SEQUENCES OF NEAR-OPTIMAL FEEDFORWARD NEURAL NETWORKS

by

PRAMOD LAKSHMI NARASIMHA

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

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2007

Dedicated to my parents.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervising professor Dr. Michael T. Manry for his constant support and invaluable guidance, which made it possible to complete this Dissertation. The knowledge I acquired under his direction and supervision has made me eligible to investigate independently in the areas of machine learning and neural networks. I would like to express my appreciation for his reviews and suggestions on a number of research papers that I have authored/coauthored during my Doctoral studies. He has dedicated lot of his time to motivate me in this research and his teachings will go with me a long way in reaching my career goals.

I would like to express my appreciation to Dr. Frank Lewis, Dr. Qilian Liang, Dr. Wei-Jen Lee, Dr. Khosrow Behbehani and Dr. Jean Gao for reviewing this material and serving on my Dissertation committee. I would also like to express my special regards to Dr. Wei-Jen Lee for providing us with a forecasting dataset.

I would like to thank my former lab mates Dr. Jiang Li and Dr. Changhua Yu for sharing their research experiences with me. I would like to thank Mr. Sanjeev Malalur and Mr. Ninad Thakoor for being great sources of inspiration to me by sharing their ideas and views on various research areas. It was an excellent learning experience having them around. I would also like to thank all my friends at UTA for sharing their joy and happiness with me.

I would like to express my deepest love and affection to my parents and family members for providing me with moral support and motivation, which was one of the main reasons that I was able to complete this work.

July 19, 2007

ABSTRACT

SEQUENCES OF NEAR-OPTIMAL FEEDFORWARD NEURAL NETWORKS

Publication No. _____

PRAMOD LAKSHMI NARASIMHA, Ph.D.

The University of Texas at Arlington, 2007

Supervising Professor: Michael T. Manry

In order to facilitate complexity optimization in feedforward networks, several integrated growing and pruning algorithms are developed. First, a growing scheme is reviewed which iteratively adds new hidden units to full-trained networks. Then, a non-heuristic one-pass pruning technique is reviewed, which utilizes orthogonal least squares. Based upon pruning, a one-pass approach is developed for producing the validation error versus network size curve. Then, a combined approach is devised in which grown networks are pruned. As a result, the hidden units are ordered according to their usefulness, and less useful units are eliminated. In several examples, it is shown that networks designed using the integrated growing and pruning method have less training and validation error. This combined method exhibits reduced sensitivity to the choice of the initial weights and produces an almost monotonic error versus network size curve.

Starting from the strict interpolation equations for multivariate polynomials, an upper bound is developed for the number of patterns that can be memorized by a non-linear feedforward network. A straightforward proof by contradiction is presented for the upper bound. It is shown that the hidden activations do not have to be analytic. Networks, trained by conjugate gradient, are used to demonstrate the tightness of the

bound for random patterns. The theoretical results agree closely to the simulations on two class problems solved by support vector machines.

We model large classifiers like Support Vector Machines (SVMs) by smaller networks in order to decrease the computational cost. The key idea is to generate additional training patterns using a trained SVM and use these additional patterns along with the original training patterns to train a much smaller neural network. Results shown verify the validity of the technique and the method used to generate additional patterns. We also generalize this idea and prove that any learning machine can be used to generate additional patterns and in turn train any other machine to improve its performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Neural networks are motivated by the functioning of the human cognitive processes but their implementation is carried out in a much different way. Neural networks have a massively parallel distributed structure made up of simple processing units that can learn and generalize. They are nonparametric, nonlinear, universal approximators that can uniformly approximate any real continuous function to an arbitrary degree of accuracy. This has been proved using Stone-Weirstrass theorem from real analysis by Hornik *et al.* [1] as pointed out by Lorentz [2] and Hecht-Nielsen [3]. Kolmogorov's theorem [4] states that arbitrary continuous function of n-variables can be approximated using linear summations and superpositions of continuously increasing functions of only one variable. This theorem has led investigators to develop neural networks, which are used to solve approximation and classification problems. Applications include power load forecasting [5, 6], ZIP code recognition [7, 8], prognostics [9], target recognition [10, 11] and stock market prediction [12] among several others.

1.1 Growing and Pruning of Feedforward Networks

When optimizing the structural complexity of feedforward neural networks, different size networks are designed which minimize the mean-square training error. Typically, the machine with the smallest validation error is saved. When different size networks are independently initialized and trained, however, the training error versus network size curve is not monotonic. Similarly, the resulting validation error versus size curve is not smooth, making it difficult to find a proper minimum. Monotonic training curves and smooth validation curves can be produced by growing methods or by pruning methods.

In growing methods [13], new hidden units are continually added to a trained network, and further training is performed [14]. In [15], the addition of new units continues as long as the sum of the accuracies for training and cross validation samples improves. This eliminates the requirement of specifying minimum accuracy for the growing network. Guan and Li [16] have used output parallelism to decompose the original problem into several sub-problems, each of which is composed of the whole input vector and a fraction of the desired output vector. They use constructive backpropagation to train modules of the appropriate size. Modules are merged to get a final network.

The main drawbacks of growing methods are that some of the intermediate networks can get trapped in local minima. Also, growing methods are sensitive to initial conditions. Unfortunately, when a sequence of grown network does not guarantee optimal training of all the hidden units. It is often likely for some hidden units to be linearly dependent.

In pruning methods, a large network is trained and then the least useful nodes or weights are removed [17, 18, 19, 20, 21]. Orthogonal least squares or Gram-Schmidt based pruning [22] has been described by Chen et al. [23] and Kaminsky and Strumillo [24] for Radial Basis Functions (RBF) and by Maldonado et al. [25] for the Multilayer Perceptron (MLP). In Crosswise Propagation, [26], linearly dependent hidden units are detected using an orthogonal projection method, and then removed. The contribution of the removed neuron is approximated by the remaining hidden units. In [27], the hidden units are iteratively removed and the remaining weights are adjusted to maintain the original input-output behavior. A weight adjustment is carried out by solving the system of linear equations using the conjugate gradient algorithm in the least squares sense. Unfortunately, if one large network is trained and pruned, the resulting error versus number of hidden units (N_h) curve is not minimal for smaller networks. In other words, it is possible, though unlikely, for each hidden unit to be equally useful after training.

A promising alternative to growing or pruning alone is to combine them. Huang et al. [28] have developed an online training algorithm for RBF based Minimal Resource Allocation Networks (MRAN), which uses both growing and pruning. However, as MRAN

is an online procedure, it does not optimize the network over all past training data. Also, MRAN network initialization requires prior knowledge about the data which may not be available.

1.2 Pattern Storage

It is important to understand the pattern memorization capability of feedforward networks for several reasons. First, the capability to memorize is related to the ability to form arbitrary shapes in weight space. Second, if a network can successfully memorize many random patterns, we know that the training algorithm is powerful [29]. Third, some useful feedforward networks such as Support Vector Machines (SVMs), memorize large numbers of training patterns [30, 31].

Many researchers have derived upper and lower bounds on the number of hidden units for a Multilayer Perceptron (MLP). Baum [32] has derived a lower bound on the pattern storage by finding the smallest MLP capable of implementing an arbitrary dichotomy of N_v points. Sartori and Antsaklis [33] have derived lower bounds on the size of a multilayer neural network that exactly implements an arbitrary training set. They have not separated the input space into particular hyperplanes but have simply satisfied the rank condition on the outputs of the hidden layer. Huang and Babri [34] have derived a lower bound on pattern storage or an upper bound for the number of hidden units required in feedforward networks with arbitrary bounded nonlinear activation functions. Huang [35] has derived a lower bound on pattern storage for two-hidden-layer feedforward networks to learn any N_v distinct samples with arbitrarily small error. Mazza [36] has derived a lower bound for the Hopfield associative memory for storing patterns, by using large deviation estimates.

Suyari and Matsuba [37] have proposed a new derivation of the storage capacity of neural networks with binary weights which is based on minimum distance between the patterns rather than the number of patterns. Cosnard et al. [38] have derived upper and lower bounds on the size of nets capable of computing arbitrary dichotomies. They

have used the minimum distance between the two classes as a parameter instead of the number of patterns. Elisseeff and Moisy [39] have also derived upper and lower bounds on the number of hidden units for exact learning, given the size of the dataset and the dimension of the input and output spaces. In [40], Ji and Psaltis derive upper and lower bounds for the information capacity of two-layer feedforward neural networks with binary interconnections. They allow hidden and output units to take integer and zero thresholds respectively and then use an approach similar to Baum's [32] to find the bounds.

Moussaoui [41] and Ma [42] have pointed out that the information capacity is reflected in the number of weights of the network. They have also shown that the space complexity of the learning machine (number of weights of the network) should be minimized in order to achieve good generalization. The proof is based on the extension of the interpolation theorem of Davis [43], which states that for any N_v distinct points there exists a unique $N_v - 1$ degree polynomial that passes through all the points.

1.3 Large, Redundant Networks

Pattern recognition is a branch of artificial intelligence which deals with extracting statistical information from raw data or using *apriori* knowledge to categorize them. Some applications are face recognition [44], fingerprint recognition [45], ZIP code recognition [8, 7], prognostics [9], and target recognition [10, 11] among several others.

A key element in a pattern recognition system is the classifier. In Nearest Neighbor Classifiers (NNCs) the input vector is compared to all the training patterns and classified as belonging to the class to which the nearest training pattern belongs. The NNC has been shown to approximate the Bayes classifier, but is computationally very expensive.

A Multilayer perceptron (MLP) trained with backpropagation [46] is a universal approximator. An MLP classifier is computationally more economic than an NNC. However, it requires more training patterns in order to generalize and it is sensitive to initial values of the weights.

Boosting by filtering [47] improves generalization. However, it requires a lot of training patterns due to the structure of the training algorithm. Boosting by subsampling [48, 49] counters the need for large training data files by resampling the data during training according to a distribution. AdaBoost belongs to this category as it permits the training data to be reused.

Support vector machines (SVMs) [30, 31] solve the problems of other classifiers by mapping the problem to a large feature space, making use of classical regularization, and allowing a subset of the training patterns to be support vectors. The curse of dimensionality is overcome by using the Structural Risk Minimization (SRM) principle. Unfortunately, SVMs follow the lower bound [34] on memorization. They are difficult to train due to the choice of various kernels and kernel parameters [50, 51], and they are too large and redundant. SVMs do not have control over the space complexity of the learning machine. Attempts have been made to reduce the size of the SVM [52] by deleting some patterns. Lee et al. [53, 54] have proposed the Reduced Support Vector Machine (RSVM) where a random subset of training set is chosen and optimized. Kruif and Vries [52] have proposed pruning of SVM by deleting some patterns from the training dataset.

The Relevance Vector Machine (RVM) [55] has a probabilistic sparse kernel model identical in functional form to the SVM. This was developed recently by Tipping to overcome the limitations of SVMs. However, these RVMs also follow lower bound on memorization as they adopt the same learning strategy as SVMs.

1.4 Work Summary

Existing growing and pruning algorithms do not order the hidden units optimally, resulting in a non-monotonic error versus number of hidden units curve. They also do not prevent linearly dependent hidden units from appearing in the final network. Previous proofs for the upper bounds on pattern storage are either overly complex, applicable only for the case of single output, or applicable to a particular type of network. There are

several large networks that have good generalization but are highly redundant, resulting in large space and time complexity. In this dissertation, we solve these problems by devising a new integrated growing and pruning technique, developing proof for an upper bound on pattern storage, and modeling large classifiers by smaller neural networks.

In Chapter 2, the basic principles behind growing and pruning methods are reviewed, along with pattern storage bounds. We also give a brief overview of modeling the large, redundant learning machines using feedforward networks. Chapter 3 first identifies problems in learning machines and summarizes the dissertation work, which aims at solving the problems stated. Chapter 4 presents techniques for combining growing and pruning methods in order to achieve smaller error for a given number of hidden units. In Chapter 5, a proof for an upper bound on pattern storage for feedforward networks is developed. Chapter 6 presents a technique for modeling SVMs using smaller networks. Two techniques have been devised for generating additional patterns which improve the modeling. Conclusions and discussion are presented in chapter 7.

CHAPTER 2

REVIEW OF LEARNING MACHINES AND STORAGE BOUNDS

In this chapter, we review basic principles behind the growing and pruning of MLPs. Pattern storage bounds, and their relevance to SVMs, are also discussed.

2.1 Multilayer Perceptron

Let $\{\mathbf{x}_p, \mathbf{t}_p\}_{p=1}^{N_v}$ be the dataset where $\mathbf{x}_p \in \Re^N$ is the input vector and $\mathbf{t}_p \in \Re^M$ is the desired output vector and N_v is the number of patterns. Figure 2.1 depicts a feedforward MLP, having one hidden layer with N_h nonlinear units and an output layer with M linear units. For the p^{th} pattern, the j^{th} hidden unit's net function and activation are

$$net_{pj} = \sum_{i=1}^{N+1} w_h(j,i) \cdot x_{pi} \quad 1 \leq p \leq N_v, 1 \leq j \leq N_h \tag{2.1}$$

$$O_{pj} = f(net_{pj}) = \frac{1}{1 + e^{-net_{pj}}} \tag{2.2}$$

Weight $w_h(j,i)$ connects the i^{th} input to the j^{th} hidden unit. Here the threshold of the j^{th} node is represented by $w_h(j, N+1)$ and by letting $x_{p,N+1}$ to be a constant, equal to one.

The k^{th} output for the p^{th} pattern is given by

$$y_{pk} = \sum_{i=1}^{N+1} w_{oi}(k,i) \cdot x_{pi} + \sum_{j=1}^{N_h} w_{oh}(k,j) \cdot O_{pj} \tag{2.3}$$

where $1 \leq k \leq M$. Here, $w_{oi}(k,i)$ is the weight connecting i^{th} input to k^{th} output and $w_{oh}(k,j)$ is the weight connecting j^{th} hidden unit to k^{th} output. There are N_v training patterns denoted by $\left\{(\mathbf{x}_p, \mathbf{t}_p)\right\}_{p=1}^{N_v}$ where each pattern consists in an input vector $\mathbf{x}_p$ and

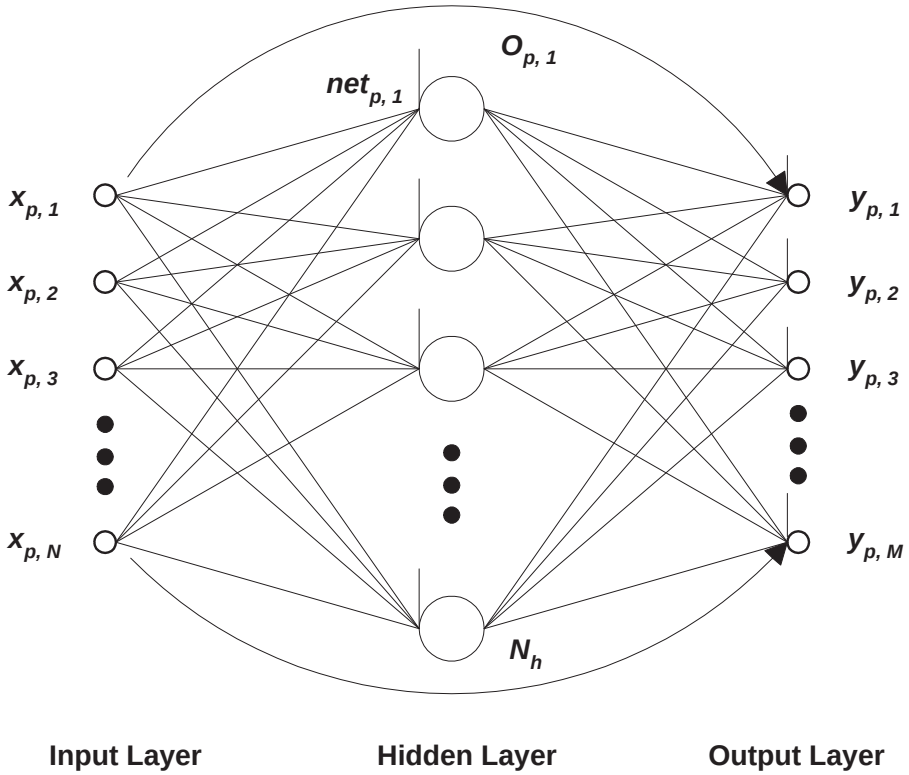


Figure 2.1. MLP with single hidden layer.

a desired output vector $\mathbf{t}_p$. For the p^{th} pattern, the N input values are x_{pi}, $(1 \leq i \leq N)$ and the M desired output values are t_{pk} $(1 \leq k \leq M)$.

The squared error for the p^{th} pattern is

$$E_p = \sum_{k=1}^{M} \left[t_{pk} - y_{pk}\right]^2 \tag{2.4}$$

In order to train a neural network for one epoch the mean squared error (MSE) for the i^{th} output unit is defined as

$$E(i) = \frac{1}{N_v} \sum_{p=1}^{N_v} \left[t_{pi} - y_{pi}\right]^2 \tag{2.5}$$

The overall performance of a MLP network, measured as MSE, can be written as

$$E = \sum_{i=1}^{M} E(i) = \frac{1}{N_v} \sum_{p=1}^{N_v} E_p \tag{2.6}$$

In many neural net training algorithms, the goal is to minimize E. Example training algorithms are backpropagation [46], Levenberg Marquart [56], Genetic Algorithms [57, 58, 59], cascade correlation [60] and Output Weight Optimization - Hidden Weight Optimization [61] which is described in the appendix A.

2.2 Structured Initialization

Initialization of a network plays an important role in the performance of MLP training, so proper initialization is critical. Assume that a set of MLPs of different numbers of hidden units, N_h, are to be designed for a given training data set. Let $E_f(N_h)$ denote the final training MSE, E, of an MLP with N_h hidden units.

Axiom 1: If $E_f(N_h) \geq E_f(N_h - 1)$, then the network having N_h hidden units is useless since the training resulted in a larger, more complex network with a larger or the same training error [13].

Three distinct types of network initialization are discussed. We give theorems related to these three methods, which give a basis for this paper. Delashmit [13] has, in his work, identified problems in MLP training and has stated them in the form of these theorems. Detailed discussions can be found in his work.

2.2.1 Randomly Initialized Networks

Randomly Initialized (RI) MLPs have no members of the set constrained with common initial weights or thresholds. In other words, the Initial Random Number Seeds (IRNS) of the networks are different. The different size networks designed using RI have statistically independent weights.

Theorem 1: If two initial RI networks (1) are the same size, (2) have the same training dataset and (3) the training dataset has more than one unique input vector, then the hidden unit basis functions are different for the two networks [13].

As the RI networks have random starting points, they have non-monotonic $E_f(N_h)$ curve. That is, for well-trained MLPs, $E_f(N_h)$ does not always decrease as N_h increases since the initial hidden unit basis functions are different. Let $E_i(N_h)$ denote the final training error for an RI network having N_h hidden units, which has been initialized using the i^{th} random number seed out of a total of N_r total seeds. Let $E_{av}(N_h)$ denote the average $E_i(N_h)$, that is

$$E_{av}(N_h) = \frac{1}{N_r} \sum_{i=1}^{N_r} E_i(N_h) \tag{2.7}$$

Delashmit [13] has developed a bound, using the Chebyshev inequality, on the probability that the average error for N_h hidden units, $E_{av}(N_h)$ is increasing and that the network with $N_h + 1$ hidden units is useless.

$$P(E_{av}(N_h + 1) > E_{av}(N_h)) \leq \frac{var(E_i(N_h + 1)) + var(E_i(N_h))}{2 \cdot N_r \cdot (m_{av}(N_h) - m_{av}(N_h + 1))^2} \tag{2.8}$$

where $var()$ represents the variance and $m_{av}(N_h)$ represent the average MSE for N_h hidden units.

2.2.2 Common Starting Point Initialized Networks

When a set of MLPs are Common Starting Point Initialized with Structured Weight Initialization (CSPI-SWI), each one starts with the same IRNS. Also, every hidden unit of the smaller network is initialized by the same weights and thresholds as the corresponding hidden unit of the larger network. Input to output weights, are also identical.

Theorem 2: If two initial CSPI-SWI networks (1) are the same size and (2) use the same algorithm for processing random numbers into weights, then they are identical [13].

As the size of the MLP is increased by adding new hidden units, a consistent technique for initializing these new units as well as the previous hidden units is needed for CSPI-SWI networks.

Theorem 3: If two CSPI-SWI networks are designed, the common subset of the initial hidden unit basis functions are identical.

The above theorems have been proved by Delashmit [13]. Unfortunately, growing with CSPI-SWI networks although better than RI, does not guarantee a monotonic $E_f(N_h)$ curve.

2.2.3 Dependently Initialized Networks

Here a series of different size networks are designed with each subsequent network having one or more hidden units than the previous network. Such Dependently Initialized (DI) networks have the advantage of using previous training on smaller networks. Trained smaller networks are used to initialize the weights and thresholds of larger networks.

Properties of DI networks: Let $E_{int}(N_h)$ denote the initial value of error during the training of an MLP with N_h hidden units and let N_{hp} denote the number of hidden units in the previous network. That is, $N_h - N_{hp}$ new hidden units are added to a well-trained smaller network, to initialize the larger one. Then [13]:

1. $E_{int}(N_h) < E_{int}(N_{hp})$
2. $E_f(N_h) \leq E_f(N_{hp})$
3. $E_{int}(N_h) = E_f(N_{hp})$

As seen in property 2, DI networks, have a monotonically decreasing $E_f(N_h)$ versus N_h curve. Unfortunately, there is no guarantee that a $E_f(N_h)$ sample represents a global minimum.

2.2.4 Pruned Networks

One of the most popular regularization procedures in neural nets is done by limiting the number of hidden units [62]. Pruning generates series of different size networks

starting from a large network and each subsequent network having one less hidden unit than the previous network. In other words, the hidden units will be ordered according to their importance and the pruning will sequentially delete them one at a time starting will the least important hidden unit. Hidden units of bigger networks will always be supersets of those of smaller networks.

Axiom 2: If $E_f(N_h)$ is the error value of a network (having N_h hidden units) in a sequence of pruned networks, then $E_f(N_h) \leq E_f(N_h - k)$ for $1 \leq k \leq N_h$.

The above Axiom indicates that the pruning produces monotonically decreasing $E_f(N_h)$ versus N_h curve.

2.3 Generation of Feedforward Networks

According to the SRM principle, the best network in an ensemble of MLPs is the one that has minimum guaranteed risk [63, 31]. However, the guaranteed risk is an upper bound for the generalization error $E_{val}(N_h)$. Hence the best network is the one for which $E_{val}(N_h)$ is the smallest. In this section we present two general approaches or design methodologies for generating sequences of networks having monotonic $E_f(N_h)$ curves.

2.3.1 Growing Methods for Training

In growing methods, we start by training a small network (with small N_h) and then successively train larger networks by adding more hidden units, producing a final training MSE versus N_h curve, $E_f(N_h)$. After training a sequence of these different size DI networks, we calculate the validation MSE versus N_h curve $E_{val}(N_h)$. The validation error will be the key to decide the network size needed. Growing methods, which are motivated by the properties of DI networks, are collectively denoted as Design Methodology-1 (DM-1).

Given training data $\{\mathbf{x}_p, \mathbf{t}_p\}_{p=1}^{N_v}$, our growing algorithm, is described as follows. First, we train a linear network (no hidden units, $N_h = 0$) using the OWO algorithm and the network obtained is represented by $\mathcal{W}_{DM1}^0$. Then we successively add a few hidden

units ($N_\epsilon = 5$ in our case[1]) and re-train the network. During re-training, for the first few iterations (say 20% of the total iterations), we train only the newly added hidden units using OWO-HWO [61] as described in appendix A. Then during the remaining iterations, we train all the weights and thresholds. The network obtained at the end of training is represented by $\mathcal{W}_{DM1}^{N_h}$. Also at each step of growing, we validate the network obtained at that point with validation dataset $\{\mathbf{x}_p^{val}, \mathbf{t}_p^{val}\}_{p=1}^{N_v^{val}}$ to help decide upon the final network size. Obviously, the input and output vectors are of the same dimension as that of the training data. Let $E_{val}(\mathcal{W}_{DM1}^{N_h})$ be the validation error for a DM-1 network with N_h hidden units. The best network among the sequence of different size networks can be chosen as

$$\mathcal{W}_{DM1}^{N_h^{opt}} = \text{argmin } E_{val}(\mathcal{W}_{DM1}^{k}) \tag{2.9}$$

where $\mathcal{W}_{DM1}^{N_h^{opt}}$ represents the DM-1 network with optimal number of hidden units N_h^{opt} and N_{hFinal} represents the maximum number of hidden units at the end of growing. This number should be conveniently large so that the best network falls within the sequence of grown DM-1 networks. The validation error is the mean squared error on the validation dataset, given by

$$E_{val} = \frac{1}{N_v^{val}} \sum_{p=1}^{N_v^{val}} \sum_{k=1}^{M} \left(t_{pk}^{val} - y_{pk}^{val} \right)^2 \tag{2.10}$$

where t_{pk}^{val} and y_{pk}^{val} are the desired and actual outputs for k^{th} unit and p^{th} pattern of the validation dataset.

In order to validate the performance of the method, we repeat the growing procedure several times with different random numbers initializing the network. Then the average MSEs are calculated, which gives the expected values of the MSEs for each size

[1]It is shown by Lehtokangas [64] that adding multiple hidden units at a time and training together is much more efficient than adapting many single units trained independently. Hence in this dissertation, we have fixed the growing step size to be five.

network. Also, to obtain a measure for the confidence interval of the method, we calculate the standard deviations of the MSEs. The average MSEs are given by

$$\mathcal{M}(E_{val}) = \frac{1}{N_r} \sum_{i=1}^{N_r} E_{val}^i \tag{2.11}$$

where E_{val}^i are the validation MSEs for the i^{th} random initial network and N_r is the number of random initial networks. The standard deviations of the MSEs are given by

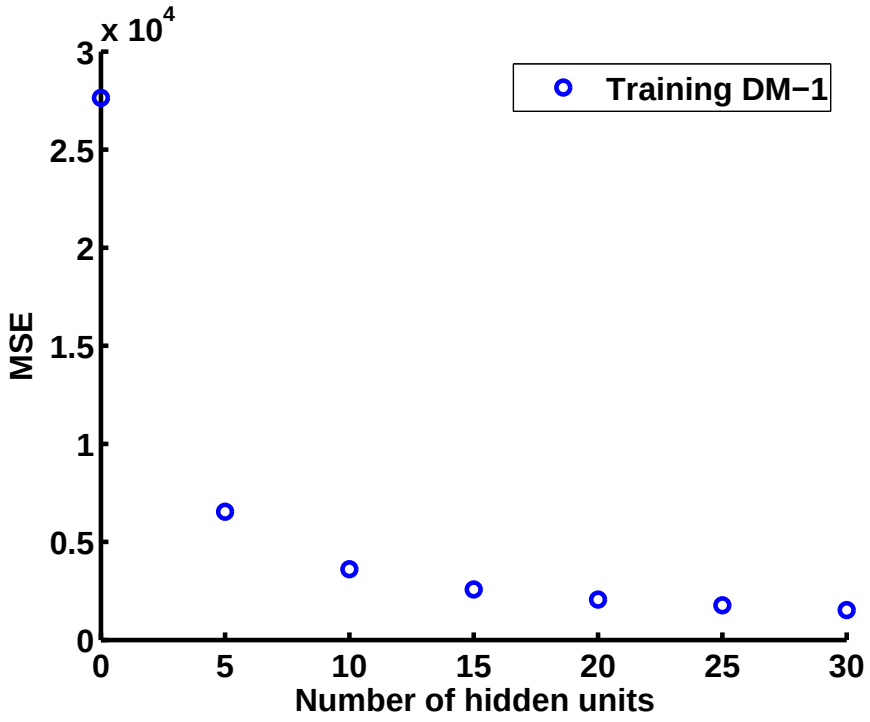
$$\mathcal{SD}(E_{val}) = \left[\frac{1}{N_r} \sum_{i=1}^{N_r} (E_{val}^i - \mathcal{M}(E_{val}))^2 \right]^{1/2} \tag{2.12}$$

Fig. 2.2 shows sample plots of the training and validation MSEs as a function of number of hidden units (N_h) for DM-1 with $N_r = 10$. Note that in this method, there will not be any intermediate size networks as the step size used to grow the network is equal to five. A more detailed analysis on the plots of means and standard deviations of the design methodologies will be made in chapter 4 and hence at this point we have not shown these plots.

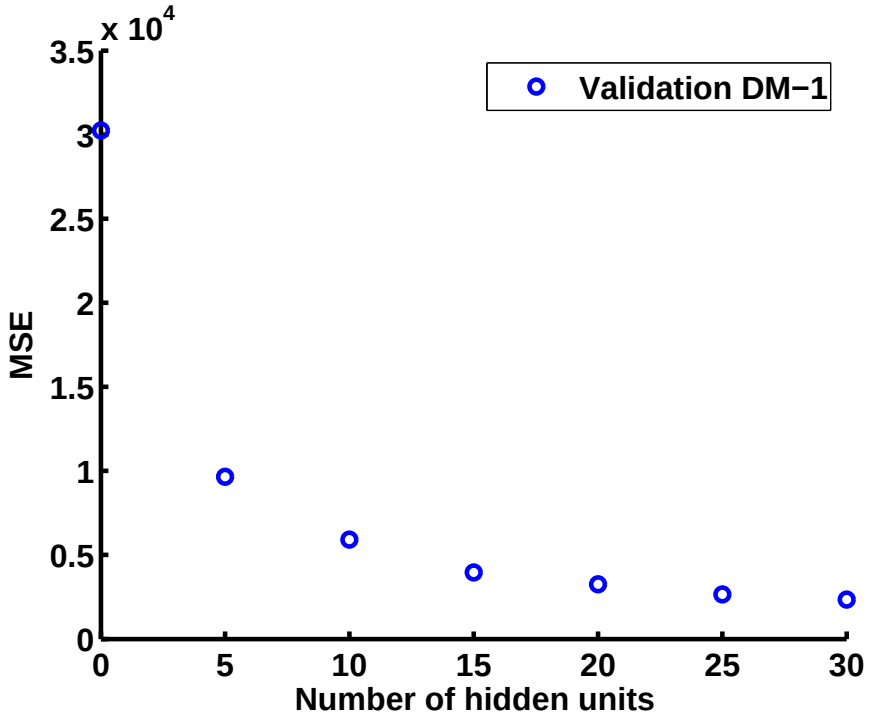
In appendix C, we discuss the constructive backpropagation and cascade correlation training methods, which are additional examples of DM-1 growing methods.

2.3.2 Pruning Methods

We call the pruning techniques Design Methodology 2 (DM-2). The goal of pruning is stepwise optimal ordering of hidden units. Here we prune a large network in order to produce a monotonic $E_f(N_h)$ curve. The pruning procedure [25] is explained in appendix B. Pruning does not change the hidden unit weights but just reorders the hidden units for a better performance. During pruning, the orthonormalization matrix $\mathbf{A} = \{a_{mk}\}$ for $1 \leq m \leq N_u$ and $1 \leq k \leq m$ is saved, where $N_u = N + N_h + 1$ is the total number of basis functions. Let $j(m)$ be an integer valued function that specifies the order in which the raw basis functions x_k are processed into orthonormal basis functions,



(a)



(b)

Figure 2.2. MSEs of Speech dataset (a) Training (b) Validation.

x'_k. The m^{th} orthonormal basis is calculated using the previous $m-1$ ordered bases. The $j()$ function defines the structure of the new hidden layer. If $j(m) = k$, then the m^{th} unit of the new structure comes from the k^{th} unit of the original structure.

The following equation gives the m^{th} orthonormal basis function.

$$x'_m = \sum_{k=1}^{m} a_{mk} \cdot x_{j(k)} \tag{2.13}$$

for $1 \leq m \leq N_u$. The network output is given by the equation

$$y_{pk} = \sum_{i=1}^{N_u} w'_o(k,i) \cdot x'_i \tag{2.14}$$

where the orthonormal weights $w'_o(k,i)$ are

$$w'_o(k,i) = \sum_{q=1}^{i} a_{iq} \cdot c(k, j(q)) \tag{2.15}$$

The error equation corresponding to the network with N_{hd} hidden units is given by

$$E_{trn}(N_{hd}) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{k=1}^{M} [t_{pk} - y_{pk}(N_{hd})]^2 \tag{2.16}$$

Here $y_{pk}(N_{hd})$ is the output of the network with N_{hd} hidden units, given by

$$y_{pk}(N_{hd}) = \sum_{i=1}^{N+1+N_{hd}} w'_o(k,i) \cdot x'_i \tag{2.17}$$

One Pass Validation

Given the matrix **A** and the MLP network with ordered hidden units, we wish to generate the validation error versus N_h curve $E_{val}(N_h)$ from the validation dataset $\{\mathbf{x}_p^{val}, \mathbf{t}_p^{val}\}_{p=1}^{N_v^{val}}$. For each pattern, we augment the input vector with a constant one and

the hidden activations. So the augmented input vector is $\mathbf{x}_p^{val} \leftarrow [\mathbf{x}_p^{val^T}, 1, \mathbf{O}_p^{val^T}]^T$. Then the augmented vector is converted into orthonormal basis functions by the transformation

$$x_p^{val'}(m) = \sum_{k=1}^{m} a_{mk} \cdot x_p^{val}(k), \quad \text{for} \quad 1 \leq m \leq N_u \tag{2.18}$$

In order to get the validation error for all size networks in a single pass through the data, we use the following strategy:

Let $y_{pi}^{val}(m)$ represent the i^{th} output of the network having m hidden units for the p^{th} pattern, let $E_{val}(m)$ represent the mean square error of the network for validation with m hidden units. First, the linear network output is obtained and the corresponding error is calculated as follows:

$$\begin{aligned} y_{pi}^{val}(0) &= \sum_{k=1}^{N+1} w_o'(i,k) \cdot x_p^{val'}(k), \quad \text{for} \;\; 1 \leq i \leq M \\ E_{val}(0) &\leftarrow E_{val}(0) + \sum_{i=1}^{M} [t_{pi}^{val} - y_{pi}^{val}(0)]^2 \end{aligned} \tag{2.19}$$

Then for $1 \leq m \leq N_h$, the following two steps are performed:

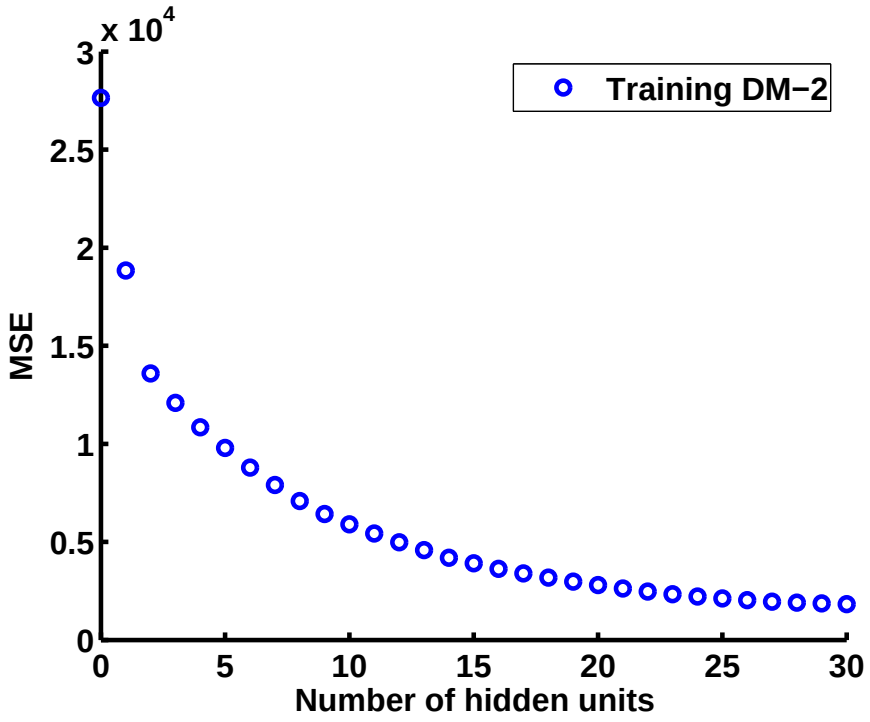
- For $1 \leq i \leq M$ compute the i^{th} output for m hidden units as

$$y_{pi}^{val}(m) = y_{pi}^{val}(m-1) + w_{oh}'(i,m) \cdot O_p^{val'}(m) \tag{2.20}$$

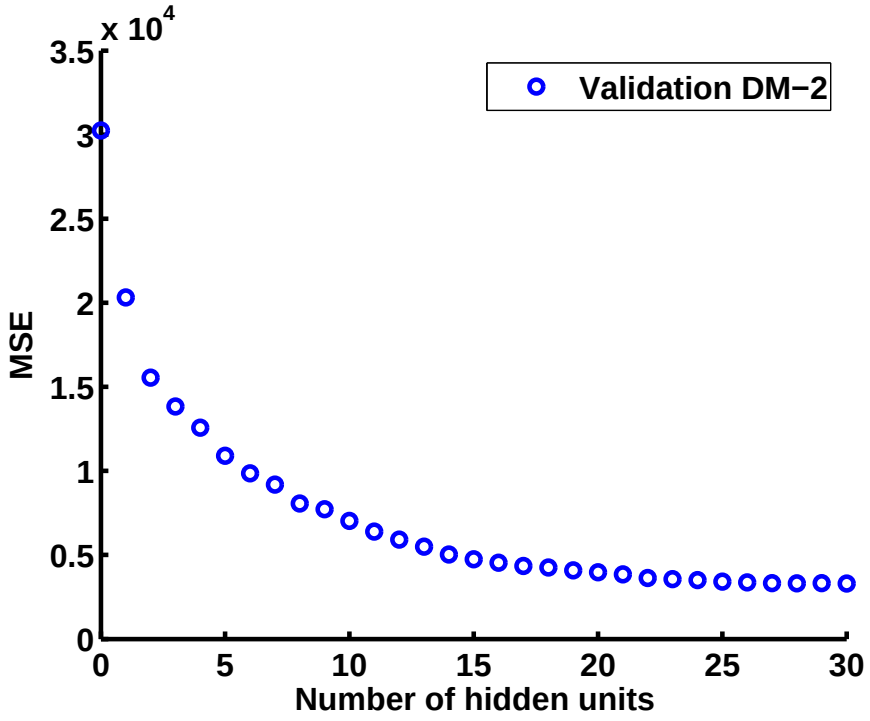
- Update the validation error of the network with m hidden units as

$$E_{val}(m) \leftarrow E_{val}(m) + \sum_{i=1}^{M} [t_{pi}^{val} - y_{pi}^{val}(m)]^2 \tag{2.21}$$

where $w_{oh}'(i,m)$ is the orthonormal weight connecting m^{th} hidden unit to i^{th} output. Apply equations 2.18, 2.19, 2.20 and 2.21 for $1 \leq p \leq N_v$ and get the total validation



(a)



(b)

Figure 2.3. MSEs of Speech dataset (a) Training (b) Validation.

error over all the patterns for each size network. Then these error values should be normalized as

$$E_{val}(m) \leftarrow \frac{E_{val}(m)}{N_v^{val}}, \quad \text{for} \;\; 0 \leq m \leq N_h \tag{2.22}$$

Thus we generate the validation error versus the network size curve in one pass through the validation dataset.

Fig. 2.3 shows sample plots of the training and validation MSEs as a function of number of hidden units (N_h) for DM-2 with $N_r = 10$.

2.4 Review of Pattern Storage Bounds

A feedforward network is said to have memorized a dataset if for every pattern, the network outputs are exactly equal to the desired outputs. Storage capacity of a feedforward network is the number (N_v) of distinct input vectors that can be mapped, exactly, to the corresponding desired output vectors resulting in zero error.

The lower bound on memorization (or the upper bound on number of hidden units) is stated in the following theorem which is due to Sartori and Antsaklis [33] and Huang [34].

Theorem 4: *For a feedforward network with N inputs, M outputs and N_h hidden units with arbitrary bounded nonlinear activation functions, at least N_h distinct patterns can be memorized.*

This can be expressed as

$$N_v \geq N_h \tag{2.23}$$

for networks having no bypass weights or output thresholds. For networks having bypass weights and the output thresholds, we generalize this to get

$$N_v \geq (N + N_h + 1) \tag{2.24}$$

Researchers are generally aware of the upper bound on number of distinct patterns N_v that can be memorized by a feedforward nonlinear network.

Theorem 5 *For a feedforward network with N inputs, M outputs and N_h hidden units with arbitrary bounded nonlinear activation functions, the number of distinct patterns that can be memorized obeys the inequality*

$$N_v \leq \left\lfloor \frac{N_w}{M} \right\rfloor \tag{2.25}$$

where N_w is the number of weights in the network.

For example, the upper bound on the number of hidden units, derived by Elisseeff and Moisy [39] is based upon (2.25). They assume that the activation functions have continuous derivatives and have finite limits L^- in $-\infty$ and L^+ in $+\infty$.

2.5 Support Vector Machines

Support Vector Machines can be used to solve both regression and classification problems. In this dissertation, we use SVMs to solve two-class classification problems. Here, we briefly review relevant properties of SVMs [30, 31]. Let the dimension of feature space be h_{svm}, which is also the number of Support Vectors (SVs). Note that h_{svm} is equivalent to number of hidden units in an MLP. Fig. 2.4 shows a diagram of an SVM.

Let $\{\mathbf{x}_p, d_p\}_{p=1}^{N_v}$ be the training dataset. Let $\{\phi_j(\mathbf{x})\}_{j=1}^{h_{svm}}$ denote nonlinear transformation from input space to feature space. In our case, they represent the Radial Basis Functions (RBF).

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_j\|^2\right) \tag{2.26}$$

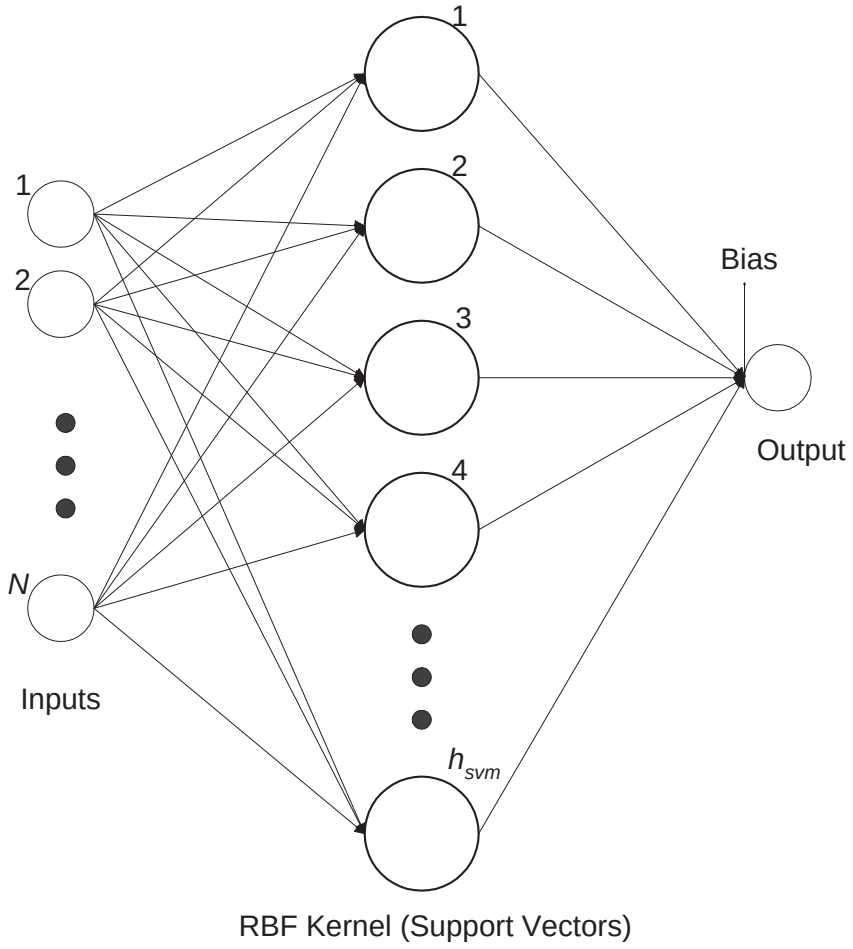


Figure 2.4. Support Vector Machine with RBF Kernel.

for $1 \leq j \leq h_{svm}$. These $\phi_j(\mathbf{x})$ are equivalent to the hidden unit activation functions in case of MLPs. The output of the SVM is given by

$$s_p = \sum_{j=1}^{h_{svm}} w_j \phi_j(\mathbf{x}_p) + b \tag{2.27}$$

The main structural differences between MLPs and SVMs are

- Unlike MLPs, bypass weights are not present in case of SVMs (weights connecting inputs directly to outputs)
- There is only one output for SVMs whereas MLPs can handle multiple outputs

The decision hyperplane constructed by SVM for a given dataset is as follows:

$$\sum_{j=1}^{h_{svm}} w_j \phi_j(\mathbf{x}) + b = 0 \tag{2.28}$$

where $\{w_j\}_{j=1}^{h_{svm}}$ denotes a set of linear weights connecting the feature space to the output space, and b is the bias. A decision is made on the given input depending on the output of the SVM. If it is greater than zero, the pattern lies on the right side of the hyperplane and is classified to be of class one and if it is less than zero, the pattern lies on the left side of the hyperplane and is classified to be of class two. For a SVM, patterns that are SVs generate zero squared error. Therefore the SVM memorizes $N_v = h_{svm}$ patterns. This corresponds to the lower bound on memorization of Theorem 4 (see equation (2.23)).

The RBF type inner-product kernels as shown in Fig. 2.4 are used which are denoted by $K(\mathbf{x}, \mathbf{x}_i)$ and defined by

$$K(\mathbf{x}, \mathbf{x}_i) = \sum_{j=0}^{h_{svm}} \phi_j(\mathbf{x}) \phi_j(\mathbf{x}_i) \tag{2.29}$$

Now, we may use the inner-product kernel $K(\mathbf{x}, \mathbf{x}_i)$ to construct the optimal hyperplane in the feature space as

$$\sum_{i=1}^{N} \alpha_i d_i K(\mathbf{x}, \mathbf{x}_i) = 0 \tag{2.30}$$

where α_i are the Lagrange multipliers [65].

2.5.1 Structural Risk Minimization (SRM) Principle

The goal in SVMs is to solve a supervised learning problem by realizing the best generalization performance of a machine, which is obtained by finding the VC dimension [31] where the minimum of the guaranteed risk (sum of training error and confidence interval) occurs. In MLP, this corresponds to finding the appropriate number of hidden units which minimizes the validation error. This is shown in the Fig. 2.5.

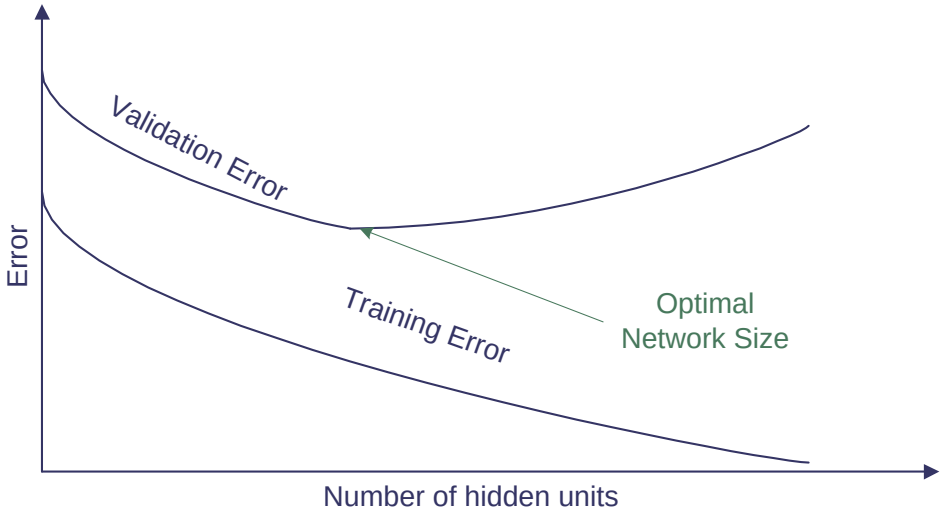


Figure 2.5. Structural Risk Minimization in MLP.

2.5.2 Memorization in Support Vector Machines

Let $\{\mathbf{x}_p, d_p\}_{p=1}^{N_v}$ be the training dataset, where $\mathbf{x}_p \in \Re^N$, is the input pattern for the p^{th} example, d_p is the corresponding class number and N_v is the number of patterns. Consider designing a Lagrangian SVM with RBF type inner product kernels denoted by $K(\mathbf{x}, \mathbf{x}_i)$. In order to solve for the decision surface, slack variables ξ_p are introduced into the definition of separating hyperplane as shown [65]

$$d_p(\mathbf{w}^T \phi_p + b) \geq 1 - \xi_p \tag{2.31}$$

for $1 \leq p \leq N_v$. Here, ϕ_p denotes the kernel function, $\mathbf{w}$ is the linear weights in the feature space and b is the bias. The ξ_p are called slack variables; they measure the deviation of a data point from the ideal condition of pattern separability. Using the method of Lagrange multipliers, a dual problem is formulated and solved for the Lagrange multipliers which can be used to determine the optimal hyperplane.

$\phi_j(\mathbf{x})$ is equivalent to the activation function of the j^{th} hidden unit. Support vectors are those particular data points that satisfy equation (2.31) precisely even if $\xi_p > 0$. These support vectors along with the corresponding weights will be used to classify the data

patterns. Hence the SVM memorizes the number of patterns equal to the number of support vectors or the number of basis functions. Therefore, SVMs follow the lower bound on pattern storage, seen in Theorem 4.

2.6 AdaBoost-Stump

Boosting is a committee machine having static structure where weak learners are combined to achieve a classifier with arbitrarily high accuracy. A weak learner is also referred to as a *base hypothesis* and the final voted hypothesis as *combined hypothesis*. Boosting can be implemented in three different ways: boosting by filtering, boosting by subsampling and boosting by reweighting [65]. In this dissertation, we use boosting by subsampling, in particular the AdaBoost algorithm.

First, the distribution is assumed uniform over all training samples and a weak learner is trained. This weak classifier correctly classifies a fraction of the training samples. The error is measured with respect to the distribution. Then the distribution is updated such that the density of misclassified patterns should increase and that of correctly classified patterns should decrease. Another weak learner is trained on the new distribution. This procedure continues until a maximum number of weak classifiers are trained and the boosting machine combines all the weak classifiers into a single final strong classifier.

Summary of AdaBoost is given below: Given training samples $\{\mathbf{x}_i, d_i\}_{i=1}^{N_v}$, distribution $\mathcal{D}$ over N_v labeled examples, weak learning model, an integer N_{it} specifying the number of iterations of the algorithm, initialize the set $\mathcal{D}_1(i) = 1/N_v$ for all i. Repeat the following steps for $n = 1, 2, \cdots, N_{it}$

1. Execute the weak learning model using the distribution $\mathcal{D}_n$ and get the base hypothesis $\mathcal{H}_n : \mathbf{x}_i \rightarrow d_i$

2. Calculate the error of hypothesis $\mathcal{H}_n$:

$$\epsilon_n = \sum_{i:\mathcal{H}_n(\mathbf{x}_i)\neq d_i} \mathcal{D}_n(i) \tag{2.32}$$

3. Set $\beta_n = \frac{\epsilon_n}{1-\epsilon_n}$
4. Update the distribution $\mathcal{D}_n$:

$$\mathcal{D}_{n+1}(i) = \frac{\mathcal{D}_n(i)}{Z_n} \times \begin{cases} \beta_n, & \text{if } \mathcal{H}_n(\mathbf{x}_i) = d_i; \\ 1/\beta_n, & \text{otherwise.} \end{cases} \tag{2.33}$$

where Z_n is a normalization constant

The final hypothesis is

$$\mathcal{H}(\mathbf{x}) = \arg\max_{d\in\mathcal{D}} \sum_{n:\mathcal{H}_n(\mathbf{x})=d} \left(\log \frac{1}{\beta_n}\right) \cdot \mathcal{H}_n(\mathbf{x}) \tag{2.34}$$

The details of AdaBoost algorithm can be found in [65] [66]. Relation of AdaBoost to Bias-Variance Theory is explained by Schapire *et al.* in their work [67]. They have explained the improvements achieved by voting classifier based on separating the expected error into a bias term and a variance term. They have presented results of several bias-variance experiments on different artificial datasets and have concluded that boosting can perform poorly only if there is insufficiency in the available training data compared to the base hypothesis complexity or the training errors of base hypothesis increases too quickly.

CHAPTER 3

PROPOSED WORK

In this chapter, we describe the problems or requirements that are identified and propose a solution for each of them.

3.1 Problems

3.1.1 Sensitivity of Trained Networks to Initial Conditions

Designing DI networks (DM-1) results in monotonic error versus network size curves, however, there is no guarantee that the error for each size network is minimized. Also the variance of the error on different random initialization is high, which indicates that the technique is very sensitive to the initial conditions. One of our goals is to minimize the sensitivity of the network to initialization.

In pruning techniques, (DM-2), the error versus network size curves are also monotonic and the variance of the error over different random initialization is smaller than DM-1. However, training a large network might drive some of the hidden units to saturation and the larger networks will not be optimal. This calls for a better technique where we can minimize errors for all network sizes while maintaining low error variance over different initialization.

3.1.2 Unclear Upper Bound on Pattern Storage

Although several researchers have worked on the upper bound on pattern storage, there is no clear statement and proof of the upper bound for the case of multiple inputs and multiple outputs and arbitrary hidden unit activation functions.

3.1.3 Redundancy in Large Learning Machines

As discussed in the previous chapter, SVMs have large numbers of support vectors, which makes them bulky and slow. In other words, the space (number of network parameters) and time (computation time) complexities are very high in case of SVMs. Compact and well generalized networks are required that can perform as well or better than SVMs. Such compact networks should eliminate the redundancy in the SVMs and speed up the processing of new data.

3.2 Proposed Goals and Tasks

The goals in the proposed research are to (1) improve and combine growing and pruning approaches, (2) develop a new proof of an upper bound on pattern storage, and (3) develop a method for generating compact models of large classifiers.

3.2.1 Combined Growing and Pruning Approaches

We propose a combined growing and pruning approach which is less sensitive to the initialization and results in smaller error on all network sizes. Its performance will be evaluated by comparing with the growing and pruning techniques presented in the previous chapter. A comparison with the existing benchmark growing techniques will also be presented. We also will discuss improvements of this technique that can lead to a solution where saturated hidden units can be removed dynamically as new hidden units are added.

We also extend the work to classification case and give the similar comparisons as the approximation case.

3.2.2 Upper Bound on Pattern Storage

We propose a proof by contradiction of the upper bound on pattern storage for the case of arbitrary hidden unit activations. We discuss the pitfalls of a direct approach.

The tightness and validity of the derived bound will be verified by a simple simulation experiment.

Using the fact that large SVMs memorizes training data inefficiently, we propose to use the upper bound of Theorem 5 to predict more appropriate sizes. Then, a method of concentrating the performance of large learning machines like SVMs into much smaller feedforward networks is proposed. The advantages of training an approximation network and converting it into classification network will be discussed.

3.2.3 Compact Modeling of Large Learning Machines

We extend the idea of memorization and model a network based on a better performing learning machine. Methods of generating additional training data using a trained machine and their effects on the performances of other machines will be presented.

3.2.4 Overview of Proposed Research

Fig. 3.1 shows the block diagram of the work proposed for this dissertation. It is shown here to help understand the connectivity of our research modules. By developing (1) a new network sizing approach, (2) an effective training algorithm and (3) a method for generating extra training data, we hope to produce smaller, more capable networks.

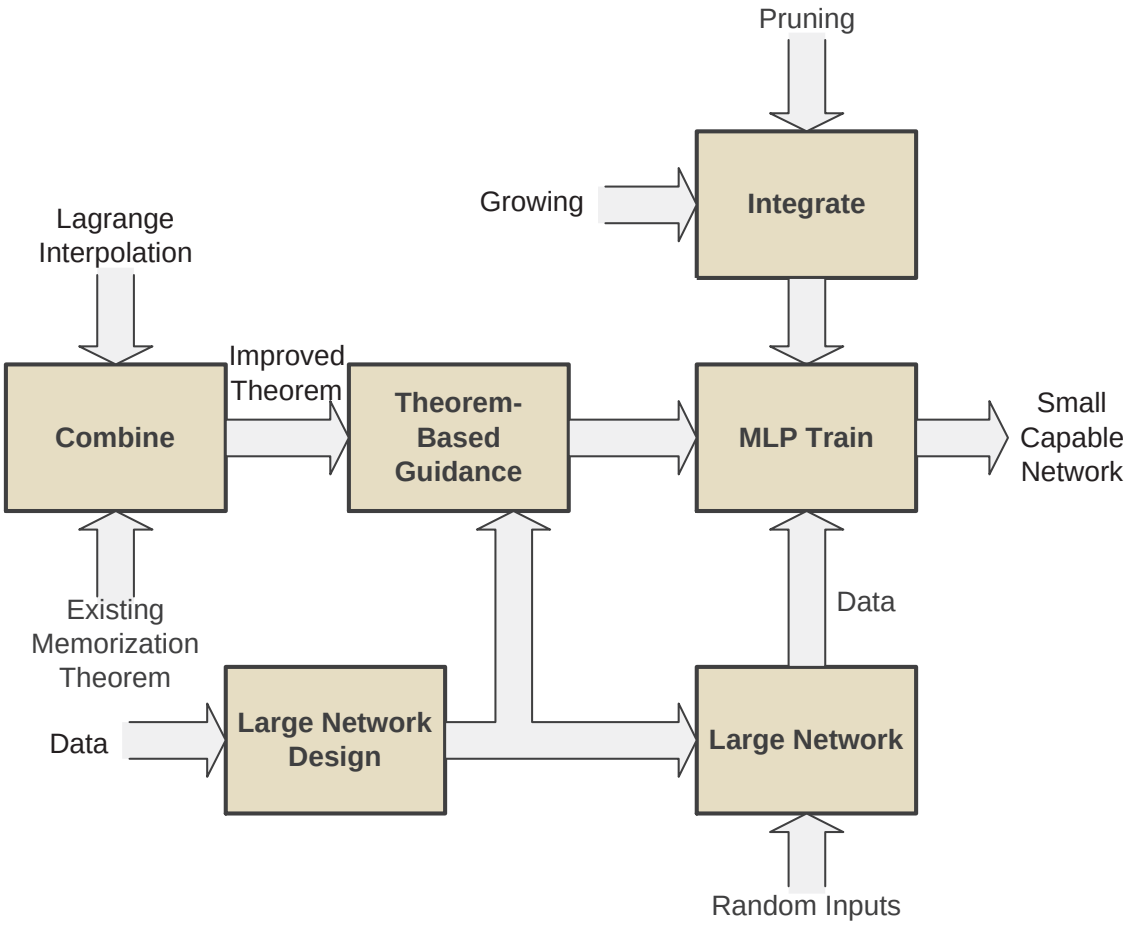


Figure 3.1. Overview of Proposed Research.

CHAPTER 4

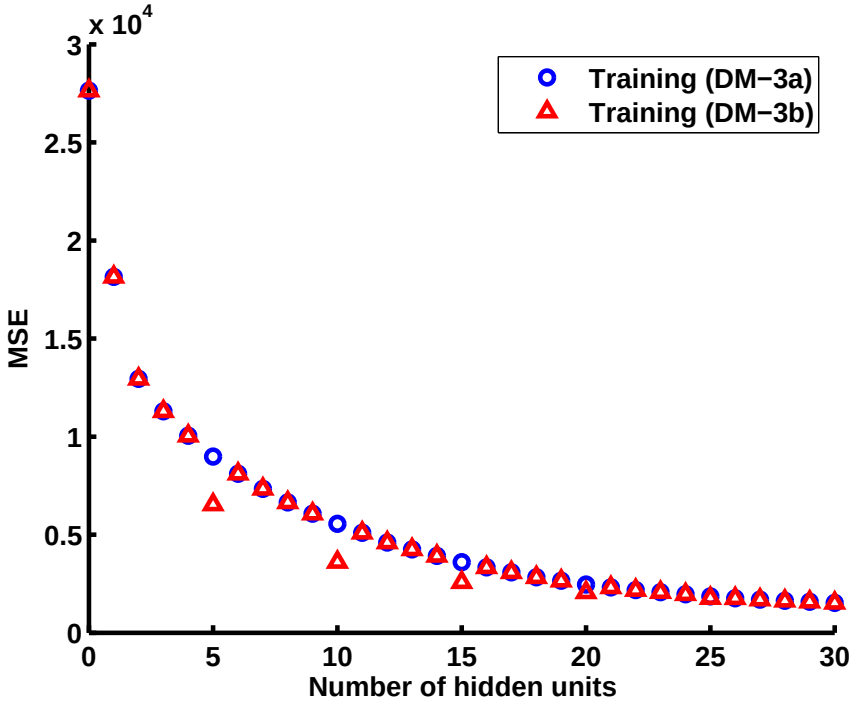
COMBINED GROWING AND PRUNING

In this chapter, we discuss work done on task 3.2.1. We have presented DM-3 and DM-4, which are combination of growing and pruning methods. We evaluate the performances of these techniques and compare them to existing growing methods. Extensive simulation results on various datasets have been tabulated. Further, we also extend the technique to the classification case and show the simulation results.

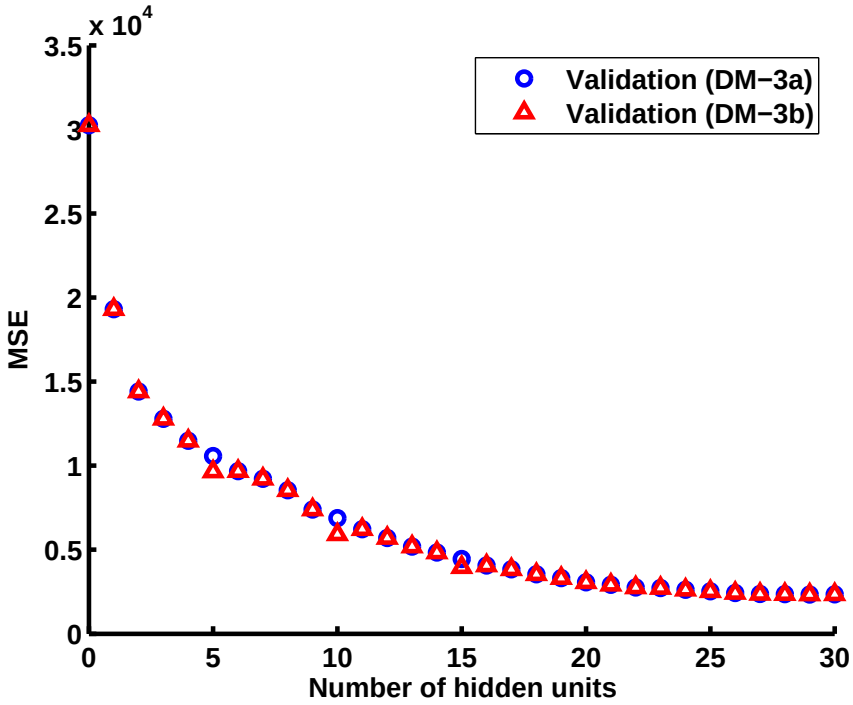
4.1 Design Methodologies for Feedforward Networks

4.1.1 Pruning a Grown Network

The DI network growing approach (DM-1) generates monotonic $E_f(N_h)$ curves. However, hidden units added earlier in the process tend to reduce the MSE more than those added later. Unfortunately, there is no guarantee that an $E_f(N_h)$ sample represents a global minimum, there is no guarantee that the hidden units are ordered properly, and useless hidden units can occur even for small values of N_h. In Design Methodology 3 (DM-3), we attempt to solve these problems by (1) Performing growing as in DI network (DM-1), and (2) Performing ordered pruning of the final network (DM-2). This forces the grown hidden units to be stepwise optimally ordered. This method works better than pruning a large trained network because the growing by DI network approach would produce sets of hidden units that are optimally placed with respect to the previously trained hidden units. Pruning the grown network (DM-3) always produces a monotonic $E_f(N_h)$ curve. Let us denote the DM-3 network for N_h hidden units by $\mathcal{W}_{DM3}^{N_h}$.



(a)



(b)

Figure 4.1. MSEs, Speech dataset.

An alternative solution would be to save the intermediate grown networks and to select the best performing network obtained either by DM-1 or DM-2 step. In other words,

$$\mathcal{W}_{DM3}^{k} = \text{argmin}_{dm \in \{DM1, DM2\}} E_{val}\Big\{\mathcal{W}_{dm}^{k}\Big\} \tag{4.1}$$

for $1 \leq k \leq N_{hFinal}$, and the optimal network is obtained by picking the network with minimum validation MSE.

$$\mathcal{W}_{DM3}^{N_h^{opt}} = \text{argmin}_{k}\ E_{val}\Big\{\mathcal{W}_{DM3}^{k}\Big\} \tag{4.2}$$

One problem with this method is that the curve $E_{trn}(N_h)$ and $E_{val}(N_h)$ are not going to be monotonic, but the MSEs for smaller number of hidden units will improve over the case where there is no minimum operator. Let us denote the former case, without the minimum operator as DM-3a and the later case, with the minimum operator as DM-3b. Fig. 4.1 shows the training and validation MSEs for the two cases on speech dataset. This data set for estimating phoneme likelihood functions in speech, has 39 inputs and 117 outputs. The speech samples are first pre-emphasized and converted into the frequency domain via the DFT. The data is passed through Mel filter banks and the inverse DFT is applied on the output to get Mel-Frequency Cepstrum Coefficients (MFCC). Each of MFCC(n), MFCC(n)-MFCC(n - 1) and MFCC(n)-MFCC(n - 2) would have 13 features, which results in a total of 39 features. The desired outputs are likelihoods for the beginning, middle, and ends of 39 phonemes. It is clearly seen that monotonicity has been lost in DM-3b. However, as the scale on the y-axis is very large, the actual value of the MSE is significantly lower for some networks in DM-3b.

Fig. 4.2 shows the block diagram of the DM-3b procedure.

Unfortunately, DM-3 does not guarantee that all the hidden units for every network are useful. Some of the hidden units may be saturated and increase complexity while contributing very little to the generalization of the network. Also these saturated hidden units might affect the learning of other hidden units.

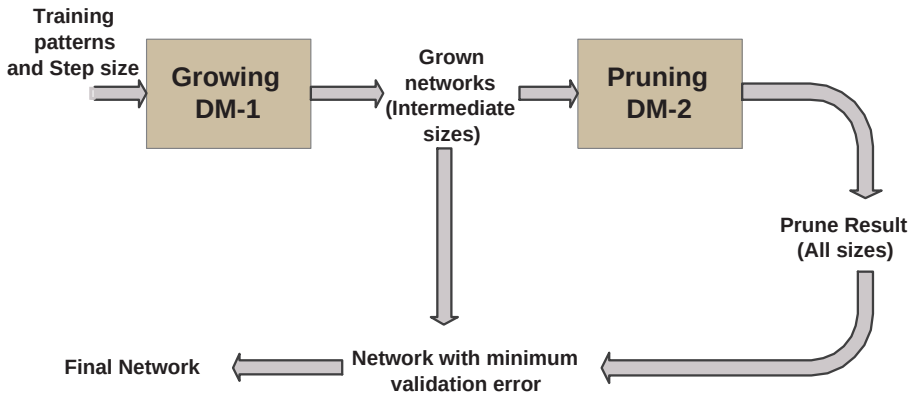


Figure 4.2. Block diagram showing the flow of DM-3b.

If there are saturated hidden units or the hidden units that are linearly dependent, those units need to be removed. This is achieved using the the simultaneous growing and pruning approach presented below, where the hidden units are pruned in every step of growing.

4.1.2 Simultaneous Growing and Pruning

The problems faced in DM-3 approaches can be solved by our last approach called DM-4, where we simultaneously grow and prune the hidden units of the network. During pruning, we delete the hidden units which contribute less than $\eta\%$ to the network performance (i.e., less than $\eta\%$ change in MSE when removed). This technique eliminates any saturated hidden units and in turn helps in adding new hidden units with different starting points. Hence the the technique is termed a Pseudo-Genetic approach.

As in DM-3, we can save all the intermediate networks and select the best performing network obtained either by the DM-1 or DM-2 step. Note that in DM-4, we have lot more networks in the pool and we should choose the one with the minimum validation error. Again, we can denote the case with no minimum operator by DM-4a and the one with the minimum operator by DM-4b.

Fig. 4.3 shows the block diagram of the DM-3b procedure.

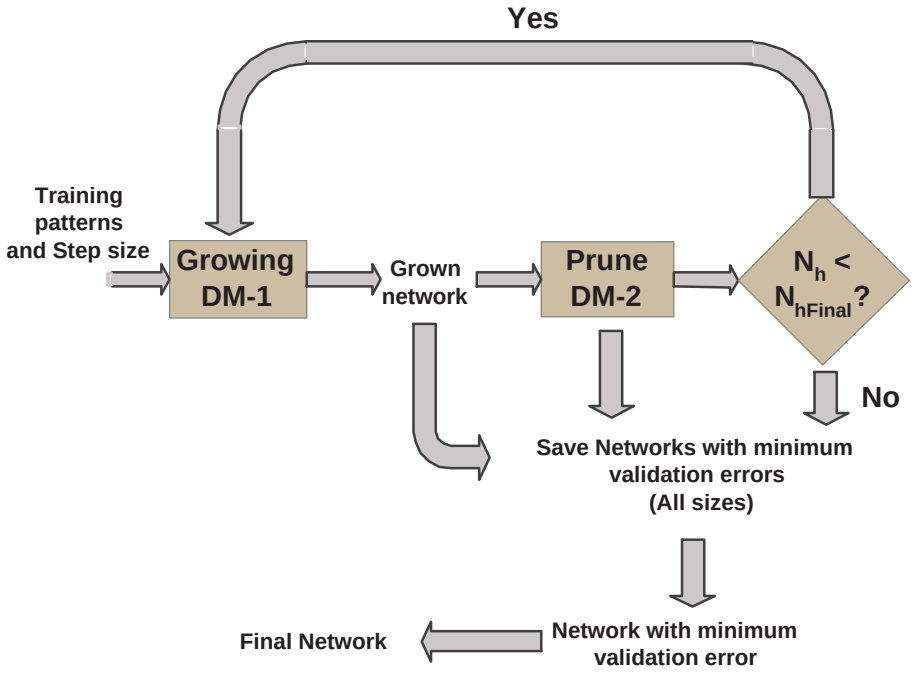


Figure 4.3. Block diagram showing the flow of DM-4b.

Algorithm (DM-4b) Given the training data and maximum number of hidden units N_{hFinal}

1. Train a linear network to get $w_{oi}(k,i) \in \mathcal{W}^0_{DM4}$ and $w_{hi}(j,i) \in \mathcal{W}^0_{DM4}$ (here $N_h = 0$), where $1 \leq k \leq M$, $1 \leq j \leq N_h$ and $1 \leq i \leq N$.
2. Add N_ϵ hidden units with random initial input weights and zero output weights ($N_h = N_h + N_\epsilon$).
3. Train only the new hidden units for 20% of the iterations and the entire network for the remaining 80% of the iterations. We will get a new set of $w_{oi}(k,i) \in \mathcal{W}^{N_h}_{DM4}$, $w_{oh}(k,j) \in \mathcal{W}^{N_h}_{DM4}$ and $w_{hi}(j,i) \in \mathcal{W}^{N_h}_{DM4}$, where $1 \leq k \leq M$, $1 \leq j \leq N_h$ and $1 \leq i \leq N$.
4. Prune the network: Order the hidden units using the Schmidt procedure explained in appendix B. Then compute the errors for each size of the network ($E_f(N_h)$) and delete those hidden units which results in less than $\eta = 1\%$ error change when

removed. The sequence of networks obtained from pruning (DM-2) the network $\mathcal{W}_{DM4}^{N_h}$ will be denoted by $\mathcal{W}_{DM4}^{(k,N_h)}$, for $1 \leq k \leq N_h$.

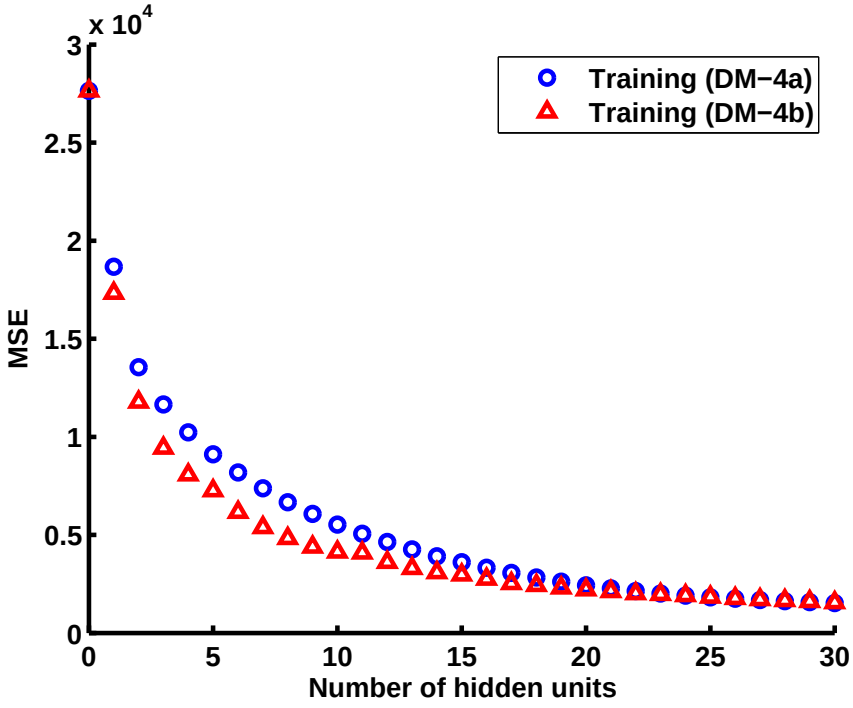
5. If $N_h < N_{hFinal}$, then goto step 2.
6. If $N_h \neq N_{hFinal}$ add appropriate number of hidden units so that the total number of hidden units equal N_{hFinal} and train the network the same way as in step 3.
7. For each size network, we have to select the network that gives minimum validation error. We get an equation similar to equation 4.1.

$$\mathcal{W}_{DM4}^{(k,N_{hFinal})} = \text{argmin}_{n \in \mathbb{N}_h} E_{val}\Big\{\mathcal{W}_{DM4}^{(k,n)}\Big\} \tag{4.3}$$

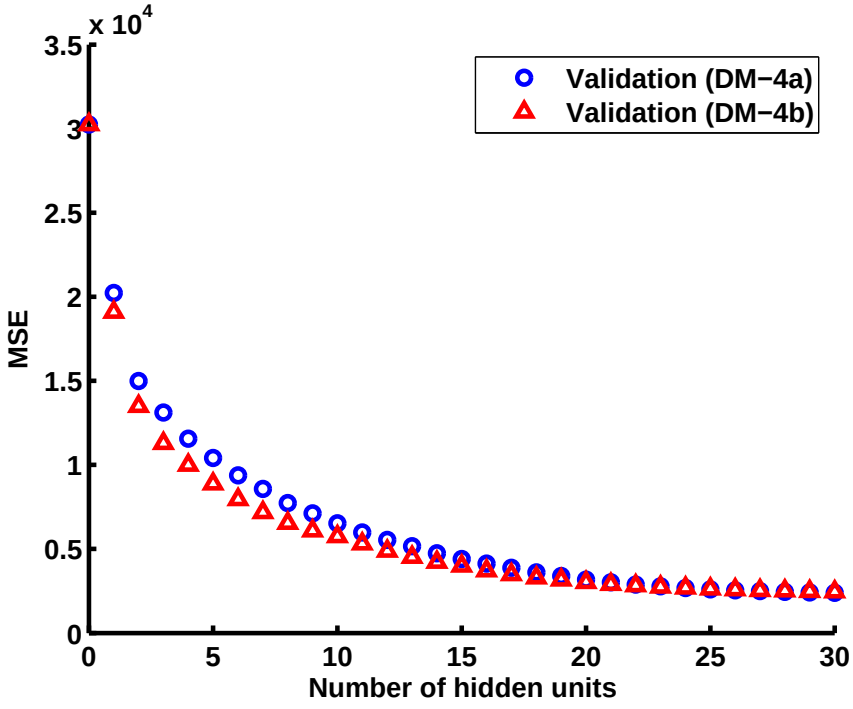
for $0 \leq k \leq N_{hFinal}$. Here $\mathbb{N}_h$ is the population of all the networks obtained during step 4 of the DM-4 procedure. The best final network chosen will be that which gives the minimum validation error over the sequence of all size networks starting from no hidden units to N_{hFinal} hidden units. We get an equation similar to equation 4.2.

$$\mathcal{W}_{DM4}^{(k^{opt},N_{hFinal})} = \text{argmin } E_{val}\Big\{\mathcal{W}_{DM4}^{(k,N_{hFinal})}\Big\} \tag{4.4}$$

It can be seen from Fig. 4.4, that the minimum operator not only decreases the MSE for smaller size networks, but also preserves the monotonicity fairly well. Hence, DM-4b is the best of the proposed methods. In the remainder of this dissertation, DM-4 refers to DM-4b.



(a)



(b)

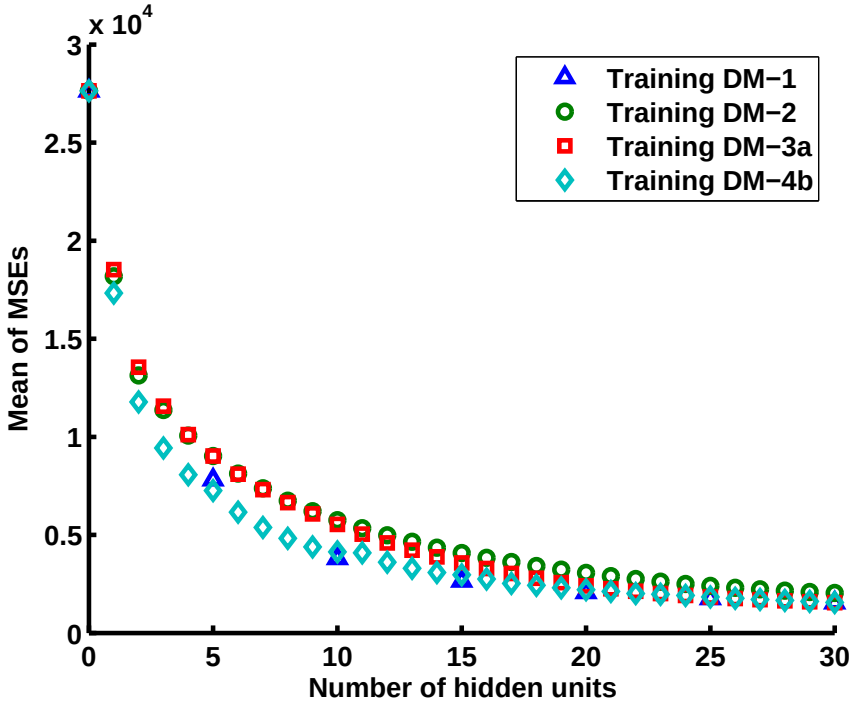
Figure 4.4. MSEs, Speech data set (a) Training (b) Validation.

4.1.3 Performance Evaluation

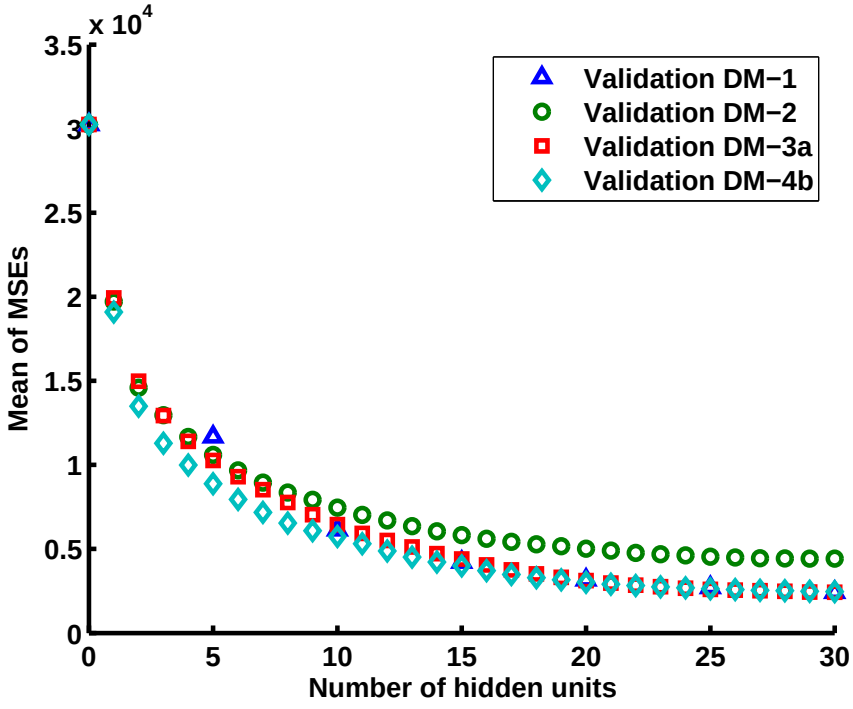
In the previous section, we show that DM-4 has advantages over other training algorithms we have developed. In this section, we compare DM-4 to two well known existing DI approaches for growing a sequence of networks. We have generated the sequence of networks on different random number seeds and the average and the standard deviation of MSEs for various network sizes are computed as explained in Chapter 2.

In Figs. 4.5 and 4.6, we have shown the performances of the four design methodologies discussed in this section. Fig. 4.5 (a) and (b) show respectively the averages of training and validation MSEs obtained over different random initialization of the network during training. Fig. 4.6 (a) and (b) show respectively the the standard deviations of training and validation MSEs over these initializations. The averages and standard deviations of MSEs are necessary to analyze the robustness of the various techniques.

The following are the comments and observations on these plots. Clearly, we can see the disadvantage of DM-2, in which we train a large network and prune to obtain different size networks. Both the training and validation errors are high in this case. Also, the standard deviation increases for larger networks (i.e., as number of hidden units N_h increases) in DM-2, which indicates that the larger networks in this technique are more sensitive to initial weights. This also demonstrates that the growing method performs better than training a randomly initialized large network. In DM-1, the average MSE for smaller networks is less than that of the other methods, but the standard deviation is very high when compared to other methods. This shows that the smaller networks in DM-1 are very sensitive to initial weights. As our goal is to find a network generation technique that is less sensitive to initial conditions and have monotonic error versus number of hidden units curve, we need to find a technique that has advantages of both DM-1 and DM-2. So, in DM-3, we first grow the MLP and later prune the final large grown network. From the plots, it can be seen that both average MSE and standard deviation of the MSE for DM-3 are less than for the other methods. Further, if there are saturated hidden units or the hidden units that are linearly dependent, those units

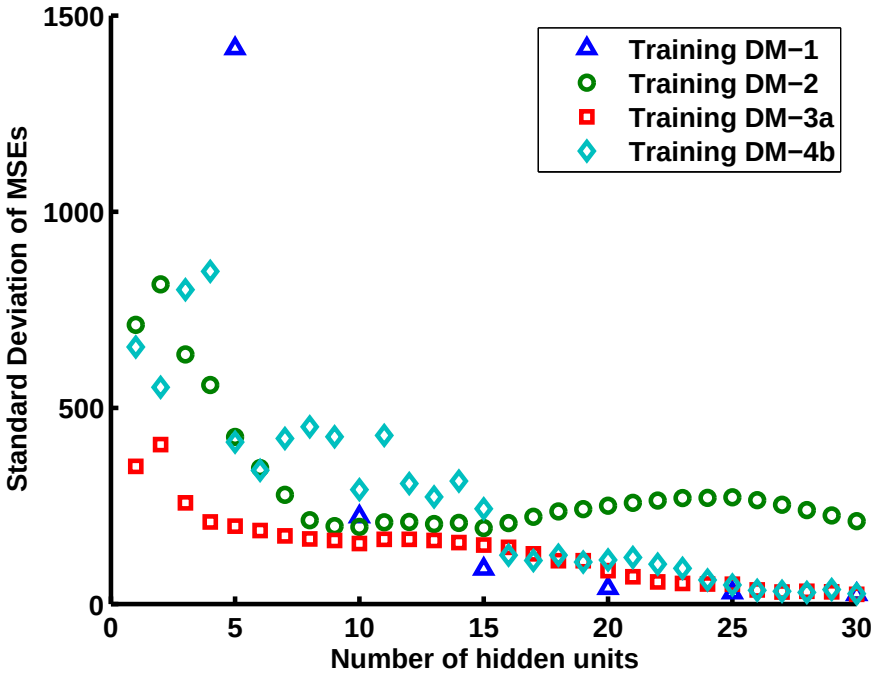


(a)

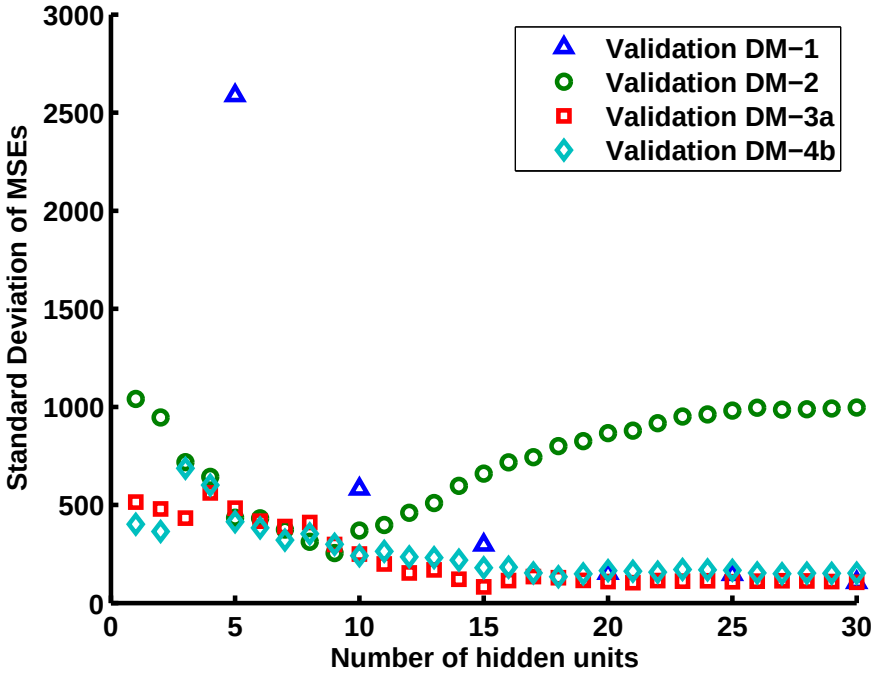


(b)

Figure 4.5. Mean MSEs, Speech data (a) Training (b) Validation.



(a)



(b)

Figure 4.6. SD of MSEs, Speech data (a) Training (b) Validation.

can be removed. This is achieved using the DM-4 approach, where the hidden units are pruned in every step of growing. Consistent to our analysis, the DM-4 method does give the best result.

Note that in DM-4, the auto and cross-correlation matrices $\mathbf{R} = \{r(i,j), 1 \leq i,j \leq N + N_h + 1\}$ and $\mathbf{C} = \{c(i,j), 1 \leq i \leq M, 1 \leq j \leq N + N_h + 1\}$ do not need to be calculated for pruning (see equations B.5 and B.11), since they are already found during training in the OWO step. This saves an extra pass through the data at each growing step. The number of multiplies eliminated for calculating the hidden unit activations, auto and cross correlation matrices is

$$N_m = N_v \cdot \left[(M+N) \cdot N_h + \frac{N_h \cdot (N_h+1)}{2} \right] \tag{4.5}$$

This savings in multiplies is significant, though small.

4.2 Results and Comparison

Here, we compare DM-4 to constructive backpropagation (CBP) and cascade correlation (CC), which are discussed in appendix C. Note that both these techniques are examples of DM-1. All the techniques were evaluated on seven different datasets. As explained in the previous section, all techniques are executed $N_r = 10$ times with different random initial weights. The average and standard deviation of MSEs for $N_h = 5, 10, 30$ on all the datasets are shown in Table 4.1. The italicized values are the ones in which our proposed methodology has sub-optimal performance compared to other algorithms. We have justified this behavior in the discussion of each dataset that is presented in the following. Also some example plots of the average and standard deviation of the MSEs versus hidden units are given for a couple of datasets. The first five datasets are available for public use [68]. The last two datasets are the well-known benchmark datasets available for public to access at StatLib repository and PROBEN1 collection.

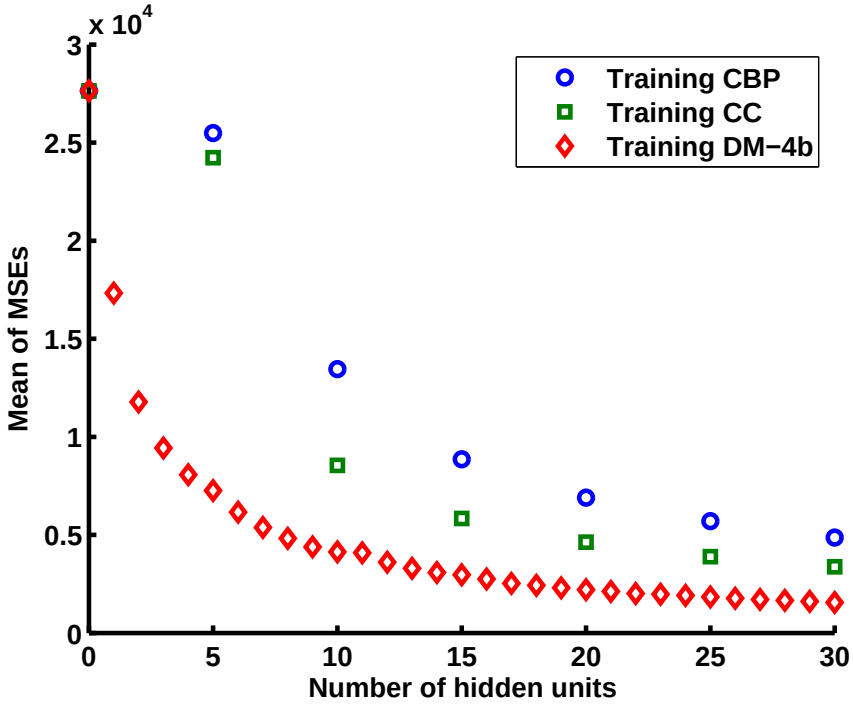
As discussed by Yu *et al.* [69], the HWO algorithm converges as the change in the error function is non-increasing and the algorithm uses backtracking in order to save the best network without diverging from the solution. It is also shown that the convergence speed of the new algorithm is higher than those of other competing techniques.

Speech: Figures 4.7 and 4.8 gives the performance comparison of the proposed method with the CBP and CC algorithms. It can be seen clearly that the averages and standard deviations of validation MSEs for all size networks for the proposed method are lesser than the CC and CBP.

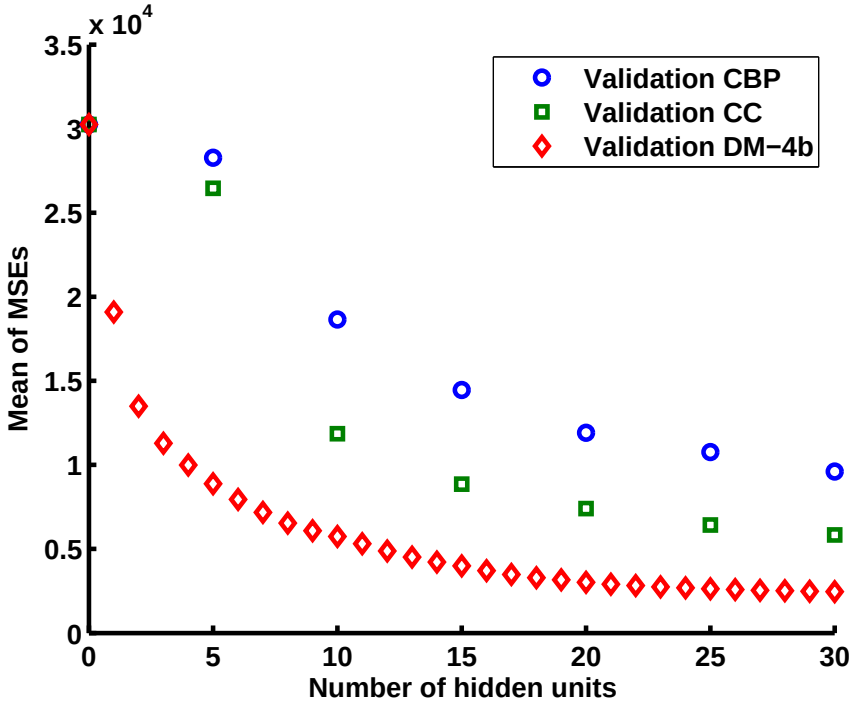
F17: This set contains prognostics data for onboard flight load synthesis (FLS) in helicopters [61], where we estimate mechanical loads on critical parts, using measurements available in the cockpit. There are 17 inputs and 9 outputs. From Table 4.1, it is clearly seen that our proposed method outperforms the existing techniques in both averages and standard deviations of validation MSE for all the network sizes considered.

Oh7: This data set is for inversion of radar scattering from bare soil surfaces [70]. It has 20 inputs and 3 outputs. The training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 degrees along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g/cubic cm. In this case, it is observed that the average and standard deviation of validation MSEs for big networks ($N_h = 30$) for the proposed method are larger than the existing methods. However, the averages and standard deviations of validation MSE for smaller network ($N_h = 5, 10$) are lesser. Hence the networks with larger MSEs are discarded.

Single2: This data set is for the inversion of surface permittivity [71]. This data has 16 inputs and 3 outputs. The inputs represent the simulated back scattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarization. The remaining 8 are various combinations of ratios of the original eight values. For this dataset, all the averages of validation MSEs for the proposed method are clearly better than for the existing methods, but the standard deviation of validation MSE for $N_h = 5$

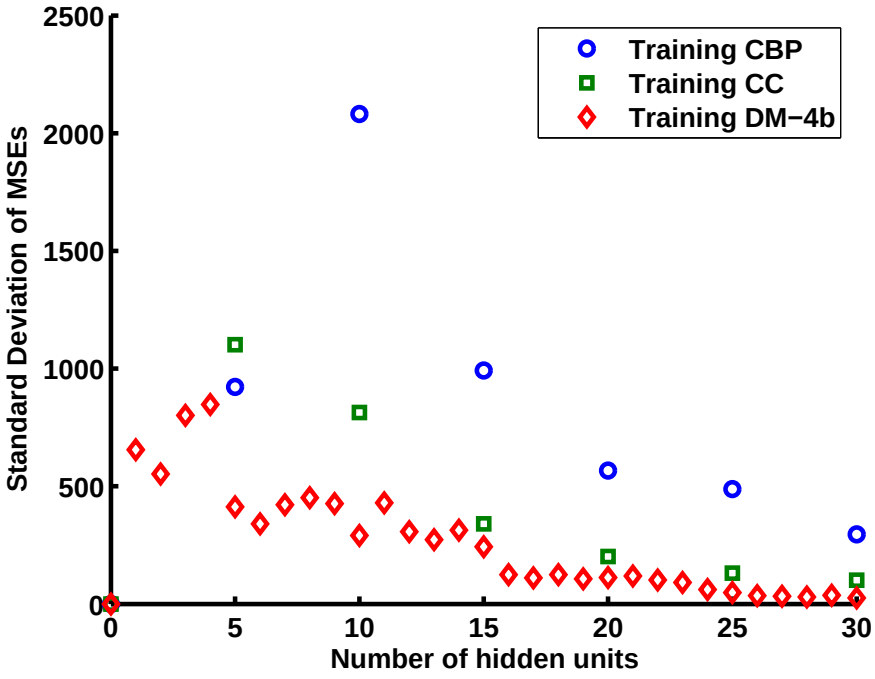


(a)

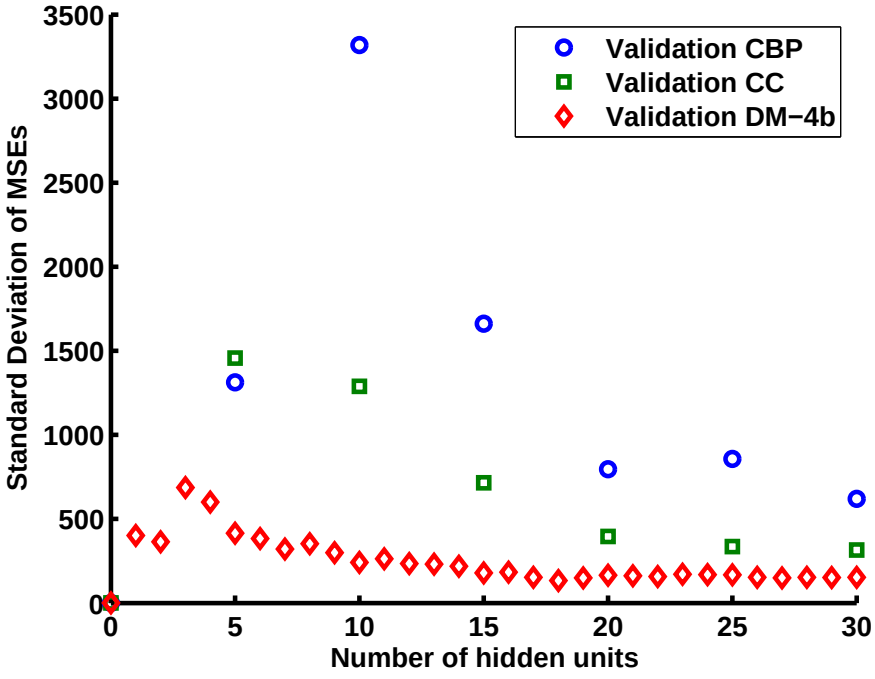


(b)

Figure 4.7. Mean MSEs, Speech data (a) Training (b) Validation.

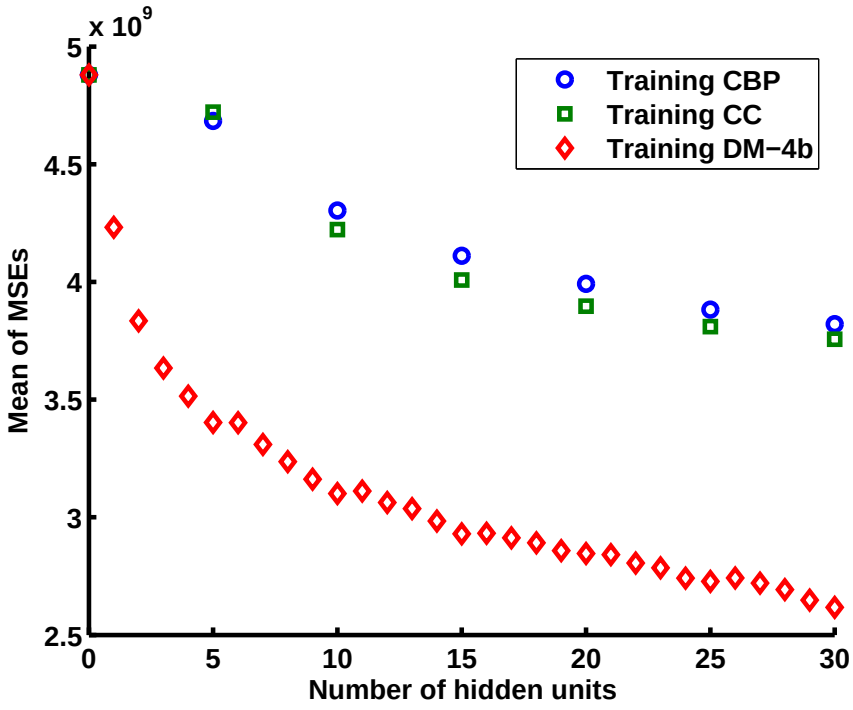


(a)

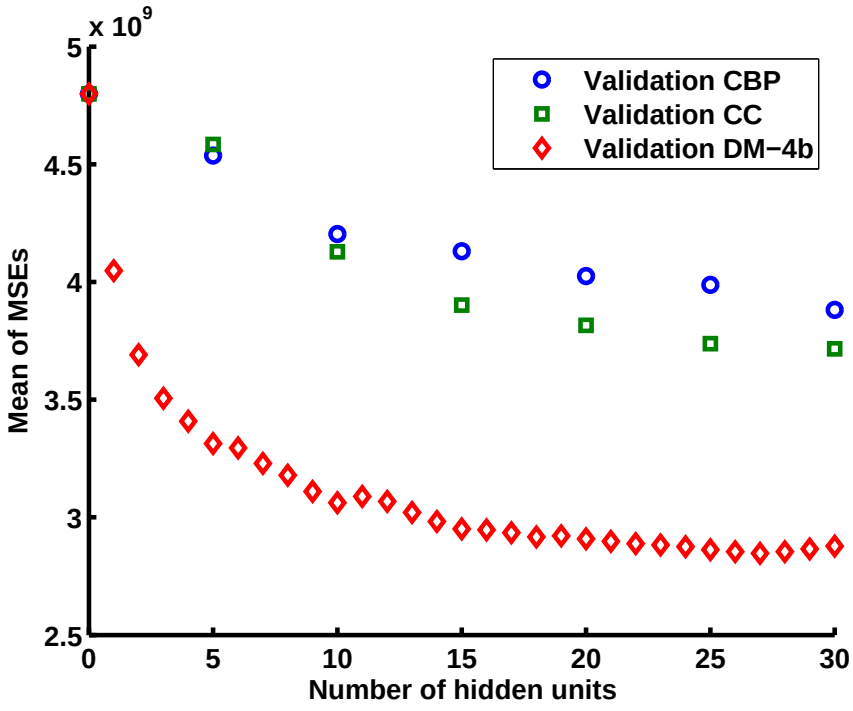


(b)

Figure 4.8. SD of MSEs, Speech data for (a) Training (b) Validation.

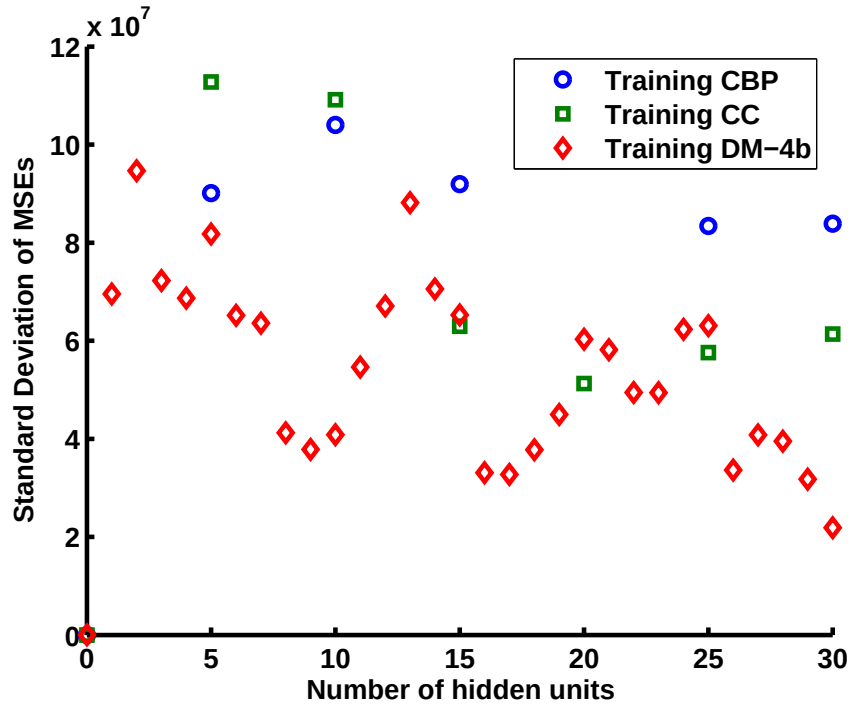


(a)

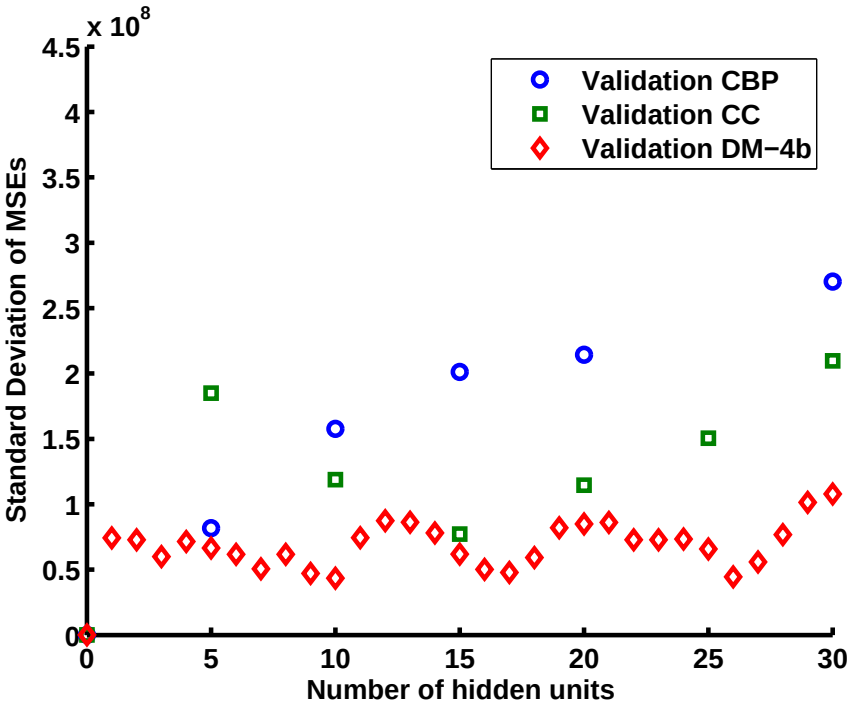


(b)

Figure 4.9. Mean MSEs, California housing data (a) Training (b) Validation.



(a)



(b)

Figure 4.10. SD of MSEs, California housing data (a) Training (b) Validation.

on the proposed method is a little higher. However, the average MSEs for proposed method are so much better than for the existing methods, that the higher standard deviation does not affect actual performance.

Twod: This training file [72, 73] 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 inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. 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 which had a joint uniform pdf. Again on this dataset, it is clearly seen that our proposed method outperforms the existing techniques in both averages and standard deviations of validation MSE for all the network sizes considered.

California Housing: This is a dataset obtained from the StatLib repository. The information was collected on the variables using all the block groups in California from the 1990 Census. In this sample a block group on average includes 1425.5 individuals living in a geographically compact area. Naturally, the geographical area included varies inversely with the population density. The distances are computed among the centroids of each block group as measured in latitude and longitude. The final data contained 20,640 observations on 9 variables. The dependent variable is ln(median house value). Figures 4.9 and 4.10 gives the performance comparison of the proposed method with the CBP and CC algorithms. It can be seen clearly that the average MSE and standard deviation of MSE for all size networks in the proposed method are lesser than the CC and CBP.

Table 4.1. Validation MSEs, mean (1st row) and SD (2nd row) ($N_r = 10$).

	$N_h = 5$			$N_h = 10$			$N_h = 30$		
Data Sets	CBP	CC	DM-4b	CBP	CC	DM-4b	CBP	CC	DM-4b
Speech	2.83e+4	2.65e+4	8.88e+3	1.86e+4	1.19e+4	5.74e+3	9.60e+3	5.83e+3	2.46e+3
	1.31e+3	1.46e+3	4.15e+2	3.32e+3	1.29e+3	2.41e+2	6.20e+2	3.15e+2	1.52e+2
F17	1.84e+8	1.83e+8	3.52e+7	1.54e+8	1.41e+8	2.31e+7	1.07e+8	9.46e+7	1.65e+7
	1.08e+7	7.13e+6	5.20e+6	1.04e+7	1.14e+7	3.33e+6	6.19e+6	5.85e+6	4.75e+6
Oh7	3.07e+0	3.01e+0	1.54e+0	1.84e+0	1.77e+0	1.51e+0	1.55e+0	1.52e+0	*1.61e+0*
	2.28e-1	1.93e-1	1.81e-2	9.63e-2	9.62e-2	1.87e-2	1.61e-2	1.58e-2	*4.54e-2*
Single2	1.49e+0	1.49e+0	6.55e-2	9.57e-1	1.12e+0	4.88e-2	4.99e-1	6.12e-1	6.08e-2
	3.00e-2	3.97e-2	*3.59e-2*	2.44e-1	1.68e-1	3.55e-2	1.76e-1	2.06e-1	4.82e-2
Twod	3.39e-1	3.37e-1	1.71e-1	2.85e-1	2.92e-1	1.37e-1	2.34e-1	2.56e-1	1.16e-1
	9.80e-3	8.39e-3	6.89e-3	1.10e-2	2.81e-2	4.30e-3	1.09e-2	2.26e-2	5.31e-3
Cal housing	4.54e+9	4.58e+9	3.31e+9	4.20e+9	4.13e+9	3.06e+9	3.88e+9	3.72e+9	2.88e+9
	8.17e+7	1.85e+8	6.65e+7	1.58e+8	1.19e+8	4.35e+7	2.70e+8	2.10e+8	1.08e+8
Building1	2.44e-2	2.41e-2	2.17e-2	2.31e-2	2.24e-2	*2.26e-2*	2.39e-2	2.52e-2	*3.07e-2*
	5.81e-4	5.56e-4	*1.76e-3*	1.66e-3	1.55e-3	1.40e-3	2.16e-3	5.36e-3	*8.11e-3*

Building1: The Building1 problem (taken from PROBEN1 benchmark collection) predicts the energy consumption in a building. It tries to predict the hourly consumption of electrical energy, hot water, and cold water, based on the date, time of day, outside temperature, outside air humidity, solar radiation, and wind speed. It has 14 inputs, 3 outputs, and 4208 patterns. Our proposed method has shown sub-optimal performance for larger networks ($N_h = 10, 30$) compared to the CBP and/or CC algorithms on this dataset. However, for $N_h = 5$, the average validation MSE of our method is lesser than the other methods considered, but the standard deviation is higher than the other methods. For $N_h = 5$, our proposed method with any random initial network performs better than or same as any of the two methods CBP and CC. Hence, we discard all the big networks and save the network which gives the least validation MSE which will be the network with $N_h = 5$.

Next, we compute the coefficients of determination (R^2) values that will give information about the goodness of fit of the model. For each dataset we have compared R^2 values (for both training and validation) of our proposed algorithm with the two benchmark techniques. For computing this, we use the sum of squared errors (SS_E) of the best network for each dataset.

$$R^2 = 1 - \frac{SS_E}{SS_T} \tag{4.6}$$

where SS_E is the sum of squared error (residual error) and SS_T is the total sum of squares of the desired outputs. Table 4.2 shows the coefficient of determination values for both training and validation. Note that in terms of model fitting (training R^2), our proposed method outperforms the other methods. In terms of generalization, our proposed method is better in all of the datasets except for Oh7 where the R^2 value is insignificantly lower compared to other methods.

In order to analyze the complexity of the training algorithms, we compare the total number of multiplies for each training algorithm to train a sequence of networks. Then

Table 4.2. Coefficient of determination (R^2) values.

	Training			Validation		
Datasets	Constructive Backpropagation	Cascade Correlation	OWO-HWO (DM-4b)	Constructive Backpropagation	Cascade Correlation	OWO-HWO (DM-4b)
Speech	0.895355	0.927280	0.967146	0.796655	0.876492	0.949425
F17	0.972791	0.975443	0.997999	0.970566	0.973926	0.996710
Oh7	0.841103	0.843756	0.883139	0.838076	0.841178	*0.832213*
Single2	0.992278	0.989352	0.999960	0.986611	0.983582	0.999400
Twod	0.803962	0.802961	0.923472	0.788988	0.770901	0.902647
Cal housing	0.714922	0.719687	0.804399	0.705777	0.718300	0.777062
Building1	0.900011	0.892534	0.934435	0.110040	0.132986	0.135555

we give comparison plots of MSE versus time in seconds for the three training algorithms on example datasets. The number of multiplies for CBP, CC and DM-4 algorithms are given respectively in the inequalities below.

$$\begin{aligned} M_{cbp} \quad \leq \quad & N_{it}N_v\Big\{0.2N_\epsilon\Big[N + N_h + \frac{1}{2}(1 - N_\epsilon) + M\Big] \\ & +0.8\Big[(N + M)N_h + \frac{N_h(N_h + 1)}{2}\Big] + \Big[N_h(N + 2M + 3)\Big]\Big\} \end{aligned} \tag{4.7}$$

$$M_{cc} \leq N_{it}\left\{N_v\Big[4MN_h + N_\epsilon(3N_h + 5M)\Big] + N_\epsilon(2M + 5)\right\} \tag{4.8}$$

$$M_{DM4} \leq M_{cbp} + N_{it}\left\{\frac{7N^2 + 3N}{2} + N_hN(N + 1)\right\} + \frac{N_h(N_h + 1)(7N^2 + 3N)}{4} \tag{4.9}$$

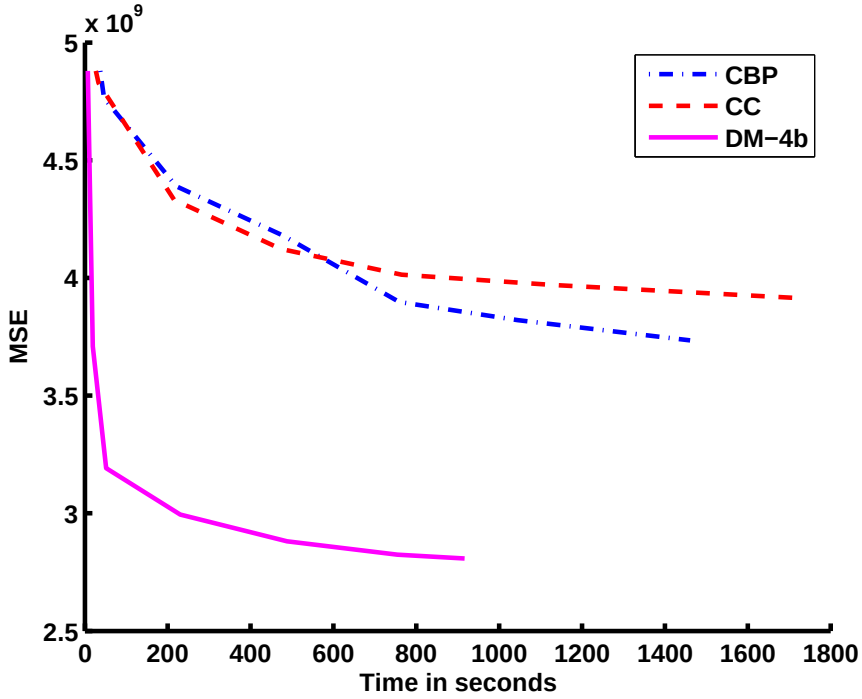
Here N_ϵ is the number of new hidden units added in every growing step (see Step 2 of Algorithm DM-4b).

Figures 4.11 (a) and (b) show the plots of the MSE versus time in seconds for California housing and speech datasets respectively. Note that the convergence time for DM-4 on California housing dataset is shorter as it reaches the local minima before it executes total number of iterations, N_{it}. This is because, if the learning factor is very small and the MSE does not decrease for more than five iterations, then a local minimal is reached and the training is stopped for that network.

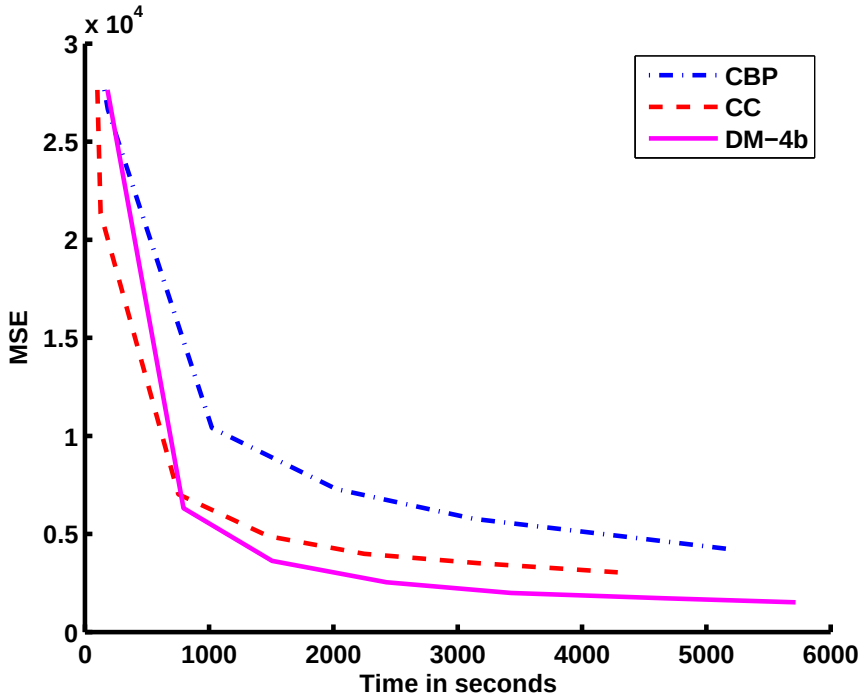
4.3 Extension to Classification Case

Growing and pruning are also applicable to classification problems. The MLP classification training that we use involve the Output Reset (OR) algorithm developed by Gore et al. [74].

We have tested the DM-4b technique for classification on several datasets and compare it to the benchmark techniques: cascade correlation and constructive backpropagation. The following are the discussion on the performances of the three techniques on different datasets. The first three datasets are available for public use [68]. The last



(a)



(b)

Figure 4.11. Convergence time (a) California housing (b) Speech datasets.

three datasets are from UCI Machine Learning Repository
(http://mlearn.ics.uci.edu/MLSummary.html).

The segmentation dataset (Comf18) [75] has 18 inputs and 4 classes and is generated using segmented images. Each segmented region is separately histogram equalized to 20 levels. Then the joint probability density of pairs of pixels separated by a given distance and a given direction is estimated. We use 0, 90, 180, 270 degrees for the directions and 1, 3, and 5 pixels for the separations. The density estimates are computed for each classification window. For each separation, the co-occurrences for for the four directions are folded together to form a triangular matrix. From each of the resulting three matrices, six features are computed: angular second moment, contrast, entropy, correlation, and the sums of the main diagonal and the first off diagonal. This results in 18 features for each classification window.

A prognostics dataset - F17C [74] has 17 inputs and 39 classes. This data file consists of parameters that are available in the basic health usage monitoring system (HUMS), plus some others. The data was obtained from the M430 flight load level survey conducted in Mirabel Canada in early 1995. The input features include: (1) CG F/A load factor, (2) CG lateral load factor, (3) CG normal load factor, (4) pitch attitude, (5) pitch rate, (6) roll attitude, (7) roll rate, (8) yaw rate, (9) corrected airspeed, (10) rate of climb, (11) longitudinal cyclic stick position, (12) pedal position, (13) collective stick position, (14) lateral cyclic stick position, (15) main rotor mast torque, (16) main rotor mast pm, (17) density ratio. The 39 classes represents different maneuvers of the flight like taking off, landing, turning right or left etc. This is an application for prognostics or flight condition recognition.

Gongtrn [76] has 16 inputs and 10 classes and corresponds to numeral recognition. The raw data consists of images from hand printed numerals collected from 3000 people by the Internal Revenue Service. We randomly chose 300 characters from each class to generate 3000 character training data. Images are 32 by 24 binary matrices. An image

scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numerals.

Glass dataset [77]: A data frame with 214 observation containing examples of the chemical analysis of 7 different types of glass. The problem is to forecast the type of class on basis of the chemical analysis. The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence (if it is correctly identified!).

Mushroom dataset: This data set [77] includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom.

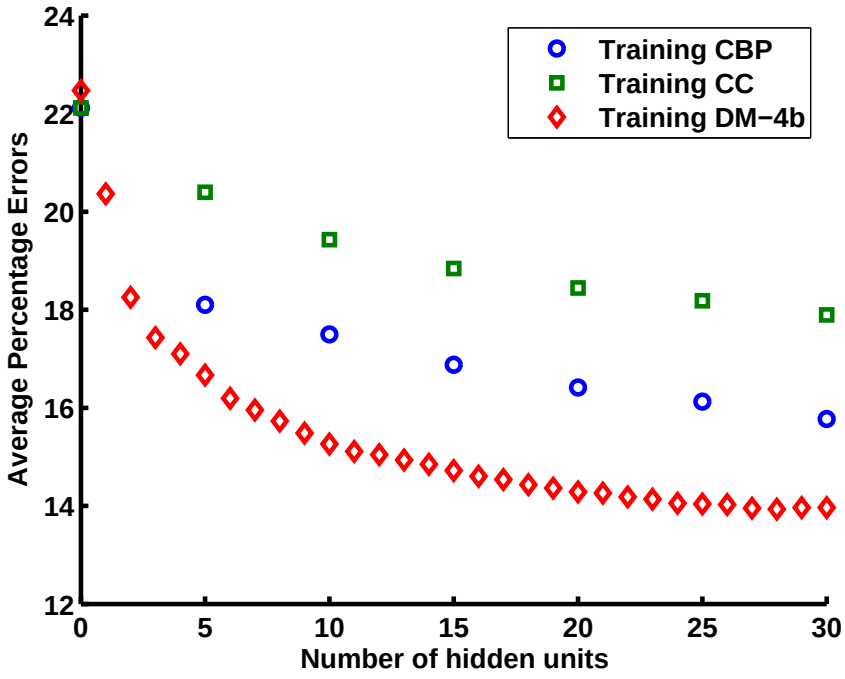
Diabetes dataset [77] contains the distribution for 70 sets of data recorded on diabetes patients (several weeks' to months' worth of glucose, insulin, and lifestyle data per patient + a description of the problem domain).

Figures. 4.12 and 4.13 show respectively the plots for average and standard deviations of error percent values over different random initialization of the network during training for the Segmentation dataset. The corresponding plots for Mushroom dataset are shown in Figures. 4.14 and 4.15. It is evident that our DM-4b outperforms the two benchmark techniques compared.

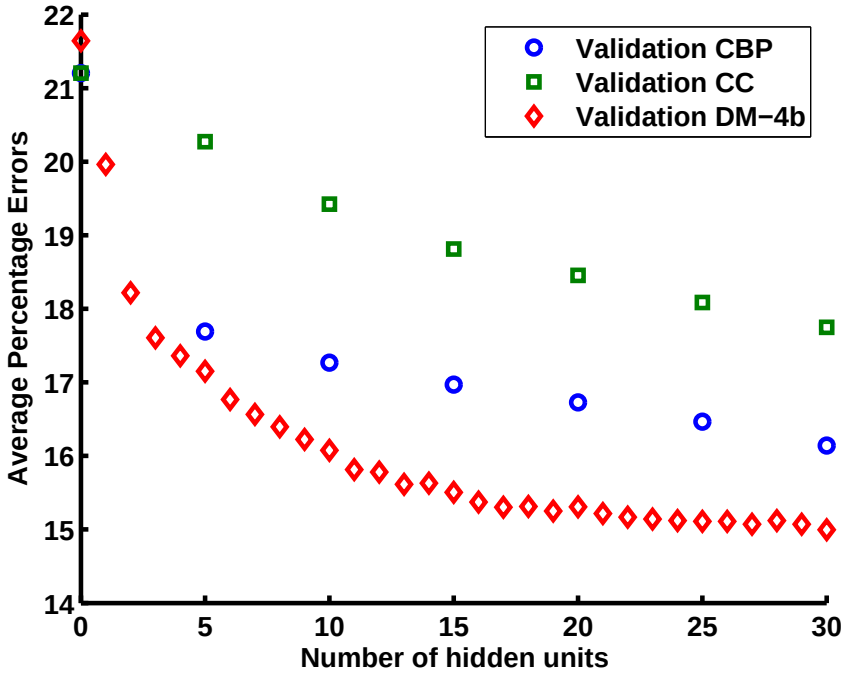
In Table 4.3 we have shown the comparison results for all the six datasets considered. Number of hidden units (N_h) for each dataset is chosen as the one that obtains minimum validation error. The minimum percentage classification errors for each dataset is indicated in bold. It is clearly seen that our method gives best results on all datasets except for Diabetes dataset. We have given the means and standard deviations of the three techniques on six different datasets.

Table 4.3. Validation errors (%), mean (1st row) SD (2nd row) ($N_r = 10$).

Data Sets	CBP	CC	DM-4b
Segmentation	16.1	17.7	**15.0**
($N_h = 30$)	3.86e-001	4.44e-001	4.37e-001
Numeral	8.64	11.4	**7.9**
($N_h = 30$)	3.80e-001	4.51e-001	3.43e-001
Flight Sim	6.47	27.0	**2.68**
($N_h = 30$)	6.26e-001	1.11e+000	4.26e-001
Diabetes	**25.0**	25.1	26.2
($N_h = 5$)	1.06e+000	8.72e-001	1.17e+000
Glass	34.8	33.8	**31.9**
($N_h = 5$)	4.66e+000	3.11e+000	2.35e+000
Mushroom	0.125	0.516	**0.0295**
($N_h = 30$)	1.27e-001	2.79e-001	4.14e-002

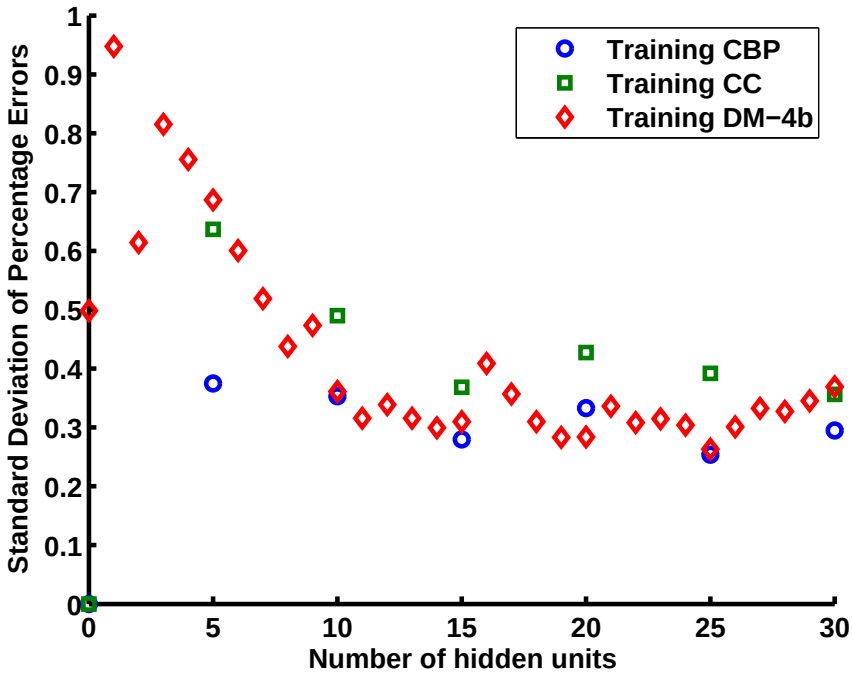


(a)

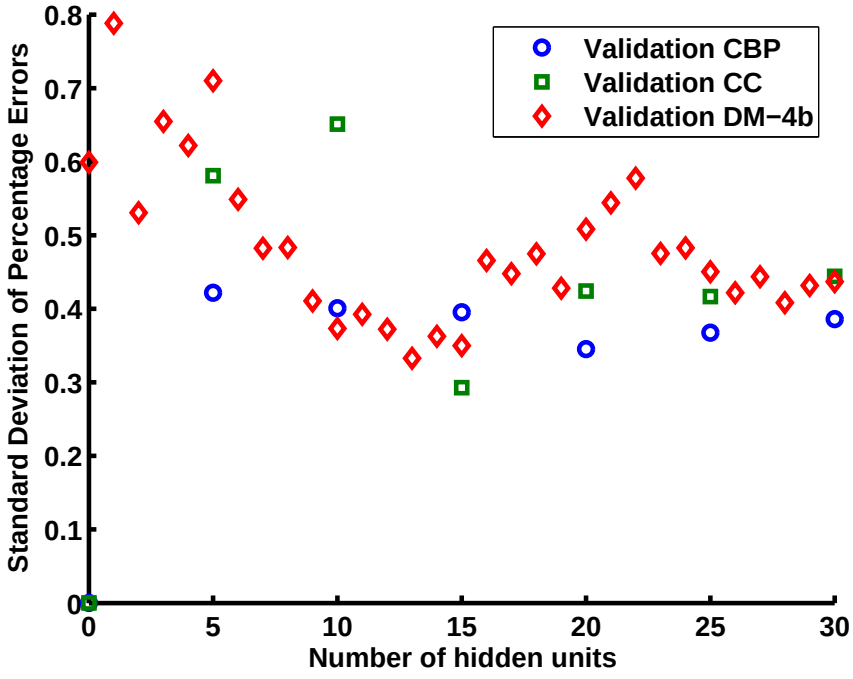


(b)

Figure 4.12. Comparison, Segmentation data (a) Training (b) Validation.

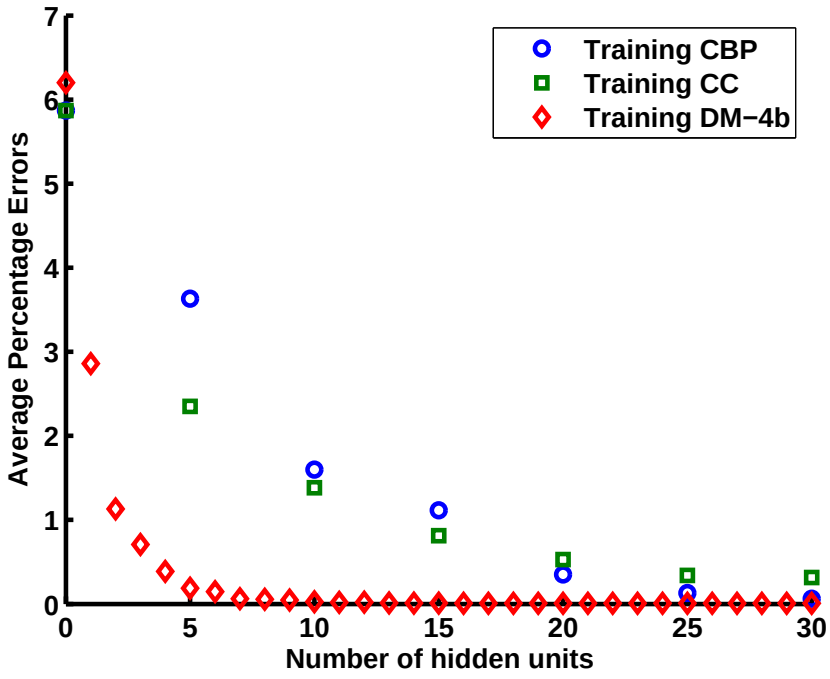


(a)

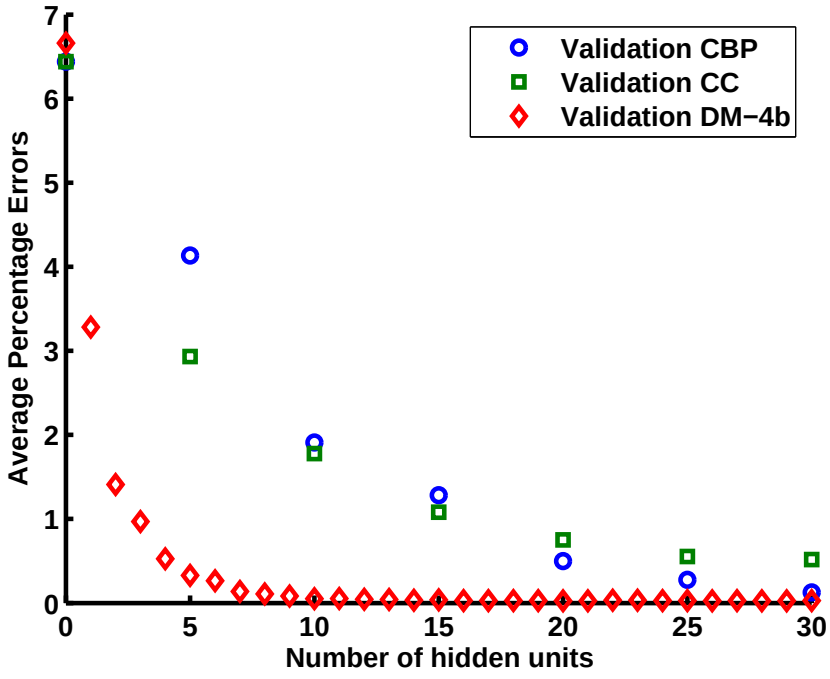


(b)

Figure 4.13. Comparison, Segmentation data (a) Training (b) Validation.

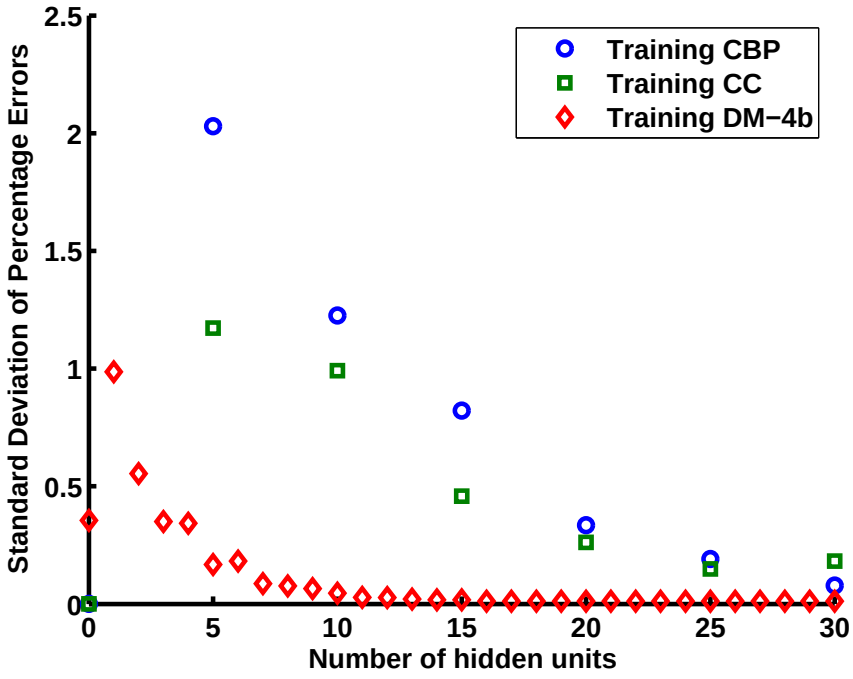


(a)

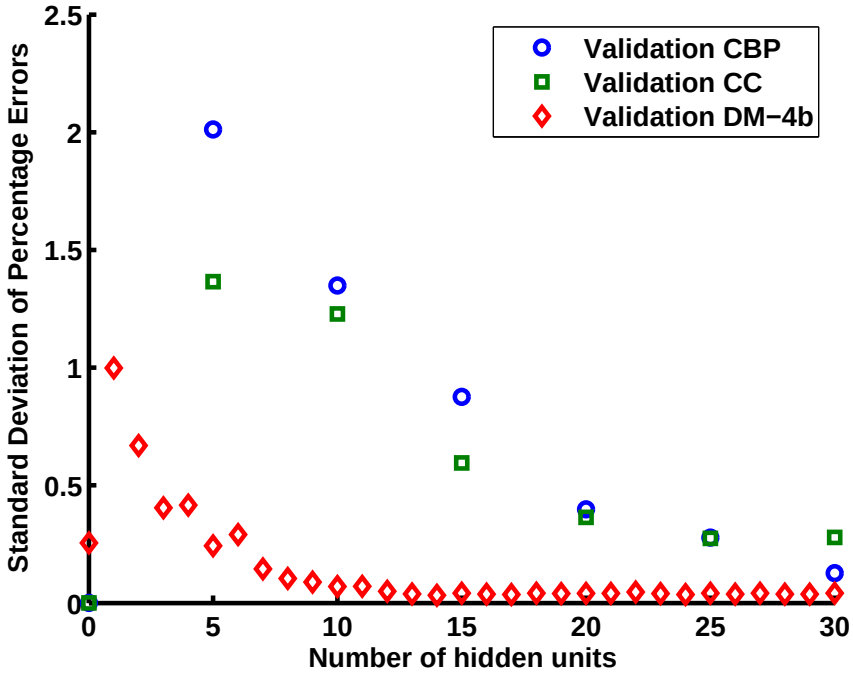


(b)

Figure 4.14. Comparison, Mushroom data (a) Training (b) Validation.



(a)



(b)

Figure 4.15. Comparison, Mushroom data (a) Training (b) Validation.

CHAPTER 5

UPPER BOUND ON PATTERN STORAGE

In this chapter, we give a straight forward proof of an upper bound on pattern storage. An example which indicates the validity of the bound is presented. Also, we use the upper bound to predict the size of MLPs that can mimic the training behavior of SVMs.

5.1 Storage Capacity and Memorization

A feedforward network is said to have memorized a dataset if for every pattern, the network outputs are exactly equal to the desired outputs. Storage capacity of a feedforward network is the number (N_v) of distinct input vectors that can be mapped, exactly, to the corresponding desired output vectors resulting in zero error.

A review of upper and lower bounds on memorization is given in Chapter 2. In the next section, we devise a proof for upper bound on pattern storage.

5.2 An Upper Bound

Here, we describe direct approach for proving the upper bound on memorization given previously in chapter 2. We outline pitfalls in this direct approach. Then a proof by contradiction is presented which assumes no restrictions on the hidden units activations.

5.2.1 Modeling of Memorization

Let us assume monomial activation functions of degree d for the hidden units. In this case,

$$O_{pj} = (net_{pj})^d \tag{5.1}$$

$$= \left(\sum_{i=1}^{N+1} w_h(j,i) \cdot x_{pi} \right)^d \tag{5.2}$$

The above equation can be written as a power series in $\mathbf{x}_p$ as in the Volterra filter. Let $\mathbf{X}_p = [X_{p1}, X_{p2}, \cdots, X_{pL}]^T$ contain the multinomial combinations of the input variables for the p^{th} example. Here, L is the number of polynomial basis functions (PBFs) of the form $x_1^n \cdot x_2^m \cdots x_N^q$. For N inputs and maximum degree d, the number of multivariate basis functions that can be derived is given by

$$L = \frac{(N+d)!}{N!d!} \tag{5.3}$$

The output in (2.3) can be written as a linear combination of these multinomial bases

$$y_{pk} = \sum_{i=1}^{L} a_{ik} \cdot X_{pi} \tag{5.4}$$

for $1 \leq k \leq M$ and $1 \leq p \leq N_v$, where N_v is the number of patterns. Note that the right hand side of (5.4) is a Gabor polynomial and is thus equivalent to the k^{th} output of a functional link net [78].

Our aim is to find the exact solution for the network weights. This requires replacement of y_{pk} with t_{pk} and making N_v equal to L. Let us define the following matrices.

$$\begin{aligned}\mathbf{X} &= \left[\mathbf{X}_1^T, \mathbf{X}_2^T, \cdots, \mathbf{X}_L^T\right]^T \\ &= \begin{pmatrix} X_{11} & X_{12} & \cdots & X_{1L} \\ X_{21} & X_{22} & \cdots & X_{2L} \\ \vdots & \vdots & \vdots & \vdots \\ X_{L1} & X_{L2} & \cdots & X_{LL} \end{pmatrix} \end{aligned} \tag{5.5}$$

$$\begin{aligned}\mathbf{A} &= [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_M] \\ &= \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ a_{L1} & a_{L2} & \cdots & a_{LM} \end{pmatrix} \end{aligned} \tag{5.6}$$

$$\begin{aligned}\mathbf{T} &= \left[\mathbf{t}_1^T, \mathbf{t}_2^T, \cdots, \mathbf{t}_L^T\right]^T \\ &= \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1M} \\ t_{21} & t_{22} & \cdots & t_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ t_{L1} & t_{L2} & \cdots & t_{LM} \end{pmatrix} \end{aligned} \tag{5.7}$$

Each row of $\mathbf{X}$ stores the basis vector $\mathbf{X}_p^T$ for one pattern. Similarly, each row of $\mathbf{T}$ stores the desired output vector for one pattern. Equation (5.4) can be written in a compact form as

$$\mathbf{XA} = \mathbf{T} \tag{5.8}$$

where $\mathbf{A}$ is an FLN coefficient matrix. We find the network weights in two steps. In the first step, we solve the above equation for $\mathbf{A}$. If the inputs are all distinct, the columns of $\mathbf{X}$ may be linearly independent and hence the rank of $\mathbf{X}$ could be equal to L. The

coefficients a_{ki} are found by solving M sets of L equations in L unknowns and it gives an exact solution for these coefficients,

$$\mathbf{A} = \mathbf{X}^{-1}\mathbf{T} \tag{5.9}$$

We know that the coefficients of the matrix $\mathbf{A}$ can be expressed as polynomial functions of the unknown weights of the network. In the second step, we solve for the actual network parameters. Let $\mathbf{f}(\mathbf{w}) = \mathbf{A}$ where $\mathbf{w}$ is a vector that contains all the network weights where the total number of network weights is N_w, hence $\mathbf{w} \in \Re^{N_w}$. Let $\mathbf{f}_1(\mathbf{w}), \mathbf{f}_2(\mathbf{w}), \cdots, \mathbf{f}_M(\mathbf{w})$ be the columns of the matrix $\mathbf{f}(\mathbf{w})$. Then

$$\mathbf{f}_i(\mathbf{w}) = \mathbf{a}_i \tag{5.10}$$

for $1 \leq i \leq M$. These equations may be solvable by using back substitution. Let us assume that we eliminate one unknown weight by equating one element of the vector $\mathbf{f}_i(\mathbf{w}) = [f_{1i}(\mathbf{w}), f_{2i}(\mathbf{w}), \cdots, f_{Li}(\mathbf{w})]^T$ to the corresponding element of $\mathbf{a}_i$. Assume $f_{ki}(\mathbf{w}) = a_{ki}$ for $1 \leq k \leq L$ can eliminate one element of $\mathbf{w}$ from the remaining equations. Thus for each i we have

$$\begin{aligned} f_{1i}(\mathbf{w}) &= a_{1i} \\ f_{2i}(\mathbf{w}) &= a_{2i} \\ &\vdots \\ f_{Li}(\mathbf{w}) &= a_{Li} \end{aligned}$$

Note that it is not possible to solve the equations by considering each output separately. There is only one set of equations, since some weights affect more than one output. For each of the $L \cdot M$ elements of $\mathbf{A}$, we have one equation in N_w unknowns, which is

$$f_{ki}(\mathbf{w}) = a_{ki} \tag{5.11}$$

So if the number of unknowns N_w, is equal to $L \cdot M$, a solution that satisfies all the above $L \cdot M$ equations is possible. In other words, for an MLP with monomial activation, the storage capacity, that is the number of patterns that can be memorized by a network with N_h hidden units is equal to the total number of weights in the network divided by the number of outputs,

$$N_v = \frac{N_w}{M} = \frac{N_h(N + M + 1) + M(N + 1)}{M} \tag{5.12}$$

There are several problems with this direct approach. The direct design of a memorization network would fail if the inverse of $\mathbf{X}$ does not exist in (5.9). Also the validity of back substitution is questionable as closed form expressions for roots of high degree polynomials do not exist. Finally, we want a proof for activations other than monomial activations.

Lagrange polynomial interpolation [79] can be used to model exact, finite degree polynomial for the activations, O_{pj}. The proof is based on the extension of the interpolation theorem of Davis [43], which states that for any N_v distinct points there exists a unique $N_v - 1$ degree polynomial that passes through all the points.

5.2.2 Arbitrary Hidden Activations

Here we relax the assumption that the activation function is a monomial and let it be arbitrary. Let us assume that the net functions for different finite inputs can be different but finite. Equating y_{pk} in (2.3) to t_{pk}, we get

$$t_{pk} = \sum_{i=1}^{N+1} w_{oi}(k,i) \cdot x_{pi} + \sum_{j=1}^{N_h} w_{oh}(k,j) \cdot O_{pj} \tag{5.13}$$

In order to model memorization above, we can use an exact, finite degree polynomial model for O_{pj}. This is accomplished by using Lagrange polynomials [79]. The $N_v - 1$ degree Lagrange polynomial that precisely satisfies $\mathcal{L}_j(net_{pj}) = O_{pj}$ is developed as

$$\mathcal{L}_j(net) = \sum_{p=1}^{N_v} O_{pj} \cdot l_p^j(net) \tag{5.14}$$

$$l_p^j(net) = \prod_{k=1,k\neq p}^{N_v} \frac{net - net_{kj}}{net_{pj} - net_{kj}} \tag{5.15}$$

The j^{th} hidden unit's activation function is precisely equal to the interpolation polynomial for our training patterns, so

$$O_{pj} = \sum_{n=0}^{N_v-1} \alpha_{jn}(net_{pj})^n \tag{5.16}$$

where α_{jn} are the Lagrange polynomial coefficients. Substituting equation (5.16) into equation (5.13), we get

$$\begin{aligned} t_{pk} = & \sum_{i=1}^{N+1} w_{oi}(k,i) \cdot x_{pi} \\ & + \sum_{j=1}^{N_h} w_{oh}(k,j) \sum_{n=0}^{N_v-1} \alpha_{jn}(net_{pj})^n \end{aligned} \tag{5.17}$$

For the memorized training patterns, note that (5.17) is exact, rather than being an approximation of (5.13). As net_{pj} is a linear combination of the inputs, the expansion

of $(net_{pj})^n$ results in polynomial combinations of the input variables of maximum degree $N_v - 1$. Thus, we get

$$t_{pk} = \sum_{i=1}^{L} a_{ki} \cdot X_{pi} \tag{5.18}$$

which is similar to the output equation obtained for the monomial activation case. The number of basis functions, L, will always be equal to N_v. With this, we can formally state our theorem on the storage capacity of feedforward networks.

5.3 Theorem and Proof

Here, we restate the Theorem 5 on the storage capacity of feedforward networks which was stated in section 2.4 and give the proof.

Theorem 5 *For a feedforward network with N inputs, M outputs, N_h hidden units and arbitrary hidden activation functions, the number of patterns N_v that can be memorized with no error is less than or equal to number of absolute free parameters of the network, N_w divided by the number of outputs, M.* □

In other words,

$$N_v \leq \left\lfloor \frac{N_w}{M} \right\rfloor \tag{5.19}$$

where $\lfloor \cdot \rfloor$ denotes truncation.

Proof Let us assume that (1) the network can memorize $\lfloor N_w/M \rfloor + 1$ patterns and (2) that we know the values of all the weights. Similar to the monomial activation case, we find the coefficients of $\mathbf{A}$ by solving the equation 5.18 in the first step. There are two cases.

Case 1 (**X** is singular for the given dataset): If the inverse of **X** does not exist, assumption (1) is disproved and the theorem is confirmed for the given dataset.

Case 2 (**X** is nonsingular): For this case, the proof continues to the second step. The number of nonlinear equations that needs to be solved is $N_v M = (N_w/M + 1)M = N_w + M$. So, we have $N_w + M$ equations in N_w unknowns.

Case 2.1 (Back-substitution eliminates exactly one equation and one unknown at a time):

Here, we solve exactly one equation at a time and substitute its solution into the remaining equations. By the end of this back substitution procedure, we will be left with M or more equations and no unknowns. Hence, the network weights that are already found must satisfy the remaining M equations in order to memorize $N_v = N_w/M + 1$ distinct patterns. This in general is not possible as we shall see in Case 2.2.

The last step in this sub-case is to show the validity of back-substitution in this proof. Without assumption (2), this would be a daunting task. Note that in (5.11), each weight can occur many times. However, (5.11) can be simplified by substituting the correct value of the weight for most occurrences of the weight variable. As an example of (5.11), consider the equation

$$5w_1 w_2^8 + 4w_3^3 w_2^5 + 3w_4^8 = 43 \tag{5.20}$$

Solving for an occurrence of w_2 in the first term, we have

$$w_2 = \frac{43 - 4w_3^3 w_2^5 - 3w_4^8}{5w_1 w_2^7} \tag{5.21}$$

where the correct numerical value of w_2 is used on the right hand side. In back substitution, a solution of the form of (5.21) would be substituted into the remaining equations. Hence, we do not need a closed form expression for roots of an eighth degree equation.

Equation number N_w may be very complicated, but it will have only one unknown, and the method described above still works. For this first sub-case, there are M equations left over that cannot be solved. For this sub-case, therefore, the theorem is proved by contradiction.

Case 2.2 (In at least one back-substitution step, more than one equation is solved):

Suppose that two or more equations are solved by a back-substitution step. For the second equation solved, we have a constant on the left hand side and a FLN coefficient a_{ki} on the right hand side, in (5.11). We now change a_{ki} slightly so that the equation is not solved. Using equation (5.8), we now generate new desired outputs for our data file, without invalidating any of the equations already successfully solved. For this second sub-case, there are again M equations left over that cannot be solved. Therefore the theorem is proved by contradiction. □

Since the proof uses only $N_h \cdot N_v$ samples of the activation, its characteristics at other values do not matter. Therefore, the activations do not need to be continuous, analytic, or even bounded, in between the known points. Now that we have proved the theorem, we can investigate the tightness of the bound.

5.4 Demonstration of the Derived Bound

In this section, we experimentally demonstrate the tightness of the upper bound. We generated a dataset having fifteen inputs, two outputs and 340 patterns. All the inputs and outputs were Gaussian random numbers with zero mean and unit standard deviation. The number of hidden units required to memorize all the N_v patterns according to equation (5.12) and Theorem 5, satisfies

$$N_h \geq \frac{N_v \cdot M - M \cdot (N+1)}{N+M+1}$$

Plugging in $N_v = 340$, $N = 15$, and $M = 2$, we get

$$N_h \geq 36 \tag{5.22}$$

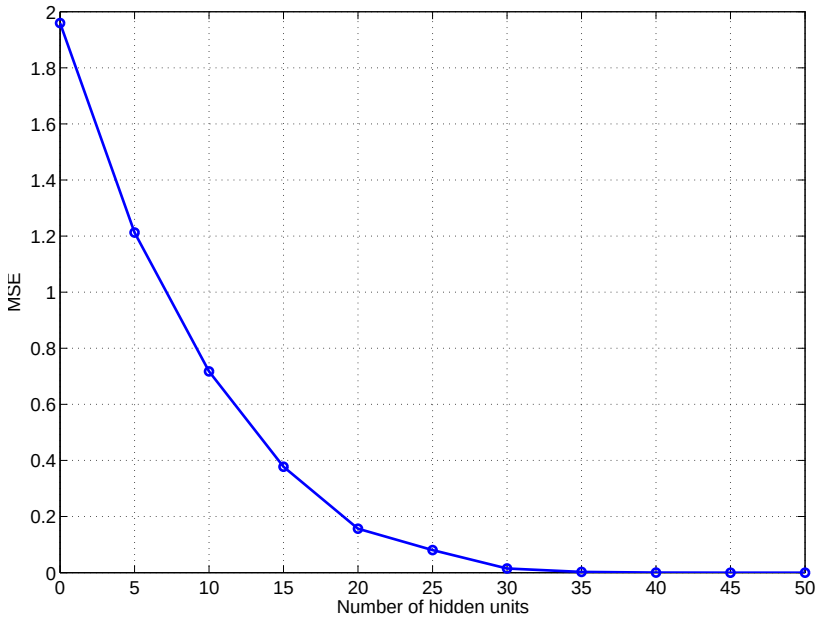


Figure 5.1. MSE versus number of hidden units (N_h).

MLP networks of different sizes were trained using the conjugate gradient algorithm [80, 81, 29] on the dataset. A plot of the MSE for different network sizes is shown in Fig. 5.1. The plot shows the MSE for networks of different sizes starting from zero hidden units up to fifty at intervals of five. It is observed that the error curve changes dramatically from the linear network case (zero hidden units) to the neighborhood of $N_h = 35$ hidden units. After 40 hidden units, the change in error is negligible. This confirms that the network takes around $N_h = 36$ hidden units to memorize all the patterns, which agrees with Theorem 5. Unfortunately, the good performance of conjugate gradient on random data does not extend to the case of correlated data [29].

5.5 Efficient Modeling of SVMs

In this section, we recall the memorization behavior of SVMs, and propose a technique to compactly model SVMs through memorization.

5.5.1 Memorization in SVMs

For an SVM, patterns that are SVs generate zero squared error. This corresponds to the lower bound on memorization [34]. Hence they follow lower bound on memorization. One logical argument would be to train a network that follows upper bound on memorization. This reduces the model size and results in faster processing.

The SVMs must have better class boundaries which are better located than those of the memorizing MLPs. What we need is a method for extracting this class boundary information and making it available to the MLP.

5.5.2 Compact Modeling Through Memorization

Kruif and Vries [52] have used pruning to reduce the training set size for function approximation using SVMs. They select those patterns that introduce minimum approximation error when omitted. However, this technique does not guarantee that the cardinality of the set of support vectors responsible for the computational complexity is minimized. Also, the SVM's ability to generate a good decision boundary is often thinly spread over thousands of SVs, so pruning of SVs does not help much. Another approach for generating a small network is needed.

Since SVM pattern memorization follows the lower bound of Theorem 4, it is natural to try to duplicate this memorization using the upper bound as in equation (5.22). One approach is to save the output values of the SVM on the training dataset to obtain a new function approximation dataset. This new dataset has the same inputs as the original training file and the SVM outputs as the desired outputs $\left\{\mathbf{x}_p, y_p^{svm}\right\}_{p=1}^{N_v}$, where y_p^{svm} is the output of the SVM for the p^{th} training pattern. We now train a feedforward network to approximate the new dataset and the network obtained is denoted

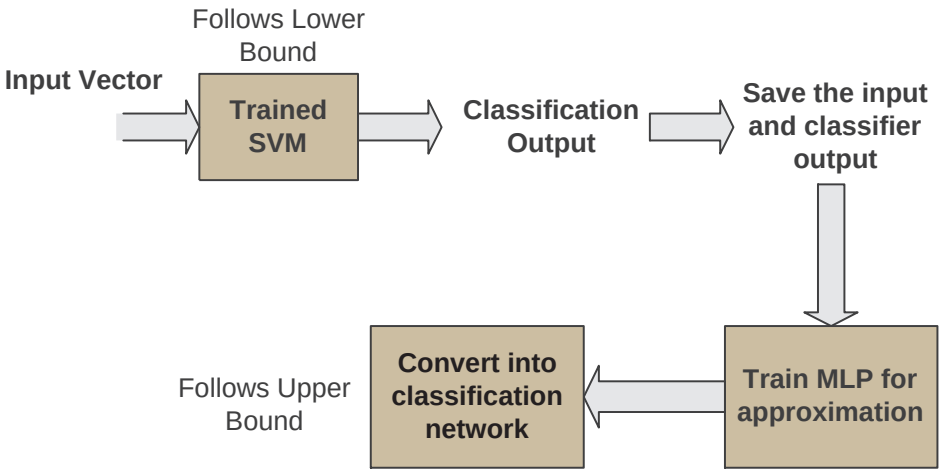


Figure 5.2. SVM modeling block diagram.

by $\mathcal{W}^{map} = \{w_{oi}^{map}, w_{oh}^{map}, w_{hi}^{map}\}$. We can also use our DM-4 approach that is discussed in chapter 4 to train the new dataset. In order to convert the network obtained into a classification network, with one output per class, all the weights are copied to the classification network without any changes. Then we add a second output unit and all the weights connecting to it are made equal to the negative of the corresponding weights connecting to the first output unit. That is

$$w_{oi}^{cls}(2,i) = -w_{oi}^{map}(1,i) \tag{5.23}$$

$$w_{oh}^{cls}(2,j) = -w_{oi}^{map}(1,j) \tag{5.24}$$

for $1 \leq i \leq N+1$ and $1 \leq j \leq N_h$. In this way we obtain a feedforward classification network denoted by $\mathcal{W}^{cls} = \{w_{oi}^{cls}, w_{oh}^{cls}, w_{hi}^{cls}\}$. We now prune [25] this network to the desired number of hidden units and compare its performance to that of the original SVM. Fig. 5.2 shows the block diagram of the modeling procedure.

Using SVM^{light} [82] which implements Vapnik's SVM [31], we have trained SVMs for several two class problems. The first dataset, Comf18, is an image segmentation dataset [75] with 18 inputs and 4 classes. Classes 1 and 2 are extracted from this dataset in order to get a binary classification problem. A prognostics dataset - F17C [74] has

Table 5.1. Modeling SVMs with MLPs.

		SVM			MLP		
Dataset	Patterns	SVs	% Training Error	% Test Error	Hidden units	% Training Error	% Test Error
Segmentation	4265	1095	7.43%	7.68%	33	6.27%	5.16%
Prognostics	238	190	5.88%	14.49%	19	0.42%	0.0%
Numeral	600	179	3.5%	5.17%	32	0.67%	8.5%
Speech	236	87	2.12%	2.63%	4	0.85%	7.89%

17 inputs and 39 classes. Here, classes 3 and 6 are extracted from the dataset to bet binary classification problem. Gongtrn [76] has 16 inputs and 10 classes and corresponds to numeral recognition. The third two class problem is formed by extracting classes 5 and 10 from the numeral recognition dataset. The fourth dataset is from a speech phoneme recognition problem, which has 39 inputs and 32 classes. We extracted classes 1 and 4 from this dataset for our experiments. The features of these datasets are already described in the previous chapter.

Table 5.1 shows the results obtained. It is clearly seen that the MLP training errors are consistently less than the SVM training errors. For the segmentation and prognostics datasets, the MLP validation errors are also smaller. Sometimes, good training performance does not carry over to the validation case, as seen in the numeral and speech datasets. Observe that the number of support vectors needed for the SVM is much larger than the number of hidden units needed for the MLP. The number of hidden units shown in the table is calculated using Theorem 5 in section 2.4. In the case of Comf18, we found that $N_h = 33$ is sufficient to get good generalization that also satisfies Theorem 5.

CHAPTER 6

COMPACT MODELING OF LARGE CLASSIFIERS

In this chapter, the extension of memorization to a workable modeling procedure is motivated and discussed. We propose two data generation methods and show the effects of additional patterns. Several simulation experiments are shown to indicate the advantages of additional patterns in training a learning machine.

6.1 Generating Additional Patterns

Although modeling through memorization works sometimes, it is unreliable. Good performance on training data is never a guarantee of good performance on validation data. Now, in order to minimize the validation error, the decision boundary of the MLP should be forced to converge to that of the best available classifier (BAC). One intuitive way of reaching the goal is by obtaining more training patterns for the MLP. Although new input vectors $\mathbf{x}_p$ can be generated, desired outputs t_p usually cannot be generated. There is a way to sidestep this problem, however. Note that in this chapter, we are considering only two class classification problems ($N_c = 2$).

6.1.1 Bias-variance Decomposition Theory

The equation for the training error, E is given by,

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} (t_p - y_p)^2 \tag{6.1}$$

where N_v is the number of training patterns, $t_p \in \{+1, -1\}$ and y_p are the desired and actual neural network outputs for the p^{th} pattern. Let s_p be the corresponding output of the BAC, for the given dataset. Borrowing the idea of the classical result of Geman's

bias-variance decomposition theory [83], we can split the training error into three parts as follows.

$$E = E_1 + E_2 + E_3 \tag{6.2}$$

where

$$E_1 = \frac{1}{N_v}\sum_{p=1}^{N_v}(t_p - s_p)^2 \tag{6.3}$$

$$E_2 = \frac{1}{N_v}\sum_{p=1}^{N_v}(s_p - y_p)^2 \tag{6.4}$$

$$E_3 = \frac{1}{N_v}\sum_{p=1}^{N_v}2(t_p - s_p)(s_p - y_p) \tag{6.5}$$

If the BAC is assumed to give the Bayesian estimate, the cross term E_3 in the above expansion will tend to zero [83, 65]. The first term E_1 is the variance of the expectational error of the considered model and is a constant with respect to the MLP weights. Rather than training a network on E, we can choose to train it using E_2. There may seem to be no advantage to this. However, note that N_v is fixed when we train with E, since we cannot generate correct desired outputs t_p for arbitrary new input vectors. For E_2, however, we can generate as many patterns as we want as follows. First, random vectors $\mathbf{z}_p$ are generated, which may come from the joint pdf $f_X(x)$ or some function of it. Then the vectors $\mathbf{z}_p$ are processed by the BAC, producing scalar outputs s_p. So the limitation on the number of training patterns for the MLP is circumvented. Let N_{va} denote the number of randomly generated data patterns $(\mathbf{z}_p, s_p)$. E_2 can now be rewritten as

$$E_2 = \frac{1}{N_v + N_{va}}\sum_{p=1}^{N_v+N_{va}}(s_p - y_p)^2 \tag{6.6}$$

As we increase N_{va}, the MLP can be trained to mimic the BAC's validation performance as well as its training performance. Thus, small MLPs can mimic the performance of far larger BACs.

6.1.2 Volumetric Method

Here, the basic idea is to form a Voronoi tessellation of the training data's feature space and generate uniformly distributed random data points in this space. The number of data points generated for each cell is therefore made proportional to the volume of that cell.

Let the training data be represented by $\{\mathbf{x}_p, d_p\}_{p=1}^{N_v}$, where $\mathbf{x}_p$ is the input vector and d_p is the correct class id for the p^{th} pattern. Using Self-Organizing Maps (SOM) (see Appendix D), we can group the inputs into K clusters. In order to compute the volume of each cluster, we first calculate the volume of the enclosing hyper-rectangle $\mathcal{R}_k$ for $1 \leq k \leq K$. This is done as follows: for each cluster k, let $\mathbf{x}^{max}(k)$ and $\mathbf{x}^{min}(k)$ be vectors containing maximum and minimum values of the features in k^{th} cluster. The n^{th} element of $\mathbf{x}^{max}(k)$ and $\mathbf{x}^{min}(k)$ are $\{x_n^{max}(k)$ and $\{x_n^{min}(k)$, where n varies from 1 to N. Then,

$$\mathcal{R}_k = \prod_{n=1}^{N} (x_n^{max}(k) - x_n^{min}(k)) \tag{6.7}$$

Next we generate a large number (say N_{vol}) of uniformly distributed data points inside the enclosing hyper-rectangle. We then calculate the number (N_{clus}) of points that actually lie inside the corresponding SOM cluster. The ratio of the number of data points that lie inside the cluster, N_{clus} to the total number of data points generated, N_{vol} is multiplied with the volume of the enclosing hyper-rectangle, $\mathcal{R}_k$ to give the approximate cluster volume $\mathcal{V}_k$.

$$\mathcal{V}_k = \frac{N_{clus}}{N_{vol}} \cdot \mathcal{R}_k \tag{6.8}$$

Once the approximate volumes of each cluster is known, we generate uniformly distributed random vectors and the total number of vectors generated within each cluster will be proportional to the estimated cluster volume. The probability of generating the additional data point in cluster k is calculated by normalizing the cluster volume as

$$p_k = \frac{\mathcal{V}_k}{\sum_{n=1}^{K} \mathcal{V}_n} \tag{6.9}$$

for $1 \leq k \leq K$.

Let N_{va} be the total number of additional patterns to be generated. For each cluster k, where $1 \leq k \leq K$, we generate $p_k \cdot N_{va}$ random input vectors whose elements are generated as

$$z_n(k) = x_n^{min}(k) + (x_n^{max}(k) - x_n^{min}(k)) \cdot r_n \tag{6.10}$$

for $1 \leq n \leq N$. Here r_n is a uniform random variable distributed between 0 and 1. Thus we are generating uniformly distributed inputs within a cluster and the number of points generated within a cluster is proportional to the volume of the cluster. We then pass these $\mathbf{z}_p$ through the BAC and record the corresponding outputs s_p. Thus the new patterns $(\mathbf{z}_p, s_p)$ for $1 \leq p \leq N_{va}$ are generated.

Let the joint pdf of the new data points, $\mathbf{z}$ in cluster k, be $f_{\mathbf{Z}}(\mathbf{z}|k)$. Since the random numbers generated have uniform distribution between $\mathbf{x}^{min}(k)$ and $\mathbf{x}^{max}(k)$ for cluster k, the joint pdf of $\mathbf{z}$ is given by

$$f_{\mathbf{Z}}(\mathbf{z}|k) = \begin{cases} \frac{1}{\mathcal{V}_k}, & \text{for } \mathbf{z} \in C_k \ ; \\ 0, & \text{elsewhere.} \end{cases} \tag{6.11}$$

where C_k represents the cluster k. The overall pdf of the newly generated input vectors $\mathbf{z}$ can be expressed as piecewise uniform distribution over different clusters. Using the probability of generating $\mathbf{z}$ in cluster k, p_k computed in equation 6.9, we can write the overall pdf as

$$f_{\mathbf{Z}}(\mathbf{z}) = \sum_{k=1}^{K} f_{\mathbf{z}}(\mathbf{z}|k) P(\mathbf{z} \in C_k) \tag{6.12}$$

where $P(\mathbf{z} \in C_k)$ is the probability of the vector $\mathbf{z}$ falling in cluster k, which is given by p_k as shown in equation 6.9.

6.1.3 Additive Noise Method

Here, instead of generating uniformly distributed data, we generate new data by adding noise to the original training data points.

First, we calculate the standard deviations of each input feature in the two classes, σ_n^i for $1 \leq n \leq N$ and $i \in \{1, 2\}$. For each training vector $\mathbf{x}_p$ in the training set, we add random vectors $\mathbf{r}_p$ with zero mean ($\mathbf{m}_r = \mathbf{0}$) and a covariance matrix given by:

$$\mathbf{\Sigma}_r = \begin{bmatrix} \sigma_{r1} & 0 & \cdots & 0 \\ 0 & \sigma_{r2} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \sigma_{rN} \end{bmatrix}_{N \times N} \tag{6.13}$$

where,

$$\sigma_{rn} = \begin{cases} \frac{\sigma_{1n}}{10}, & \text{if } d_p = 1; \\ \frac{\sigma_{2n}}{10}, & \text{if } d_p = 2. \end{cases} \tag{6.14}$$

for $1 \leq n \leq N$. Thus we generate a new input vector $\mathbf{z}_p$. Then we pass this $\mathbf{z}_p = \mathbf{x}_p + \mathbf{r}_p$ through the BAC and record the output s_p. The new input pattern $(\mathbf{z}_p, s_p)$ is thus formed.

In order to verify the statistics of the generated patterns, we do the following analysis. Let the mean vector and covariance matrix for the original training inputs be denoted by $\mathbf{m}_x$ and $\mathbf{\Sigma}_x$ respectively. Let the mean vector and covariance matrix of the noise $\mathbf{r}$ be $\mathbf{m}_r$ and $\mathbf{\Sigma}_r$ respectively. Let $\mathbf{z} = \mathbf{x} + \mathbf{r}$. Let $\mathbf{m}_z$ and $\mathbf{\Sigma}_z$ be its mean vector and covariance matrix respectively. The mean vector of the new patterns is

$$\begin{aligned} \mathbf{m}_z &= E\{\mathbf{z}\} \\ &= E\{\mathbf{x} + \mathbf{r}\} \\ &= E\{\mathbf{x}\} + E\{\mathbf{r}\} \\ &= \mathbf{m}_x + \mathbf{m}_r \end{aligned}$$

The covariance matrix of the new patterns is

$$\begin{aligned}\mathbf{\Sigma}_z &= E\{(\mathbf{z}-\mathbf{m}_z)\cdot(\mathbf{z}-\mathbf{m}_z)^T\} \\ &= E\{(\mathbf{x}+\mathbf{r}-(\mathbf{m}_x+\mathbf{m}_r))\cdot(\mathbf{x}+\mathbf{r}-(\mathbf{m}_x+\mathbf{m}_r))^T\} \\ &= E\{((\mathbf{x}-\mathbf{m}_x)+(\mathbf{r}-\mathbf{m}_r))\cdot((\mathbf{x}-\mathbf{m}_x)+(\mathbf{r}-\mathbf{m}_r))^T\}\end{aligned}$$

Since $\mathbf{x}$ and $\mathbf{r}$ are independent of each other, the above equation becomes

$$\begin{aligned}\mathbf{\Sigma}_z &= E\{(\mathbf{x}-\mathbf{m}_x)\cdot(\mathbf{x}-\mathbf{m}_x)^T+(\mathbf{r}-\mathbf{m}_r)\cdot(\mathbf{r}-\mathbf{m}_r)^T\} \\ &= \mathbf{\Sigma}_\mathbf{x}+\mathbf{\Sigma}_\mathbf{r}\end{aligned}$$

Since the noise has zero mean, $\mathbf{m}_z = \mathbf{m}_x$ and the noise standard deviation is $1/10^{\text{th}}$ the standard deviation of the inputs, $\mathbf{\Sigma}_z$ will be a diagonal matrix with diagonal elements equal to $\sqrt{1.1}$ times the diagonal values of $\mathbf{\Sigma}_x$.

Now, let us analyze the probability density function (pdf) of the new dataset. Let the pdfs of the original dataset, noise and the new dataset be represented by $f_\mathbf{X}(\mathbf{x})$, $f_\mathbf{R}(\mathbf{r})$ and $f_\mathbf{Z}(\mathbf{z})$. Let the joint pdf of $\mathbf{X}$ and $\mathbf{R}$ be denoted by $f_{\mathbf{X},\mathbf{R}}(\mathbf{x},\mathbf{r})$. The cumulative distribution function (cdf) of $\mathbf{Z}$ is given by [84]

$$\begin{aligned}F_\mathbf{Z}(\mathbf{z}) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\mathbf{z}-\mathbf{x}} f_{\mathbf{X},\mathbf{R}}(\mathbf{x}',\mathbf{r}')d\mathbf{r}'d\mathbf{x}' \\ &= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\int_{-\infty}^{z_1-x_1}\cdots\int_{-\infty}^{z_N-x_N} f_\mathbf{X}(x_1,x_2,\cdots,x_N)f_\mathbf{R}(r_1,r_2,\cdots,r_N) \\ &\qquad dr_N\cdots dr_1dx_N\cdots dx_1 \qquad (6.15)\end{aligned}$$

As $\mathbf{X}$ and $\mathbf{R}$ are independent random variables, their joint density is equal to the product of individual marginal densities. The pdf of $\mathbf{Z}$ is obtained by differentiating the cdf as follows

$$\begin{aligned} f_{\mathbf{Z}}(\mathbf{z}) &= \frac{d^N}{d\mathbf{z}} F_{\mathbf{Z}}(\mathbf{z}) = \frac{d^N}{dz_1 \cdots dz_N} F_{\mathbf{Z}}(z_1, \cdots, z_N) \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_{\mathbf{X}}(x_1, x_2, \cdots, x_N) \\ &\quad \frac{\partial^N}{\partial z_1 \cdots \partial z_N} \int_{-\infty}^{z_1 - x_1} \cdots \int_{-\infty}^{z_N - x_N} f_{\mathbf{R}}(r_1, r_2, \cdots, r_N) dr_N \cdots dr_1 dx_N \cdots dx_1 \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f_{\mathbf{X}}(x_1, x_2, \cdots, x_N) f_{\mathbf{R}}(z_1 - x_1, \cdots, z_N - x_N) dx_N \cdots dx_1 \quad (6.16) \end{aligned}$$

Note that the derivative turns into partial derivative when absorbed inside the integral in the above equation, which is a result of Leibniz integral rule. Hence the above integral can be expressed in vector form as

$$f_{\mathbf{Z}}(\mathbf{z}) = \int_{-\infty}^{\infty} f_{\mathbf{X}}(\mathbf{x}') f_{\mathbf{R}}(\mathbf{z} - \mathbf{x}') d\mathbf{x}' \quad (6.17)$$

Thus the pdf of new dataset is given by the convolution integral of the marginal pdf's of the original dataset and the additive noise. In other words, we are just smoothing or blurring the pdf of the original distribution for the new data.

6.2 Numerical Results and Analysis

Here we consider two class problems from different datasets and analyse the performances of SVM, AdaBoost and MLPs trained using constructive backpropagation (see appendix C). Here, the MLPs are trained similar to the technique used in chapter 5. First an approximation MLP is trained for one output case and then it is converted into a classifier by including the second output unit. Negated weights connecting the first output is used for the second output.

The variation of performances of MLP when introduced with additional data patterns is analyzed. This technique of using additional data is also called semi-supervised learning [85]. We have used both volumetric method and additive noise method as described in section 6.1.3 to generate the additional patterns and combine them with the original training data. The behavior of the machines are analysed as the number of additional patterns increase. SVM and AdaBoost are considered to be candidates for the BAC. Later, we generalize this technique and show that any learning machine can be used to generate the additional patterns, which the other machines can take advantage of during training.

6.2.1 SVM as BAC

In this subsection, we generate additional patterns using SVM as BAC. Depending on the size of the dataset, we split the data into k parts and perform k-fold cross-validation. It should be noted that here we are using one part for training and $k-1$ parts for validation, hence the percentage error values we get here is different from the results shown in Table 5.1. The new dataset for MLP training consists of the original training data along with the SVM generated data. We have used CBP for training the MLP networks and the number of hidden units (N_h) is fixed from the upper bound theorem that we proved in the previous chapter.

Simulation results show how the additional patterns generation techniques (volumetric method and additive noise method) discussed in the previous section improve the performance of the MLPs.

Volumetric Case

Figs. 6.1 and 6.2 show comparison plots of percentage classification error (average cross-validation error) versus number of additional patterns, N_{va} for Numeral dataset and Segmentation dataset respectively. Here, we have used volumetric technique to generate additional patterns.

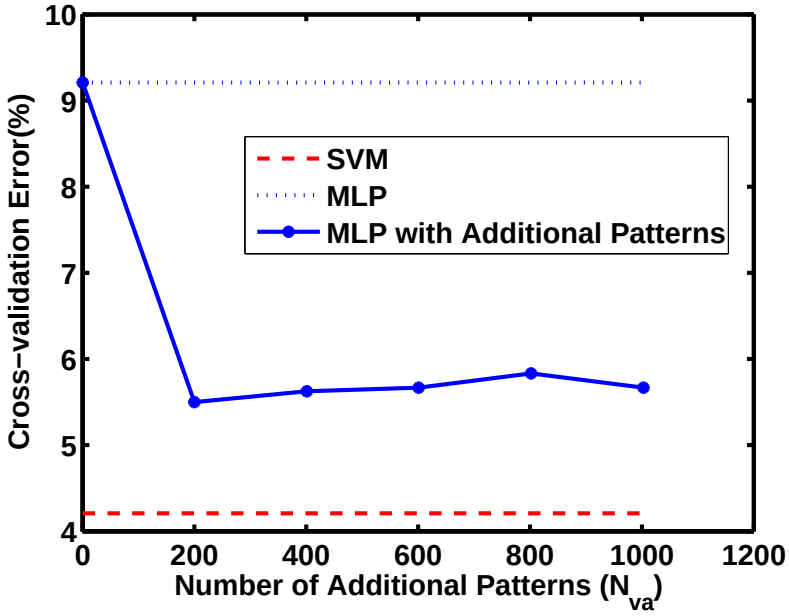


Figure 6.1. Volumetric Method (SVM as BAC), Numeral dataset.

Additive Noise Case

Figs. 6.3 and 6.4 show comparison plots of percentage classification error (average cross-validation error) versus number of additional patterns, N_{va} for the Numeral and Segmentation datasets respectively. Here, we have used additive noise technique to generate the additional patterns.

We can see that the SVM performs better than the MLP on original datasets. As we increase the number of additional patterns included for training, MLP's error decreases significantly. This indicates that the information extracted from the SVM in the form of additional patterns are being utilized constructively. It can be seen that by including the additional patterns, it is possible to get a classifier that is better than the SVM. This is possible because the MLP type training forces all the patterns to be support vectors and the additional patterns that descend from SVM will be used effectively during the MLP training.

Table 6.1 shows the average cross-validation error percentage values for various datasets that are considered here. The additional patterns are generated using the trained

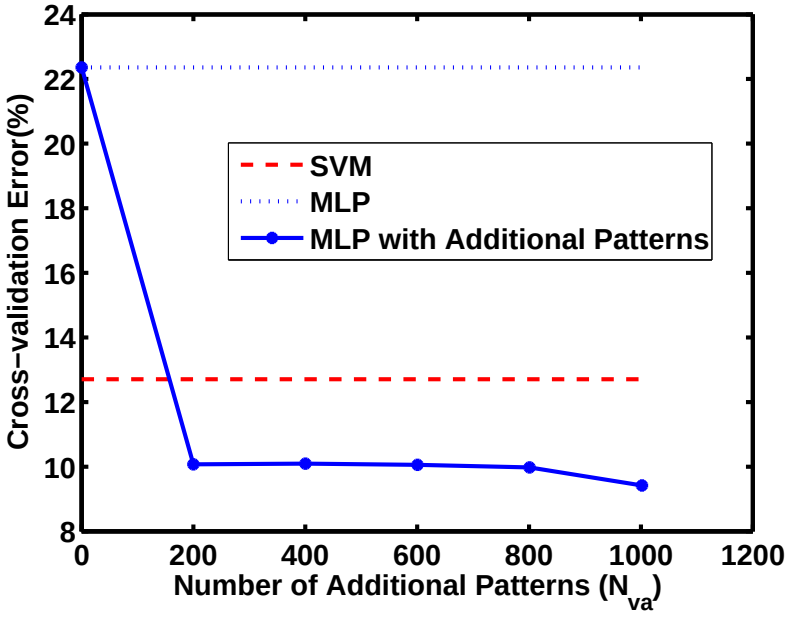


Figure 6.2. Volumetric Method (SVM as BAC), Segmentation dataset.

Table 6.1. Average validation error percentage (SVM as BAC).

Dataset	k fold	N_h	SVM Val Error %	MLP Val Error %	MLP Val Err% Additive Noise	MLP Val Err% Volumetric
Numeral	3	22	4.2	9.2	5.37	5.5
Segmentation	15	5	12.7	22.35	14.049	9.41
Flight Sim	3	20	6.34	7.8	7.8	5.85
Speech	5	2	9.76	18.8	18.8	18.08

SVM and minimum average cross-validation error obtained on both additive noise method and volumetric method are shown. The details of these datasets are given in the previous chapter. It should be noted that the features in fight simulation and the speech datasets are normalized in order to improve the accuracy. It can be seen that both data generation techniques reduce the MLP error most of the time.

Incorrect Models

Let us consider two learning machines $\mathcal{A}$ and $\mathcal{B}$. Let us assume that $\mathcal{A}$ performs better than $\mathcal{B}$ on a particular dataset. If we include the additional patterns generated

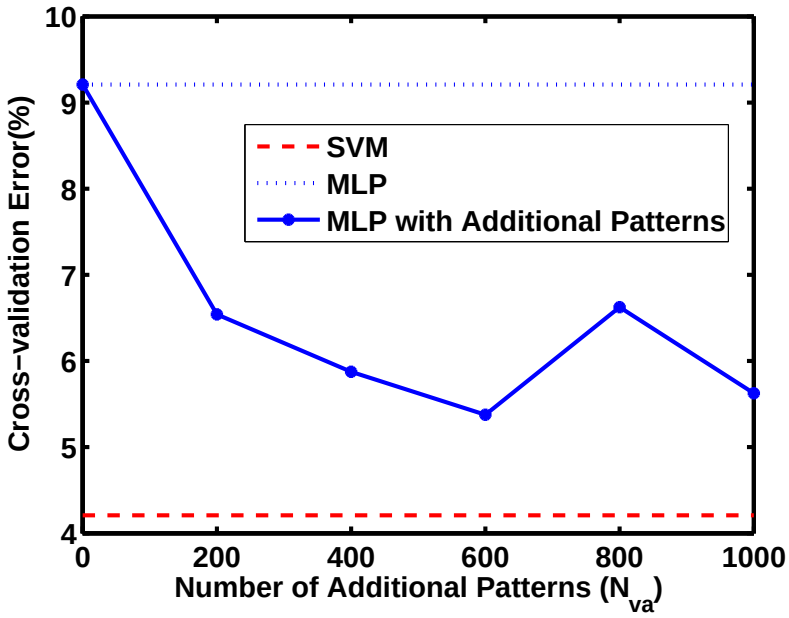


Figure 6.3. Additive Noise Method (SVM as BAC), Numeral dataset.

using $\mathcal{B}$ to train $\mathcal{A}$, we are in effect providing bad data for the better performing learning machine. This is equivalent to assume that the BAC is not used to generate additional patterns. This shows up in the performance resulting in increased validation error. Hence the additional pattern generation technique does not work constructively in such cases. The technique can be applied to train $\mathcal{B}$ using pattern generated by $\mathcal{A}$ and should be used only when there is an advantage of using $\mathcal{B}$ over $\mathcal{A}$ even when their generalization errors are same. This problem is discussed in detail with examples in [85]. They give the theory behind this phenomenon and relate this to the Bias-Variance effect.

An example of the general effects of incorrect models is shown in Fig. 6.5 for Numeral recognition dataset. Here SVM is used to generate additional data, however, AdaBoost performs better than SVM on the original training dataset. When AdaBoost is trained using the newly generated dataset, its performance degrades. This verifies that the additional patterns generated have the expected effect on the learning machines analyzed.

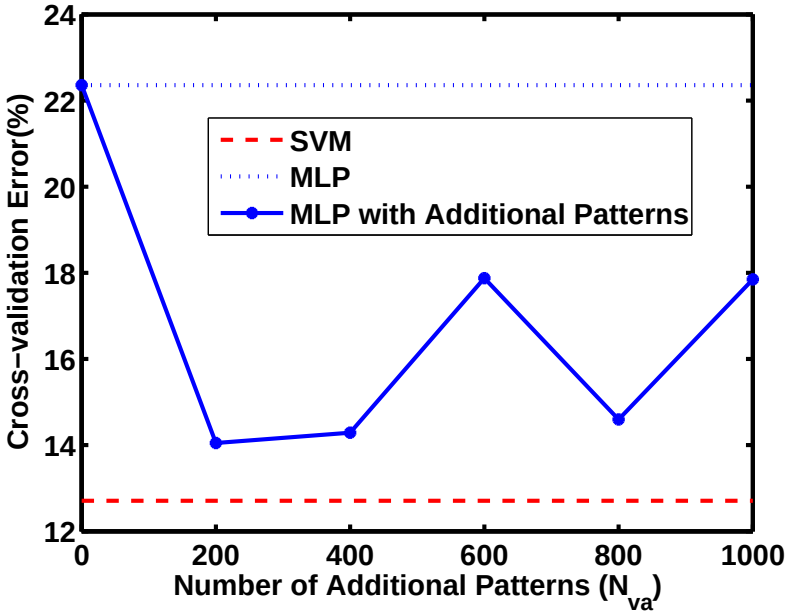


Figure 6.4. Additive Noise Method (SVM as BAC), Segmentation dataset.

6.2.2 AdaBoost as BAC

Since there is significant improvement in MLP's performance by using SVM generated additional data patterns, and as the AdaBoost sometimes performs better than SVM (see last example in previous subsection), we have tried modeling MLPs using additional data patterns generated by a trained AdaBoost classifier. Instead of passing the newly generated input vectors through trained SVM, we pass them through trained AdaBoost and record the output. The new pattern is generated by pairing the input vector with the corresponding AdaBoost output. Again, we have experimented with both volumetric method and additive noise method of generating additional patterns and have shown the results. The new dataset for MLP training consists of the original training data along with the AdaBoost generated data. We have used CBP for training the MLPs.

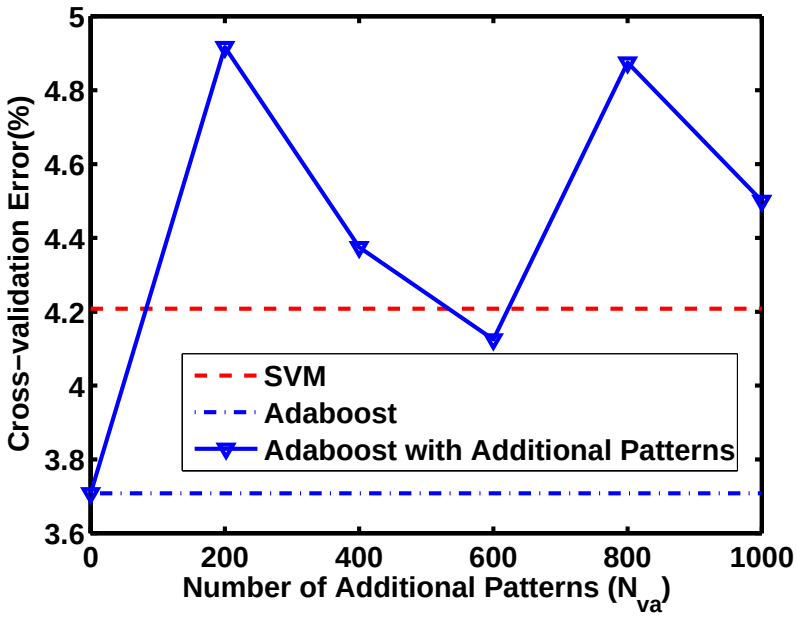


Figure 6.5. SVM as BAC on Numeral dataset.

Volumetric Case

In Figs. 6.6 and 6.7, we show how that the MLP's error can be decreased by generating additional patterns, N_{va} using trained AdaBoost network. In this case, we have generated additional patterns using volumetric method.

Additive Noise Case

In Figs. 6.8 and 6.9, we show how that the MLP's error can be decreased by generating additional patterns, N_{va} using trained AdaBoost network. In this case, we have generated additional patterns using additive noise method.

Similar to that of SVM, we have generated another table for the AdaBoost case. Table 6.2 shows the average cross-validation error percentage values for the datasets that are considered. The additional patterns are generated using the trained AdaBoost and minimum average cross-validation error obtained on both additive noise method and volumetric method are shown. The details of these datasets are given in the previous

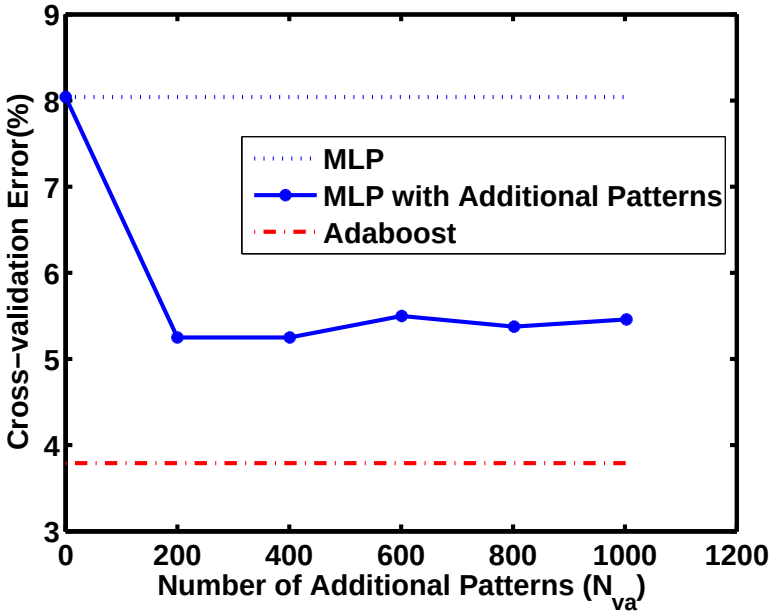


Figure 6.6. Volumetric Method (AdaBoost as BAC), Numeral dataset.

Table 6.2. Average cross-validation error percentage (AdaBoost as BAC).

Dataset	k fold	N_h	AdaBoost Val Error %	MLP Val Error %	MLP Val Err% Additive Noise	MLP Val Err% Volumetric
Numeral	3	22	4.41	7.916	5.12	5.25
Segmentation	15	5	9.62	13.95	8.5	7.9
Flight Sim	3	20	10.08	7.31	7.31	6.5
Speech	5	2	17.04	19.12	16.8	16.4

chapter. It can be seen even here that both data generation techniques reduce the MLP error most of the time.

Incorrect Models

Here, we show a plot where an MLP is trained on AdaBoost generated additional patterns, when MLP better generalizes on the original dataset. This is also equivalent to assume that the BAC is not used to generate additional patterns. Fig 6.10 shows a plot of MLP's error approaching the error of trained AdaBoost network for Flight Simulation dataset.

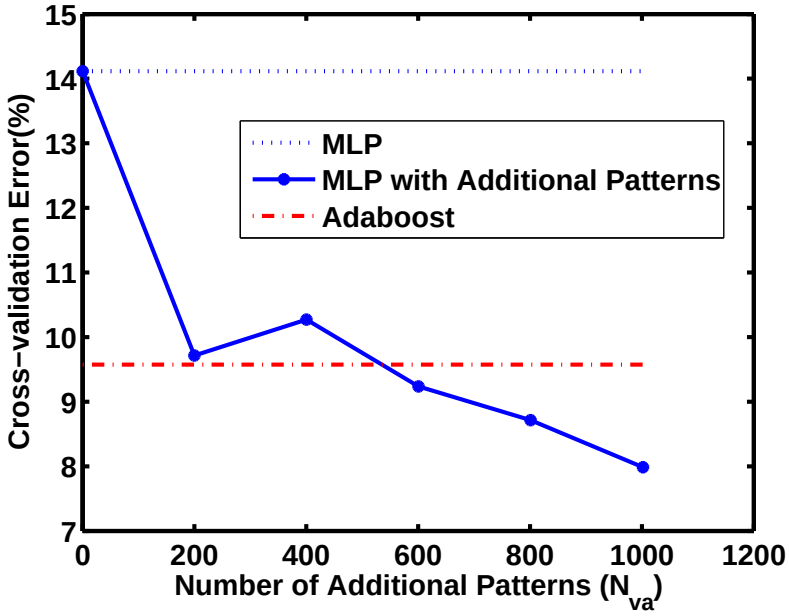


Figure 6.7. Volumetric Method (AdaBoost as BAC), Segmentation dataset.

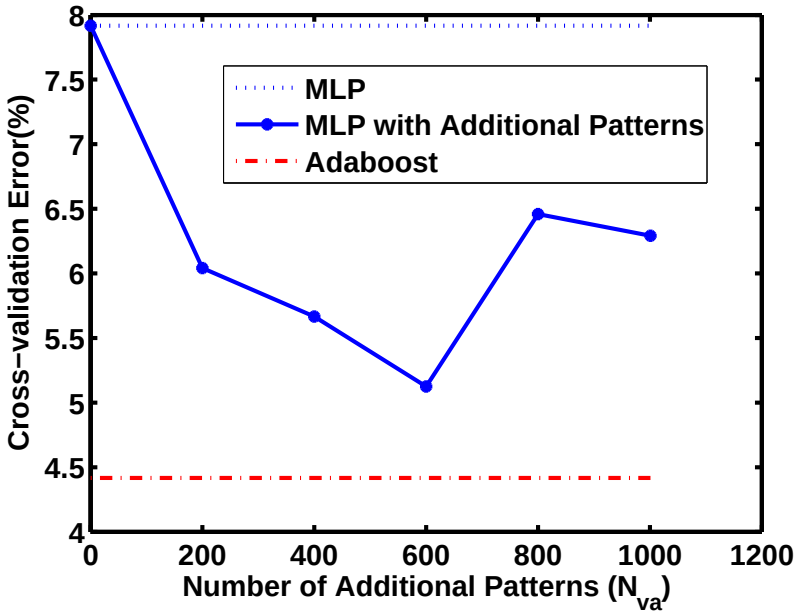


Figure 6.8. Additive Noise Method (AdaBoost as BAC), Numeral dataset.

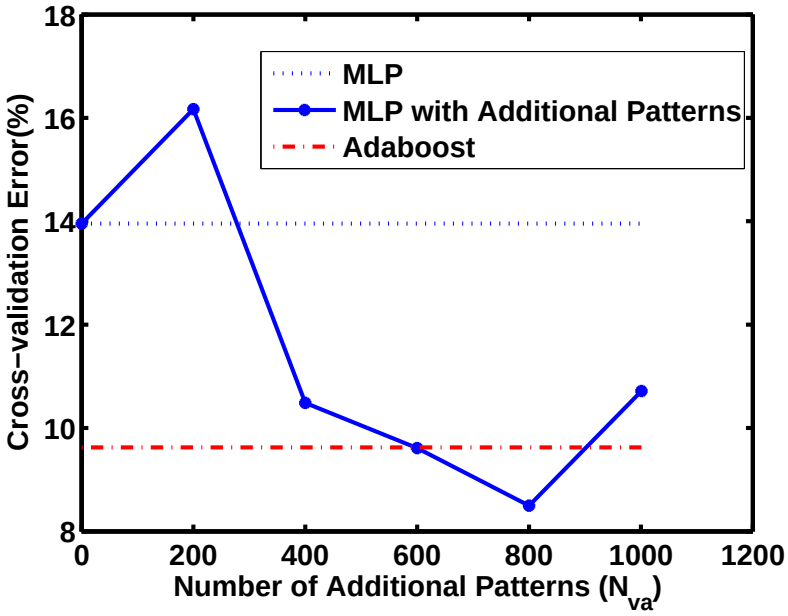


Figure 6.9. Additive Noise Method (AdaBoost as BAC), Segmentation dataset.

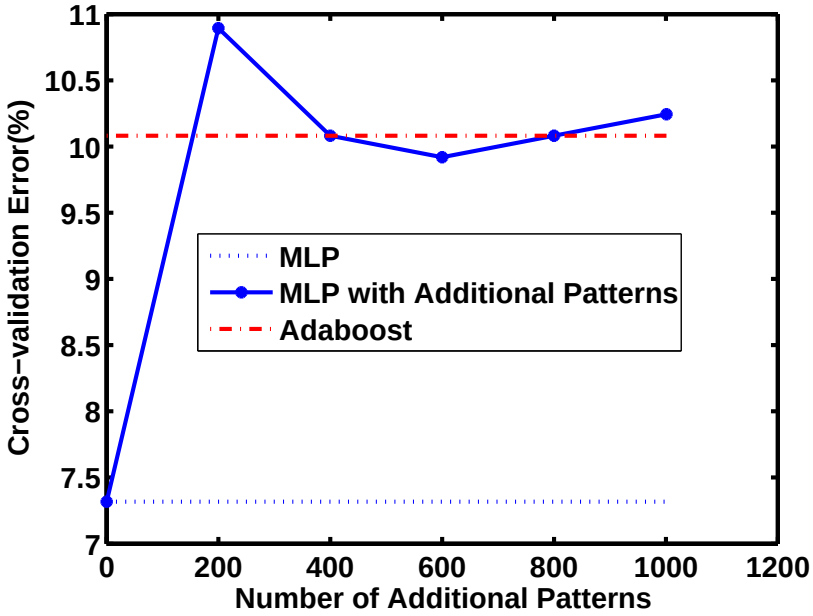


Figure 6.10. AdaBoost as BAC on Flight Simulation dataset.

CHAPTER 7

CONCLUSIONS AND DISCUSSION

We have explored four design methodologies for producing sequences of trained and validated feedforward networks. In the growing approach (DM-1), dependently initialized networks result in monotonically decreasing $E_f(N_h)$ curves. A pruning method (DM-2) is shown that requires one pass through the data. A method is also described for simultaneously validating many different size networks simultaneously, using a single data pass. In the third, combined approach (DM-3), ordered pruning is applied to grown networks. Lastly, our final combined approach (DM-4) successively grows the network by a few units, and then prunes. These methodologies produce networks of different sizes, which generalize well and result in small training and validation errors. The methods also produce sequences of networks that have monotonic MSE versus N_h curves. As seen in the simulations, the DM-4 approach usually produces smaller training and validation errors than the other methodologies.

The DM-4 method was compared with the two other well known growing techniques: constructive backpropagation (CBP) and cascade correlation (CC). On seven different datasets, the results show that our proposed method performs significantly better than the existing methods. We have also extended the work to classification case and have given the performance comparison with the CBP and CC algorithms.

We have developed a simple proof of an upper bound on the storage capacity of feedforward networks with arbitrary hidden unit activation functions. The validity and tightness of the upper bound has been demonstrated through simulation.

We have also developed a technique to obtain small models of large BACs. We have discussed the memorization and redundancy in SVM. Also, we have discussed two

methods of generating additional training data and their effects on the performances of MLP and Adaboost algorithms.

More work remains to be done. In DM-4, the standard deviation of training and validation errors have not been driven to zero. In chapter 6, the validation error curves are too oscillatory. Also, MLP validation error curves do not always approach those of the SVM and AdaBoost.

APPENDIX A

OWO-HWO ALGORITHM

In the original OWO-HWO algorithm [61], the net functions are updated in gradient direction. It is well known that optimizing in the gradient direction, as in steepest descent, is very slow.

Let the new desired net function net_{pjd} for the p^{th} training pattern and j^{th} hidden unit be

$$net_{pjd} = net_{pj} + Z \cdot \Delta net^*_{pj} \tag{A.1}$$

where Δnet^*_{pj} written as

$$\Delta net^*_{pj} = net^*_{pj} - net_{pj} \tag{A.2}$$

is the difference between current net function net_{pj} and the optimal value net^*_{pj}. Now the net function approaches the optimal value net^*_{pj}, instead of moving in the negative gradient direction. The current task for constructing the new HWO algorithm is to find Δnet^*_{pj}.

Using Taylor series expansion on equation 2.2 about net_{pj}, we get

$$O^*_{pj} = f(net^*_{pj}) = O_{pj} + f'_{pj} \cdot Z \cdot \Delta net^*_{pj} \tag{A.3}$$

where f'_{pj} is the derivative of the activation function $f(net_{pj})$ with respect to net_{pj}. Now Δnet^*_{pj} can be derived based on

$$\frac{\partial E}{\partial net_{pj}}\Big|_{net_{pj}=net^*_{pj}} = 0 \tag{A.4}$$

The desired change in net function can be derived as

$$\Delta net^*_{pj} = \frac{\delta_{pj}}{(f'_{pj})^2 \sum_{i=1}^{M} w^2_{oh}(i,j)} \tag{A.5}$$

It can be shown that minimizing E is equivalent to minimizing the weighted hidden error function

$$E_\delta(k) = \frac{1}{N_v}\sum_{p=1}^{N_v}(f'_{pk})^2\left[\Delta net^*_{pk} - \sum_{n=1}^{N+1} e(k,n)x_{pn}\right]^2 \tag{A.6}$$

where $e(k,n)$ is the change in hidden weight. The auto and cross correlation matrices are found as

$$R(n,m) = \frac{1}{N_v}\sum_{p=1}^{N_v}(x_{pn}\cdot x_{pm})\cdot(f'_{pk})^2 \tag{A.7}$$

and

$$C_\delta(k,m) = \frac{1}{N_v}\sum_{p=1}^{N_v}(\Delta net^*_{pk}\cdot x_{pm})\cdot(f'_{pk})^2 \tag{A.8}$$

We have $(N+1)$ equations in $(N+1)$ unknowns for the k^{th} hidden unit. After finding the learning factor Z, the hidden weights are updated as

$$w_{hi}(k,n) \leftarrow w_{hi}(k,n) + Z\cdot e(k,n) \tag{A.9}$$

This modified algorithm converges as the change in the error function E is non-increasing and the algorithm uses backtracking in order to save the best network without diverging from the solution.

APPENDIX B

SCHMIDT PROCEDURE

Here we describe the Schmidt procedure that is used to orthonormalize the MLP basis functions and then we describe the pruning procedure [25]. For the ease of notation, in this section, we use the augmented input vector, i.e., $\mathbf{x}_p \leftarrow [\mathbf{x}_p^T, 1, \mathbf{O}_p^T]^T$, where 1 is for the threshold. The new dimension of $\mathbf{x}_p$ is $N_u = N + N_h + 1$. Also, for simplicity, the subscript p indicating the pattern number will not be used unless it is necessary. The output of the network in equation 2.3, can be rewritten as

$$y_i = \sum_{k=1}^{N_u} w_o(i,k) \cdot x_k \tag{B.1}$$

where $x_k = O_{p,(k-N-1)}$ for $N+2 \leq k \leq N_u$, where N_u is the total number of units equal to $N + N_h + 1$. In equation B.1, the signals x_k are the raw basis functions for producing y_i.

The normal Gram-Schmidt procedure [22] is a recursive process that requires obtaining scalar products between raw basis functions and orthonormal basis functions. The disadvantage in this process is that it requires one pass through the training data to obtain each new basis function. In this section a more useful form of the Schmidt process is reviewed, which will let us express the orthonormal system in terms of autocorrelation elements.

Basic Algorithm

The m^{th} orthonormal basis function x'_m, can be expressed as

$$x'_m = \sum_{k=1}^{m} a_{mk} \cdot x_k \tag{B.2}$$

From equation B.2, for $m = 1$, the first basis function is obtained as

$$x'_1 = \sum_{k=1}^{1} a_{1k} \cdot x_k = a_{11} x_1 \tag{B.3}$$

$$a_{11} = \frac{1}{\|x_1\|} = \frac{1}{r(1,1)^{1/2}} \tag{B.4}$$

where

$$r(i,j) = \langle x_i, x_j \rangle = \frac{1}{N_v} \sum_{p=1}^{N_v} x_{pi} \cdot x_{pj} \tag{B.5}$$

For values of m between 2 and N_u, c_i is first found for $1 \leq i \leq m-1$ as

$$c_i = \sum_{q=1}^{i} a_{iq} \cdot r(q,m) \tag{B.6}$$

Then obtain m coefficients b_k as,

$$b_k = \begin{cases} -\sum_{i=k}^{m-1} c_i \cdot a_{ik}, & 1 \leq k \leq m-1; \\ 1, & k = m. \end{cases} \tag{B.7}$$

Finally for the m^{th} basis function the new a_{mk} coefficients (for $1 \leq k \leq m$) are found as

$$a_{mk} = \frac{b_k}{\left[r(m,m) - \sum_{i=1}^{m-1} c_i^2\right]^{1/2}} \tag{B.8}$$

The output equation B.1 can be written as

$$y_i = \sum_{q=1}^{N_u} w'_o(i,q) \cdot x'_q \tag{B.9}$$

where the weights in the orthonormal system are

$$w'_o(i,q) = \sum_{k=1}^{q} a_{qk} \cdot \langle x_k, t_i \rangle = \sum_{k=1}^{q} a_{qk} \cdot c(i,k) \tag{B.10}$$

where elements of the cross correlation matrix $\mathbf{C}$ are defined as.

$$c(i,k) = \frac{1}{N_v} \sum_{p=1}^{N_v} t_{pi} \cdot x_{pk} \tag{B.11}$$

Using equation B.2, we obtain output weights for the system as

$$w_o(i,k) = \sum_{q=k}^{N_u} w'_o(i,q) \cdot a_{qk} \tag{B.12}$$

Substituting equation B.9 into equation 2.5, we obtain $E(i)$ in orthonormal system as

$$E(i) = \left\langle (t_i - \sum_{k=1}^{N_u} w'_o(i,k) \cdot x'_k), (t_i - \sum_{q=1}^{N_u} w'_o(i,q) \cdot x'_q) \right\rangle \tag{B.13}$$

If we decide to use the first N_{hd} hidden units in our original network, the training error is

$$E(i) = \langle t_i, t_i \rangle - \sum_{k=1}^{N+1+N_{hd}} (w'_o(i,k))^2 \tag{B.14}$$

Modifying equation B.12, the output weights would be

$$w_o(i,k) = \sum_{q=k}^{N+1+N_{hd}} w'_o(i,q) \cdot a_{qk} \tag{B.15}$$

The purpose of pruning is to eliminate useless inputs and hidden units as well as hidden units that are less useful. Useless units are those which (1) have no information relevant for estimating outputs or (2) are linearly dependent on inputs or hidden units that have already been orthonormalized.

We modify the Schmidt procedure so that during pruning, the hidden units are ordered according to their usefulness and useless basis functions x'_m are eliminated.

Let $j(m)$ be an integer valued function that specifies the order in which raw basis functions x_k are processed into orthonormal basis functions x'_k. Then x'_m is to be calculated from $x_{j(m)}, x_{j(m-1)}$ and so on. This function also defines the structure of the new hidden layer where $1 \leq m \leq N_u$ and $1 \leq j(m) \leq N_u$. If $j(m) = k$ then the m^{th} unit of the new structure comes from the k^{th} unit of the original structure.

Given the function $j(m)$, and generalizing the Schmidt procedure, the m^{th} orthonormal basis function is described as

$$x'_m = \sum_{k=1}^{m} a_{mk} \cdot x_{j(k)} \tag{B.16}$$

Initially, x'_1 is found as $a_{11} \cdot x_{j(1)}$ where

$$a_{11} = \frac{1}{\|x_{j(1)}\|} = \frac{1}{r(j(1), j(1))^{1/2}} \tag{B.17}$$

For $2 \leq m \leq N_u$, we first perform

$$c_i = \sum_{q=1}^{i} a_{iq} \cdot r(j(q), j(m)), \quad \text{for} \;\; 1 \leq i \leq m-1 \tag{B.18}$$

Second, we set $b_m = 1$ and get

$$b_k = -\sum_{i=k}^{m-1} c_i \cdot a_{ik}, \quad \text{for} \;\; 1 \leq k \leq m-1 \tag{B.19}$$

Lastly, we get coefficients a_{mk} as

$$a_{mk} = \frac{b_k}{\left[r(j(m), j(m)) - \sum_{i=1}^{m-1} c_i^2\right]^{1/2}}, \quad \text{for} \;\; 1 \leq k \leq m \tag{B.20}$$

Then weights in the orthonormal system are found as

$$w'_o(i, m) = \sum_{k=1}^{m} a_{mk} \cdot c(i, j(k)), \quad \text{for} \;\; 1 \leq i \leq M \tag{B.21}$$

The goal of pruning is to find the function $j(m)$ which defines the structure of the hidden layer. Here it is assumed that the original basis functions are linearly independent i.e., the denominator of equation B.20 is not zero.

Since we want the effects of inputs and the constant "1" to be removed from orthonormal basis functions, the first $N+1$ basis functions are picked as,

$$j(m) = m, \quad \text{for} \;\; 1 \leq m \leq N+1 \tag{B.22}$$

The selection process will be applied to the hidden units of the network. We now define notation that helps us specify the set of candidate basis function to choose in a given iteration. First, define $S(m)$ as the set of indices of chosen basis functions where m is the number of units of the current network (i.e., the one that the algorithm is processing). Then $S(m)$ is given by

$$S(m) = \begin{cases} \{\phi\}, & \text{for } m = 0; \\ \{j(1), j(2), \cdots, j(m)\}, & \text{for } 0 < m \leq N_u. \end{cases} \tag{B.23}$$

Starting with an initial linear network having 0 hidden units, where m is equal to $N+1$, the set of candidate basis functions is clearly $S^c\{m\} = \{1, 2, 3, \cdots, N_u\} - S(m)$, which is $\{N+2, N+3, \cdots, N_u\}$. For $N+2 \leq m \leq N_u$, we obtain $S^c\{m-1\}$. For each trial value of $j(m) \in S^c\{m-1\}$, we perform operations in equations B.18, B.19, B.20 and B.21. Then $P(m)$ is

$$P(m) = \sum_{i=1}^{M} \left[w_o'(i,m)\right]^2 \tag{B.24}$$

The trial value of $j(m)$ that maximizes $P(m)$ is found. Assuming that $P(m)$ is maximum when validation the i^{th} element, then $j(m) = i$. $S(m)$ is updated as

$$S(m) = S(m-1) \cup \{j(m)\} \tag{B.25}$$

Then for the general case the candidate basis functions are, $S^c(m-1) = \{1, 2, 3, \cdots, N_u\} - \{j(1), j(2), \cdots, j(m-1)\}$ with $N_u - m + 1$ candidate basis function. By using equation B.24, after validation all the candidate basis function, $j(m)$ takes its value and

$S(m)$ is updated according to equation B.25. Defining N_{hd} as the desired number of units in the hidden layer, the process is repeated until $m = N + 1 + N_{hd}$. Then the orthonormal weights are mapped to normal weights according to equation B.30. Considering the final value of function $j(m)$ row reordering of the original input weights matrix is performed for generating the right O_{pj} values when applying equation B.1. After reordering the rows, because only the N_{hd} units are kept then the remaining units (O_{pj} with $N_{hd} < j \leq N_h$) are pruned by deleting the last $N_h - N_{hd}$ rows.

Linear dependency condition

Unfortunately, ordered pruning by itself is not able to handle linearly dependent basis functions. A minor modification is necessary. Assume that raw basis function $x_{j(m)}$ is linearly dependent on previously chosen basis functions, where $j(m)$ denotes an input ($1 \leq m \leq N$) and $j(m)$ has taken on a trial value. Then

$$x_{j(m)} = \sum_{k=1}^{m-1} d_k \cdot x'_k \tag{B.26}$$

Now the denominator of a_{mk} in B.20 can be rewritten as

$$g = \langle z_m, z_m \rangle^{1/2} \tag{B.27}$$

where

$$z_m = x_{j(m)} - \sum_{i=1}^{m-1} \langle x'_i, x_{j(m)} \rangle \cdot x'_i \tag{B.28}$$

Substituting B.26 into B.28, however, we get

$$\langle x'_i, x_{j(m)} \rangle = d_i \tag{B.29}$$

and z_m and g are both zero.

If $j(m)$ denotes an input and g is infinitesimally small, then we equate $j(k)$ to k for $1 \leq k < m$ and $j(k) = k + 1$ for $m \leq k \leq N$. In effect we decrease N by one and let the $j(k)$ function skip over the linearly dependent input. If $j(m)$ denotes a hidden unit, the same procedure is used to determine whether or not $x_{j(m)}$ is useful. If $x_{j(m)}$ is found to be linearly dependent, the current, bad value of $j(m)$ is discarded before a_{mk} is calculated.

Once we get these orthonormal weights, the hidden units and their weights are reordered in the descending order of their energies. Then the output weights of the original system are obtained using the equation

$$w_o^{DM2}(i,k) = \sum_{q=k}^{N_u} w_o'(i,q) \cdot a_{qk} \tag{B.30}$$

Here $w_o^{DM2} \in \mathcal{W}_{DM2}^{N_h}$ is the output weights obtained using Schmidt procedure.

APPENDIX C

REVIEW OF EXISTING GROWING METHODS

Here, we briefly discuss two existing well known DI approaches for growing a sequence of networks.

Constructive Backpropagation (CBP)

In the constructive backpropagation algorithm of Lehtokangas [64], the backpropagation algorithm is used to train the network. Let the initial network trained be linear ($N_h = 0$) and be denoted by $\mathcal{W}^0_{cbp}$. Then add a batch of hidden units ($N_h = N_h + N_\epsilon$) and train them to get another network $\mathcal{W}^{N_h}_{cbp}$. Once the new hidden units are trained, their input and output weights are frozen and further training is continued only on the newly added hidden units. There is only one error function to be minimized, which is the squared error between the previous error of the output unit and the weighted sum of the newly added hidden units.

$$\begin{aligned} E_{cbp} &= \sum_{p=1}^{N_v}\sum_{k=1}^{M}\Big\{t_{pk} - y^{cbp}_{pk}\Big\}^2 \\ &= \sum_{p=0}^{N_v}\sum_{k=1}^{M}\Big\{t_{pk} - \big(\sum_{i=1}^{N+1} w^{cbp}_{oi}(k,i)\cdot x_{pi} + \sum_{j=1}^{N_h-N_\epsilon} w^{cbp}_{oh}(k,j)\cdot O_{pj}\big) \\ &\quad - \sum_{n=N_h-N_\epsilon+1}^{N_h} w^{cbp}_{oh}(k,n)\cdot O_{pn}\Big\}^2 \\ &= \sum_{p=1}^{N_v}\sum_{k=1}^{M}\left\{e^{cbp}_{pk} - \sum_{n=N_h-N_\epsilon+1}^{N_h} w^{cbp}_{oh}(k,n)\cdot O_{pn}\right\}^2 \end{aligned} \tag{C.1}$$

Here y^{cbp}_{pk} is the k^{th} output of the p^{th} pattern of the network $\mathcal{W}^{N_h}_{cbp}$, e_{pk} is the error between the desired output and the output of the previous network $\mathcal{W}^{N_h-N_\epsilon}_{cbp}$. The weights connecting the inputs and hidden units to the output units respectively are represented by $w^{cbp}_{oi} \in \mathcal{W}^{N_h}_{cbp}$ and $w^{cbp}_{oh} \in \mathcal{W}^{N_h}_{cbp}$.

Cascade Correlation (CC)

The cascade correlation learning procedure [60] is similar to constructive back-propagation, in that the initial network is linear (i.e., $N_h = 0$). The network obtained is represented by $\mathcal{W}_{cc}^0$. We have slightly modified the original cascade correlation algorithm so that we can add multiple hidden units at each growing step making fair comparison to other methods. Also, the cascade correlation architecture can be viewed as standard single hidden layer MLP with additional lateral weights in the hidden layer connecting every hidden unit to all its previous hidden units. This set of lateral weights can be represented by $w_{hh}^{cc}(i,j) \in \mathcal{W}_{cc}^{N_h}$ for $2 \leq i \leq N_h$ and $1 \leq j < i$. Therefore, $w_{hh}^{cc}(i,j)$ is the weight going from j^{th} hidden unit to i^{th} hidden unit. The cost function to be maximized is

$$E_{cc} = \sum_{j=N_h-N_\epsilon+1}^{N_h} \sum_{k=1}^{M} \left| \sum_{p=1}^{N_v} (O_{pj} - \hat{O}_j) \cdot (E_{pk} - \hat{E}_k) \right| \tag{C.2}$$

where $\hat{O}_j$ is the mean of the activations of the j^{th} hidden unit and $\hat{E}_k$ is the mean of the errors of the k^{th} output unit over all the patterns. Here also, once the hidden unit's input and output weights are trained, they will be frozen and the next step in growing will adapt only the weights connecting the new hidden units.

APPENDIX D

SELF ORGANIZING MAPS

In Self Organizing Map (SOM) network [86], the high dimensional input vector is projected nonlinearly onto a regular two dimensional grid. This mapping tends to preserve the topological relation between the input elements. Let us assume there are K clusters. Each cluster k is associated with a parametric reference vector $\mathbf{m}_k = [m_{k1}, m_{k2}, \cdots, m_{kN}] \in \mathbb{R}^N$ for $1 \leq k \leq K$. In an abstract scheme, it may be imagined that the input $\mathbf{x}_p$, by means of some parallel computing mechanisms, is compared with all the $\mathbf{m}_k$, and the location of best match in some metric is defined as the location of the "response". By computer programming, of course, the location of the best match is a trivial task. The exact magnitude of the response need not be determined; the input is simply mapped into this location, like in a set of decoders. For an input $\mathbf{x}_p$, the Euclidean distances $\|\mathbf{x}_p - \mathbf{m}_k\|$ is computed and the smallest distance is defined as the best-matching node, denoted by subscript c.

$$c = \arg\min_k \{\|\mathbf{x}_p - \mathbf{m}_k\|\} \tag{D.1}$$

which is same as

$$\|\mathbf{x}_p - \mathbf{m}_c\| = \min_k \|\mathbf{x}_p - \mathbf{m}_k\| \tag{D.2}$$

The N-dimensional input vector is mapped onto nearest center (reference) vector $\mathbf{m}_c$. Thus the entire input space is divided into Voronoi regions. During learning, those nodes that are topographically close in the array up to a certain geometric distance will activate each other to learn something from the same input $\mathbf{x}_p$.

The SOM learning algorithm is as follows: Given K, number of clusters, N_v vectors $\mathbf{x}_p$ of dimension N, decreasing functions $z(t)$ and $N(t)$ and the number of iterations, N_{it},

1. Find the mean (μ_i) and standard deviation (σ_i) for each of the N inputs, $1 \leq i \leq N$.
2. Initialize the cluster means (reference vectors) as $\mathbf{m}_i$ as Gaussian random vectors with i^{th} element having mean μ_i and standard deviation σ_i. Initialize iteration number to zero, $i_t = 0$
3. $i_t = i_t + 1$. Quit if $i_t > N_{it}$

4. For $1 \leq p \leq N_v$,

   $t = p + (i_t - 1)N_v$

   Find c such that, $\text{dist}(\mathbf{x}_p, \mathbf{m}_c)$ is minimum

   Update $\mathbf{m}_k$ as

   $\mathbf{m}_k = \mathbf{m}_k + z(t)(\mathbf{x}_p - \mathbf{m}_k)$ for $|c - k| \leq N(t)$

   end

5. Goto step 3

Following exponential functions are used for the gains and neighborhoods.

$$z(t) = a_1 e^{-t/T_1}$$
$$N(t) = a_2 + a_3 e^{-t/T_2} \tag{D.3}$$

where,

$$T_1 = \frac{N_v N_{it}}{3}, T_2 = \frac{N_v N_{it}}{10}$$
$$a_1 = \frac{K}{N_v}, a_2 = 0, a_3 = \frac{K}{10}$$

Once the trained center vectors from SOM is obtained, they can be used to initialize the RBF means. This would help the model to adapt to the system quickly.

REFERENCES

[1] K. Hornik, M. Stinchombe, and H. White, "Multilayer feedforward networks are universal function approximators," *IEEE Transactions Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] G. G. Lorentz, *Approximation of Functions.* Holt, Reinhart and Winston, 1966.

[3] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," *Proceedings of the International Conference on Neural Networks*, vol. 3, pp. 11–14, 1987.

[4] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Dolk. Akad Nauk (USSR), in Russian*, vol. 114, pp. 952–956, 1957.

[5] K. Liu, S. Subbarayan, M. T. Manry, C. Kwan, F. L. Lewis, and J. Naccarino, "Comparison of very short-term load forecasting techniques," *IEEE Transactions on Power Systems*, vol. 11, no. 2, pp. 877–882, May 1996.

[6] M. T. Manry, R. Shoults, and J. Naccarino, "An automated system for developing neural network short term load forecasters," in *Proceedings of the 58th American Power Conference*, Apr 1996, pp. 237–241.

[7] Y. Saifullah and M. T. Manry, "Classification based segmentation of zip codes," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, pp. 1437–1443, Sep/Oct 1993.

[8] Y. LeCun, B. Boser, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[9] M. T. Manry, C.-H. Hsieh, and H. Chandrasekaran, "Near-optimal flight load synthesis using neural nets," in *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*, 1999, pp. 535–544.

[10] E. F. M. Filho and A. C. P. L. de Carvalho, "Target recognition using evolutionary neural networks," in *Proceedings of Vth Brazilian Symposium on Neural Networks, 1998*, Dec 1998, pp. 226–231.

[11] G. Edwards and J. P. Tate, "Target recognition and classification using neural networks," in *Proceedings of MILCOM 2002*, vol. 2, Oct 2002, pp. 1439–1442.

[12] E. W. Saad, D. V. Prokhorov, and D. C. W. II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Transaction on Neural Networks*, vol. 9, no. 6, pp. 1456–1470, Nov 1998.

[13] W. H. Delashmit, "Multilayer perceptron structured initialization and separating mean processing," Dissertation, University of Texas at Arlington, December 2003.

[14] F. L. Chung and T. Lee, "Network-growth approach to design of feedforward neural networks," *IEE Proceedings Control Theory*, vol. 142, no. 5, pp. 486–492, September 1995.

[15] R. Sentiono, "Feedforward neural network construction using cross validation," *Neural Computation*, vol. 13, pp. 2865–2877, 2001.

[16] S.-U. Guan and S. Li, "Parallel growing and training of neural networks using output parallelism," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 542–550, May 2002.

[17] P. V. S. Ponnapalli, K. C. Ho, and M. Thomson, "A formal selection and pruning algorithm for feedforward artificial neural network optimization," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 964 – 968, July 1999.

[18] H. Amin, K. M. Curtis, and B. R. H. Gill, "Dynamically pruning output weights in an expanding multilayer perceptron neural network," in *13th International Conference on DSP*, vol. 2, 1997, pp. 991–994.

[19] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm that varies the number of hidden units," in *Neural Networks*, vol. 4, 1991, pp. 61–66.

[20] I. I. Sakhnini, M. T. Manry, and H. Chandrasekaran, "Iterative improvement of trigonometric networks," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, July 1999, pp. 1275–1280.

[21] R. Reed, "Pruning algorithms - a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, September 1993.

[22] G. Strang, *Linear Algebra and its application.* New York: Harcourt Brace, 1988.

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

[24] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1177–1183, September 1997.

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

[26] X. Liang and L. Ma, "A study of removing hidden neurons in cascade correlation neural networks," in *IEEE International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 1015–1020.

[27] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 519–531, May 1997.

[28] G. B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning rbf neural network for function approximation," *IEEE Trans. on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.

[29] T.-H. Kim, J. Li, and M. T. Manry, "Evaluation and improvement of two training algorithms," in *The 36th Asilomar Conference on Signals, Systems, and Computers*, 2002, pp. 1019–1023.

[30] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, Sep 1999.

[31] ——, *Statistical Learning Theory.* New York, Wiley, 1998.

[32] E. B. Baum, "On the capabilities of multilayer perceptrons," *Journal of Complexity*, vol. 4, pp. 193–215, 1988.

[33] M. A. Sartori and P. J. Antsaklis, "A simple method to derive bounds on the size and to train multilayer neural networks," *IEEE Transactions in Neural Networks*, vol. 2, no. 4, pp. 467–471, Jul 1991.

[34] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation function," *IEEE Transactions in Neural Networks*, vol. 9, no. 1, pp. 224–229, Jan 1998.

[35] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Transactions in Neural Networks*, vol. 14, no. 2, pp. 274–281, Mar 2003.

[36] C. Mazza, "On the storage capacity of nonlinear neural networks," *Neural Networks*, vol. 10, no. 4, pp. 593–597, 1997.

[37] H. Suyari and I. Matsuba, "New approach to the storage capacity of neural networks using the minimum distance between input patterns," in *International Joint Conference on Neural Networks*, vol. 1, Jul 1999, pp. 427–431.

[38] M. Cosnard, P. Koiran, and H. Paugam-Moisy, "Bounds on the number of units for computing arbitrary dichotomies by multilayer perceptrons," *Journal of Complexity*, vol. 10, pp. 57–63, 1994.

[39] A. Elisseeff and H. Paugam-Moisy, "Size of multilayer networks for exact learning: Analytic approach," in *Neural Information Processing Systems*, 1996, pp. 162–168.

[40] C. Ji and D. Psaltis, "Capacity of two-layer feedforward neural networks with binary weights," *IEEE Transaction on Information Theory*, vol. 44, no. 1, pp. 256–268, 1998.

[41] Y. S. A. Moussaoui and H. A. Abbassi, "Hybrid hot strip rolling force prediction using a bayesian trained artificial neural network and analytical models," *American Journal of Applied Sciences*, vol. 3, no. 6, pp. 1885–1889, 2006.

[42] S. Ma and C. Ji, "Performance and efficiency: Recent advances in supervised learning," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1519–1535, Sep 1999.

[43] P. J. Davis, *Interpolation and Approximation*. Blaisdell Publishing Company, 1963.

[44] S. Z. Li and A. K. Jain, *Handbook of face recognition*. New York : Springer, 2005.

[45] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, *Handbook of fingerprint recognition*. New York : Springer, 2003.

[46] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986, vol. 1.

[47] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, pp. 197–227, 1990.

[48] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, Bari, Italy, 1996, pp. 148–156.

[49] ——, "Game theory, on-line prediction and boosting," in *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, Desenzano del Garda, Italy, 1996, pp. 325–332.

[50] F.Valafar, "Pattern recognition techniques in microarray data analysis: A survey," *Techniques in Bioinformatics and Medical Informatics*, vol. 980, pp. 41–64, Dec 2002.

[51] M. P. S. Brown, W. N. Grundy, N. C. C. W. S. D. Lin, T. S. Furey, M. A. Jr., and D. Haussler, "Knowledge-based analysis of microarray gene expression data by using support vector machines," in *Proceedings of the National Academy of Science*, ser. 1, vol. 97, Jan 2000, pp. 262–267.

[52] B. J. de Kruif and T. J. A. de Vries, "Pruning error minimization in least squares support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 696–702, May 2003.

[53] Y.-J. Lee and S.-Y. Huang, "Reduced support vector machines: A statistical theory," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 1–13, Jan 2007.

[54] Y.-H. Lee and O. L. Mangasarian, "Rsvm: Reduced support vector machines," in *Proceedings of SIAM International Conference on Data Mining*, SIAM, Philadelphia, 2001.

[55] M. Tipping, "The relevance vector machine," in *Advances in Neural Information Processing Systems, San Mateo, CA*. Morgan Kaufmann, 2000.

[56] M. H. Fun and M. T. Hagan, "Levenberg-marquardt training for modular networks," in *IEEE International Conference on Neural Networks*, vol. 1, 1996, pp. 468–473.

[57] P. Arena, R. Caponetto, L. Fortuna, and M. G. Xibilia, "Genetic algorithms to select optimal neural network topology," in *Proceedings of the 35th Midwest Symposium on Circuits and Systems*, vol. 2, 1992, pp. 1381–1383.

[58] J. Davila, "Genetic optimization of nn topologies for the task of natural language processing," in *Proceedings of International Joint Conference on Neural Networks*, vol. 2, 1999, pp. 821–826.

[59] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," in *IEEE Transactions on Neural Networks*, ser. 1, vol. 5, 1994, pp. 39–53.

[60] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, 1990, p. 524.

[61] H. H. Chen, M. T. Manry, and H. Chandrasekaran, "A neural network training algorithm utilizing multiple sets of linear equations," *Neurocomputing*, vol. 25, no. 1-3, pp. 55–72, April 1999.

[62] C. M. Bishop, *Neural Networks and Machine Learning*, C. M. Bishop, Ed. Springer, 1998, vol. 168.

[63] V. N. Vapnik, "Principles of risk minimization for learning theory," *Advances in Neural Information Processing Systems*, vol. 4, pp. 831–838, 1992.

[64] M. Lehtokangas, "Modelling with constructive backpropagation," *Neural Networks*, vol. 12, pp. 707–716, 1999.

[65] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Pearson Education, 2004.

[66] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, 1995, pp. 23–37.

[67] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," in *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 322–330.

[68] U. of Texas at Arlington, "Training datasets," http://www-ee.uta.edu/eeweb/IP/training_data_files.htm.

[69] C. Yu, M. T. Manry, J. Li, and P. L. Narasimha, "An efficient hidden layer training method for the multilayer perceptron," *Neurocomputing*, vol. 70, pp. 525–535, 2006.

[70] Y. K. Sarabandi and F. T. Ulaby, "An empirical model and an inversion technique for radar scattering from bare soil surfaces," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 30, no. 2, pp. 370–381, March 1992.

[71] A. K. Fung, Z. Li, and K. S. Chen, "Back scattering from a randomly rough dielectric surface," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 30, no. 2, pp. 356–369, March 1992.

[72] M. S. Dawson, A. K. Fung, and M. T. Manry, "Surface parameter retrival using fast learning neural networks," in *Remote Sensing Reviews*, vol. 7, no. 1, 1993, pp. 1–18.

[73] M. S. Dawson, J. Olvera, A. K. Fung, and M. T. Manry, "Inversion of surface parameters using fast learning neural networks," in *Proceedings of IGARSS*, vol. 2, May 1992, pp. 910–912.

[74] R. Gore, J. Li, M. T. Manry, L. M. Liu, C. Yu, and J. Wei, "Iterative design of neural network classifiers through regression," in *International Joint Conference on Neural Networks*, vol. 14, no. 1 & 2, 2005.

[75] R. R. Bailey, E. J. Pettit, R. T. Borochoff, M. T. Manry, and X. Jiang, "Automatic recognition of usgs land use/cover categories using statistical and neural network classifiers," in *Proceedings of SPIE OE/Aerospace and Remote Sensing*, 1993.

[76] W. Gong, H. C. Yau, and M. T. Manry, *Progress in Neural Networks*. Ablex Publishing Corporation, 1994, vol. 2, ch. Non-Gaussian Feature Analysis Using a Neural Network, pp. 253–269.

[77] D. N. A. Asuncion, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[78] Y.-H. Pao and Y. Takefuji, "Functional-link net computing: Theory, system architecture, and functionalities," *IEEE Computer*, vol. 25, no. 5, pp. 76–79, 1992.

[79] H. Jeffreys and B. S. Jeffreys, *Methods of Mathematical Physics*, 3rd ed. Cambridge, England: Cambridge University Press, 1988, ch. Lagrange's Interpolation Formula, p. §9.011 pp. 260.

[80] T.-H. Kim, M. T. Manry, and F. Maldonado, "New learning factor and testing methods for conjugate gradient training algorithm," in *International Joint Conference on Neural Networks*, vol. 3, Jul 2003, pp. 2011–2016.

[81] E. M. Johansson, F. U. Dowla, and D. M. Goodman, "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method," *Journal of Neural Systems*, vol. 2, no. 4, pp. 291–301, 1992.

[82] T. Joachims, *SVMlight: Support Vector Machine*, 2004, software available at http://svmlight.joachims.org/.

[83] S. Geman, E. Bienemstock, and R. Doursat, "Neural networks and bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.

[84] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, 2nd ed. Pearson Education, 2005.

[85] N. Sebe, I. Cohen, A. Garg, and T. Huang, *Machine Learning in Computer Vision*. Springer, 2005, vol. 29.

[86] T. Kohonen, *Self-Organizing Maps*. Springer, 1995.

BIOGRAPHICAL STATEMENT

Pramod Lakshmi Narasimha was born in Bangalore, India in 1979. He received his B.E. in Telecommunications Engineering from Bangalore University, India. He joined the Neural Networks and Image Processing Lab in the Department of Electrical Engineering (EE) at the University of Texas at Arlington (UTA) as a research assistant in 2002. He received his M.S. degree in 2003 and Ph.D degree in 2007, both from the Department of Electrical Engineering at UTA. His research interests focus on machine learning, neural networks, estimation theory, pattern recognition and computer vision. He is a member of Tau Beta Pi and a student member of the IEEE.