

CHAPTER 1

INTRODUCTION

Today the term artificial neural network has come to mean any computing architecture that consists of massively parallel interconnection of simple neural processors. The motivation for the development of neural networks comes mainly from the ability of biological systems to understand the behavior of unknown processes.

There are many commercial applications for neural networks. General areas of application involve signal processing, control, pattern recognition, medicine, speech recognition and production and business [1]. The level of development for these applications ranges from merely promising to full-scale commercial success.

Many signal processing applications require that N -dimensional input vectors be mapped or estimated into N_{out} -dimensional output vectors. Mapping or function approximation [2] and estimation are two processes which associate input vectors with output vectors. Processing of signals containing additive noise is done using estimation. The required mathematical function that generates the outputs from the inputs is usually unknown beforehand, in many applications, and training algorithms are used to approximate this function.

In the study presented in this thesis, a kind of Fourier series is implemented as the function that calculates the outputs corresponding to the neural network's inputs.

1.1 The N -dimensional Fourier Series

The Fourier series $\hat{s}(t)$ of a function $s(t)$ can be represented as a weighted sum of a set of basis functions [3], $e^{j2\pi n t/T}$, $n = 0, \pm 1, \pm 2, \dots$

$$\begin{aligned}\hat{s}(t) &= \sum_{n=-\infty}^{\infty} \alpha_n e^{jn\omega_0 t} & t_1 \leq t \leq t_1 + T \\ \alpha_n &= \frac{1}{T} \int_{t_1}^{t_1+T} s(t) e^{-jn\omega_0 t} dt\end{aligned}\tag{1.1}$$

where $\omega_0 = 2\pi/T$ and T is the period of $s(t)$. The coefficients α_n are chosen so as to minimize the mean square error between $s(t)$ and $\hat{s}(t)$.

A function must meet certain mathematical conditions in order for its Fourier series to converge [3]. These conditions are known as the *Dirichlet conditions* and state that a function has a Fourier representation if:

1. $\int |s(t)| dt$ over one period is finite.
2. There exist a finite number of maxima and minima in any finite time period.
3. There exist a finite number of discontinuities in any finite time period.

The above representation of the Fourier series is called the *exponential representation*. Another representation in terms of sines and cosines is given by:

$$\hat{s}(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos n\omega_0 t + b_n \sin n\omega_0 t$$

where

$$a_n = \frac{2}{T} \int_T s(t) \cos n\omega_0 t \, dt \quad n = 0, 1, 2, \dots\tag{1.2}$$

$$b_n = \frac{2}{T} \int_T s(t) \sin n\omega_0 t \, dt \quad n = 1, 2, \dots$$

and where a_n and b_n are real numbers if $s(t)$ is real. If two or more variables are involved, the N -dimensional Fourier series for a single output is introduced [3-6]:

$$\hat{s}(\mathbf{x}) = \frac{C_c(0)}{2} + \sum_{i=1}^{\infty} [C_s(i) \sin(v(i)) + C_c(i) \cos(v(i))] \quad (1.3)$$

$$v(i) = \sum_{k=1}^N \frac{2\pi m(k,i)}{t(k)} x(k)$$

where $x(k)$ is the k th element of the input vector and $t(k)$ is its period, $C_s(i)$ and $C_c(i)$ are the coefficients of the sine and cosine terms, respectively, for the i th harmonic, and $m(k,i)$ is an integer dependent on the harmonic and k th variable.

One problem with multi-dimensional Fourier series is *combinatorial explosion*. This is where the number of harmonics needed to approximate the signal with little error increases drastically as the number of input variables increases. A second problem is that $s(\mathbf{x})$ is unavailable making closed form expressions for $C_c(i)$ and $C_s(i)$ useless. These problems can be overcome by using a neural network approach.

1.2 The Neural Network Model

From a computational viewpoint neural networks offer many primary attractions such as learning, representation, and robust behavior with noisy data. In theory, neural networks exhibit fault tolerance when some neurons are damaged [7].

Many models have been proposed since the neural network evolution began in the 1940s. Rosenblatt introduced his popular model known as the *perceptron* [8]. A network of these units, known as the *multi-layer perceptron* [9,11], can generate nonlinear discriminant functions. Another model used to generate nonlinear terms from inputs is the *functional link network* [10]. Widrow [11] used ADALINES or *adaptive linear combiners* to illustrate his model. Regardless of the model used, the most common measure of the performance of estimation and mapping networks is the *mean square error (MSE)*. Another type of error, measured when dealing with classification networks, is the percentage of the presented input

patterns that the network classifies incorrectly. Mean square error calculations can also be applied to classification networks.

One drawback of neural networks is that many engineers don't understand them and therefore hesitate to use them in their signal processors. Another disadvantage is the very slow training that can result from zero or nearly zero gradients of the activation functions or the mean square error.

To solve the combinatorial explosion problem of multidimensional Fourier series and the unpopularity of neural networks, the *trigonometric* neural network was introduced by Dr. Li-Min Liu in his Ph.D. dissertation in 1995 [36]. This network solves the first problem by relaxing the harmonic relationship between the $m(k,i)$ terms of the Fourier series of equation 1.3. The second problem was eliminated by taking advantage of the fact that electrical and computer engineers are already familiar with the Fourier series and should have no problems dealing with this type of network.

Despite this, Dr. Liu's model suffers from two problems. The first problem is the absence of good training algorithms comparable to those available to the multi-layer perceptrons having sigmoidal activation functions.

The second problem is the absence of theoretical justification for certain aspects of the network.

1.3 Objective and Outline

In this thesis, a more advanced and efficient training algorithm is implemented for the trigonometric network, resulting in a very efficient signal processor that can compete with the MLP. The network is then analyzed for the orthogonality of its basis functions, the applicability of Parseval's equation to the network, and its similarity to networks using complex valued basis functions.

Another important issue that was investigated is finding a good structure of the proposed network. An efficient structure was reached by having networks with fewer useless units and merely contains units that would contribute positively to the performance of the network as a whole.

In chapter 2 of this thesis, an overview of the trigonometric network is presented along with two types of training algorithms. The type of network discussed in this chapter can be used for mapping or estimation.

In chapter 3, simulation results for trigonometric networks using mapping data files are presented and compared to the values obtained for a MLP with sigmoidal activation functions.

In chapter 4, a classification trigonometric network is introduced along with the algorithm used in the training of such networks followed by the presentation of the simulation results of the trigonometric network using classification training files in chapter 5.

The analysis of the trigonometric network in terms of Parseval's equation, orthogonality, and network structure along with simulation results is presented in chapter 6.

Chapter 7 presents a comparison between the proposed trigonometric network and some other possible networks like the complex and sine networks. Chapter 8 concludes this thesis with discussion for future research related to this work.

CHAPTER 2

THE TRIGONOMETRIC NETWORK

2.1 Introduction

The N-dimensional Fourier series, discussed in chapter 1, can be considered as a subset of the *trigonometric series* which takes a similar form. However, the basis functions of a trigonometric series are not necessarily orthogonal [6, 12].

Expanding the concept of the multi-dimensional Fourier series to neural networks, consider a network having N input units and N_{out} output units. For the p th pattern, the N -dimensional Fourier series $y_p(j)$ for the j th output unit can be expressed as [3-6, 36]:

$$y_p(j) = \sum_{i=1}^{N_F} [C_s(i, j) \sin(v_p(i)) + C_c(i, j) \cos(v_p(i))] \quad (2.1)$$

$$v_p(i) = \sum_{k=1}^N \frac{2\pi m(k, i)}{t(k)} x_p(k) \quad (2.2)$$

where N_F is the number of sine and cosine terms. N_F depends on the maximum harmonic number, H , for which the output $y_p(j)$ is approximated and is equal to $\lceil (2H+1)^N/2 \rceil$, where $\lceil \bullet \rceil$ denotes the next highest integer. $C_s(i, j)$ and $C_c(i, j)$ denote the coefficients of the sine and cosine terms of the Fourier series, respectively, for the i th harmonic and the j th output unit. $v_p(i)$ is an inner product of a scaled harmonic number and the inputs, $m(k, i)$ is an element of the coefficient matrix of harmonic number with value $|m(k, i)| \leq H$ and $m(k, 1) = 0$. The period $t(i)$ can be chosen such that $t(i) \geq \max(x(i)) - \min(x(i))$, where $\max(x(i))$ and $\min(x(i))$ denote the maximum and minimum values of the i th input feature, respectively.

The number of units in the hidden layer is found in the next section to be equal to

$2N_F$. This associates a problem to the Fourier series network. The problem is that the number of hidden nodes, which is a function of N_F , can be very large and can drastically increase as a function of the number of inputs or the number of harmonics needed. The following two examples illustrate this point. The first example deals with a network having 2 inputs ($N=2$), 1 output ($N_{\text{out}}=1$) and 3 harmonics ($H=3$). The number of net functions in the hidden layer is 25 and hence 50 output layer weights are needed. This number drastically increases as the number of inputs or the number of harmonics increase. To illustrate this point, we next consider the same structure of the previous example but with 3 inputs instead of two. Using this architecture, the number of hidden net functions is found to be 172 giving 344 output weights. The same argument can be applied if the number of harmonics is increased.

Another problem with the Fourier series network is that the generalization property is limited because of the fixed values of input weights feeding into the hidden layer. This comes directly from equation 2.1. Still two further problems with this kind of network are that the number of harmonics needed for the mapping or classification is unknown and that the optimal period of each of the inputs is unknown.

The above problems can be solved by introducing the trigonometric network. In this network the input weights in each hidden layer are allowed to adapt. This clearly eliminates the generalization problem of the Fourier series network. Doing this will also increase the pattern storage capability of the network. Implementing an adaptive set of input weights also eliminates the need for periods for the inputs. Another benefit is that the combinatorial explosion problem is eliminated. The resulting network is called the trigonometric network.

2.2 Structure of Trigonometric Networks

Trigonometric networks generally have three or more layers. Weights are connected in the forward direction. Input nodes are just connecting nodes with no transfer functions as

well as the output nodes which are generally linear nodes. The hidden nodes are nonlinear, with sines and cosines used for their activation functions, instead of the sigmoidal functions used in sigmoidal MLP networks.

2.2.1 Trigonometric Networks with One Hidden Layer

Trigonometric networks with a single hidden layer can be implemented as shown in figure 2.1. The net function for the m th-hidden node and the p th pattern can be given by:

$$net_p(2, m) = \theta(2, m) + \sum_{k=1}^N w(2,1, m, k)x_p(k), \quad (2.3)$$

where $1 \leq m \leq N_u(2)/2$, $N_u(2)$ being the number of sines and cosines in the hidden layer (second layer) and is equal to $2N_F$. $x_p(k)$ is the k th input for the p th pattern, $w(i,j,m,k)$ is the weight connecting the k th node of the j th source layer to the m th node of the i th destination layer. $\theta(2,m)$ is the threshold of the m th node of the hidden layer. In the above equation, the input weights are known in the case of a Fourier series network. In the case of the trigonometric network, the input weights are not predefined and are updated by a training algorithm.

The i th hidden node output is

$$o_p(2, i) = \sin(net_p(2, m)) \quad (2.4a)$$

if $i = 2m - 1$, and

$$o_p(2, i) = \cos(net_p(2, m)) \quad (2.4b)$$

if $i = 2m$, where $1 \leq m \leq N_u(2)/2$. It follows from equations 2.1 and 2.4 that the net function $net_p(2,m)$ in equation 2.3 is equivalent to the inner product $v_p(i)$ in equation 2.2.

Finally, the j th network output for the p th pattern, where $1 \leq j \leq N_{out}$, can be given by

$$y_p(j) = \theta(3, j) + \sum_{i=1}^{N_u(2)} w(3,2, j, i)o_p(2, i).$$

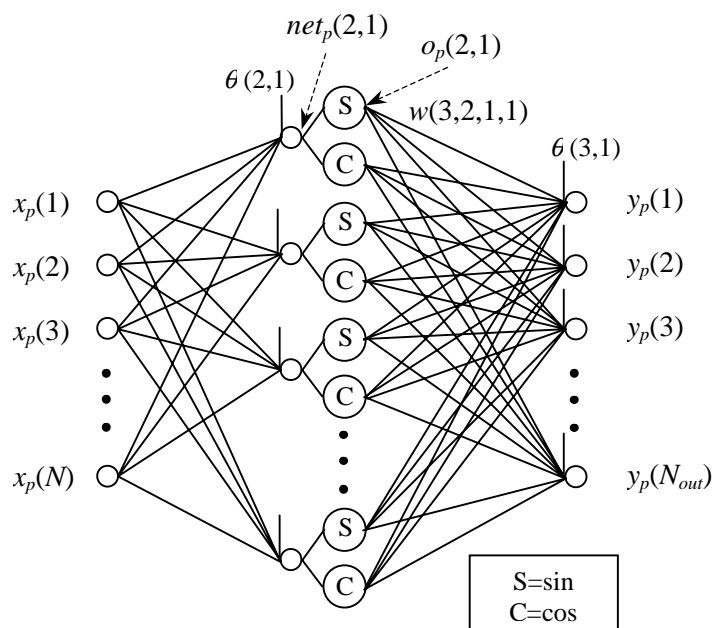


Figure 2.1. Trigonometric network with one hidden layer.

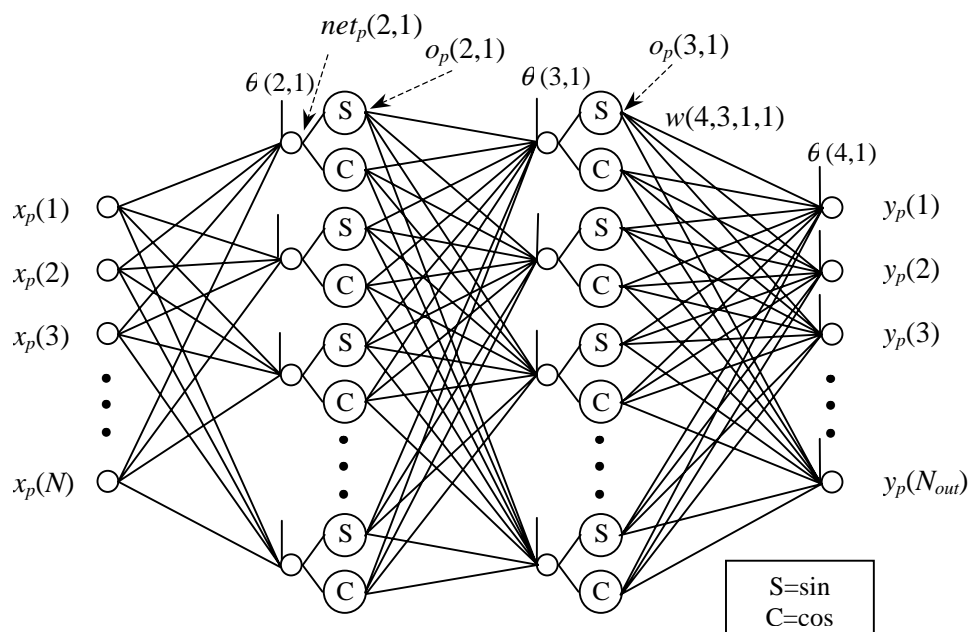


Figure 2.2. Trigonometric network with two hidden layers.

2.2.2 Trigonometric Networks with Two Hidden Layers

Adding another hidden layer can decrease the number of hidden nodes required in the previous structure. This is true because the added layer's activation functions generate higher harmonics of the input signal. Figure 2.2 shows the structure of a two hidden layer network.

In the first hidden layer, the m th net function and i th output is given in equations 2.3 and 2.4 as before.

Now the m th net function of the second hidden layer can be expressed as:

$$net_p(3, m) = \theta(3, m) + \sum_{k=1}^{N_u(2)} w(3, 2, m, k) o_p(2, k),$$

and the output of the i th unit is given by

$$o_p(3, i) = \sin(net_p(3, m))$$

if $i = 2m - 1$, and

$$o_p(3, i) = \cos(net_p(3, m))$$

if $i = 2m$, where $1 \leq m \leq N_u(3)/2$. Finally, the j th output of the network for the p th pattern is

$$y_p(j) = \theta(4, j) + \sum_{i=1}^3 \sum_{m=1}^{N_u(i)} w(4, i, j, m) o_p(i, m),$$

for $1 \leq j \leq N_{out}$.

Again, the input weights are just stated in the case of Fourier series networks, and allowed to adapt in the case of trigonometric networks.

2.3 Training Algorithms

Several training algorithms for multi-layered neural networks have been devised. A very popular method, namely *output weight optimization-backpropagation* (OWO-BP), has been used by many researchers [10,13-16,23]. Linear equations are solved to find the output weights and backpropagation [2,7] is used for calculating the hidden or input weights.

A new training algorithm, *output weight optimization-hidden weight optimization* (OWO-HWO), was developed by Dr. Chen in his dissertation. In this algorithm a good convergent procedure is used to modify the hidden weights.

2.3.1 The Error Function

One measure of the performance of the trigonometric network is the mean square error (MSE). Let $t_p(k)$ be the desired value of the k th output for the p th pattern, and $y_p(k)$ be the actual output for the k th output node. The mapping error [17] for the p th pattern can be represented by

$$E_p = \sum_{k=1}^{N_{out}} [t_p(k) - y_p(k)]^2,$$

where N_{out} is the number of nodes in the output layer.

To train a neural net in the *batch* mode we define the mapping error for the k th output as

$$E(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} [t_p(k) - y_p(k)]^2,$$

where N_v is the total number of patterns in the training data. From the above equations, we can calculate the total mean square error (MSE) as

$$E = \sum_{k=1}^{N_{out}} E(k) = \frac{1}{N_v} \sum_{p=1}^{N_v} E_p. \quad (2.5)$$

2.3.2 Output Weight Optimization- Backpropagation (OWO-BP)

2.3.2.1 Output Weight Optimization

Several researchers have investigated fast training techniques to find the weights of networks by solving a set of linear equations [10,18]. As mentioned earlier, the nodes in the output layer are in general linear and therefore, the output weights form a set of linear equations. The output weights can then be found by solving this set of linear equations.

The output weights should be updated in such a way that the total MSE is minimum. For the p th pattern, the k th output of the network can be given by

$$y_p(k) = \sum_{i=L_f(N_L)}^{N_L-1} \sum_{j=1}^{N_u(i)} w(N_L, i, k, j) o_p(i, j),$$

where N_L is the number of network layers, $N_u(i)$ is the number of units in the i th layer, and $L_f(i)$ denotes the least layer the i th layer is connected to. In order to handle hidden and output layer thresholds, we let $N_u(1)$ equal $N+1$, and let $o_p(1, N+1)=1$.

The MSE, from equation 2.5, is then minimized by first taking it's gradient with respect to the output weights,

$$\begin{aligned} g(N_L, i, m, j) &= \frac{\partial E}{\partial w(N_L, i, m, j)} \\ &= -\frac{2}{N_v} \sum_{p=1}^{N_v} [t_p(m) - y_p(m)] o_p(i, j), \end{aligned}$$

where, $o_p(i, j)$ is the output of the j th node in the i th layer which in this case is equal to N_L-1 .

To minimize the error, set the gradient to zero and rearrange to get

$$\sum_{k=L_f(N_L)}^{N_L-1} \sum_{l=1}^{N_u(k)+1} r(k, i, l, j) w(N_L, k, m, l) = s(m, i, j), \quad (2.6)$$

where

$$r(k, i, l, j) = r(i, k, j, l) = \sum_{p=1}^{N_v} o_p(k, l) o_p(i, j),$$

$$s(m, i, j) = \sum_{p=1}^{N_v} t_p(m) o_p(i, j).$$

Equation 2.6 can be written in the matrix form

$$\mathbf{R}_o \mathbf{W}_o = \mathbf{S}_o,$$

where \mathbf{R}_o is a square and symmetric auto-correlation matrix of size $N_w \times N_w$, \mathbf{W}_o and \mathbf{S}_o are the weight matrix and the cross-correlation matrix, respectively, of sizes $N_w \times N_{out}$, where N_w is the number of output weights and N_{out} is the number of outputs. This set of linear equations can be solved by several methods including Gaussian elimination, singular value decomposition (SVD) [19,25], and conjugate gradient [19-22].

2.3.2.2 Backpropagation

The output weights are computed by OWO as described in the previous section followed by the hidden weight computations using backpropagation. In this method the weights and thresholds are computed such that the error is decreased after each pattern is presented at the input. The following analysis are done for a three-layer trigonometric network, however, the same reasoning can be applied for networks having more than three layers. The hidden weights are updated as follows:

$$w(2,1, j, i)^{(It+1)} = w(2,1, j, i)^{(It)} - Z \frac{\bar{\delta} E_p}{\partial w(2,1, j, i)},$$

and the thresholds are updated as:

$$\theta(2, j)^{(It+1)} = \theta(2, j)^{(It)} - Z \frac{\bar{\delta} E_p}{\partial \theta(2, j)},$$

where It indicates the iteration number, and Z is the learning factor which usually satisfies $0 < Z < 1$. Using the chain rule, the gradient can be expanded as

$$\frac{\bar{\partial} E_p}{\partial w(2,1,j,i)} = \frac{\bar{\partial} E_p}{\partial net_p(2,j)} \frac{\bar{\partial} net_p(2,j)}{\partial w(2,1,j,i)}, \quad (2.7)$$

$$\frac{\partial E_p}{\partial \theta(2,j)} = \frac{\partial E_p}{\partial net_p(2,j)} \frac{\partial net_p(2,j)}{\partial \theta(2,j)}. \quad (2.8)$$

A *delta* function is now introduced to decrease the computational effort,

$$\delta_p(m,j) \equiv -\frac{\bar{\partial} E_p}{\partial net_p(m,j)} \quad m = 2,3, \quad (2.9)$$

we also have,

$$\begin{aligned} \frac{\bar{\partial} o_p(m,j)}{\partial net_p(m,j)} &= f'(net_p(m,j)), \\ \frac{\partial net_p(m,j)}{\partial w(m,n,j,i)} &= o_p(n,i), \\ \frac{\partial net_p(m,i)}{\partial o_p(n,j)} &= w(m,n,i,j). \end{aligned}$$

Equations 2.7 and 2.8 are then written as

$$\begin{aligned} \frac{\bar{\partial} E_p}{\partial w(2,1,j,i)} &= -\delta_p(2,j)x_p(i), \\ \frac{\partial E_p}{\partial \theta(2,j)} &= -\delta_p(2,j), \end{aligned}$$

where the delta functions are computed for the output and hidden nodes respectively as

$$\begin{aligned} \delta_p(3,k) &= -2(t_p(k) - y_p(k))f'(net_p(3,k)) & 1 \leq k \leq N_{out}, \\ \delta_p(2,j) &= f'(net_p(2,j)) \sum_{l=1}^{N_{out}} \delta_p(3,l)w(3,2,l,j) & 1 \leq j \leq N_u(2), \end{aligned} \quad (2.10)$$

where f' is the first derivative of the activation function.

The performance of classical backpropagation [2,7] can be affected by the order of the training patterns since the weights are changed for every pattern. Adopting batch mode backpropagation, in which the gradients are accumulated over all the training patterns before changing the weights, can reduce this effect. For full batching, the hidden weights and thresholds are calculated as

$$w(2,1, j, i)^{(It+1)} = w(2,1, j, i)^{(It)} - Z \frac{\bar{\delta} E}{\partial w(2,1, j, i)},$$

$$\theta(2, j)^{(It+1)} = \theta(2, j)^{(It)} - Z \frac{\bar{\delta} E}{\partial \theta(2, j)},$$

where

$$\frac{\partial E}{\partial w(2,1, j, i)} = \sum_{p=1}^{N_v} \frac{\partial E_p}{\partial w(2,1, j, i)},$$

$$\frac{\partial E}{\partial \theta(2, j)} = \sum_{p=1}^{N_v} \frac{\partial E_p}{\partial \theta(2, j)}.$$

In most cases, the output layer nodes are linear and therefore have $f'(net_p(3,j))=1$. In the case of the hidden nodes, the activation functions are sines and cosines and thus have $f'(net_p(2,j))$ equal to $\cos(net_p(2,j))$ and $-\sin(net_p(2,j))$, respectively.

In the case of a three-layer network, the delta functions of the output layer nodes are computed using equation 2.10. The delta functions of the hidden nodes are then computed as

$$\begin{aligned} \delta_p(2, j) = & \cos(net_p(2, j)) \sum_{l=1}^{N_{out}} \delta_p(3, l) w(3,2, l, 2j-1) \\ & - \sin(net_p(2, j)) \sum_{l=1}^{N_{out}} \delta_p(3, l) w(3,2, l, 2j), \end{aligned} \quad (2.11)$$

where $1 \leq j \leq N_u(2)/2$.

2.3.2.3 Calculating the Learning Factor

The proper value of the learning factor (Z) is difficult to determine. However, an automated algorithm can be developed in order to calculate this learning factor [37,46]. Assuming that the learning factor is small, we can approximate the change in error for a three-layer trigonometric network as

$$\Delta E = \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \left(\frac{\partial E}{\partial w(2,1,j,i)} \cdot \Delta w(2,1,j,i) \right) + \frac{\partial E}{\partial \theta(2,j)} \cdot \Delta \theta(2,j) \right],$$

$$\Delta E = -Z \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \left(\frac{\partial E}{\partial w(2,1,j,i)} \right)^2 + \left(\frac{\partial E}{\partial \theta(2,j)} \right)^2 \right].$$

Now assume that the error of E is to be reduced by a factor α which is close but less than 1,

$$\Delta E = \alpha E - E = -Z \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \left(\frac{\partial E}{\partial w(2,1,j,i)} \right)^2 + \left(\frac{\partial E}{\partial \theta(2,j)} \right)^2 \right],$$

therefore,

$$Z = \frac{Z'E}{\sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \left(\frac{\partial E}{\partial w(2,1,j,i)} \right)^2 + \left(\frac{\partial E}{\partial \theta(2,j)} \right)^2 \right]}, \quad (2.12)$$

where $Z' = 1 - \alpha$ and $0.0 < Z' < 0.1$. Hence the learning factor is automatically determined from the gradient and Z' .

2.3.3 Output Weight Optimization-Hidden Weight Optimization (OWO-HWO)

The previous section introduced the backpropagation method as an algorithm for calculating hidden weights. Unfortunately, this algorithm is not very efficient [23]. A more efficient updating algorithm is introduced in this section where the hidden weights are

calculated by solving a set of linear equations [24]. This would require that the desired hidden net functions be given, which is not normally the case. However, approximations of these desired net functions can be made based on the current net function values plus a designed net change. Therefore, for the p th pattern, the j th unit of the hidden layer can have its desired net function approximated by [24,37,46]:

$$net_{pd}(2, j) \cong net_p(2, j) + Z \cdot \delta_p(2, j)$$

where $1 \leq j \leq N_u(2)/2$, $net_{pd}(2, j)$ is the desired net function, $net_p(2, j)$ is the actual net function, Z is a learning factor, and $\delta_p(2, j)$ is the delta function from equation 2.11.

The hidden weights can therefore be updated according to the following equation:

$$w(2,1, j, i)^{(It+1)} = w(2,1, j, i)^{(It)} + Z \cdot e(2,1, j, i) \quad (2.13)$$

for $1 \leq j \leq N_u(2)/2$, $1 \leq i \leq N$, and where $e(2,1, j, i)$ serves the same purpose as the negative gradient element, $-\partial E / \partial w(2,1, j, i)$, in backpropagation. The hidden thresholds can be dealt with as weights having unit input.

Recalling that

$$net_p(2, j) = \sum_{i=1}^{N+1} w(2,1, j, i) o_p(1, i),$$

where $o_p(1, N+1) = 1$, $o_p(1, i) = x_p(i)$ for $1 \leq i \leq N$, and $w(2,1, j, N+1) = \theta(2, j)$, we can use the following equation to solve for the changes in the hidden weights,

$$net_p(2, j) + Z \cdot \delta_p(2, j) \cong \sum_{i=1}^N [w(2,1, j, i) + Z \cdot e(2,1, j, i)] \cdot x_p(i).$$

By manipulating the above equation we can obtain a relation between $\delta(\bullet)$ and $e(\bullet)$:

$$\delta_p(2, j) \cong \sum_{i=1}^N e(2,1, j, i) \cdot x_p(i).$$

We now define an objective function for the j th unit in the hidden layer as:

$$E_{\delta}(2, j) = \sum_{p=1}^{N_v} \left[\delta_p(2, j) - \sum_{i=1}^N e(2,1, j, i) \cdot x_p(i) \right]^2.$$

By taking the gradient of $E_{\delta}(2, j)$ with respect to weight changes, we get:

$$g(1, l) = \frac{\partial E_{\delta}(2, j)}{\partial e(2,1, j, l)} = -2 \left(s(1, l) - \sum_{i=1}^N e(2,1, j, i) \cdot r(i, l) \right)$$

where

$$r(i, l) = \sum_{p=1}^{N_v} x_p(i) \cdot x_p(l)$$

$$s(1, l) = \sum_{p=1}^{N_v} \delta_p(2, j) \cdot x_p(l) = -\frac{\partial E}{\partial w(2,1, j, l)}.$$

To minimize $E_{\delta}(2, j)$, set $g(1, l)$ to zero to get the set of linear equations:

$$\sum_{i=1}^N e(2,1, j, i) \cdot r(i, l) = -\frac{\partial E}{\partial w(2,1, j, l)}.$$

By solving this set of linear equations for $e(2,1, j, i)$, we can update the hidden weights as in equation 2.13.

2.3.3.1 Calculating the learning factor

As mentioned earlier, it is hard to determine a correct value for the learning factor. Therefore, an automated procedure [46] for determining the learning factor is proposed.

Consider the change in the error:

$$\Delta E \cong \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \frac{\partial E}{\partial w(2,1, j, i)} \cdot \Delta w(2,1, j, i) \right] = Z \cdot \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \frac{\partial E}{\partial w(2,1, j, i)} \cdot e(2,1, j, i) \right].$$

Now suppose that we want the error to be reduced by a factor α which is less than but close to 1,

$$\Delta E = \alpha E - E \cong Z \sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \frac{\partial E}{\partial w(2,1,j,i)} \cdot e(2,1,j,i) \right],$$

the learning factor can be calculated as

$$Z = \frac{-Z'E}{\sum_{j=1}^{N_u(2)/2} \left[\sum_{i=1}^N \frac{\partial E}{\partial w(2,1,j,i)} \cdot e(2,1,j,i) \right]}$$

$$Z' = 1 - \alpha,$$

where Z' is a number such that $0 < Z' < 0.1$.

2.4 Thresholds in Trigonometric Networks

A very exciting feature of trigonometric networks is that the thresholds of the hidden layer can be safely removed in the testing phase of the network without ever affecting its performance. This can be done by linearly transforming the output weights to anticipate for the removed thresholds.

Consider the subnet shown in figure 2.3. In this configuration, three inputs feeding a single hidden unit having a sine and a cosine with θ being the threshold and a and b being weights of the output layer. The net function of the given configuration can be written as:

$$net_A = \theta + \sum_{i=1}^3 w_i x_i$$

The same network with the threshold removed and the output weights taking new values c and d is shown in figure 2.4. The net function of this portion of the network can be written as:

$$net_B = \sum_{i=1}^3 w_i x_i$$

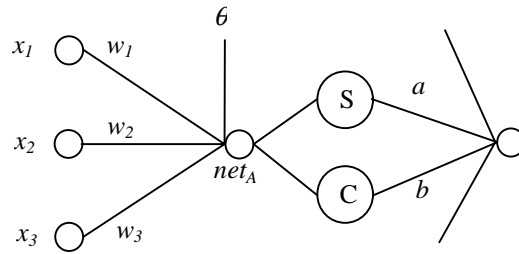


Figure 2.3. Subnet with thresholds.

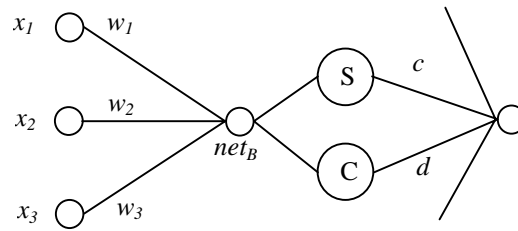


Figure 2.4. Subnet without thresholds.

For the two networks to be equivalent, we should have the following:

$$\begin{aligned} & a \sin(\text{net}_A) + b \cos(\text{net}_A) \\ &= c \sin(\text{net}_B) + d \cos(\text{net}_B). \end{aligned}$$

Where

$$\sin(\text{net}_A) = \sin(\theta) \cos(\text{net}_B) + \cos(\theta) \sin(\text{net}_B),$$

and

$$\cos(\text{net}_A) = \cos(\theta) \cos(\text{net}_B) - \sin(\theta) \sin(\text{net}_B).$$

For the above equations to be true we have

$$\begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}. \quad (2.14)$$

From equation 2.14 it is clear that we can write the output weights of the threshold-less network as a linear transformation of the output weights for a network having thresholds and yet having the same output.

Unfortunately, this is not the case when training the network. Thresholds should be present and updated by the training algorithm during the training phase of the network as the following discussion shows.

Consider a trigonometric network being trained using the OWO-BP training algorithm. Removing the thresholds from the network would remove its effect from the denominator of equation 2.12. This would change the value of the learning factor and would probably increase the value of the learning factor. The removal of the threshold also plays a role in changing the value of the weight gradient in the equation

$$\frac{\partial E_p}{\partial w(2,1,j,i)} = -\delta_p(2,j)x_p(i).$$

This is because $\delta_p(2,j)$ is a function of $\delta_p(3,j)$ which depends on the hidden layer thresholds. The change to this gradient would change the change the values of $\Delta w(2,1,j,i)$ and there will be no guarantee that this would lead to a faster converging network.

Besides the discussion in the previous paragraph, the removal of the thresholds would no longer give the network the ability to adjust the means of the outputs of the hidden layer. This by itself can lead to instability and reduction in the performance.

In chapter 3, a fair comparison between the performance of a trigonometric network and a MLP having sigmoidal activation functions is presented using different mapping data files.

CHAPTER 3

SIMULATION OF MAPPING NETWORKS

In chapter 2, the structure and training algorithms for a multi-layer trigonometric network are presented. In this chapter the performance of the proposed network is compared to that of the sigmoidal multi-layer perceptron network

A fair comparison between the two types of networks restricts the choice of the number of hidden units in each network. For the comparison to be fair, the number of hidden units is chosen such that the two networks have the same pattern storage capability. Pattern storage can be defined as the number of patterns that a neural network can learn perfectly such that its outputs exactly match the desired output values.

3.1 Pattern Storage Bounds

The upper bound on pattern storage can be derived from that of the functional link network (FLN). The FLN network is capable of storing patterns with 100% efficiency. The pattern storage of a FLN is the number of weights and thresholds per output [33,35] which is written as

$$M_{FLN} = \frac{N'_w}{N_{out}}, \quad (3.1)$$

where N'_w is the total number of weights and thresholds in the network and N_{out} is the number of outputs. This storage capability of the FLN can be regarded as the upper bound on the pattern storage for the trigonometric network as well as the MLP.

From the above discussion, we can give expressions for the upper bound for the trigonometric network. Using a fully connected feedforward network,

$$N'_w = N \cdot N_{out} + \frac{1}{2} N \cdot N_u(2) + N_u(2) \cdot N_{out} + \frac{1}{2} N_u(2) + N_{out}$$

if $N_L = 3$, and

$$\begin{aligned} N_w = N \cdot N_{out} + \frac{1}{2} N \cdot N_u(2) + \frac{1}{2} N \cdot N_u(3) + \frac{1}{2} N_u(2) \cdot N_u(3) \\ + N_u(2) \cdot N_{out} + N_u(3) \cdot N_{out} + \frac{1}{2} N_u(2) + \frac{1}{2} N_u(3) + N_{out} \end{aligned}$$

if $N_L = 4$, where N_L is the number of layers, N , N_{out} and $N_u(i)$ are the number of units in input, output and i th layer, respectively. The upper bound can then be calculated from equation 3.1.

For the MLP network, the number of weights and thresholds can be given by:

$$N'_w = N \cdot N_{out} + N \cdot N_u(2) + N_u(2) \cdot N_{out} + N_u(2) + N_{out}$$

for $N_L = 3$, and

$$\begin{aligned} N'_w = N \cdot N_{out} + N \cdot N_u(2) + N \cdot N_u(3) + N_u(2) \cdot N_u(3) \\ + N_u(2) \cdot N_{out} + N_u(3) \cdot N_{out} + N_u(2) + N_u(3) + N_{out} \end{aligned}$$

for $N_L = 4$.

Given the number of hidden units for one of the networks, the number of hidden units for the other network can be found by equating the upper bounds on pattern storage for the two networks. For the three-layer network, the number of MLP hidden units can be given by

$$N_{u,MLP}(2) = \frac{N_{u,Trig}(2) \cdot (2N_{out} + N + 1)}{2 \cdot (N_{out} + N + 1)}, \quad (3.2)$$

where $N_{u,MLP}(2)$ and $N_{u,Trig}(2)$ are the number of hidden units in the MLP and the trigonometric networks, respectively.

3.2 Simulation Results

3.2.1 OWO-BP and OWO-HWO

Training Results

In this subsection, simulation results for sigmoidal MLPs and trigonometric networks using several mapping data files are presented. Table 3.1 shows a summary of the architectures of the networks and data files used in the training followed by simulation graphs of the training MSE for the two networks using two different architectures. All the following results are based on equal pattern storage of both the sigmoidal MLP and the trigonometric network.

Table 3.1. Network Topologies used in OWO-BP/OWO-HWO training

<i>Training file</i>	<i>Sigmoidal MLP Topology</i>	<i>Trigonometric network topology</i>	<i>Used in</i>
TWOD.TRA	8-12-7	8-16-7	Figure 3.1
TWOD.TRA	8-16-7	8-22-7	Figure 3.2
SINGLE2.TRA	16-25-3	16-44-3	Figure 3.3
SINGLE2.TRA	16-32-3	16-56-3	Figure 3.4
OH7.TRN	20-25-3	20-46-3	Figure 3.5
OH7.TRN	20-32-3	20-58-3	Figure 3.6

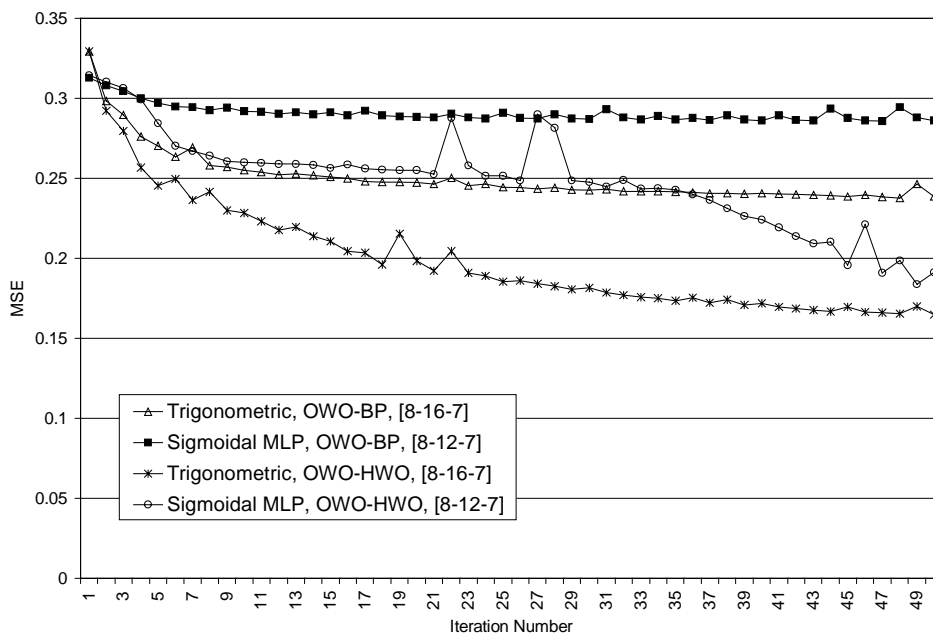


Figure 3.1. Training using TWOD.TRA mapping data file.

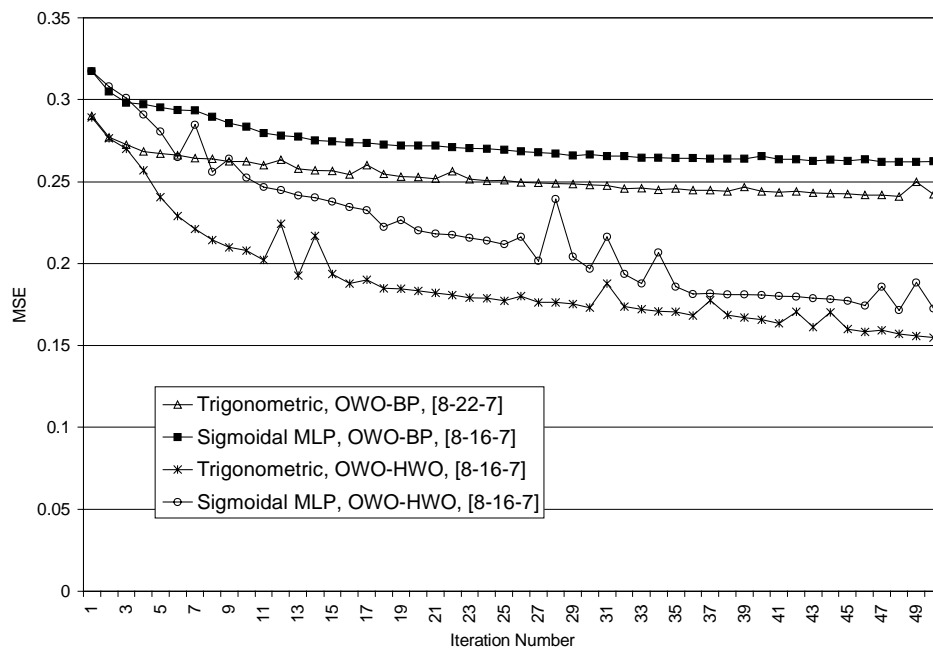


Figure 3.2. Training using TWOD.TRA mapping data file.

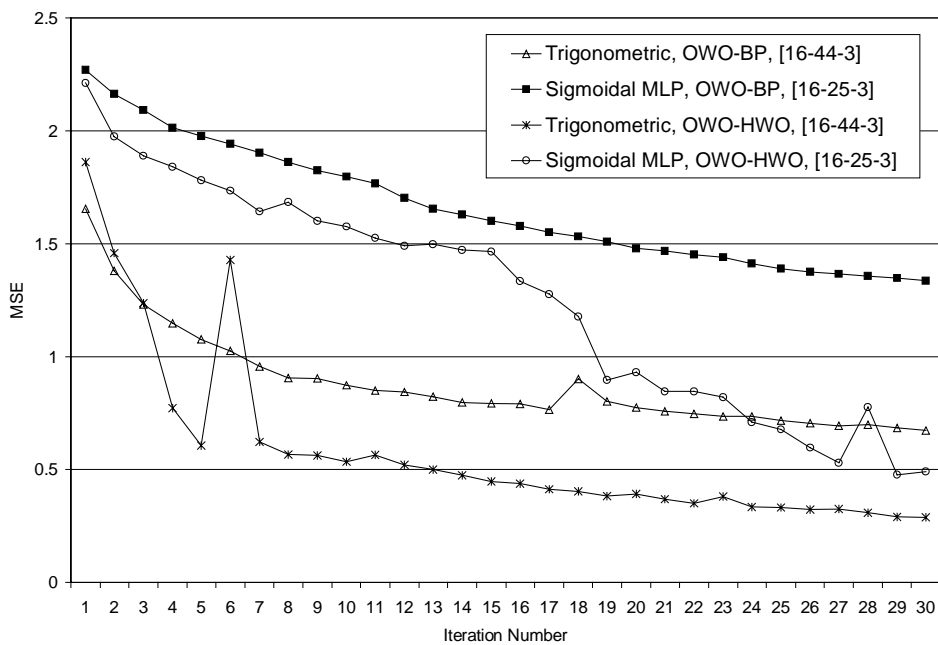


Figure 3.3. Training using SINGLE2.TRA mapping data file.

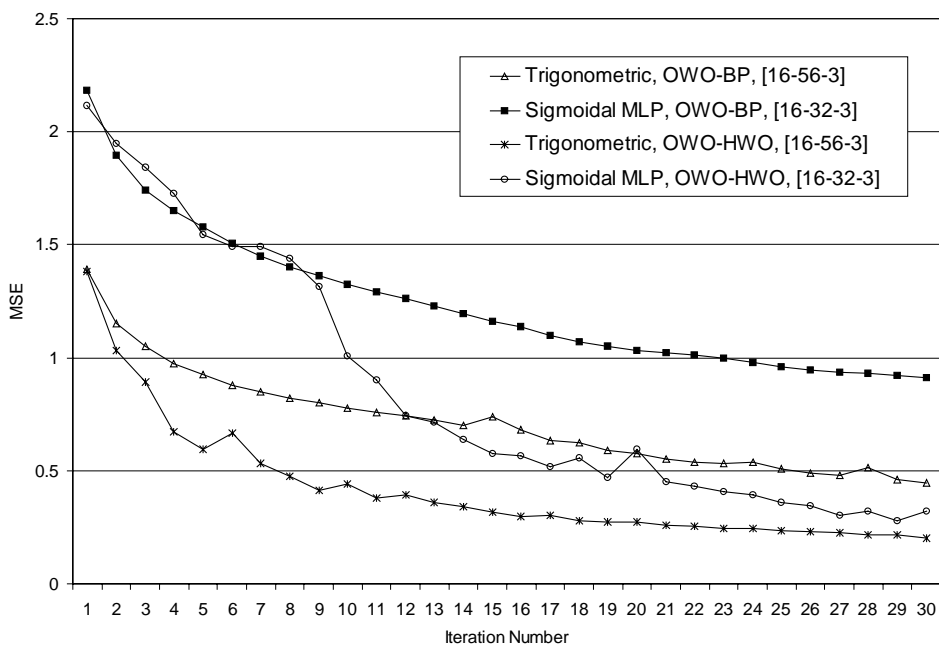


Figure 3.4. Training using SINGLE2.TRA mapping data file.

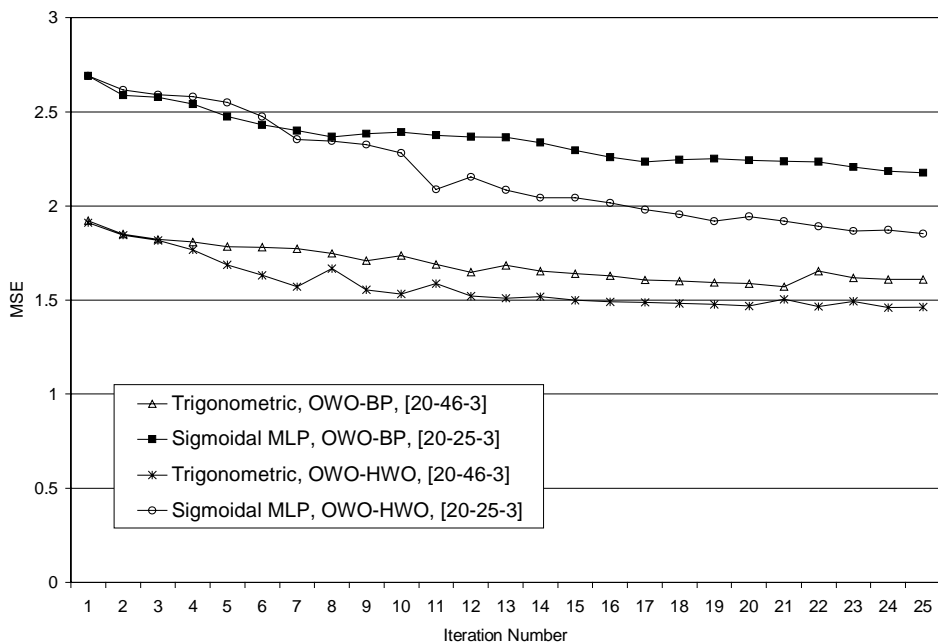


Figure 3.5. Training using OH7.TRN mapping data file.

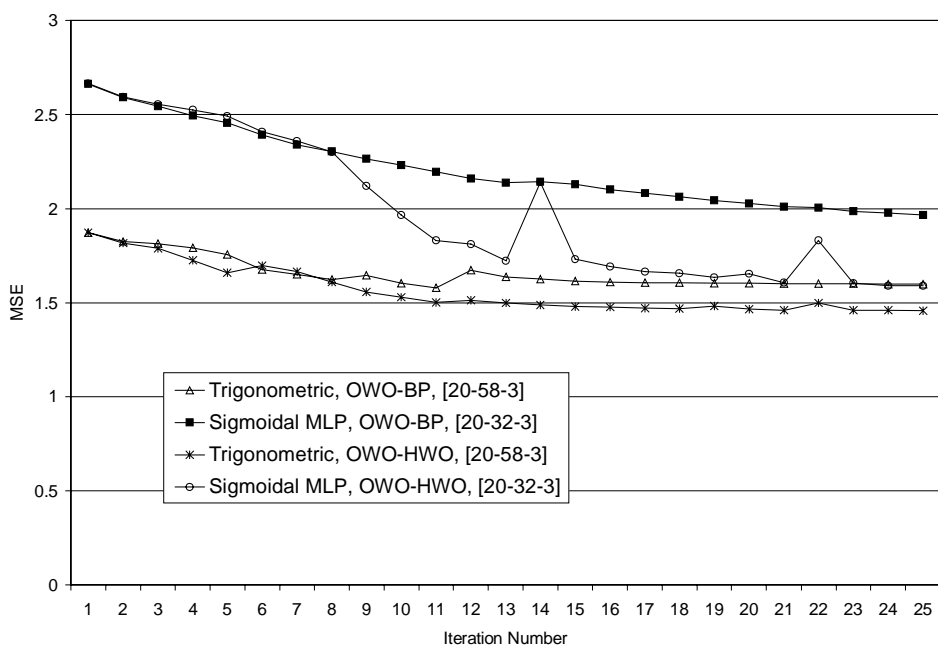


Figure 3.6. Training using OH7.TRN mapping data file.

3.2.2 OWO-HWO, Trigonometric-MLP Comparison

As promised the simulation results show a big difference in the performance of both training algorithms. Trigonometric networks, also as promised, are superior to the sigmoidal MLP networks and give a better performance.

Trigonometric MSE values converge faster than its MLP counterpart. However, both networks converge to approximately same MSE values when the number of iterations is very large.

One reason for the faster convergence of the trigonometric network is that the change of the hidden weights depends on the delta function defined earlier in this thesis. This delta function in turn depends on the derivative of the activation function. In the case of the MLP this derivative can be close to zero in the saturated part of the nonlinearity. This leads to an infinitesimal change in the hidden weights and correspondingly to the MSE. In the trigonometric network, however, the activation functions are sines and cosines and the derivatives of both cannot be zero at the same time. This will contribute to a larger change in the hidden weights and the MSE.

The simulations also show a considerably faster convergence for the OWO-HWO when compared to the OWO-BP training algorithm. A reason for this OWO-HWO superiority is that a set of linear equations involving approximate desired hidden net values is used. These desired net values can contribute to better values of the gradients and hence improve the training algorithm.

CHAPTER 4

CLASSIFICATION TRIGONOMETRIC NETWORKS

4.1 Introduction

Classification is the assignment of each input vector to a specific class which may be one of many predetermined groups. A training set representative to all classes, along with class membership information for each pattern, is presented to the input layer of the network. Using this data set, the network deduces rules for membership in each class which can then be used to assign other patterns the correct class according to these rules.

In classification, we map the input vectors into a vector of discriminant functions. The class is then found by finding the maximum (in the positive or in the negative side depending on the designers assignments) of the discriminant functions. Different classifiers are proposed, the optimal Bayesian classifier tends to determine the most likely class to the given pattern [26]. The nearest neighbor classifier associates a given pattern to a class according to a minimum distance criterion [27,28].

4.2 The Classification Error

In classification networks a different type of training error is looked into. The MSE used in mapping networks may not be a very good indication of the performance of the network as a classifier. Classification error is the percentage of patterns that the network classifies incorrectly. During training the value of MSE can be decreasing while there is no improvement in the classification error. Adding the same constant to each of the outputs changes the MSE but has absolutely no effect on the classification error. Another observation

is that when an output has the same sign as the desired output associated with it but with a larger magnitude, increasing the magnitude of the desired output will decrease the MSE but will have no effect on the classification error.

4.3 Output Reset Algorithm

The output reset algorithm [36] is used as a training algorithm for classification networks to overcome the MSE related problems in the network performance.

Consider the MSE defined in equation 2.5 and repeated here for convenience

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{k=1}^{N_c} [t_p(k) - y_p(k)]^2, \quad (4.1)$$

where N_c is the number of classes which is the same as the number of units in the output layer, and $t_p(k)$ and $y_p(k)$ being the desired and the actual k th output unit for the p th pattern. Modify E in equation 4.1 as:

$$E' = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{k=1}^{N_c} [t'_p(k) - y_p(k)]^2, \quad (4.2)$$

where $t'_p(k)$ can be written as $t_p(k) + a_p + d_p(k)$. Here a_p and $d_p(k)$ are functions to be defined later. The aim of the algorithm is to find the values of a_p and $d_p(k)$ that minimize E' in equation (4.2) and satisfy the following conditions:

1. The difference $|t'_p(k_c) - t'_p(k_d)|$ must be larger than or equal to $|t_p(k_c) - t_p(k_d)|$, where k_c and k_d are the output nodes equal and not equal to the target class, respectively. $t_p(k) = -b$ if $k = k_c$ and $t_p(k) = +b$ if $k = k_d$ where b is the initial value assigned to the desired output. This condition ensures that E' can be minimized without the network weights and the difference $|t'_p(k_c) - t'_p(k_d)|$ be equal to zero.

2. During training, changes made to a_p , $d_p(k)$ and $y_p(k)$ must be able to reduce E' . Improper setting of a_p , $d_p(k)$ or $y_p(k)$ can increase the error function.

Thus far the conditions governing a_p , $d_p(k)$ and $y_p(k)$ were presented. Next, the required update changes of these values are presented.

Changes to a_p

Setting the first derivative of E' with respect to a_p to zero, we find that

$$a_p = \frac{1}{N_c} \sum_{k=1}^{N_c} [y_p(k) - t_p(k) - d_p(k)]. \quad (4.3)$$

If we are dealing with a_p alone, $d_p(k)$ can be set to zero.

Adding a_p to each desired output, for the p th pattern, will maintain the distances between the desired outputs and hence satisfying condition 1. Condition 2 is automatically satisfied by the attempt of minimizing E' with respect to a_p in equation (4.3).

Changes to $d_p(k)$

Values of $d_p(k)$ offer different bias values to different output units' desired values. Setting the term $[t_p(k) + a_p + d_p(k) - y_p(k)]^2$ to zero will satisfy condition 2 yielding $d_p(k) = y_p(k) - t_p(k) - a_p$. Condition 1 is then satisfied by modifying values of $d_p(k)$ such that $d_p(k_c) \geq 0$ and $d_p(k_d) \leq 0$. Doing this we have:

1. If $y_p(k_c) \geq t_p(k_c) + a_p$, then choose $d_p(k_c) = y_p(k_c) - t_p(k_c) - a_p$.
2. If $y_p(k_d) \leq t_p(k_d) + a_p$, then choose $d_p(k_d) = y_p(k_d) - t_p(k_d) - a_p$.
3. Otherwise, choose $d_p(k_c) = 0$ or $d_p(k_d) = 0$.

If we are dealing with $d_p(k)$ alone, a_p can be set to zero.

Changes to $y_p(k)$

After finding $d_p(k)$ and a_p , E' can be minimized with respect to the output weights using output weight optimization (OWO) as in section 2.3.2.1.

4.4 Implementing the Output Reset Algorithm

First, initialize a_p as in equation (4.3) and $d_p(k)$'s to zero. Perform the following steps in training a classification network:

1. Assign $t_p(k)$ values to each input vector \mathbf{x}_p as $t_p(k_c) = -p$ and $t_p(k_d) = +p$, where p is any real number, 0.5 was used in the simulations in the next chapter.
2. For each input vector \mathbf{x}_p do the following:
 - (a) Initialize a_p and $d_p(k)$.
 - (b) Calculate the outputs $y_p(k)$ using the current weights.
 - (c) Iterate through values of a_p and $d_p(k)$ computing $t'_p(k)$ until a satisfactory E' value is reached.
 - (d) Calculate auto-correlations and cross-correlations from input \mathbf{x}_p and $t'_p(k)$.
3. Update output and hidden weights by OWO-HWO.
4. Go to step 2 for another iteration, if necessary.

In chapter 5, simulation results are presented as a comparison of training a trigonometric classification network and a MLP classification network having sigmoidal activation functions.

CHAPTER 5

SIMULATION OF CLASSIFICATION NETWORKS

In this chapter the output reset algorithm described in chapter 4 is implemented and the networks simulated. The idea used in the mapping networks in choosing the number of units in the hidden layer in both competing networks units is also applied here. In this case however, there are two ways to measure and compare the performances of the two networks. The classification error is measured for the two networks as well as the MSE which, as discussed previously, is not related to the classification error. Three classification data files were used in the simulations. These data files are described in the appendix at the end of this thesis.

Table 5.1 shows the topologies used in the simulations for the different data files and figures 5.1 through 5.12 show the simulation results.

Table 5.1 Network Topologies used in classification network training

<i>Training file</i>	<i>Number of classes</i>	<i>Sigmoidal MLP Topology</i>	<i>Trigonometric network topology</i>	<i>Classification error used in</i>	<i>MSE used in</i>
GRNG	4	16-24-4	16-40-4	Figure 5.1	Figure 5.7
GRNG	4	16-30-4	16-50-4	Figure 5.2	Figure 5.8
GONGTRN	10	16-20-10	16-30-10	Figure 5.3	Figure 5.9
GONGTRN	10	16-28-10	16-40-10	Figure 5.4	Figure 5.10
COMF18	4	18-24-4	18-40-4	Figure 5.5	Figure 5.11
COMF18	4	18-30-4	18-50-4	Figure 5.6	Figure 5.12

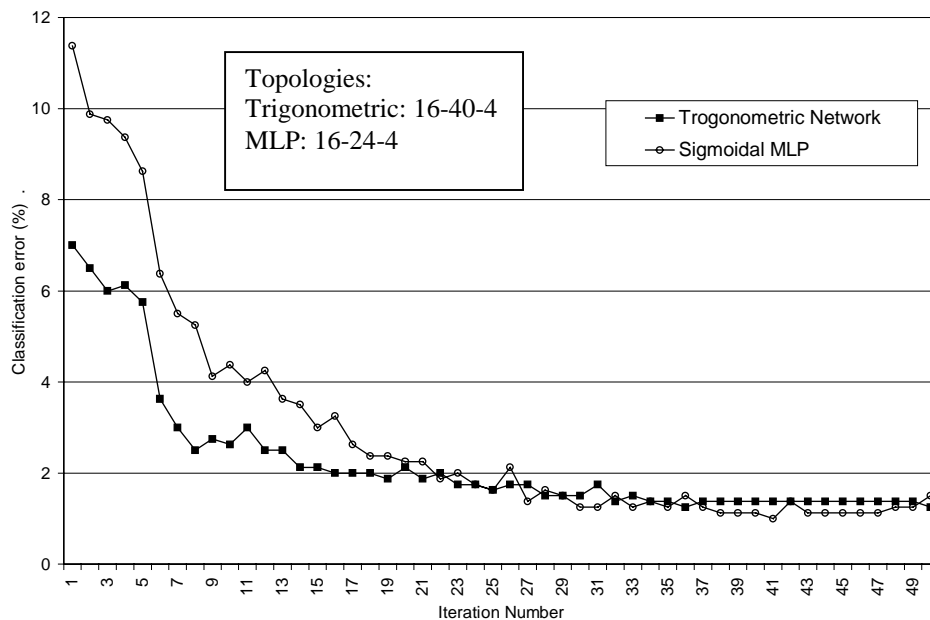


Figure 5.1. Classification error for data file GRNG.

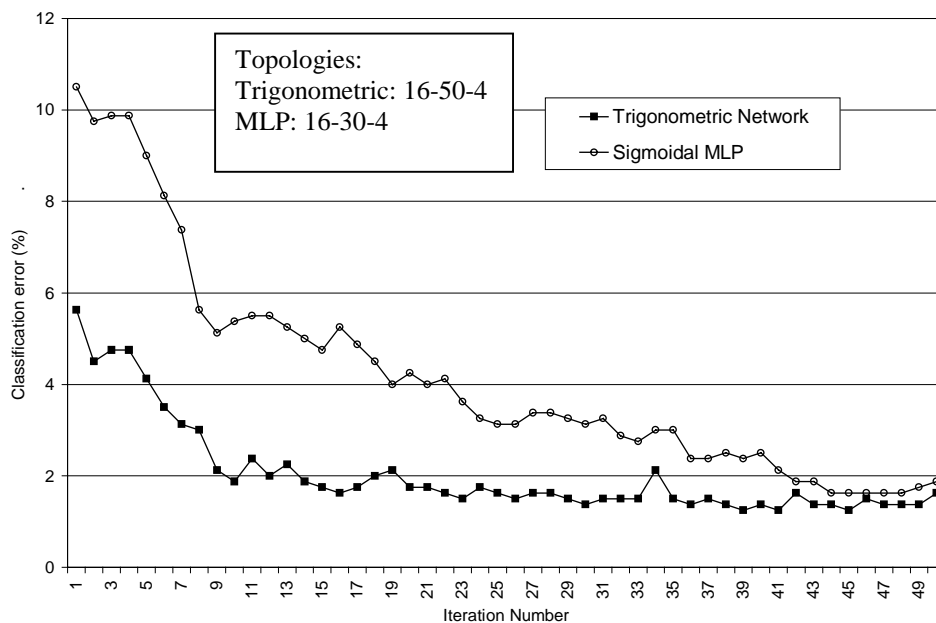


Figure 5.2. Classification error for data file GRNG.

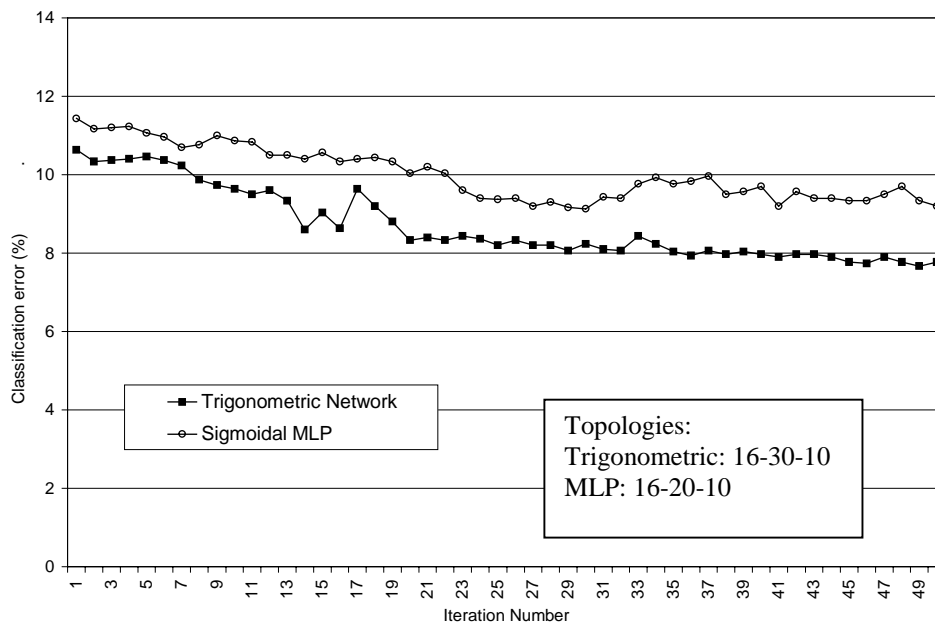


Figure 5.3. Classification error for data file GONGTRN.

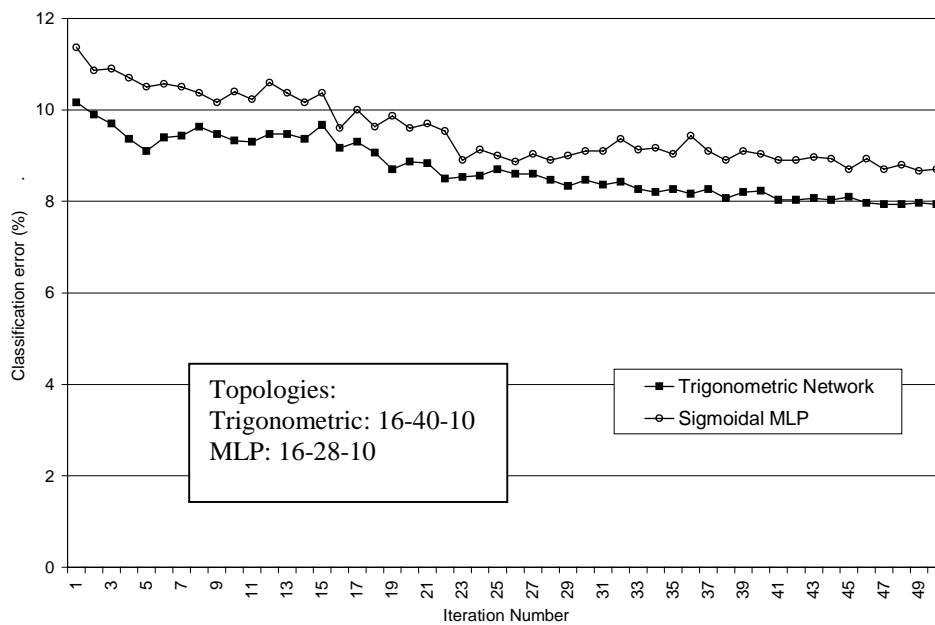


Figure 5.4. Classification error for data file GONGTRN.

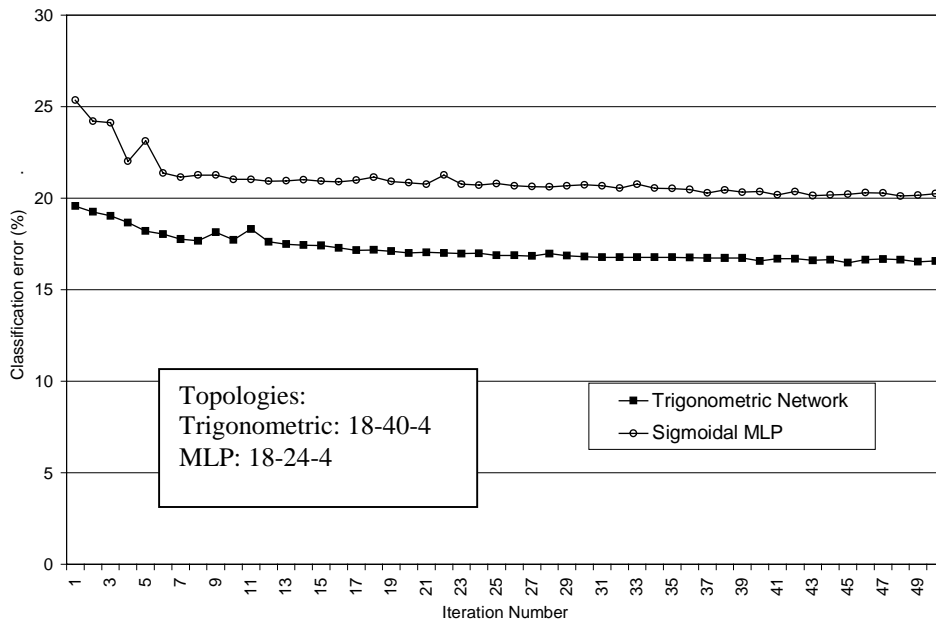


Figure 5.5. Classification error for data file COMF18.

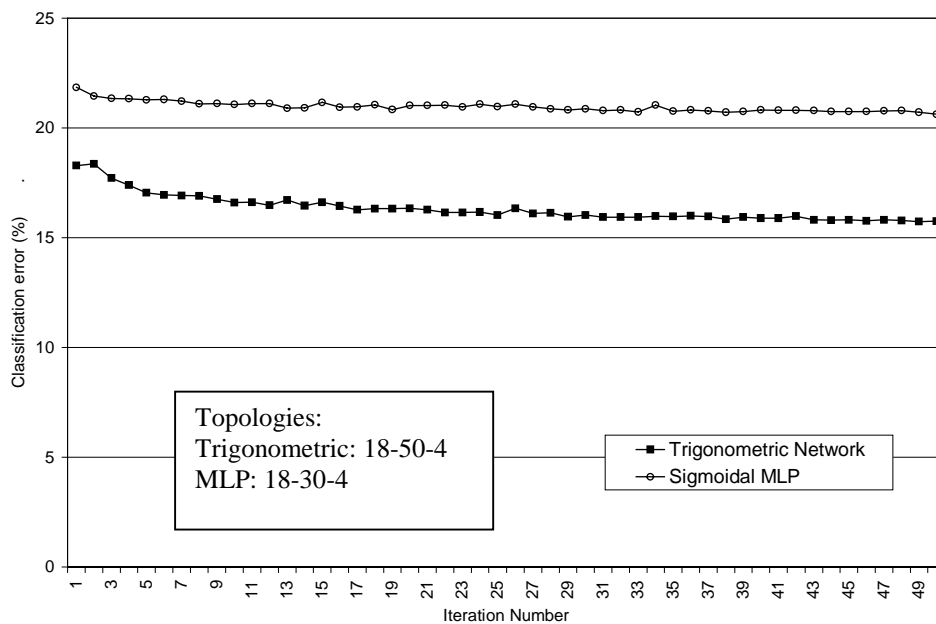


Figure 5.6. Classification error for data file COMF18.

As can be seen from the above figures, the trigonometric network converges faster than the sigmoidal MLP network. OWO-HWO was used to train the hidden weights as in step 3 in section 4.4. This proves that trigonometric network can also be used as a classifier with good performance.

The remaining figures, 5.7 through 5.12, shows the MSE for the classification networks for the trigonometric and the sigmoidal MLP networks.

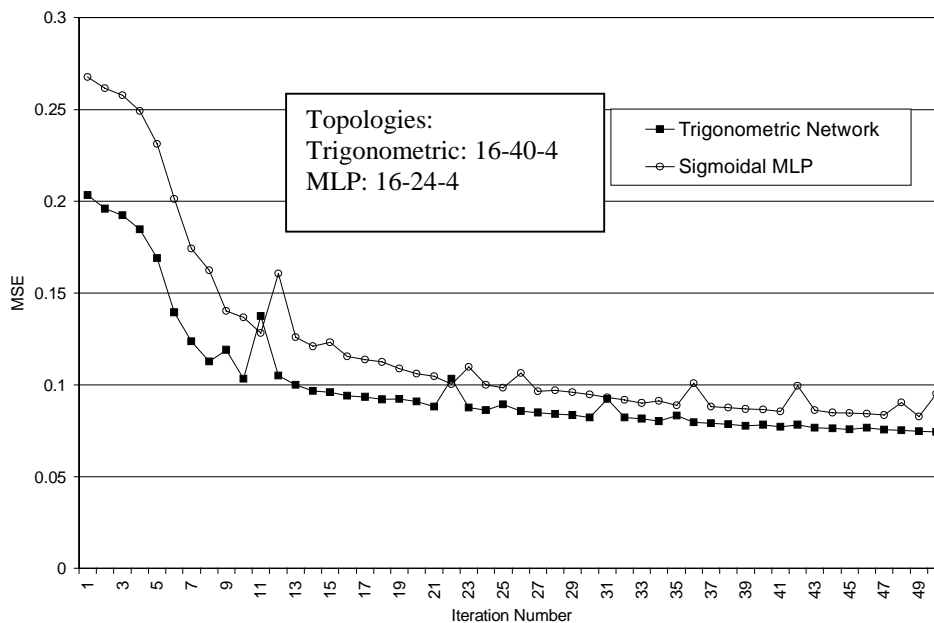


Figure 5.7. MSE error for data file GRNG.

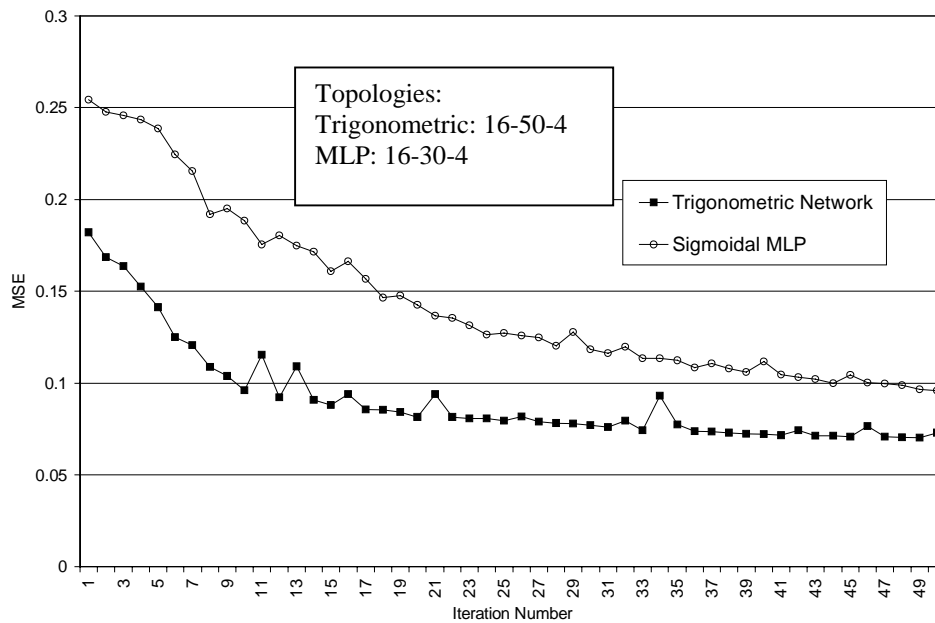


Figure 5.8. MSE error for data file GRNG.

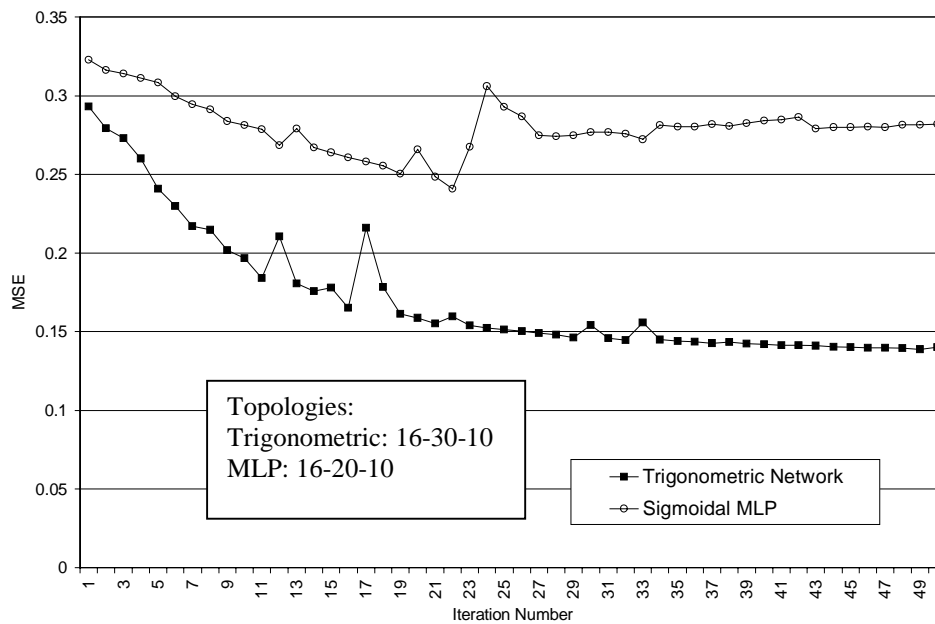


Figure 5.9. MSE error for data file GONGTRN.

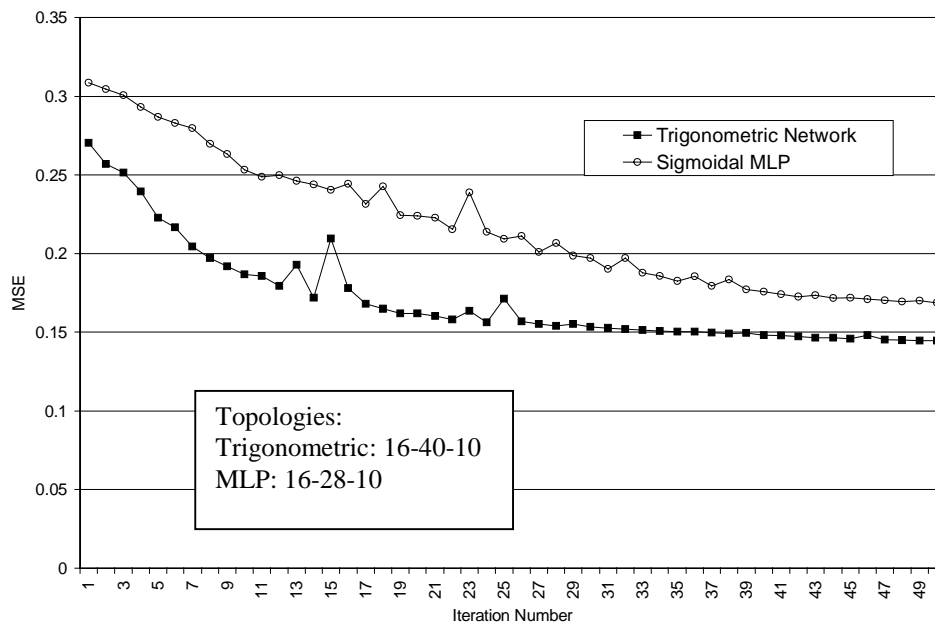


Figure 5.10. MSE error for data file GONGTRN.

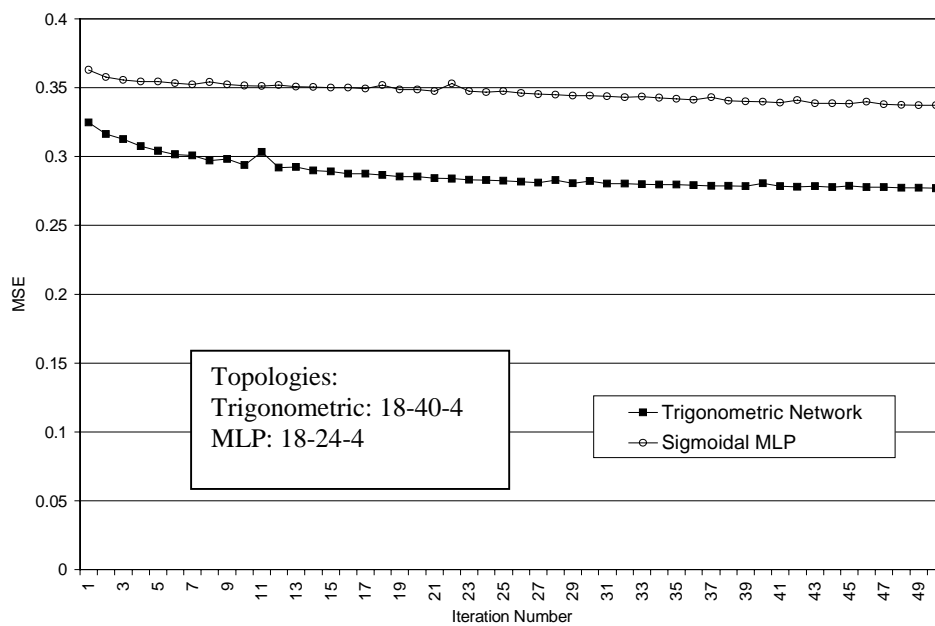


Figure 5.11. MSE error for data file COMF18.

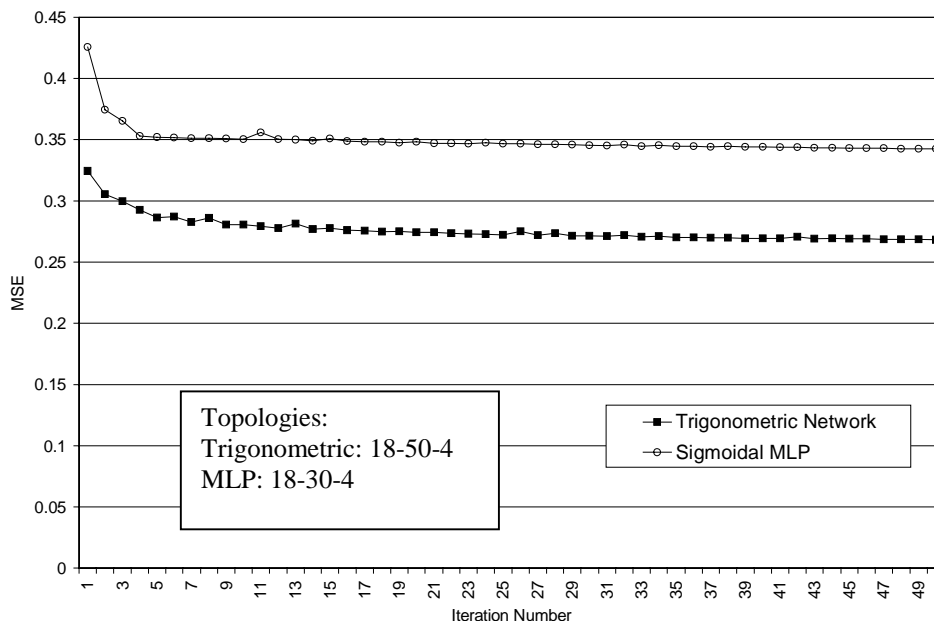


Figure 5.12. MSE error for data file COMF18.

As in the classification error, the MSE values of the trigonometric network also have better values than those of the sigmoidal MLP. Another interesting behavior of the classification error and the MSE, is that the MSE converges more uniformly than the classification error.

In chapter 6, the properties of the basis functions of a trigonometric network are presented along with an algorithm for finding efficient architectures of the network by removing the useful hidden units.

CHAPTER 6

ANALYSIS OF TRIGONOMETRIC NETWORKS

The nonlinearity within a trigonometric network is very distinctive when compared to activation functions of other neural networks. Being periodic in its nature, the sine and cosine activation functions can generate harmonics at the output layer. However, having sines and cosines as activation functions does not ensure any kind of orthonormality for the basis functions fed into the output layer. Because of this, Parseval's theorem may not apply to this kind of network.

6.1 Basis Functions in Trigonometric Networks

Throughout this work, a fully connected feed-forward trigonometric network is used. In a fully connected architecture, each layer is connected to all subsequent layers. This means that in a three-layer network we will have weights connecting the input to the output layer.

The basis functions that will be discussed will then consist of a set of vectors containing the inputs augmented with the outputs of the hidden layer and a value of '1' corresponding to the thresholds of the output layer as shown in figure 6.1. The basis functions can be represented by

$$\mathbf{b} = [x_{1p}, x_{2p}, \dots, x_{N_p}, 1, o_{1p}, o_{2p}, \dots, o_{N_{hp}}]^T,$$

where $[\cdot]^T$ is the transpose operation, \mathbf{b} is of dimension $N_u = N + N_h + 1$, x_{kp} and o_{kp} are random variables, where N_h is the number of hidden units, and $1 \leq p \leq N_v$, N_v being the number of patterns.

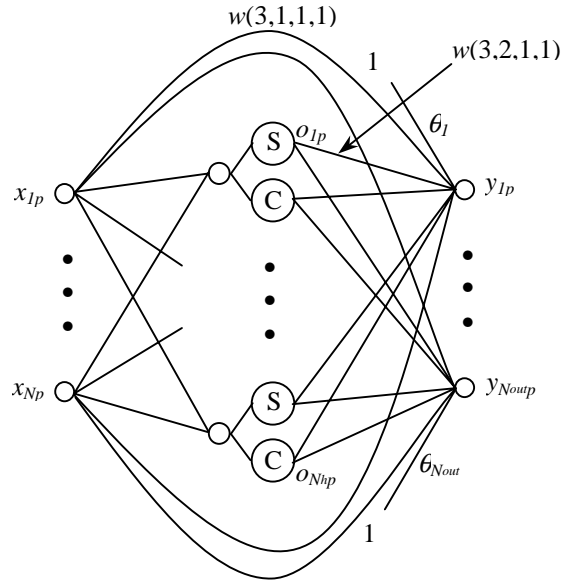


Figure 6.1. A fully connected trigonometric network.

Parseval's theorem is a very useful tool in estimating the amplitudes of the Fourier coefficients of a given Fourier series. These coefficients correspond to the weights of the output layer in the case of a trigonometric network. This can be extremely useful since the weights of the output layer can be solved for if the basis function and the desired values of the output units are known, as can be shown later in this chapter.

Parseval's theorem for a one-dimensional Fourier series having the sine and cosine representation of equation 1.2 can be shown to be

$$\int f_T(t) \cdot s^2(t) \cdot dt = \frac{a_0^2}{4} + \sum_{n=1}^{\infty} \left[\frac{a_n^2}{2} + \frac{b_n^2}{2} \right], \quad (6.1)$$

where $f_T(t)$ is the probability density function of the random variable t .

This equation can be applied to the trigonometric network having \mathbf{x} as the input vector and y_i as the output of the i th unit in the output layer as

$$\sum_{i=1}^{N_{out}} \int y_i^2(\mathbf{x}) f_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} = \sum_{i=1}^{N_{out}} \left[\frac{1}{4} \left\{ \theta_i^2 + \sum_{n=1}^{N_h} \left(w^2(3,2,i,n) \right) \right\} \right],$$

where N_{out} is the number of outputs, N_h is the number of units in the hidden layer, θ_i is the thresholds of the i th output unit and $w(3,2,i,n)$ is the weight connecting the n th unit in the hidden layer to the i th unit in the output layer.

Parseval's equation for the trigonometric network may not, however, be applied directly to the network because it is only valid in the case of orthonormal basis functions. By orthonormalizing the basis functions we can also derive a relationship between the MSE and the weights of the output layer and use this relation to determine the usefulness of a particular unit.

To test for the orthonormality of these basis vectors, a matrix \mathbf{P} is constructed having the dimensions $N_u \times N_u$. The \mathbf{P} matrix consists of the inner product of b_i and b_j , the i th and the j th row elements of \mathbf{b}

$$p_{ij} = \langle b_i, b_j \rangle \quad 1 \leq i, j \leq N_u. \quad (6.2)$$

This inner product can be defined as [29]:

$$\langle b_i, b_j \rangle \equiv E[b_i, b_j],$$

where $E[\]$ is the expectation value operation.

The above equation can be approximated as:

$$E[b_i, b_j] \approx \frac{1}{N_v} \sum_{p=1}^{N_v} b_i^p \cdot b_j^p.$$

where p_k^p is the k th element of \mathbf{b} for the p th pattern.

From the above discussion, it can be seen that if the basis vectors are orthonormal, \mathbf{P} should be a diagonal matrix with $p_{ij} = 1$ for $i = j$ and $p_{ij} = 0$ for $i \neq j$. However, if this is applied directly to the trigonometric network, this will not be the case. This is illustrated by the following example for a network having 4 inputs, 4 hidden units and 1 output using the X.DAT training data file. For this example $p_{ij} =$

```

0.914345 0.895544 0.860008 0.829902 -0.081696 0.889429 -0.058514 0.880302 0.928783
0.895544 0.914503 0.895513 0.859995 -0.058483 0.889872 -0.024427 0.879386 0.928875
0.860008 0.895513 0.914221 0.895237 -0.012196 0.890490 0.028728 0.879020 0.928741
0.829902 0.859995 0.895237 0.913958 0.029326 0.893237 0.071143 0.881221 0.928621
- 0.081696 -0.058483 -0.012196 0.029326 0.081348 -0.017038 0.090380 -0.019281 -0.019499
0.889429 0.889872 0.890490 0.893237 -0.017038 0.918876 0.014799 0.907324 0.957998
- 0.058514 -0.024427 0.028728 0.071143 0.090380 0.014799 0.103891 0.011795 0.013279
0.880302 0.879386 0.879020 0.881221 -0.019281 0.907324 0.011795 0.896333 0.945597
0.928783 0.928875 0.928741 0.928621 -0.019499 0.957998 0.013279 0.945597 1.000224

```

which is clearly far from diagonal.

These basis functions can be forced to be orthonormal after which Parseval's equation can be safely applied. One method to orthonormalize these basis functions is the Schmidt process.

6.2 Orthonormalizing the Basis Functions

The Schmidt process [30] is an iterative procedure of expressing each basis vector as a weighted sum of other basis vectors as to be illustrated.

6.2.1 The Schmidt Process

First define the norm of the random variable b_k as

$$\|b_k\| = \langle b_k, b_k \rangle^{1/2}.$$

Consider the linearly independent random variables b_k , $1 \leq k \leq N_u$, define another set of random variables, z_k , $1 \leq k \leq N_u$, to be the desired orthonormal basis functions. Find

$$z_1 = \frac{b_1}{\|b_1\|}. \quad (6.3)$$

Next calculate

$$c_1 = \langle z_1, p_1 \rangle$$

and find z_2 as

$$z_2 = \frac{b_2 - c_1 z_1}{\|b_2 - c_1 z_1\|}. \quad (6.4)$$

Continue iterating for the next basis vector by calculating c_1 and c_2 as:

$$\begin{aligned} c_1 &= \langle z_1, p_3 \rangle \\ c_2 &= \langle z_2, p_3 \rangle \end{aligned}$$

and calculate

$$z_3 = \frac{b_3 - c_1 z_1 - c_2 z_2}{\|b_3 - c_1 z_1 - c_2 z_2\|}. \quad (6.5)$$

Continue this process until all z_k 's up to $k = N_u$ are found.

This process ensures that the resulting basis functions z_k are orthonormal to each other (i.e., $\langle z_i, z_j \rangle = 1$ for $i = j$ and $\langle z_i, z_j \rangle = 0$ for $i \neq j$ for $1 \leq i, j \leq N_u$). Unfortunately, this process is not very efficient in neural network simulations. This is because each iteration requires a scan through the data to obtain all N_v values for b_k and find inner product values. If the

procedure is taken as is, a minimum of N_u passes through the data is required which degrades the speed performance of the simulation drastically. The process can be modified such that a single pass through the data is required to complete the orthonormalization process.

6.2.2 The Modified Schmidt Process

In this subsection, a more efficient flow of the Schmidt process is presented. Consider forming the desired basis functions as a weighted sum of b_k 's as

$$z_k = \sum_{n=1}^k a_{kn} \cdot b_n \quad (6.5)$$

where a_{kn} is an element of a triangular matrix \mathbf{A} and will be found later on. Note that the above equation is consistent with equation 6.3, 6.4 and 6.5.

With this in mind, values for $\langle z_i, b_j \rangle$ can be found as

$$\begin{aligned} \langle z_i, b_j \rangle &= \left\langle \sum_{n=1}^i a_{in} \cdot b_n, b_j \right\rangle \\ &= \sum_{n=1}^i a_{in} \langle b_n, b_j \rangle = \sum_{n=1}^i a_{in} \cdot p_{nj}. \end{aligned} \quad (6.6)$$

This simplification can save several unnecessary passes through the data.

With one pass, \mathbf{P} can be calculated as in 6.2. Equation 6.6 can then be used to simplify the Schmidt process and general formulas for \mathbf{A} and c_k can then be found in terms of \mathbf{P} only. These general formulas can then be used to develop the modified algorithm as follows:

1. For $k = 1$, find a_{11} as

$$a_{11} = \frac{1}{\sqrt{p_{11}}}.$$

2. For $2 \leq k \leq N_u$ do the following:

(a) Find

$$c_i = \sum_{n=1}^i a_{in} \cdot p_{nk} \quad 1 \leq i \leq k-1.$$

(b) Calculate

$$a_{ki} = -\sum_{n=i}^{k-1} c_n a_{ni} \quad 1 \leq i \leq k-1$$

$$a_{kk} = 1$$

(c) Replace each value of a_{kn} as

$$a_{kn} \leftarrow \frac{a_{kn}}{\sqrt{\sum_{m=1}^k \sum_{j=1}^k a_{kj} a_{km} \cdot p_{jm}}} \quad 1 \leq n \leq k$$

By following this procedure, values of a_{kn} can be found and the triangular matrix \mathbf{A} can then be used to calculate the orthonormal basis functions as in equation 6.5.

However, if the input vector \mathbf{x} contains elements that are dependent among each other, the Schmidt process would fail. To avoid this situation, the linear part of the network is removed from the basis functions. The basis functions would therefore be written as

$$\mathbf{b} = [1, o_{1p}, o_{2p}, \dots, o_{N_h p}]^T.$$

In this case $N_u = N_h + 1$.

6.2.3 Updating the Output Weights

By performing the Schmidt process, the values of the basis functions are changed. This will consequently affect the values of the output units. To maintain these values, the

output layer weights should be updated in a way to cancel the effect of the basis functions change. These new output weights could be calculated as follows:

$$y_k = \sum_{i=1}^{N_u} w'(3,2,k,i) b_i + \sum_{l=1}^N w(3,1,k,l) \cdot x_l = \sum_{j=1}^{N_u} w(3,2,k,j) \cdot z_j + \sum_{q=1}^N w(3,1,k,q) \cdot x_q$$

$$\sum_{i=1}^{N_u} w'(3,2,k,i) \cdot \langle b_i, z_m \rangle = \sum_{j=1}^{N_u} w(3,2,k,j) \cdot \langle z_j, z_m \rangle$$

for $1 \leq k \leq N_{out}$, where y_k is the k th output and $w'(3,2,k,i)$ and $w(3,2,k,i)$ are the original and updated weights from the i th hidden unit to the k th output unit, respectively, and $w(3,1,k,l)$ are the weights from l th input to the k th output.

The right hand side of the above equation will be zero for all $m \neq j$. making use of this and of equation 6.6, the values of the updated weights can be written as:

$$w(3,2,k,m) = \sum_{i=1}^{N_u} w'(3,2,k,i) \sum_{n=1}^m a_{mn} \cdot p_{in} \quad 1 \leq k \leq N_{out}, \quad 1 \leq m \leq N_u. \quad (6.7)$$

Using the above equation, we can update the output weights to get the same output values using the original weights, **A** and **P**.

6.3 The MSE

Orthonormalizing the basis vectors can be used to simplify certain calculations and form them in such a way that makes them useful. One important simplification is that of the MSE. Consider the MSE of the k th output unit which can be expressed as

$$E_k = \left\langle t_k - \sum_{i=1}^{N_u} z_i \cdot w(3,2,k,i) - \sum_{l=1}^N x_l \cdot w(3,1,k,l), \right. \\ \left. t_k - \sum_{j=1}^{N_u} z_j \cdot w(3,2,k,j) - \sum_{q=1}^N x_q \cdot w(3,1,k,q) \right\rangle$$

where t_k is the desired value for the k th output unit.

To find the values of the output weights that minimizes this error, take the first derivative of the error with respect to the output weights and equate it to zero

$$\frac{\partial E_k}{\partial w(3,2,k,j)} = -2\langle t_k, z_j \rangle + 2 \sum_{i=1}^N \langle x_i, z_j \rangle w(3,1,k,i) + 2w(3,2,k,j) = 0 \quad (6.8)$$

$$w(3,2,k,j) = \langle t_k, z_j \rangle - \sum_{i=1}^N \langle x_i, z_j \rangle w(3,1,k,i) \quad (6.9)$$

The error can then be found to be

$$E_k = E'_k - \sum_{i=1}^{N_u} w^2(3,2,k,i) \quad 1 \leq k \leq N_{out} \quad (6.10)$$

where

$$E'_k = \langle t_k, t_k \rangle - 2 \sum_{i=1}^N \langle t_k, x_i \rangle w(3,1,k,i) + \sum_{l=1}^N \sum_{j=1}^N \langle x_l, x_j \rangle w(3,1,k,l) w(3,1,k,j)$$

and the total MSE can be written as

$$E = \sum_{k=1}^{N_{out}} E_k. \quad (6.11)$$

Equation 6.10 is a very interesting result as it shows the MSE for an output unit as a function of the sum of the squares of the output weights. There are several advantages of this representation of the MSE. First, it is easier and faster to calculate the MSE using 6.10. Simulation results show that the two MSE values calculated using 6.10 and calculated using the traditional way, equation 2.5, are similar.

The second and more important benefit of the representation of equation 6.9 is that the level of importance of nodes in the hidden layer is a function of the square of the weights leading away from that node. This result is discussed in the following section.

6.4 Efficient Architectures of Trigonometric Networks

The result obtained in the previous section indicates that the MSE in an orthonormalized trigonometric network can be directly related to the square of the weights of the output layer. Since this is true, this equation can indicate the most important as well as the least important units in the hidden layer. Removing a unit from the hidden layer is followed by the removal of its weights connecting to the output layer. This would increase the MSE by an amount proportional to the square of its weights, as can be inferred from equation 6.10. Similar work has been done on RBF networks [31] where the units of the hidden layer were ordered from the most important to the least important.

In the following discussion, an algorithm used to order the units of the hidden layer of a trigonometric network is presented.

Equation 6.10 was a direct consequence of an orthonormal set of basis functions. In order to apply equation 6.10 to the network and hence acquire an ordering algorithm, a certain type of orthonormalizing scheme should be applied to the basis functions, in this case augmented vectors containing the number '1' corresponding to the thresholds, and the N_h hidden layer outputs. The ordering of the basis functions will only involve the ordering of the hidden layer units which correspond to the last N_h members of the basis functions.

Applying the Schmidt process directly to the basis functions and calculating the square of the weights leading away from the hidden units is not the proper way to do the ordering. A more convenient way to do the ordering process is to apply an iterative version of the Schmidt process.

6.4.1 Ordering the Hidden Units

For the ordering process to be as accurate as possible, an iterative approach to the Schmidt process is applied which can be presented by the following. First use the Schmidt

process to orthonormalize the first element of the basis functions. This includes the threshold input. To find the most important unit, orthonormalize the i th and the $(i+1)$ th elements of the remaining N_h , $1 \leq i \leq N_h - 1$. These two elements correspond to the sine and cosine outputs of the hidden layer. Each time the two elements are orthonormalized, the corresponding weights are calculated by using equation 6.7. These weights are then squared and the values of all the $N_h/2$ pairs are compared. The pair having the highest value of the square of its weights are shifted to the top of the other $N_h - 2$ terms and treated as the most important sine and cosine pair. With these two elements orthonormalized and shifted to the top, the same process is applied to the remaining $N_h - 2$ units and a second most important pair is found and shifted to the top of the remaining $N_h - 4$ terms. This process is repeated until all the pairs have been ordered. Performing the following step can do this ordering process:

- (a) Orthonormalize the first basis function using the modified Schmidt process and find a_{11} .
 (b) For the remaining basis function do the following:

Define $h(i) = i$, $1 \leq i \leq N_u$.

1. For $1 \leq j \leq N_h/2$ do steps 2 through 9.
2. For $j \leq l \leq N_h/2$ do steps 3 through 7.
3. For $2l \leq k \leq 2l + 1$ do steps 4 through 6.

4. Find
$$c_i = \sum_{n=1}^i a_{h(i),h(n)} \cdot Ph(n),h(k) \quad 1 \leq i \leq k - 1$$

$$a_{h(k),h(i)} = - \sum_{n=i}^{k-1} c_n a_{h(n),h(i)} \quad 1 \leq i \leq k - 1$$

$$a_{k(k),h(k)} = 1$$

$$d = \left(\sum_{m=1}^k \sum_{n=1}^k a_{h(k),h(n)} \cdot a_{h(k),h(m)} \cdot Ph(n),h(m) \right)^{1/2}$$

5. Replace

$$a_{h(k),h(n)} \leftarrow \frac{a_{h(k),h(n)}}{d} \quad 1 \leq n \leq k$$

6. Compute

$$w_{h(k)} = \sum_{m=1}^{N_{out}} \sum_{i=1}^{N_u} w'_{m,h(i)} \sum_{n=1}^k a_{h(k),h(n)} \cdot Ph(i),h(n)$$

where $w'_{m,h(i)}$ is the old weights connecting the $h(i)^{\text{th}}$ element in the hidden layer to the m th element of the output layer.

7. Compute

$$sw_l = \sum_{k=2l}^{2l+1} w_{h(k)}^2$$

8. Find m such that $sw_m = \max \{sw_l, j \leq l \leq N_h/2\}$.

9. Do the following:

$$temp_1 = h(2m),$$

$$temp_2 = h(2m + 1),$$

$$h(i) \leftarrow h(i - 2), \quad 2j + 2 \leq i \leq 2m + 1,$$

$$h(2j) = temp_1,$$

$$h(2j + 1) = temp_2.$$

(c) $h(i)$ will contain the indices of the ordered hidden sine and cosine units, calculate the orthonormalized basis functions as

$$z_k = \sum_{n=1}^k a_{h(k),h(n)} \cdot b_{h(n)} \quad \text{for } 1 \leq k \leq N_u$$

and

$$w_{m,h(k)} = \sum_{i=1}^{N_u} w'_{m,h(i)} \sum_{n=1}^k a_{h(k),h(n)} \cdot Ph(i),h(n) \quad \text{for } 1 \leq k \leq N_u, 1 \leq m \leq N_{out}$$

where z_k and $w_{m,h(k)}$ will contain the ordered basis functions and ordered new weights, respectively.

The reason for doing this process in ordering the hidden units comes directly from equation 6.5. This equation indicates that each basis function is a weighted sum of all the previously orthonormalized basis functions and that the first basis function does not depend on any of the other basis functions. Applying the above process will ensure that each pair of sine and cosine units when evaluated will not depend on any of the other basis functions except for the first term, which will be common in all evaluations, and all previously ordered hidden units, which would also be common in all evaluations. This is necessary because the importance of a unit is a measure of the increase in the MSE after the elimination of that particular unit. To compare this increase fairly, the nodes to be evaluated should be dependent on the same number of basis function as all the other units.

After the units have been ordered, the MSE is measured after each of the ordered units is removed one at a time. Fortunately, an advantage of a neural network is that it can tolerate the effect of a missing unit and we can retrain the network and update the remaining output layer weights in such a way to minimize the MSE. After a unit has been removed, the remaining output weights are then updated using the conjugate gradient method and the training MSE measured. In the ideal case, the MSE after eliminating the most important unit should be higher than that resulting in eliminating a unit less important.

6.4.2 Simulation Results

The above theory is applied on a trigonometric network that was trained using several mapping data files. The MSE results are shown in table 6.1 which lists the training MSE as compared to the MSE values obtained from equation 6.11 for the three data files used after the basis functions have been orthonormalized. The two values of the MSE are close to each other with a little amount of error ranging from 0.47 % to 4.40 %.

Table 6.1. Training MSE using different data files

	<i>Training Data File</i>		
	TWOD.TRA	POWER14	OH7.TRN
Structure	8-16-7	14-16-1	20-16-3
Training MSE, equation (2.5)	0.193823	7890.457199	1.711528
Training MSE, equation (6.11)	0.192913	8238.022016	1.692547
Error in MSE	-0.47 %	4.40 %	-1.11 %

Table (6.2) shows the simulation results for the ordering process. After the network has been trained, the new output weights of different hidden units are calculated using equation (6.7). The squares of these weights are shown in table (6.2) in descending order of the importance of their corresponding units. The values shown in the table are the squares of the weights leading away from the given unit including those of the sine and cosine terms.

After the hidden units have been ordered, units are eliminated one at a time in the order of their importance. Table 6.3 shows the indices of the units in order of their importance for the different data files. The network is then retrained and new values for the output weights are calculated to account for the missing unit. The network is then tested and the input data files are applied and the MSE calculated and compared to the MSE of the network without any units missing. The results of the MSE with the least significant and the most significant group of hidden units removed are shown in table 6.4. In this experiment,

the least important half of the hidden units are removed and the MSE measured. Then the most important half of the hidden units are removed and the MSE measured. This table also shows the percentage this MSE has changed from that of the full network.

As expected, the MSE calculated after removing a more important group of units will be higher than that of a less important one.

This is a very interesting result since it indicates that units can be removed from a trigonometric network but yet have acceptable performance. This reduces the size of the network and makes it more efficient. The process of ordering the hidden layer's units leads to more efficient trigonometric networks by eliminating useless units.

Finally, it is worthwhile to mention that this process is not restricted to the trigonometric network and can be extended, with minor changes, to any type of neural network, including radial basis function networks [31].

Table 6.2. Square of output weights after applying the Schmidt process

	<i>Training Data File</i>		
	TWOD.TRA	POWER14	OH7.TRN
Structure	8-16-7	14-16-1	20-16-3
Square of weights of most important unit	3.688695	17462.307525	73.602461
•	1.133037	7404.058093	4.690563
•	0.568761	1340.865442	2.047977
•	0.118638	759.989614	0.848651
•	0.074249	47.560128	0.259905
	0.028230	11.449756	0.282699
	0.016351	1.844053	0.189556
Square of weights of least important unit	0.011916	0.111985	0.062086

Table 6.3. Hidden unit order in the original architecture for different training data files

	<i>Training Data File</i>		
	TWOD.TRA	POWER14	OH7.TRN
Structure	8-16-7	14-16-1	20-16-3
Most important unit index	4	2	6
	2	7	2
•	3	3	4
•	7	6	8
•	1	8	3
	6	1	5
	5	5	7
Least important unit index	8	4	1

Table 6.4. MSE after removing the least and the most important halves of the hidden layer

<i>Training Data File</i>					
TWOD.TRA		POWER14		OH7.TRN	
8-16-7		14-16-1		20-16-3	
MSE	Change (%)	MSE	Change (%)	MSE	Change (%)
0.247012	27.442277	7926.093472	0.451638	2.112652	23.436543
0.265279	36.866647	12143.880684	53.905919	3.214587	87.819663

CHAPTER 7

RELATED NETWORKS

7.1 Sine Networks

A reasonable modification of the trigonometric network is a MLP having only sine, or cosine, activation functions. Simulation results show that these networks are inferior to the previously discussed trigonometric network. A theoretical justification is presented along with the simulation results.

7.1.1 Backpropagation in Sine Networks

Training trigonometric networks using backpropagation was discussed in detail in chapter 2. In this training algorithm the weights of the hidden layer are updated in a way that minimizes the MSE with respect to these weights. This update algorithm is however dependent on the on the gradient of the MSE with respect to the hidden weights. This gradient in turn depends on the gradient of the error with respect to the net functions of the hidden layer, defined as the delta function, as in equations 2.7, 2.8 and 2.9. For the trigonometric network case, the delta function is stated in equation 2.11.

For a three-layer sine network, these delta functions can be rewritten in the following form:

$$\delta_p(2, j) = \cos(\text{net}_p(2, j)) \sum_{l=1}^{N_{out}} \delta_p(3, l) w(3, 2, l, j). \quad (7.1)$$

The error gradients are then

$$\begin{aligned}\frac{\partial E}{\partial w(2,1,j,i)} &= \sum_{p=1}^{N_p} -\delta_p(2,j)x_p(i), \\ \frac{\partial E}{\partial \theta(2,j)} &= \sum_{p=1}^{N_p} -\delta_p(2,j),\end{aligned}\tag{7.2}$$

where $x_p(i)$ is the i th input for the p th pattern. This gradient is then used to update the weights.

An interesting fact is that when the derivative of the activation function, which appears in the delta functions, goes to zero, the delta functions in equation 7.1 go to zero and the gradients in equation 7.2 also go to zero. This leads to a very small or even zero change in the values of the weights or the thresholds which can degrade the performance of the training algorithm. Whenever we come close to the saturation of the activation function, or a flat portion, the delta functions will take smaller and smaller values leading to infinitesimal or zero change in the hidden weights and thresholds.

This cannot occur in the trigonometric network. Consider the delta function in equation 2.10. Due to the phase shift of $\pi/2$ between the sine and the cosine, these two activation functions do not reach a minimum or maximum simultaneously. Whenever one of the activation functions has a zero derivative, the derivative of the other activation function would still have a finite value, thus preventing the delta functions from taking a zero value. This means that trigonometric networks are less likely to encounter regions of slow learning in the weight space.

7.1.2 Simulation results

The sine network was simulated using several mapping data files and its performance compared to that of the trigonometric network having the same pattern storage capability as described in section 3.1. Table 7.1 shows the topologies and the training data files used in these simulations.

Table 7.1. Network Topologies used in sinusoidal MLP training

<i>Training file</i>	<i>Sinusoidal MLP Topology</i>	<i>Trigonometric network topology</i>	<i>Used in</i>
TWOD.TRA	8-12-7	8-16-7	Figure 7.1
SINGLE2.TRA	16-25-3	16-44-3	Figure 7.2
OH7.TRN	20-25-3	20-46-3	Figure 7.3

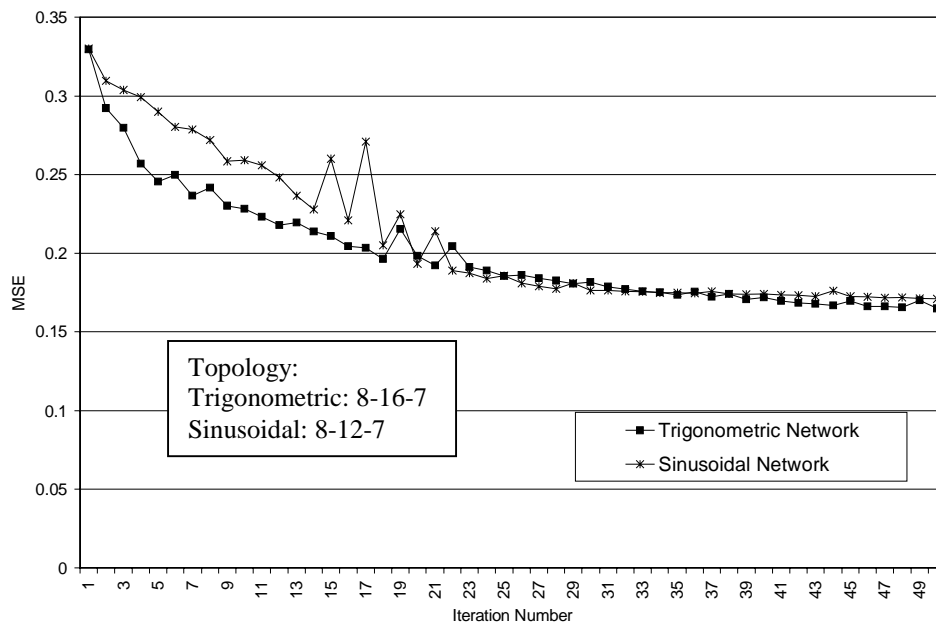


Figure 7.1. Sine network training vs. a trigonometric network using TWOD.TRA.

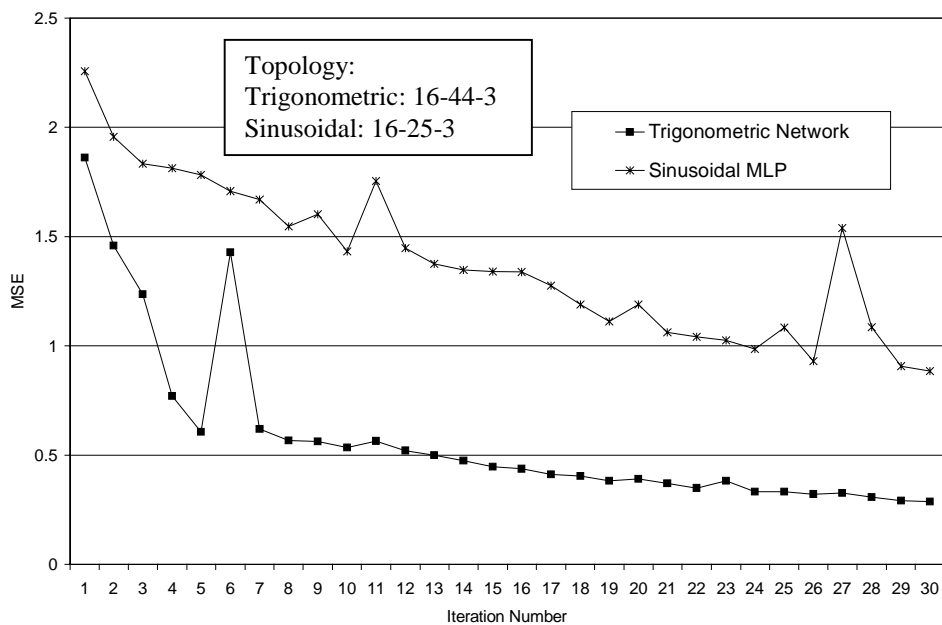


Figure 7.2. Sine network training vs. a trigonometric network using SINGLE2.TRA.

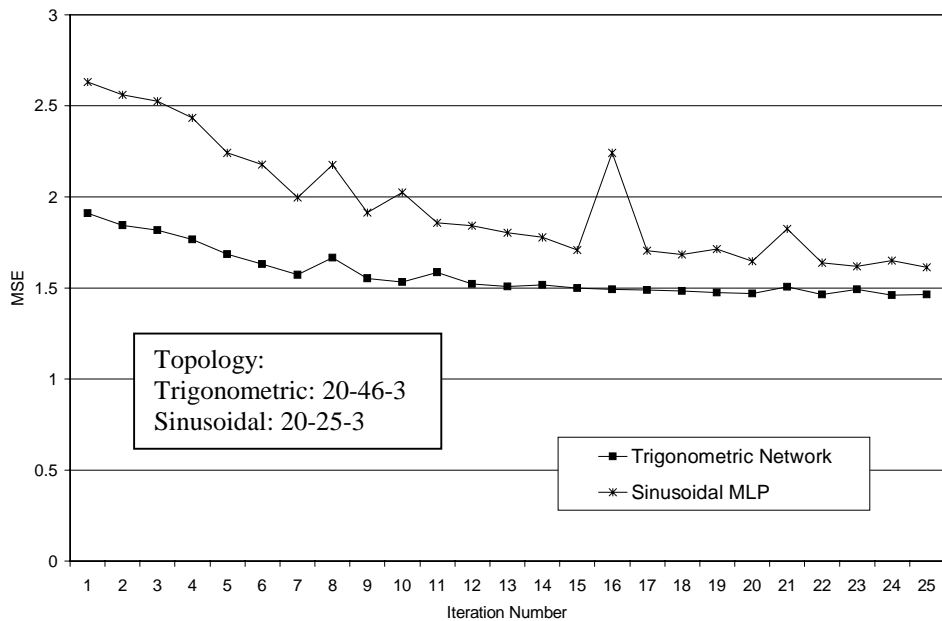


Figure 7.3. Sine network training vs. a trigonometric network using OH7.TRN.

As can be seen from the previous simulation graphs, the sine network is strongly dependent on the input data file. For the data file twod.tra, the sine network performed as well as the trigonometric network. For the other two data files, however, the sine network is inferior to the trigonometric network.

7.2 The Trigonometric Network as a Complex Network

Thus far the trigonometric network model used the real representation of the Fourier series. The complex representation of the series can also be implemented. Since the data used are real, both representations of the Fourier series will be exactly the same with somewhat different coefficients. A relationship between the two coefficients comes directly from the two representations of the trigonometric network. The j th output of the network is given by

$$y_p(j) = \sum_{i=1}^{N_F} [C_s(i, j) \sin(v_p(i)) + C_c(i, j) \cos(v_p(i))] \\ v_p(i) = \sum_{k=1}^N w(2,1,i,k) \cdot x_p(k).$$

The complex form of this output can be given by

$$y_p(j) = \sum_{i=1}^{N_F} [D(i, j) e^{jv_p(i)}].$$

The relationship between the two coefficients can be written as [3]:

$$C_c(i, j) = D(i, j) + D^*(i, j) = 2 \operatorname{Re}(D(i, j)) \\ C_s(i, j) = j(D(i, j) - D^*(i, j)) = -2 \operatorname{Im}(D(i, j)) \\ \text{and } D(i, j) = \frac{1}{2}(C_c(i, j) - jC_s(i, j)),$$

where $D^*(i, j)$ is the complex conjugate of $D(i, j)$.

A similar network with the complex representation was built by Zhu and Paul [32] using different network architecture. Several types of training algorithms have been proposed for complex networks [34,35]. However, the algorithms discussed in this thesis have proven to be more efficient.

CHAPTER 8

CONCLUSIONS

The idea of implementing neural networks in real world applications is not yet very popular. One reason is that people often hesitate to use processors that they do not understand. In this thesis, however, a special type of neural network that can eliminate this drawback is presented. Electrical and computer engineers, already familiar with the Fourier series, can easily understand and use the trigonometric network as a processor.

In this thesis, an architecture for the trigonometric network is reviewed. Training algorithms are developed for this type of network. The network is then compared to one of the most popular neural networks, the sigmoidal multi-layer perceptron. The results of the comparison show the superiority of the proposed trigonometric network over the MLP using both mapping and classification data files.

An algorithm for finding an efficient architecture for the trigonometric network was then developed. This efficient architecture was reached by imposing the conditions necessary for the network to meet Parseval's equation. Useless hidden units were then determined and removed from the network leaving the most important units in the architecture. Hence the hidden units were ordered relative to their importance. Conjugate gradient proved to be efficient in solving linear equations for the output weights. An alternative approach, however, can be used in solving for the output weights by taking advantage of the orthonormalizing process where the weights can be directly found from an inner product of the desired output values and the orthonormal outputs of the hidden layer, as described in the thesis. This method, although faster than the conventional conjugate gradient method, can fail if the basis functions are linearly dependent. This problem can be eliminated by

determining and removing the linearly dependent basis function and proceeding with the Schmidt process.

Based on the results in this thesis, research can be done on the input data file itself. As the Schmidt process was used to determine useless hidden units, a similar algorithm can be developed that can determine which are the least important, or more important, elements in the input vector. This way some amount of redundancy in the input patterns can be eliminated and the network would be more efficient.

A lot of work is being done in the field of artificial neural computing in an attempt to build systems that resemble the biological neuron in its ability to learn. The idea of trying to build an artificial human brain, however, has become obsolete and researchers are focusing their efforts on a task that makes more sense: the production of smaller and more accurate signal processors. The field of neural computing was and will stay one of the most important sciences in the challenge to ensure a better way of life for all mankind.