SECOND ORDER TRAINING ALGORITHMS FOR RADIAL BASIS FUNCTION NEURAL NETWORK

By

KANISHKA TYAGI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

DECEMBER 2011

To my Grandmother

士不可不弘毅，任重而道远---**孔夫子**
('shibukeyibuhongyi, renzhongerdaoyuan')
(*An educated gentleman cannot but be resolute and broad-minded, for he has taken up a heavy responsibility and a long course—Confucius)*

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervising professor Dr Michael T. Manry, who gave me this interesting and challenging problem and helped me through my struggles with it. I thank him for giving me his constant guidance, consistent encouragement and financial support throughout my Master’s program. I am grateful to him for the countless hours he spend discussing various aspects of the research and revising drafts of my papers and the thesis. His method of research and level of perfection in every aspect was a learning experience for me.

Thanks to my lab-mates, especially Dr.Xun Cai for her constant support. I thank her for spending valuable time on debugging the codes in all the wee hours and providing me motivation whenever I needed. It is my pleasure to offer my love and respect to my friends Sandy, Xueyang, Himanshu, Rahul, Sayan and Rajinikant for their generous help. Special thanks to my friends Abhinav and Astha for being extremely supportive during the writing stage of my thesis.

I express my sincere gratitude to my grandparents and parents back at home for their sacrifice, encouragement and patience. Special thanks to my uncle, Pankaj Tyagi for his endless love and support during all this years of studies. Thanks to my cousin sisters Anubhuti, Anshika, Aditi, my brother Harshvardhan and my cousin brothers, Anubhav, Aman, Priyansh for always being cheerful and supportive.

November 18, 2011

ABSTRACT

SECOND ORDER TRAINING ALGORITHMS FOR RADIAL BASIS FUNCTION NEURAL NETWORKS

Kanishka Tyagi, M.S.

The University of Texas at Arlington, 2011

Supervising Professor: Michael T Manry

A systematic two step batch approach for constructing and training of Radial basis function (RBF) neural networks is presented. Unlike other RBF learning algorithms, the proposed paradigm uses *optimal learning factors* (OLF's) to train the network parameters, i.e. spread parameters, mean vector parameters and weighted *distance measure* (DM) coefficients. Newton's algorithm is proposed for obtaining *multiple optimal learning factors* (MOLF) for the network parameters. The weights connected to the output layer are trained by a supervised-learning algorithm based on *orthogonal least squares* (OLS). The error obtained is then back-propagated to tune the RBF parameters. The proposed hybrid training algorithm has been compared with the Levenberg Marquardt and recursive least square based RLS-RBF training algorithms. Simulation results show that regardless of the input data dimension, the proposed algorithms are a significant improvement in terms of convergence speed, network size and generalization over conventional RBF training algorithms which use a *single optimal learning factor* (SOLF). Analyses of the proposed training algorithms on noisy input data have also been carried out. The ability of the proposed algorithm is further substantiated by using *k-fold* cross validation. Initialization of network parameters using Self Organizing Map (SOM), efficient calculation of Hessian matrix for network parameters, Newton's method for optimization, optimal learning factors and orthogonal least squares are the subject matter of present work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Radial basis function neural network

The Radial basis function (RBF) network is a three layer supervised feed-forward network [1] used in interpolation, probability density function estimation and approximation of smooth multivariate functions [2]-[7]. The RBF was first introduced as a solution to the real multivariate interpolation problem. The RBF training algorithm can approximate any multivariate continuous function arbitrarily well on a compact domain if a sufficient number of radial basis functions are given [4].

The most important feature that distinguishes the RBF network from earlier radial basis function based training algorithms is its adaptive nature which generally allows it to utilize a relatively small number of locally tuned units (RBF's). RBF networks were independently proposed by [8]-[11]. The RBF networks covered in the literature mostly differ in their training algorithm. Broadly they can be classified into three paradigms:

(1) No training: in this case, all parameters are calculated and fixed and no training is required. The network in this case is too large, slow and do not have much practical application.

(2) Half training: in this case, the hidden later parameters are calculated either heuristically or by some clustering algorithm and fixed in advance. Only the output weights are trained using the least squares algorithm.

(3) Full training: in this case all parameters and weights are trained using a training algorithm.

Most of the RBF networks studied in the literature are trained using batch learning algorithms, though some sequential learning algorithms, such as the generalized growing and pruning RBF (GGAP-RBF) algorithm have been proposed [12]. However the convergence and generalization performance of the network is an “open problem”. It has been proposed to optimize hidden units in the network to obtain a more compact network as compared to large conventional RBF networks [13]-[15]. However to select a subset network, the algorithm uses a subset selection method based on orthogonal least squares (OLS) or regularized OLS where a full network is designed after all the training observations are presented. Based on this subset selection technique, investigators have proposed a regularized forward selection (RFS) algorithm that combines forward subset selection and zero-order regularization to achieve better generalization [16].

In order to train the RBF network, mostly first order methods are used. Gradient descent learning [17], [18] offers a balance between performance and training speed. These networks are compared with sigmoid hidden unit based feed forward neural networks in [18]-[20]. The combination of steepest descent and Newton’s method would seem to be more promising for unconstrained optimization problems [21]. This method is convergent and has a high convergence rate. Since Newton’s method for the RBF often has non-positive definite or even singular Hessian, Levenberg-Marquardt (LM) or other methods are used instead. However second order methods do not scale well and suffer from heavy computational cost. Although first order methods scale better, they are sensitive to input means and gain factors [22]. A hybrid learning procedure is proposed in [11] which employs clustering algorithms like *K*-means or determining the center for radial basis function and supervised learning for updating output weights connecting the radial basis function unit (hidden unit) to the output unit. In [23] a gradient training algorithm for updating the network parameters (mean vector parameters, spread parameters) and output weights is presented. A novel space filling curves with genetically evolving parameters is proposed in [24].

1.2 Research objectives

Formulation of RBF network via interpolation theory is rather neat. However the use of interpolation based on noisy data could lead to misleading results. The calculation of the pseudo inverse weight matrix increases the cost of calculations when the number of data and hidden units increases. In order to handle this, it is suggested to use orthogonal least squares algorithm to obtain weight matrix [26]. The choice of network parameters is also a bottleneck in designing the RBF neural network. The performance of RBF training can be greatly improved by choosing an efficient output layer training strategy. A hybrid approach is proposed in [25] where parameters are determined, fixed and the output weights are updated using recursive least squares algorithm. However, even the hybrid learning approach [25] suggests that there is an absence of an overall optimal criterion that combines the training of the hidden and output layers and assumes that the whole system is optimality in a statistical sense. Hence a different approach to the design of an RBF network is needed. Our investigation's focus is on the efficient second order optimization of network parameters and the effects of optimal learning factors on the RBF training.

1.3 Organization of thesis

Chapter 2 reviews the construction, notation and training of conventional RBF networks. It introduces the proposed training algorithm and explains the basic topology and notation used in describing the family of proposed RBF training algorithms. We also introduce a new parameter vector of distance measure coefficients. Non heuristic initializations for all the parameters in the proposed training algorithm are discussed. Then we review existing training algorithms that are later used for comparisons with our family of RBF training algorithms. The chapter concludes with problems prevalent in existing RBF training algorithms while initializing the parameters as well when using Newton’s method

Chapter 3 presents the theoretical background and motivation for using distance measure parameter coefficients. Next is the mathematical formulation for updating this novel parameter vector with a learning factor using Newton’s method. We then study the effect of linearly dependent inputs on gradient and Hessian calculations. Next we investigate the effects of noise on the RBF training algorithms. We compare the performance of these training algorithms with recursive least square based RBF training algorithm.

In a similar manner, chapters 4 and 5 present detailed analyses of the spread parameters vector and the mean vector parameters respectively. In addition, chapter 5 describes the use of single and multiple optimal learning factors for updating the mean vector parameters.

Chapter 6 investigates the various combinations possible within the family of proposed training algorithms leading to a set of final training algorithms that are subsequently used for further analyses. We study the effect of using Newton’s method repeatedly on different parameters, the effect of using the proposed weighted DM coefficient on various training algorithms. In the end we study the effect of applying Newton’s method on all the three parameters which paves way for concluding a final set of training algorithms.

Chapter 7 focus on the experimental analyses on the final two training algorithms that performs best in our family of training algorithms and there comparisons with an advance second order method called Levenberg Marquardt (LM) algorithm and RLS-RBF algorithm. Since LM is computationally expensive, we analyze the computational burden of our set of RBF training algorithms and compare it its performance with LM and RLS-RBF. The generalization capability is finally shown using k-fold validation. Chapter 8 presents the conclusion and future work.

CHAPTER 2

CONSTRUCTION AND TRAINING OF RBF NEURAL NETWORK

In this chapter, the conventional RBF network notation and architecture are introduced. We then present an improved RBF network which includes a distance measure (DM) parameter vector and discuss the initialization of RBF network parameters. Three training algorithms are described next. The chapter concludes with a list of problems in existing RBF training algorithms.

2.1 Conventional RBF topology and notation

Without the loss of generality, we restrict ourselves to a three-layer fully connected RBF with non-linear activation functions. The structure of the RBF is shown in Figure 1. The training dataset consists of N_v training patterns $\{\mathbf{x_p}, \mathbf{t_p}\}$ where the *p*th input vector $\mathbf{x_p}$, and its corresponding *p*th desired output vector $\mathbf{t_p}$ have dimensions *N* and *M* respectively. Let the input vectors be augmented by an extra element $\mathrm{x_p}(\mathrm{N}+1)$=1, so that $\mathbf{x_p} = [\mathrm{x_p}(1), \mathrm{x_p}(2) \ldots, \mathrm{x_p}(\mathrm{N}+1)\mathrm{T}$.

The input units are directly connected to the single hidden layer with N_h hidden nodes. It should be noted here that N_h is the key factor not only for the performance but also the computational complexity of the network. For the *k*th hidden unit, $\mathbf{m}_k$(*k=1, 2…*N_h) denotes the *mean vector* of *k*th cluster. $\mathbf{m}_k$ is also known as the kernel vector or center vector.

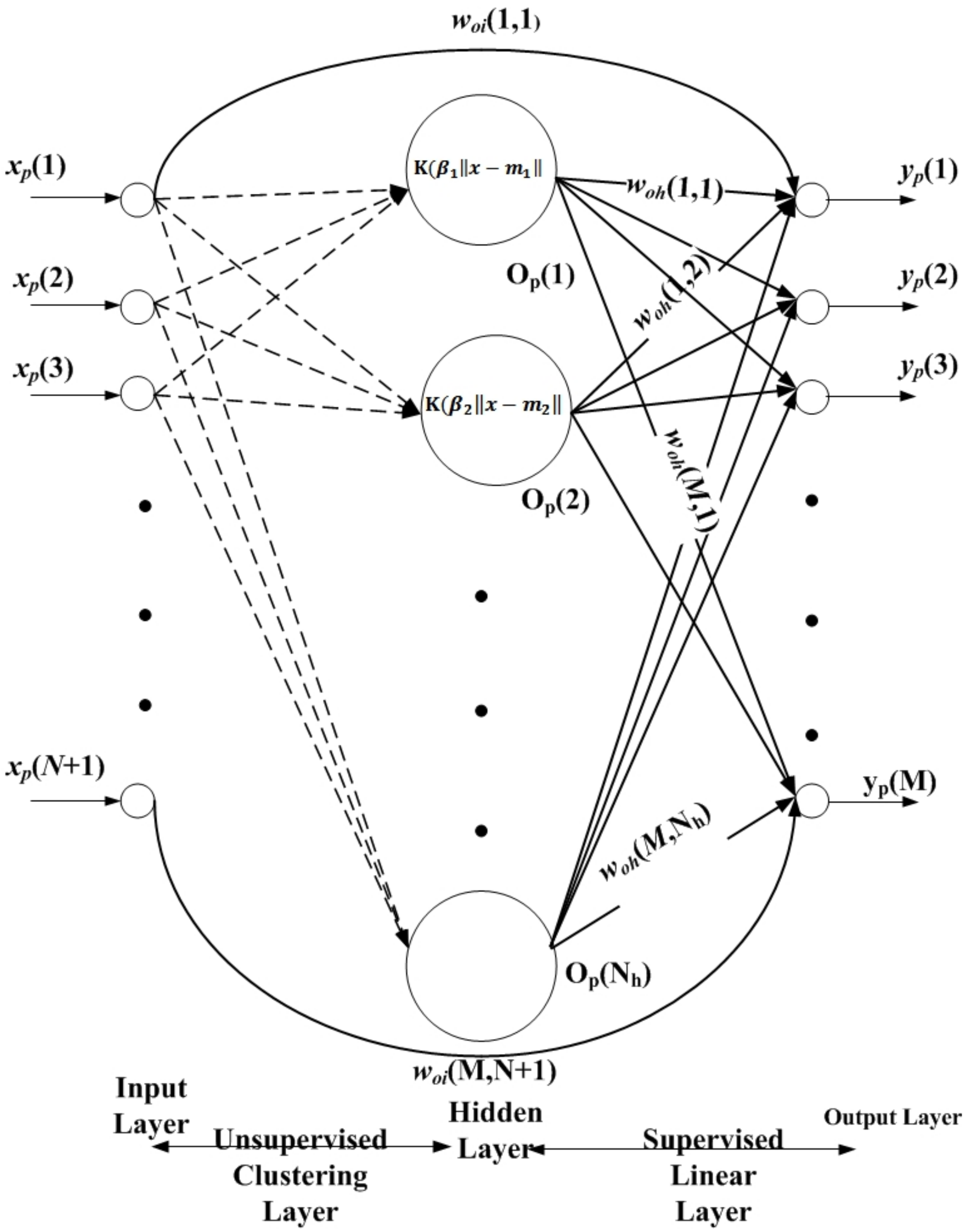


Figure 2.1 Topology of a fully connected RBF Neural Network

In this case, for the *p*th training pattern, the *k*th hidden unit Euclidean *net function* is:

$$net_p(k) = \sum_{n=1}^{N}(x_p(n) - m_k(n))^2 \tag{2. 1}$$

where $m_k(n)$ is the *n*th element of $\mathbf{m}_k$ corresponding to the *n*th input unit.

In Figure1, the dotted lines between input and hidden units signify that instead of weighted sum/Gaussian activation, each hidden unit output $O_{\boldsymbol{p}}(k)$ is obtained by calculating the '*closeness'* of the input $\mathbf{x_p}$ to $\mathbf{m}_{\boldsymbol{k}}$ associated with the *k*th hidden unit. In this case, the *k*th hidden unit output $O_p(k)$ is calculated as a *Gaussian basis function, f*or the *p*th pattern:

$$O_p(k) = e^{-\beta(k)net_p(k)} \tag{2. 2}$$

Here $\beta(k)$ is the *spread parameter* defined as the inverse of the width of the *k*th hidden unit Gaussian function with mean vector $\mathbf{m}_{\boldsymbol{k}}$. The mean vector parameter $\mathbf{m}_{\boldsymbol{k}}$ and spread parameter $\beta(k)$ are conventional parameters of RBF neural networks. The hidden layer is fully connected to the output layer via output weights. If $w_{oh}(i,k)$ denotes the weight connecting the *k*th hidden unit's activation $O_p(k)$ to the *i*th output $y_p(i)$ then the output $y_p(i)$ for the *p*th training pattern is:

$$y_p(i) = \sum_{k=1}^{N_h} w_{oh}(i,k) \cdot O_p(k) + \sum_{n=1}^{N+1} w_{oi}(i,n) \cdot x_p(n) \tag{2. 3}$$

We also include the bypass weights $w_{oi}(i,n)$ connecting the *n*th input unit *to the i*th output unit. The linear output response vector $\boldsymbol{y_p}$in (2.3) can be re-written as:

$$\boldsymbol{y_p} = \boldsymbol{W_{oi}} \cdot \boldsymbol{x_p} + \boldsymbol{W_{oh}} \cdot \boldsymbol{O_p} \tag{2. 4}$$

For convenience, we can augment the input vector as:

$$\bar{x}_p(n) = \begin{cases} x_p(n) & n = 1,2,\dots,N \\ 1 & n = N+1 \\ O_p(n-N-1) & n = N+2,\dots.,N_u \end{cases} \tag{2. 5}$$

where N_u denotes $N+1+N_h$. In vector notation, (2.5) can be rewritten as where $\bar{\boldsymbol{x}}_{\boldsymbol{p}} = \left[\boldsymbol{x}_{\boldsymbol{p}}^T{:}\,\mathbf{1}{:}\,\boldsymbol{O}_{\boldsymbol{p}}^T\right]$. Similarly the weights can be denoted as:

$$w_o(i,k) = \begin{cases} w_{oh}(i,k) & 1 \le k \le N \\ w_{oi}(i,N+1) & k = N+1 \\ w_{oi}(i,k-N-1) & N+2 \le k \le N_u \end{cases} \tag{2. 6}$$

In vector notation (2.6) can be re-written as $\boldsymbol{W_o} = [\boldsymbol{W_{oi}}{:}\,\boldsymbol{W_{oh}}]$ where $\boldsymbol{W_o}$denotes all the output weights. Using the augmented input vector in (2.5) and output weights in (2.6), (2.4) can be re-written as:

$$y_p(i) = \sum_{k=1}^{N_u} w_o(\,i,k)\bar{x}_p(k) \tag{2.7}$$

In vector notation (2.7) can be rewritten as:

$$\boldsymbol{y_p} = \boldsymbol{W}_o \cdot \bar{\boldsymbol{x}}_{\boldsymbol{p}} \tag{2.8}$$

The training error for each pattern is

$$\boldsymbol{E_p} = \sum_{i=1}^{M} \left[t_p(i) - y_p(i)\right]^2 \tag{2.9}$$

Here $t_p(i)$ denotes the *i*th desired outputs for the *p*th input pattern. In batch mode training, the error function of a RBF is measured using the *Mean Square Error* (MSE) as:

$$\boldsymbol{E} = \frac{1}{N_v}\sum_{\mathrm{p}=1}^{N_\mathrm{v}} E_p = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - y_p(i)\right]^2 = \frac{1}{N_v}\sum_{\mathrm{p}=1}^{N_\mathrm{v}}\left(\boldsymbol{t_p} - \boldsymbol{y_p}\right)^T\left(\boldsymbol{t_p} - \boldsymbol{y_p}\right) \tag{2.10}$$

where $\boldsymbol{t_p}$ and $\boldsymbol{y_p}$ are column vectors. The general purpose of minimizing the MSE is weight optimization [27]-[29].

2.2 RBF networks initialization

In this sub-section we introduce and initialize the weighted distance measure (DM) coefficients vector. We then initialize conventional network parameters including the spread parameters and mean vector parameters.

2.2.1 Initializing the weighted distance measure

Let the N-dimensional vector $\boldsymbol{c}$ store coefficients for the weighted DM vector. Let $\sigma(n)$ represent the standard deviation of input $x(n)$.Then $c(n)$ is defined as:

$$c(n) = 1 \Big/ \left[\sum_{m=1}^{N} \frac{1}{\sigma^2(m)}\right] \sigma^2(n) \qquad ((2.11)$$

For the p^{th} training pattern, the k^{th} hidden unit *net function* is now modified as:

$$\widehat{net}_p(k) = \sum_{n=1}^{N+1} c(n) \cdot (x_p(n) - m_k(n))^2 \qquad ((2.12)$$

where $\widehat{net}_p(k)$ is the weighted DM. The *Gaussian basis function* is:

$$\hat{O}_p(k) = e^{-\beta(k)\widehat{net}_p(k)} \qquad (2.13)$$

Following (2.3), the i^{th} output for the p^{th} training pattern when the weighted DM is used is,

$$\hat{y}_p(i) = \sum_{k=1}^{N_h} w_{oh}(i,k) \cdot \hat{O}_p(k) + \sum_{n=1}^{N+1} w_{oi}(i,n) \cdot x_p(n) \qquad ((2.14)$$

2.2.2 Initializing spread parameters and mean vector parameters

We use self-organizing feature map (SOM) clustering in order to initialize the spread parameter vector $\boldsymbol{\beta}$ and mean vector parameter $\boldsymbol{m_k}$. SOM is a variation of adaptive *K*-means clustering where we cluster the data set based upon the relative distance.

For initializing $\boldsymbol{\beta}$, let $\mathbf{D}$ be an N_h- by-$N_h$matrix comprising of elements $\mathrm{d}(\boldsymbol{m_k}, \boldsymbol{m_u})$ which are the distances between the mean vector parameters,

$$\mathrm{d}(\boldsymbol{m_k}, \boldsymbol{m_u}) = \sum_{\mathrm{n}=1}^{\mathrm{N}} [m_k(n) \ - m_u(n)\ ]^2 \qquad ((2.15)$$

We reorder each row in $\mathbf{D}$ so that the smallest elements come first while ignoring the zero valued distances. For the k^{th} hidden unit, if L is the user-chosen number of mean vectors $\boldsymbol{m_k}$ that we want to activate in $O_p(k)$such that:

$$O_p(k) \geq e^{-1} \qquad (2.16)$$

then we choose the L^{th} smallest distance for each row of $\mathbf{D}$ to form the vector $\boldsymbol{d}$. For the k^{th} hidden unit, $d(k)$, the elements of vector $\boldsymbol{d}$ should satisfy the relation:

$$e^{-\beta(k)d(k)} = e^{-1} \qquad (2.17)$$

Hence,

$$\beta(k) = \frac{1}{d(k)} \qquad (2.18)$$

for $1 \leq k \leq N_h$. This method of initialization ensures that the $\beta(k)$ coefficients are neither too big nor too small. Geometrically, this condition makes sure that individual Gaussian functions are not too peaked nor too flat; both conditions should be avoided in practice. For a global network L can be equal to N_h. In our experiments we use local RBF networks and therefore choose the value of L to be $\frac{N_h}{4}$.

2.3 Existing training algorithms

Current RBF training algorithms can be classified as the first order and the second order methods [30], [31]. Gradient descent methods are first order and Newton related methods are second order [32]-[34]. In [30] Battiti reviewed first and second order algorithms for learning in neural networks. First order methods are fast and effective for large scale problems [35] while second order techniques have higher precision for small scale. We present three algorithms based on different learning paradigms in order to distinguish them from our family of proposed RBF training algorithms.

2.3.1 K-means RLS algorithm

A hybrid learning procedure for training RBF networks is described in [25]. The *K*-means algorithm (see Appendix A for details) for training the hidden layer is applied first. It is then followed by the recursive least squares algorithm (RLS) [53] for training the output layer.

The input layer has the dimensionality N as before. Given the desired value of N_h and training data, the mean vector parameter $\boldsymbol{m_k}$ are computed from the *K*-means algorithm. To determine the spread parameters, a heuristic presented in [25], [36] states that spread parameter is initialized so as to cover the input space with receptive fields as uniformly as possible. Based on this heuristic the present hybrid algorithm assigns the same width vector σ to all the Gaussian functions given by $\sigma = d_{max}/\sqrt{2N_h}$, where d_{max} is the maximum distance between the chosen centers [8].

Once the training of the hidden layer is complete, the output weights are optimized. For discrete time instant *t,* the N_h*-by-1* hidden output vector is:

$$\boldsymbol{O_p^t} = \left[O_p^t(1), O_p^t(1), \ldots\ O_p^t(N_h)\right]^T \qquad (2.19)$$

where,

$$net_p^t(k) = \left\|\boldsymbol{x_p^t} - \boldsymbol{m_k}\right\|^2 \qquad (2.20)$$

The output response $\boldsymbol{y_p}$ is then given by

$$\boldsymbol{y_p} = \boldsymbol{W_{oh}}.\boldsymbol{O_p^t} \qquad (2.21)$$

The supervised training for the output layer is carried out using the recursive least square (RLS) algorithm. Being a recursive algorithm, *'n'* represents the value at current instant of time '*t*' and '*n-1*' is in the previous instant. We start by calculating the autocorrelation matrix $\boldsymbol{R_a^n}$ defined as:

$$\boldsymbol{R_a^n} = \sum_{t=1}^{n} \boldsymbol{O_p^t} \cdot (\boldsymbol{O_p^t})^T \qquad ((2.22)$$

Similarly the cross correlation matrix $\boldsymbol{C_a^n}$ is defined as:

$$\boldsymbol{C_a^n} = \sum_{t=1}^{n} \boldsymbol{O_p^t} \cdot (\boldsymbol{t_p^t})^T \qquad ((2.23)$$

If $\boldsymbol{W_o^n}$ is the output weight matrix, then we have the following linear equation:

$$\boldsymbol{R_a^n} \cdot (\boldsymbol{W_o^n})^T = \boldsymbol{C_a^n} \qquad (2.24)$$

In order to solve (2.24) we could first invert the autocorrelation matrix $\boldsymbol{R_a^n}$ and then multiply the resulting inverse matrix $(\boldsymbol{R_a^n})^{-1}$by the cross-correlation matrix $\boldsymbol{C_a^n}$ which is what we do in least squares. However when the size of the hidden layer N_h is large, which is often the case, computation of the inverse matrix $(\boldsymbol{R_a^n})^{-\mathbf{1}}$ is computationally expensive. The RLS algorithm takes care of this computational difficulty. Reformulating (2.23), we have:

$$\begin{aligned}\boldsymbol{C_a^n} &= \sum_{t=1}^{n-1} \boldsymbol{O_p^t} \cdot (\boldsymbol{t_p^t})^T + \boldsymbol{O_p^n} \cdot (\boldsymbol{t_p^n})^T \\ &= \boldsymbol{C_a^{n-1}} + \boldsymbol{O_p^n} \cdot (\boldsymbol{t_p^n})^T \\ &= \boldsymbol{R_a^{n-1}} \cdot (\boldsymbol{W_o^{n-1}})^T + \boldsymbol{O_p^n} \cdot (\boldsymbol{t_p^n})^T\end{aligned} \qquad ((2.25)$$

where, we first isolate the term corresponding to discrete time instant *t=n* from the summation in (2.23) and in the last line we use (2.24) replacing *n* with *n-1*. Next, we add and subtract $\boldsymbol{O}_{\boldsymbol{p}}^{n}\left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T}(\boldsymbol{W}_{\boldsymbol{o}}^{n-1})^{T}$ leaving the equation unchanged. We factor out the common terms:

$$\boldsymbol{C}_{\boldsymbol{a}}^{n} = \left[\boldsymbol{R}_{\boldsymbol{a}}^{n-1} + \boldsymbol{O}_{\boldsymbol{p}}^{n} \cdot \left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T}\right](\boldsymbol{W}_{\boldsymbol{o}}^{n-1})^{T} + \boldsymbol{O}_{\boldsymbol{p}}^{n}\left[\left(\boldsymbol{t}_{\boldsymbol{p}}^{n}\right)^{T} - \left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T} \cdot (\boldsymbol{W}_{\boldsymbol{o}}^{n-1})^{T}\right] \quad ((2.26)$$

The first set of brackets on the right side is recognized as the auto correlation function:

$$\boldsymbol{R}_{\boldsymbol{a}}^{n} = \boldsymbol{R}_{\boldsymbol{a}}^{n-1} + \boldsymbol{O}_{\boldsymbol{p}}^{n} \cdot \left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T} \quad (2.27)$$

Using the second set of brackets on the right side, we introduce a new term called the *prior estimation error* α^{n}*,*

$$\begin{aligned}\boldsymbol{\alpha}^{n} &= \left(\boldsymbol{t}_{\boldsymbol{p}}^{n}\right)^{T} - \left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T} \cdot (\boldsymbol{W}_{\boldsymbol{o}}^{n-1})^{T} \\ &= \left(\boldsymbol{t}_{\boldsymbol{p}}^{n}\right)^{T} - (\boldsymbol{W}_{\boldsymbol{o}}^{n-1})^{T} \cdot \left(\boldsymbol{O}_{\boldsymbol{p}}^{n}\right)^{T}\end{aligned} \quad ((2.28)$$

$\alpha^{n}$is based on the old estimate $\boldsymbol{W}_{\boldsymbol{o}}^{n-1}$ of the weight vector that we had before the weight estimate was updated.

Hence we can rewrite (2.26) as:

$$\boldsymbol{C}_{\boldsymbol{a}}^{n} = \boldsymbol{R}_{\boldsymbol{a}}^{n} \cdot \left(\boldsymbol{W}_{\boldsymbol{o}}^{n-1}\right)^{T} + \boldsymbol{O}_{\boldsymbol{p}}^{n}\alpha^{n} \quad (2.29)$$

Applying this equation in (2.25) yields:

$$\boldsymbol{R}_{\boldsymbol{a}}^{n} \cdot (\boldsymbol{W}_{\boldsymbol{o}}^{n})^{T} = \boldsymbol{R}_{\boldsymbol{a}}^{n} \cdot \left(\boldsymbol{W}_{\boldsymbol{o}}^{n-1}\right)^{T} + \boldsymbol{O}_{\boldsymbol{p}}^{n}\alpha^{n} \quad (2.30)$$

which may be expressed in the desired from for updating the weight vector after multiplying by the inverse matrix $(\boldsymbol{R}_{\boldsymbol{a}}^{\boldsymbol{n}})^{-1}$,

$$\boldsymbol{W}_{\boldsymbol{o}}^{n} = \boldsymbol{W}_{\boldsymbol{o}}^{n-1} + (\boldsymbol{R}_{\boldsymbol{a}}^{n})^{-1} \cdot \boldsymbol{O}_{\boldsymbol{p}}^{n}\alpha^{n} \quad (2.31)$$

Using matrix inversion and the symmetry property, we use a recursive formulation to compute $(\boldsymbol{R}_{\boldsymbol{a}}^{\boldsymbol{n}})^{-1}$by introducing two new definitions

1) Let $(\boldsymbol{R}_{\boldsymbol{a}}^{\boldsymbol{n}})^{-1} = \boldsymbol{P}^{\boldsymbol{n}}$, where the $\boldsymbol{P}^{\boldsymbol{n}}$ matrix is defined as:

$$\boldsymbol{P}^n = \boldsymbol{P}^{n-1} - \frac{\boldsymbol{P}^n \boldsymbol{O}_{\boldsymbol{p}}^n \left(\boldsymbol{O}_{\boldsymbol{p}}^n\right)^{\boldsymbol{T}} \boldsymbol{P}^{n-1}}{\boldsymbol{1} + \left(\boldsymbol{O}_{\boldsymbol{p}}^n\right)^{\boldsymbol{T}} \boldsymbol{P}^{n-1} \boldsymbol{O}_{\boldsymbol{p}}^n} \tag{(2.32)}$$

2) Let the *gain vector* $\boldsymbol{g}^{\boldsymbol{n}}$ be defined as:

$$\begin{aligned} \boldsymbol{g}^{\boldsymbol{n}} &= (\boldsymbol{R}_{\boldsymbol{a}}^n)^{-1} \boldsymbol{O}_{\boldsymbol{p}}^n \\ &= \boldsymbol{P}^n \boldsymbol{O}_{\boldsymbol{p}}^n \end{aligned} \tag{(2.33)}$$

We update the old estimate $\boldsymbol{W}_{\boldsymbol{o}}^{n-1}$ to its new value $\boldsymbol{W}_{\boldsymbol{o}}^{n}$ as follows:

$$\boldsymbol{W}_{\boldsymbol{o}}^{n-1} \leftarrow \boldsymbol{W}_{\boldsymbol{o}}^{n-1} + \boldsymbol{g}^n \alpha^n \tag{2.34}$$

To initialize the algorithm we set $\boldsymbol{W}_{\boldsymbol{o}}^{0} = \boldsymbol{0}$ and $\boldsymbol{P}^0 = \epsilon^{-1}\boldsymbol{I},$ where ε is a small positive constant

2.3.2 Training algorithm based on output weight optimization

Introducing the weighted DM coefficients, we develop a hybrid algorithm consisting of 2 steps:

(1) Initialize the RBF network parameters based on the method of section 2.2.

(2) Fix these parameters and use output weight optimization (OWO) for solving the output weights linear equation.

OWO is a technique to solve for weights connected to the outputs of the network [37], [38]. Since the outputs have linear activations, finding the weights connected to the outputs is equivalent to solving a system of linear equations.

Including the weighted DM coefficients based Gaussian basis function of (2.13), we find the gradient of E with respect to the output weights for *k*th hidden unit and *m*th output unit as:

$$\begin{aligned} \hat{g}_o(m,k) &= \frac{\partial E}{\partial w_o(m,k)} = -\frac{2}{N_v} \sum_{p=1}^{N_v} \left[t_p(m) - \sum_{i=1}^{N_u} w_o(m,i) \bar{x}_p(i) \right] \cdot \bar{x}_p(k) \\ &= -2 \left[c(m,k) - \sum_{i=1}^{N_u} w_o(m,i) \cdot r(i,k) \right] \end{aligned} \tag{(2.35)}$$

where the elements of autocorrelation matrix $\boldsymbol{R}$ has the following form :

$$r(k,i) = \frac{1}{N_v} \sum_{p=1}^{N_v} \bar{x}_p(k) \cdot \bar{x}_p(i) \tag{2.36}$$

and the elements of cross correlation matrix $\boldsymbol{C}$ has the following form:

$$c(k,m) = \frac{1}{N_v} \sum_{p=1}^{N_v} \bar{x}_p\,(k) \cdot t_p(m) \tag{2.37}$$

Setting $\hat{g}_o(m,k) = 0$, the k^{th} equation in the m^{th} set of equation is:

$$c(k,m) = \sum_{i=1}^{N_u} r(k,i) \cdot w_o(m,i) \tag{2.38}$$

(2.38) can be written in vector form as:

$$\boldsymbol{C} = \boldsymbol{R} \cdot \boldsymbol{W_o}^T \tag{2.39}$$

We often have (2.39) ill-conditioned [39], [40], meaning that the determinant of $\boldsymbol{R}$ is close to 0, it is therefore often unsafe to use *Gauss-Jordan* elimination. Therefore the *singular value decomposition* (SVD) [41], *LU decomposition* [42] and *conjugate gradient algorithm* (CG) are better. However Equation (2.38) is most easily solved using *orthogonal least squares* (OLS) which is equivalent to using the QR decomposition [41].

2.3.3 Levenberg-Marquardt algorithm

The *Levenberg-Marquardt* (LM) algorithm is a compromise between Newton's method which converges rapidly near local or global minima but may diverge and gradient descent which has assured convergence through a proper selection of step size parameter but converge slowly [43], [44]. In Newton's method, we minimize the quadratic approximation of the error function $E(\boldsymbol{w})$ around the current weights $\boldsymbol{w}$:

$$E(\boldsymbol{w} + \Delta\boldsymbol{w}) \approx E(\boldsymbol{w}) + \Delta\boldsymbol{w}^T\boldsymbol{g} + \frac{1}{2}\Delta\boldsymbol{w}^T\boldsymbol{H}\Delta\boldsymbol{w} \tag{2.40}$$

Here $\boldsymbol{w}$ is the weight vector of dimension N_w in which all training weights are stored, $\boldsymbol{g}$ is the negative gradient vector (negative Jacobian) for training error $E(\boldsymbol{w})$ as:

$$g(m) = -\frac{\partial E}{\partial w(m)} \tag{2.41}$$

For Hessian matrix $\boldsymbol{H}$, which is the second order partial derivative of $E(\boldsymbol{w})$ with respect to $\boldsymbol{w}$**,** the *m*th row and *n*th column element is:

$$h(m,n) = \frac{\partial^2 E}{\partial w(m) \partial w(n)} \tag{2.42}$$

(2.40) is minimized when,

$$\boldsymbol{g} + \boldsymbol{H}\Delta\boldsymbol{w} = \boldsymbol{0} \tag{2.43}$$

This leads to the weight updating strategy of Newton's method:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{H}^{-1} \cdot \boldsymbol{g} \tag{2.44}$$

The LM algorithm [42]-[44] is a sub-optimal method which updates all weights as:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + [\boldsymbol{H} + \lambda \cdot diag(\boldsymbol{H})]^{-1} \cdot \boldsymbol{g} \tag{2.45}$$

where λ is a controlling factor which tunes LM either towards the first order or the second order methods. $\boldsymbol{I}$ is the identity matrix.

For training the network using LM we take the following steps:

(1) Present all patterns to the network and compute the *Mean Square Error* (MSE).

(2) Compute the Hessian and gradient matrices for all the weights.

(3) Calculate the updated weights using (2.40)

(4) Re-compute the MSE by using the updated weights, if the new error is smallest than that computed in (1) then reduce λ and go back to (1); if the error is not reduced increase λ.

(5) The algorithm converges when the norm of the gradient is less than some pre-determined value, so the MSE has been reduced to a fixed error.

LM converges faster [45] than many other training algorithms. However, it requires $O(N_w^3)$ storage and calculations. Further, owing to the effort required to compute $\boldsymbol{H}$ and λ, LM is practical only for small networks.

2.4 Problems with existing RBF training algorithms

Existing RBF training algorithms suffer from five major problems:

1. The network parameters are not trained well.
2. Noise input contaminates the network parameters.
3. Newton’s method cannot find all weights when Hessian is singular.
4. The clustering process used to obtain the mean vector parameters is not optimized for the network.
5. The algorithm used to obtain the output weight matrix is usually either steepest descent based or recursive.

Most of the RBF training algorithms have two stages per iteration or a total of two stages, resulting in poor performance, since output errors are not fed back to modify input weights. Investigators compensate for this by adding a lot of hidden units such as in support vector machines (SVM). An example of this is the extreme learning machine (ELM) [45] which is a large MLP trained only with OWO.

CHAPTER 3

OPTIMIZATION OF WEIGHTED DISTANCE MEASURE

After the network parameters have been initialized as in section 2.2, the basic framework for the proposed family of RBF training algorithms is set. We now turn our focus on the second stage where Newton's method is used to optimize the weighted distance measure (DM).

Most real world data sets are contaminated with noises either in the context of pattern classification or nonlinear regression hence it is logical to see the effect of noise on the proposed RBF training algorithms.

3.1 Theoretical background

Euclidean distance measure is the criterion for clustering of the training data in conventional training algorithms. In [46] it has been proposed that the conventional Euclidean DM function does not include the perceptual quality in its definition. Many other literatures, particularly in the field of speech processing [47] suggest that using a weighted DM gives better quality results and less spectral distortion than its conventional Euclidean counterpart. Therefore including a weighted DM in the proposed training algorithm seems a logical extension. The SOM that we use in our proposed training algorithms and the initial spread parameters and mean vector parameters employs the use the weighted Euclidean distance measure.

3.2 Mathematical treatment

A typical error function used in the training of RBF is described in (2.9). The hidden unit output $\hat{O}_p(k)$ is given by (2.13) and $\widehat{net}_p(k)$ is as defined in (2.12). Including the weighted DM learning factor vector $\boldsymbol{d_c}$ for updating $\boldsymbol{c}$ we have:

$$\widehat{net}_p(k) = \sum_{n=1}^{N}(c(n) + d_c(n)) \cdot [x_p(n) - m_k(n)]^2 \qquad (3.1)$$

where $\boldsymbol{c}$ is the weighted DM coefficients vector that will be optimized. The output vector $\hat{y}_p(i)$ is same as described in (2.14). By calculating the first order gradient of $\boldsymbol{c}$ we obtain:

$$g_c(n) = -\frac{\partial E}{\partial d_c(n)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \hat{y}_p(i)] \cdot \frac{\partial \hat{y}_p(i)}{\partial d_c(n)} \qquad ((3.\ 2)$$

where

$$\frac{\partial \hat{y}_p(i)}{\partial d_c(n)} = -\sum_{k=1}^{N_h} w_o(i,k) \cdot \hat{O}_p(k) \cdot (-\beta(k)) \cdot \left[x_p(n) - m_k(n)\right]^2 \qquad ((3.\ 3)$$

The Hessian matrix element is:

$$h_c(u,v) = \frac{\partial^2 E}{\partial d_c(u)\partial d_c(v)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\left[\sum_{i=1}^{M}\frac{\partial \hat{y}_p(i)}{\partial d_c(u)} \cdot \frac{\partial \hat{y}_p(i)}{\partial d_c(v)}\right] \qquad ((3.\ 4)$$

This results in the following linear equation:

$$\boldsymbol{H_c} \cdot \boldsymbol{d_c} = \boldsymbol{g_c} \qquad (3.\ 5)$$

Equation (3.5) looks similar to the linear equations in (2.39). The linear equation in (3.5) can be solved using linear equation solver such as steepest descent, conjugate gradient algorithm and OLS. Owing to single step minimization, and ability to overcome the computational complexity of second order Newton's method without compromising the convergence issue, OLS is the preferred choice. For the *n*th input unit, $c(n)$ is then updated as:

$$c(n) \leftarrow c(n) + d_c(n) \tag{3. 6}$$

3.3 Algorithm for optimizing weighted DM coefficients

We follow a nomenclature for all the algorithm and figures discussed here and subsequent chapters. In *{N(param1) + param2},* '*N'* denotes Newton's method applied on parameter '*param1'*, while '*param2'* are the parameters which have been fixed during the initialization part and have not been modified or *"tuned"* during the training process. For example *{N(c)+B+m}* means Newton's method has been applied only on the weighted DM keeping spread parameters vector and mean vector parameters fixed during the training process.

The formal algorithm for optimized weighted DM RBF (OWDM-RBF) training algorithm is described as follows:

(1) Read the data file and normalizing the inputs to zero mean and unit variance.

(2) Initialize the weighted DM coefficients, spread parameters vector and mean vector parameters. In each iteration of the training algorithm, the steps are as follows

(3) Calculate gradient of $\boldsymbol{c}$ from (3.2) and Hessian matrix elements from (3.4).

(4) Solve the linear equation in (3.5) via OLS.

(5) Update weighted DM coefficients using (3.6).

(6) Solve linear equations for all the output weights using OWO.

Contrary to [25] the updated weighted DM are fed back to calculate $\boldsymbol{c}$ for next iteration in our proposed training algorithm.

3.4 Effect of linear dependence in the input layer

As discussed in section 2.4, the singularity of the Hessian matrix is a major drawback of Newton's method. We now investigate as to how the Hessian matrix is modified if a linearly dependent input is used in the training algorithm. A linearly dependent input can be modeled as:

$$x_p(N+1) = \sum_{n=1}^{N} b(n)x_p(n) \tag{(3. 7)}$$

During the weighted DM adaption, the expression for gradient given by (3.2) can be rewritten as:

$$g_c(n) = -\frac{\partial E}{\partial d_c(n)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - \hat{y}_p(i) - w_{oi}(i, N+1)\sum_{n=1}^{N} b(n)x_p(n)\right] \cdot \frac{\partial \hat{y}_p(i)}{\partial d_c(n)} \tag{3. 8}$$

The hidden unit output $\hat{O}_p(k)$ in $\frac{\partial \hat{y}_p(i)}{\partial d_c(n)}$ gets additional terms as:

$$O'_p(k) = \hat{O}_p(k) \cdot e^{-\beta(k)\cdot[c(N+1)[x_p(N+1) - m_k(N+1)]^2]} \tag{3. 9}$$

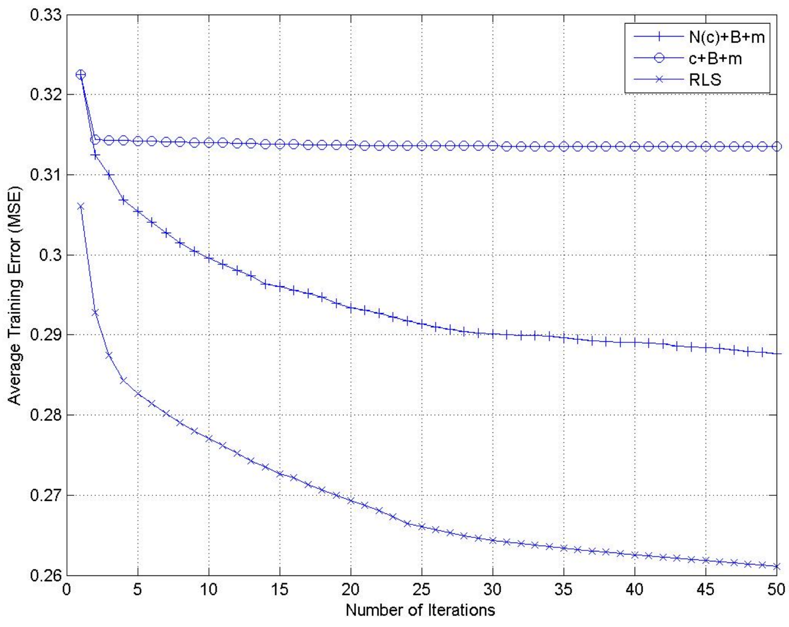
and the expression for Hessian matrix can be re-written as:

$$\begin{aligned} h_c^{'}(u,v) &= h_c(u,v) \\ &+ \sum_{k=1}^{N_h} w_o(i,k)O'_p(k)\big(\beta(k)\big)\left[\left[\sum_{n=1}^{N} b(u)x_p(u)\right]^2\right. \\ &+ 2\big[x_p(u) - m_k(u)\big]\left[\sum_{n=1}^{N} b(u)x_p(u)\right] + \left[\sum_{n=1}^{N} b(v)x_p(v)\right]^2 \\ &\left. + 2\big[x_p(v) - m_k(v)\big]\left[\sum_{n=1}^{N} b(v)x_p(v)\right]\right] \end{aligned} \tag{(3. 10)}$$

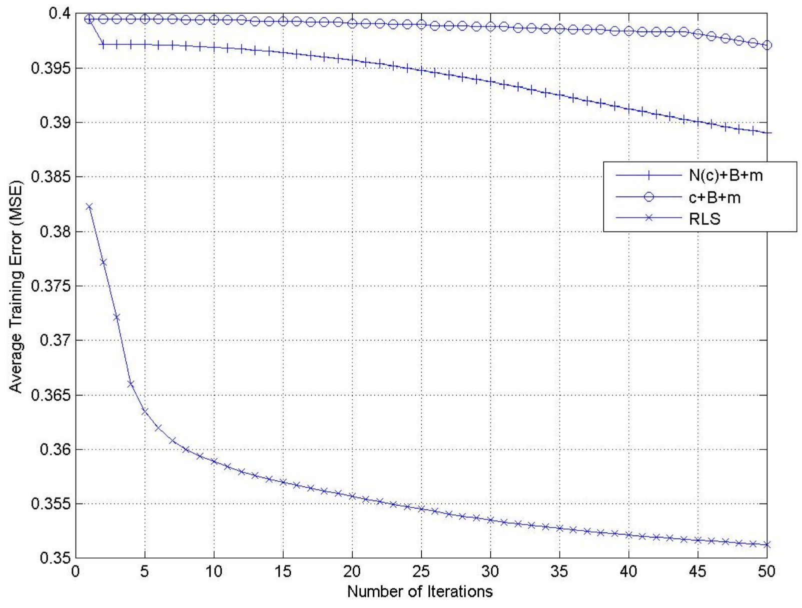
Comparing (3.4) and (3.10) we see that some additional terms appear as sum of products within the square brackets in the expressions for gradient and Hessian in the presence of linearly dependent input. Clearly, these cross terms will cause the training of RBF training algorithm different for the case if linear dependent inputs are added thereby not forcing $\boldsymbol{H}_c^{'}$ to be singular. Thereby from (3.12) we see that the Hessian $\boldsymbol{H}_c^{'}$ simply gains first and second degree terms unlike traditional Newton's method, where a linearly dependent input cause the Hessian matrix to be singular.

3.5 Experimental analyses and verification

We take *twod*, *concrete* and *mattrn* datasets (see Appendix B for details) to study the performance of OWDM-RBF (denoted by {N(c)+B+m}) and compare its performance with the OWO-RBF training algorithm (denoted by {c+B+m}) where no parameter (weighted DM, spread parameters and mean vector parameters) have been optimized and RLS-RBF training algorithm discussed in [25]. We analyzed various training algorithms on two bases, firstly the rate of convergence of error and secondly, its response to the Gaussian random noise which is added into the input data set.

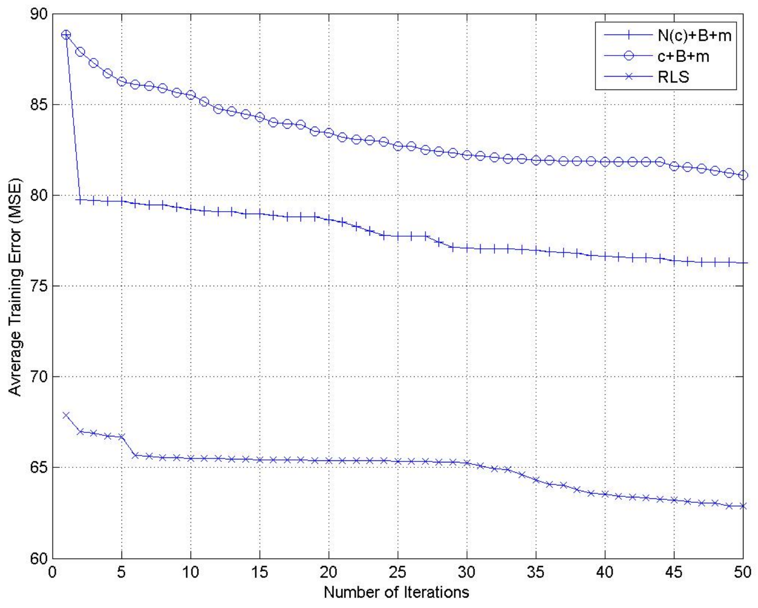


(a)

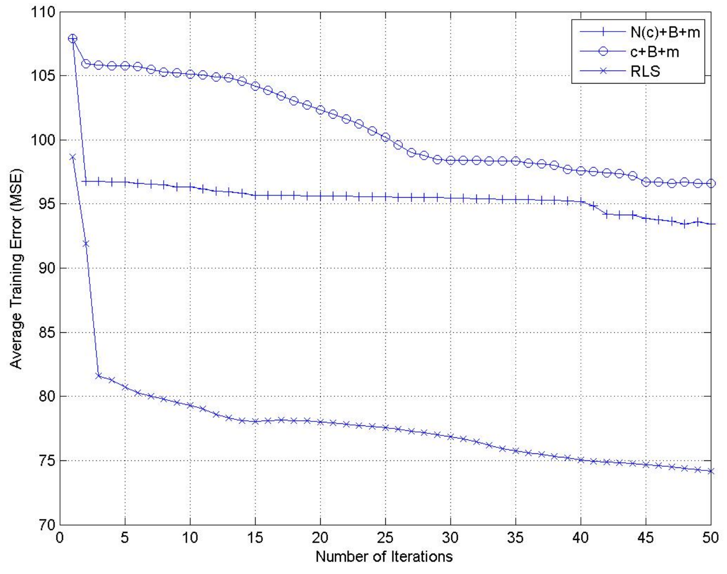


(b)

Figure 3.1 Performance of OWDM-RBF training algorithm with OWO-RBF and RLS-RBF training algorithms for *twod* dataset (a) normal condition (b) noisy input condition

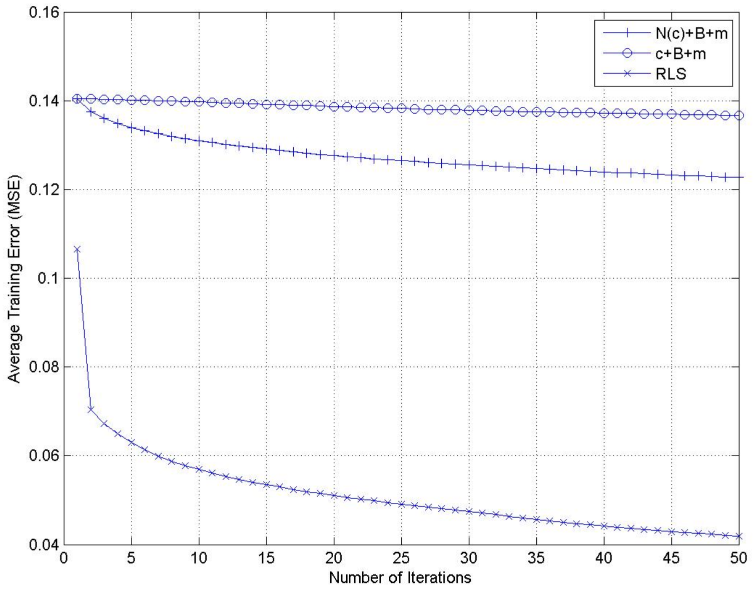


(a)

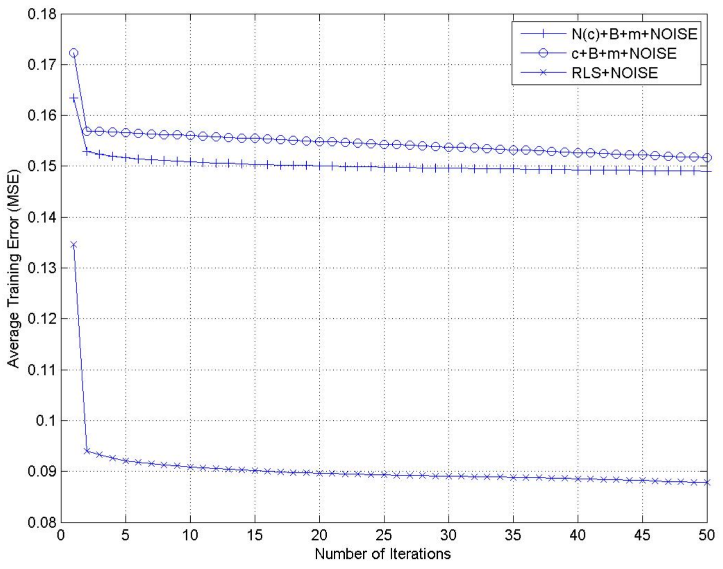


(b)

Figure 3.2 Performance of OWDM-RBF training algorithm with OWO-RBF and RLS-RBF training algorithms for *concrete* dataset (a) normal condition (b) noisy input condition



(a)



(b)

Figure 3.3 Performance of OWDM-RBF training algorithm with OWO-RBF and RLS-RBF training algorithms for *mattrn* dataset (a) normal condition (b) noisy input condition

We make the following observations from the figures:

1) The OWDM-RBF training algorithm and OWO-RBF training algorithm starts at the same error value due to same OWO step. The rate of decrease of error is prominent for the OWDM-RBF training algorithm indicating its efficiency. The RLS-RBF training algorithm not only starts at the lower error value but also decrease at much faster rate.
2) The difference in the error values between the training algorithm and its equivalent noise training algorithm indicates the efficiency and sensitivity of the algorithm towards Gaussian random noise.
3) The error curve for OWDM-RBF training algorithm and the OWO-RBF training algorithm in the noisy data indicates there poor performance for the noisy data. Even the rate of decrement of error is severely affected by the noisy data.

We therefore conclude that the RLS-RBF training algorithm has the lowest error rate and least susceptibility to noise. The OWDM-RBF training algorithm is a primitive training algorithm paving way for further improved training algorithms based on similar optimization scheme which will be discussed in subsequent chapters.

CHAPTER 4

OPTIMIZATION OF SPREAD PARAMETERS ALONG WITH WEIGHTED DM

After the optimization of the weighted DM, we now describe the optimization of the spread parameters vector and investigate its effect on the training of RBF neural network. Mathematical background for optimizing the spread parameters, effect of linearly dependent input units and experiment verification is the discussion area of this chapter.

4.1 Theoretical background

In Chapter 2 we have initialized the spread parameters and defined it as the inverse of the standard deviations of input units. In a typical Gaussian function, the standard deviation controls the width of the Gaussian *"Mexican hat shaped*" curve. Geometrically, in a 3-D space it can be visualized as the parameter which defines the area of the *"Mexican hat"* to be included in the input space.

4.2 Mathematical treatment

In changing the *"width"* (spread parameters) and keeping mean vector parameters (kernel vector) fixed, we use the same error function as in (2.9). For the *p*th pattern, the *k*th hidden unit output is:

$$\hat{O}_p(k) = e^{(-(\beta(k)+d_\beta(k))\cdot \widehat{net}_p(k)} \tag{4. 1}$$

where $\widehat{net}_p(k)$ is defined as in (2.11) and $d_\beta(k)$ is the learning factor for $\boldsymbol{\beta}$ in each k^{th} hidden unit. The output vector $\hat{y}_p(i)$ is same as described in (2.14).

We calculate the gradient for $\boldsymbol{\beta}$ as:

$$g_\beta(k) = -\frac{\partial E}{\partial d_\beta(k)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \hat{y}_p(i)] \cdot \frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)} \tag{4. 2}$$

Following a similar procedure to that for the weighted DM, we get,

$$\frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)} = -w_o(i,k) \cdot \hat{O}_p(k) \sum_{n=1}^{N+1} c(n) \cdot \left[x_p(n) - m_k(n)\right]^2 \tag{4. 3}$$

Combining (4.3) and (4.4) we obtain $\boldsymbol{g_\beta}$. Calculating the Hessian matrix element we get,

$$h_\beta(u,v) = \frac{\partial^2 E}{\partial d_\beta(u) \partial d_\beta(v)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\left[\sum_{i=1}^{M}\frac{\partial \hat{y}_p(i)}{\partial d_\beta(u)} \cdot \frac{\partial \hat{y}_p(i)}{\partial d_\beta(v)}\right] \tag{4. 4}$$

We now use the realization of the following linear equation:

$$\boldsymbol{H_\beta} \cdot \boldsymbol{d_\beta} = \boldsymbol{g_\beta} \tag{4. 5}$$

This looks similar to the linear equations as in (3.7). Solving (4.5) via OLS, we get the optimal $\boldsymbol{d_\beta}$**.** For the k^{th} hidden unit**,** $\boldsymbol{\beta}$ is updated according to:

$$\beta(k) \leftarrow \beta(k) + d_\beta(k) \tag{4. 6}$$

4.3 Algorithm for optimizing spread parameters

We now describe the formal algorithm for optimized spread parameter RBF (OSP-RBF) training algorithm. After normalizing the input patterns to zero mean and unit variance and initializing different parameters, for each iteration of the training algorithm, the steps are as follows:

(1) Calculate gradient for $\boldsymbol{\beta}$ from (4.2) and Hessian matrix elements from (4.4).

(2) Solve the linear equation in (4.5) via OLS.

(3) Update spread parameters vector using (4.6).

(4) Solve linear equations for all the output weights using OWO.

The updated spread parameters vector are fed back to calculate $\boldsymbol{\beta}$ for next iteration in our proposed training algorithm contrary to [25]. This completes our description for the OSP-RBF training algorithm. It should be noted here that in case of combined optimization of spread parameter and weighted DM (OSPWDM-RBF) training algorithm, each parameter's optimization is followed by an OWO step.

4.4 Effect of linear dependence in the input layer

A linearly dependent input can be modeled as in (3.9). During the spread parameter adaption, the expression for gradient given by (4.2) can be rewritten as:

$$g_\beta(k) = -\frac{\partial E}{\partial d_\beta(k)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - \hat{y}_p(i) - w_{oi}(i,N+1)\sum_{n=1}^{N} b(n)x_p(n)\right]\cdot\frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)} \qquad (4.\ 7)$$

where

$$\frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)} = -w_o(i,k)\cdot O_p^{'}(k)\sum_{n=1}^{N} c(n)\cdot\left[x_p(n) - m_k(n)\right]^2 + c(N+1) \cdot\left[\sum_{n=1}^{N} b(n)x_p(n) - m_k(N+1)\right]^2 \qquad (4.\ 8)$$

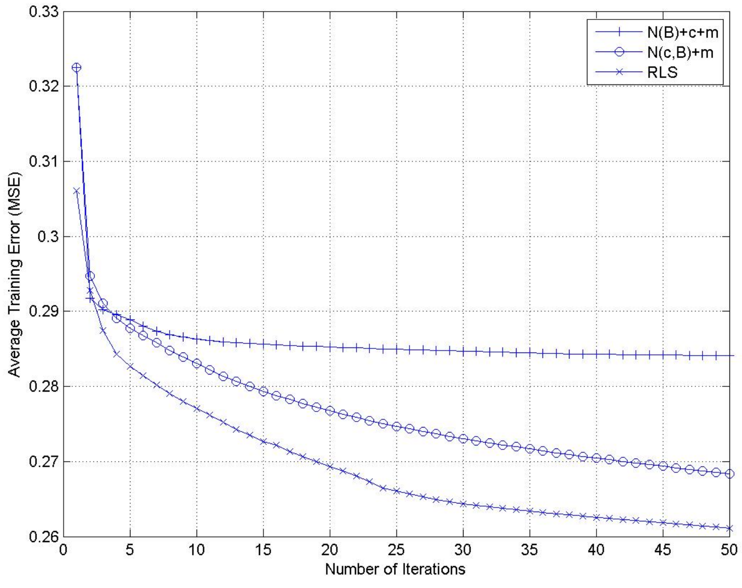
The hidden unit output $\hat{O}_p(k)$ in $\frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)}$ gets additional terms as in (3.11). Hessian matrix element will be:

$$h_{\beta}^{'}(u,v) = \frac{2}{N_v}\sum_{p=1}^{N_v}\left[\sum_{i=1}^{M}\left[-w_o(i,u)O_p^{'}(u)\sum_{n=1}^{N}c(n)\left[x_p(n)-m_u(n)\right]^2 + c(N+1)\left[\sum_{n=1}^{N+1}b(n)x_p(n) - m_u(N+1)\right]^2\right]\left[-w_o(i,k)O_p^{'}(v)\sum_{n=1}^{N}c(n)\left[x_p(n)-m_v(n)\right]^2 + c(N+1)\left[\sum_{n=1}^{N+1}b(n)x_p(n) - m_v(N+1)\right]^2\right]\right] \tag{4. 9}$$

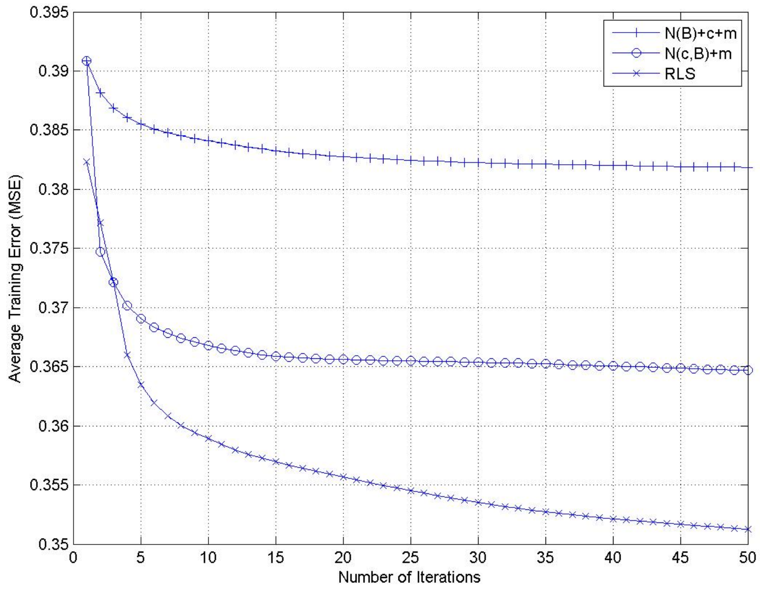
Similar to (3.12), comparing (4.4) with (4.9), we see that some additional terms appear as sum of products within the square brackets in the expressions for gradient and Hessian in the presence of linearly dependent inputs. Clearly, these cross terms will cause the training of the RBF training algorithm to be different for the case, if linear dependent inputs are added thereby not forcing $\boldsymbol{H}_{\boldsymbol{c}}^{'}$to be singular. Thereby from (4.9) we see that the Hessian $\boldsymbol{H}_{\boldsymbol{c}}^{'}$ simply gains the first and the second degree terms unlike traditional Newton's method, where a linearly dependent input cause the Hessian matrix to be singular.

4.5 Experimental analyses and verification

We take *twod*, *concrete* and *mattrn* datasets (see Appendix B for details) to study the performance of OSP-RBF training algorithm (denoted by {N(B)+c+m}) and compare its performance with the OSPWDM-RBF training algorithm (denoted by {N(c,B)+m}) and RLS-RBF training algorithm discussed in [25]. The performance of these training algorithms has also been seen with random Gaussian noise data added to the input units.

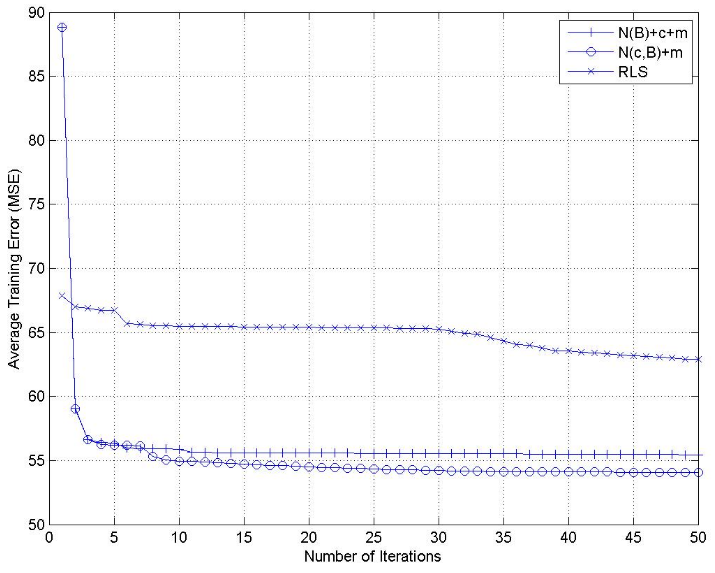


(a)

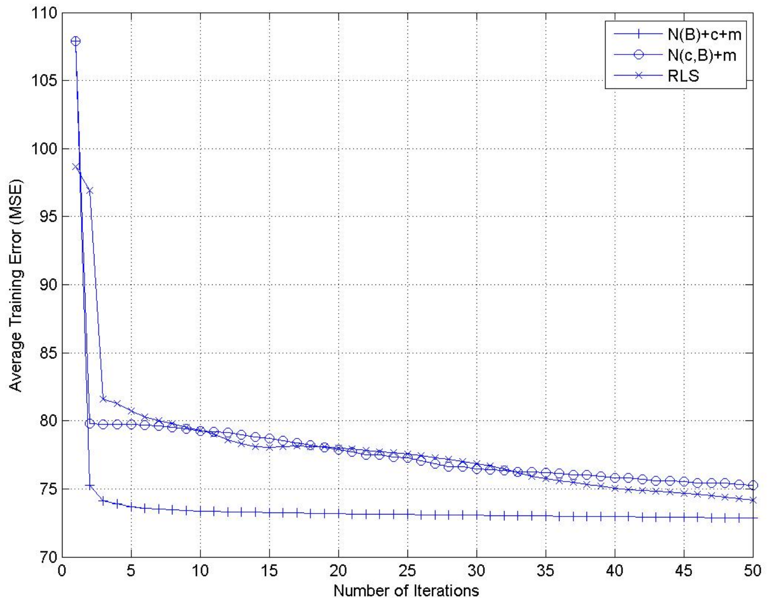


(b)

Figure 4.1 Performance of OSP-RBF and OSPWDM-RBF training algorithms with RLS-RBF training algorithm on *twod* dataset (a) normal condition (b) noisy input condition

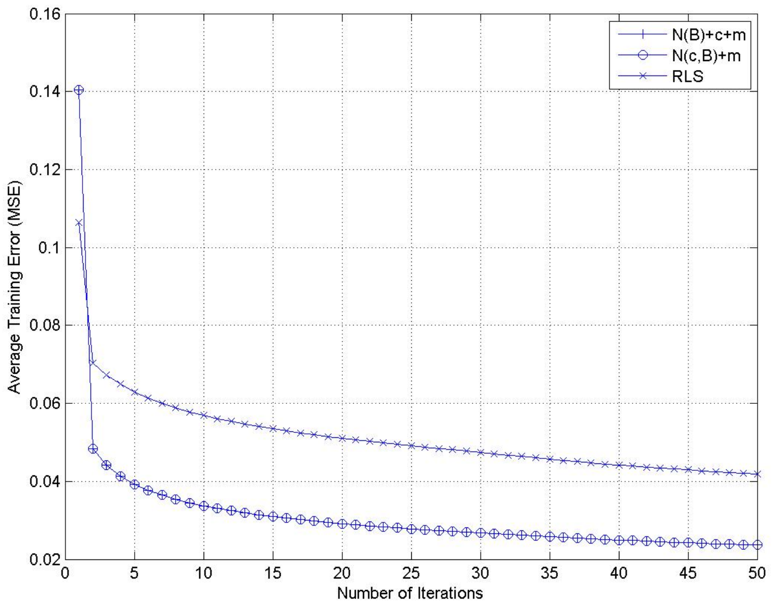


(a)

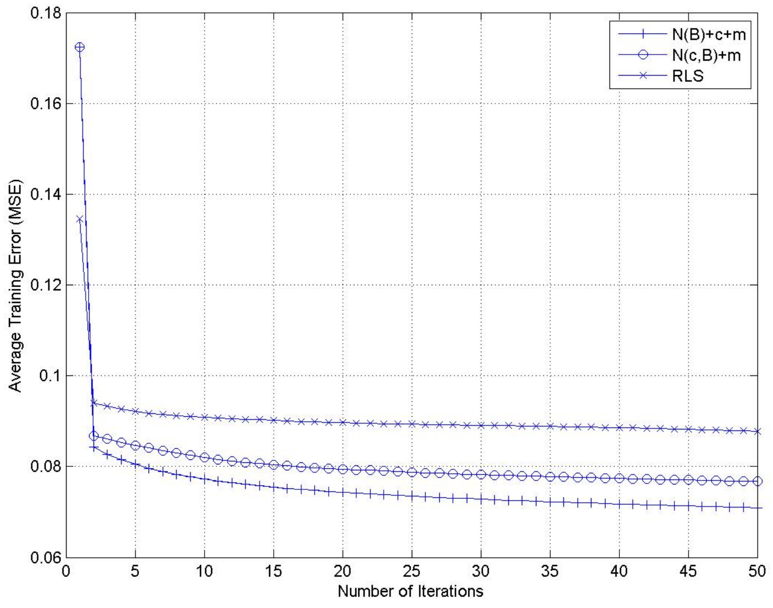


(b)

Figure 4.2 Performance of OSP-RBF and OSPWDM-RBF training algorithms with RLS-RBF training algorithm on *concrete* dataset (a) normal condition (b) noisy input condition



(a)



(b)

Figure 4.3 Performance of OSP-RBF and OSPWDM-RBF training algorithms with RLS-RBF training algorithm on *mattrn* dataset (a) normal condition (b) noisy input condition

Referring to the figures, we observe that although OSP-RBF training algorithm has high error curve for Gaussian noise data, including the OSPWDM-RBF training algorithm improves the performance significantly when compared to the algorithm where only weighted DM is optimized as shown in the plots. We have not included the OWO-RBF training algorithm owing to its poor performance. Following observation are made from the error plots for three datasets:

(1) The error curve for the OSP-RBF training algorithm is comparable to the RLS-RBF error curve. Hence there is an improvement over the earlier OWDM-RBF training algorithm discussed in Chapter 3. One point to be noted here is that the OSPWDM-RBF training algorithm also gives a similar error curve as the other training algorithm but its sensitivity to the Gaussian noise data signifies its usefulness over the algorithm where only the spread parameter (OSP-RBF, denoted by {N(B)+c+m}) has been optimized.

(2) The error decrement rate for all training algorithms is now comparable to the RLS-RBF training algorithm. Although the error rate decrement is significantly high for the OSP-RBF training algorithm on Gaussian noise data but it settles at a much higher value after 50 iterations.

We conclude that optimizing spread parameter is indeed an improvement over the previous training algorithm discussed in Chapter 3. The combined optimization of spread parameter and weighted DM training algorithm does offer a significantly low error rate and less sensitive to noise with a comparable performance with RLS-RBF training algorithm. It will be logical extension to extend the idea of optimization on the mean vector parameters and experiment with its various combinations with other two parameters which will be the subject matter of the next chapter.

CHAPTER 5

OPTIMZATION OF MEAN VECTOR PARAMETERS ALONG WITH WEIGHTED DM AND SPREAD PARAMETERS

In this chapter, we describe the optimization of the mean vector parameters and investigate its effect on the training of RBF neural network. Mathematical background for optimizing mean vector parameters, single and multiple optimal learning factors, effect of linearly depended input units and experiment verification is the discussion area of this chapter.

5.1 Theoretical background

From a regression point of view, mean vector defines the position of the Gaussian “Mexican hat” shape in the input space. It is therefore important to optimize the position to obtain better reconstruction of the input surface. It will be a matter of experimentation as to how the optimization or “tuning” of the mean vector parameters alone performs and in comparison to the other two parameter vectors

5.2 Mathematical treatment

Following the same approach as in previous sections, we obtain a “tuning “procedure for changing mean vector parameters. For *p*th input pattern and *k*th hidden unit, including mean vector learning factor ***z***, $\widehat{net}_p(k)$ is:

$$\widehat{net}_p(k) = \sum_{n=1}^{N} c(n) \cdot [x_p(n) - (m_k(n) + z_k \cdot g_{mean}(k,n))]^2) \qquad (5.1)$$

The output vector $\hat{y}_p(i)$ is as described in (2.14). We solve for a vectors of learning factors, one per hidden unit.

Unlike the previous gradients, we have a gradient matrix as:

$$g_{mean}(k,n) = -\frac{\partial E}{\partial m_k(n)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \hat{y}_p(i)] \cdot \frac{\partial \hat{y}_p(i)}{\partial m_k(n)} \tag{5. 2}$$

Applying the chain rule we get:

$$\frac{\partial \hat{y}_p(i)}{\partial m_k(n)} = 2 \cdot w_o(i,k) \cdot \hat{O}_p(k) \cdot \beta(k) \cdot c(n) \cdot \left[x_p(n) - m_k(n)\right] \tag{5. 3}$$

Combining (5.2) with (5.3) we get the gradient matrix elements$g_{mean}(k,n)$. We now provide the mathematical framework necessary for the derivation of single and multiple optimal learning factors.

5.3 Learning factor

During our investigation we compared our algorithms with RBF training algorithms using a single OLF which is a one variable form of Newton's method. Here we discuss the single OLF case for updating mean vector parameters. In the single OLF case, the calculation of partial derivatives requires one pass through the data.

5.3.1 Single OLF for optimizing mean vector parameters

Given the error function as defined in (2.9) we have:

$$\hat{y}_p(i) = \sum_{k=1}^{N_h} w_o(i,k) \cdot e^{-\beta(k)\widehat{net}_p(k)} + \sum_{n=1}^{N+1} w_{oi}(i,n) \cdot x_p(n) \tag{5. 4}$$

where $\widehat{net}_p(k)$ is now modified with single OLF as:

$$\widehat{net}_p(k) = \sum_{n=1}^{N} c(n) \cdot [x_p(n) - (m_k(n) + z \cdot g_{mean}(k,n))]^2) \tag{5. 5}$$

Notice the difference in (5.5) and (5.1). Instead of using a multiple OLF vector, $\boldsymbol{z}$, we optimize each hidden unit by using a single OLF z. The first partial derivative of E with respect to z is:

$$g_z = -\frac{\partial E}{\partial z} = -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \hat{y}_p(i)] \cdot \frac{\partial \hat{y}_p(i)}{\partial z} \tag{5. 6}$$

where,

$$\frac{\partial \hat{y}_p(i)}{\partial z} = 2\sum_{k=1}^{N_\mathrm{h}} w_o(i,k) \cdot \hat{O}_p(k) \cdot \beta(k) \sum_{n=1}^{N} c(n)\big[x_p(n) - m_k(n)\big] \cdot g_{mean}(k,n) \tag{5. 7}$$

The Gauss-Newton approximation of the second partial is:

$$\frac{\partial^2 E}{\partial z^2} \approx \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left(\frac{\partial \hat{y}_p(i)}{\partial z}\right)^2 \tag{5. 8}$$

Now from 2nd order Taylor series we get:

$$E(z) = E(0) + z \cdot \frac{\partial E}{\partial z} + \frac{1}{2} \cdot \frac{\partial^2 E}{\partial z^2} \cdot z^2 \tag{5. 9}$$

Hence substituting $\partial E(z)/\partial z = 0$ we get:

$$z = -\frac{\partial E/\partial z}{\partial^2 E/\partial z^2} \tag{5.10}$$

In [48], better performance of multiple optimal learning factor (MOLF) on OWO based backpropagation network is reported. We therefore update the mean vector parameters based on MOLF which is optimal for individual hidden unit.

5.3.2 Multiple optimal learning factors

After the output weights in the RBF training algorithm are trained using OWO, instead of using a single OLF for updating all the parameters, we use Newton's method to estimate a vector of optimal learning factors as MOLF. The basic idea for MOLF is that while updating the mean vector parameters for each iteration, instead of using a single OLF z we use a vector $\boldsymbol{z}$ of dimensionN_h called MOLF. One of our investigation aims is to show this novel method to update the parameters to be better than the single optimal learning case.

We now derive the expression for the MOLF as used in updating mean vector parameters. Considering the error function as in (2.9), we have:

$$g_z(k) = -\frac{\partial E}{\partial z(k)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - \hat{y}_p(i)] \cdot \frac{\partial \hat{y}_p(i)}{\partial z(k)} \tag{5.11}$$

where:

$$\frac{\partial \hat{y}_p(i)}{\partial z(k)} = -2 \cdot w_o(i,k) \cdot \hat{O}_p(k) \cdot \beta(k)\left[\sum_{n=1}^{N+1} c(n) \cdot \left[x_p(n) - m_k(n)\right] \cdot g_{mean}(k,n)\right] \tag{5.12}$$

Combining (5.11) and (5.12) we get the optimum value of learning vector $g_z(k)$.Using Gauss-Newton's updates, the second partial derivative element of the Hessian $\boldsymbol{H_z}$ are:

$$h_z(u,v) = \frac{\partial^2 E}{\partial z(u)\partial z(v)} \approx \frac{2}{N_v}\sum_{p=1}^{N_v}\left[\sum_{i=1}^{M}\frac{\partial \hat{y}_p(i)}{\partial z(u)} \cdot \frac{\partial \hat{y}_p(i)}{\partial z(v)}\right] \tag{5.13}$$

where,

$$\frac{\partial \hat{y}_p(i)}{\partial z(u)} = w_o(i,u) \cdot \frac{\partial \hat{O}_p(u)}{\partial z(u)} \tag{5.14}$$

$$\frac{\partial \hat{y}_p(i)}{\partial z(v)} = w_o(i,v) \cdot \frac{\partial \hat{O}_p(v)}{\partial z(v)} \tag{5.15}$$

The Gauss-Newton's update guarantees that $\boldsymbol{H_z}$ is non-negative definite. Given the negative gradient vector $\boldsymbol{g_z} = \left[-\frac{\partial E}{\partial z_1}, -\frac{\partial E}{\partial z_2} \ldots, -\frac{\partial E}{\partial z_{N_h}}\right]^T$ and the Hessian $\boldsymbol{H_z}$, we minimize E with respect to the vector MOLF $\boldsymbol{z}$ using following linear equations:

$$\boldsymbol{H_z} \cdot \boldsymbol{z} = \boldsymbol{g_z} \tag{5.16}$$

Thus we get:

$$\boldsymbol{z} = \boldsymbol{H}_{\boldsymbol{z}}^{-\mathbf{1}} \cdot \boldsymbol{g}_{\boldsymbol{z}} \tag{5.17}$$

During each iteration, for kth hidden unit and nth input, mean vector parameter is updated as:

$$m_k(n) \leftarrow m_k(n) + z_k \cdot g_{mean}(k, n) \tag{5.18}$$

5.4 Algorithm for optimizing mean vector parameters

After normalizing the input patterns to zero mean and unit variance and initializing different parameters, we describe a formal algorithm for various RBF training algorithms. For each iteration of the training algorithm, the steps are as follows:

(1) For optimized mean vector parameter RBF training algorithm with SOLF (SOMV-RBF), calculate the gradient from (5.6). For optimized mean vector parameters RBF (OMV-RBF) training algorithm with MOLF, calculate gradient from (5.11) and Hessian matrix elements from (5.13).

(2) For SOMV-RBF, update mean vector parameters using (5.18) but instead of using vector $\boldsymbol{z_k}$, we use the SOLF z from (5.10). For OMV-RBF, solve the linear equation in (5.16) via OLS and update mean vector parameters using (5.18).

(3) Solve linear equations for all the output weights using OWO.

It should be noted here that each parameter's optimization is followed by an OWO step in case of combined optimization of spread parameter and mean vector parameters (OSPMV-RBF) training algorithm, combined optimization of mean vector parameters and weighted DM coefficients (OMVWDM-RBF) training algorithm and training algorithm which includes the combined optimization of all parameters (OAP-RBF).

Contrary to [25], the updated mean vector parameters are fed back to calculate $\boldsymbol{m_k}$ for next iteration. This completes our description for the SOLF and MOLF based RBF training algorithms. It should be pointed out here that in case the multiple OLF being unable to train the RBF training algorithm we collapse it into a single OLF.

5.5 Effect of linear dependence in the input layer

A linearly dependent input can be modeled as in (3.9). During the mean vector parameters optimization, the expression for gradient for MOLF can be rewritten as:

$$g_z(k) = -\frac{\partial E}{\partial z(k)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\left[t_p(i) - \hat{y}_p(i) - w_{oi}(i, N+2)\sum_{n=1}^{N} b(n)x_p(n)\right]\cdot\frac{\partial \hat{y}_p(i)}{\partial z(k)} \tag{5.20}$$

where,

$$\frac{\partial \hat{y}_p(i)}{\partial z(k)} = -2w_o(i,k)O_p'(k)\beta(k)\left[\sum_{n=1}^{N} c(n)\left[x_p(n) - m_k(n)\right]g_{mean}(k,n) + c(N+1)\left[\sum_{n=1}^{N+1} b(n)x_p(n) - m_k(N+1)\right]g_{mean}(k,N+1)\right] \tag{5.21}$$

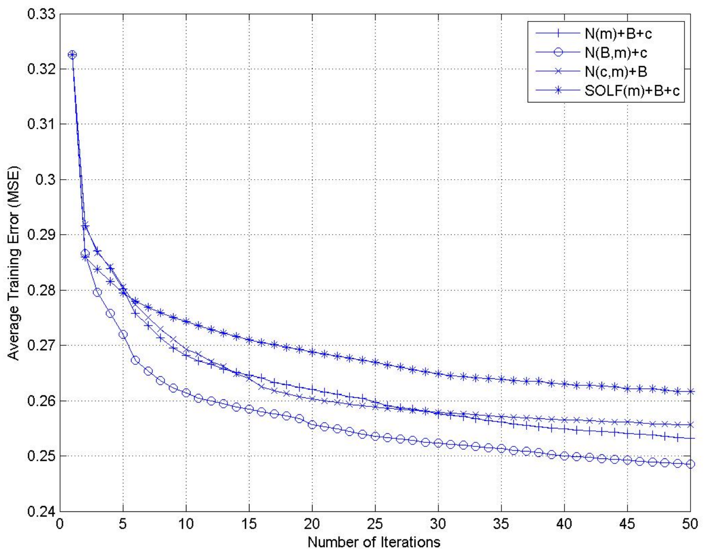
The hidden unit output $\hat{O}_p(k)$ in $\frac{\partial \hat{y}_p(i)}{\partial d_\beta(k)}$ gets additional terms as $O_p'(k)$ given in (3.11). The expression for Hessian matrix element can be re-written as:

$$h_c'(u,v) = \frac{2}{N_v}\sum_{p=1}^{N_v}\left[\sum_{i=1}^{M}\left[2w_o(i,u)O_p'(u)\beta(u)\left[\sum_{n=1}^{N} c(n)\left[x_p(n) - m_u(n)\right]g_{mean}(u,n) + c(N+1)\left[\sum_{n=1}^{N} b(n)x_p(n) - m_u(N+1)\right]g_{mean}(u,N+1)\right]\right]\left[2w_o(i,v)O_p'(v)\beta(v)\left[\sum_{n=1}^{N} c(n)\left[x_p(n) - m_v(n)\right]g_{mean}(v,n) + c(N+1)\left[\sum_{n=1}^{N} b(n)x_p(n) - m_v(N+1)\right]g_{mean}(v,N+1)\right]\right]\right] \tag{5.22}$$

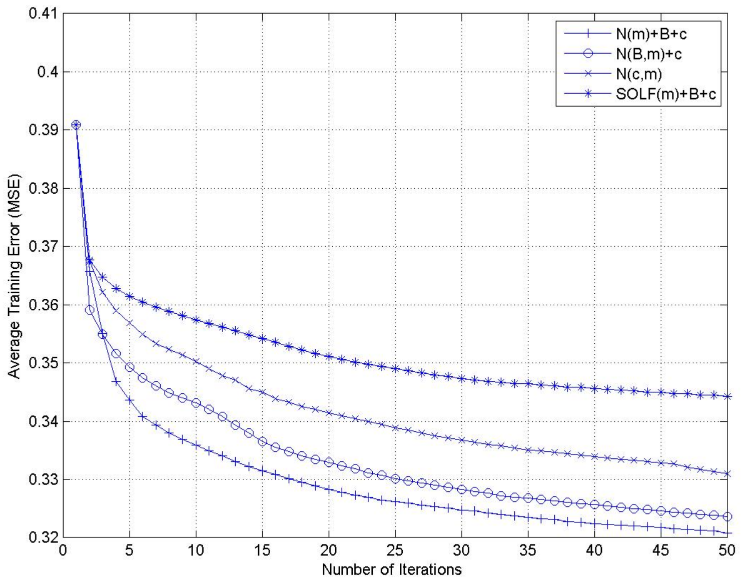
Comparing (5.14) and (5.21) we see that some additional terms appear as sum of products within the square brackets in the expressions for gradient and hessian matrices thereby not forcing $\boldsymbol{H}_z'$to be singular. Therefore from (5.21) we see that the Hessian $\boldsymbol{H}_z'$ simply gains first and second degree terms unlike traditional Newton's method, where a linearly dependent input cause the Hessian matrix to be singular.

5.6 Experimental analyses and verification

We take *twod*, *concrete* and *mattrn* datasets to study the performance of SOMV-RBF (denoted by SOLF(m)+c+B), OMV-RBF (denoted by N(m)+B+m), OSPMV-RBF (denoted by N(B,m)+c and OMVWDM-RBF (denoted by N(c,m)+B). The performances of these training algorithms have also been shown with Gaussian noise data added to the input units

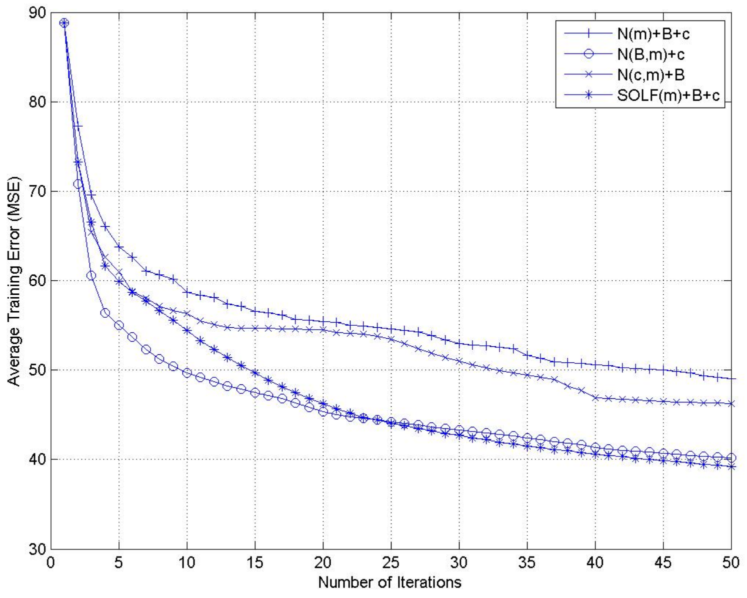


(a)

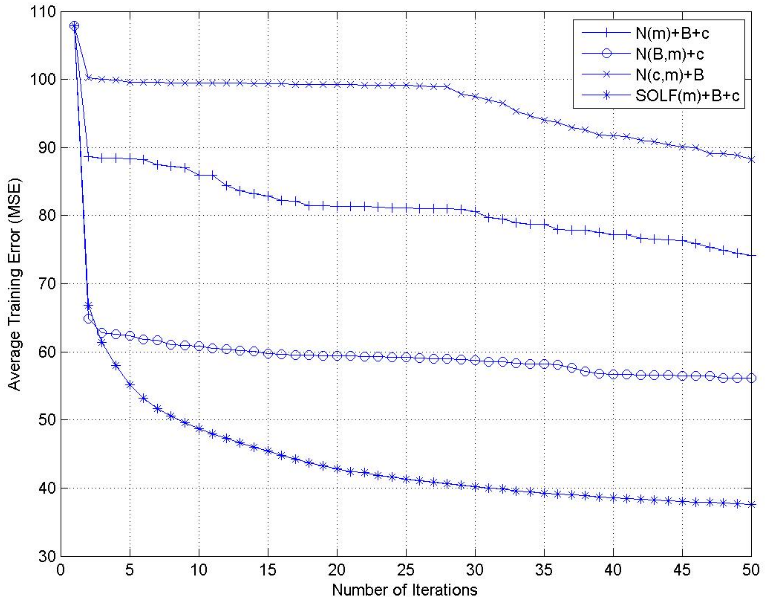


(b)

Figure 5.1 Performance of single OLF training algorithm with other multiple OLF based training algorithms on *twod* data set (a) normal condition (b) noisy input condition

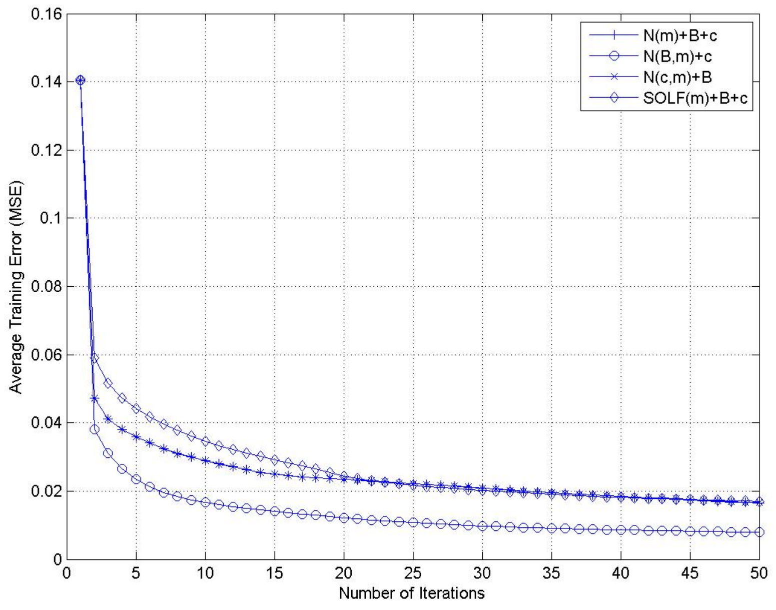


(a)

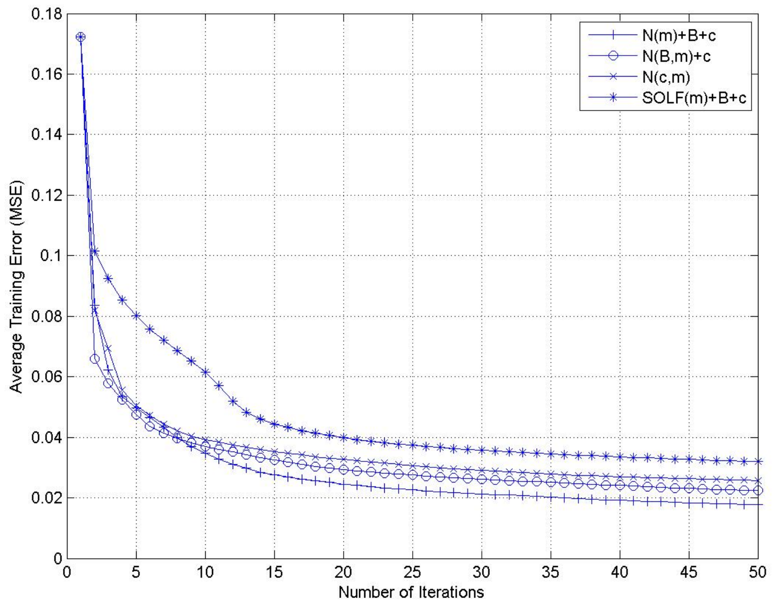


(b)

Figure 5.2 Performance of single OLF training algorithm with other multiple OLF based training algorithms on *concrete* data set (a) normal condition (b) noisy input condition



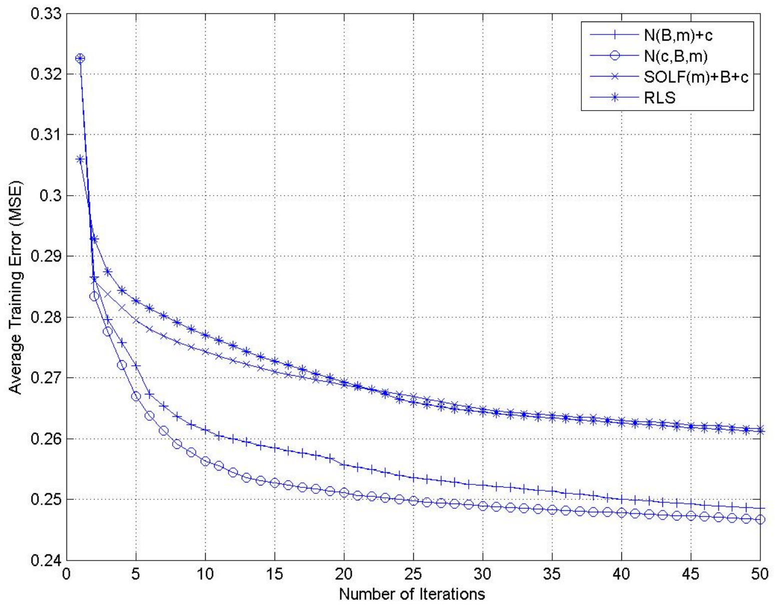
(a)



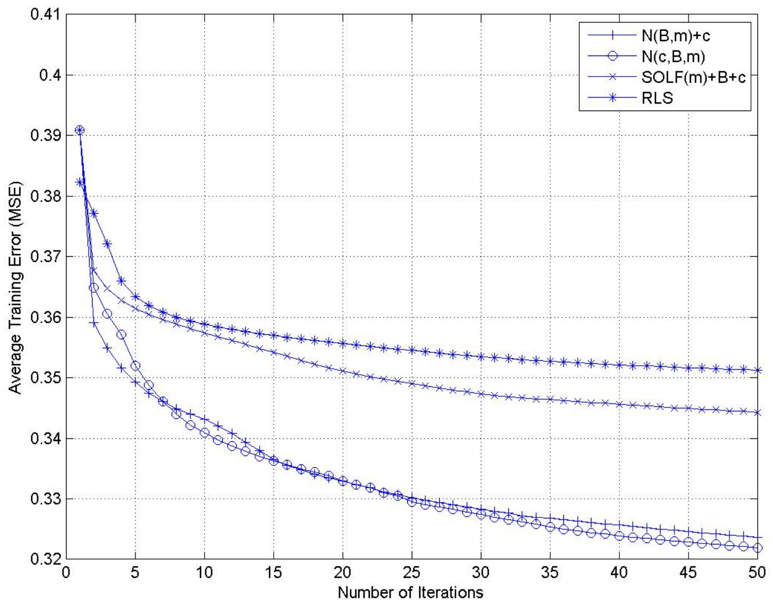
(b)

Figure 5.3 Performance of single OLF training algorithm with other multiple OLF based training algorithms on *mattrn* data set (a) normal condition (b) noisy input condition

We observe that OSPMV-RBF and SOMV-RBF training algorithms perform best hence we now compare its performance with RLS-RBF training algorithm [25] and OAP-RBF training algorithm. The performance of these training algorithms will also be seen with Gaussian noise data.

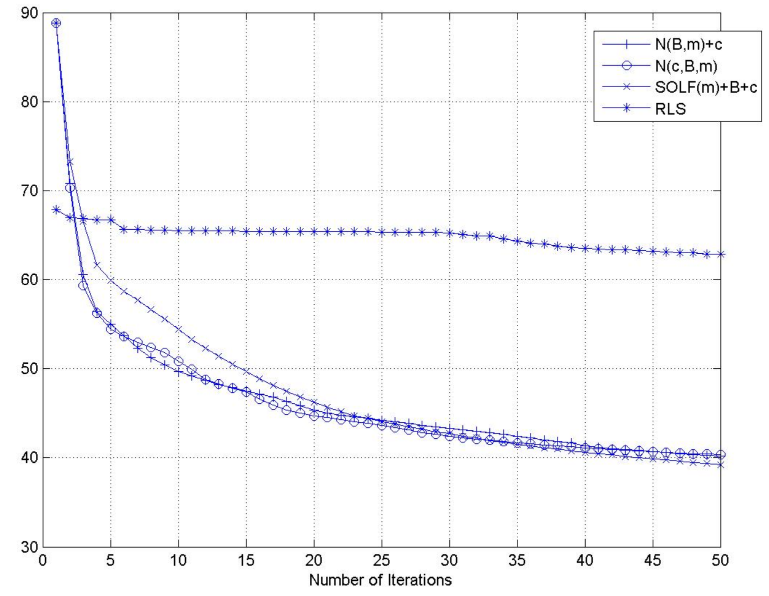


(a)

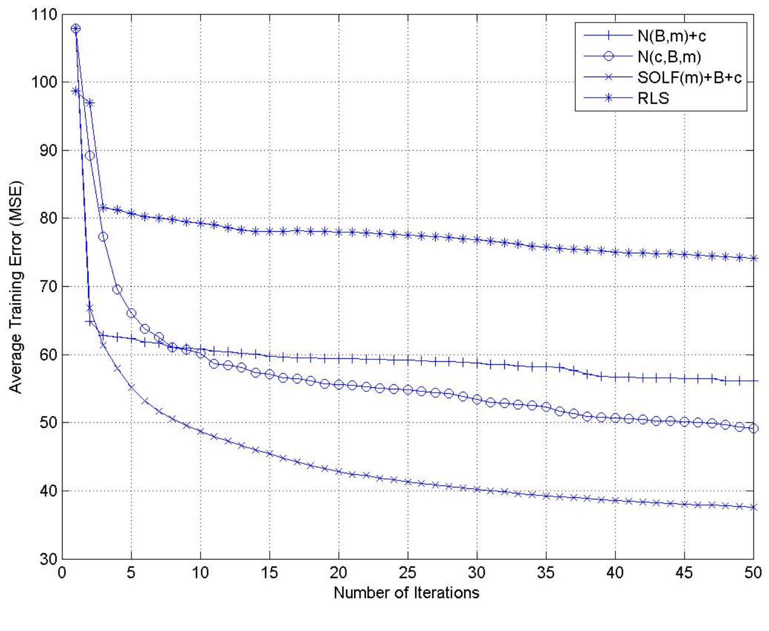


(b)

Figure 5.4 Performance of single and multiple OLF based training algorithms with RLS-RBF training algorithm on *twod* dataset normal condition (b) noisy input condition

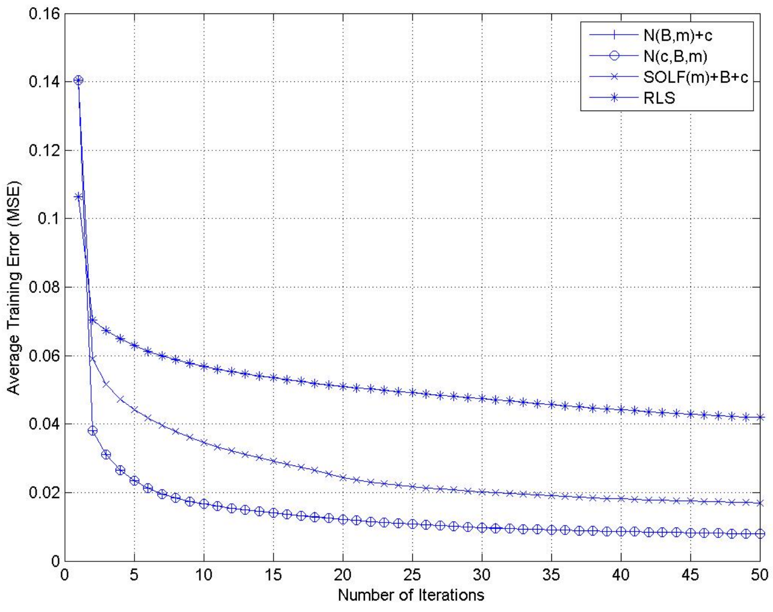


(a)

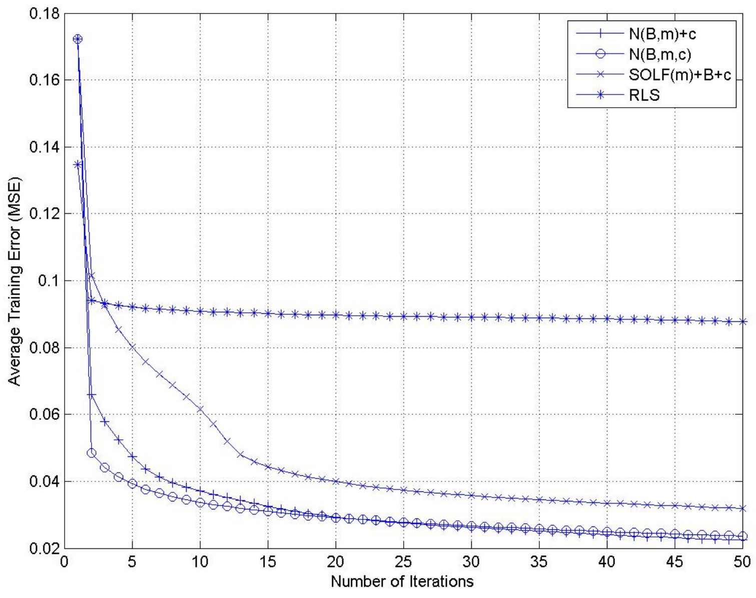


(b)

Figure 5.5 Performance of single and multiple OLF based training algorithms with RLS-RBF training algorithm on *concrete* dataset (a) normal condition (b) noisy input condition



(a)



(b)

Figure 5.6 Performance of single and multiple OLF based training algorithms with RLS-RBF training algorithm on *mattrn* dataset (a) normal condition (b) noisy input condition

We observe that the SOMV-RBF training algorithm is comparable to the RLS-RBF training algorithm and the noise sensitivity is low for both. However since the training algorithm we discuss optimizes mean vector parameters with a single OLF, we see a clear improvement when mean vector parameters optimization is performed with MOLF. For un-correlated datasets, optimizing mean vector parameters through single OLF has no effects. Above figures help us to identify few interesting observations which are as follows:

1) Compared to SOLF, the training algorithms using MOLF perform better.
2) Compared to all other training algorithm discussed, the OSPMV-RBF and OAP-RBF training algorithm not only outperforms the RLS-RBF training algorithm but also shows extremely good performance in presence of Gaussian noise.

This section concludes an important point that mean vector parameters optimization training algorithms perform better than optimizing other two parameters. A striking difference can be observed from figures in previous chapter where we started using the optimization technique on weighted DM training algorithms and spread parameters vector. Not only the training algorithms improved at every step of including all parameters in the optimization process but their responses to the noisy data suggest that they become less sensitive for it. This will be the subject matter of the next chapter.

CHAPER 6

ANALYSES OF VARIOUS RBF TRAINING ALGORITHMS

We investigate the performance of the training algorithms discussed up-till now based on situations described in following subsections. We will analyze their performance using *twod* dataset.

6.1 <u>Effect of using weighted DM on different RBF training algorithms</u>

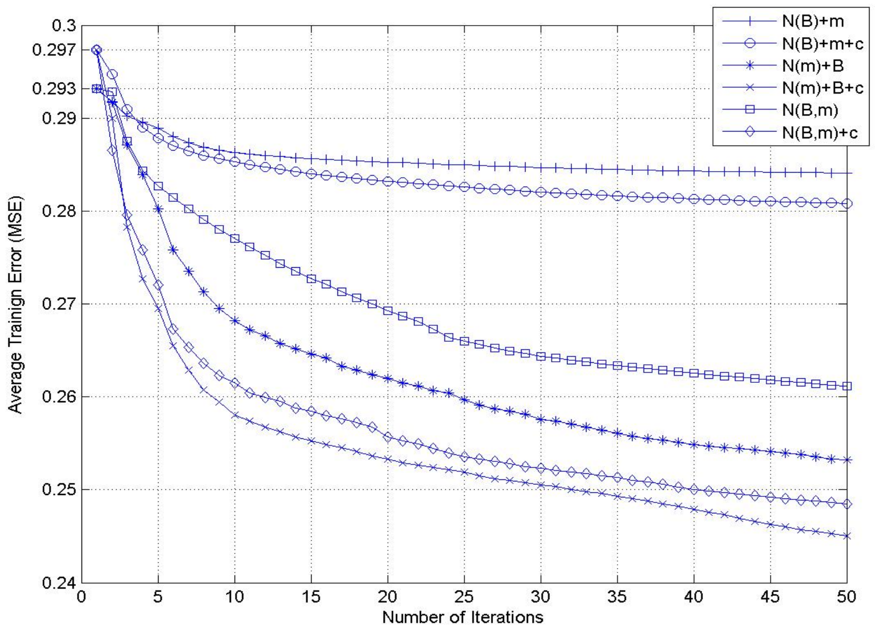


Figure 6.1 Effect of weighted DM on RBF training algorithms for *twod* dataset

Figure 6.1 helps us to identify the effectiveness of the weighted distance measure on the family of propose algorithms. The observations are given below:

1) The training algorithms where we have not applied weighted distance measure (DM), we observe that the training algorithm in which only the mean vector parameters has been changed keeping the spread parameter constant during the training process performs the best. This seems to be logical since once the width has been fixed to a value, then during the training process we optimize position of the mean vector parameters on the local induced region of high activation. By optimizing $\boldsymbol{m_k}$, we essentially place these *"caps"* of fixed width Gaussian function covering the entire weight space. This leads to better generalization.
2) We notice that although the starting point of the starting training error is higher after including weighted DM but the convergence rate is faster than the non-weighted DM curves
3) There is improvement for those training algorithms where weighted DM has been introduced as compared to those discussed in (1). This supports our theoretical explanation of the advantage of using a weighted DM. Except from the plots with only spread parameters optimization, we see that introducing weighted DM significantly improves all the other training algorithms. We also observe that there is a vast improvement in the combined optimization of mean vector parameters and spread parameters training algorithm if we introduce weighted DM. This establishes the fact that using weighted DM is a better choice than normal Euclidean distance.
4) The gradient of the plots tells the convergence rate of the training. We observe that for all the training algorithms, there is a significant error convergence. This suggests that changing only $\boldsymbol{\beta}$ is not a good approach for better training performance.

6.2 Effect of applying Newton's method on different RBF parameters

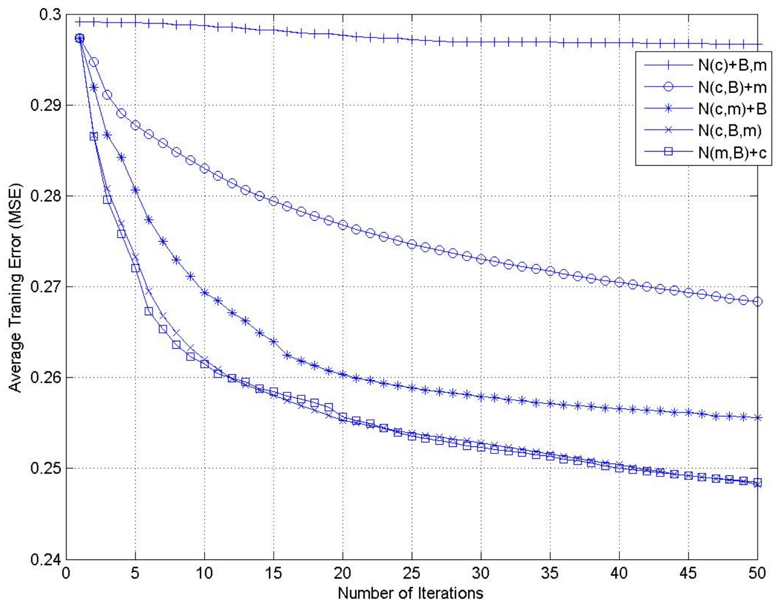


Figure 6.2 Effect of using Newton's method on different parameters for *twod* dataset

Figure 6.2 discusses the Newton's method approach. Following observations are made:

1) Applying Newton's method on the DM alone is not very beneficial. It can be visualized that since we are trying to encompass the entire input space with uneven and un-tuned Gaussian functions whose width (controlled by spread parameter) and position of peak (mean vector) are not fully optimized, it will not be completely covered thus leading to poor generalization.

2) The faster convergence is observed when we apply Newton's method to either mean or spread parameters or to all of them.

3) Not much improvement is seen if we leave kernel vector to optimize via Newton's method. Therefore based on this figure we can say that for a better generalized RBF network,

tuning mean vector parameters is of much greater importance than the spread parameters. This is an important conclusion made.

6.3 Effect of applying Newton's method on weighted DM on different RBF training algorithms

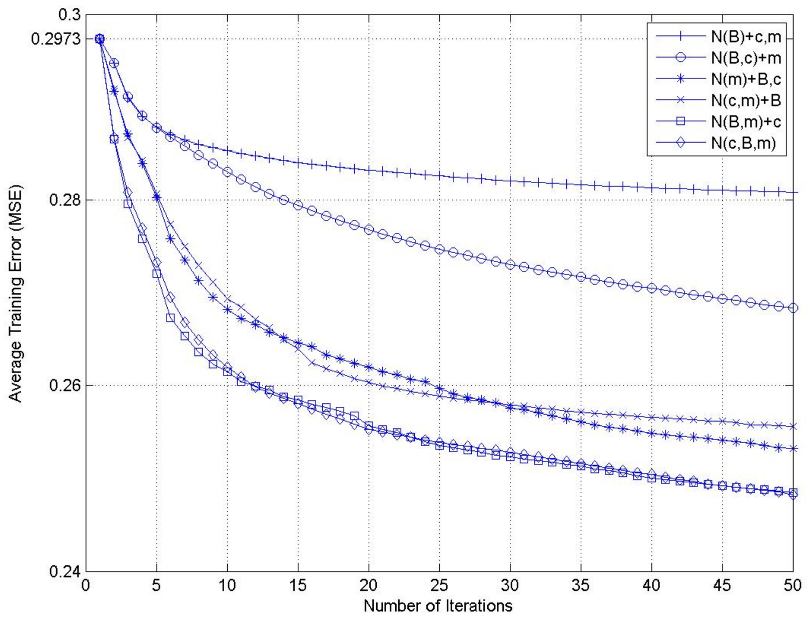


Figure 6.3 Effect of Newton's method on weighted DM on various RBF training algorithms for *twod* dataset

Figure 6.3 discusses the effect of Newton's method on weighted DM in various RBF training algorithms. After we propose a weighted DM in the conventional RBF training algorithms, one approach we try is to see the effect of combining it with the conventional network parameters and other is to apply Newton's method to optimize it.

Figure 6.3 reveals several important facts about the effect of applying Newton's method to distance measure.

1) By combining the optimization of spread parameter and weighted DM, we see a significant improvement.

2) Although optimizing mean vector parameters still governs the overall performance of the networks, we observe from plot 4 that applying Newton's method on weighted DM gives smooth convergence behavior

3) Once the Newton's method has been applied to spread parameters, not much difference is observed in the error curve.

We conclude that optimizing weighted DM coefficients has more effect on the performance of the network than the spread parameters but less than the mean vector parameters.

6.4 Effect of applying Newton's method repeatedly

In Figure 6.4, we try to see the effect of applying repetitive Newton's method on various parameters. The main constraint seen here is on computational time which increases considerably. We observe that while we apply Newton's method 40 times to the spread parameters, the performance is not much improved. Similar studies have been done on other parameters leading to the conclusion that after a certain threshold, the training reaches saturation.

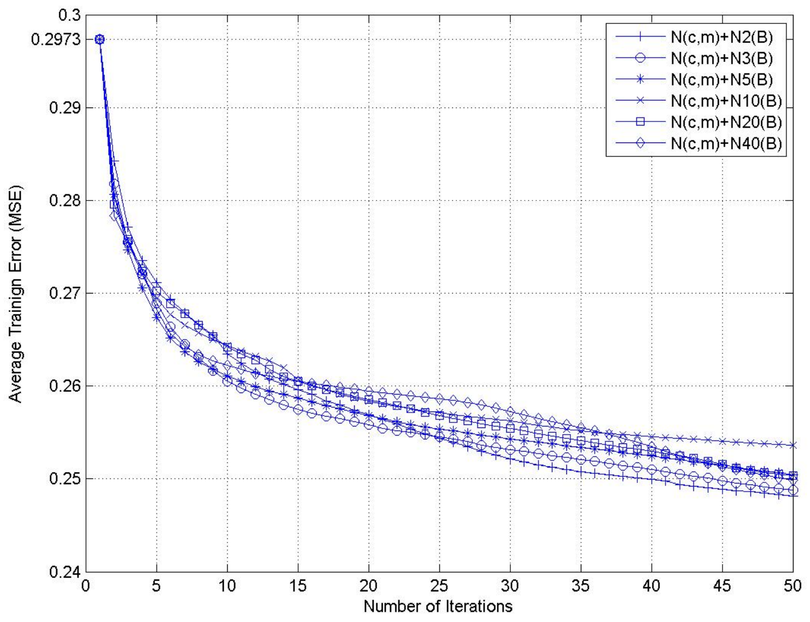


Figure 6.4 Effect of repetitive Newton's method on spread parameter on *twod* dataset

An important conclusion coming from this chapter is that a change in mean vector parameters either by optimizing it via Newton's method or adding weighted DM to it has more profound effect than optimizing the spread parameters. The experimental results further bolster this conclusion. A theoretical explanation can be given on this based on sensitive analyses where we observe that error function is more prone to change in mean vector parameters than to the spread parameters. Visualizing this, once the spread parameter is set, mean vector parameters is deciding the position of the Gaussian function to cover the entire input space. Hence the effect of the position of the Gaussian function is more important than the width of each of them.

CHAPTER 7

FINAL RBF TRAINING ALGORITHMS ANALYSES

Based on the conclusion made in Chapter 6, we take two training algorithms for further investigation. Firstly, the training algorithm where does not have weighted DM and conventional parameters (spread parameters and mean vector parameters) are optimized and Secondly, the training algorithm in which we optimize all three parameters. These two training algorithms will be compared with LM and RLS-RBF training algorithms.

Figure 7.1 shows the flowchart for the two-step hybrid learning procedure to train the proposed RBF network by optimizing all the three parameters. Deleting the DM optimization step from the flowchart will result in the algorithm for optimizing only conventional parameters.

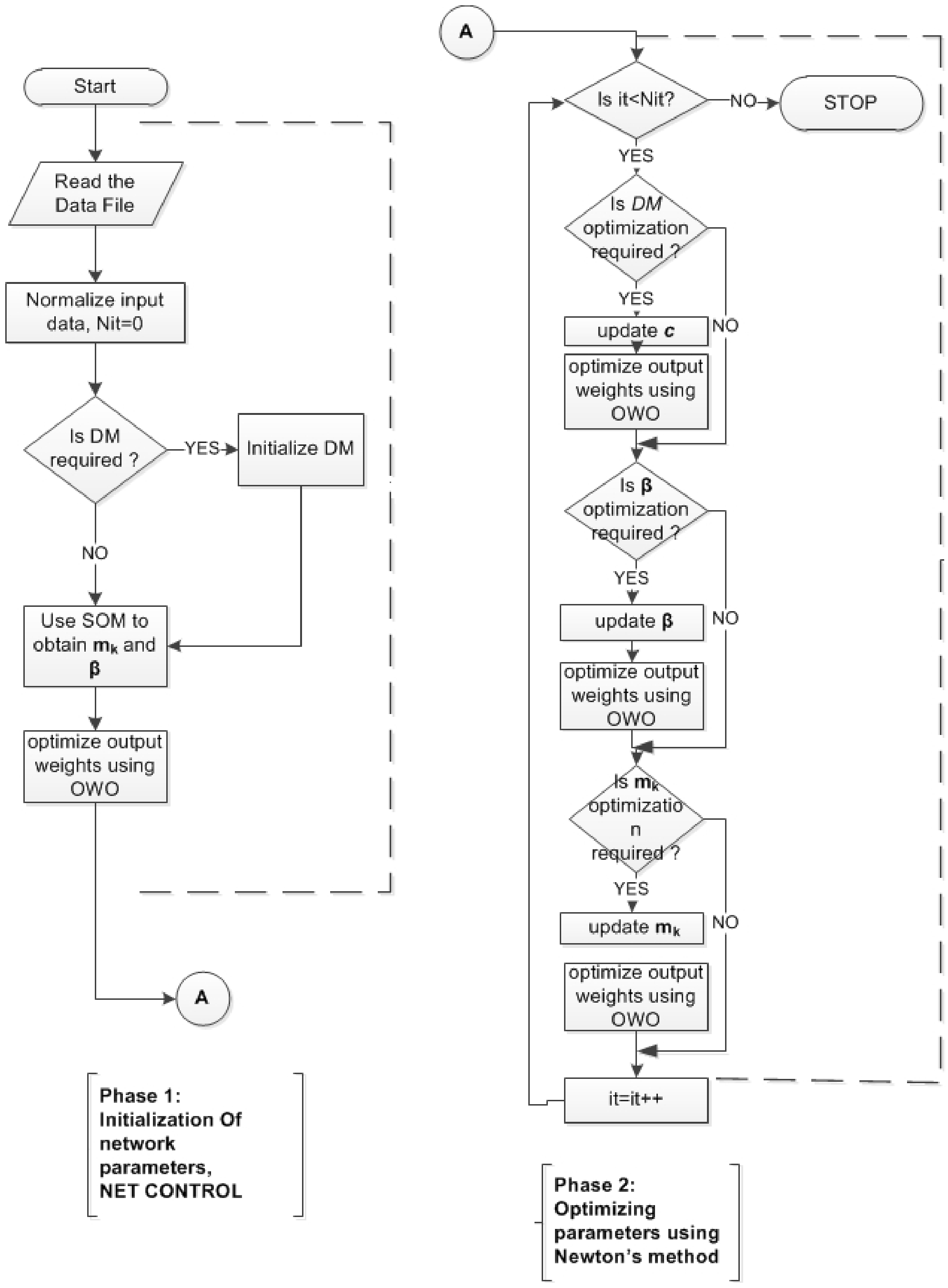

Start
Read the Data File
Normalize input data, Nit=0
Is DM required ?
YES
Initialize DM
NO
Use SOM to obtain m_k and β
optimize output weights using OWO
A
Phase 1: Initialization Of network parameters, NET CONTROL
A
Is it<Nit?
NO
STOP
YES
Is DM optimization required ?
YES
update c
NO
optimize output weights using OWO
Is β optimization required ?
YES
update β
NO
optimize output weights using OWO
Is m_k optimization required ?
YES
update m_k
NO
optimize output weights using OWO
it=it++
Phase 2: Optimizing parameters using Newton's method

Figure 7.1 Flowchart for optimizing all three parameters

7.1 <u>Computational burden</u>

In this section, we describe the computational burden for the proposed training algorithms that we finally take up for comparison with RLS-RBF and LM training algorithms. The total number of weights in the network is denoted as:

$$N_w = M(N + N_h + 1) + N_h(N + 1) \tag{7.1}$$

Let *net control* denote the step involving the initialization of the parameters as discussed in sub-section 2.2. The number of multiplications required per training iteration for this is denoted by M_{nc} which is given by:

$$M_{nc} = 2N_hN \tag{7.2}$$

The number of multiplications required per training iteration to solve for output weights by using orthogonal least squares [49] is M_{ols}, which is given by:

$$M_{ols} = N_u(N_u + 1)\left[M + \frac{1}{6}N_u(2N_u + 1) + \frac{3}{2}\right] \tag{7.3}$$

The number of multiplication required per training iteration in optimizing the spread parameters is given by:

$$M_\beta = M_{nc} + M_{ols} + N_v[N_h\big(N + N_h + M(N_h + 1)\big) + MN_u] \tag{7.4}$$

and the number of multiplication required per training iteration in optimizing the mean vector parameters is given by:

$$M_m = M_{nc} + M_{ols} + N_v[N_h(2N + 3M + 1) + N_h^2(M + 1) + MN_u + N^2] \tag{7.5}$$

Therefore the training algorithm in which we jointly optimize the spread parameters and mean vector parameters, the number of multiplication is given by:

$$M_{\beta m} = M_{\beta} + M_m \tag{7.6}$$

Let M_c denotes the number of multiplication required per training iteration for calculating the weighted DM coefficients which is given by:

$$M_c = N_h N \tag{7.7}$$

The number of multiplication in the training algorithm where we jointly optimize all the three parameters is denoted by $M_{c\beta m}$ and is given as :

$$M_{c\beta m} = M_c + M_{\beta m} \tag{7.8}$$

For RLS-RBF training algorithm, the number of multiplication required per training iteration is denoted by $M_{RLS}$and is given by:

$$M_{RLS} = N_h N + M_{ols} \tag{7.9}$$

For LM training algorithm, the number of multiplications required per training iteration during the backpropagation step is given by:

$$M_{bp} = N_v[MN_u + 2N_h(N+1) + M(N + 6N_h + 4)] + N_w \tag{7.10}$$

Thus the total multiplications per training iteration in LM training algorithm are:

$$M_{lm} = M_{bp} + N_v\left[MN_u\left(N_u + 3N_h(N+1)\right) + 4N_h^2(N+1)^2\right] + N_w^3 + N_w^2 \tag{7.11}$$

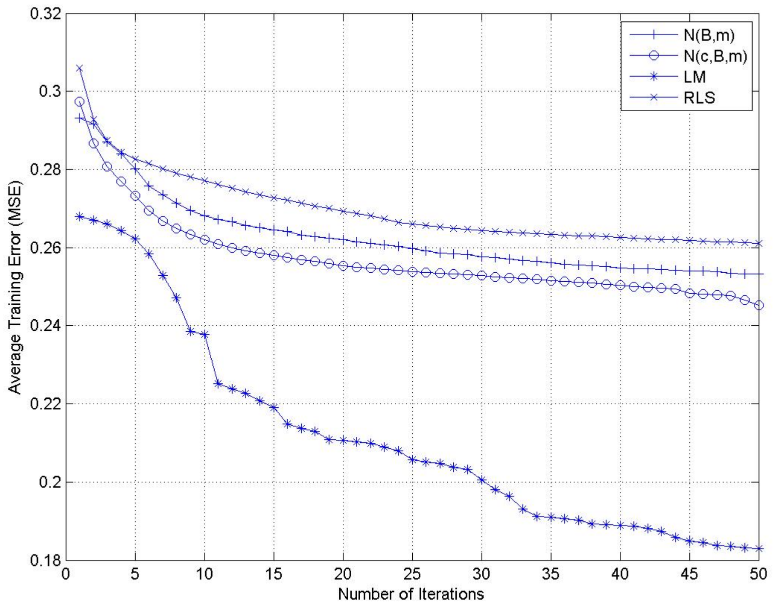
7.2 Experimental analyses

We now compare the performance of our two training algorithms with LM and RLS-RBF training algorithms. For a given network, we obtain the training error and the number of multiplications required for each training iteration. We also obtain the validation error for a fully trained network. This information is used to subsequently generate the plots and compare performances. We use *k-fold* cross validation procedure to compare the generalization performance of the training algorithms. During our investigation we take two highly correlated

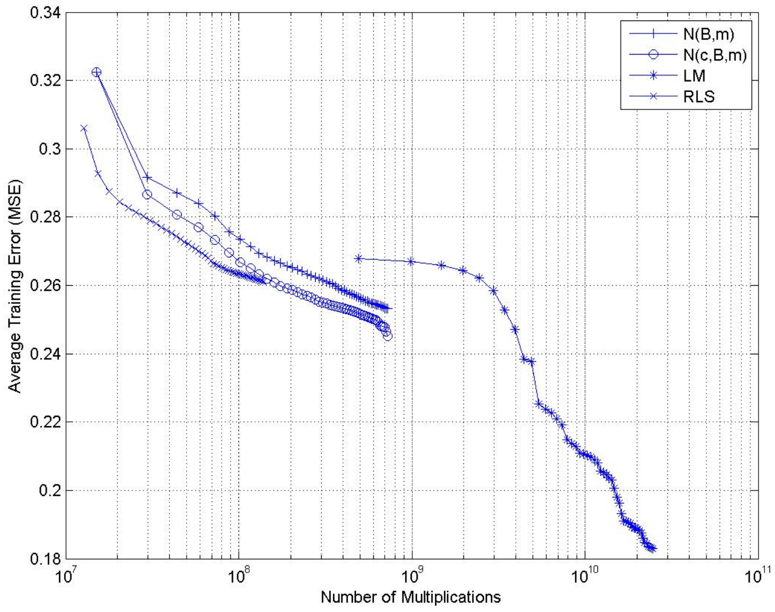
($\rho > 0.8$) datasets and other two as least correlated ($\rho < 0.2$) datasets. Here ρ is the correlation coefficient.

7.2.1 twod dataset

'twod.tra' is a highly correlated input data set. We trained all the training algorithms with hidden unit as 20. In Fig. 5, the Average mean square error (MSE) for training versus the number of iterations (N_{it}) is plotted for each algorithm in figure 7.1.



(a)



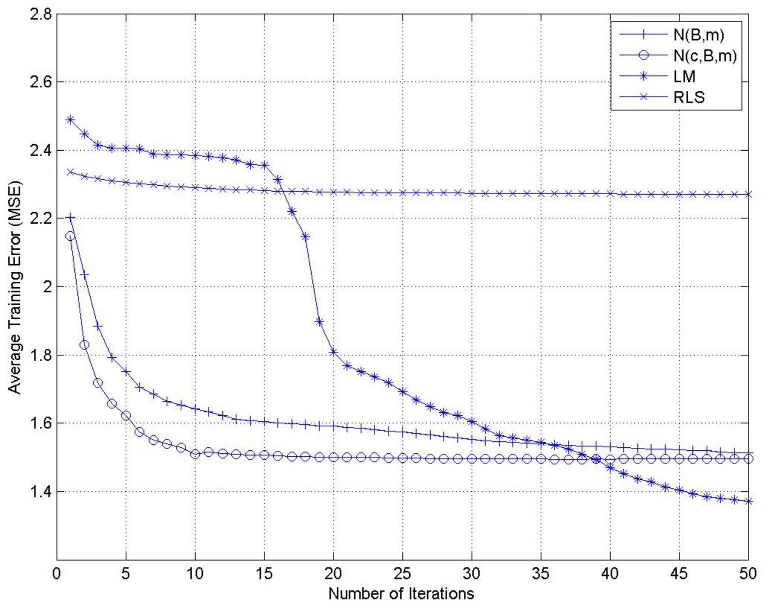
(b)

Figure 7.2 Performance of final training algorithms with LM and RLS-RBF on *twod* (a) Number of Iterations *vs* Average Training Error (b) Number of Multiplications *vs* Average Training Error

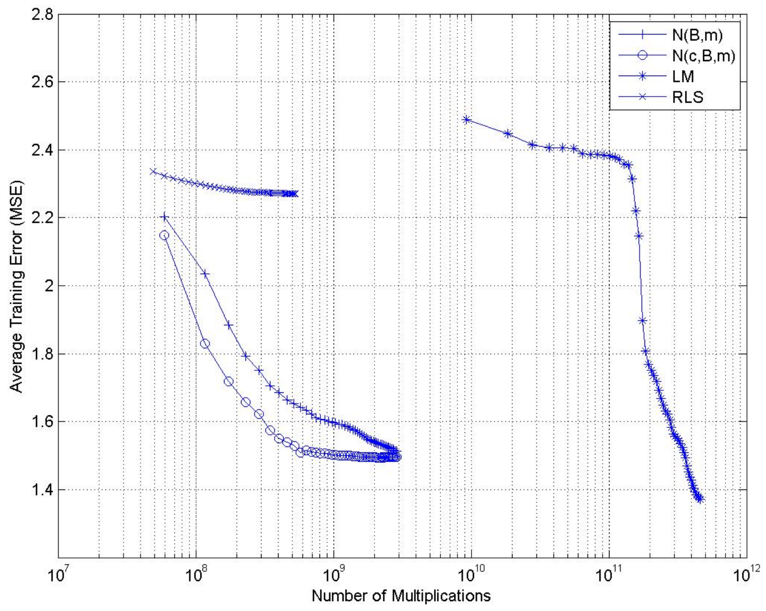
From the plot we deduce that the network including weighted DM improves the performance of the training algorithm as compared to one in which no weighted DM is used. However LM performs well from both the training algorithms. Figure 7.2 plots the Number of Multiplications *vs* the Average Training Error for training which reveals the downside of using LM. Although LM performs better than both the two training algorithms but it also takes a large computation time owing to the large number of numerical calculation involved. Hence its practical application on large data set is extremely limited. This is where our training algorithms are highly efficient since they are small and powerful and performs well on large dataset**.**

7.2.2 oh7 dataset

In *oh7* dataset, not only the input values but also one of the output values is highly correlated with the input values unlike *twod*. We train all the RBF training algorithms with hidden unit as 20. Figure.7.3, plots the MSE for training versus the number of iterations for all algorithms. The observations are similar to that of previous dataset but we observe that due to high input-output correlation the algorithms which include weighted DM optimization as well as those which optimizes only $\boldsymbol{\beta}$ and $\boldsymbol{m}$, both, comes close towards the optimal performance.



(a)



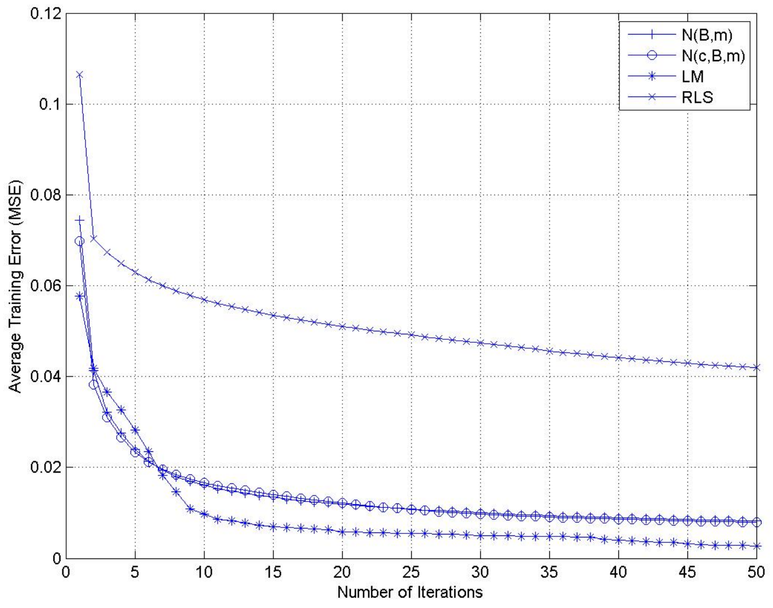
(b)

Figure 7.3 Performance of final training algorithms with LM and RLS-RBF on *oh7* (a) Number of Iterations *vs* Average Training Error (b) Number of Multiplications *vs* Average Training Error

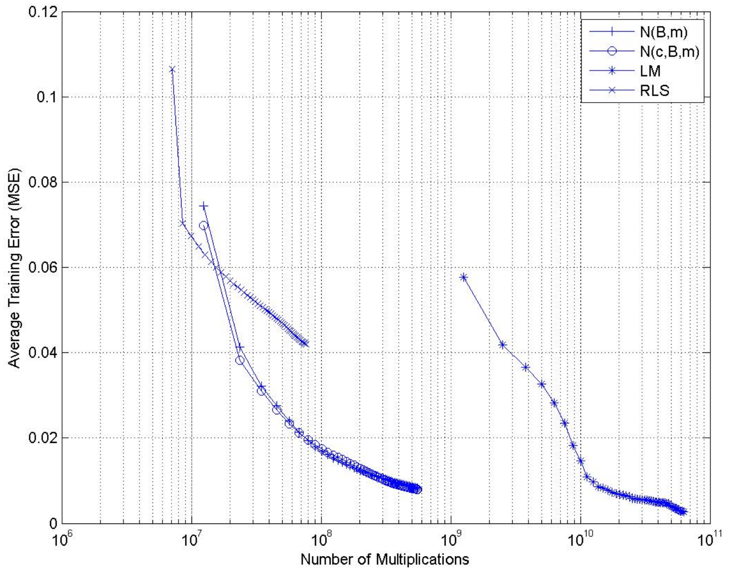
From figure 7.3 we see that LM performs marginally well than our proposed training algorithm but in figure 7.4 it is revealed that LM takes fairly large amount of multiples making it a slow training algorithm for large data set. It will be interesting to see the performance of the above three training algorithms on uncorrelated data set which will be discussed in next section.

7.2.3 mattrn dataset

In *mattrn* dataset, each pattern consists of 4 input features and 4 output features. For this data set, all the RBF training algorithms are trained with hidden unit as 15. *mattrn* data set is un-correlated.



(a)



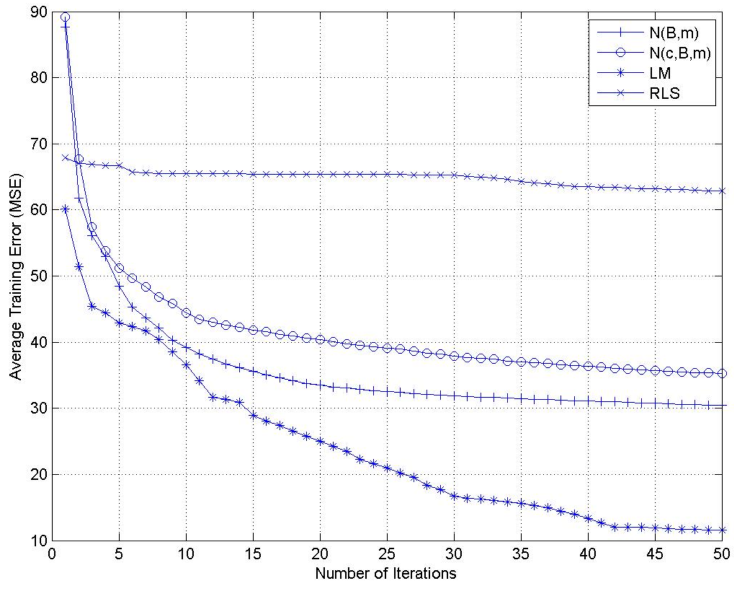
(b)

Figure 7.4 Performance of final training algorithms with LM and RLS-RBF on *mattrn (*a) Number of Iterations *vs* Average Training Error (b) Number of Multiplications *vs* Average Training Error

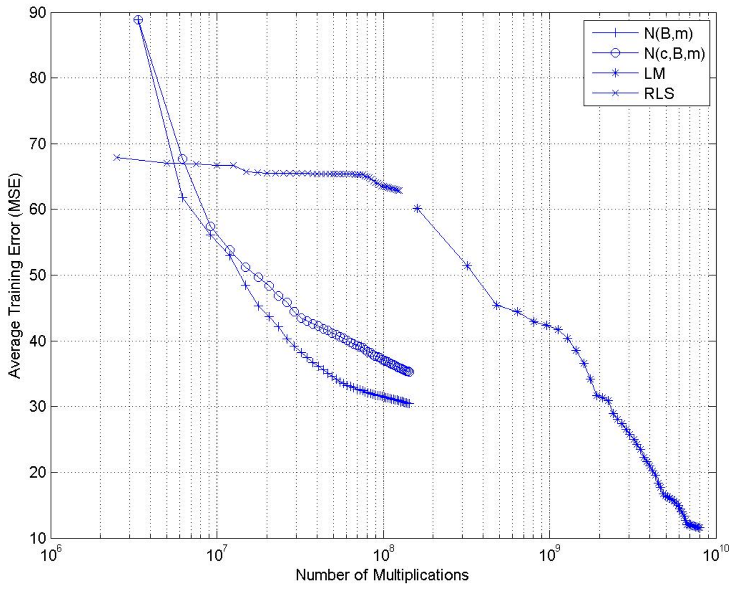
From figure 7.4 we make an important observation. The training algorithm with weighted DM optimization performs closely with the training algorithm based on $\boldsymbol{\beta}$ and $\boldsymbol{m}$ optimization. This concludes that including weighted DM has not much effect on the uncorrelated dataset and since they have slightly more number of multiples, it will be computationally economical to use the training algorithm based on $\boldsymbol{\beta}$ and $\boldsymbol{m}$ optimization alone. LM performs almost similar to our training algorithms here but again owing to the large number of multiples as shown in figure 7.6 it is practically not a viable training algorithm to train the network.

7.2.4 concrete dataset

concrete is an un-correlated dataset used to approximate the nonlinear function of age and ingredients of concrete compressive strength. With a total number of patterns as 1030, this dataset consists of 8 inputs and 1 output and hidden units are takes as 20.This dataset is also least correlated and we see the same trend as we see in *mattrn*.



(a)



(b)

Figure 7.5 Performance of final training algorithms with LM and RLS-RBF on *concrete* (a) Number of Iterations *vs* Average Training Error (b) Number of Multiplications *vs* Average Training Error

Again the performance of the training algorithm with $\boldsymbol{\beta}$ and $\boldsymbol{m}$ optimization is close to the training algorithm having weighted DM optimization.

7.3 *k-fold* cross validation

We now use the *k-fold* cross validation procedure to show the generalization abilities of all the three training algorithms. For each dataset, we split it randomly into 10 non-overlapping parts of equal size, and use 9 parts of total data for training and leave the remaining one part for testing. This procedure was repeated till we have exhausted all 10 combinations. Then, by training all these combinational datasets, we got the average of training errors. Also, the validation error of each dataset was obtained by averaging all corresponding testing errors on every testing dataset. The training MSEs and test MSEs of *k-fold* crossing validation on these four datasets are listed as Table 1 respectively.

Table 7.1 Comparison of *k-fold* cross validation on different datasets

Data Set		**N(B,m)**	**N(c,B,m)**	**LM**
Twod	E_{trn}	0.2503	0.2417	0.2005
	E_{val}	0.3589	0.3270	0.2344
oh7	E_{trn}	1.5069	1.4643	1.2687
	E_{val}	1.7056	2.3987	2.1407
Mattrn	E_{trn}	0.0084	0.0076	0.0048
	E_{val}	0.0199	0.0180	0.0235
Concrete	E_{trn}	39.647	40.2735	23.3564
	E_{val}	40.2950	40.8721	31.9576

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this thesis, the optimization of RBF neural network parameters with Newton's method is analyzed. Optimizing the RBF neural network parameters with Newton's method is an improvement over the existing training algorithms in the literature. The proposed training algorithms are simple to train and yet powerful with minimum number of hidden units. We are successfully able to train small but powerful network with better generalization and training capability. Apart from the conventional parameters we also introduced weighted DM. Training of the DM weights significantly improves the RBF network. From our experimental results we conclude that the performance of the various training algorithms is affected by the data set correlation. The effect of Newton's method is more pronounced in correlated datasets. For un-correlated dataset the effect of distance measure (DM) is not much and its performance is same with training algorithms where only conventional parameters have been optimized. Therefore DM is significantly helpful when the dataset is correlated. The generalization ability is further substantiated by the *k-fold* validation. Newton's method on all three parameters helps significantly in improving the performance of RBF networks.

We also experimented with single and multiple OLF and combined it with spread parameters and mean vector parameters. We concluded that the multiple OLF training of the mean vector parameters is more effective than the training of $\boldsymbol{\beta}$.The experimental results further bolster this conclusion. It came out from our analyses that optimizing mean vector parameters has more effect on the training result than any other parameter. We also mathematically observed the reason of non-singularity of Hessian matrix in case of linear dependency of inputs Incorporating a method to optimally select the number of hidden units based on different dataset is definitely necessary. Experimenting with L-2 error norm, VC dimension for the proposed training algorithms are some of the future areas that needs to be explored

APPENDIX A

CLUSTERING ALGORITHMS

1. *K*-means clustering

K-means clustering algorithm is used to locate a set of *k* RBF center between the training set vector x_k and the nearest of the *k* receptive mean vectors $\boldsymbol{m_k}$[51]. *K*-means algorithm allocated each data point to one of the ***c*** cluster to minimize the within cluster sum of square:

$$\sum_{i=1}^{c} \sum_{k \in A_i} \|x_k - m_i\|_2 \tag{1.1}$$

whereA_i is the data points in the *i*th cluster and m_i is the mean for that points over the cluster ***i***. (1.1) denotes a distance norm which is minimized in the mean square sense.

2. Self-Organizing Map (SOM)

In our work we use Self-Organizing feature Map (SOM) for1 dimensional cluster in order to set up initial mean vector parameters and spread parameters value. SOM is a variation to the adaptive *K*-means where we cluster the data set based upon the relative distance. The advantage of using SOM over *K*-means is that the former take care of the re-ordering of the cluster. While in SOM, the cluster generated do not form an optimal quantize, but may serve as an initial cluster generator for other clustering techniques.

APPENDIX B

DESCRIPTION OF TRAINING DATA SETS

In this appendix, we give some description about the training data sets which are used through the thesis.

Training data set *twod* is available on the Image Processing and Neural Networks Lab repository [50]. It contains simulated data based on training algorithms from back-scattering measurements [50]. This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements. The training data set has 8 inputs, 7 outputs, and 1768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters σ^o at V and H polarizations and four incident angles $(10^o, 30^o, 50^o, 70^o)$. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo of an in-homogenous irregular layer above a homogenous half space.

Training data set *mattrn* is available on the Image Processing and Neural Networks Lab repository [50]. It contains the data for inversion of random two by two matrices [50]. Each of the 2000 patterns consist of 4 input features and 4 output features. The input features, which are uniformly distributed between 0 and 1, represent matrix elements and the four output features are elements of the corresponding inverse matrix. The determinants of the input matrices are considered to be between 0.3 and 2.

oh7 is available on the Image Processing and Neural Networks Lab repository [50]. Inputs for the training data set *oh7* are *VV* and *HH* polarizations at *L*30, 40 deg, C 10, 30,40,50, 60 deg, and X 30,40,50 deg [50]. The corresponding desired outputs are $\Theta = \{\tau, l, m_v\}^T$, where

$\tau$the rms surface height is, l is the surface correlation length; m_v is the volumetric soil moisture content in g/cm^3. There are 20 inputs, 3 outputs, 10453 training patterns.

concrete data file is available on the UCI Machine Learning Repository [51]. It contains the actual concrete compressive strength (MPa) for a given mixture under a specific age (days) determined from laboratory. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, super plasticizer, coarse aggregate, and fine aggregate. The data set consists of 8 inputs and 1 output per pattern, with a total of 1030 patterns.

REFERENCES

[1] Kumar,S., *Neural Network: a classroom approach,* International ed. McGraw Hill Press, 2005, pp 304-314.

[2] Medgassy, P. 1961. *Decomposition of superposition of distributed functions,* Hungarian academy of Sciences, Budapest.

[3] Micchelli, C.A.1986. *Interpolation of scattered data: Distance and conditionally positive definite functions*, Constructive Approximations, vol.2, pp. 11-22.

[4] Powell, M.J.D 1987. Radial Basis functions for multivariate interpolation: a review, in algorithms for the approximation of Functions and Data. J.C.Mason and M.G.Cox eds., Clarendon Press, Oxford, England.

[5] Duda, R.O., and Hart, P.E.1973. *Pattern Classification and Scene Analysis*, Wiley, New York.

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

[7] Poggio, T., and Girosi, F. 1989. *A theory of Networks for Approximation and Learning*. A.I. Memo 1140, MIT, Cambridge

[8] Broomhead, D.S. and Lowe, D. (1988), " Multivariate functional interpolation and adaptive networks," *Complex Systems*,2,321-355

[9] Lee, S. and Kil, R. (1988). "Multilayer feed-forward potential function networks," in *Proceedings of the IEEE Second International conference on Neural Networks* (San Diego 1988), vol, 161-171, IEEE, New York.

[10] Niranjan,M. and Fallside, F. (1988). "Neural Networks and Radial Basis Functions in classifying static speech patterns," *Technical Report CUEDIF-INFENGI7R22*, Engineering Department, Cambridge University.

[11] J.E. Moddy and C.J.Darken," Fast learning in networks of locally-tuned processing units," *Neural Computation*. vol.1, pp 281-294,1989.

[12] G.B.Huang, P.Saratchandran and N.Sundararajan, "A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation," *IEEE Trans. Neural Networks*, vol16, no 1, pp.57-67, Jan 2005.

[13] S.Chen, C.F.N. Cowan, and P.M. Grant, "Orthogonal Least Squares learning algorithms for radial basis functions networks," *IEEE Trans. Neural Network*. vol 2, no2,pp.302-309, Mar.1991.

[14] S.Chen, E.S.Chng, and K.Alkadhimi, "Regularized orthogonal least square algorithm for construction radial basis function networks," Int*. J. Control*, vol.64, no.5, pp.829-837, 1996.

[15] E.S.Ching, S. Chen, and B. Mulgrw," Gradient radial basis function network for nonlinear and nonstationary time series prediction," *IEEE Trans. Neural Networks*, vol.7, no.1, pp.190-194, Jan.1996.

[16] M.J.L. Orr, "Regularization on the selection of radial basis function centers," *Neural Computation*. Vol.7, pp.606-623,1995

[17] Karayiannis, N.B., "Gradient descent learning of radial basis neural networks," *in Proc.1997 IEEE Int. conf. Neural Networks*, vol.3, Houston, TX, June 9-12, 1997, pp.1815-1820.

[18] Karayiannis, N.B., "Learning algorithms for reformulated radial basis neural networks," in *Proc.1998 Int. Joint Conf. Neural Networks*, Anchorage, AK, 1998, pp.2230-2235.

[19] Karayiannis, N.B., "Reformulated radial basis neural networks trained by gradient descent," *IEEE Trans. Neural Networks*, vol.10, pp.657-671, May 1999.

[20] Karayiannis, N.B., and Behnke,S., 2000 "New radial basis neural networks and their application in a large-scale handwritten digit recognition problem," in *Recent Advances in Artificial Neural Networks: Design and Applications*, L. C. Jain and A. M. Fanelli, Eds. Boca Raton, FL: CRC, pp. 39–94.

[21] Shi, Y., Globally convergent algorithms for unconstrained optimization, *Computational Optimization and Application* vol.16, pp 295-308, 2000.

[22] Malalur, S., Manry, M.,” Feed-Forward Network Training Using Optimal Input Gains,” *International Conference on Neural Networks*, Atlanta, pp. 1953-1960, Jun 2009.

[23] Cha, I., and Kassam,S.A., “ Interference cancellation using radial basis function networks,” *Signal Processing*, vol.47, pp.247-268, 1995

[24] Whitehead, B.A., and Chaote, T.D., “Evolving space-filling curves to distribute radial basis functions over an input space,” *IEEE Trans. Neural Networks*, vol.5, pp. 15-23, Jan. 1994.

[25] Haykin, S., *Neural Network: A Comprehensive Foundation*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 2009

[26] Chen. S., Cowan. C.F.N., and Grant P.M., “Orthogonal least Square Learning algorithm for Radial Basis Function Networks”, *IEEE Trans. On Neural Networks*, Vol2, No 2 (Mar) 1991.

[27] Haykin, S*., Adaptive Filter Theory,* 3rd ed. Englewood Cliffs, N.J.: Prentice Hall, 1996.

[28] Manry, M., Apollo, S.J. and Yu. Q, “Minimum mean square estimation and neural networks,” *Neurocomputing*, vol 13.

[29] Barnard. E., “Optimization for training neural nets,” *IEEE Trans. Neural Networks*, vol.3, no.2, pp.232-240, 1992.

[30] Battiti. R., “First and second order methods of learning: between steepest descent and Newton’s method,” *Neural Computation*, vol.4, no.2, pp.141-166,1992.

[31] Bishop, C., “Exact calculation of the hessian matrix for the multilayer perceptron,” *Neural Computation*, vol 4, no. 4, pp. 494-501,1992

[32] Flethcer, R., *Practical Methods of Optimization, 2nd ed*., Chichester, NY: John Wiley & Sons, 1987.

[33] Moller, M., *Efficient training of feed forward neural Networks*, Ph.D. dissertation Aarhus University, Denmark, 1997.

[34] Amapzis, N and Perantonis. S.j, " Two highly efficient second order algorithms for training feedforward networks," *IEEE Trans. Neural Networks*, vol.13, pp.1064-1074,2002.

[35] Irwin. G., Lightbody., G. and Mcloone., S., "Comparison of gradient based training algorithms for multilayer perceptrons" in *IEEE Colloquium on advances in Neural Networks for Control and Systems*, 1994, May, pp.11/1-11/6.

[36] Wasserman, P.D., *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, New York, 1993

[37] Subramanian., C, Manry., M and Naccarino., J., "Reservoir inflow forecasting using neural networks," in *Proceedings of the American Power Conference*, vol 61, 1999, pp. 220-225.

[38] Manry. M et al., " Fast Training of neural networks for remote sensing," *Remote Sensing Reviews*, vol9, pp.77-96, 1994.

[39] Saarein, S., Bramley, R., and Cybenko, G., "Ill-conditioning in neural network training problems." *SIAM Journal on Scientific Computing*, vol.14, pp 693-714, 1993.

[40] Smagt, P., and Hirzinger, G., *Solving the ill-conditioning in neural networks learning, ser. Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science 1524, G.Orr and K.R.Muller, Eds, Springer Verlag, 1998.

[41] Golub G., Loan C., "*Matrix Computations*", Johns Hopkins university press, 3rd ed, 1996.

[42] Press. W.H., et al., " *Numerical Recipes in C* "., New York: Cambridge University Press, 2005.

[43] Levenberg, K.,"A method for the solution of certain problems in least squares," *Quart. Appl. Math.*,pp. 2, pp.164.pp.168,1944.

[44] Marquardt, D., An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, pp. 11, pp.431-441, 1963

[45] Huang, G.B, Zhou H., Ding., X., and Zhang., R., " Extreme Learning machine for regression and multiclass classification*,"(in press) IEEE Transaction on systems, Man and Cybernetics- Part B: Cybernetics*, 2011

[46] Doost. R., Sayadian., and Shamsi., H., "A new perceptually weighted distance measure for vector quantization of STFT amplitudes in the speech application", *IEICE Electron. Express*, vol 6, No. 12, pp. 824-830, (2009).

[47] Wachter., M, Demuynck., K., Wambacq., P, Compernolle., D., " A locally weighted distance measure for example based speech recognition", *Proc. of IEEE International conference on Acoustic, Speech and Signal Processing*, vol1., pp 181-4, 2004.

[48] Malalur, S., and Manry,M., "Multiple optimal learning factors for feed-forward networks," *Proc. of SPIE: Independent Component Analyses, Wavelets, Neural Networks, Bio-systems, and Nanoengineering* VIII, Orlando Florida, vol. 7703 pp. 77030F-1 – 77030F-12, April 7-9, 2010.

[49] Maldonado. F.J., Manry., M., Kim. T.H., "Finding optimal neural network basis function subsets using the Schmidt procedure", *Proc. of the International Joint Conference on Neural Networks,*vol 1, pp. 444-449, 20-24 July 2003.

[50] Source: http://www-ee.uta.edu/eewb/ip/training_data_files.htm

[51] Source: http://archive.ics.uci.edu/ml/

[52] MacQueen, J.B., "Some Method for classification and Analysis of Multivariate observations", *Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability*. University of California Press. Pp.281-297. MR0214227.

[53] Rifkin, R.M., "*Everything old is new again: a fresh look at historical approaches in machine learning"*, Ph.D. thesis, MIT, 2002.

BIOGRAPHICAL INFORMATION

Kanishka Tyagi was born in Meerut, India, in 1984. He received his Bachelor Degree in Electrical Engineering in 2008 from G.B.Pant University of Agriculture and Technology, Pantnagar, India. From 2008 to 2009 he was a Research Associate with Dr.P.K.Kalra at the Department of Electrical Engineering, Indian Institute of Technology Kanpur, Kanpur, India. There he worked on the development of audio classification algorithms, blind source separation algorithm and virtual foot balloon system. He is currently a Graduate student in Image Processing and Neural Networks Lab, The University of Texas at Arlington, USA. During spring 2011, he worked as intern at Verizon Wireless, Irving, Texas to develop recommendation systems for FiOS project module. During summer 2011, he was a visiting research student at Multimedia Signal Processing Lab with Dr. Nojun Kwak at Ajou University, South Korea. Currently, he is working as an intern at Siemens Energy, Richland, Mississippi to develop embedded based communication layer and firmware code for next generation voltage regulators.

His specific research interests are non-linear neural networks; L-1 based subspace algorithms, signal processing and image segmentation and face-detection. He is a member of Tau Beta Pi, Engineering Honor Society and recipient of 2007 IEEE Computational Intelligence Society outstanding student paper travel grant award and 2011 IEEE FUZZ outstanding student paper travel grant award.