\right|^k > \left|\frac{dx_n}{ds}\right|^k \; \forall\, m \neq n \}$$

3) Calculate the tangent

$$\frac{dx_m}{ds} = 1$$

$$\left[\frac{dx(s)}{ds}\right]^{k+1}_{-m} = \left[-[H_x]^{-1}_{-m}\left|\frac{\partial h(x)}{\partial x_m}\right|_{-m}\right]^{k}$$

4) Determine the direction of the tangent

If the inner product between the k$^{\underline{th}}$ and k+1$^{\underline{th}}$ iteration of $\frac{dx_n}{ds} < 0$ then

$$\left[\frac{dx(s)}{ds}\right]^{k+1} = -\left[\frac{dx(s)}{ds}\right]^{k+1}$$

5) Compute the predictor step

$$[x]^{k+1} = x^k + \frac{\Delta\left[\frac{dx(s)}{ds}\right]^{k+1}}{\left\|\frac{dx(s)}{ds}\right\|^{k}}$$

6) Corrector step

Solve $h(x) = 0$ with x_m fixed using Newton's method

$$[x]^{j+1}_{-m} = [x]^{j}_{-m} - \left[H_x(x^j)\right]^{-1}_{-m}\left[h(x^j)\right]_{-m}$$

where j and k denote the iteration numbers.

7) Repeat steps 2 through 6 until $t = 1$

The Δ in step 5 denotes the incremental step size and can be varied depending on the steepness of the curve tangent. Other homotopy algorithms can be found in [39].

CHAPTER IV.

APPLICATION OF HOMOTOPY METHODS TO NEURAL NETWORKS

4.1. Introduction

This chapter describes the application of the homotopy continuation method to the specific task of training neural networks. As presented in Chapter 2, neural networks are trained by adaptively changing the interconnection weights so as to minimize the network classification error. Conventional gradient methods for error minimization result in convergence to a local minimum. The objective of this chapter, therefore, is to solve equations (2.7), (2.11), (2.12), and (2.13) using the homotopy continuation method whereby optimum classfication performance is achieved.

The all-solution homotopy method is an ideal choice for this application since it guarantees exhaustive solutions to the system of interest. This will then enable the user to determine the globally optimum solution to the problem. However, his method is applicable only to polynomial systems where the total number of solutions is clearly defined. The minimization equations for the neural network application do not comprise of polynomial functions. Rather, these equations consist of transcendental functions due to the sigmoidal activation functions of the nodes. As a result, the number of

solutions cannot be determined a priori which makes the all-solution homotopy approach difficult to apply. The difficulty arises mainly because an appropriate choice of g(x) cannot be made to properly track all solutions of f(x).

Two approaches have been considered to overcome this problem. The first approach involves modeling the sigmoid function using a polynomial function such that the approximation is valid in a specified domain. The all-solution homotopy method can then be used to track all solutions to the system of minimization equations. The order of this system increases drastically as the number of weights in the network increase. This presents severe problems during implementation as will be discussed in detail in the next section.

The second approach is much simpler than the first approach. Instead of changing the nodal activation function to accommodate the use of all-solution homotopy method, the sigmoid function is retained and a different homotopy method is employed. The fixed-point homotopy method is used due to both the simplicity of implementation and the global convergence property. In addition, the fixed point homotopy method can be implemented such that only real solutions to the system are tracked. This is especially important in the neural network application where only the real valued weights are considered.

4.2. Generalized Polynomial Approximation Method

In order to fully utilize the all-solution homotopy method, the nodal

activation function is modeled using a polynomial function to approximate the sigmoid function such that the approximation is valid in a bounded domain [40]. The domain is kept bounded to prevent the nodal output from diverging. In general, there are two restrictions which must be placed on the approximation function. First, the bounded domain should be large enough to allow the output to adequately converge to either 0 or 1. Second, the order of the modeling equation should be kept relatively low to reduce the complexity of the system which is inherently high considering the number of weights in the network. In order to meet the above criteria, the sigmoid function given in equation (2.2) is approximated with a third degree polynomial function as

$$p(x) = 0.5 + 0.19745\,x + 2.5224 \times 10^{-8}\,x^2 - 4.3917 \times 10^{-3}\,x^3 \qquad (4.1)$$

$$-3.8 \le x \le 3.8$$

where the bounded domain is chosen as [-3.8, 3.8.]. The nodal output is expressed as

$$f(x) = \begin{pmatrix} 0 & x < -3.8 \\ p(x) & -3.8 \le x \le 3.8 \\ 1 & x > 3.8 \end{pmatrix} \qquad (4.2)$$

Both the sigmoid function and the approximating polynomial are shown in Figure 4.1.

The minimization equations to be solved are as shown below.
For the output nodes,

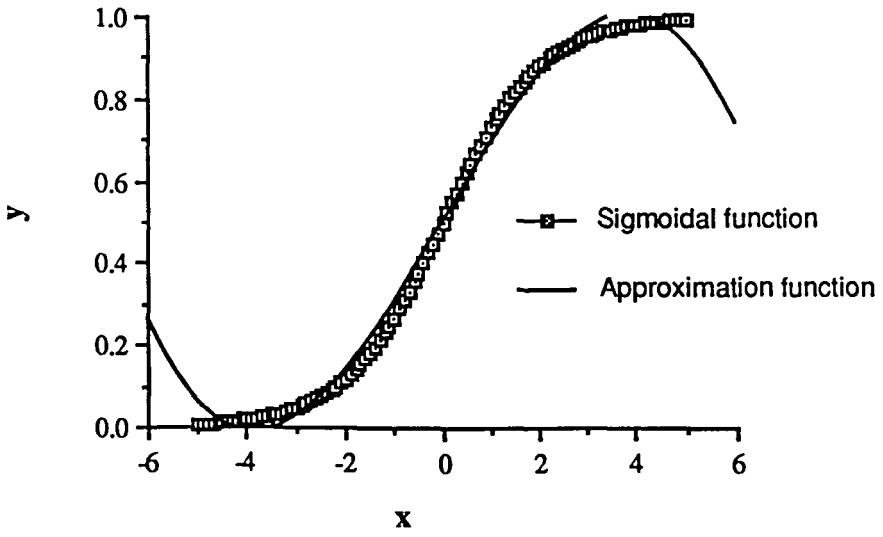


Figure 4.1 Sigmoid function and approximation function

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ij}} \tag{2.7}$$

Similarly, for the hidden nodes

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial w_{ki}} \tag{2.11}$$

By substituting equation (4.1) into equations (2.7) and (2.11), it can be shown that each equation results in a fifth degree nonhomogeneous polynomial function. Using the network structure given in Figure 2.4 which contains six weights in the networks (ie. six equations in the system), the degree of the resultant system becomes $5^6 = 15625$. In other words, there are 15625 possible solutions for the system. Since the neural network demands only real valued weights, the selection of g(x) is also limited to polynomial

functions with only real solutions. One method for generating the known system, g(x), in general, is to randomly select 15625 unique real numbers and forming a system of equations using these solutions. For example, for a polynomial system consisting of six fifth order equations, the system can be expressed in a general form as shown below:

$$a_0x_1^5 + a_1x_2^5 + a_3x_3^5 + \cdots\cdots + a_{n-2}x_6 + a_{n-1} = 0 \quad (4.3)$$

$$b_0x_1^5 + b_1x_2^5 + b_3x_3^5 + \cdots\cdots + b_{n-2}x_6 + b_{n-1} = 0 \quad (4.4)$$

$$c_0x_1^5 + c_1x_2^5 + c_3x_3^5 + \cdots\cdots + c_{n-2}x_6 + c_{n-1} = 0 \quad (4.5)$$

$$d_0x_1^5 + d_1x_2^5 + d_3x_3^5 + \cdots\cdots + d_{n-2}x_6 + d_{n-1} = 0 \quad (4.6)$$

$$e_0x_1^5 + e_1x_2^5 + e_3x_3^5 + \cdots\cdots + e_{n-2}x_6 + e_{n-1} = 0 \quad (4.7)$$

$$f_0x_1^5 + f_1x_2^5 + f_3x_3^5 + \cdots\cdots + f_{n-2}x_6 + f_{n-1} = 0 \quad (4.8)$$

where x_i, i = 1, ..., 6 represent the unknown weights in the network, and a_j, b_j, c_j, d_j, e_j, and f_j, j = 0, 1, $\cdots$, n-1, represent the coefficients of the polynomial equations. For this example, n can be shown to be 462 using equation (2.3-9) in [1]. To formulate the known system, g(x), the coefficients of the system must first be determined. The coefficients can be obtained by substituting the randomly chosen real solutions into equations (4.3) through (4.8) and solving the system of equations simultaneously. The resulting overdetermined system can be solved using a least square approximation method [42] to estimate the coefficients. However, the results obtained tend to be extremely noisy with the signal to noise ratio being close to one, which presents further problems. The development of a robust method for the generation of g(x) is, therefore, necessary for the application of the all-

solution homotopy method for training neural networks [43].

4.3. Fixed-Point Method

The second approach considered here is more robust than the polynomial modeling technique. This approach retains the use of sigmoid function as the nonlinear nodal activation function [41]. The approach employs the fixed-point homotopy method to track one solution at a time. This method is superior to the conventional gradient methods in that it is globally convergent. In addition, this method is much simpler to implement in terms of computational resources. The only drawback is that the solution exhaustive property is no longer guaranteed since the number of solutions to a system of transcendental functions cannot be predetermined.

Consider the fixed-point homotopy function,

$$\mathbf{h}(\mathbf{x},t) = (1-t)\,(\mathbf{x}-\mathbf{x}_0) + t\,\mathbf{f}(\mathbf{x}) \tag{3.25}$$

where $\mathbf{f}(\mathbf{x})$ denotes the system of equations given in equations (2.7), (2.11), (2.12), and (2.13). To track a solution, a starting vector point $\mathbf{x}_0$ is picked randomly to start the process. The tracking procedure incrementally traces a solution path from $\mathbf{x}_0$ to a solution of $\mathbf{f}(\mathbf{x})$. The solution obtained is used as the weight values to calculate the actual output of the neural network. The result is then used to determine the error corresponding to the weights obtained. It is helpful to point out that, typically, several solutions need to be tracked before a satisfactory answer can be attained. This is done by using

several of the previously tracked solutions to predict the location of the global minimum.

CHAPTER V.

MULTI-RESULTANT METHOD

5.1. Introduction

This chapter describes the multi-resultant method as an alternate approach for the application of training neural networks. The multi-resultant method is a numerical technique for determining all real solutions to a set of polynomial equations [44] [45]. As in many other engineering applications, the training of neural networks requires only real valued solutions to a given system of equations. In this case, the real valued solutions are the interconnection weights of the network. In contrast to homotopy methods, the algorithm does not require the generation of the polynomial system, g(x), for obtaining the desired solution. In addition, the algorithm allows the user to look for solutions in specific intervals of interest. The multi-resultant method for a system of homogeneous equations is described next.

5.2. Homogeneous Systems

Consider a system of homogeneous polynomials with real coefficients, in n variables.

$$f(x) = \begin{pmatrix} f_0(x_0, \cdots\cdots, x_{n-1}) \\ \vdots \\ f_{n-1}(x_0, \cdots\cdots, x_{n-1}) \end{pmatrix} = 0 \qquad (5.1)$$

We then define for each variable x_i, a multi-resultant $R_i(x_i) = \det M_i(x_i)$ where M_i is a large sparse matrix known as the multi-resultant matrix. It has been shown in [47] that $R_i(x_i) = 0$ is a necessary condition on the i^{th} component of any zero of **f**. If z_i denotes zero of $R_i(x_i) = 0$, then any zero of **f** belongs to the Cartesian product

$$\prod_{i=0}^{n-1} z_i \qquad (5.2)$$

The actual zero points of **f** can be determined numerically from the set described in equation (5.2). Since the equation, $R_i(x_i) = 0$, is a high order polynomial equation, the problem of solving this equation is unstable. An alternate solution is to replace $R_i(x_i) = 0$ with the equivalent condition [45]

$$\min_{\|v\| = 1} \| M_i(x_i)v \|^2 = 0 \qquad (5.3)$$

where v is a column vector. Equation (5.3) has been shown to be numerically stable. The problem, thus, entails the calculation of the smallest eigenvalue of $M_i(x_i)^T M_i(x_i)$.

The procedure for constructing the multi-resultant matrix of a polynomial system is best illustrated through an example. For convenience, the variables x, y, z are used in place of x_1, x_2, and x_3. Consider the system of polynomials

$$p_o = x^2 + yz - 3y^2$$
$$p_1 = xy - 2z^2 \qquad (5.4)$$
$$p_2 = y^2 + yz - xz$$

Let d_i be the degree of p_i. For the system in (5.4), we have
$d_1 = d_2 = d_3 = 2$ and $n = 3$.
Let

$$L = 1 + \Sigma (d_i - 1) = 4 \qquad (5.5)$$

The basis for $V_{n,L}$, the vector space of homogeneous polynomials in n variables of degree L, is obtained as

$$\left(z^4, yz^3, y^2z^2, y^3z, y^4, xz^3, xyz^2, xy^2z, xy^3, x^2z^2, x^2yz, x^2y^2, x^3z, x^3y, x^4\right)$$

The sets s_i are then constructed by selecting monomials that are divisible by $x_i^{d_i}$. This gives

$$s_o = \left(x^2z^2, x^2yz, x^2y^2, x^3z, x^3y, x^4\right)$$
$$s_1 = \left(y^2z^2, y^3z, y^4, xy^2z, xy^3\right)$$
$$s_2 = \left(z^4, yz^3, xz^3, xyz^2\right)$$

The corresponding sets, T_i, obtained by dividing s_i by $x_i^{d_i}$, are

$$T_o = \left(z^2, yz, y^2, xz, xy, x^2 \right)$$
$$T_1 = \left(z^2, yz, y^2, xz, xy \right)$$
$$T_2 = \left(z^2, yz, xz, xy \right)$$

The muti-resultant matrix, M, is formed by multiplying elements of T_i by p_i and writing the coefficients in the reverse lexicographical order. For instance, the 8^{th} row of the multi-resultant matrix is formed by taking the second element of T_1 (skipping over the 6 elements in T_0 and the first element in T_1) and multiplying it by P_1 to get $xy^2 - 2yz^3$. Writing the coefficients in the reverse lexicographical form yields

(0 -2 0 0 0 0 0 1 0 0 0 0 0 0 0)

The remaining rows of the multi-resultant matrix can be built using the same method. The dimension of the matrix is m x m where

$$m = \begin{pmatrix} \sum d_i \\ n-1 \end{pmatrix} \qquad (5.6)$$

and n is the system dimension. The solutions to system (5.4) occurs at the minimum eigenvalues

$$\lambda_{min}(x_i) = \min_{||v|| = 1} || M_i(x_i)v ||^2 = 0 \qquad (5.7)$$

The solutions to system (5.4) are obtained by numerically searching for the minimum eigenvalue $\lambda_{min}(x_i)$ over the interval x_i in [a,b] where a and b are the interval bounds. The corresponding value of x_i is a solution to system (5.4).

The multi-resultant method, developed by Allgower, Georg, and Miranda, is theoretically capable of finding real valued solutions to polynomial systems having real coefficients. The application of this method for the training of neural networks, again, requires the nodal activation function to be modeled by a polynomial equation. However, the difficult task of generating a known system, g(x), is no longer necessary. The procedure for applying the multi-resultant method to the problem of training neural networks is described next.

5.3. Neural Network Example

The procedure involved in applying the multi-resultant method to neural networks is similar to that of the generalized polynomial approximation method described in Chapter 4. The nodal activation function is modeled using a polynomial approximation. The polynomial system to be solved is formulated using equations (2.7) and (2.11). This system is described by equation

$$f(w) = \begin{pmatrix} \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{pmatrix} = 0 \qquad (5.8)$$

The multi-resultant matrix, M, is obtained using method previously described. A major disadvantage in applying the multi-resultant method to neural network training is the enormous dimensions of the multi-resultant matrix. In the case of the example given in section 4.1 where the system consists of six equations of fifth order polynomial functions, the dimensions of the resultant matrix, M, is obtained to be

$$m = \binom{30}{5} = 142506 \qquad (5.9)$$

Therefore, the resultant matrix has dimensions of 142506 x 142506. Although the multi-resultant matrix is sparse, the computational effort involved in calculating such a large matrix is still expensive. Hence the application of the multi-resultant method to practical neural network structure requires the development of numerical techniques for handling large sparse matrices. However, the method theoretically represents an approach for finding the globally optimum real solution of a polynomial system which is the general objective of this thesis.

CHAPTER VI.

RESULTS AND DISCUSSIONS

6.1. Introduction

This chapter demonstrates the validity of the fixed-point homotopy method for training neural networks. The performance of the fixed-point homotopy method for training a multilayer perceptron network is evaluated using a two class problem. The set of test problems in a two dimensional feature space with linear and nonlinear decision surfaces are shown in Figures 6.1, 6.2, 6.3, and 6.4. The sample patterns from the two classes used to train the network are also shown. The neural network used to classify the above problems is shown in Figure 6.5.

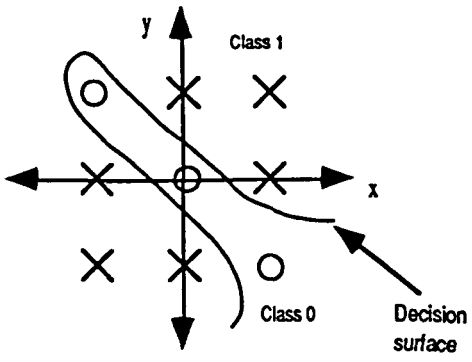


Figure 6.1 Input sample problem #1

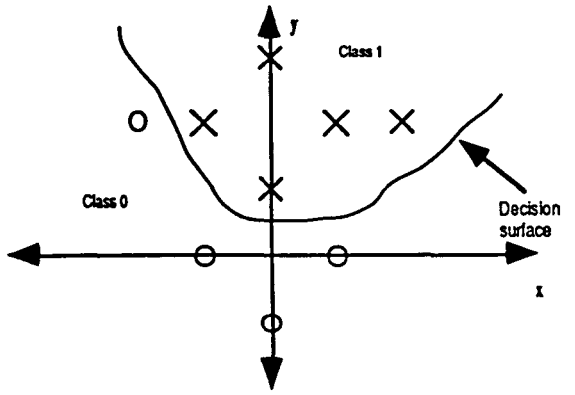


Figure 6.2 Input sample problem #2

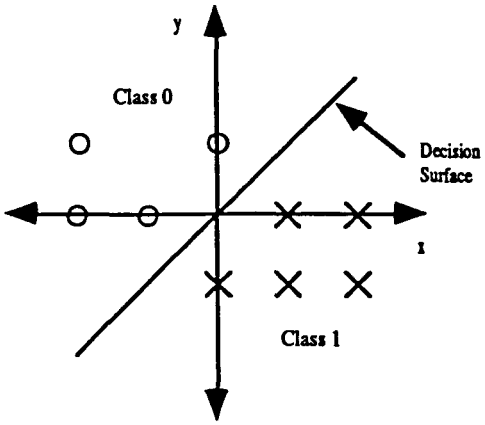


Figure 6.3 Input sample problem #3

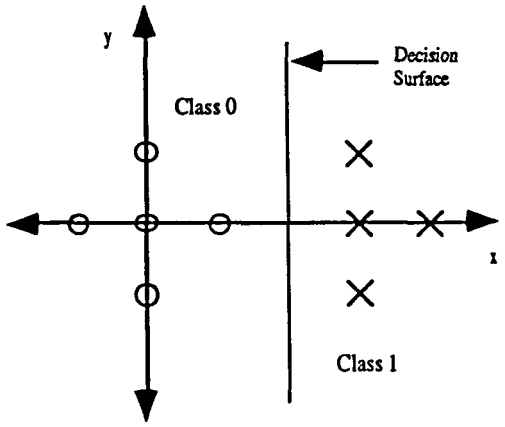


Figure 6.4 Input sample problem #4

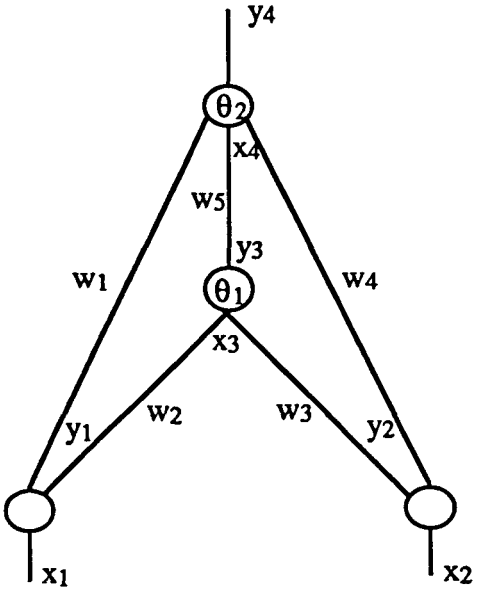


Figure 6.5 Experimental network

This network contains five interconnection weights designated w_1, ..., w_5 as shown in Figure 6.5. In addition, two bias variables, θ_1 and θ_2, are included to improve convergence [46]. θ_1 is assigned to the hidden node and θ_2 is assigned to the output node. The bias variables are adjusted along with the weights to produce a minimum energy in the output error.

6.2. Problem Formulation

Using the nodal activation function as given in equation (6.1),

$$y = \frac{1}{1 + e^{-(x+\theta)}} \tag{6.1}$$

the minimization equations to be solved for the output connections are,

$$\frac{\partial E}{\partial w_1} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial w_1} = 0 \tag{6.2}$$

$$\frac{\partial E}{\partial w_4} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial w_4} = 0 \tag{6.3}$$

$$\frac{\partial E}{\partial w_5} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial w_5} = 0 \tag{6.4}$$

Using the results in section 2.3, this set of equations reduces to

$$\frac{\partial E}{\partial w_1} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,y_1(c) = 0 \tag{6.5}$$

$$\frac{\partial E}{\partial w_4} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,y_2(c) = 0 \tag{6.6}$$

$$\frac{\partial E}{\partial w_5} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,y_3(c) = 0 \tag{6.7}$$

The corresponding equations for the hidden node connections are

$$\frac{\partial E}{\partial w_2} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial y_3(c)} \frac{\partial y_3(c)}{\partial x_3(c)} \frac{\partial x_3(c)}{\partial w_2} = 0 \tag{6.8}$$

$$\frac{\partial E}{\partial w_3} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial y_3(c)} \frac{\partial y_3(c)}{\partial x_3(c)} \frac{\partial x_3(c)}{\partial w_3} = 0 \tag{6.9}$$

which can be rewritten in the form

$$\frac{\partial E}{\partial w_2} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,w_5\,(1-y_3(c))\,y_3(c)\,y_1(c) = 0 \tag{6.10}$$

$$\frac{\partial E}{\partial w_3} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,w_5\,(1-y_3(c))\,y_3(c)\,y_2(c) = 0 \tag{6.11}$$

Similarly for the bias variables, we have

$$\frac{\partial E}{\partial \theta_1} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial x_4(c)} \frac{\partial x_4(c)}{\partial y_3(c)} \frac{\partial y_3(c)}{\partial \theta_1} = 0 \tag{6.12}$$

$$\frac{\partial E}{\partial \theta_2} = \sum_c \frac{\partial E}{\partial y_4(c)} \frac{\partial y_4(c)}{\partial \theta_2} = 0 \tag{6.13}$$

which can be rewritten as

$$\frac{\partial E}{\partial \theta_1} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c)\,w_5\,(1-y_3(c))\,y_3(c) = 0 \tag{6.14}$$

$$\frac{\partial E}{\partial \theta_2} = \sum_c (y_4(c) - d(c))\,(1-y_4(c))\,y_4(c) = 0 \tag{6.15}$$

The system to be solved can be finally represented in vector form as

$$f(w) = \begin{pmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_5} \\ \frac{\partial E}{\partial \theta_1} \\ \frac{\partial E}{\partial \theta_2} \end{pmatrix} = 0 \tag{6.16}$$

with the fixed-point homotopy function being of the form

$$h(w,t) = f(w)\,t + (w - w_o)\,(1 - t) \tag{6.17}$$

The fixed-point homotopy continuation method was implemented using a numerical predictor-corrector scheme with t as the tracking parameter. The training patterns and the same initial weights were also input to the gradient descent method. The results are summarized in Table 1.

6.3. Discussions

The results shown in Table 1 indicate that the fixed-point homotopy method has considerable potential as a tool for training neural networks. The major advantages offered by this technique are global convergence and significantly lower training time.

The backpropagation algorithm has been used extensively for the classification of data. However, parameters such as initial weights, bias value, and learning rate have to be chosen on the basis of trial and error. Very often this results in excessive training time before the appropriate combination of initial parameters are found. In this regard, the global convergence property of the fixed point homotopy method is particularly advantageous. Currently, the fixed-point method requires several initial points to be evaluated before the desired solution can be obtained. However, this problem can be overcome once the all-solution homotopy method is implemented. The all-solution homotopy method allows the optimal solution to be found in one iteration.

In the case of linearly separable problems, both methods converged to the desired solution for most initial values. However, for larger values of

Table 1. Summary of Simulation Results

Pattern Number	Desired Output	Homotopy Method	Gradient Method
1	0.0000	0.1136	0.0000
	1.0000	0.9785	0.0000
	1.0000	0.9014	0.0000
	1.0000	0.8790	0.0000
	0.0000	0.1095	0.0000
	1.0000	0.9794	0.0000
	1.0000	0.9979	1.0000
	1.0000	0.8739	0.9570
	0.0000	0.1055	0.0000
2	1.0000	0.9899	0.0000
	1.0000	0.9709	0.0000
	1.0000	0.9981	0.0000
	1.0000	0.8719	0.0000
	0.0000	0.0285	0.0000
	1.0000	0.9995	0.0000
	0.0000	0.0076	0.9999
	0.0000	0.1456	0.9999
	0.0000	0.0396	0.9999

Table 1. (continued)

Pattern Number	Desired Output	Homotopy Method	Gradient Method
3	0.0000	0.0000	0.9999
	0.0000	0.1218	0.9999
	0.0000	0.0000	0.9999
	0.0000	0.0026	0.9999
	1.0000	0.9999	0.9999
	1.0000	0.9999	0.9999
	1.0000	0.9999	0.9999
	1.0000	0.9999	0.9999
	1.0000	1.0000	1.0000
4	0.0000	0.0050	0.0003
	0.0000	0.0026	0.0002
	0.0000	0.0249	0.0009
	0.0000	0.2187	0.0040
	0.0000	0.1265	0.0028
	1.0000	0.9999	0.9986
	1.0000	0.9999	0.9996
	1.0000	0.9999	0.9999
	1.0000	0.9999	0.9984

tinitial weights, the gradient method failed to converge, whereas the fixed-point homotopy method is still capable of converging to the desired solution as seen in patterns #3 and #4. As the decision surface becomes more and more complex, the gradient method, in general, fails more frequently than the fixed-point homotopy method. The performance of the two methods for data set #1 and #2, starting from the same initial points, are shown in Table 1.

However, the merit of the backward error propagation method still lies in the ease with which it can handle higher dimensional problems associated with multilayered networks. Further work remains to be done before the fixed-point homotopy method can be adapted for training layer networks of larger dimesnsions.

CHAPTER VII.

SUMMARY AND FUTURE WORK

This thesis presents an innovative approach for training multilayer perceptron neural networks. The overall objective is to achieve the globally minimum error which ensures the best possible training parameters for subsequent classifications. The network parameters include the interconnection weight variables and the nodal bias variables.

A brief description of the topic of neural networks is presented. The concept of neural networks is based on an extremely simplified model of the human nervous system. Each neuron in the network can generate one of two outputs: an output one which signifies the firing state of a neuron cell or an output zero which represents the resting state of a neuron cell. The network is first trained by presenting the input sample patterns along with the desired output to the network. The discrepancy between the network calculated output and desired output represents the training error which is used to adjust the interconnection weights so that the energy in the error is minimized. Consequently, it is essential that the minimum error obtained is a global minimum and not a local minimum. The training technique most commonly used to find the minimum point is the backward error propagation algorithm, which is based on the gradient descent method. Two major criticisms of the backward error propagation algorithm are: 1) Excessive training time, and 2)

lack of convergence, or convergence to the local minimum of the error surface.

The approach proposed in this thesis for overcoming these drawbacks involves the use of the homotopy continuation method for minimizing the error function. Homotopy continuation methods are numerical methods used to obtain the zeros of nonlinear systems. Homotopy methods offer two important advantages, namely, global convergence and when the system of interest consists of polynomial functions, all-solution homotopy methods guarantee exhaustive solutions. However, in order to apply the all-solution method to neural network training, the sigmoidal activation function must be modeled by a polynomial function. The drawback with this approach is the difficulty in generating the known polynomial system of equations, g(x), with only real-valued solutions. However, at the expense of losing the all-solution property, a variant of the homotopy method known as the fixed-point homotopy method can be used. This method is globally convergent and it tracks one solution at a time. The fixed-point homotopy method is often used in problems where the number of solutions is not known a priori. Moreover, the fixed-point method is particularly used in applications where only real valued solutions are desired, as in the case of neural networks.

The contribution of this research lies in the formulation of the neural network equations within a framework suitable for the application of the homotopy continuation method. Application of the fixed-point homotopy method was performed by computing the Jacobian matrix and developing the program codes for the predictor-corrector numerical scheme.

Simulation results for some two dimensional, two class data sets have been presented. The results demonstrate the capability of the proposed method to converge to the desired minimum in contrast to the gradient method which was seen to result in oscillation or converged to a local minimum.

In conclusion, the fixed-point homotopy method has been demonstrated to be superior to gradient search methods largely due to its global convergence property. However, the work presented in this thesis is by no means complete. Further research remains to be done before the proposed method can be applied for finding the globally optimum weights in training neural networks. This requires development of robust techniques for generating the polynomial system, g(x). Once this problem is resolved, the all-solution homotopy method can be employed to yield the globally optimum solution with respect to the classification error. Another area that needs further attention is the development of numerical techniques for implementing the multi-resultant method for training neural networks. Finally, the application of homotopy methods for training neural networks other than the multilayer perceptron network, such as the Kohonen network needs to be studied.

REFERENCES

[1] J. T. Tou and R. C. Gonzalez, Pattern Recognition Principles, Addison-Wesley, Reading, Mass, 1974

[2] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Imminent in Nervous Activity," Bulletin of Mathematical Biophysics, 5, 1943, pp. 115-133

[3] D. O. Hebb, The Organization of Behavior, John Wiley & Sons, New York, NY, 1949

[4] R. Rosenblatt, Principles of Neurodynamics, Spartan Books, New York, NY, 1959

[5] R. P. Lippmann, "An Introduction to Computing With Neural Nets," IEEE ASSP Magazine, April 1987, pp. 4-22

[6] J. Chow, "Pattern Recognition Using Singular Value Decomposition," EE-653 Project Report, Colorado State University, 1990

[7] D. C. Plaut, S. J. Nowlan and G. E. Hinton, "Experiments on Learning by Back Propagation," CMU-CS-86-126, Carnegie Mellon University, Pittsburgh, PA, June 1986

[8] V. L. Stonick, "Global Methods of Pole/Zero Modeling for Digital Signal Processing using Homotopy Continuation Methods," Ph.D. Dissertation, North Carolina State University, 1989

[9] E. L. Allogower and K. Georg, Introduction to Numerical Continuation Methods, Springer-Verlag, Berlin, 1990

[10] M. F. Kaplan and S. Schwartz, Human Judgement and Decision Processes, Academic Press, New York, NY, 1975

[11] P. K. Simpson, Artificial Neural Systems, Pergamon Press, Elmsford, NY, 1990

[12] D. E. Rumelhart and J. L. McClelland, Parallel Distributed Processing, Vol. I, Massachusetts Institute of Technology, Cambridge, MASS, 1989

[13] T. Y. Li, "On Chow, Mallet-Paret, and Yorke Homotopy for Solving Systems of Polynomial," Bull. Inst. Math. Acad. Sinica, 1983, pp. 433-437

[14] J. J. Hopfield, "Neural Networks and Physical Systems With Emergent Collective Computational Abilities," Proceedings of National Acad. Sci., Vol. 79, April 1982, pp. 2554-2558

[15] T. Kohonen, Self-Organization and Associative Memory, Berlin, Springer-Verlag, 1984

[16] R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, John Wiley & Sons, New York, NY, 1973

[17] J. A. Hartigan, Clustering Algorithms, John Wiley & Sons, New York, NY, 1975

[18] C. B. Garcia and W. I. Zangwill, "Determining All Solutions to Certain Systems of Nonlinear Equations," in Mathematics of Operations Research, Vol. 4, Feb. 1979, pp. 1-14

[19] C. B. Garcia and W. I. Zangwill, "Finding All Solutions to Polynomial Systems and Other Systems of Equations," in Mathematical Programming, Vol. 16, 1979, pp. 159-176

[20] A. P. Morgan, Solving Polynomial Systems Using Continuation for Engineering and Scientific Computations, Prentice-Hall, Englewood Cliffs, NJ, 1987

[21] W. Press, et al., Numerical Recipes, Cambridge University Press, Cambridge, 1989

[22] C. B. Garcia and W. I. Zangwill, "Global Continuation Methods for Finding All Solutions to Polynomial Systems of Equations in N Variables," Center for Math Studies in Business and Economics Report No. 755, University of Chicago, 1977

[23] F. J. Drexler, "A Homotopy Method for The Calculation of Zeroes of Zero-Dimensional Polynomial Ideals," in Continuation Methods (H. G. Wacker, Ed.), Academic, NY, 1978, pp. 69-94

[24] C. B. Garcia and F. J. Gould, "Relations Between Several Path Following Algorithms and Local and Global Newton Methods," SIAM Review, Vol. 22, July 1980, pp. 263-274

[25] H. B. Keller, "Global Homotopies and Newton's Methods," in Recent Advances in Numerical Analysis, Academic Press, 1978, pp. 73-94

[26] S. N. Chow, J. Mallet-Paret and J. A. Yorke, "A Homotopy Method for Locating All Zeroes of A System of Polynomials," in *Functional Differential Equations and Approximation of Fixed Points*, Lecture Notes in Math 730, Springe, N. W., 1979

[27] A. H. Wright, "Finding All Solutions to A System of Polynomial Equations," Math. Comp., Vol. 44, 1985, pp. 125-133

[28] A. P. Morgan, "A Transformation to Avoid Solutions at Infinity for Polynomial Systems," Appl. Math. Comput., Vol. 18, 1986, pp.77-86

[29] L. W. Tsai and A. P. Morgan, "Solving The Kinematics of The Most General Six-and Five-Degree of Freedom Manipulators by Continuation Methods," ASME J. Mechanisms, Transmissions and Automation in Design, 1985, pp. 48-57

[30] K. Meintjes and A. P. Morgan, "A Methodology for Solving Chemical Equilibrium Systems," Appl. Math. Comput., 1987, pp. 333-361

[31] C. B. Garcia and W. I. Zangwill, Pathways to Solutions, Fixed Points and Equilibria, Prentice-Hall, Englewood Cliffs, NJ, 1981

[32] C. B. Garcia and T. Y. Li, "On the Number of Solutions to Polynomial Systems of Equations," SIAM J. Numer. Anal., Vol. 17, Aug. 1980, pp. 540-546

[33] V. L. Stonick and S. T. Alexander, "On Reduced Computations for Global IIR Filtering," ISCAS 1990, New Orleans, 1990, pp.1327-1331

[34] V. L. Stonick and S. T. Alexander, "ARMA Parameter Estimation Using Continuation Methods," Proceedings: 22nd Asilomar Conf. Sig. Syst. Comput., Monterey, Ca, Oct. 31 - Nov. 1 1988, pp. 184-188

[35] L. T. Watson, et al., "Globally Convergent Homotopy Methods for The DC Operating Point Problem," Department of Computer Science, Virginia Polytechnic Institute and State University, 1990

[36] G. Vasudevan, et al., "Homotopy Approach for Solving Constrained Optimization Problems," IEEE Trans. on Auto. Control, Vol. 36, April 1991, pp. 494-498

[37] E. Ikeno and A. Ushida, "The Arc-Length Method for the Computation of Characteristic Curves," IEEE Trans. Circuit Syst., Vol. 23, 1976, pp. 181-183

[38] E. L. Allgower and K. Georg, "Predictor-Corrector and Simplicial Methods for Approximating Fixed Points and Zero Points of Nonlinear Mappings," in Mathematical Programming, The State of The Art, Springer-Verlag, 1983, pp. 15-56

[39] L. T. Watson, S. C. Billups and A. P. Morgan, "HOMPACK: A Suite of Codes for Globally Convergent Homotopy Algorithms," ACM Trans.on Math., Sept. 1987, pp. 281-310

[40] J. Chow, L. Udpa and S. Udpa, "Homotopy Continuation Methods for Neural Networks," ISCAS Conference, Singapore, June 1991, pp 2483-2486

[41] J. Chow, L. Udpa and S. Udpa, "Neural Network Training Using Homotopy Continuation Methods," to appear in IJCNN, Singapore, Nov. 1991

[42] G. H. Golub and C. F. Van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, MD, 1983

[43] E. L. Allgower, Oral Discussion, Colorado State University, Dec. 1990

[44] E. L. Allgower, K. Georg and R. Miranda, "Computing Real Solutions of Polynomial Systems," Department of Mathematics, Colorado State University, 1990

[45] E. L. Allgower, K. Georg and R. Miranda, "The Method of Resultants for Computing Real Solutions of Polynomial Systems," to appear in SIAM J. Numer. Anal.

[46] D. E. Rumhelhart and J. L. McClelland, Parallel Distributed Processing, Vol. II, Massachutts Institute of Technology, Cambridge, MASS, 1989

[47] B. L. van der Waerden, Modern Algebra, Volume II, Springer-Verlag, Berlin, 1940

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my adviser, Dr. L. Udpa, for her continued guidance and encouragement throughout this research.

I would also like to extend my gratitude to Dr. S. Udpa for his knowledgeable insights which have helped me in overcoming many obstacles throughout my work. The most important thing I have learned in working with Dr. L. Udpa and Dr. S. Udpa is not about homotopy theory, nor is it about neural networks, instead, it is about how to be tolerant and forgiving toward other people. I am deeply indebted to them for showing me the more important things in life.

Further, I would like to thank Dr. E. Bartlett, Dr. J. Cornette, Dr. J. Davidson, and Dr. W. Lord for their generous assistance in reviewing my thesis. In addition, I would like to thank Dr. E. Allgower for his insightful suggestions. I would also like to thank my fellow graduate student, Mike Chan, for his help throughout my work. Not only did I gain valuable knowledge about UNIX, but most importantly, I have gained a life-long friend.

Next, I would like to thank the dearest person in my life, Anastasia Waguespack, for her support and understanding. She was there beside me every step along the way as we have vowed to do for each other.

Finally, I would like to thank my family, especially my Mom and Dad for their encouragement and support, both financially and spiritually. They have devoted their lives toward the making of my success and have sacrificed so much in doing so.

APPENDIX A.

NEURAL NETWORK TRAINING PROGRAM USING FIXED-POINT HOMOTOPY METHOD

```
program igiveup

c       This program trains a multilayer perceptron neural network
c       using the fixed-point homotopy method. The training algorithm
c       employed is the conventional error back propagation algorithm.
c       This program does not employ the numerical predictor-corrector
c       scheme, instead, the tracking parameter t is incremented based
c       upon the user-input step size. The iterative tracking process
c       is stopped when t is equal to or greater than 1.
c
```

```
**********************************************************************
**********************************************************************
****                                                              ****
****  Author:          Joseph C. Chow                             ****
****  Thesis Topic:    Neural Networks Using                      ****
****                   Homotopy Continuation Methods              ****
****  Date:            06/11/1991                                 ****
****  Time:            10:56:23 AM                                ****
****                                                              ****
**********************************************************************
**********************************************************************

======================================================================
======================================================================
====                                                              ====
====                                                              ====
====                    Variable Dictionary                       ====
====                                                              ====
====                                                              ====
==== a      =    Dummy array used for passing arguments           ====
```

====		to and from LINPACK subroutines. ====
==== b	=	Dummy array used for passing arguments ====
====		to and from LINPACK subroutines. ====
==== change	=	Amount of correction added to the ====
====		weight vectors at each iteration. ====
==== constant	=	Array representing the initial starting ====
====		points. ====
==== d	=	Desired outputs. ====
==== det	=	Determinant of the matrix used by ====
====		LINPACK subroutines. ====
==== df	=	Jacobian matrix of the unknown system. ====
==== dg	=	Jacobian matrix of the known system. ====
==== dh	=	Jacobian matrix of the homotopy system. ====
==== dick3	=	Dummy function name for the sigmoid ====
====		function. ====
==== error	=	The training error. ====
==== f	=	Array representing the system to be ====
====		solved. ====
==== flag	=	Flag used to determine which iteration. ====
==== g	=	Array representing the initial known ====
====		system. ====
==== h	=	Array representing the homotopy system. ====
==== hessold	=	Dummy array representing the Hessian ====
====		matrix of the system in interest. ====
==== rcond	=	Condition number of a matrix, used by ====
====		the LINPACK subroutines. ====
==== i	=	Dummy counter variable. ====
==== ipvt	=	Array variable used by the LINPACK ====
====		subroutines. ====
==== j	=	Dummy counter variable. ====
==== lda	=	Dummy variable used by the LINPACK ====
====		subroutines. ====
==== n	=	Number of variables in the system. ====
==== step	=	Step size to increment the tracking ====
====		parameter, t. ====
==== t	=	Tracking parameter ====
==== w	=	Array representing the five weights and ====
====		two bias variables in the network. ====
==== x3	=	Array representing different sample ====
====		cases of input x3. ====

x4	=	Array representing different sample cases of input x4.
y1	=	Array representing different sample cases of output y1.
y2	=	Array representing different sample cases of output y2.
y3	=	Array representing different sample cases of output y3.
y4	=	Array representing different sample cases of output y4.
z	=	Dummy array variable used by LINPACK subroutines.

Subroutines Called

dfn	=	Subroutine used to calculate the Jacobian matrix of the system in interest.
dfn1	=	Same subroutine as dfn except only called during the first iteration of the tracking process.
dgeco	=	LINPACK subroutine. Please see LINPACK manual for further information.
dgesl	=	LINPACK subroutine. Please see LINPACK manual for further information.
fcn1	=	Subroutine used to evaluate the minimization equation derived from w1.
fcn2	=	Subroutine used to evaluate the minimization equation derived from w2.
fcn3	=	Subroutine used to evaluate the minimization equation derived from w3.
fcn4	=	Subroutine used to evaluate the

```
====                minimization equation derived from w4.                ====
====  fcn5  =       Subroutine used to evaluate the                       ====
====                minimization equation derived from w5.                ====
====  fcn6  =       Subroutine used to evaluate the                       ====
====                minimization equation derived from theta1.            ====
====  fcn7  =       Subroutine used to evaluate the                       ====
====                minimization equation derived from theta2.            ====
====                                                                      ====
================================================================
================================================================

================================================================
================================================================
====                                                              ====
====                        File Names                            ====
====                                                              ====
====                                                              ====
====  inputs  =     Input file where the input sample               ====
====                patterns and the desired outputs are          ====
====                read.                                         ====
====  outputp =     Output file where the final weight              ====
====                values, bias values, and error level          ====
====                are stored.                                   ====
====  roots   =     Input file where the initial starting           ====
====                points are read.                              ====
====                                                              ====
================================================================
================================================================
```

c Declare variables

```
      double precision a(7,7),b(7),change(7)
      double precision constant(7),d(9),det,df(7,7)
      double precision dg(7,7),dh(7,8),dick3
      double precision error,f(7),g(7),h(7),hessold(7,7)
      double precision rcond,step,t,w(7),work(7),x3(9),x4(9)
      double precision y1(9),y2(9),y3(9),y4(9),z(7)

      integer flag,i,ipvt(7),j,n
```

```
      data lda /7/

c Define the nodal sigmoid function

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c Open data files

      open(unit=7,file='inputs')
      open(unit=8,file='roots')
      open(unit=9,file='outputp')

c Read input vector and desired outputs

      read(7,*)y1(1),y2(1),d(1),y1(2),y2(2),d(2)
      read(7,*)y1(3),y2(3),d(3),y1(4),y2(4),d(4)
      read(7,*)y1(5),y2(5),d(5),y1(6),y2(6),d(6)
      read(7,*)y1(7),y2(7),d(7),y1(8),y2(8),d(8)
      read(7,*)y1(9),y2(9),d(9)
      close(7)

c Define number of variables in the system

      n=7

c Read the initial starting point. The starting point
c is also used to form the known system.

5     do 10 j=1,n
        read(8,*)w(j)
        constant(j)=w(j)
10    continue

      flag=1
```

```
c User input incrementing step size of t

      write(6,*)'input step size'
      read(5,*)step
      write(6,*)''


c this part of program will begine the tracking process

      do 10000 t=0.d0,1.d0,step

      write(6,*)'t =',t


c reset f, g, h, df, dg, dh

20    do 32 i=1,n
        do 30 j=1,n
          df(i,j)=0.d0
          dg(i,j)=0.d0
          dh(i,j)=0.d0
30      continue
        f(i)=0.d0
        g(i)=0.d0
        h(i)=0.d0
32    continue

c compute the unknown system f(w)

      call fcn1(n,y1,y2,d,f(1),w)
      call fcn2(n,y1,y2,d,f(2),w)
      call fcn3(n,y1,y2,d,f(3),w)
      call fcn4(n,y1,y2,d,f(4),w)
      call fcn5(n,y1,y2,d,f(5),w)
      call fcn6(n,y1,y2,d,f(6),w)
      call fcn7(n,y1,y2,d,f(7),w)

c compute the first iteration of df
```

```
      if (flag .eq. 1) then
       call dfn1(n,y1,y2,d,hessold,w)
       flag=0
      endif

c compute df

      call dfn(n,y1,y2,d,df,hessold,w)

c compute the known system g(w)

      g(1)=w(1)-constant(1)
      g(2)=w(2)-constant(2)
      g(3)=w(3)-constant(3)
      g(4)=w(4)-constant(4)
      g(5)=w(5)-constant(5)
      g(6)=w(6)-constant(6)
      g(7)=w(7)-constant(7)

c compute the derivative dg


      do 40 i=1,n
       dg(i,i)=1.d0
40    continue


c setting up the homotopic equations

      do 50 i=1,n
       h(i)=(1.d0 - t)*g(i) + t*f(i)
50    continue

c calculate the Jacobian of the homotopy functions

      do 75 i=1,n
       do 70 j=1,n
        dh(i,j)=(1.d0 - t)*dg(i,j) + t*df(i,j)
        a(i,j)=dh(i,j)
70     continue
```

```
75      continue


        call dgeco(a,lda,n,ipvt,rcond,z)
        if (rcond .eq. 0.0) goto 10020

c assign the B column vector

        do 120 i=1,n
          b(i)=-h(i)
120     continue

        call dgesl(a,lda,n,ipvt,b,0)


        do 140 i=1,n
          change(i)=b(i)
140     continue

c Adjust the weight vectors based on the change calculated

        do 200 i=1,n
          w(i)=w(i)+change(i)
200     continue

        do 300 i=1,n
        if(dabs(change(i)) .gt. 0.001) goto 20
300     continue

10000         continue

        write(6,*)"

c This portion of the program calculates the output of the network
c and also the square error associated with the calculated weights

c
c compute input to layer 2
```

```
      do 9010 i=1,9
       x3(i)=w(2)*y1(i) + w(3)*y2(i)
9010  continue


c compute output to layer 2

      do 9040 i=1,9
       y3(i)=dick3(x3(i),w(6))
9040  continue

      do 9060 i=1,9
       x4(i)=w(1)*y1(i)+w(5)*y3(i)+w(4)*y2(i)
9060  continue

      do 9080 i=1,9
       y4(i)=dick3(x4(i),w(7))
9080  continue


c check for weights that yield the lowest error

      error=0.d0

      do 9085 i=1,9
       error=0.5d0*(y4(i)-d(i))**2.d0+error
9085  continue

c PRINT OUT THE RESULTS!!

      write(9,*)'The error is: ',error
      write(6,*)'The error is: ',error
      write(9,*)''
      write(9,*)'y4(1) =',y4(1)
      write(9,*)'y4(2) =',y4(2)
      write(9,*)'y4(3) =',y4(3)
      write(9,*)'y4(4) =',y4(4)
      write(9,*)'y4(5) =',y4(5)
      write(9,*)'y4(6) =',y4(6)
      write(9,*)'y4(7) =',y4(7)
```

```
      write(9,*)'y4(8) =',y4(8)
      write(9,*)'y4(9) =',y4(9)
      write(9,*)''
      write(9,*)'w1 =',w(1)
      write(9,*)'w2 =',w(2)
      write(9,*)'w3 =',w(3)
      write(9,*)'w4 =',w(4)
      write(9,*)'w5 =',w(5)
      write(9,*)'theta1 =',w(6)
      write(9,*)'theta2 =',w(7)
      write(9,*)''
c     write(9,*)''

      goto 10030

c Display error message

10020       write(6,*)'On set #',count
      write(6,*)'singular matrix was encountered!'


10030       continue

      stop
      end
```

```
      subroutine fcn1(n,y1,y2,d,f1,w)

c This subroutine evaluates the minimization equation
c derived from w1.

c Please see main program for definition of variables
c passed.

      double precision dick3,y1(9),y2(9),d(9),f1,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c     output connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
       x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
       y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
       x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
       y4(i)=dick3(x4(i),w(7))
40    continue

      f1=0.d0
      do 50 i=1,9
```

```
        dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*y1(i)
        f1=f1+dummy(i)
50      continue

        return
        end
```

```
      subroutine fcn2(n,y1,y2,d,f2,w)

c This subroutine evaluates the minimization equation
c derived from w2.

c Please see main program for definition of variables
c passed.


      double precision dick3,y1(9),y2(9),d(9),f2,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))
c
c     hidden connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
       x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
       y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
       x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
       y4(i)=dick3(x4(i),w(7))
40    continue

      f2=0.d0
```

```
      do 50 i=1,9
       dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*w(5)*
     &         (1.d0-y3(i))*y3(i)*y1(i)
       f2=f2+dummy(i)
50    continue

      return
      end
```

```
      subroutine fcn3(n,y1,y2,d,f3,w)

c This subroutine evaluates the minimization equation
c derived from w3.

c Please see main program for definition of variables
c passed.

      double precision dick3,y1(9),y2(9),d(9),f3,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c     hidden connection


      n=7

c     compute input to layer 2

      do 10 i=1,9
        x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
        y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
        x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
        y4(i)=dick3(x4(i),w(7))
40    continue

      f3=0.d0
```

```
      do 50 i=1,9
       dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*w(5)*
     &         (1.d0-y3(i))*y3(i)*y2(i)
       f3=f3+dummy(i)
50    continue

      return
      end
```

```
      subroutine fcn4(n,y1,y2,d,f4,w)

c This subroutine evaluates the minimization equation
c derived from w4.

c Please see main program for definition of variables
c passed.


      double precision dick3,y1(9),y2(9),d(9),f4,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c     output connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
       x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
       y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
       x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
       y4(i)=dick3(x4(i),w(7))
40    continue

      f4=0.d0
```

```
      do 50 i=1,9
        dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*y2(i)
        f4=f4+dummy(i)
50    continue

      return
      end
```

```
      subroutine fcn5(n,y1,y2,d,f5,w)

c This subroutine evaluates the minimization equation
c derived from w5.

c Please see main program for definition of variables
c passed.

      double precision dick3,y1(9),y2(9),d(9),f5,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c     output connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
       x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
       y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
       x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
       y4(i)=dick3(x4(i),w(7))
40    continue

      f5=0.d0
```

```
      do 50 i=1,9
       dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*y3(i)
       f5=f5+dummy(i)
50    continue

      return
      end
```

```
      subroutine fcn6(n,y1,y2,d,f6,w)

c This subroutine evaluates the minimization equation
c derived from theta1.

c Please see main program for definition of variables
c passed.


      double precision dick3,y1(9),y2(9),d(9),f6,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c
c     hidden connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
        x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
        y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
        x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
        y4(i)=dick3(x4(i),w(7))
40    continue

      f6=0.d0
```

```
      do 50 i=1,9
       dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)*w(5)*
     &          (1.d0-y3(i))*y3(i)
       f6=f6+dummy(i)
50    continue

      return
      end
```

```
subroutine fcn7(n,y1,y2,d,f7,w)

c This subroutine evaluates the minimization equation
c derived from theta2.

c Please see main program for definition of variables
c passed.

      double precision dick3,y1(9),y2(9),d(9),f7,w(7)
      double precision dummy(9)
      double precision x3(9),y3(9),x4(9),y4(9)
      integer n

      dick3(x,y)=1.d0/(1.d0+exp(-x-y))

c     output connection

      n=7

c     compute input to layer 2

      do 10 i=1,9
       x3(i)=w(2)*y1(i)+w(3)*y2(i)
10    continue

      do 20 i=1,9
       y3(i)=dick3(x3(i),w(6))
20    continue

      do 30 i=1,9
       x4(i)=w(1)*y1(i)+w(4)*y2(i)+w(5)*y3(i)
30    continue

      do 40 i=1,9
       y4(i)=dick3(x4(i),w(7))
40    continue

      f7=0.d0
      do 50 i=1,9
```

```
        dummy(i)=(y4(i)-d(i))*(1.d0-y4(i))*y4(i)
        f7=f7+dummy(i)
50      continue

        return
        end
```

```
      subroutine dfn1(n,y1,y2,d,hessold,w)


c This subroutine computes the Jacobian matrix of the
c system in interest. This program is only called during
c the first iteration of each tracking process. This is
c used to provide numerical stability in subsequent calls
c of subroutine dfn.
c
c Please see main program for definition of variables passed.


      integer i,n
      double precision dummy(7),y1(9),y2(9),d(9),w(7)
      double precision f,for,rev,hstep,hessold(7,7)

      n=7
      do 20 i=1,n
       dummy(i)=w(i)
20    continue

      hstep = 1.d-5

c     compute first iteration of hessian


c Compute derivative with respect to w1

      do 40 i=1,n
       w(i)=dummy(i)+hstep
       call fcn1(n,y1,y2,d,f,w)
       for=f
       w(i)=dummy(i)-hstep
       call fcn1(n,y1,y2,d,f,w)
       rev=f
       hessold(1,i)=(for-rev)/(2.d0*hstep)
40    continue

c Compute derivative with respect to w2
```

```
      do 60 i=1,n
       w(i)=dummy(i)+hstep
       call fcn2(n,y1,y2,d,f,w)
       for=f
       w(i)=dummy(i)-hstep
       call fcn2(n,y1,y2,d,f,w)
       rev=f
       hessold(2,i)=(for-rev)/(2.d0*hstep)
60    continue

c Compute derivative with respect to w3

      do 80 i=1,n
       w(i)=dummy(i)+hstep
       call fcn3(n,y1,y2,d,f,w)
       for=f
       w(i)=dummy(i)-hstep
       call fcn3(n,y1,y2,d,f,w)
       rev=f
       hessold(3,i)=(for-rev)/(2.d0*hstep)
80    continue


c Compute derivative with respect to w4

      do 100 i=1,n
       w(i)=dummy(i)+hstep
       call fcn4(n,y1,y2,d,f,w)
       for=f
       w(i)=dummy(i)-hstep
       call fcn4(n,y1,y2,d,f,w)
       rev=f
       hessold(4,i)=(for-rev)/(2.d0*hstep)
100   continue


c Compute derivative with respect to w5

      do 120 i=1,n
       w(i)=dummy(i)+hstep
```

```
        call fcn5(n,y1,y2,d,f,w)
        for=f
        w(i)=dummy(i)-hstep
        call fcn5(n,y1,y2,d,f,w)
        rev=f
        hessold(5,i)=(for-rev)/(2.d0*hstep)
120     continue

        do 130 i=1,n
c Compute derivative with respect to w6

        w(i)=dummy(i)+hstep
        call fcn6(n,y1,y2,d,f,w)
        for=f
        w(i)=dummy(i)-hstep
        call fcn6(n,y1,y2,d,f,w)
        rev=f
        hessold(6,i)=(for-rev)/(2.d0*hstep)
130     continue

c Compute derivative with respect to w7

        do 135 i=1,n
        w(i)=dummy(i)+hstep
        call fcn7(n,y1,y2,d,f,w)
        for=f
        w(i)=dummy(i)-hstep
        call fcn7(n,y1,y2,d,f,w)
        rev=f
        hessold(7,i)=(for-rev)/(2.d0*hstep)
135     continue


        do 140 i=1,n
        w(i)=dummy(i)
140     continue

        return
        end
```

```
      subroutine dfn(n,y1,y2,d,df,hessold,w)
c
c This subroutine computes the jacobian matrix of
c the system in interest.
c
c See main program for definition of variables passed.


      integer i,n
      double precision w(7),hessold(7,7),df(7,7)
      double precision y1(9),y2(9),d(9),for,rev,hstep
      double precision f,dummy(7)

      n=7

      hstep =1.d-5

      do 20 i=1,n
       dummy(i)=w(i)
20    continue

c     compute the gradient (analytical) wrt w1

      do 30 i=1,n
       w(i)=dummy(i)+hstep
       call fcn1(n,y1,y2,d,f,w)
       for=f
       w(i)=dummy(i)-hstep
       call fcn1(n,y1,y2,d,f,w)
       rev=f
       df(1,i)=(for-rev)/(2.d0*hstep)
       df(1,i)=hessold(1,i)+0.1d0*(df(1,i)-hessold(1,i))
       hessold(1,i)=df(1,i)
30    continue


c     compute the gradient (analytical) wrt w2

      do 40 i=1,n
       w(i)=dummy(i)+hstep
```

```
      call fcn2(n,y1,y2,d,f,w)
      for=f
      w(i)=dummy(i)-hstep
      call fcn2(n,y1,y2,d,f,w)
      rev=f
      df(2,i)=(for-rev)/(2.d0*hstep)
      df(2,i)=hessold(2,i)+0.1d0*(df(2,i)-hessold(2,i))
      hessold(2,i)=df(2,i)
40    continue

c     compute the gradient (analytical) wrt w3


      do 60 i=1,n
      w(i)=dummy(i)+hstep
      call fcn3(n,y1,y2,d,f,w)
      for=f
      w(i)=dummy(i)-hstep
      call fcn3(n,y1,y2,d,f,w)
      rev=f
      df(3,i)=(for-rev)/(2.d0*hstep)
      df(3,i)=hessold(3,i)+0.1d0*(df(3,i)-hessold(3,i))
      hessold(3,i)=df(3,i)
60    continue


c     compute the gradient (analytical) wrt w4

      do 80 i=1,n
      w(i)=dummy(i)+hstep
      call fcn4(n,y1,y2,d,f,w)
      for=f
      w(i)=dummy(i)-hstep
      call fcn4(n,y1,y2,d,f,w)
      rev=f
      df(4,i)=(for-rev)/(2.d0*hstep)
      df(4,i)=hessold(4,i)+0.1d0*(df(4,i)-hessold(4,i))
      hessold(4,i)=df(4,i)
80    continue
```

```
c       compute the gradient (analytical) wrt w5


        do 100 i=1,n
         w(i)=dummy(i)+hstep
         call fcn5(n,y1,y2,d,f,w)
         for=f
         w(i)=dummy(i)-hstep
         call fcn5(n,y1,y2,d,f,w)
         rev=f
         df(5,i)=(for-rev)/(2.d0*hstep)
         df(5,i)=hessold(5,i)+0.1d0*(df(5,i)-hessold(5,i))
         hessold(5,i)=df(5,i)
100     continue

c       compute the gradient (analytical) wrt w6


        do 110 i=1,n
         w(i)=dummy(i)+hstep
         call fcn6(n,y1,y2,d,f,w)
         for=f
         w(i)=dummy(i)-hstep
         call fcn6(n,y1,y2,d,f,w)
         rev=f
         df(6,i)=(for-rev)/(2.d0*hstep)
         df(6,i)=hessold(6,i)+0.1d0*(df(6,i)-hessold(6,i))
         hessold(6,i)=df(6,i)
110     continue

c       compute the gradient (analytical) wrt w7

        do 115 i=1,n
         w(i)=dummy(i)+hstep
         call fcn7(n,y1,y2,d,f,w)
         for=f
         w(i)=dummy(i)-hstep
         call fcn7(n,y1,y2,d,f,w)
         rev=f
         df(7,i)=(for-rev)/(2.d0*hstep)
```

```
        df(7,i)=hessold(7,i)+0.1d0*(df(7,i)-hessold(7,i))
        hessold(7,i)=df(7,i)
115     continue


        do 120 i=1,n
          w(i)=dummy(i)
120     continue

        return
        end
```